Tools for Reproducible Workflows in R

April, 2023

Table of Contents

# About this Course

Reproducibility of data analyses can be enhanced through the use of tools designed to manage the complexity involved in any data analysis designed to address an important scientific question. We focus on a few software tools that aid in project organization, collaboration, auditability of analyses, and maintaining the integrity of data and code. In this course, we view a data analysis as a complex system with many integrated parts that together produce analytic results. The tools we focus on here allow data analysts to diagnose unexpected results, quickly identify problems with data and code, and provide a basis for managing the dynamic nature of data analysis.

This initiative is funded by the following grant: R25GM141505 from the National Institute of General Medical Sciences (NIGMS). Except where otherwise indicated, the contents of this course are available for use under the Creative Commons Attribution 4.0 license. You are free to adapt and share the work, but you must give appropriate credit, provide a link to the license, and indicate if changes were made. Sample attribution: Tools for Reproducible Workflows in R by [Fred Hutchinson Data Science Lab](https://hutchdatascience.org/) and [University of Texas, Austin](https://stat.utexas.edu/) (CC-BY 4.0). You can download the illustrations by clicking [here](https://docs.google.com/presentation/d/1MNHf8JpolaEP_vQ_kB-1xRBF9wo3haCArRu117hBoHA/edit?usp=sharing).

## 0.1 Available course formats

This course is available in multiple formats which allows you to take it in the way that best suites your needs. You can take it for certificate which can be for free or fee.

* The material for this course can be viewed without login requirement on this [Bookdown website](https://hutchdatascience.org/Tools_for_Reproducible_Workflows_in_R/). This format might be most appropriate for you if you rely on screen-reader technology.
* This course can be taken for [free certification through Leanpub](LINK%20HERE).
* This course can be taken on [Coursera for certification here](LINK%20HERE) (but it is not available for free on Coursera).
* Our courses are open source, you can find the [source material for this course on GitHub](https://github.com/fhdsl/Tools_for_Reproducible_Workflows_in_R).

# 1 Introduction

In this course we will explore a variety of tools that can assist with data analysis from a broad range of fields. The tools we will cover may take some time to get used to, but the payoff will be immeasurable. Not only are these skills valuable for career advancement, they will also make your work-life easier. The tools will enhance your ability to reproduce your work across similar projects, stay organized, collaborate with others effectively, and more.

## 1.1 Motivation

Many researchers are self-taught when it comes to computer science. However, data analysis has become a requirement for most researchers. The ability to smoothly work in a reproducible manner not only makes for easier more maintainable workflows, it also improves scientific rigor and transparency.

This course will help learners to use tools that will make their data analytic workflows more organized, more understandable to collaborators (and your future self!), and ultimately more efficient.

## 1.2 Target Audience

This course is intended for people conducting data analyses at the level of a graduate student or higher. The course is designed so that the majority of the material is presented in a high-level manner that should be applicable to researchers working in a broad range of areas. The course is centered around the R programming language, a widely used statistical analysis software package.

## 1.3 Curriculum

The course covers…

## 1.4 Learning Objectives

* Implement basic project organization tools:
* R Studio tips and tricks for efficiency
* R Markdown to create reports
  + Setup and configure RStudio/RStudio projects for data analysis (here package and file structure/paths)
  + Install and configure ProjectTemplate package for formalizing and automating workflows
* Apply the pointblank package for validation of tabular data
* Write functions and package them
* Apply the testthat package for building software unit tests
* Setup and use Git repositories for version control of code
* Interface with GitHub to share Git repositories for collaboration; execute GitHub-based workflows
  + Pull Requests
  + Code review
  + Issues
  + Discussions

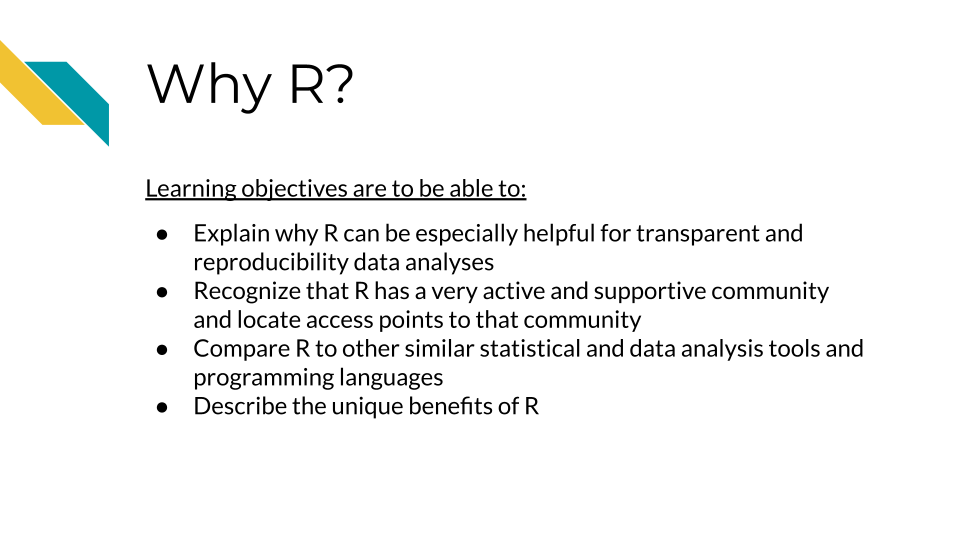
References will include Gillespie and Lovelace ([2021](#ref-gillespie_efficient_2021)), Riederer ([2020](#ref-riederer_column_2020)), Timbers, Campbell, and Lee ([2022](#ref-timbers_data_nodate)).

Code review references will include “About Scientific Code Review” ([n.d.](#ref-hutchdatascience_code_review)), Radigan ([n.d.](#ref-radigan_what_nodate)), Parker ([2017](#ref-parker_opinionated_2017)), Bodner ([2018](#ref-bodner_10_2018)).

# 2 R for Reproducibility

## 2.1 Learning Objectives

Before we begin to jump into additional tools that R can help us with to be work more efficiently and in a more reproducible manner, it is helpful to first discuss why we should consider R in the first place. After completing this section you will be able to:

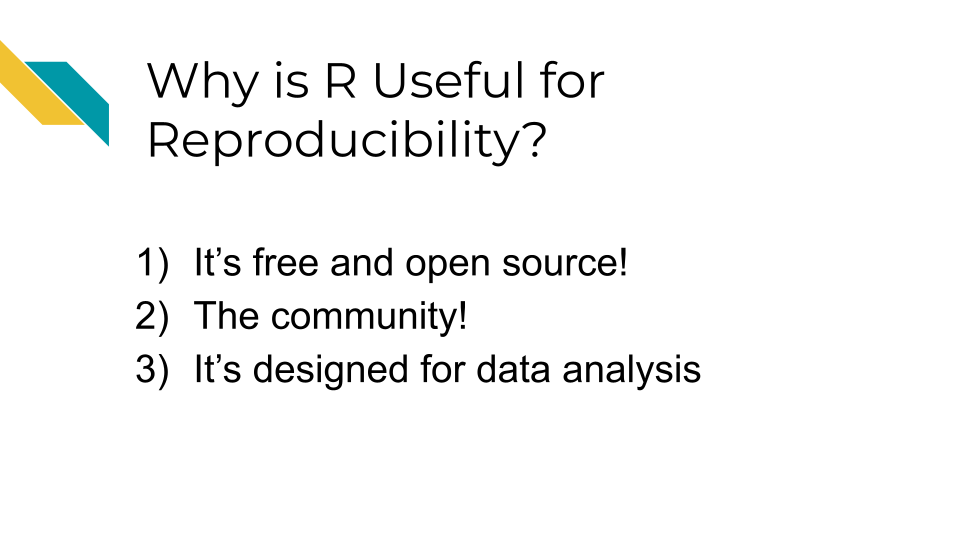


## 2.2 Why R



[R](https://www.r-project.org/) is a [programming language](https://en.wikipedia.org/wiki/Programming_language) for working with data, performing statistical analyses, and for creating plots and graphics that was developed in 1991 by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand ([“R (Programming Language)” 2023](#ref-r_2023); [“R: The R Project for Statistical Computing”](#ref-r_project)). Countless contributors have made R what it is today.

There are some especially useful aspects about R that make it a great option for creating reproducible data analyses.



## 2.3 It is free and open source

The first is that R is free and [open source](https://opensource.com/resources/what-open-source).



The term **open source** means that the code is publicly available. Thus all of the code involved in creating R is actually publicly available! This enables users to check what code is used in a particular **package** (a set of code that allows you to do various things) so that they can modify or build upon the code if they would like to.

In fact, many users create their own R **packages** to share their code with others. There are places such as the Comprehensive R Archive Network ([CRAN](https://cran.r-project.org/)) and elsewhere that allow users to publish their own packages for others to use.

* **programming language** - A specified set of notations to tell a computer what to do
* **R** - Programming language for working with data to perform statistical analyses and for creating plots and other graphics
* **open source** - Code is publicly available
* **R package** - A set of code that can be shared between users

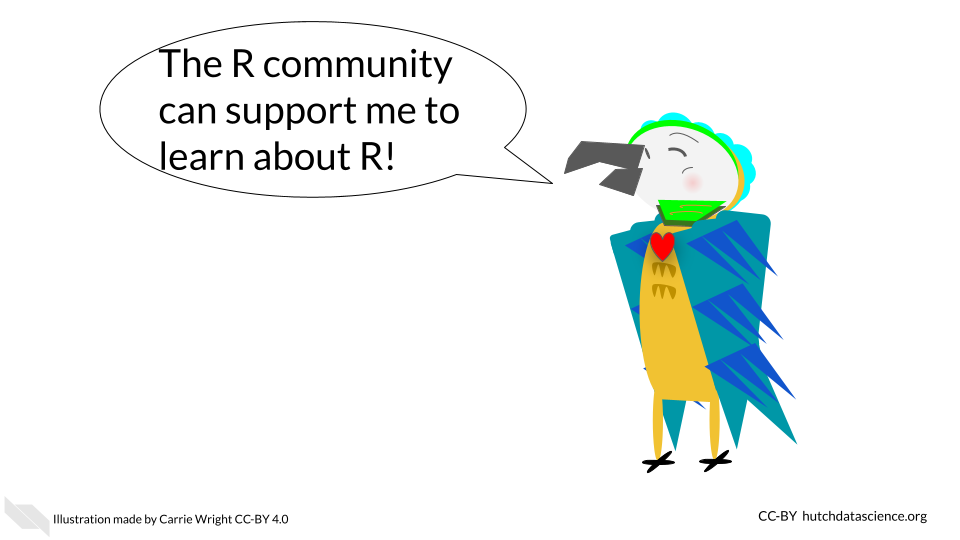
Why are these aspects good for reproducibility?

* Since R is free, it is accessible to anyone. Therefore, anyone could run your code if you shared it with them, without them needing to buy software.
* Since R is open source, if you use packages from others, people can determine what underlying code your code used (if you tell them what version you used - more on that later!)

## 2.4 The community

R has a very rich and active community!

This makes it easier to reach out to others for help, find support, find tutorials, and more.



There are several R community groups that are especially helpful:

* [R Ladies](https://www.rladies.org/) - a support group that is not just for ladies, but is open to anyone who wants to improve their R skills! There are local chapters in many large cities that often have in-person meetings.
* There are lots of useful resources, such as the [R for Data Science book](https://r4ds.had.co.nz/) (written by two developers at Posit (formally called RStudio) which develops lots of core R packages), resources and online courses from the [Johns Hopkins Data Science Lab](https://jhudatascience.org/courses.html) including [Open Case Studies](https://www.opencasestudies.org/), resources and workshops from [Data Carpentry](https://datacarpentry.org/), [Dataquest](https://www.dataquest.io/v2/), [DataTrail](https://datatrail-jhu.github.io/DataTrail/) and more!

See this [link](https://jhudatascience.org/intro_to_r/resources.html) for more R resources.

Why is this rich community good for reproducibility?

* Overall your code has a better chance of being more accessible than if it were written in a language that is not open source or that has limited support.
* You can also find support to make sure your code does what you want it to, as well as support to make your code as reproducible as possible.

## 2.5 Designed for data

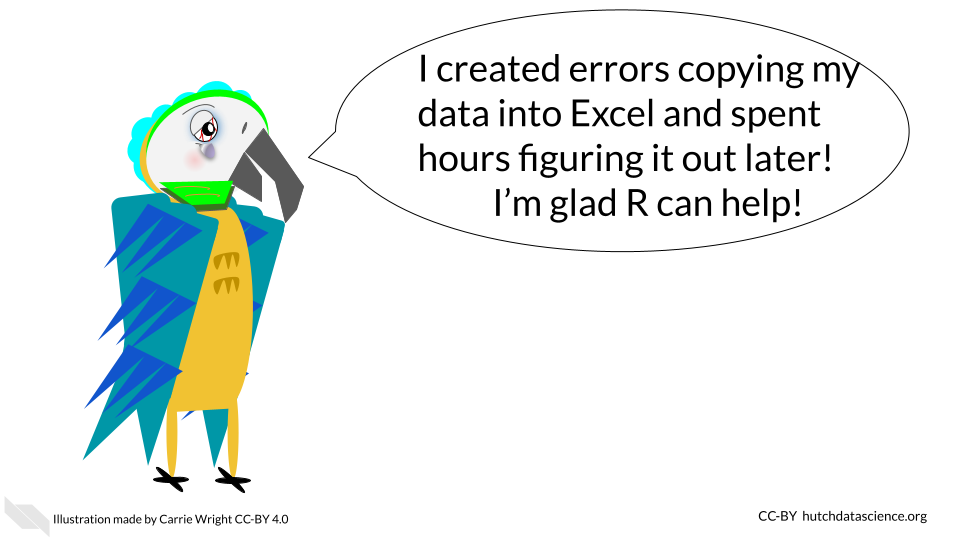
R is a statistical programming language, meaning it was designed to help you analyze data. It is the main focus of the language. This is one of the major advantages of using R over other programming languages that have more general purposes.

Because of this many people have designed useful packages that are especially relevant to:

1. Dealing with messy data in a systematic and reproducible way to get it into a state that is useful for data analysis
2. Producing statistical analysis of data
3. Creating effective plots of data

Although other options like [SPSS](https://www.ibm.com/products/spss-statistics) and [SAS](https://www.sas.com/) (which are not free!) can also be helpful for statistical analysis, R is especially powerful at getting messy data ready to analyze and for creating useful plots to represent patterns in data. Conveniently, R can do all of these steps in a data project and does not require users to switch between different programs to perform these tasks. R also helps create reports that can demonstrate to collaborators and others exactly how analysis was performed, aiding in the transparency of how the data was used from start to finish.

R can also import data from many different sources that other statistical software can’t handle (including scraping data from websites or [PDFs](https://www.adobe.com/acrobat/about-adobe-pdf.html)). This allows users much more flexibility to use data as close to the source as possible. This can enable users to stop copy and pasting data and reduce the risk of human error. If you are interested, see [Open Case Studies](https://www.opencasestudies.org/) for more guidance on importing many different kinds of data.



Why are these design features especially helpful for creating reproducible analyses?

1. It enables users to work with messy data and get it ready for analysis, as opposed to requiring users to use other programs. The tidyverse a suite of very helpful packages has many data wrangling packages that are especially intuitive for others to read and understand your code.
2. Users can create effective plots using the same program as for data prep and analysis. The ggplot2 package is famous for making really effective and customizable plots.
3. It helps create reports that can show the entire data analysis process from importing the data to making plots. R Markdown reports are very helpful for this.
4. It is easier to import data closer to the original source, rather than converting files or copy and pasting data, which can result in accidental modifications of the data.

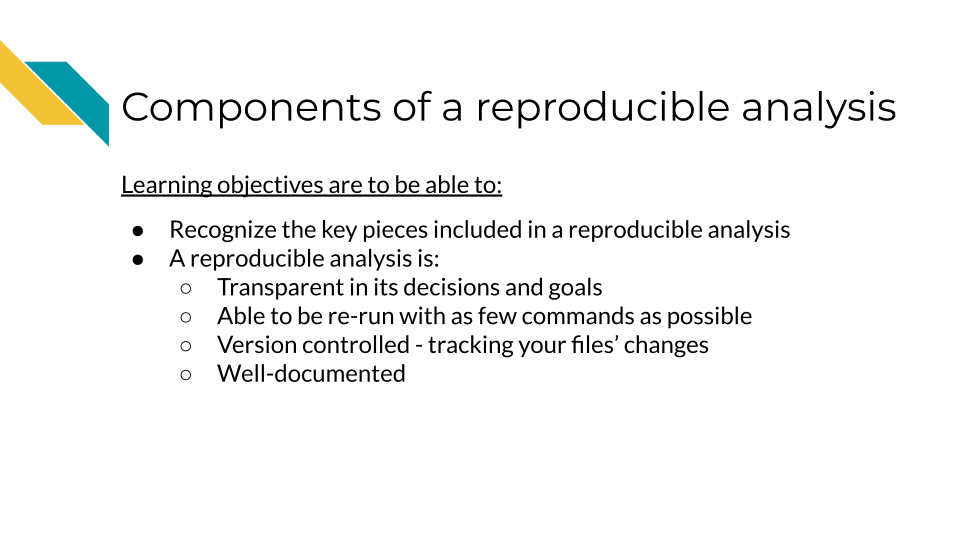
## 2.6 Conclusion

In summary, R can be especially useful if you want to make your data analyses more transparent and reproducible for the following reasons:

1. It is free and open source, meaning that code that you might incorporate in your analyses is accessible to anyone. Secondly, others can use your code without needing to buy software.
2. There is a rich R community that can help you make the most out of your code and learn how to write your code in a more reproducible manner.
3. R is particularly powerful for preparing data for analysis and for creating visual representations of data. Beyond being free, these unique benefits make R a particularly good statistical tool.
4. R is especially designed to analyze data and for the entirety of the process, which makes it great for creating transparent information about how you actually worked with data from start to finish.

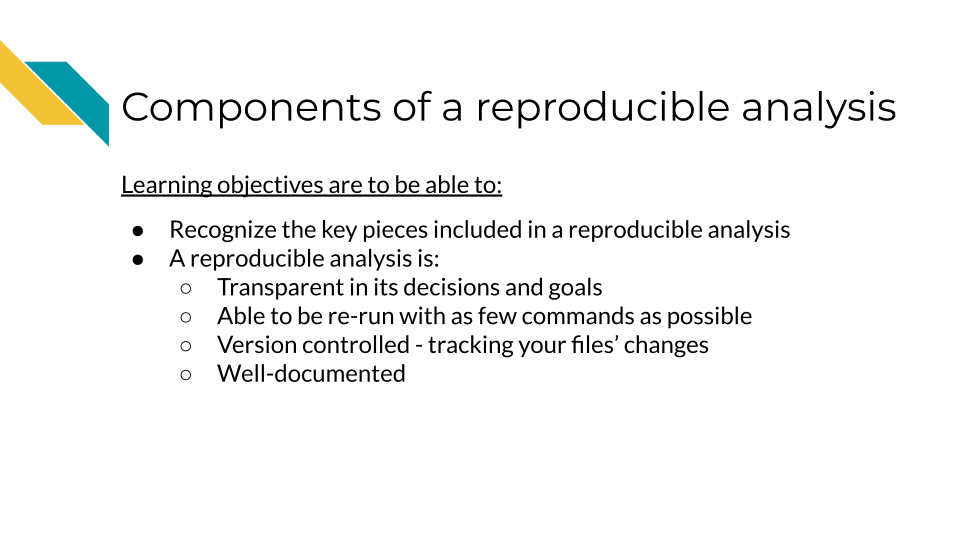
# 3 Components of a reproducible analysis

In this chapter, we will discuss what components of an analysis make it reproducible.



## 3.1 Reproducibility is iterative work

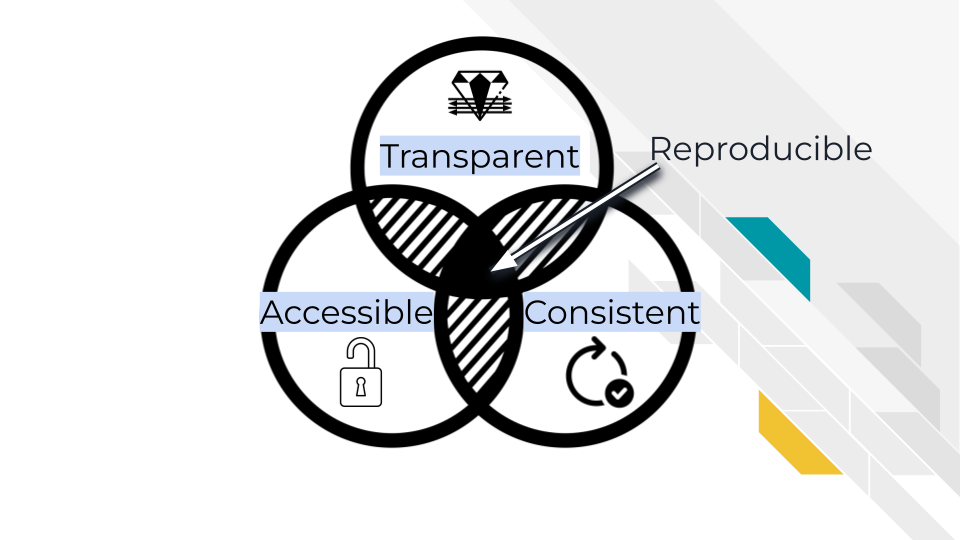
Making an analysis isn’t something that happens on the first try. Working on a project iteratively and continuing to improve the reproducibility of it is the best approach. In this manner, we can view reproducibility on a continuum. Some projects are just run once but aren’t really needed anymore, and don’t become very polished or reproducible. But as we continue to work on a project and polish its reproducible components, it continues to be more perfected. However, because of the moving nature of some reproducibility components, no project is really perfectly reproducible in every context throughout time.



## 3.2 Components of reproducibility

A reproducible analysis is transparent, consistent, and accessible.

* **Transparency** refers to the idea that it is well communicated and everything is displayed: data, code, goals, methods, and decisions. There are no secrets in a reproducible analysis/
* **Consistency** refers to the idea that the code can be consistently run, but also everything follows a particular system, conventions and design.
* **Accessibility** refers to the idea that anyone anywhere should be able to run and/or examine the analysis. No pay walls or expensive software should be required.



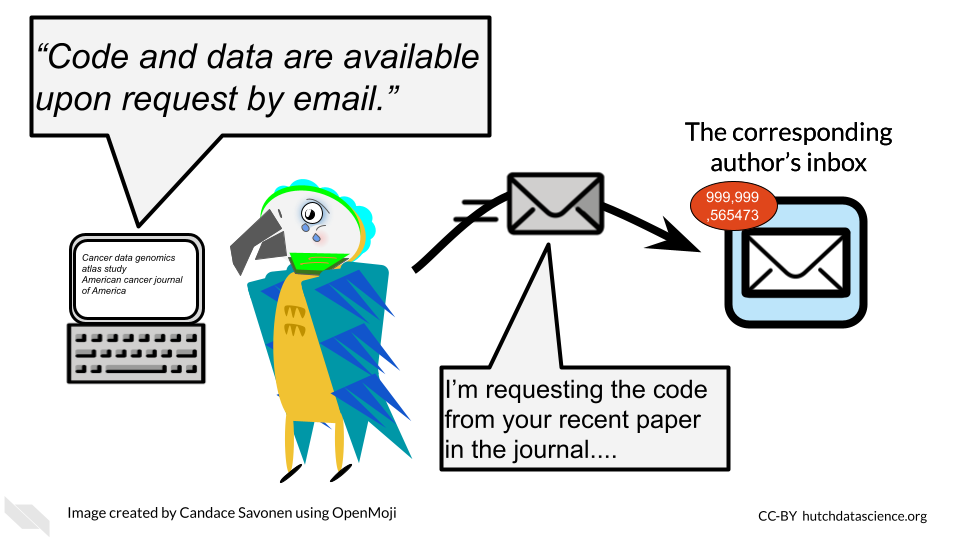
## 3.3 Transparent

One essential piece of a reproducible analysis is that the code runs reliably. However, to really make an analysis reproducible, the decisions made in the analysis should also be clearly communicated. A transparent analysis is not only well communicated, but also shared publicly in a way that others can comment and contribute ideas and suggestions to or borrow methods and strategies for their own analyses.

### 3.3.1 Open source

Open source means not only making code and data publicly available, but also enabling others to modify or comment on the code. This doesn’t mean that any and all modifying contributions need to be accepted, because some level of standards and quality checks need to be maintained by the owners of the analysis, but just that anyone online could propose a contribution if they wanted to.

For an analysis to be truly open source, it needs to be easily accessed by others and stored online. Code that can be emailed, for example, is not considered open source.



For reproducibility, keeping your code on [GitHub](https://github.com/) is a great open source solution. GitHub is a code hosting platform that allows people to access code and sometimes data. It is commonly used, and has a built in system that allows others to contribute changes in a way that can be methodically reviewed by you (this is called the pull request system and we will talk about it more).

**GitHub** - An online platform for sharing and managing code and files in an open source manner

### 3.3.2 Data is publicly available

A transparent analysis has data that is publicly shared so that others can re-run the analysis as you have. Data should be provided in a way that it can be programmatically accessed (downloaded by a script). Data also need to be well-documented in the form of metadata.

Data sharing is a critical piece for promoting the open sourceness of your analysis, however this often needs to be balanced with privacy if you work with human data or samples. These data will likely contain personal identifiable information (PII) and protected health information (PHI). For more details on this, we encourage you to see this [course about data management](https://jhudatascience.org/Ethical_Data_Handling_for_Cancer_Research/data-privacy.html).

While it’s imperative that you protect human data, that doesn’t mean that your analysis cannot be publicly shared! These are not mutually exclusive goals, but will take a bit of thoughtful planning. In the upcoming chapters we will provide additional ideas and information for how you can conduct an open source analysis while appropriately protecting sensitive data.

### 3.3.3 Readable code

Readable code is much more important than clever code. If you are the only one who knows what your code is doing, it will not only be difficult for others to contribute or vet your analysis, but in the future, you will probably not understand what your code is doing either.

[Read this course chapter from the ITCR training network about how to write durable code](https://jhudatascience.org/Reproducibility_in_Cancer_Informatics/writing-durable-code.html).

### 3.3.4 Well-documented

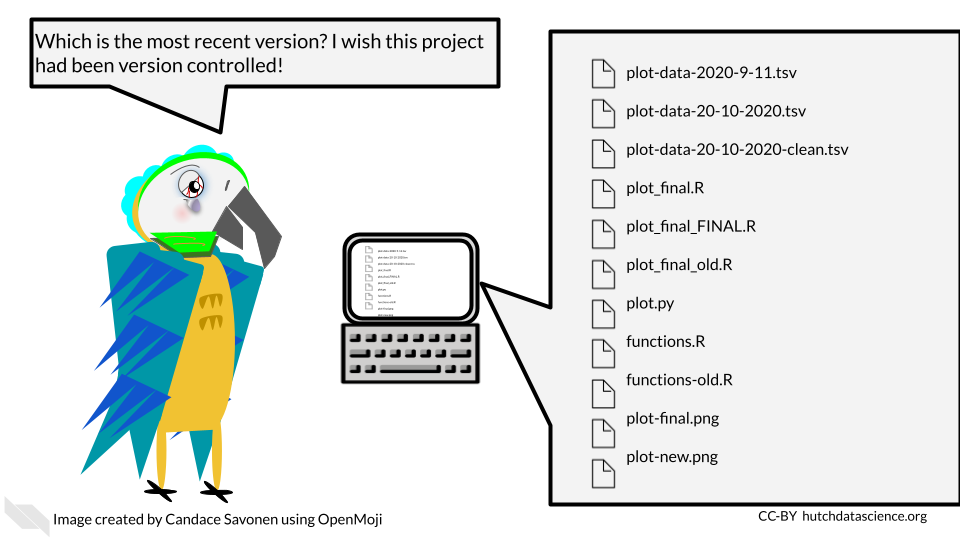
A well-documented analysis is a reproducible analysis. If analyses didn’t require a lot of decisions and human comprehension than documentation wouldn’t be necessary – but also a lot of data analysts would be out of a job because robots would be able to do it! Analysts and developers often think of documentation as an after-thought, but good documentation should be actively developed along with the code. Arguably, it is more important to have clear documentation than even working code, because if broken code is well-documented, others may be able to help make suggestions for how it can be fixed.

Good documentation not only describes what happened in an analysis, but why it happened – why did the analyst choose this method or parameter as opposed to others? Was there an additional analysis, literature, or other resource that led us to this conclusion? Documentation should describe not only what is happening, but the thought process that led us here.

### 3.3.5 Version controlled

A reproducible analysis is a version controlled analysis. Analyses go through many iterations, side quests, and occasional dead ends – and this is okay, it is how data science works! – but if not done properly with version control, this can lead to an unruly code base and a lot of confused team members. Version control is a method for tracking changes to files in a systematic manner. One such method of version control is called git and we will talk about how to use git and its online website GitHub, in a future chapter.

Version control helps maintain the history of your project in a way that will allow you to recover old versions if necessary, or otherwise have documentation on what has happened. It can also be useful for rectifying different versions of a code base between team members.



**version control** - A method of tracking and handling files as they are changed over the course of a project

## 3.4 Consistent

A reproducible analysis is consistent. It should consistently run and consistently produce the same results. It should also be written in a manner that follows a consistent style and project organization scheme.

### 3.4.1 Re-runs consistently and easily

Ideally, a reproducible analysis should be able to re-run with one command that is explicitly stated in a README file. This is a file that explains what all the rest of the files are and the point of the project. If an individual has a copy of the analysis project, it should include everything that is needed to re-run that analysis and the number of steps needed to re-run the analysis should be the lowest number possible. The more steps that are needed, the less likely it will be that someone will be able to reproduce the analysis.

This also generally means that analyses that can be performed through programmatic scripts are more reproducible than those performed by GUI’s (graphic user interfaces). GUIs are programs on computers that are used by pointing and clicking buttons whereas command line programs are used by typing in commands. Command line programs generally take scripts that allow you to have each step written in the script which can be easily recalled to re-run the entire analysis. Most GUI’s, although sometimes more intuitive to use, are unfortunately less reproducible because they require more manual steps by clicking various buttons.

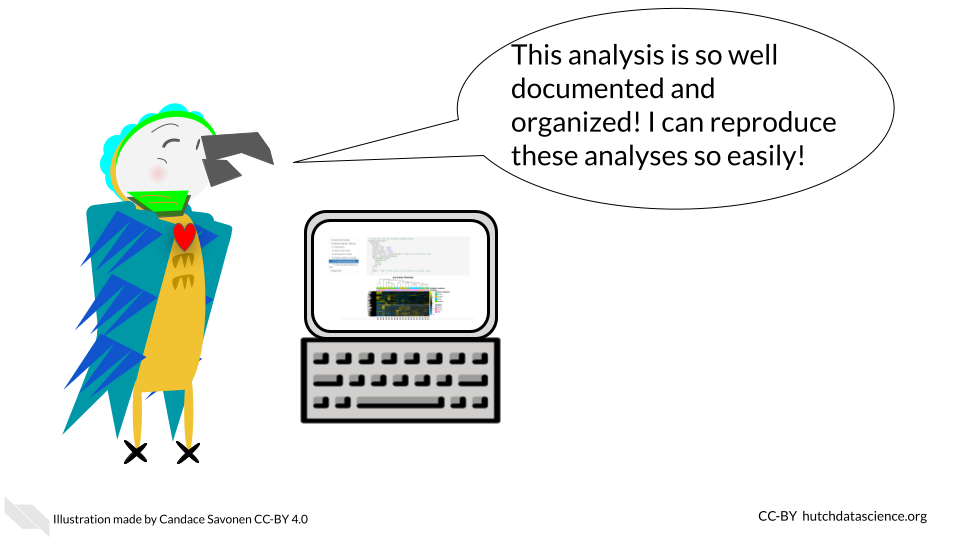
**GUI (graphic user interface)** - A type of program on a computer that you use by pointing and clicking with a mouse **Command line** - A type of program on a computer that you use by typing in commands or writing scripts that can be run

### 3.4.2 Follows a code style

Code style is important because it not only makes code more readable, but it also lends a certain confidence to the reader of the code, that this code has been thought through and perhaps polished more than code that is less consistent in its style.

### 3.4.3 Have an organizational scheme

Project organization is a major component of reproducibility. If you are not able to find your files, then chances are individuals who are attempting to reproduce your analysis also will not be able to understand where to find things. We will discuss in a later chapter strategies for keeping projects organized, while realizing that project organization is an ongoing, dynamic task.

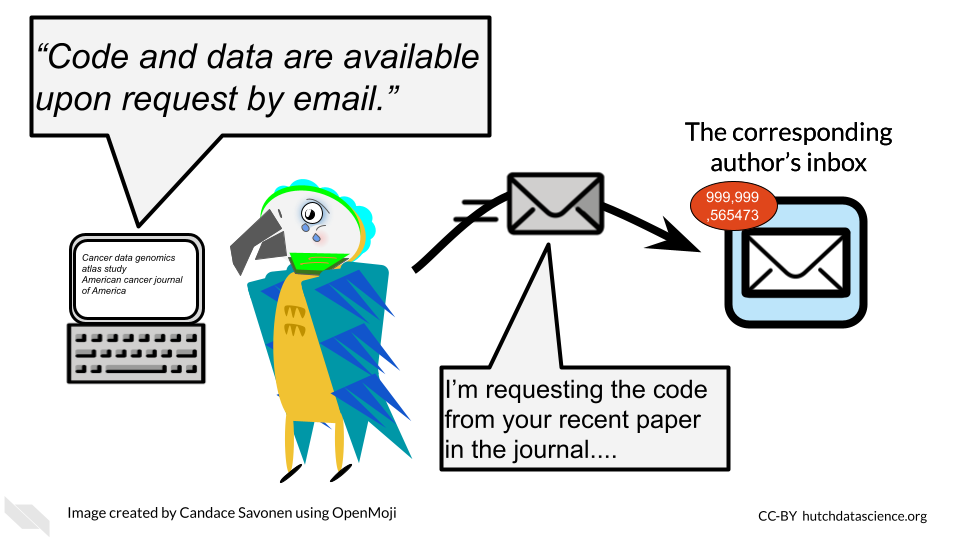


## 3.5 Accessible

We discussed that we use R because it is open source and free. This makes it conducive for making reproducible analyses. Accessibility is important for reproducibility. This means minimizing the number of hoops others have to jump through to re-run your analysis.

Accessibility also involves prioritizing democratizing science and enabling as many people as possible to understand what you did for your analyses. We encourage you to realize that science does best when everyone has access to it and that includes code and data analyses! Making your data and code accessible, allows everyone to contribute and learn from your analysis. Note that if you are concerned about being scooped, you can make your code private on GitHub while you are working on it and then make it public once you release a preprint of your results. We will talk more about this later.

Accessibility means that anyone should be able to access it – whether or not their funding is in ample supply. So be sure to publish in code repositories that do not require membership fees or any other kinds of paywalls. Make an effort to publish in journals that are freely available as well.



Sometimes even if something is accessible in that it is “free” monetarily it doesn’t mean that it is free in the sense of the amount of time it takes to access it. If your code and data does need some sort of controlled access features for privacy and ethical concerns of protecting data, make sure that the paperwork hoops that are put in place are truly there in the spirit of protecting the data and not instead to keep data and code hidden from others.

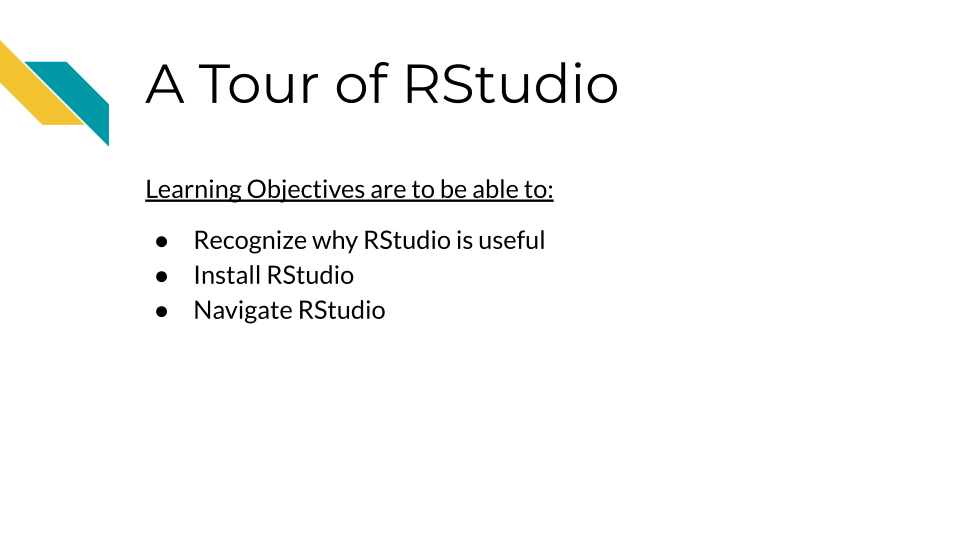
## 3.6 Conclusion

In this chapter, we gave a high level overview of reproducible analyses. We discussed that reproducible analyses are transparent, consistent, and accessible. This means in practical terms, reproducible analyses:

* Are open source
* Have data that is publicly available (when appropriate)
* Have readable code
* Are well-documented
* Re-run easily
* Have an organizational scheme
* Follow a code style
* Do not have paywalls or other barriers (except for ethical or data privacy reasons)

# 4 A Tour of RStudio

In this chapter we will talk about a very useful R-related tool called RStudio. RStudio is an environment for using R that can be extremely helpful for writing code and making your analyses reproducible.



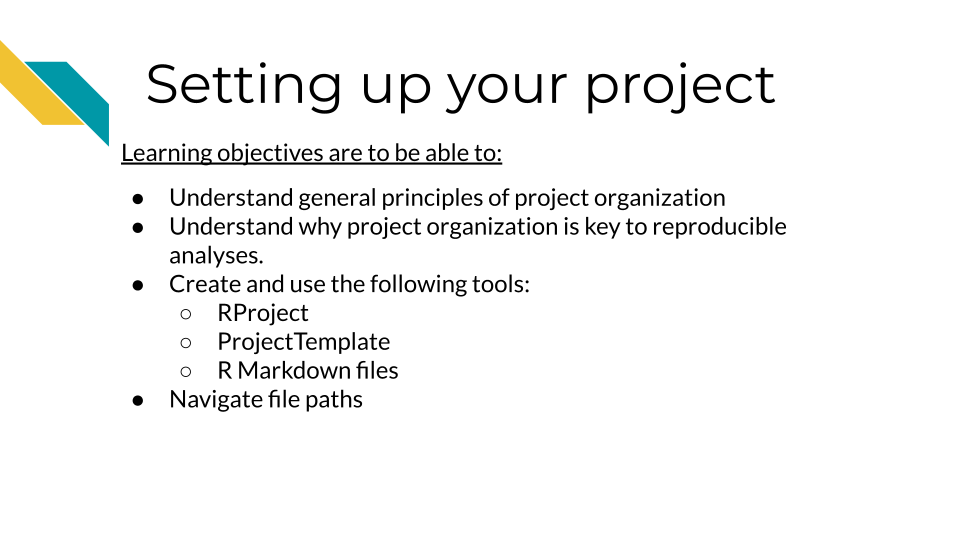
## 4.1 Why use RStudio?

RStudio is what is called an integrated development environment(IDE) for writing code in R (although it also has compatibility for other languages).

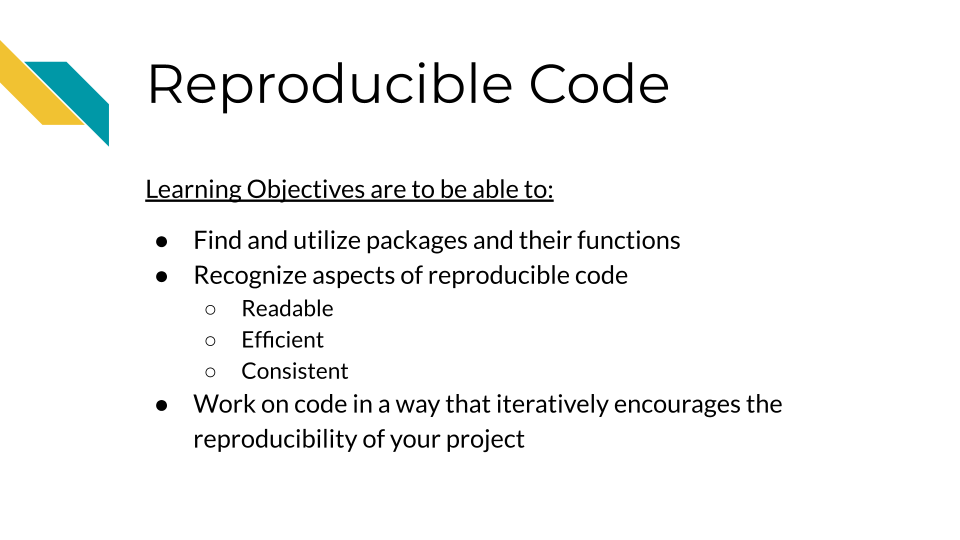
It is designed to make working in R easier in a variety of ways:

* it helps you write code by suggesting code based on what you have written - currently this is mostly for suggesting package names or functions (which are specific pieces of code that accomplish a particular task, often packages have several functions)
* it helps you view the output of your code in an easier manner, especially plots
* it helps you find errors in your code
* it helps you keep track of any objects that you have assigned in R
* it helps you orient yourself in terms of the files on your computer
* can be used to help track changes in your code and other files over time
* **IDE** - Integrated Development Environment - a computing environment for writing code, debugging code, and looking at the output of your code
* **RStudio** - an IDE designed especially for writing R code
* **function** - a specific piece of code that performs a task - packages in R often have several functions

# 5 Setting up your project



# 6 Reproducible Code



## 6.1 Reproducibility means we don’t need to reinvent the wheel!

When you realize something that you need done, you should first use Google and look on GitHub and StackOverflow to see if someone else has written something that works really well. Where at all possible, borrow good code and attribute the author – no need to reinvent the wheel. As we said previously, R has a great community of users who are constantly creating new and great code, often in the form of packages that are ready for you to install and use.

*This is the beauty of reproducibility, if you or someone else makes great code, it can not only be re-run but it can be repurposed!*

### 6.1.1 Tips for choosing packages to use:

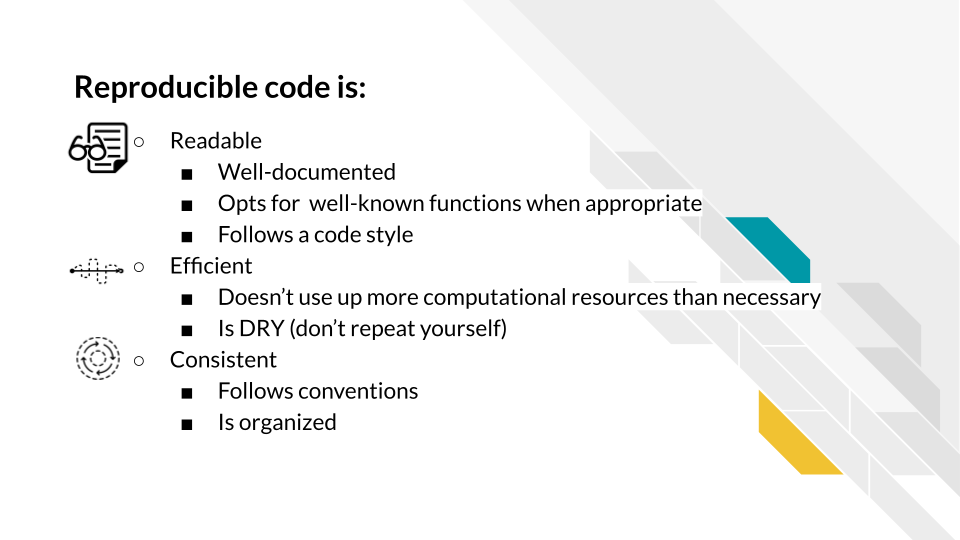
* Does the package have easy to use documentation to help guide you on how to use it properly? If the package has underdeveloped documentation it may be difficult for you or others to understand and use properly.
* Is the code actively being developed or maintained? Packages that are no longer being maintained will likely deprecate more quickly rendering them unusable in the future.
* Is it a package that is commonly recognized by the community? Well recognized packages will be easier for others to comment on and help you with.

## 6.2 The importance of iterative work

We’ve mentioned previously that reproducibility is iterative work. This way of working refers to code work as much as anything else. You won’t ever write perfect code on the first try, instead aim for each chunk to work one step at a time. Once it is working, take a break (perhaps until the next day), then return to it and look for ways to polish it and make it more efficient.

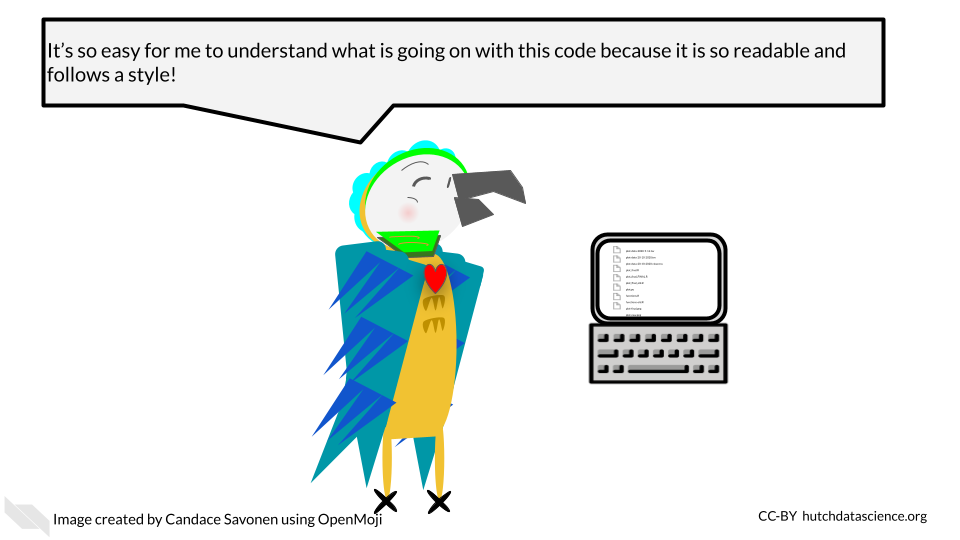
[For more tips on how to work read this blog](https://www.ccdatalab.org/blog/not-so-secret-sauce-for-efficient-workflows).

## 6.3 Aspects of Reproducible code



## 6.4 Readable

Reproducible code is readable code. In order for other people to use your code, they will need to be able to understand it. Because of that, code being readable is more important than code being innovative or clever.



### 6.4.1 Well-documented

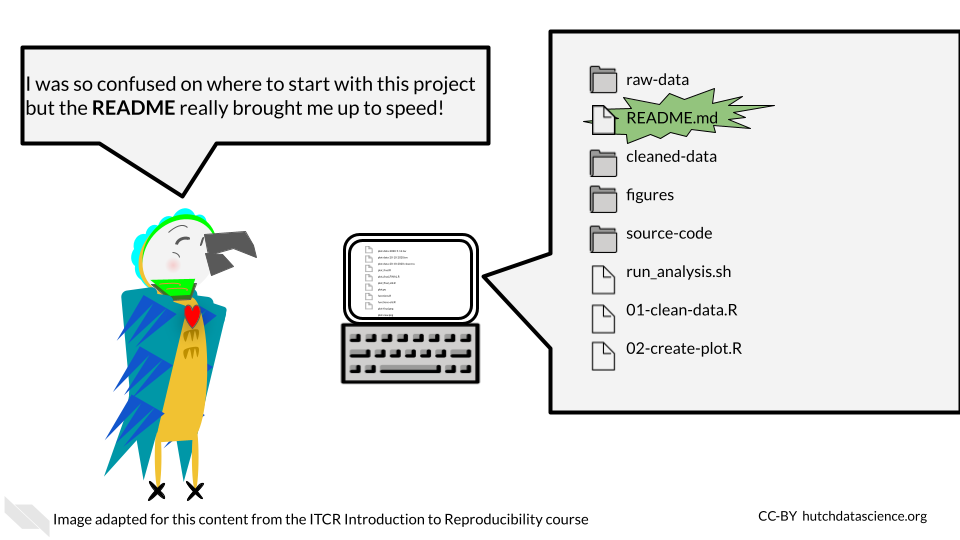
Reproducible code is well documented code! This includes (but isn’t limited to):

* A README that can get individuals up to speed on the project quickly
* Code and notebooks that have a healthy amount of comments

These bits of documentation are not only helpful for others reading your project but for you! As time passes, future you will forget what you of today was thinking when you wrote this code. Helpful code comments can help jog your memory of what the code is doing and perhaps what the next steps in the project need to be.

#### 6.4.1.1 READMEs

READMEs are a universal signal to people looking at the project that they should READ this file to get a rundown on the project.



READMEs should include:

1. A summary of the goals and intentions of the project.
2. Usage instructions that explain exactly what commands and packages need to be used to re-run analyses.
3. Explanation of what software dependencies are needed for your project.
4. A basic summary of what files are there; which are input and output files.
5. Any other information that would be relevant to someone trying to understand the project.

Here’s a [template README](https://raw.githubusercontent.com/jhudsl/Reproducibility_in_Cancer_Informatics/main/resources/README-template.md) that you can use an example.

* **README** - A file in a project that has the start up summary information that could get someone acclimated to the project.

#### 6.4.1.2 Code comments

A healthy amount of code comments doesn’t mean that every line needs a comment (though perhaps at sections that need future explanation they might). Helpful code comments don’t just echo what the code is doing but are explanatory. [StackOverflow has a great article](https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/) about rules for writing helpful code comments. we’ll echo the rules here:

Rule 1: Comments should not duplicate the code.

Rule 2: Good comments do not excuse unclear code.

Rule 3: If you can’t write a clear comment, there may be a problem with the code.

Rule 4: Comments should dispel confusion, not cause it.

Rule 5: Explain unidiomatic code in comments.

Rule 6: Provide links to the original source of copied code.

Rule 7: Include links to external references where they will be most helpful.

Rule 8: Add comments when fixing bugs.

Rule 9: Use comments to mark incomplete implementations.

### 6.4.2 Follows a code style

Code style helps make code readable. Appropriate spacing, punctuation, and grammar are not always essential for getting a message across, but it can certainly b3 dis-tRaCTIng to readers if conventions aren’t followed.

**Basic Example:**

# Bad: Should use <- and have a variable name that is informative  
x = c(1, 4, 5, 10)  
  
# Bad: Irregular spacing is distracting  
numbers<- c(1, 4,5,10)  
  
# Good!  
numbers <- c(1, 4, 5, 10)

Here’s some style guides you can use:

* [Google’s R Style Guide](https://google.github.io/styleguide/Rguide.html)
* [Tidyverse Style Guide](https://style.tidyverse.org/index.html)

[R packages like styler](https://www.tidyverse.org/blog/2017/12/styler-1.0.0/) can automatically style code for you.

## 6.5 Efficient

Reproducible code is efficient code. Efficiency helps reproducibility in that code that takes up less resources and is not redundant can be re-run and debugged more easily.

### 6.5.1 Doesn’t use up more computational resources than necessary

R is not meant to be a fast language. R code can be computationally costly if it’s written in the wrong way.

You can identify what parts of your R code are the slowest or otherwise computationally costly by using the [profvis](https://rstudio.github.io/profvis/) and [bench](https://bench.r-lib.org/) packages.

One popular example is R loops which can be particularly slow in R. Note that this doesn’t mean you shouldn’t ever use loops or other items in R, just that you should be aware that some items in R are particularly slower than others.

**R ‘for loop’ alternatives:**

* [Using apply functions (an older option that comes with the basic installation of R)](https://statisticsglobe.com/avoid-for-loop-in-r)
* [Advanced R discusses alternative strategies](https://adv-r.hadley.nz/perf-improve.html?q=perfor#avoid-copies)
* [The across function](https://dplyr.tidyverse.org/reference/across.html)
* [Using the purrr package](https://purrr.tidyverse.org/)
* [A nice summary of these functions](https://jhudatascience.org/intro_to_r/modules/Functions/Functions.html#18)

**Further reading:**

* Hadley Wickham has a [great chapter in Advanced R](https://adv-r.hadley.nz/perf-improve.html?q=perfor#perf-improve) that covers these concepts in more detail.

### 6.5.2 Is DRY (don’t repeat yourself)

DRY is an acronym for “don’t repeat yourself”. Non-redundant code is more reproducible because it is easier to maintain and to read.

Let’s take a look at an example from [this Reproducibility in Cancer Informatics course](https://jhudatascience.org/Reproducibility_in_Cancer_Informatics/writing-durable-code.html#dry-up-your-code) about what DRY vs non-DRY code might look like:

Non-DRY or WET (write everything twice) code might look like this:

paste('Hello','John', 'welcome to this course')  
paste('Hello','Susan', 'welcome to this course')  
paste('Hello','Matt', 'welcome to this course')  
paste('Hello','Anne', 'welcome to this course')  
paste('Hello','Joe', 'welcome to this course')  
paste('Hello','Tyson', 'welcome to this course')  
paste('Hello','Julia', 'welcome to this course')  
paste('Hello','Cathy', 'welcome to this course')

Note that if you want to change something in eight of these messages you would have to change all eight lines.

To DRY up this code, we could functionalize it:

GreetStudent <- function(name) {  
 greeting <- paste('Hello', name, 'welcome to this course')  
 return(greeting)  
}  
  
class\_names <- c('John', 'Susan', 'Matt' ,'Anne', 'Joe', 'Tyson', 'Julia', 'Cathy')  
  
lapply(class\_names, GreetStudent)

Now, if we wanted to edit the greeting pasted, we’d only have to change it once.

* **DRY code** - Code that doesn’t repeat itself and because of that is more efficient

## 6.6 Consistent

Consistency is key for reproducibility. Not only do we want code to run consistently, but it will be more understandable to our future selves and to others if it follows a pattern.

### 6.6.1 Follows conventions

Although there’s always a time to break conventions, often times conventions lend to readability. For example, in R using <- for assignments is less likely to be distracting than using =. This is related to following a style guide. In general we recommend using the tidyverse conventions and style.

However, this advice, like a lot of the advice in this chapter is highly dependent on the context and goals of the project and code being written. There are times that the conventional way to write something might not suit the project because it is inefficient or otherwise clashes with other goals of reproducibility.

### 6.6.2 Is organized

In the previous chapter, we discussed how projects should be organized in order to be reproducible, but this also applies to code. Sometimes as you have been working on code, you may realize that as it has been developing it doesn’t flow in an organized manner.

Just as with regular writing that is disorganized, code that is disorganized can be hard to follow and hard to bug. For example, if you have hard coded a multiple variables or have loaded multiple packages, it makes sense to group these items together so they are easier to find and fix.

A code outline for an analysis notebook for example might look like:

* Describe the goals
* Load in the libraries and any source code
* Declare any hard coded variables
* Read in the data
* Clean the data
* Make plots and gather statistics
* Summarize results
* Print out the session info we’ll discuss more about this in a future chapter.

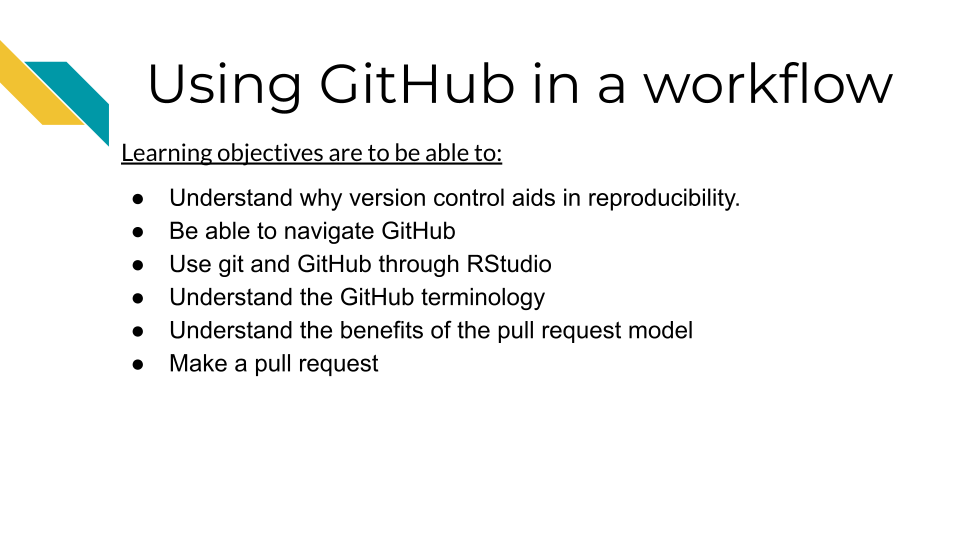
## 6.7 Conclusion

The best way to find out if your code meets these concepts or how it can better become more reproducible is through code review. We will briefly discuss code review in future chapters.

In summary, reproducible code is:

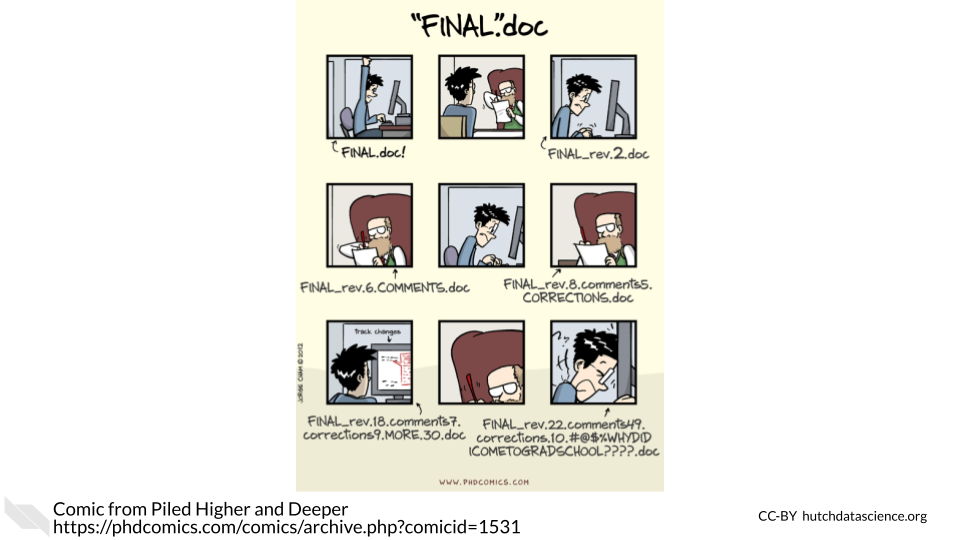
* Readable
  + Well-documented
  + Follows a code style
* Efficient
  + Computationally non wasteful
  + DRY
* Consistent
* Follows convention (when appropriate)
* Organized

# 7 Using GitHub in a workflow



## 7.1 What is version control

Version control is system that allows you to track your files as you work on them. Whether you’ve wrote a lot of code or written other documents, you’ve likely encountered the need for version control. As this comic from Piled Higher and Deeper describes, files can go through a lot of edits and revisions (this is true of code too)! And it doesn’t take long for it to be difficult or even impossible to track the various revisions of even one file let alone dozens or hundreds of files that may be a part of a data science project.



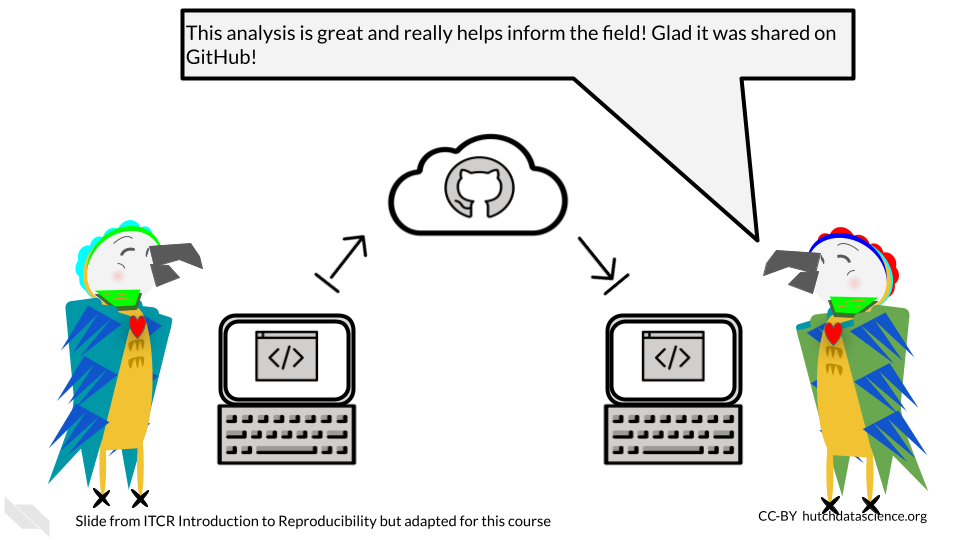
This problem is what git, a version control system can address. It is a system that allows you to track your files, keep a history of them and otherwise handle changes through the history of your project.

## 7.2 What’s GitHub?

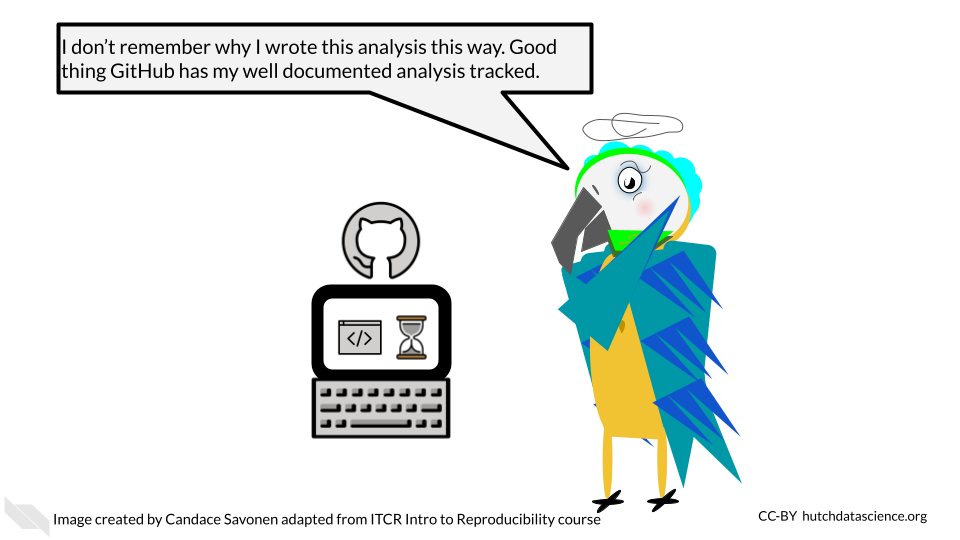
The most common way git is used is in conjunction with an online platform called GitHub. [GitHub](https://github.com/) is an online platform for sharing code.

* **version control** a method of tracking files as they are changed throughout a project.
* **GitHub** an online platform for sharing code in a version controlled manner.

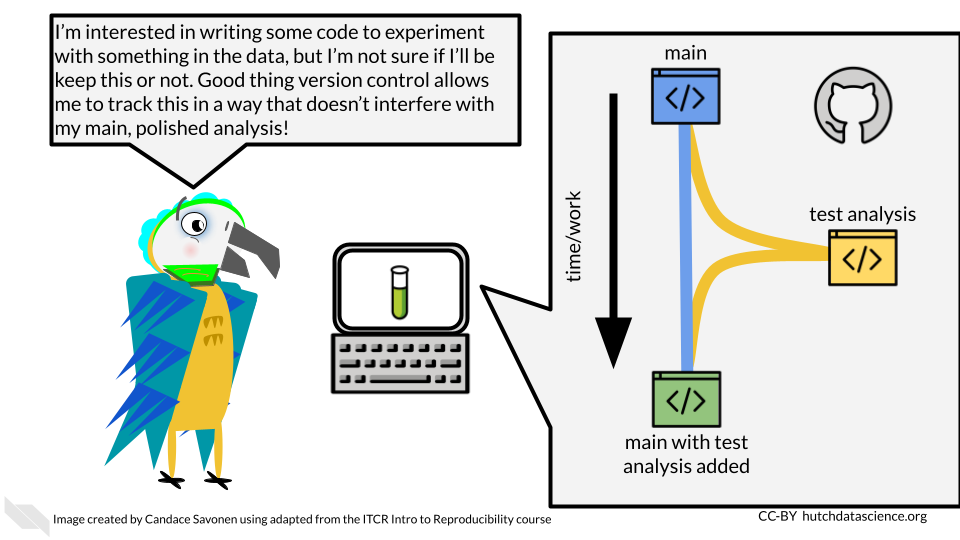
It aids reproducibility by being online in a way that is easily shareable to others in a version controlled way. GitHub allows anyone at anytime to take a look at and obtain your code.



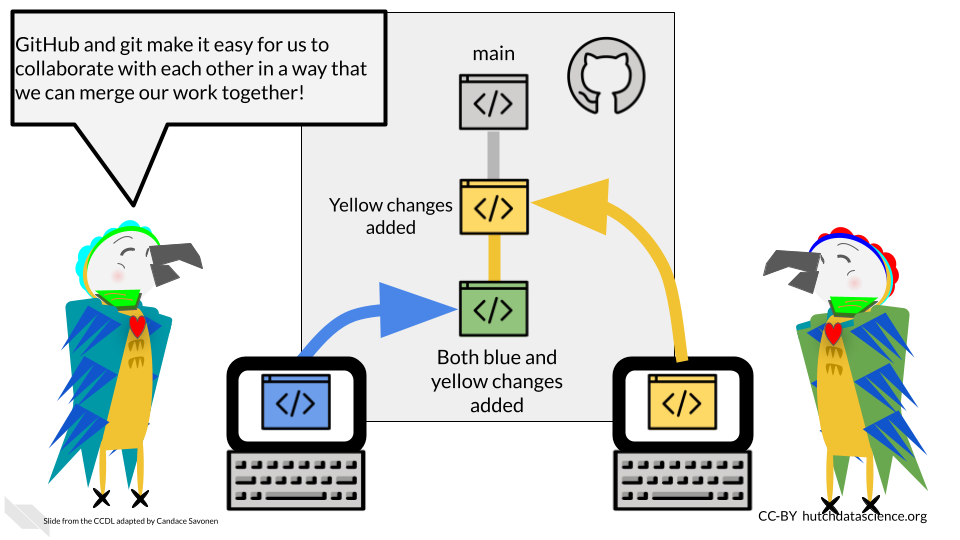
Because code on GitHub is version controlled, it allows you to track your code and project files as you and others continue to work on them. This can be really useful for documenting not only the changes to your analysis, but the rationale and communications that led to those changes.



GitHub and its version control also allows you to be able to take the side journeys that often pop up with data science projects but in a way that allows you to ensure that the main files stay safe as you experiment.

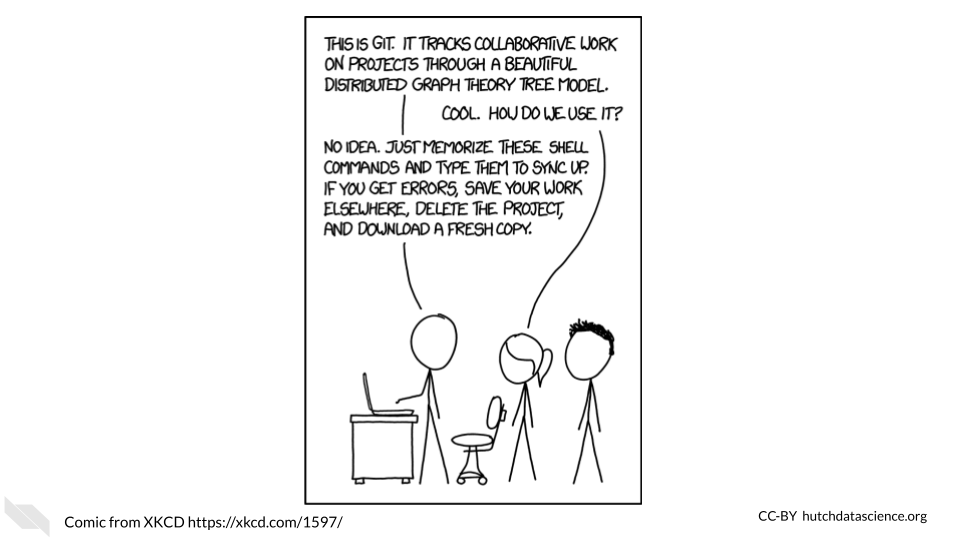


GitHub is also handy for collaborating with others on your code, not only because it is online, but the version controlled nature of it, allows you and any team members to work on the same files simultaneously without fear that the changes will be impossible to merge together. Git and GitHub have systems to do just this.

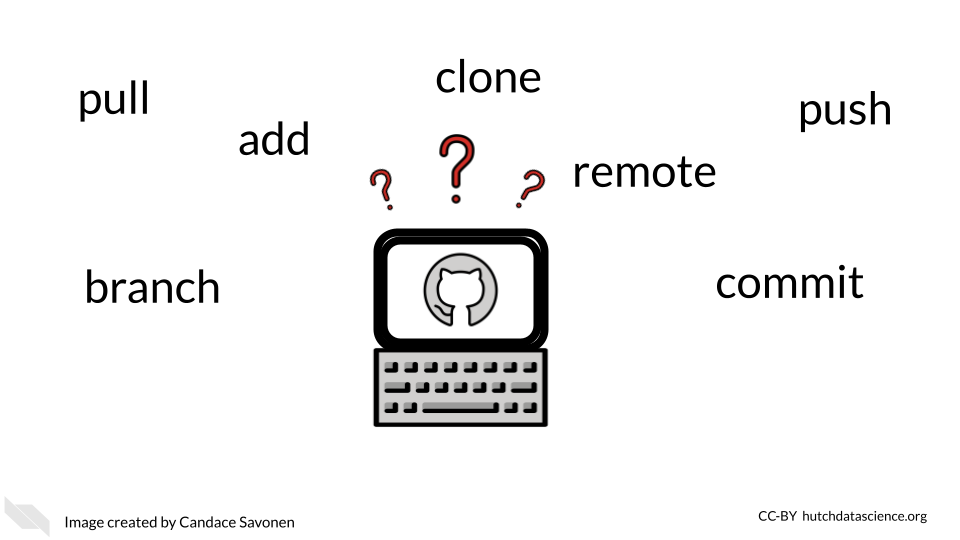


## 7.3 GitHub Workflow

Git can feel overwhelming to a lot of folks and it has a really deep and complicated system. However, truthfully for most instances you will only need the same few commands in the same series of steps which we will cover here.



One reason that git and GitHub can feel overwhelming to folks is that there are a number of terms that are used to describe the different commands. We will walk through the typical workflow steps, and define these terms as we use them.



### 7.3.1 Create your GitHub account

Before we get started with GitHub, if you do not have a GitHub account, [go here to make one.](https://github.com/)

### 7.3.2 Creating a new repository

Whenever you are starting a new project you will want to start by creating a new repository on GitHub. You can generally do this by going to GitHub and [choosing “New repository” from the menu](https://github.com/new).

* **repository** - a group of project files

On this new repository page, you will want to create a description that summarizes what this project will be (you can always change this later).

Also choose the “Add a README file” option because every project should have a README.

For the Add .gitignore option, it would be handy to choose the R gitignore template. [gitignore files are handy](https://git-scm.com/docs/gitignore#:~:text=A%20gitignore%20file%20specifies%20intentionally,gitignore%20file%20specifies%20a%20pattern.). They are a way to tell GitHub that you don’t want a particular file tracked. It’s just important for us to track the important files as it is for us to declutter our project by not adding unnecessary files.

You may want to choose a license. We recommend choosing something that allows others to freely use your code but with attribution like a Creative Commons license.

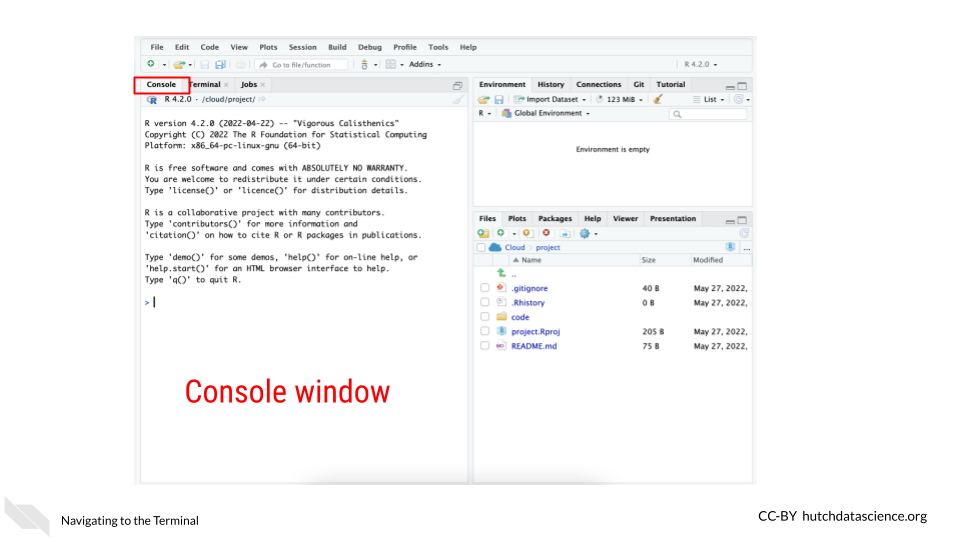
Then you are ready to click “Create repository”. Keep in mind what the location and name of this repository is. In GitHub repositories are named like this: username/repository\_name.

### 7.3.3 Setting up your repository locally

\*This section is adapted from the [DataTrail course](https://datatrail-jhu.github.io/DataTrail/cloning-a-repository.html#step-3-set-up-github-credentials).

In order to be able to access everything in your GitHub repository from RStudio cloud, you will need to set up GitHub credentials. You should only need to do this once per project.

1. In your RStudio interface, make sure that you are in the Console tab.



1. Now use the command below to install the package usethis. Copy and paste it in the Console window and click Enter on your keyboard. This package will help us manage our GitHub credentials from RStudio more easily.

install.packages("usethis")

This will take a minute or so to install. Remember that red text doesn’t mean an error necessarily.

1. Now to use this package, we need to attach its library using the following command:

library("usethis")

1. RStudio and GitHub require you make a special fancy password to use as credentials called a **GitHub Personal Access Token** (sometimes abbreviated as a “PAT”).

To create a ‘PAT’ from RStudio we can run this handy command:

usethis::create\_github\_token()

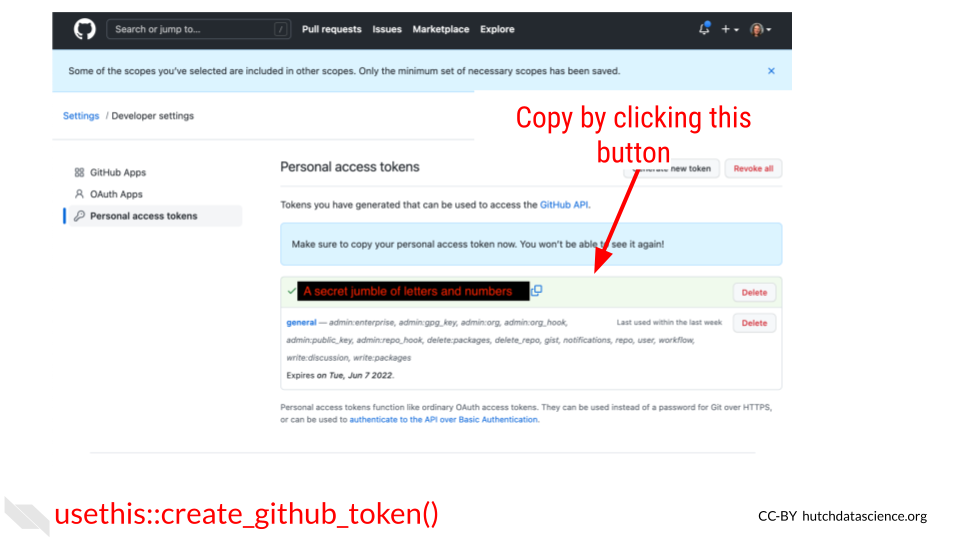
Running this command will open up a window in your GitHub that will ask you for your password. Login to GitHub as you normally would.

This will open up a page in GitHub for creating a New personal access token.

Underneath the Note put something that reminds you what this PAT is for. Something like RStudioCloud Access. (Note that each PAT you make needs its own unique Note though).

Underneath the Select scopes section **you don’t need to do anything**, usethis package already chose the permissions we need.

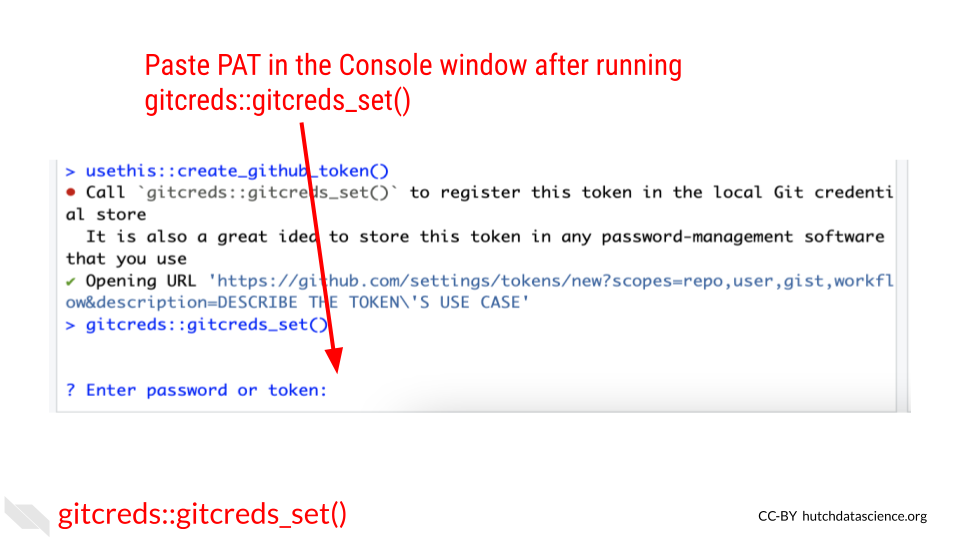
Scroll all the way down on this page and click Generate Token. You’ve created your first PAT! **Do not close this window**, keep it handy for now. Note that in the image below we blocked out our PAT, but yours will show a jumble of letters and numbers



1. Return back to your RStudio while keeping your PAT handy. In the Console window, run this command:

gitcreds::gitcreds\_set()

It will ask you to ? Enter password or token. Copy your PAT and paste it into the command window and press Enter.



After you enter your PAT here you should get a message like:

-> Adding new credentials...  
-> Removing credetials from cache...  
-> Done.

You are now free to close that GitHub PAT window. Note that you will want to be very careful with your PAT. **Do not share it or put it anywhere that others could see it or access it!**

1. Now we also need to add your username and email to the RStudio GitHub credentials by running a command like below. But replace the example **username** and **email** with what corresponds to your GitHub account.

use\_git\_config(user.name = "Jane", user.email = "jane@example.org")

Run this in the Console tab as well and click Enter.

1. Now to double check that everything is set, we can run this command to have the usethis package echo back our credentials:

git\_sitrep()

It will give you output that looks similar to this: (but note it will have your own user name, and repository name and etc.)

Git config (global)  
• Name: 'Jane'  
• Email: 'jane@example.org'  
• Global (user-level) gitignore file: <unset>  
• Vaccinated: FALSE  
ℹ See `?git\_vaccinate` to learn more  
• Default Git protocol: 'https'  
• Default initial branch name: <unset>  
GitHub  
• Default GitHub host: 'https://github.com'  
• Personal access token for 'https://github.com': '<discovered>'  
• GitHub user: 'Jane'  
• Token scopes: 'gist, repo, user, workflow'  
• Email(s): 'jane@example.org (primary)'  
✖ Local Git user's email ('jane@example.org') doesn't appear to be registered with GitHub.  
Git repo for current project  
• Active usethis project: '/cloud/project'  
• Default branch: 'master'  
• Current local branch -> remote tracking branch:  
 'master' -> 'origin/master'  
GitHub remote configuration  
• Type = 'theirs'  
• Host = 'https://github.com'  
• Config supports a pull request = FALSE  
• origin = 'JaneEverydayDoe/first\_project' (can not push)  
• upstream = <not configured>  
• Desc = The only configured GitHub remote is 'origin', which  
 you cannot push to.  
 If your goal is to make a pull request, you must fork-and-clone.  
 `usethis::create\_from\_github()` can do this.  
  
 Read more about the GitHub remote configurations that usethis supports at:  
 'https://happygitwithr.com/common-remote-setups.html'

You should see that Name, email have your credentials set as well as a Personal access token for 'https://github.com': '<discovered>'

You can run git\_sitrep() at anytime to see what your credentials and settings are.

Yay! Now you should be able to use GitHub from RStudio!

### 7.3.4 Cloning your repository

In git, to clone something is to get a copy of your project to your computer to work from.

* **clone** - making a copy of a code base on your computer.

In RStudio we can do this using the usethis package again. In GitHub repositories are named like this: username/repository\_name. Sometimes instead of a username it may be a github organization.

create\_from\_github("username/repository\_name")

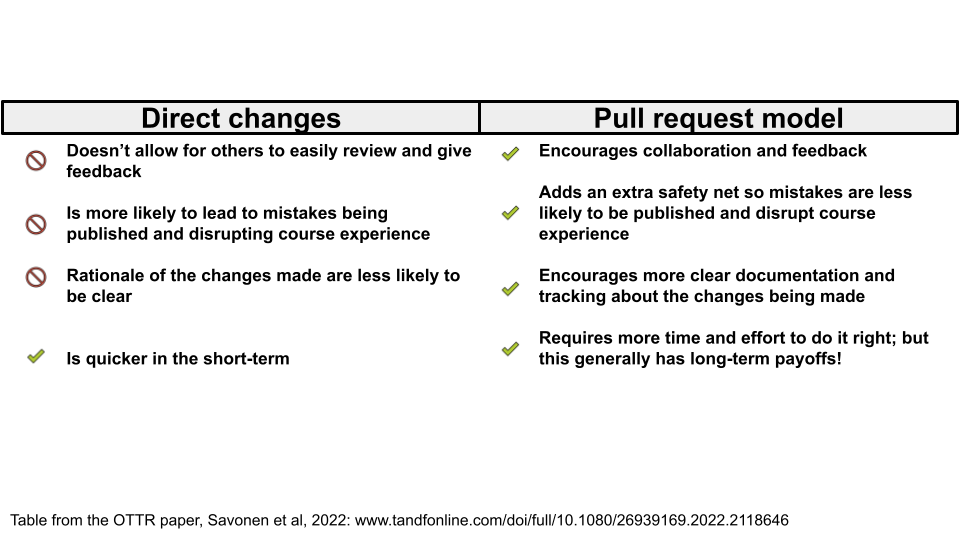
If this happens successfully, you should see this kind of message:

ℹ Defaulting to 'https' Git protocol  
✔ Setting `fork = FALSE`  
✔ Creating 'some-file-path-on-your-computer/repository\_name'  
✔ Cloning repo from 'https://github.com/username/repository\_name.git' into 'some-file-path-on-your-computer/repository\_name'  
✔ Setting active project to 'some-file-path-on-your-computer/repository\_name'  
ℹ Default branch is 'main'  
✔ Opening 'some-file-path-on-your-computer/repository\_name' in new RStudio session  
✔ Setting active project to '<no active project>'

This will also open up a new RStudio window. Now you are ready to get to work!

### 7.3.5 Opening a PR

A pull request (sometimes abbreviated PR) is a way of being able to review changes before you incorporate them into your main, more polished product. It is a highly effective system for doing code review and otherwise communicating about your data analysis to increase its reproducibility.



Pull requests are based on copies of the project repository that are called branches. When we do work on a project we will want to do it somewhere that is separate from our main set of code. Branches are yet another copy of the code that is used for developing purposes.

* **pull request** - a method of working on and incorporating file changes in a way that allow things to be reviewed and discussed on GitHub.
* **branch** - another copy and version of your project that you can work from and create a pull request.

#### 7.3.5.1 Creating a new branch

A main branch is where you will keep your best, most vetted version of the project. Your main branch will already exist on your project when you create it. When others come to look at your project, they will see the main branch first.

Other branches are generally made with the purpose of eventually having them reviewed and polished to the point that they are to add their changes to the main branch.

pr\_init(branch = "new\_branch\_name")

This branch will be called “new\_branch\_name” but you should call your branch whatever would be representative of the work you will be doing on this. For example, if you are adding documentation, you could call your branch “adding-docs”.

This brings us to another point. For keeping yourself organized as well as making it easier for others to follow, it’s best to keep a branch and subsequent pull request focused on one task. If one pull request tries to do too many things, it will be more difficult for it to be communicated and reviewed properly.

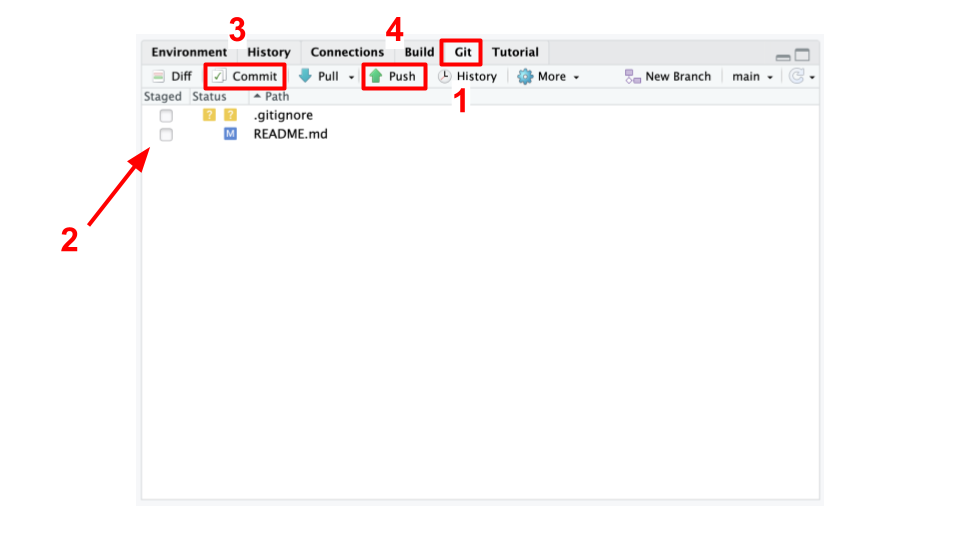
Now that you have created a branch, you are ready to make changes to your files.

Now make any change to a file for the purposes of this tutorial. For example, you could add a sentence to your README file to explain what this project will be. Then save the file change.

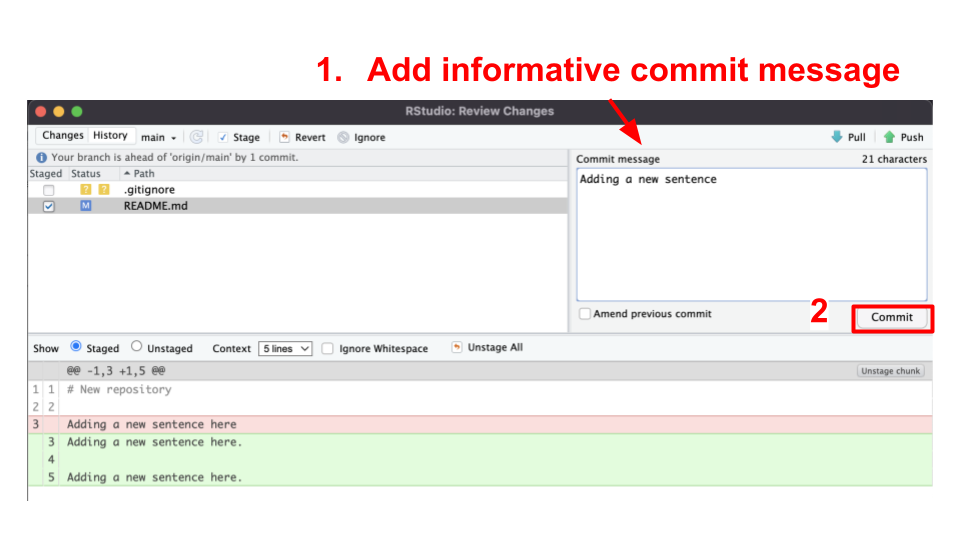
In RStudio, if we go to the Git tab we should see that the file we changed has an M next to it to signify it has been modified. In order to officially add these changes that we’ve made to our branch, we need to commit them.

* **commit** - the action of officially adding a file change to a branch.

To do this, we first need to check the box(es) next to the files we’d like to commit then we click the commit button.



After clicking the commit button, a new window will pop up that shows us the changes we are committing. In this window, we will need to add a commit message. Again, we should try to write something informative about what we were doing to these files we are committing. Then after we’ve written this commit message, we need to click Commit in this window.



Now our files are on our branch!

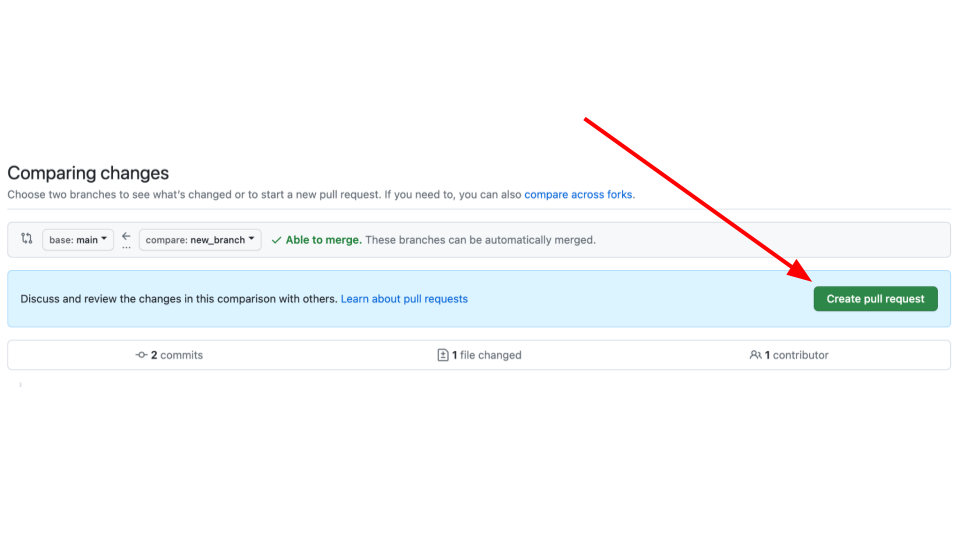
When we’d like these changes to be online on GitHub for others to see, we can push these changes. To push changes means to send them online to GitHub.

* **push** - the action of sending a branch and its file changes to GitHub so it is online where others can see it.

With the usethis package, we can use the pr\_push() function.

pr\_push()

This should open a new window on GitHub that will look like this:



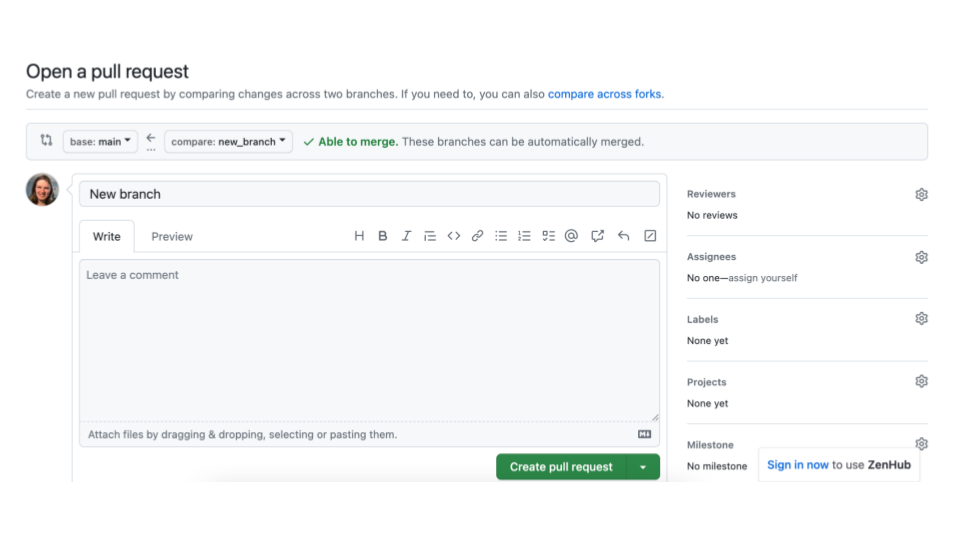
It will also print out some messages like this:

✔ Pushing local 'branch\_name' branch to 'origin' remote.  
• Create PR at link given below  
✔ Opening URL 'https://github.com/username/repository\_name/compare/branch\_name'

On this window, click create pull request. This will bring you to another page on GitHub where you can create your pull request. On this page you will want to describe the changes you are making with this pull request.

You should include information like:

1. The background behind the changes you are making. What is the problem you are solving? Link to any relevant conversations.
2. What changes are you making specifically and how do these address the problem?
3. What work is left to be done?
4. What help could you use from others? Is there something in particular you would like to be looked at? Is there something not yet added that should be added?



Now if you continue to work with your files, you can go through the same steps of:

1. Making a file change
2. Check the box(es) next to the file changes you’d like to add
3. Commit the file
4. Add a commit message
5. Commit the changes
6. Push the changes by running pr\_push() (You can also click the green arrow to push if you prefer).

You can repeat these steps as many times as you need until you feel the file changes on your PR page are ready for someone else to review. If you don’t have someone else on your team to review the changes, you can alternatively let them sit for a day or two and let “future You” review your file changes. Future you or others on your project might have a fresh take on these files. We will talk more about code review in a later chapter.

Code review is perhaps the most powerful tool for making reproducible analyses. And now that you know how to make pull requests on GitHub, you have an excellent platform and system for version controlling and tracking, and reviewing your files! Congrats!

### 7.3.6 Merging a pull request

In the Files changed tab, you or others can leave comments about your file changes. This is an excellent way to document the rationale of these file changes as well as discuss any alternatives.

After some back and forth discussion (whether this discussion be between you and yourself or someone else) you may decide that the code on this pull request is ready for primetime! In other words, it is ready to be incorporated into the main branch.

Recall that we said the main branch is the most polished and readily viewable version of your project. To bring the changes in your pull request into the main branch, we will need to perform an action called a merge.

This step of incorporating changes into a branch and combining two branches together is called merging. The goal of a merge is to combine two branches in such a way that keeps your desired changes from both copies.

* **main** - the branch name that is typically used for the main, most polished and live version of your project that others will be brought to first.
* **merge** - combining the files from two branches into one.

On your pull request page, scroll to the bottom to the big green button where it says Merge pull request. Be careful to not press this button before you are sure that everyone on your team is ready.

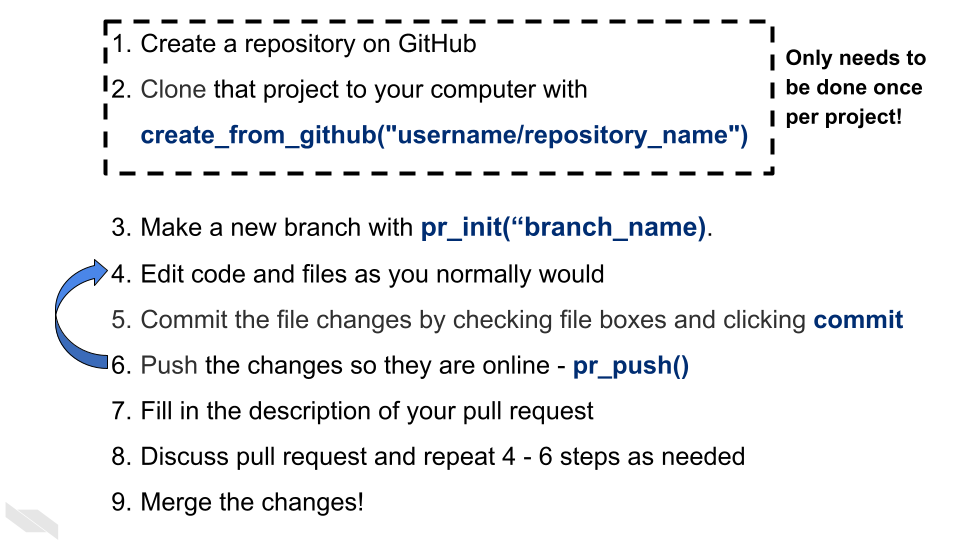
After you’ve clicked it, hooray! You’ve completed your first merge of a pull request.

It may feel like we’ve described a lot of steps, but getting into the GitHub workflow and utilizing its benefits is all about habits! We promise it will increase the reproducibility of your analyses if you are able to stick with this process!

## 7.4 Conclusion

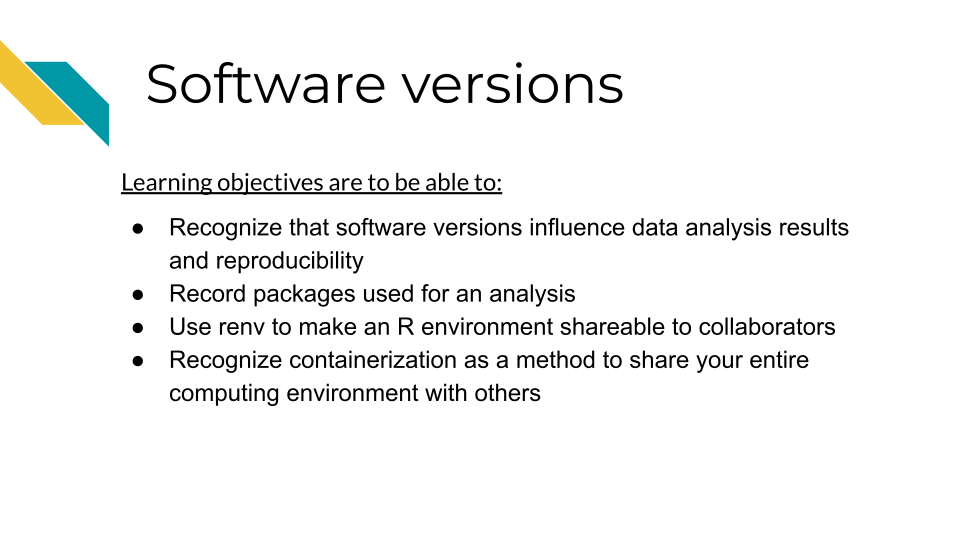
We also discussed in this chapter how version control and GitHub are great tools for reproducibility and we walked through the GitHub workflow process as completed in RStudio.

To summarize, the Github workflow process looks like this:



# 8 Software versions

## 8.1 Learning Objectives



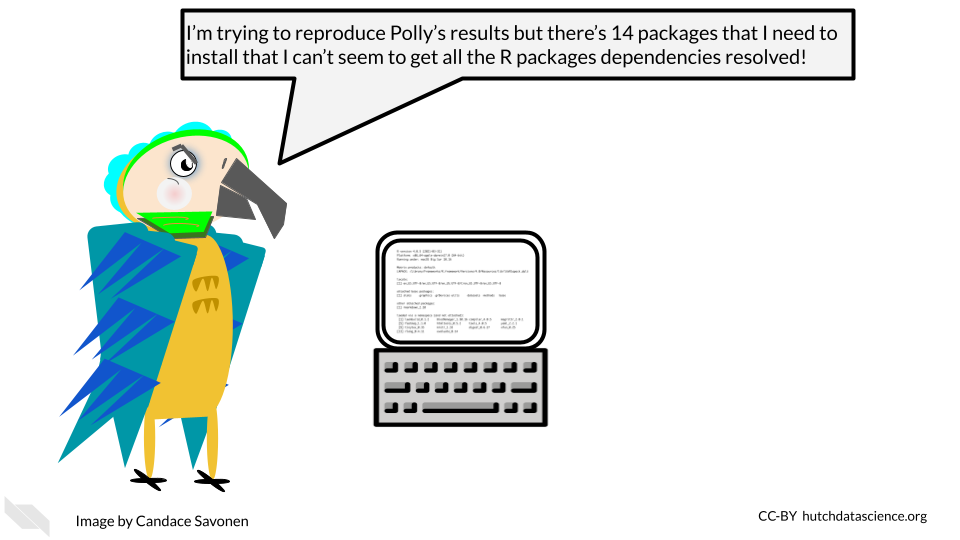
As we discussed, reproducibility is on a continuum, meaning that it can range from being impossible to very easy to reproduce any given results. Some results can be effectively impossible to reproduce if there are too many barriers and set up needed to re-run the analysis. One of the most common barriers is the computing environment used run the analysis.

**computing environment** - All the relevant pieces of software and their dependencies that were used on a computer at the time that an analysis or other project was run

## 8.2 No two computers are the same

A computing environment not only consists of the direct software that we use to analyze data, but all of the other software that our main pieces of software require to install and run properly.

As we use our computers daily for work, we are constantly installing, updating, and removing software packages. Sometimes our computers do this automatically without us knowing. These software packages interact with and depend on each other, meaning it can be quite frustrating to try update even a single piece of software if it exists in a tangled mess of software dependencies. Computer scientists sometimes call this “[dependency hell](https://en.wikipedia.org/wiki/Dependency_hell)”.



As developers and maintainers of software continue to make updates and fixes to the software, the developers and maintainers of other interdependent software are doing similarly, meaning that software dependencies and the computing environments are not only a complicated mess at times, but also a moving target!

## 8.3 Software and package versions affect results!

Sometimes if we have generally the same software installed for reproducing an analysis, we may feel that that is “close enough”. And given all the other technical aspects of reproducibility, it can be easy to overlook what versions of software packages we are using. However, controlling for software versions is critical for creating reproducible analyses. Software versions can directly affect not only whether an analysis will be able to run, but the results of the analysis ([Beaulieu-Jones and Greene 2017](#ref-BeaulieuJones2017)).

## 8.4 Session Info

Perhaps the easiest way to begin to address computing environment variability is to record what the computing environment looks like at the time an analysis is run. In R, this is a fairly straightforward task.

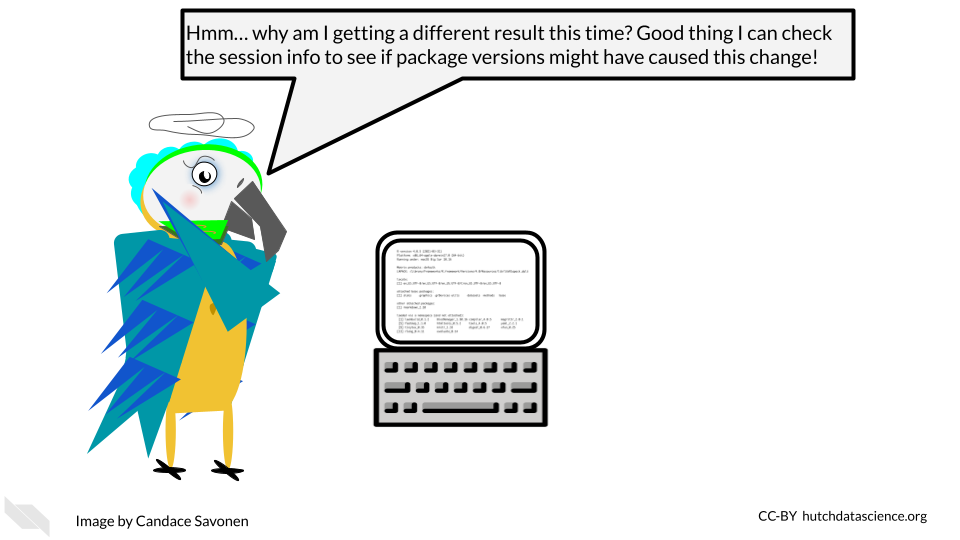
Generally at the end of your R notebook, you will want to print out your session info. You can do this by running the function sessionInfo() or the tidyverse version of this function from the devtools package, devtools::session\_info().

We can run sessionInfo in this book (this book was created using R tools).

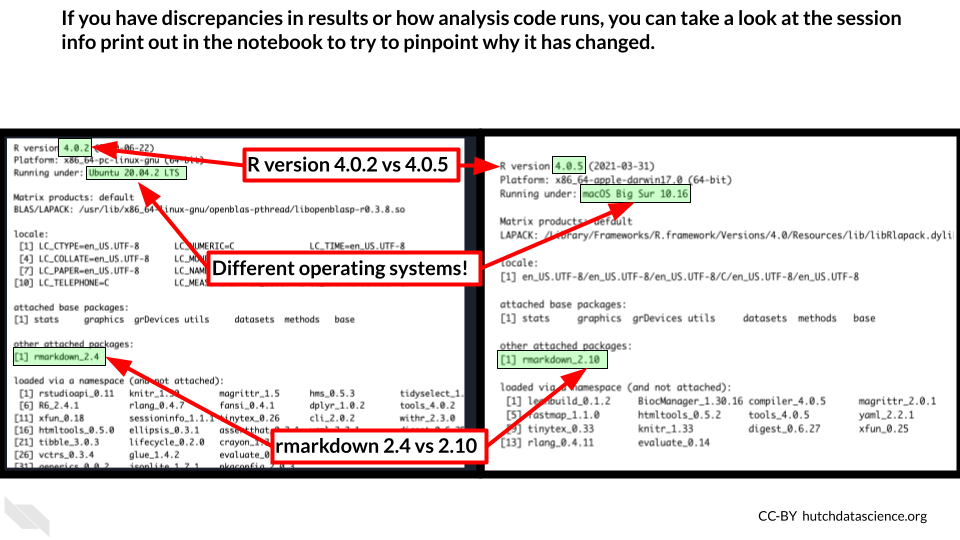
sessionInfo()

## R version 4.0.2 (2020-06-22)  
## Platform: x86\_64-pc-linux-gnu (64-bit)  
## Running under: Ubuntu 20.04.5 LTS  
##   
## Matrix products: default  
## BLAS: /usr/lib/x86\_64-linux-gnu/openblas-pthread/libblas.so.3  
## LAPACK: /usr/lib/x86\_64-linux-gnu/openblas-pthread/liblapack.so.3  
##   
## locale:  
## [1] LC\_CTYPE=en\_US.UTF-8 LC\_NUMERIC=C   
## [3] LC\_TIME=en\_US.UTF-8 LC\_COLLATE=en\_US.UTF-8   
## [5] LC\_MONETARY=en\_US.UTF-8 LC\_MESSAGES=en\_US.UTF-8   
## [7] LC\_PAPER=en\_US.UTF-8 LC\_NAME=C   
## [9] LC\_ADDRESS=C LC\_TELEPHONE=C   
## [11] LC\_MEASUREMENT=en\_US.UTF-8 LC\_IDENTIFICATION=C   
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## loaded via a namespace (and not attached):  
## [1] knitr\_1.33 magrittr\_2.0.3 hms\_0.5.3 R6\_2.4.1   
## [5] rlang\_1.1.0 fastmap\_1.1.1 fansi\_0.4.1 stringr\_1.4.0   
## [9] httr\_1.4.2 tools\_4.0.2 xfun\_0.26 utf8\_1.1.4   
## [13] cli\_3.6.1 htmltools\_0.5.5 ottrpal\_1.0.1 yaml\_2.2.1   
## [17] digest\_0.6.25 tibble\_3.2.1 lifecycle\_1.0.3 bookdown\_0.24   
## [21] readr\_1.4.0 vctrs\_0.6.1 fs\_1.5.0 curl\_4.3   
## [25] glue\_1.4.2 evaluate\_0.20 rmarkdown\_2.10 stringi\_1.5.3   
## [29] compiler\_4.0.2 pillar\_1.9.0 pkgconfig\_2.0.3

Now we have recorded what some key aspects of our computing environment looked like at the time that this book was rendered last. This print out may seem like a lot of nonsense at first, but it gives us some useful information in a pinch!



If we take a look at two different session info printouts, we can begin to spot the differences. These differences may give us clues into why an analysis ran differently.



Printing out session info is an easy way to record your computing environment in hopes of increasing the reproducibility of your analysis!

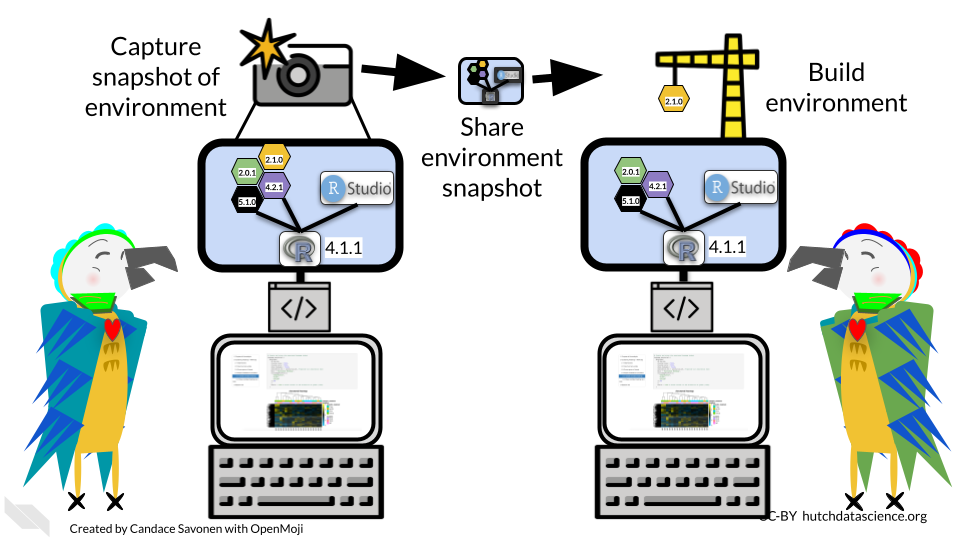
**session info** - A printout in R that displays information about the software and packages that were being used at the time the sessionInfo() or devtools::session\_info() functions were run.

## 8.5 Snapshots with renv

However, you may realize that while session info is useful for recording this information, it doesn’t mitigate the frustration of setting up a computing environment in R. Nor does it help us with being able to directly share our computing environments.

It can be incredibly handy for reproducibility purposes to be able to share the R computing environment you used for completing an analysis. This is not only helpful for others who may be interested in reproducing your analysis, but also for future you! If you come back to this analysis and attempt to re-run it, it is likely you’ve changed your R computing environment over time by installing or removing packages. renv will allow you to return to the environment you used at the time that you ran the analysis.

For that, we need a slightly more involved solution of using [renv](https://rstudio.github.io/renv/articles/renv.html). renv is an R package that allows you to take ‘snapshots’ of your R computing environment and use those to track, share, and build R environments.

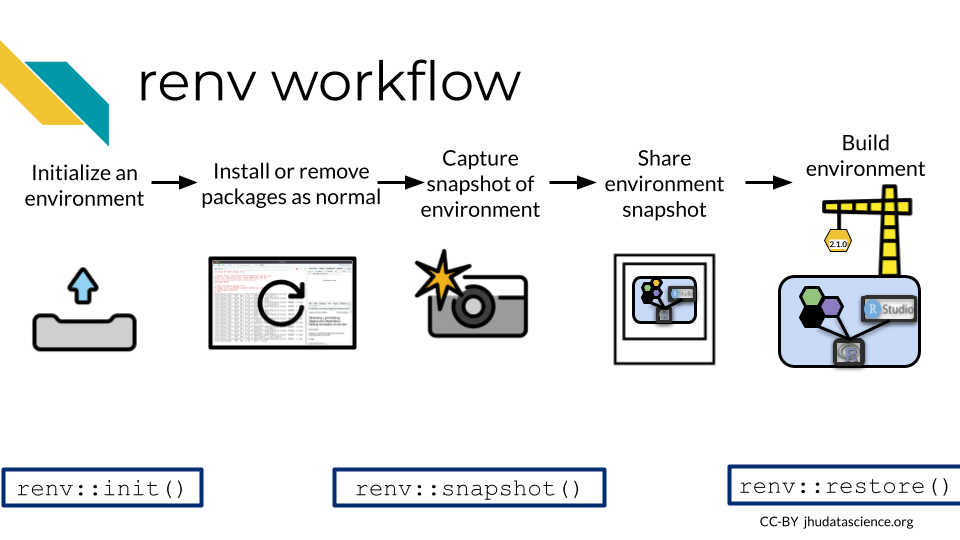


The renv workflow looks like this (as described by their documentation):

1. Call renv::init() to initialize a new project-local environment with a private R library
2. Work in the project as normal, installing and removing new R packages as they are needed in the project
3. Call renv::snapshot() to save the state of the project library to the lockfile (called renv.lock)
4. Continue working on your project, installing and updating R packages as needed
5. Call renv::snapshot() again to save the state of your project library if your attempts to update R packages were successful, or call renv::restore() to revert to the previous state as encoded in the lockfile if your attempts to update packages introduced some new problems

To make this shareable to others, you will need to do two things:

1. Be sure to commit and push the renv.lock file to your GitHub repository for your project.
2. Be sure to describe that your project uses renv in the README of this project (commit and push this to your GitHub repository also).



The limitations of this method, [as noted by the renv authors](https://rstudio.github.io/renv/articles/renv.html#caveats), is that it really only tracks packages in R and cannot help track or enforce items that may affect the computing environment outside of R. So while it will aid in the reproducibility of your analysis, it will not cover everything.

**renv** - An R package that helps you to share and record your R specific computing environment

## 8.6 Containerization

In order to truly reproduce a result with an identical computing environment you would need to use a containerized approach. To containerize a computing environment is to truly create an environment that is shippable to others. A container is analogous to a virtual machine. A computer runs a computing environment inside of it that is separate from the rest of the computer (hence why its called a container).

One of the most popular containerization softwares is Docker. Docker allows you to build your computing environment and share it on its online platform in the form of images that you can download and run. In fact, this book is rendered by a Docker container!

If you will be using a container with PHI or PII or other protected information, we recommend you take a look at [this resource](https://www.cleardata.com/wp-content/uploads/2017/07/SET-MKTG-WP-36_HIPAA_Compliant_Containers_05-04-2017.pdf) to understand best practices for using Docker with sensitive data.

* **container** - A method for running software in a way that is shareable and Reproducible
* **Docker** - A popular platform for containers

We will not cover Docker here but if you are interested in using a containerized approach like Docker, here are additional resources for learning:

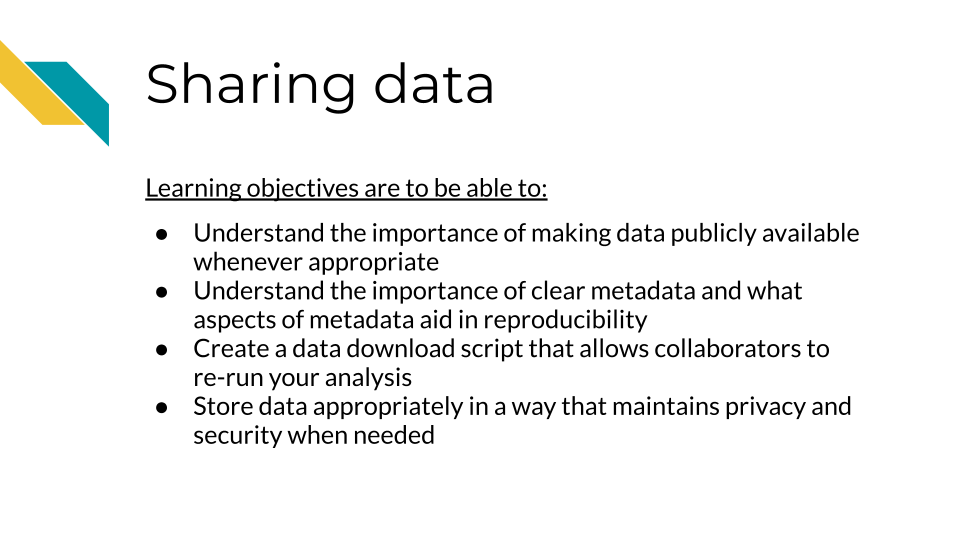
* [Software Carpentries course on Docker](https://carpentries-incubator.github.io/docker-introduction/introduction/index.html)
* [ITCR Training Network chapters about Docker](https://jhudatascience.org/Adv_Reproducibility_in_Cancer_Informatics/launching-a-docker-image.html)
* [Docker documentation about getting started](https://www.docker.com/get-started/)
* [How to ensure your Docker usage is HIPAA-Compliant](https://www.atlantic.net/hipaa-compliant-hosting/best-practices-for-creating-a-hipaa-compliant-docker-host/)
* [HIPAA Compliant Containers](https://www.cleardata.com/wp-content/uploads/2017/07/SET-MKTG-WP-36_HIPAA_Compliant_Containers_05-04-2017.pdf)
* [Singularity is a different container platform that does some encryption](https://docs.sylabs.io/guides/latest/user-guide/) – this can help if you are using data that needs to be protected.

## 8.7 Conclusion

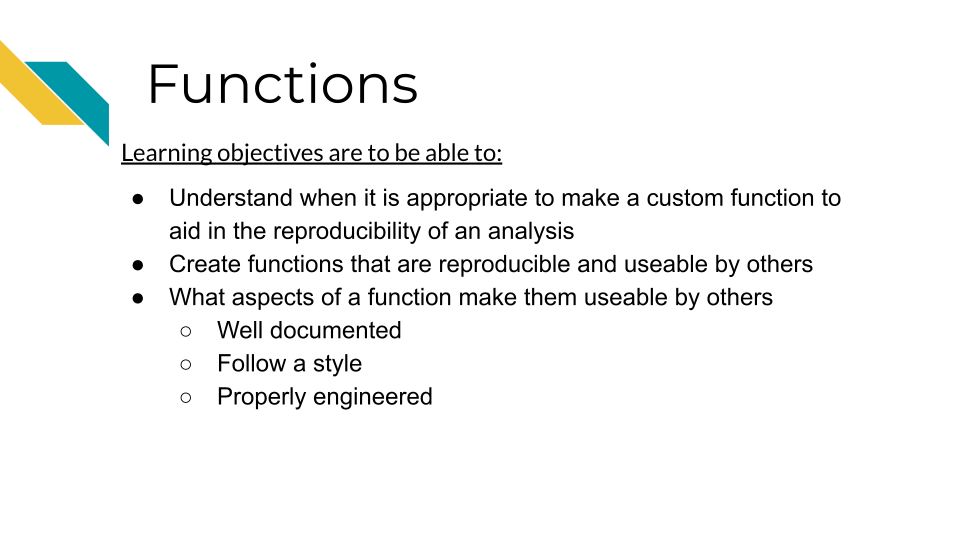
In summary:

* Software versions affect the reproducibility of an analysis.
* Printing out session info is a great way to record software versions.
* renv is an R package that allows you to share your R specific computing environment.
* Containerization softwares like Docker allow you to more completely share a replicate computing environment.

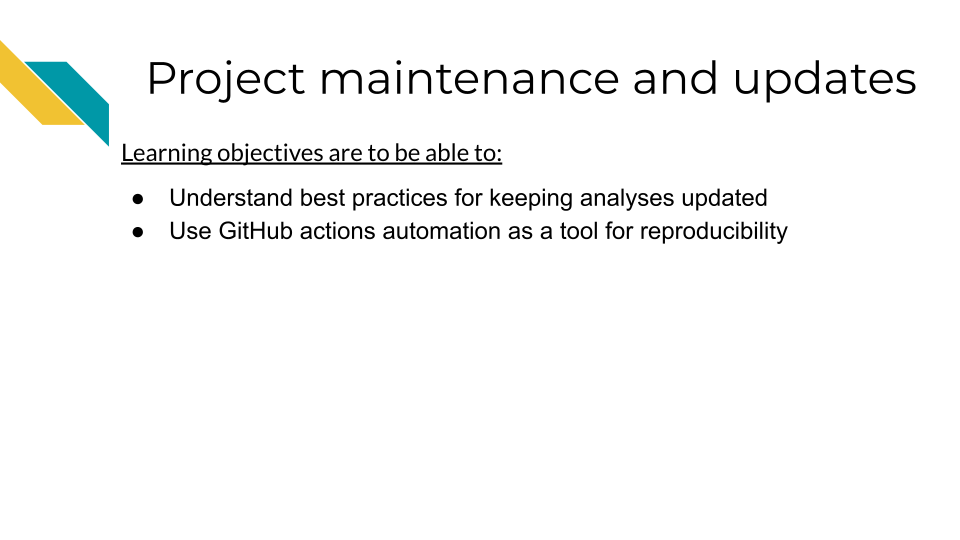
# 9 Sharing Data



# 10 Functions



# 11 Project maintenance and updates



# 12 Collaborations through GitHub



# About the Authors

These credits are based on our [course contributors table guidelines](https://www.ottrproject.org/more_features.html#giving-credits-to-contributors).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | [FirstName LastName](link%20to%20personal%20website) |
| Lecturer(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | Delivered the course in some way - video or audio |
| Content Author(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | If any other authors besides lead instructor |
| Content Contributor(s) (include section name/link in parentheses) - make new line if more than one section involved | Wrote less than a chapter |
| Content Editor(s)/Reviewer(s) | Checked your content |
| Content Director(s) | Helped guide the content direction |
| Content Consultants (include chapter name/link in parentheses or word “General”) - make new line if more than one chapter involved | Gave high level advice on content |
| Acknowledgments | Gave small assistance to content but not to the level of consulting |
| **Production** |  |
| Content Publisher(s) | Helped with publishing platform |
| Content Publishing Reviewer(s) | Reviewed overall content and aesthetics on publishing platform |
| **Technical** |  |
| Course Publishing Engineer(s) | Helped with the code for the technical aspects related to the specific course generation |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottrpal](https://github.com/jhudsl/ottrpal)) [Candace Savonen](https://www.cansavvy.com/), [John Muschelli](https://johnmuschelli.com/), [Carrie Wright](https://carriewright11.github.io/) |  |
| **Art and Design** |  |
| Illustrator(s) | Created graphics for the course |
| Figure Artist(s) | Created figures/plots for course |
| Videographer(s) | Filmed videos |
| Videography Editor(s) | Edited film |
| Audiographer(s) | Recorded audio |
| Audiography Editor(s) | Edited audio recordings |
| **Funding** |  |
| Funder(s) | Institution/individual who funded course including grant number |
| Funding Staff | Staff members who help with funding |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.5 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2023-04-17   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.5)   
## bookdown 0.24 2023-03-28 [1] Github (rstudio/bookdown@88bc4ea)   
## cachem 1.0.7 2023-02-24 [1] CRAN (R 4.0.2)   
## callr 3.5.0 2020-10-08 [1] RSPM (R 4.0.2)   
## cli 3.6.1 2023-03-23 [1] CRAN (R 4.0.2)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.20 2023-01-17 [1] CRAN (R 4.0.2)   
## fastmap 1.1.1 2023-02-24 [1] CRAN (R 4.0.2)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.4.2 2020-08-27 [1] RSPM (R 4.0.5)   
## htmltools 0.5.5 2023-03-23 [1] CRAN (R 4.0.2)   
## knitr 1.33 2023-03-28 [1] Github (yihui/knitr@a1052d1)   
## magrittr 2.0.3 2022-03-30 [1] CRAN (R 4.0.2)   
## memoise 2.0.1 2021-11-26 [1] CRAN (R 4.0.2)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.4.0 2020-10-07 [1] RSPM (R 4.0.2)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## rlang 1.1.0 2023-03-14 [1] CRAN (R 4.0.2)   
## rmarkdown 2.10 2023-03-28 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.3 2022-04-02 [1] CRAN (R 4.0.2)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2023-03-28 [1] Github (R-lib/testthat@e99155a)   
## usethis 1.6.3 2020-09-17 [1] RSPM (R 4.0.2)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2023-03-28 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library

# 13 References

“About Scientific Code Review.” n.d. Accessed January 26, 2023. <https://hutchdatascience.org/code_review/>.

Beaulieu-Jones, Brett K, and Casey S Greene. 2017. “Reproducibility of Computational Workflows Is Automated Using Continuous Analysis.” *Nature Biotechnology* 35 (4): 342–46. <https://doi.org/10.1038/nbt.3780>.

Bodner, Herbert. 2018. “10 Reasons Why Code Reviews Make Better Code and Better Teams.” *Simple Programmer*. <https://simpleprogrammer.com/why-code-reviews-make-better-code-teams/>.

Gillespie, Colin, and Robin Lovelace. 2021. *Efficient R Programming*. O’Reilly Media Inc. <https://csgillespie.github.io/efficientR/>.

Parker, Hilary. 2017. “Opinionated Analysis Development.” Preprint. PeerJ Preprints. <https://doi.org/10.7287/peerj.preprints.3210v1>.

“R (Programming Language).” 2023. *Wikipedia*. <https://en.wikipedia.org/w/index.php?title=R_(programming_language)&oldid=1144363470>.

“R: The R Project for Statistical Computing.” <https://www.r-project.org/>.

Radigan, Dan. n.d. “What Is a Code Review & How It Can Save Time.” Atlassian. Accessed January 26, 2023. <https://www.atlassian.com/agile/software-development/code-reviews>.

Riederer, Emily. 2020. “Column Names as Contracts.” Emily Riederer. September 2020. <https://emilyriederer.netlify.app/post/column-name-contracts/>.

Timbers, Tiffany, Trevor Campbell, and Melissa Lee. 2022. *Data Science*. <https://datasciencebook.ca>.