WEEK

#01

WHAT IS DATA SCIENCE

THE DATA SCIENTIST'S TOOLBOX

What is data science?

Data science can involve:

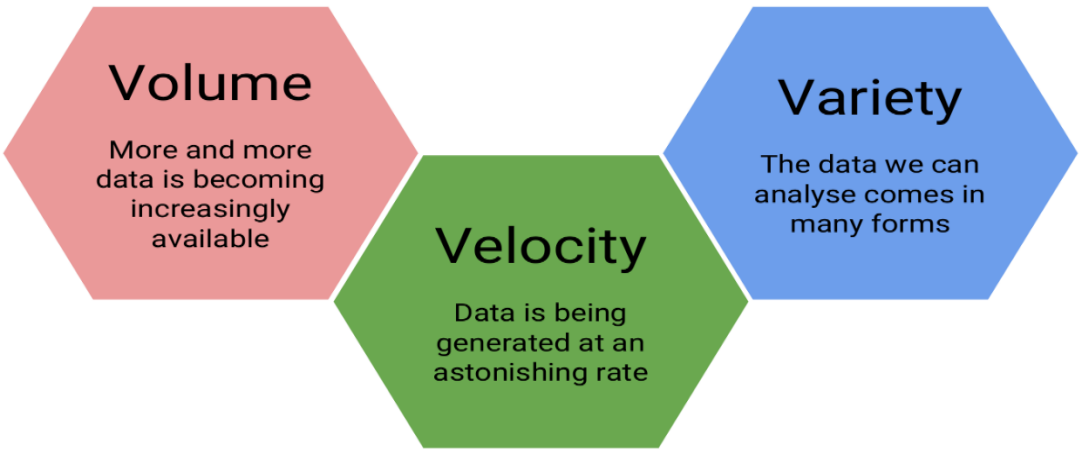
- Statistics, computer science, mathematics
- Data cleaning and formatting
- Data visualization
-

Why do we need data science?

- vast amount of data currently available and being generated.
- we simultaneously have the rise of inexpensive computing.
- In the third century BC, the Library of Alexandria was believed to house the sum of human knowledge. Today, there is enough information in the world to give every person alive 320 times as much of it as historians think was stored in Alexandria's entire collection.

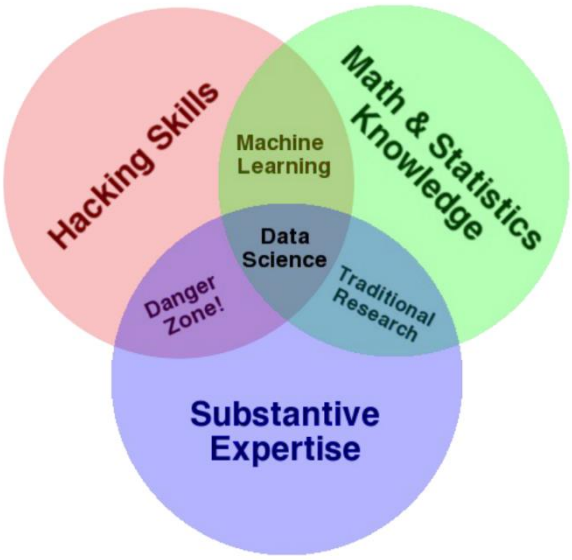
What is big data?

- The first is **volume**
- The second quality of big data is **velocity**.
- The third quality of big data is **variety**.



What is a data scientist?

- Data scientist is somebody who uses data to answer questions.



Why do Data Science

- Demand
- according to glassdoor, in which they ranked the top 50 best jobs in America

=X=X=X=X=

DATA

What is Data?

- Information, especially facts or numbers, collected to be examined and considered and used to help decision-making.
- A set of values of qualitative or quantitative variables.

"A set of values of qualitative or quantitative variables"

Set: In statistics, the population you are trying to discover something about

Variable: Measurements or characteristics of an item

Qualitative variable: Measurements or information about qualities

Quantitative variable: Measurements or information about quantities or numerical items

Messy Data

- have to work to extract the information you need to answer your question.
- Sequencing data
- Population census data
- Electronic medical records (EMR), other large databases
- Geographic information system (GIS) data (mapping)
- Image analysis and image extrapolation
- Messy data: Sequencing

  One type of data, that I work with regularly, is sequencing data. This data is generally first encountered in the FASTQ format, the raw file format produced by sequencing machines. These files are often hundreds of millions of lines long, and it is our job to parse this into an understandable and interpretable format and infer something about that individual's genome. In this case, this data was interpreted into expression data, and produced a plot called a "volcano plot".

- Messy data: Census information

  One rich source of information is country wide censuses. In these, almost all members of a country answer a set of standardized questions and submit these answers to the government. When you have that many respondants, the data is large and messy; but once this large database is ready to be queried, the answers embedded are important.Here we have a very basic result of the last US census - in which all respondants are divided by sex and age, and this distribution is plotted in this population pyramid plot.

- Messy data: Electronic medical records (EMR)

Electronic medical records are increasingly prevalent as a way to store health information, and more and more population based studies are using this data to answer questions and make inferences about populations at large, or as a method to identify ways to improve medical care. For example, if you are asking about a population's common allergies, you will have to extract many individuals' allergy information, and put that into an easily interpretable table format where you will then perform your analysis.

- Messy data: Image analysis/extrapolation

  A more complex data source to analyse are images/videos. There is a wealth of information coded in an image or video, and it is just waiting to be extracted. An example of image analysis that you may be familiar with is when you upload a picture to Facebook and not only does it automatically recognize faces in the picture, but then suggests who they may be. A fun example you can play with is the DeepDream software that was originally designed to detect faces in an image, but has since moved on to more *artistic* pursuits.

=x=x=x=x=

GETTING HELP

Why we need to get help?

- One of the main skills you are going to be called upon for as a data scientist is your ability to solve problems. And sometimes to do that, you need help. The ability to solve problems is at the root of data science; so the importance of being able to do so is paramount.

Why is knowing how to get help important?

- troubleshooting and figuring out solutions to problems is a great, transferable skill! It will serve you well as a data scientist, but so much of what any job often entails is problem solving. Being able to think about problems and get help effectively is of benefit to you in whatever career path you find yourself in!

Before You ask for Help

- One of your first stops for data analysis problems should be reading the manuals or help files (for R problems, try typing ?command) – if you post a question on a forum that is easily answered by the manual, you will often get a reply of "Read the manual" which is not the easiest way to get at the answer you were going for!
- Next steps are searching on Google and searching relevant forums. Common forums for data science problems include StackOverflow and CrossValidated. Additionally, for you in this class, there is a course forum that is a great resource and super helpful! Before posting a question to any forum, try and double check that it hasn't been asked before, using the forums' search functions.

- While you are Googling, things to pay attention to and look for are: tutorials, FAQs, or vignettes of whatever command or program is giving you trouble. These are great resources to get you started – either in telling you the language/words to use in your next searches, or outright showing you how to do something.

First steps for solving coding problems

As you get further into this course and using R, you may run into coding problems and errors and there are a few strategies you should have ready to deal with these. In my experience, coding

problems generally fall into two categories: your command produces no data and spits out an error message OR your command produces an output, but it is not at all what you wanted. These two problems have different strategies for dealing with them.

If it's a problem producing an error message:

- Check for typos!
- **Read the error message and make sure you understand it**
- Google the error message, exactly

Next steps

Alright, you've done everything you are supposed to do to solve the problem on your own – you need to bring in the big guns now: other people!

Easiest is to find a peer with some experience with what you are working on and ask them for help/direction. This is often great because the person explaining gets to solidify their understanding while teaching it to you, and you get a hands on experience seeing how they would solve the problem. In this class, your peers can be your classmates and you can interact with them through the course forum (double check your question hasn't been asked already!).

But, outside of this course, you may not have too many data science savvy peers – what then?

"Rubber duck debugging" is a long held tradition of solitary programmers everywhere. In the book "The Pragmatic Programmer," there is a story of how stumped programmers would explain their problem to a rubber duck, and in the process of explaining the problem, identify the solution.

When all else fails - posting to forums

You've done your best. You've searched and searched. You've talked with peers. You've done everything possible to figure it out on your own. And you are still stuck. It's time. Time to post your question to a relevant forum.

Before you go ahead and just post your question, you need to consider how you can best ask your question to garner (helpful) answers.

How to effectively ask questions on forums

Details to include:

- The question you are trying to answer
- How you approached the problem, what steps you have already taken to answer the question
- What steps will reproduce the problem (including sample data for troubleshooters to work from!)
- What was the expected output

- What you saw instead (including any error messages you received!)
- What troubleshooting steps you have already tried
- Details about your set-up, eg: what operating system you are using, what version of the product you have installed (eg: R, Rpackages)
- Be specific in the title of your questions!

Forum etiquette

Following a lot of the tips above will serve you well in posting on forums and observing forum etiquette. You are asking for help, you are hoping somebody else will take time out of their day to help you – you need to be courteous. Often this takes the form of asking specific questions, doing some troubleshooting of your own, and giving potential problem solvers easy access to all the information they need to help you. Formalizing some of these do's and don't's, you get the following lists:

Do's

- Read the forum posting guidelines
- Make sure you are asking your question on an appropriate forum!
- Describe the goal
- Be explicit and detailed in your explanation
- Provide the minimum information required to describe (and replicate) the problem
- Be courteous! (Please and thank you!)
- Follow up on the post OR **post the solution**

=X=X=X=X=

The Data Science Process

Our goal in this lesson is to expose you to the process one goes through as they carry out data science projects.

The Parts of a Data Science Project

- starts with a question that is to be answered with data.
- The second step is **finding or generating the data** you're going to use to answer that question.
- With the question solidified and data in hand, the **data are then analyzed**, first by **exploring the data** and then often by **modeling the data**, which means using some statistical or machine learning techniques to analyze the data and answer your question.
- After drawing conclusions from this analysis, the project has to be **communicated to others**.

=X=X=X=X=

WEEK

#02

R AND RStudio

Installing R

Now that we've got a handle on what a data scientist is, how to find answers, and then spent some time going over a data science example, it's time to get you set up to start exploring on your own. And the first step of that is installing R.

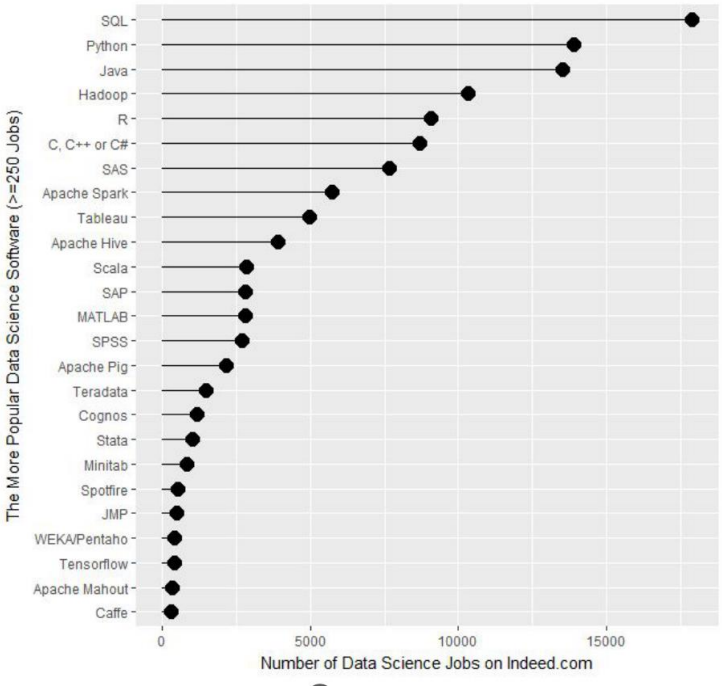
What is R? What is CRAN?

R is both a programming language and an environment, focused mainly on statistical analysis and graphics. It will be one of the main tools you use in this and following courses.

R is downloaded from the Comprehensive R Archive Network, or CRAN, and while this might be your first brush with it, we will be returning to CRAN time and time again, when we install packages - so keep an eye out!

Why should you use R?

Its popularity



Its cost

FREE!

This one is pretty self-explanatory - every aspect of R is free to use, unlike some other stats packages you may have heard of (eg: SAS, SPSS), so there is no cost barrier to using R!

Its extensive functionality

R is a very versatile language - we've talked about its use in stats and in graphing, but its use can be expanded to many different functions - from making websites, making maps using GIS data, analysing language and even making these lectures and videos! For whatever task you have in mind, there is often a package available for download that does exactly that!

Installation - for Windows

If you are on a Windows computer, follow the link Download R for Windows, and follow the directions there - if this is your first time installing R, go to the base distribution and click on the link at the top of the page that should say something like "Download R [version number] for Windows." This will download an executable file for installation.

=X=X=X=X=

Installing RStudio

We've installed R and can open the R interface to input code, but there are other ways to interface with R - and one of those ways is using RStudio. In this lesson, we'll get RStudio installed on your computer.

What is RStudio?

RStudio is a graphical user interface for R, that allows you to write, edit and store code, generate, view and store plots, manage files, objects and dataframes, and integrate with version control systems – to name a few of its functions. We will be exploring exactly what RStudio can do for you in future lessons, but for anybody just starting out with R coding, the visual nature of this program as an interface for R is a huge benefit.

Installing RStudio

Thankfully, installation of RStudio is fairly straightforward. First, you go to the RStudio download page. We want to download the RStudio Desktop version of the software, so click on the appropriate "Download" , under that heading and you will see a list of "Installers for supported platforms".

Installing RStudio - Windows

For Windows, select the RStudio installer for the various Windows editions (Vista, 7, 8, 10). This will initiate the download process. When the download is complete, open this executable file to access the installation wizard. You may be presented with a security warning at this time - allow it to make changes to your computer.

Following this, the installation wizard will open. Following the defaults on each of the windows of the wizard is appropriate for installation. In brief, on the welcome screen, click next. If you want RStudio installed elsewhere, "Browse" through your file system. Otherwise, it will likely default to the "Program Files" folder - this is appropriate. Click next. On this final page, allow RStudio to create a Start menu shortcut. Click Install. RStudio is now being installed. Wait for this process to finish; RStudio is now installed on your computer. Click Finish.

Check that RStudio is working appropriately by opening it from your Start menu.

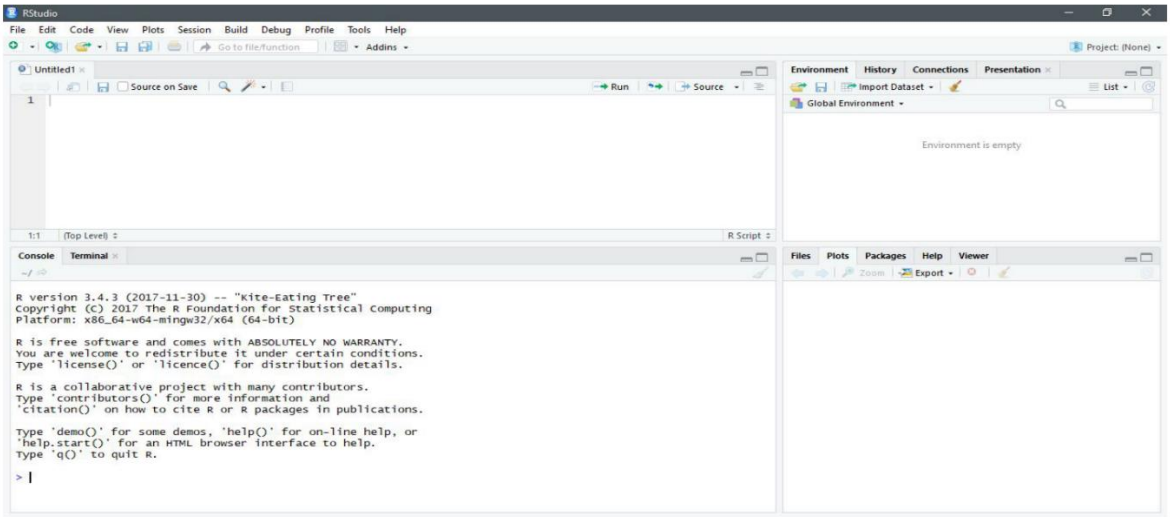
=X=X=X=X=

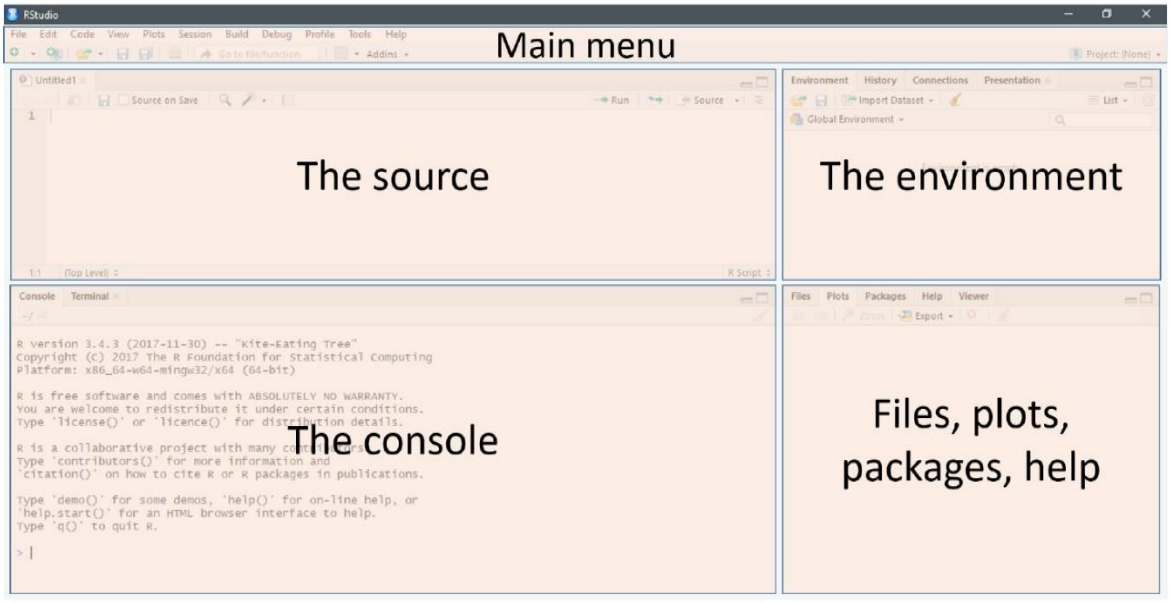
RStudio Tour

Now that we have RStudio installed, we should familiarize ourselves with the various components and functionality of it! RStudio provides a cheatsheet of the RStudio environment - warning: this link initiates a download of a PDF from the RStudio GitHub.

The various quadrants

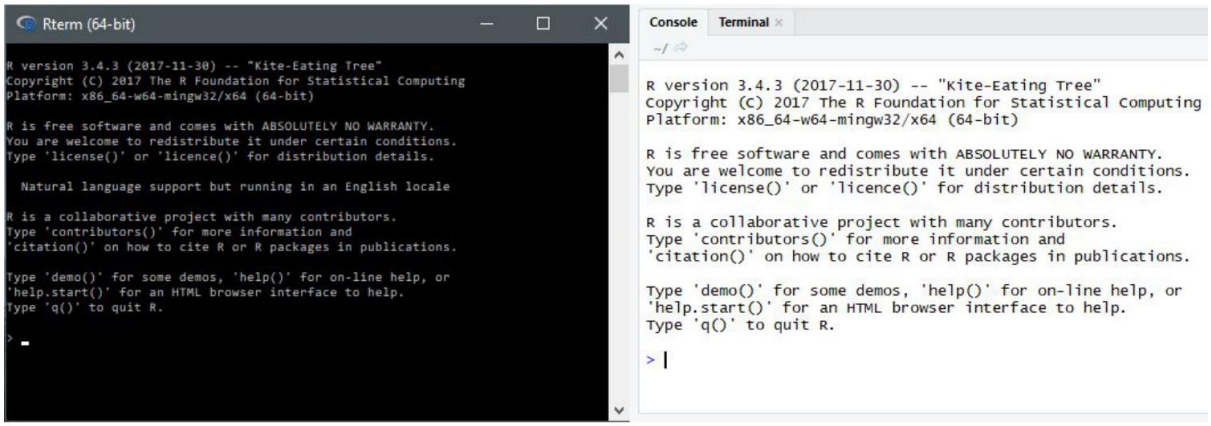
Rstudio can be roughly divided into four quadrants, each with specific and varied functions, plus a main menu bar. When you first open RStudio, you should see a window that looks roughly like this:





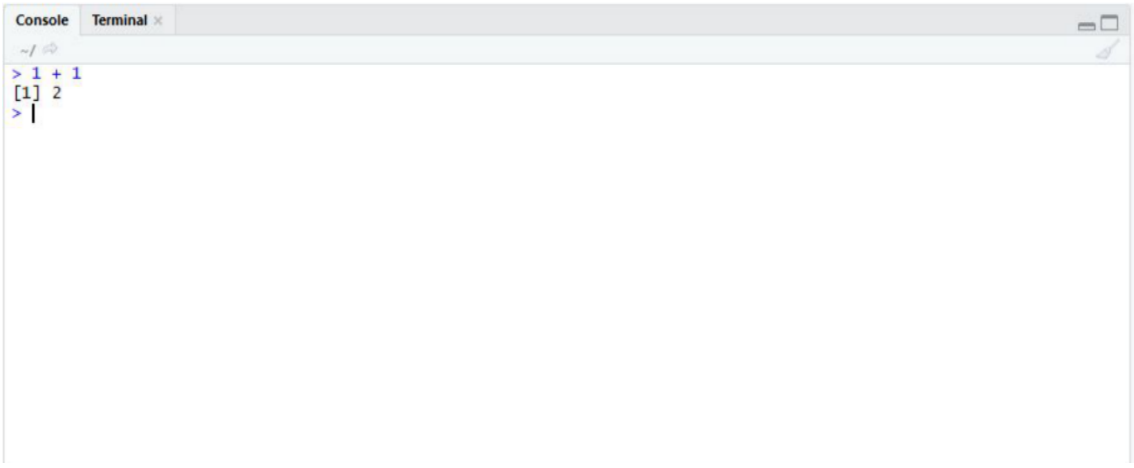
The console

This region should look familiar to you - when you opened R, you were presented with the console. This is where you type and execute commands, and where the output of said command is displayed.



The console

To execute your first command, try typing 1 + 1 then enter at the > prompt. You should see the output [1] 2 below your command.



Typing into the console and getting an output

Now copy and paste the following into your console and hit enter.

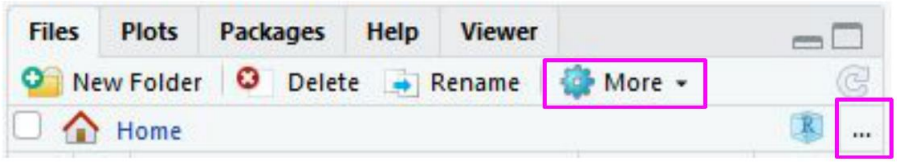
```
example <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8), nrow = 4, ncol = 2)
```

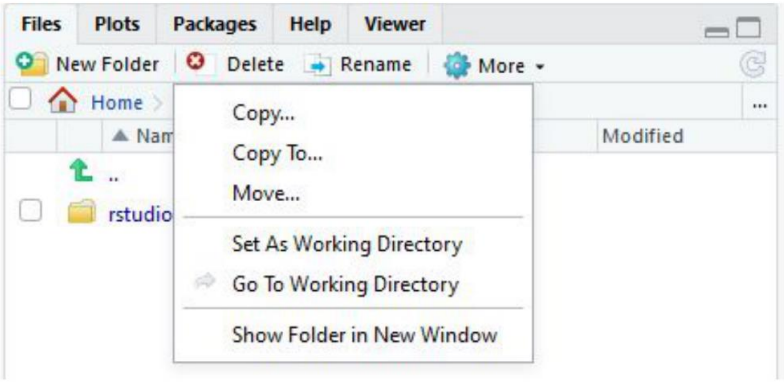
This creates a matrix with four rows and two columns, with the numbers 1 through 8.

Files/help/plots/packages panel

The final region we'll look at occupies the bottom right of the RStudio window. In this quadrant, five tabs run across the top: Files, Plots, Packages, Help, and Viewer.

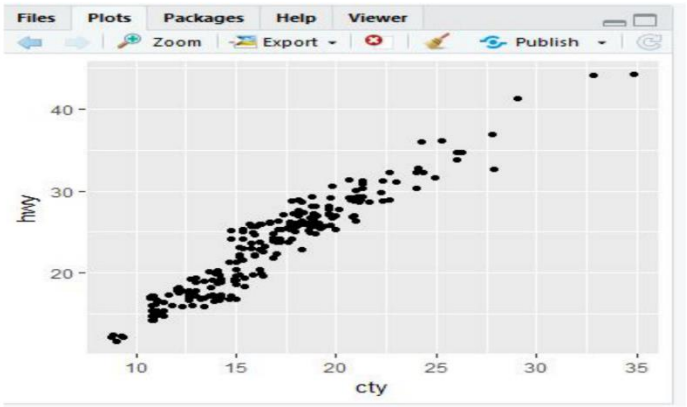
In Files, you can see all of the files in your current working directory. If this isn't where you want to save or retrieve files from, you can also change the current working directory in this tab using the ellipsis at the far right, finding the desired folder, and then under the "More" cogwheel, setting this new folder as the working directory.





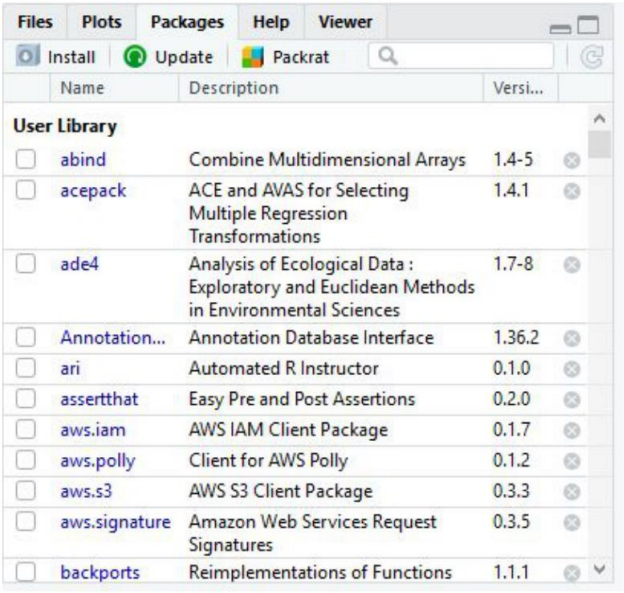
The files tab

In the Plots tab, if you generate a plot with your code, it will appear here. You can use the arrows to navigate to previously generated plots. The Zoom function will open the plot in a new window, that is much larger than the quadrant. Export is how you save the plot. You can either save it as an image or as a PDF. The broom icon clears all plots from memory.



The packages tab

The Packages tab will be explored more in depth in the next lesson on R packages. Here you can see all the packages you have installed, load and unload these packages, and update them.



=X=X=X=X=

What is an R package?

So far, anything we've played around with in R uses the "base" R system. Base R, or everything included in R when you download it, has rather basic functionality for statistics and plotting but it can sometimes be limiting. To expand upon R's basic functionality, people have developed **packages.** A package is a collection of functions, data, and code conveniently provided in a nice, complete format for you. At the time of writing, there are just over 14,300 packages available to download - each with their own specialized functions and code, all for some different purpose. For a really in depth look at R Packages (what they are, how to develop them), check out Hadley Wickham's book from O'Reilly, "R Packages."

Side note: A package is not to be confused with a **library** (these two terms are often conflated in colloquial speech about R). A library is the place where the package is located on your computer. To think of an analogy, a library is, well, a library… and a package is a book within the library. The library is where the books/packages are located.

Packages are what make R so unique. Not only does base R have some great functionality but these packages greatly expand its functionality. And perhaps most special of all, each package is developed and published by the R community at large and deposited in **repositories.**

What are repositories?

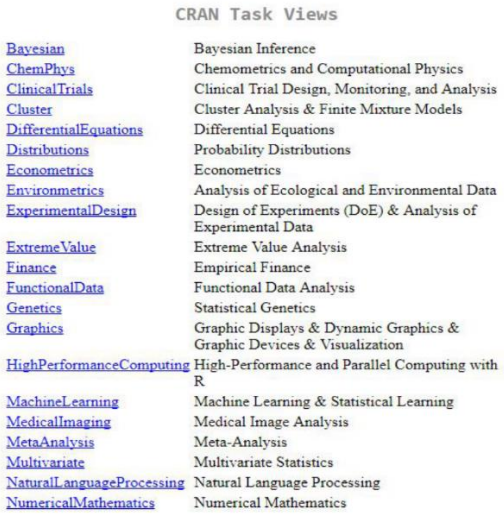
A repository is a central location where many developed packages are located and available for download.

There are three big repositories:
1. **CRAN (Comprehensive R Archive Network):** R's main repository (>12,100 packages available!)
2. **BioConductor:** A repository mainly for bioinformatic-focused packages
3. **GitHub:** A very popular, open source repository (not R specific!)

How do you know what package is right for you?

So, you know where to find packages… but there are so many of them, how can you find a package that will do what you are trying to do in R? There are a few different avenues for exploring packages.

First, CRAN groups all of its packages by their functionality/topic into 35 "themes." It calls this its "Task view." This at least allows you to narrow the packages you can look through to a topic relevant to your interests.



CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
ExtremeValue	Extreme Value Analysis
Finance	Empirical Finance
FunctionalData	Functional Data Analysis
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics

Second, there is a great website, RDocumentation, which is a search engine for packages and functions from CRAN, BioConductor, and GitHub (ie: the big three repositories). If you have a task in mind, this is a great way to search for specific packages to help you accomplish that task! It also has a "task" view like CRAN, that allows you to browse themes.

More often, if you have a specific task in mind, googling that task followed by "R package" is a great place to start! From there, looking at tutorials, vignettes, and forums for people already doing what you want to do is a great way to find relevant packages.

How do you install packages?

Great! You've found a package you want… How do you install it?

Installing from CRAN

If you are installing from the CRAN repository, use the install.packages() function, with the name of the package you want to install in quotes between the parentheses (note: you can use either single or double quotes). For example, if you want to install the package "ggplot2", you would use: install.packages("ggplot2")

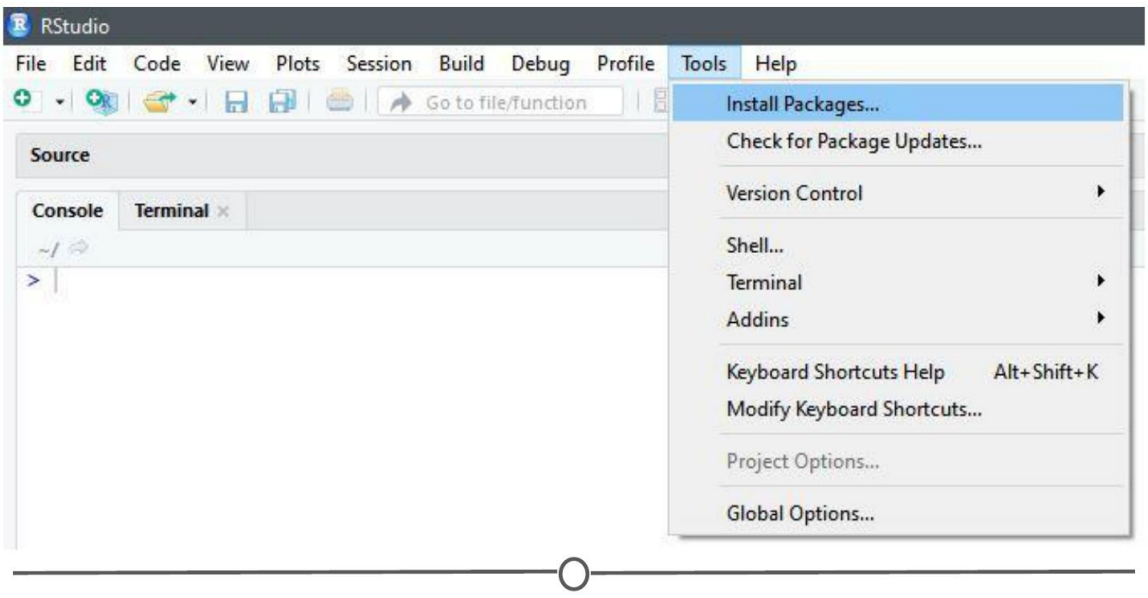
Try doing so in your R console! This command downloads the "ggplot2" package from CRAN and installs it onto your computer.

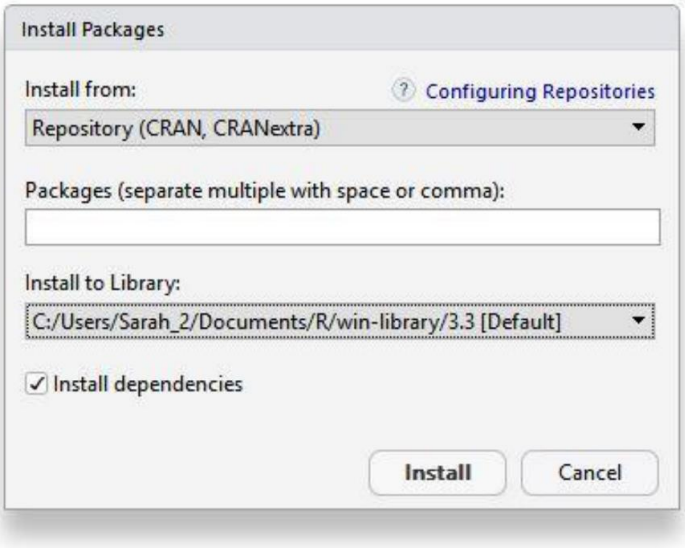
If you want to install multiple packages at once, you can do so by using a character vector, like: install.packages(c("ggplot2", "devtools", "lme4"))

If you want to use RStudio's graphical interface to install packages, go to the Tools menu, and the first option should be "Install packages…" If installing from CRAN, select it as the repository and type the desired packages in the appropriate box

```
install.packages("ggplot2")
```

```
install.packages(c("ggplot2", "devtools", "lme4"))
```





Installing from Bioconductor

The BioConductor repository uses their own method to install packages. First, to get the basic functions required to install through BioConductor, use: source("https://bioconductor.org/biocLite.R")

This makes the main install function of BioConductor, biocLite(), available to you. Following this, you call the package you want to install in quotes, between the parentheses of the biocLite command, like so: biocLite("GenomicFeatures")

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

```
source("https://bioconductor.org/biocLite.R")
```

```
biocLite()
```

```
biocLite("GenomicFeatures")
```

Installing from GitHub

This is a more specific case that you probably won't run into too often. In the event you want to do this, you first must find the package you want on GitHub and take note of both the package name AND the author of the package. Check out this guide for installing from GitHub, but the general workflow is:

install.packages("devtools") - only run this if you don't already have devtools installed. If you've been following along with this lesson, you may have installed it when we were practicing installations using the R console

library(devtools) - more on what this command is doing immediately below this

install_github("author/package") replacing "author" and "package" with their GitHub username and the name of the package.

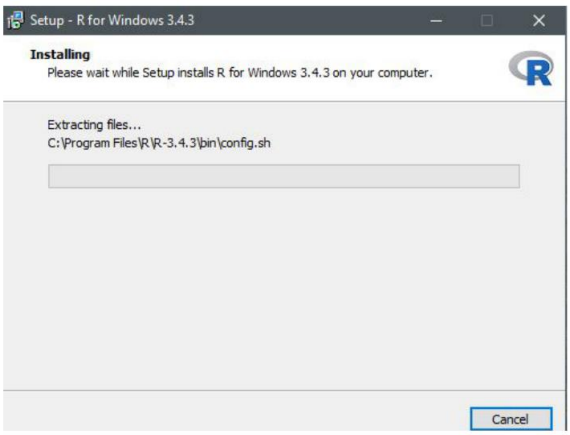
Loading packages

Installing a package does not make its functions immediately available to you. First you must **load** the package into R; to do so, use the library() function. Think of this like any other software you install on your computer. Just because you've *installed* a program, doesn't mean it's automatically running - you have to open the program. Same with R. You've installed it, but now you have to "open" it. For example, to "open" the "ggplot2" package, you would run library(ggplot2)

NOTE: Do **not** put the package name in quotes! Unlike when you are installing the packages, the library() command does not accept package names in quotes!

Step 1: Install

```
install.packages("ggplot2")
```



Step 2: Load

```
library()
```

```
library(ggplot2)
```

Updating, removing, unloading packages

Once you've got a package, there are a few things you might need to know how to do:

Checking what packages you have installed

If you aren't sure if you've already installed a package, or want to check what packages are installed, you can use either of: installed.packages() or library() with nothing between the parentheses to check!

In RStudio, that package tab introduced earlier is another way to look at all of the packages you have installed.

Updating packages

You can check what packages need an update with a call to the function old.packages() This will identify all packages that have been updated since you installed them/last updated them.

To update all packages, use update.packages(). If you only want to update a specific package, just use once again install.packages("packagename")

What packages are installed?

```
installed.packages() or library()
```

Updating packages

```
old.packages()
```

```
update.packages()
```

```
install.packages("packagename")
```

R Projects

One of the ways people organize their work in R is through the use of R Projects, a built-in functionality of RStudio that helps to keep all your related files together. RStudio provides a great guide on how to use Projects so definitely check that out!

What is an R Project?

When you make a Project, it creates a folder where all files will be kept, which is helpful for organizing yourself and keeping multiple projects separate from each other. When you re-open a project, RStudio remembers what files were open and will restore the work environment as if you had never left - which is very helpful when you are starting back up on a project after some time off! Functionally, creating a Project in R will create a new folder and assign that as the working directory so that all files generated will be assigned to the same directory.

What are the benefits to using Projects?

The main benefit of using Projects is that it starts the organization process off right! It creates a folder for you and now you have a place to store all of your input data, your code, and the output of your code. Everything you are working on within a Project is self-contained; which often means finding things is much easier - there's only one place to look!

Also, since everything related to one project is all in the same place, it is much easier to share your work with others - either by directly sharing the folder/files, or by associating it with version control software. We'll talk more about linking Projects in R with version control systems in a future lesson entirely dedicated to the topic!

Finally, since RStudio remembers what documents you had open when you closed the session, it is easier to pick a project up after a break - everything is set-up just as you left it!

Creating a Project

There are three ways to make a Project:
1) From scratch - this will create a new directory for all your files to go in
2) From an existing folder - this will link an existing directory with RStudio
3) From version control - this will "clone" an existing project onto your computer (Don't worry too much about this one, you'll get more familiar with it in the next few lessons)

Let's create a Project from scratch, which is often what you will be doing!

Open RStudio, and under File, select "New Project". You can also create a new Project by using the Projects toolbar and selecting "New Project" in the drop down menu, or there is a "New Project" shortcut in the toolbar.

WEEK

#03

VERSION CONTROL SYSTEM

Version Control

Now that we've got a handle on R, RStudio, and projects, there are a few more things we want to set you up with before moving on to the other courses - understanding version control, installing Git, and linking Git with RStudio. In this lesson, we'll give you a basic understanding of version control.

What is version control?

First things first: What is version control?

i) Version control is a system that records changes that are made to a file or a set of files over time.

ii) As you make edits, the version control system takes snapshots of your files and the changes, and then saves those snapshots so you can refer or revert back to previous versions later if need be!

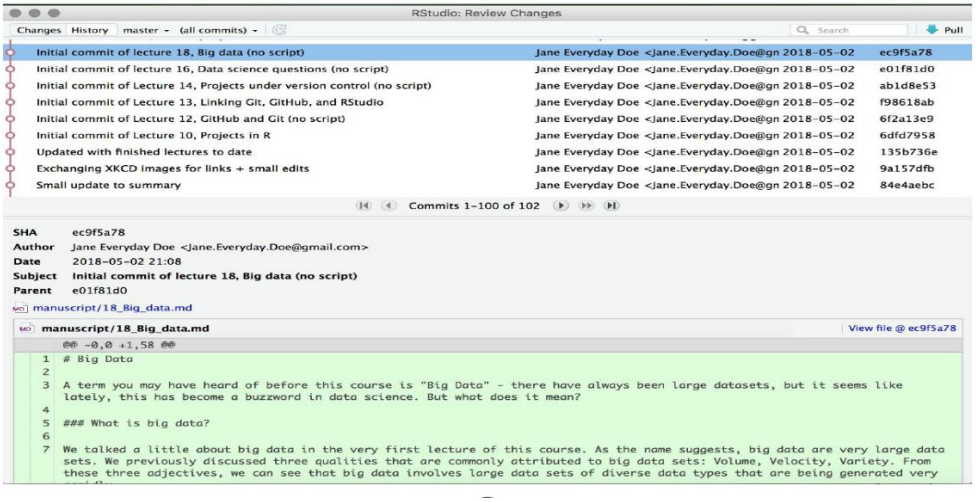
iii) Version control systems, like Git, are like a more sophisticated "Track changes" - in that they are far more powerful and are capable of meticulously tracking successive changes on many files, with potentially many people working simultaneously on the same groups of files.

What are the benefits of using version control?

i) As we've seen in the example, without version control, you might be keeping multiple, very similar copies of a file. And this could be dangerous - you might start editing the wrong version, not recognizing that the document labelled "FINAL" has been further edited to "FINAL2" - and now all your new changes have been applied to the wrong file!

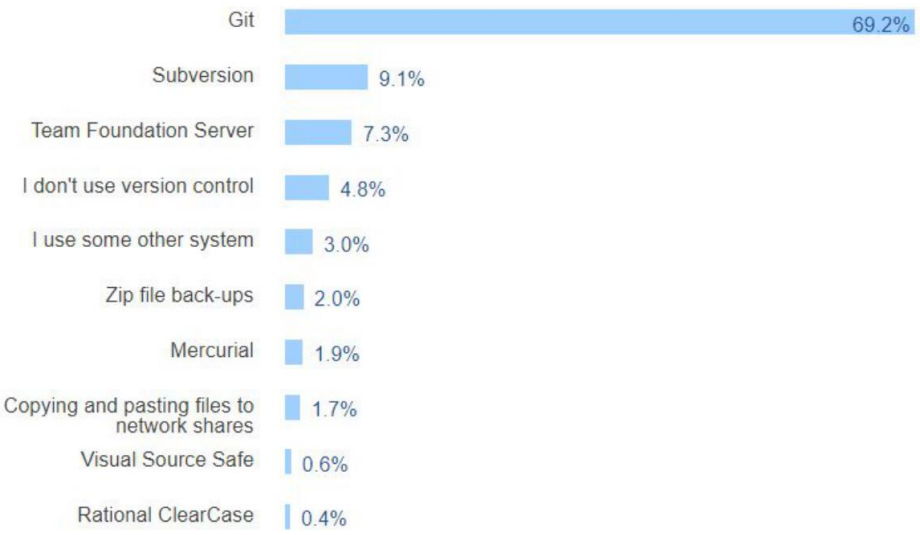
ii) Version control systems help to solve this problem by keeping a single, updated version of each file, with a record of *all* previous versions AND a record of exactly what changed between the versions.

Which brings us to the next major benefit of version control: It keeps a record of all changes made to the files. This can be of great help when you are collaborating with many people on the same files - the version control software keeps track of who, when, and why those specific changes were made. It's like "Track changes" to the extreme!



What is Git? Why should you use it?

Git is a free and open source version control system. It was developed in 2005 and has since become *the* most commonly used version control system around! StackOverflow, which should sound familiar from our Getting Help lesson, surveyed over 60,000 respondents on which version control system they use, and as you can tell from the chart below, Git is by far the winner.



What is GitHub?

GitHub is an online interface for Git. Git is software used locally on your computer to record changes. GitHub is a host for your files and the records of the changes made. You can sort of think of it as being similar to DropBox - the files are on your computer, but they are also hosted online and are accessible from any computer. GitHub has the added benefit of interfacing with Git to keep track of all of your file versions and changes.

Version control vocabulary

There is a lot of vocabulary involved in working with Git, and often the understanding of one word relies on your understanding of a different Git concept. Take some time to familiarize yourself with the words below and read over it a few times to see how the concepts relate.

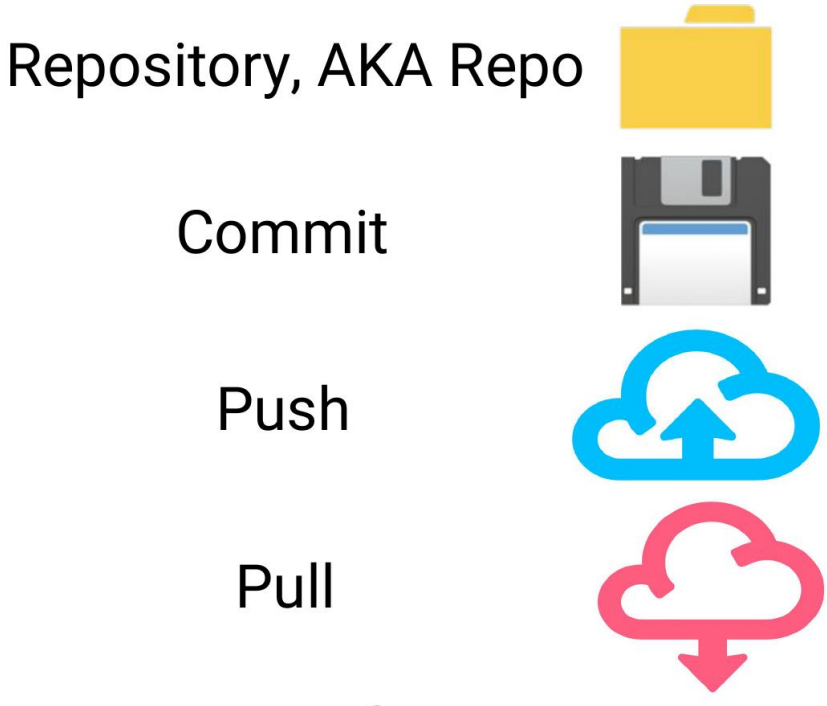
i) **Repository:** Equivalent to the project's folder/directory - all of your version controlled files (and the recorded changes) are located in a repository. This is often shortened to **repo**. Repositories are what are hosted on GitHub and through this interface you can either keep your repositories private and share them with select collaborators, or you can make them public - anybody can see your files and their history.

ii) **Commit:** To commit is to save your edits and the changes made. A commit is like a snapshot of your files: Git compares the previous version of all of your files in the repo to the current version and identifies those that have changed since then. Those that have not changed, it maintains that previously stored file, untouched. Those that have changed, it compares the files, logs the changes and uploads the new version of your file. We'll touch on this in the next section, but when you commit a file, typically you accompany that file change with a little note about what you changed and why.

When we talk about version control systems, commits are at the heart of them. If you find a mistake, you revert your files to a previous *commit.* If you want to see what has changed in a file over time, you compare the *commits* and look at the messages to see why and who.

iii) **Push:** Updating the repository with your edits. Since Git involves making changes locally, you need to be able to share your changes with the common, online repository. Pushing is sending those committed changes to that repository, so now everybody has access to your edits.

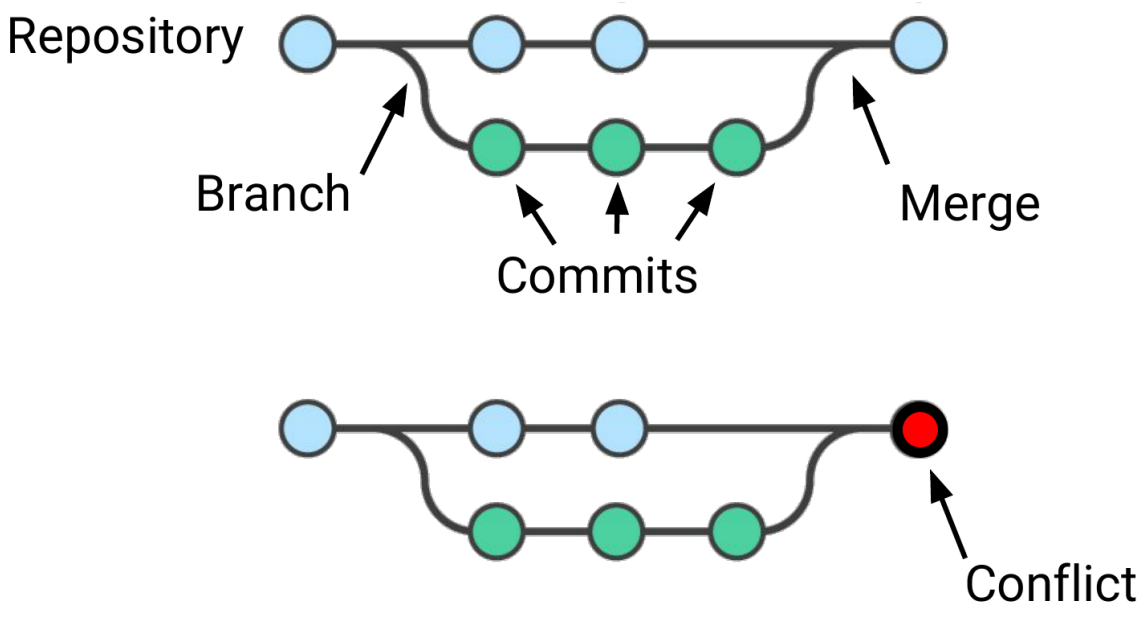
iv) **Pull:** Updating your local version of the repository to the current version, since others may have edited in the meanwhile. Because the shared repository is hosted online and any of your collaborators (or even yourself on a different computer!) could have made changes to the files and then pushed them to the shared repository, you are behind the times! The files you have locally on *your* computer may be outdated, so you pull to check if you are up to date with the main repository.

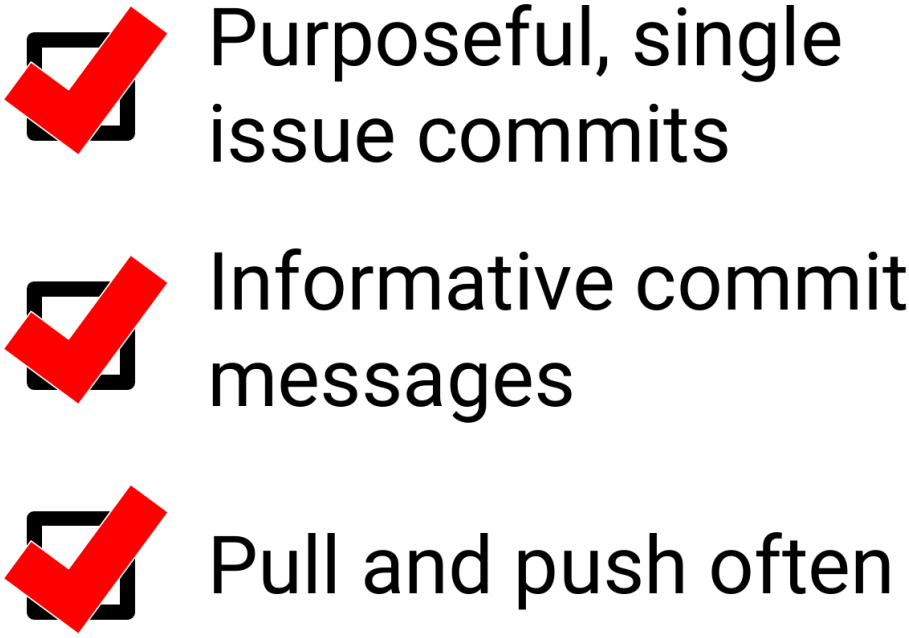


v) **Staging:** The act of preparing a file for a commit. For example, if since your last commit you have edited three files for completely different reasons, you don't want to commit all of the changes in one go; your message on why you are making the commit and what has changed will be complicated since three files have been changed for different reasons. So instead, you can stage just one of the files and prepare it for committing. Once you've committed that file, you can stage the second file and commit it. And so on. Staging allows you to separate out file changes into separate commits. Very helpful!

vi) **Branch:** When the same file has two simultaneous copies. When you are working locally and editing a file, you have created a branch where your edits are not shared with the main repository (yet) - so there are two versions of the file: the version that everybody has access to on the repository and your local edited version of the file. Until you push your changes and merge them back into the main repository, you are working on a branch. Following a branch point, the version history splits into two and tracks the independent changes made to both the original file in the repository that others may be editing, and tracking your changes on your branch, and then merges the files together.

vii) **Merge:** Independent edits of the same file are incorporated into a single, unified file. Independent edits are identified by Git and are brought together into a single file, with both sets of edits incorporated. But, you can see a potential problem here - if both people made an edit to the same sentence that precludes one of the edits from being possible, we have a problem! Git recognizes this disparity (**conflict**) and asks for user assistance in picking which edit to keep.





GitHub and Git

Now that we've got a handle on what version control is, in this lesson, you will sign-up for a GitHub account, navigate around the GitHub website to become familiar with some of its features, and install and configure Git; all in preparation for linking both with your RStudio!

What is GitHub?

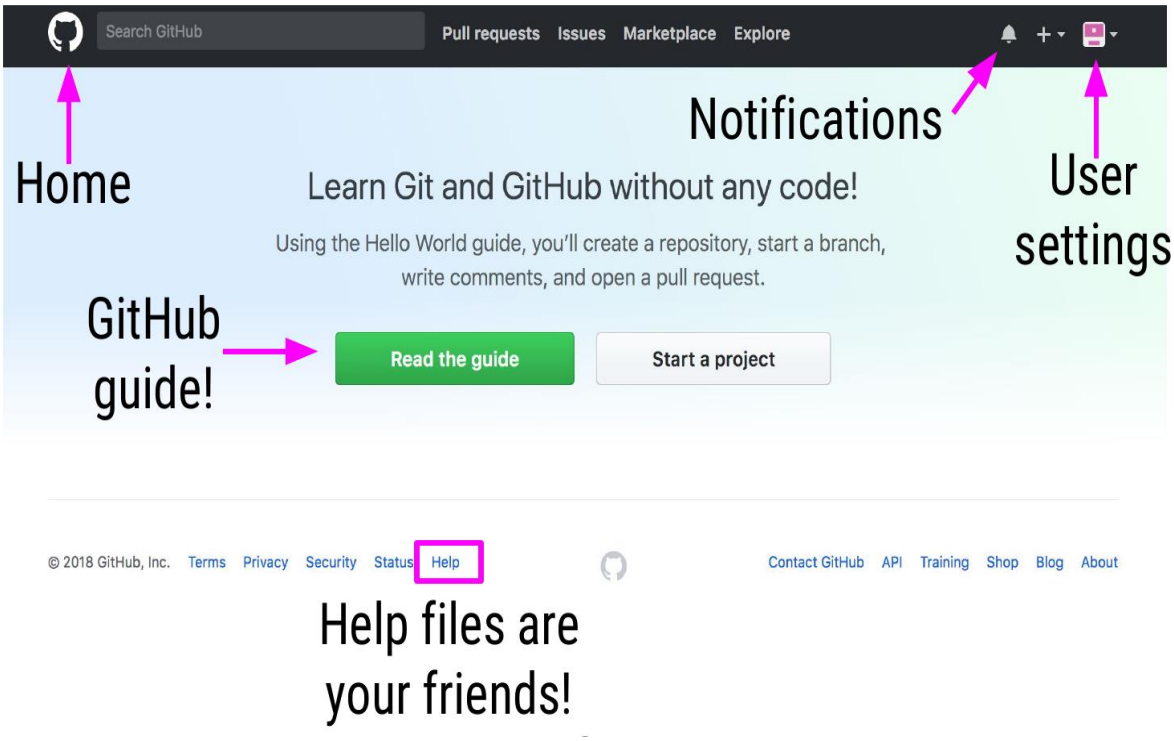
As we previously learned, GitHub is a cloud-based management system for your version controlled files. Like DropBox, your files are both locally on your computer *and* hosted online and easily accessible. Its interface allows you to manage version control and provides users with a web-based interface for creating projects, sharing them, updating code, etc.

The homepage

We're going to take a quick tour of the GitHub website, and we'll particularly focus on these sections of the interface:

1. User settings
2. Notifications
3. Help files
4. The GitHub guide

Following this tour, we'll make your very first repository using the GitHub guide!



User settings

Now that you've logged on to GitHub, we should fill out some of your profile information and get acquainted with the account settings. In the upper right corner, there is an icon with an arrow beside it, click this and go to "Your profile"

Your profile

Since you are just starting out, you aren't going to have any repositories or contributions yet - but hopefully we'll change that soon enough! What we can do right now is edit your profile.

Go to "Edit profile" along the lefthand edge of the page. Here, take some time and fill out your name and a little description of yourself in the "Bio" box, and if you like, upload a picture of yourself! When you are done, click "Update profile"

Editing your profile page

Along the lefthand side of this page, there are many options for you to explore. Click through each of these menus to get familiar with the options available to you. To get you started, go to the account page.

Your account page

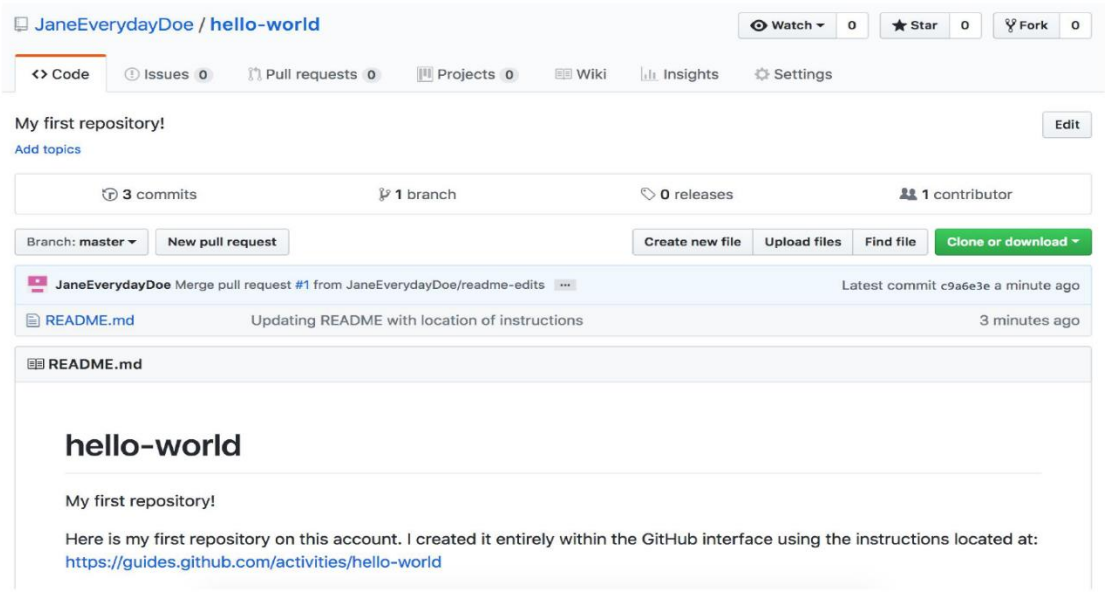
Here, you can edit your password or if you are unhappy with your username, change it. Be careful though, there can be unintended consequences when you change your username - if you are just starting out and don't have any content yet, you'll probably be safe though.

Continue looking through the personal setting options on your own. When you are done, go back to your profile.

Once you've had a bit more experience with GitHub, you'll eventually end up with some repositories to your name. To find those, click on the "Repositories" link on your profile.

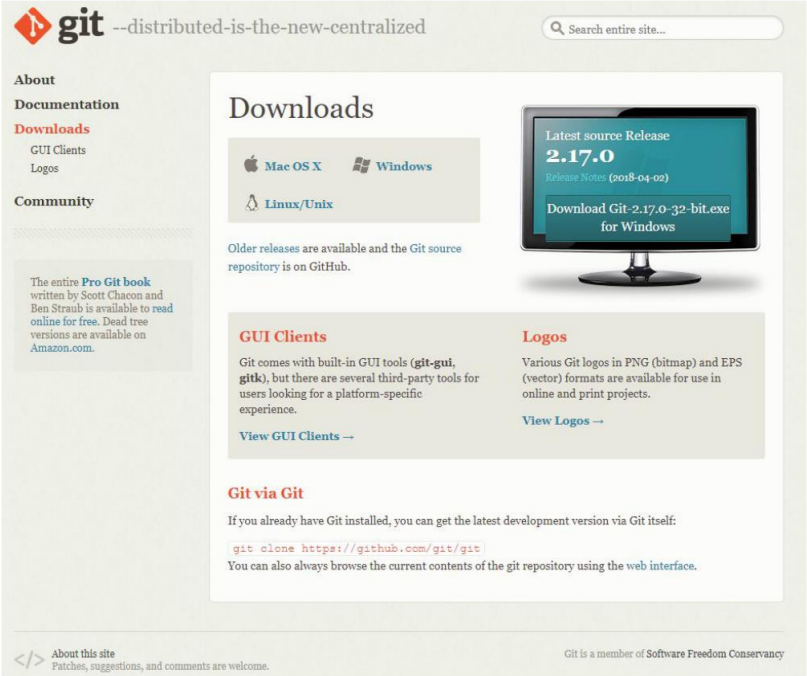
The GitHub guide

GitHub recognizes that this can be an overwhelming process for new users, and as such have developed a mini tutorial to get you started with GitHub. Go through this guide now and create your first repository! When you are done, you should have a repository that looks something like this:



Downloading and installing Git

To download Git, go to https://git-scm.com/download. You should arrive at a webpage like this:

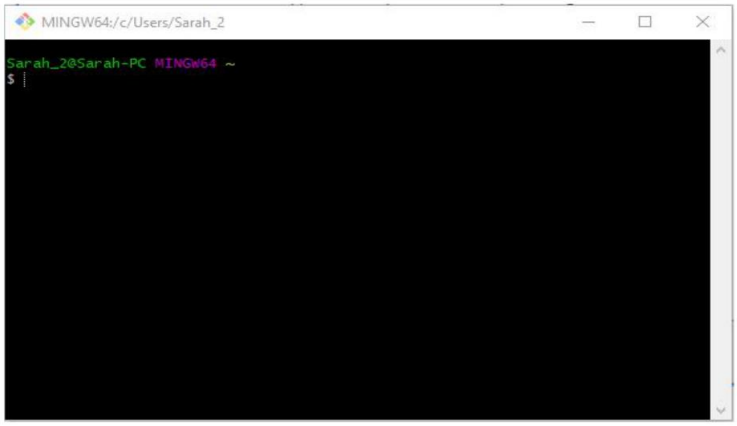


For Windows

Once the download is finished, open the .exe file to initiate the installation wizard. If you receive a security warning, click “Run” and/or “Allow.” Following this, click through the installation wizard, generally accepting the default options unless you have a compelling reason not to.

Finishing the install process

Doing so, a command line environment will open. Provided you accepted the default options during the installation process, there will now be a Start menu shortcut to launch Git Bash in the future. You have now installed Git.

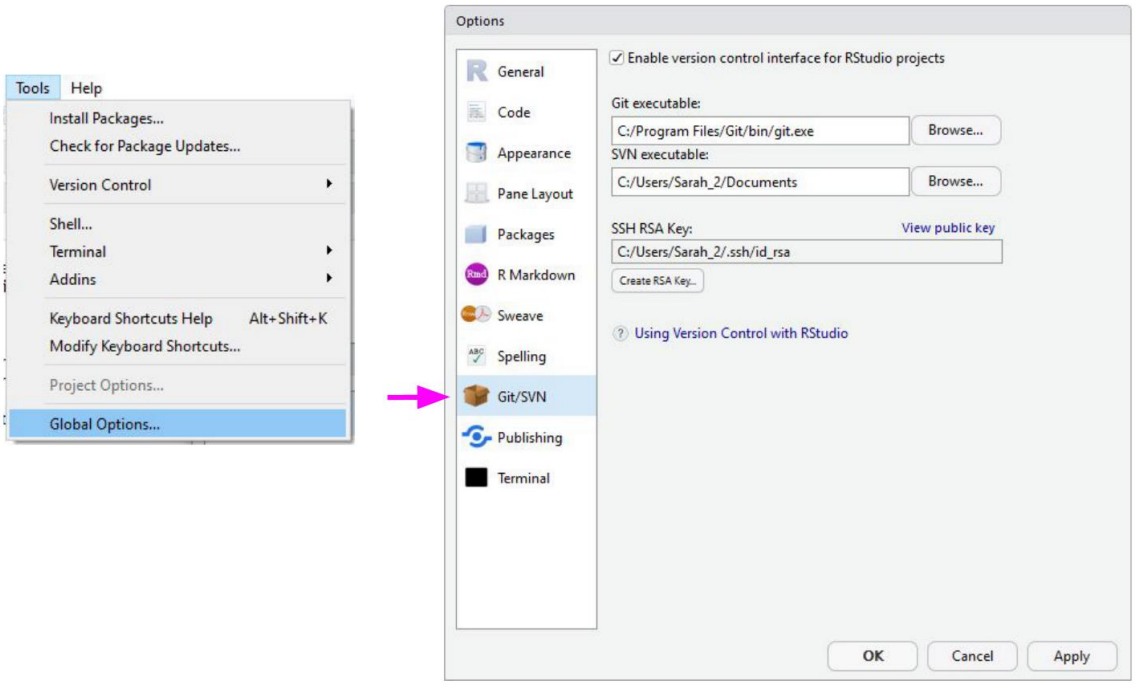


Linking Git/GitHub and RStudio

Now that we have both RStudio and Git set-up on your computer and a GitHub account, it's time to link them together so that you can maximize the benefits of using RStudio in your version control pipelines.

Linking RStudio and Git

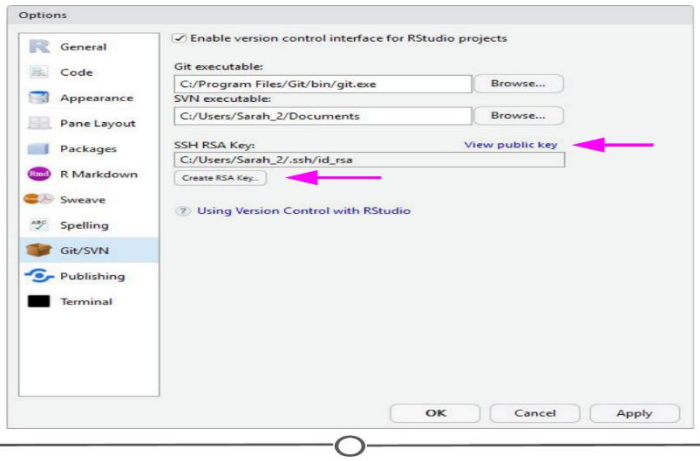
In RStudio, go to Tools > Global Options > Git/SVN



Linking RStudio and GitHub

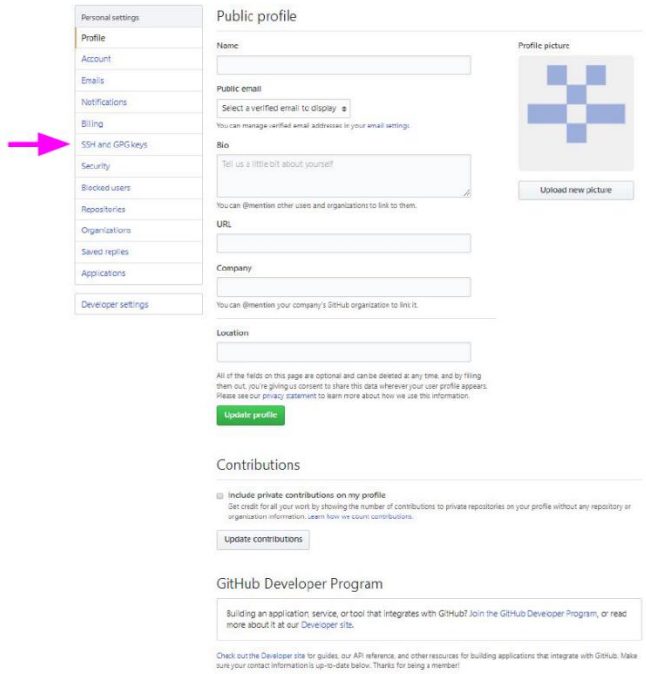
In that same RStudio option window, click "Create RSA Key" and when this completes, click "Close."

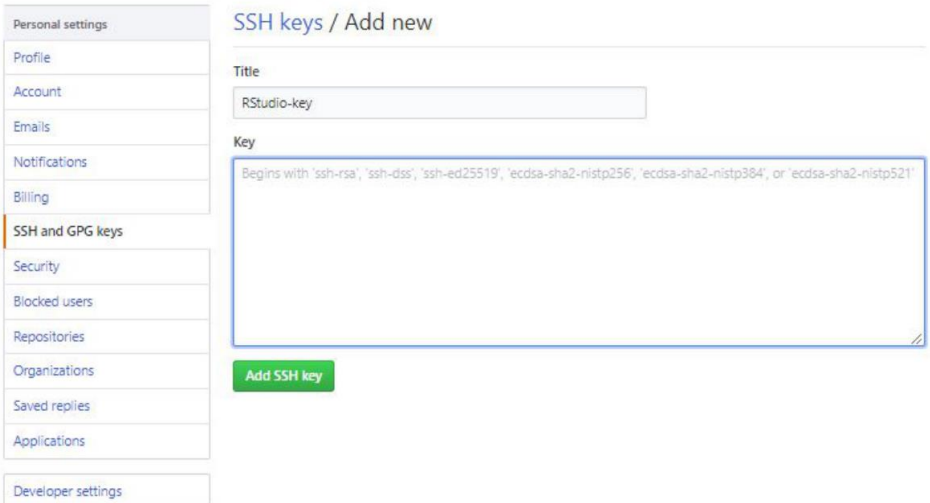
Following this, in that same window again, click "View public key" and copy the string of numbers and letters. Close this window.



You have now created a key that is specific to you which we will provide to GitHub, so that it knows who you are when you commit a change from within RStudio.

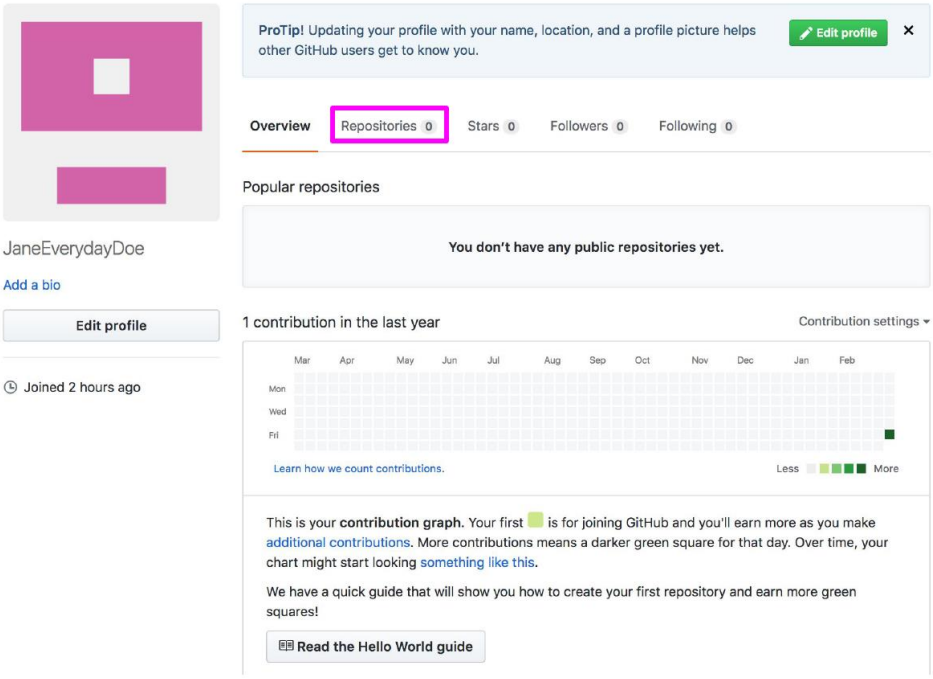
To do so, go to github.com/, log-in if you are not already, and go to your account settings. There, go to “SSH and GPG keys” and click “New SSH key”. Paste in the public key you have copied from RStudio into the Key box and give it a Title related to RStudio. Confirm the addition of the key with your GitHub password.

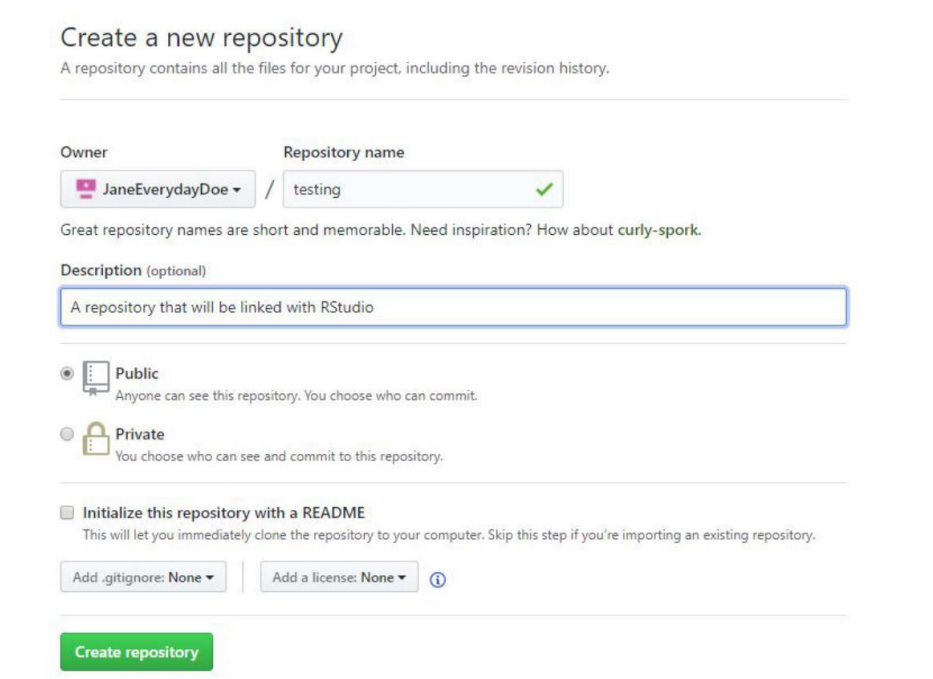




Create a new repository and edit it in RStudio

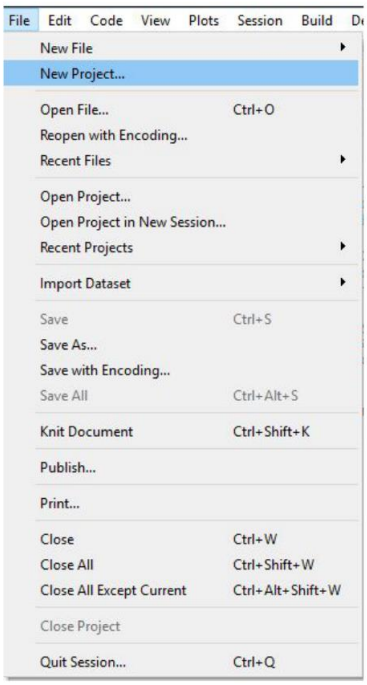
On GitHub, create a new repository (github.com > Your Profile > Repositories > New). Name your new test repository and give it a short description. Click Create repository. Copy the URL for your new repository.

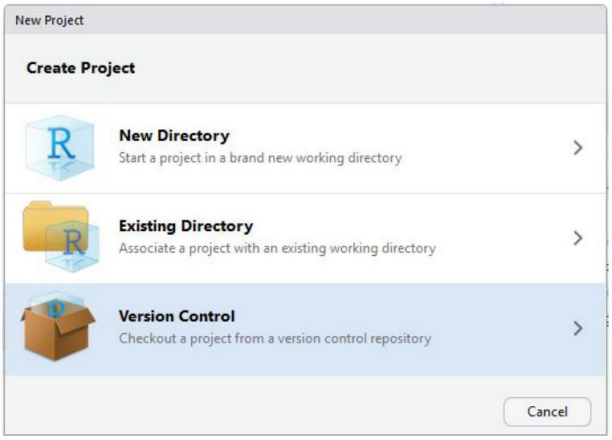


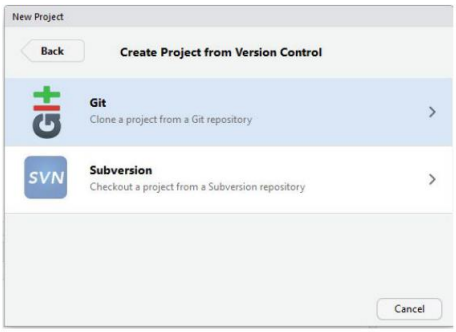


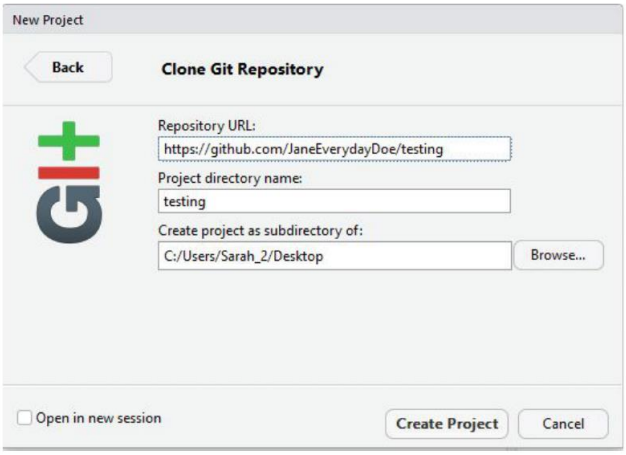
Creating a new repository on GitHub

In RStudio, go to File > New Project. Select Version Control. Select Git as your version control software. Paste in the repository URL from before, select the location where you would like the project stored. When done, click on “Create Project”. Doing so will initialize a new project, linked to the GitHub repository, and open a new session of RStudio.





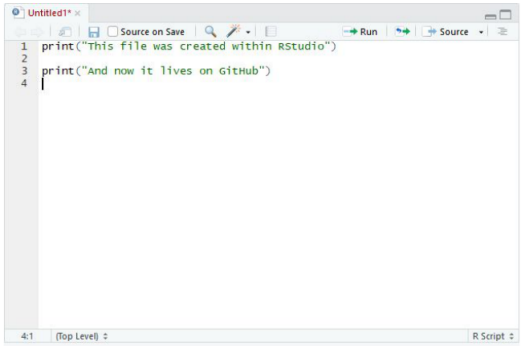


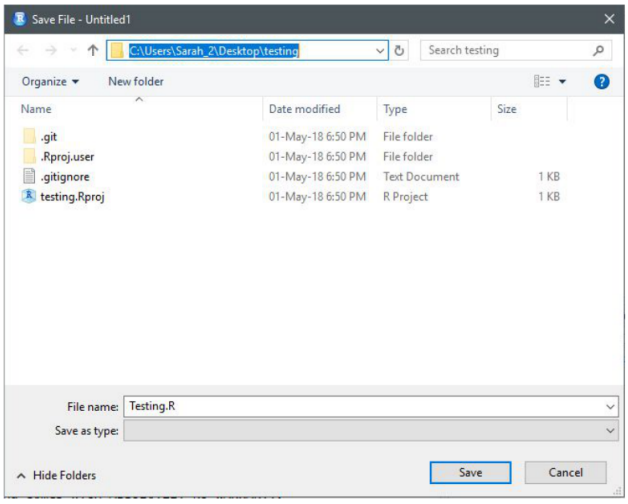


Create a new R script (File > New File > R Script) and copy and paste the following code:

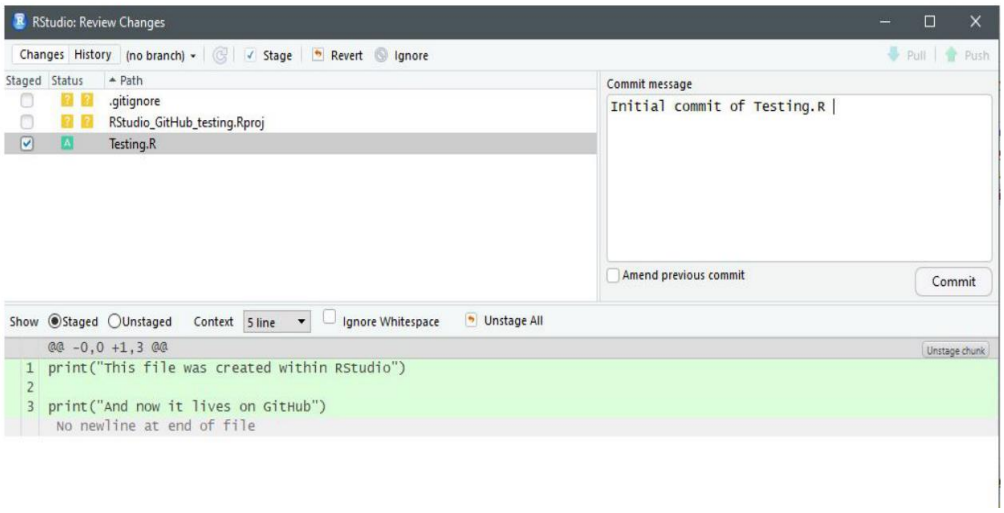
```
print("This file was created within RStudio")

print("And now it lives on GitHub")
```

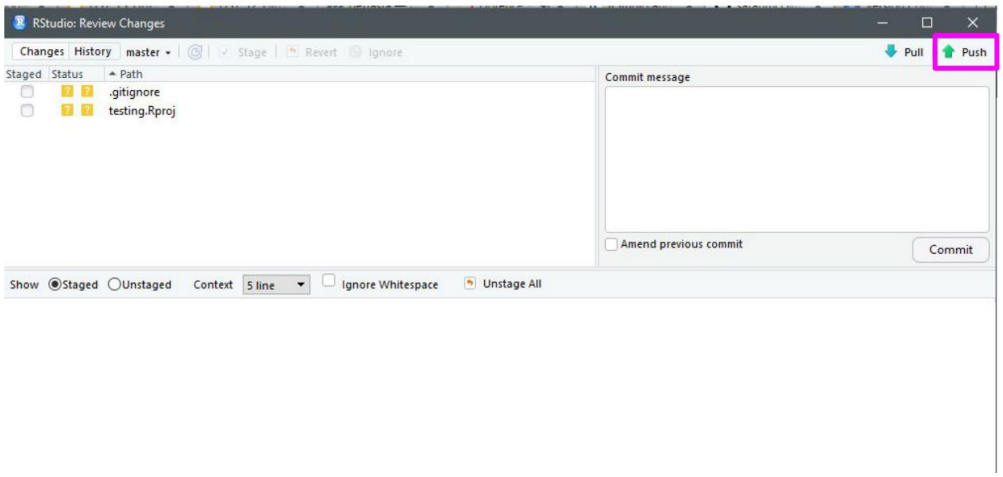
Save the file. Note that when you do so, the default location for the file is within the new Project directory you created earlier.





Click “Commit”. A new window should open, that lists all of the changed files from earlier, and below that shows the differences in the staged files from previous versions. In the upper quadrant, in the “Commit message” box, write yourself a commit message. Click Commit. Close the window.




RStudio: Review Changes
Changes
History
master
Stage
Revert
Ignore
Pull
Push
Staged
Status
Path
.gitignore
testing.Rproj
Commit message
Amend previous commit
Commit
Show
Staged
Unstaged
Context
5 line
Ignore Whitespace
Unstage All

=x=x=x=x=x=

WEEK

#04

R MARKDOWN, SCIENTIFIC THINKING, AND BIG DATA

R Markdown

We've spent a lot of time getting R and RStudio working, learning about projects and version control - you are practically an expert at this! There is one last major functionality of R/RStudio that we would be remiss to not include in your introduction to R - Markdown!

What is R Markdown?

R Markdown is a way of creating fully reproducible documents, in which both text and code can be combined. In fact, these lessons are written using R Markdown! That's how we make things:

- bullets
- **bold**
- *italics*
- links
- or run inline `r code`

Despite these documents all starting as plain text, you can render them into HTML pages, or PDFs, or Word documents, or slides! The symbols you use to signal, for example, **bold** or *italics* is compatible with all of those formats.

Why use R Markdown?

i) One of the main benefits is the reproducibility of using R Markdown. Since you can easily combine text and code chunks in one document, you can easily integrate introductions, hypotheses, your code that you are running, the results of that code and your conclusions all in one document.

ii) Sharing what you did, why you did it and how it turned out becomes so simple - and that person you share it with can re-run your code and *get the exact same answers you got.* That's what we mean about reproducibility.

iii) But also, sometimes you will be working on a project that takes many weeks to complete; you want to be able to see what you did a long time ago (and perhaps be reminded exactly why you were doing this) and you can see exactly what you ran AND the results of that code - and R Markdown documents allow you to do that.

iv) Another major benefit to R Markdown is that since it is plain text, it works very well with version control systems.

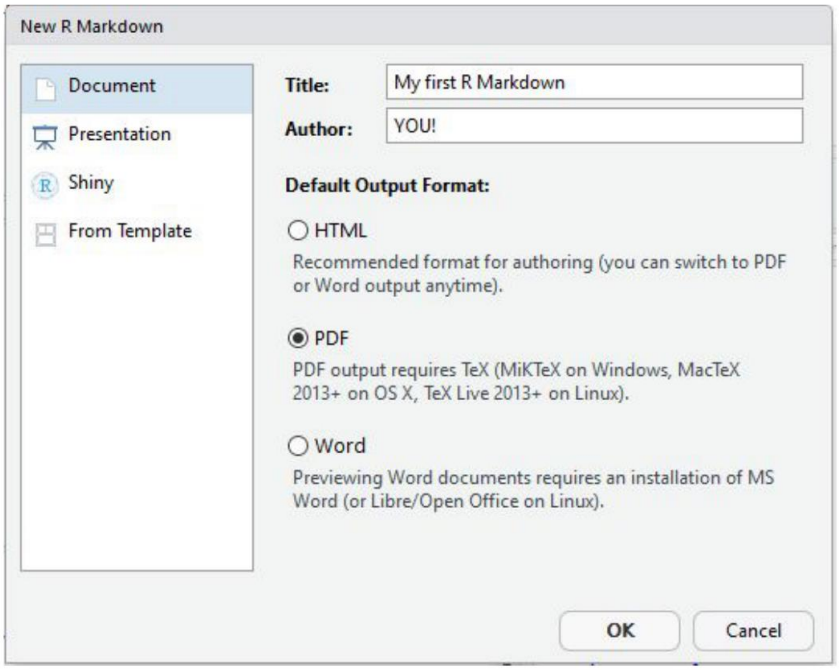
v) It is easy to track what character changes occur between commits; unlike other formats that aren't plain text. For example, in one version of this lesson, I may have forgotten to bold **this** word. When I catch my mistake, I can make the plain text changes to signal I would like that word bolded, and in the commit, you can see the exact character changes that occurred to now make the word bold.

Installation

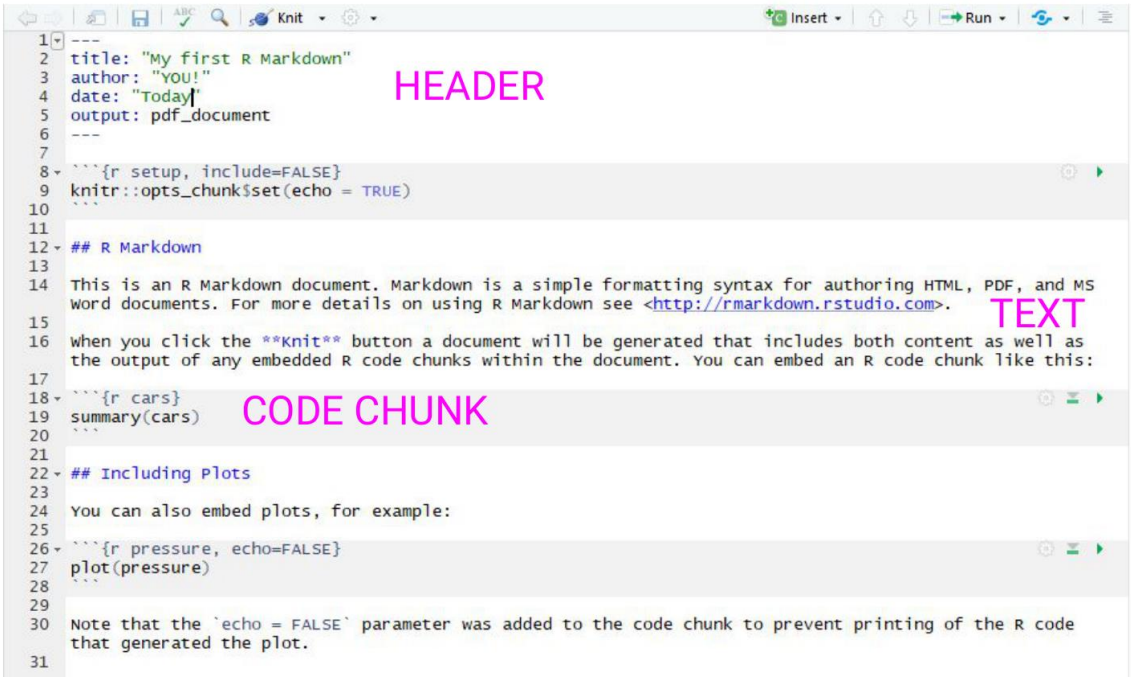
Another (selfish) benefit of R Markdown is how easy it is to use! Like everything in R, this extended functionality comes from an R package - "rmarkdown." All you need to do to install it is run `install.packages("rmarkdown")`

Getting started with R Markdown

To create an R Markdown document, in R Studio, go to File > New File > R Markdown. You will be presented with the following window:



I've filled in a title and an author and switched the output format to a PDF. Explore around this window and the tabs along the left to see all the different formats that you can output to. When you are done, click OK, and a new window should open with a little explanation on R Markdown files.




```
1  ---
2  title: "My first R Markdown"
3  author: "YOU!"
4  date: "Today"
5  output: pdf_document
6  ---
7
8  ```{r setup, include=FALSE}
9  knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS
   Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
15
16 When you click the **Knit** button a document will be generated that includes both content as well as
   the output of any embedded R code chunks within the document. You can embed an R code chunk like this:
17
18 ```{r cars}
19 summary(cars)
20 ```
21
22 ## Including Plots
23
24 You can also embed plots, for example:
25
26 ```{r pressure, echo=FALSE}
27 plot(pressure)
28 ```
29
30 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code
   that generated the plot.
31
```

There are three main sections of an R Markdown document. The first is the **header** at the top, bounded by the three dashes. This is where you can specify details like the title, your name, the date, and what kind of document you want output. If you filled in the blanks in the window earlier, these should be filled out for you.

Also on this page, you can see **text sections**, for example, one section starts with "## R Markdown" - We'll talk more about what this means in a second, but this section will render as text when you produce the PDF of this file - and all of the formatting you will learn generally applies to this section.

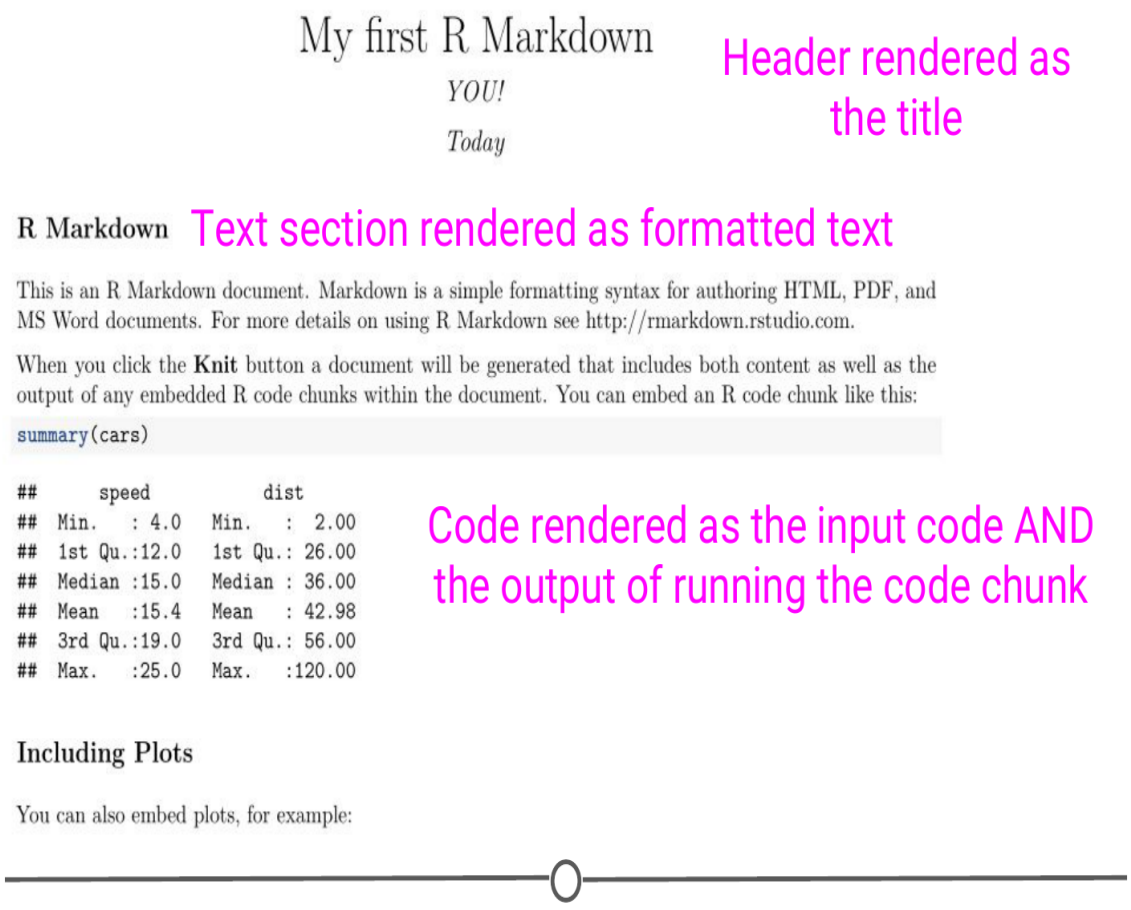
And finally, you will see **code chunks**. These are bounded by the triple backticks. These are pieces of R code ("chunks") that you can run right from within your document - and the output of this code will be included in the PDF when you create it.

The easiest way to see how each of these sections behave is to produce the PDF!

"Knitting" documents

When you are done with a document, in R Markdown, you are said to **"knit"** your plain text and code into your final document. To do so, click on the "Knit" button along the top of the source panel. When you do so, it will prompt you to save the document as an RMD file. Do so.

You should see a document like this:



Some easy Markdown commands?

To start, let's look at bolding and italicising text.

i) To bold text, you surround it by two asterisks on either side. Similarly, to italicise text, you surround the word with a single asterisk on either side. `**bold**` and `*italics*` respectively.

ii) We've also seen from the default document that you can make section headers. To do this, you put a series of hash marks (#). The number of hash marks determines what level of heading it is. One hash is the highest level and will make the largest text (see the first line of this lecture), two hashes is the next highest level and so on. Play around with this formatting and make a series of headers, like so:

```
# Header level 1
## Header level 2
### Header level 3...
```

iii) The other thing we've seen so far is code chunks. To make an R code chunk, you can type the three backticks, followed by the curly brackets surrounding a lower case R, put your code on a new line and end the chunk with three more backticks.

If you aren't ready to knit your document yet, but want to see the output of your code, select the line of code you want to run and use Ctrl+Enter or hit the "Run" button along the top of your source window.

The text "Hello world" should be output in your console window. If you have multiple lines of code in a chunk and you want to run them all in one go, you can run the entire chunk by using Ctrl+Shift+Enter OR hitting the green arrow button on the right side of the chunk OR going to the Run menu and selecting Run current chunk.

Lists are easily created by preceding each prospective bullet point by a single dash, followed by a space. Importantly, at the end of each bullet's line, end with **TWO spaces. This is a quirk of R Markdown that will cause spacing problems if not included.**

=x=x=x=x=x=x=

Types of data science questions

The main divisions of data science questions

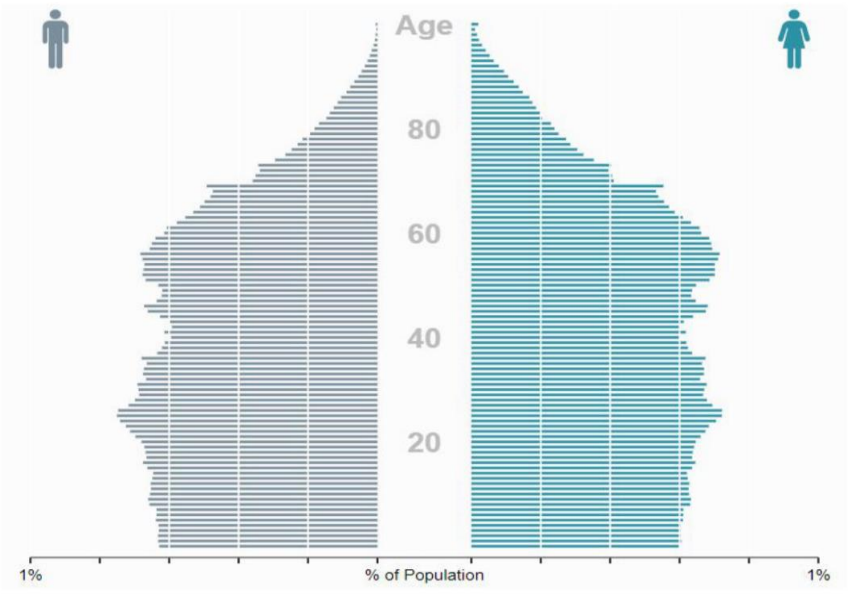
There are, broadly speaking, six categories in which data analyses fall. In the approximate order of difficulty, they are:

1. Descriptive
2. Exploratory
3. Inferential
4. Predictive
5. Causal
6. Mechanistic

Descriptive analysis

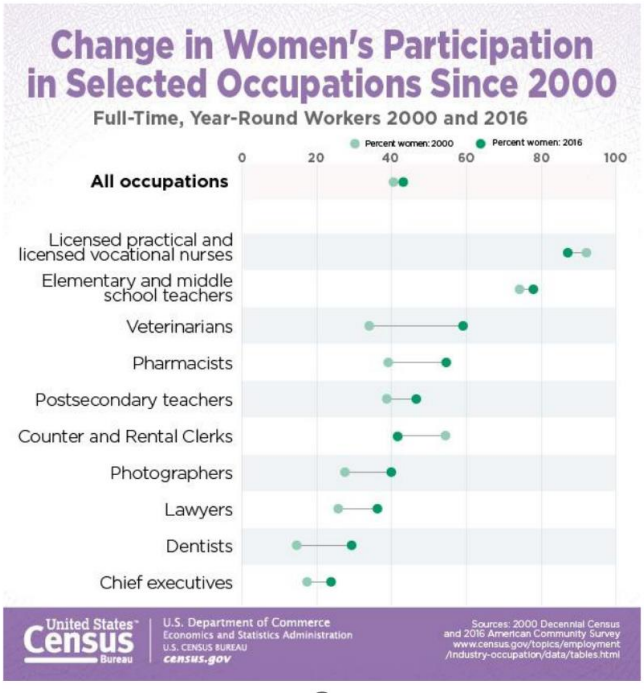
- The goal of descriptive analysis is to **describe** or **summarize** a set of data. Whenever you get a new dataset to examine, this is usually the first kind of analysis you will perform.
- Descriptive analysis will generate simple summaries about the samples and their measurements. You may be familiar with common descriptive statistics: measures of central tendency (eg: mean, median, mode) or measures of variability (eg: range, standard deviations or variance).
- This type of analysis is aimed at summarizing your sample – not for generalizing the results of the analysis to a larger population or trying to make conclusions. Description of data is separated from making interpretations; generalizations and interpretations require additional statistical steps.

Some examples of purely descriptive analysis can be seen in censuses. Here, the government collects a series of measurements on all of the country's citizens, which can then be summarized. Here, you are being shown the age distribution in the US, stratified by sex. The goal of this is just to describe the distribution. There is no inferences about what this means or predictions on how the data might trend in the future. It is just to show you a summary of the data collected.



Exploratory analysis

- The goal of exploratory analysis is to examine or **explore** the data and find **relationships** that weren't previously known.
- Exploratory analyses explore how different measures might be related to each other but do not confirm that relationship as causitive. You've probably heard the phrase "Correlation does not imply causation" and exploratory analyses lie at the root of this saying. Just because you observe a relationship between two variables during exploratory analysis, it does not mean that one necessarily causes the other.
- Because of this, exploratory analyses, while useful for discovering new connections, should not be the final say in answering a question! It can allow you to formulate hypotheses and drive the design of future studies and data collection, but exploratory analysis alone should never be used as the final say on why or how data might be related to each other.



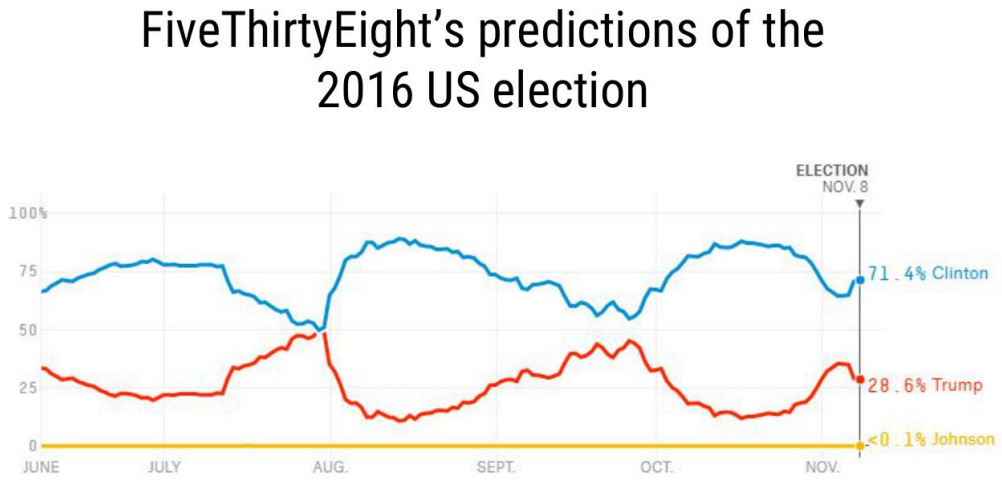
Inferential analysis

- The goal of inferential analyses is to use a relatively **small sample** of data to **infer** or say something about the **population** at large.
- Inferential analysis is commonly the goal of statistical modelling, where you have a small amount of information to extrapolate and generalize that information to a larger group.

Predictive analysis

- The goal of predictive analysis is to use **current** data to make **predictions** about **future** data. Essentially, you are using current and historical data to find patterns and predict the likelihood of future outcomes.

- Like in inferential analysis, your accuracy in predictions is dependent on measuring the right variables. If you aren't measuring the right variables to predict an outcome, your predictions aren't going to be accurate.
- Additionally, there are many ways to build up prediction models with some being better or worse for specific cases, but in general, having more data and a simple model generally performs well at predicting future outcomes.
- All this being said, much like in exploratory analysis, just because one variable may predict another, it does not mean that one causes the other; you are just capitalizing on this observed relationship to predict the second variable.
- A common saying is that prediction is hard, especially about the future. There aren't easy ways to gauge how well you are going to predict an event until that event has come to pass; so evaluating different approaches or models is a challenge.



Causal analysis

The caveat to a lot of the analyses we've looked at so far is that we can only see correlations and can't get at the cause of the relationships we observe. Causal analysis fills that gap; the goal of causal analysis is to see what happens to one variable when we manipulate another variable - looking at the **cause** and **effect** of a **relationship**.

Mechanistic analysis

Mechanistic analyses are not nearly as commonly used as the previous analyses - the goal of mechanistic analysis is to understand the **exact changes in variables** that lead to **exact changes in other variables**.

Experimental Design

Now that we've looked at the different types of data science questions, we are going to spend some time looking at experimental design concepts. As a data scientist, you are a *scientist* and as such, need to have the ability to design proper experiments to best answer your data science questions!

What does experimental design mean?

Experimental design is organizing an experiment so that you have the correct data (and enough of it!) to clearly and effectively answer your data science question. This process involves clearly formulating your question in advance of any data collection, designing the best set-up possible to gather the data to answer your question, identifying problems or sources of error in your design, and only then, collecting the appropriate data.

Why should you care?

Going into an analysis, you need to have a plan in advance of what you are going to do and how you are going to analyse the data. If you do the wrong analysis, you can come to the wrong conclusions!

We've seen many examples of this exact scenario play out in the scientific community over the years - there's an entire website, Retraction Watch, dedicated to identifying papers that have been retracted, or removed from the literature, as a result of poor scientific practices. And sometimes, those poor practices are a result of poor experimental design and analysis.

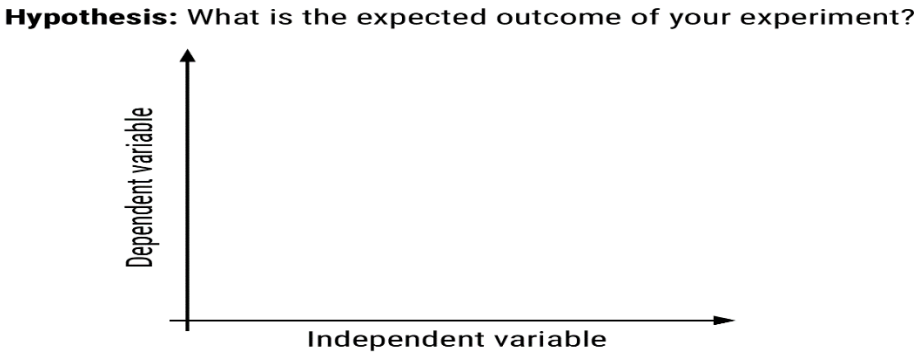
Principles of experimental design

There are a lot of concepts and terms inherent to experimental design. Let's go over some of these now!

Independent variable (AKA factor): The variable that the experimenter manipulates; it does not depend on other variables being measured. Often displayed on the x-axis.

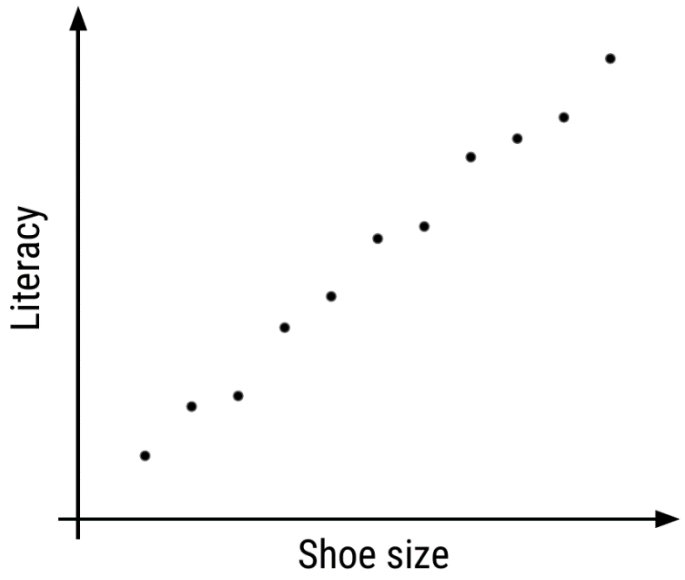
Dependent variable: The variable that is expected to change as a result of changes in the independent variable. Often displayed on the y-axis, so that changes in X, the independent variable, effect changes in Y.

So when you are designing an experiment, you have to decide what variables you will measure, and which you will manipulate to effect changes in other measured variables. Additionally, you must develop your **hypothesis**, essentially an educated guess as to the relationship between your variables and the outcome of your experiment.



Let's do an example experiment now! Let's say for example that I have a hypothesis that as shoe size increases, literacy also increases. In this case, designing my experiment, I would choose a measure of literacy (eg: reading fluency) as my variable that *depends* on an individual's shoe size.

Hypothesis: As shoe size increases, literacy also increases



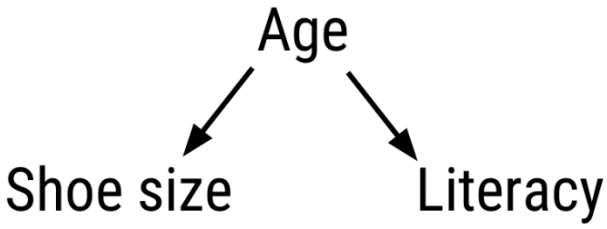
Confounder: An extraneous variable that may affect the relationship between the dependent and independent variables.

To **control** for this, we can make sure we also measure the age of each individual so that we can take into account the effects of age on literacy, as well. Another way we could **control** for age's effect on literacy would be to **fix** the age of all participants. If everyone we study is the same age, then we have removed the possible effect of age on literacy.

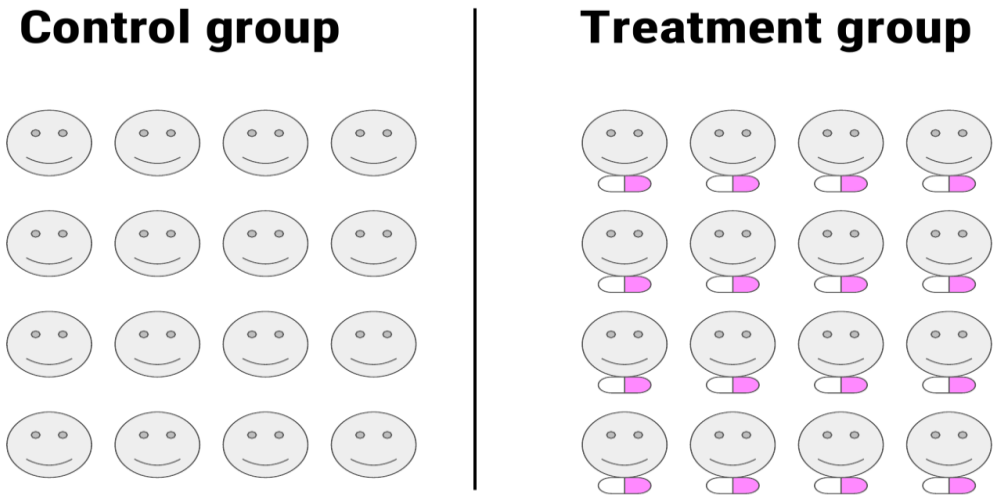
Hypothesis

Shoe size ⟶ Literacy

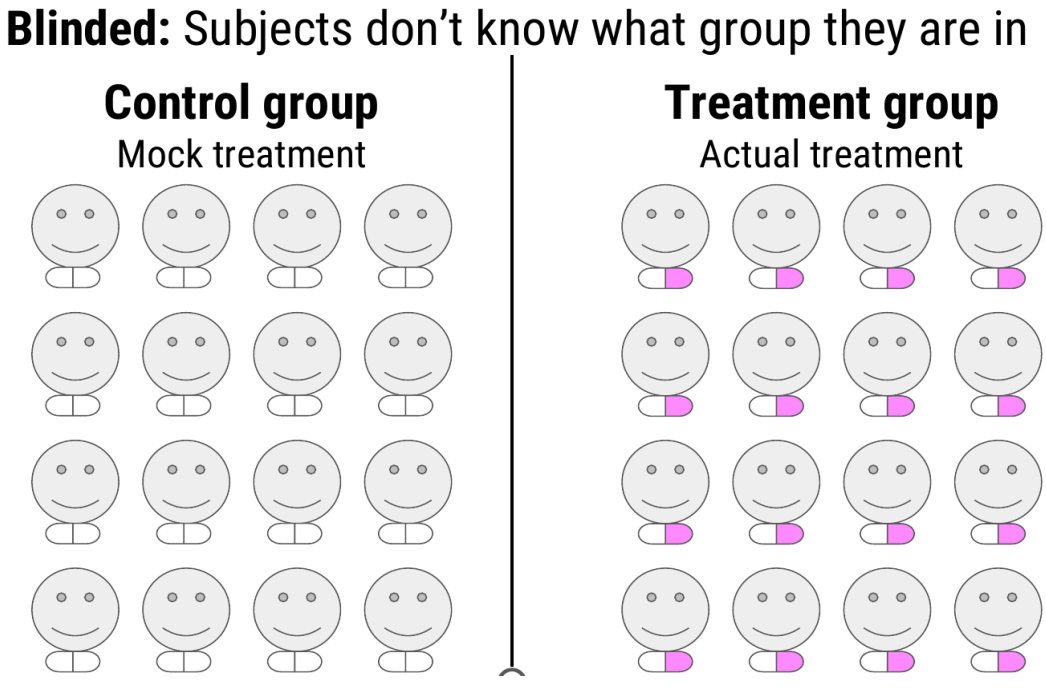
Confounder

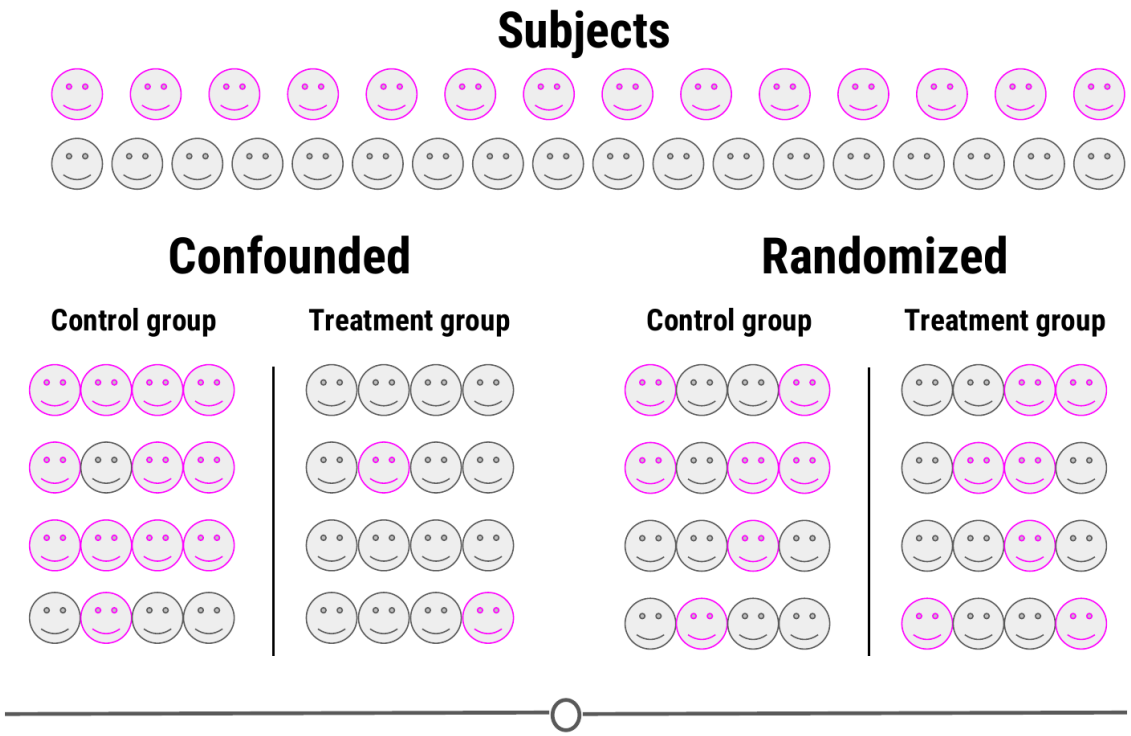


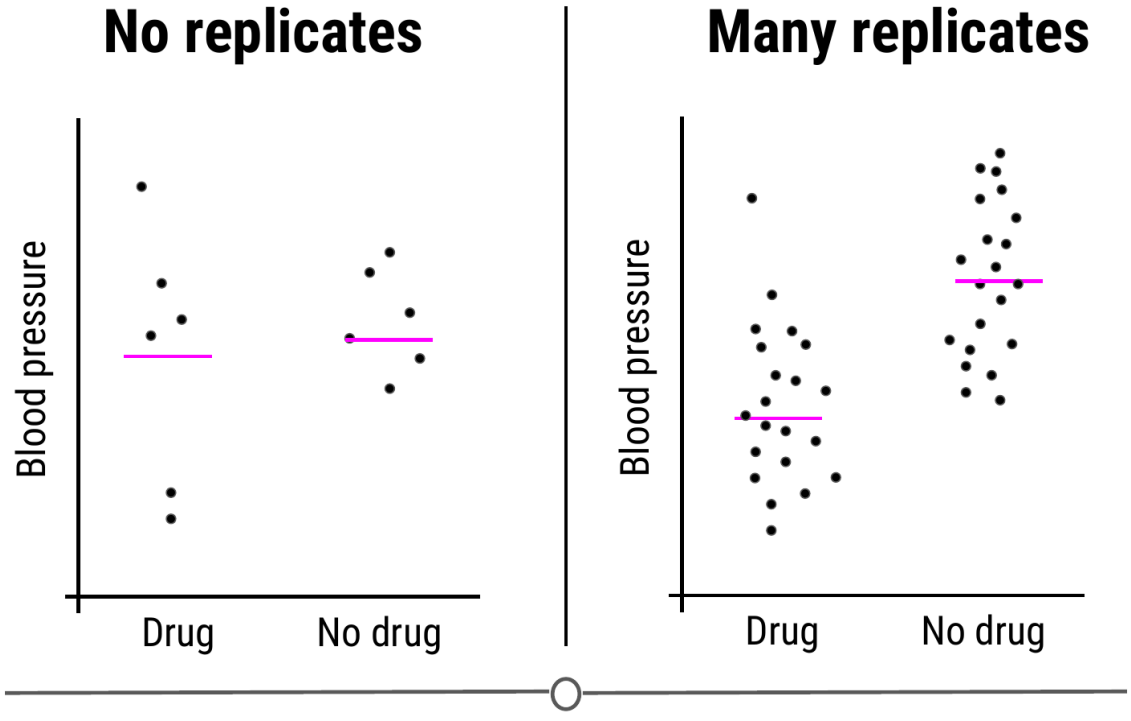
In other experimental design paradigms, a **control group** may be appropriate. This is when you have a group of experimental subjects that are *not* manipulated. So if you were studying the effect of a drug on survival, you would have a group that received the drug (**treatment**) and a group that did not (**control**). This way, you can compare the effects of the drug in the treatment versus control group.



In these study designs, there are other strategies we can use to control for confounding effects. One, we can **blind** the subjects to their assigned treatment group. Sometimes, when a subject knows that they are in the treatment group (eg: receiving the experimental drug), they can feel better, not from the drug itself, but from knowing they are receiving treatment. This is known as the **placebo effect**. To combat this, often participants are blinded to the treatment group they are in; this is usually achieved by giving the control group a mock treatment (eg: given a sugar pill they are told is the drug). In this way, if the placebo effect is causing a problem with your experiment, both groups should experience it equally.




Subjects
Confounded
Control group
Treatment group
Randomized
Control group
Treatment group


No replicates
Blood pressure
Drug
No drug
Many replicates
Blood pressure
Drug
No drug

Sharing data

Once you've collected and analysed your data, one of the next steps of being a good citizen scientist is to share your data and code for analysis. Now that you have a GitHub account and we've shown you how to keep your version controlled data and analyses on GitHub, this is a great place to share your code!

In fact, hosted on GitHub, our group, the Leek group, has developed a guide that has great advice for how to best share data!

Beware p-hacking!

One of the many things often reported in experiments is a value called the **p-value**. This is a value that tells you the probability that the results of your experiment were observed by chance. This is a very important concept in statistics that we won't be covering in depth here.

=x=x=x=x=x=x=x=