# Part 5: Working in RStudio.Cloud

In this chapter, we're going to learn about R packages, navigating the RStudio panes, some additional benefits of working in Rmarkdown and plain text, and how to make and monitor changes with Github.

---

## Navigating the panes in our workbench

The next few sections will walk through a few of RStudio.Cloud's panes. To recap:

*1) We uploaded files into the `project` folder, which now has the following structure:*

```
project/
    ├── CHANGELOG.txt
    ├── README.Rmd
    ├── README.md
    ├── code/
    ├── data/
    ├── dem-pres-debate-2019.Rproj
    ├── figs/
    └── project.Rproj
```

*2) There are three folders in this project: `code/`, `data/`, and `figs/`.*

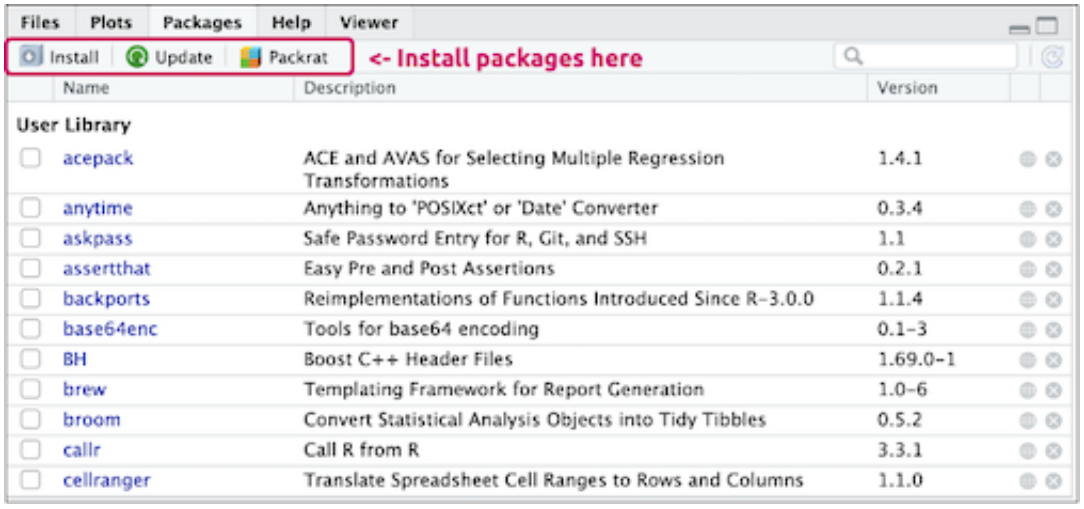
*3) There are three types of plain text files in the parent folder: `.md`, `.Rmd` and `.txt`, and two project files: `dem-pres-debate-2019.Rproj` and `project. Rproj`.*

We'll go over these folders and files in more depth in the following sections.

### Packages

Packages are a vital part of the R ecosystem. R packages are collections of functions and objects collected together with a specific purpose. When we started our RStudio.Cloud session, a few packages get loaded automatically. R users typically refer to these default packages and functions as "base R." Base R packages cover a wide range of statistical procedures and visualizations. Unfortunately, base R can also be challenging to learn because of its inconsistent syntax and style conventions.

A list of the packages and their descriptions are available in the **Packages** pane. We've installed the packages we will use with the `00.1-inst-packages.R` file. If we wanted to, we could also install more packages using *Install* icon, and then enter its name. We don't recommend this, though, because you'll want a record of the packages you used in the project.



User Library

| Name | Description | Version |
|---|---|---|
| acepack | ACE and AVAS for Selecting Multiple Regression Transformations | 1.4.1 |
| anytime | Anything to 'POSIXct' or 'Date' Converter | 0.3.4 |
| askpass | Safe Password Entry for R, Git, and SSH | 1.1 |
| assertthat | Easy Pre and Post Assertions | 0.2.1 |
| backports | Reimplementations of Functions Introduced Since R-3.0.0 | 1.1.4 |
| base64enc | Tools for base64 encoding | 0.1-3 |
| BH | Boost C++ Header Files | 1.69.0-1 |
| brew | Templating Framework for Report Generation | 1.0-6 |
| broom | Convert Statistical Analysis Objects into Tidy Tibbles | 0.5.2 |
| callr | Call R from R | 3.3.1 |
| cellranger | Translate Spreadsheet Cell Ranges to Rows and Columns | 1.1.0 |

You can click on the names of each package to learn more about them and load them into the RStudio session. One of the great things about R is the many user-created packages that greatly expand the number of functions. At the time of this writing, R users have contributed 14638 packages available for us to download and use.

## The `tidyverse`

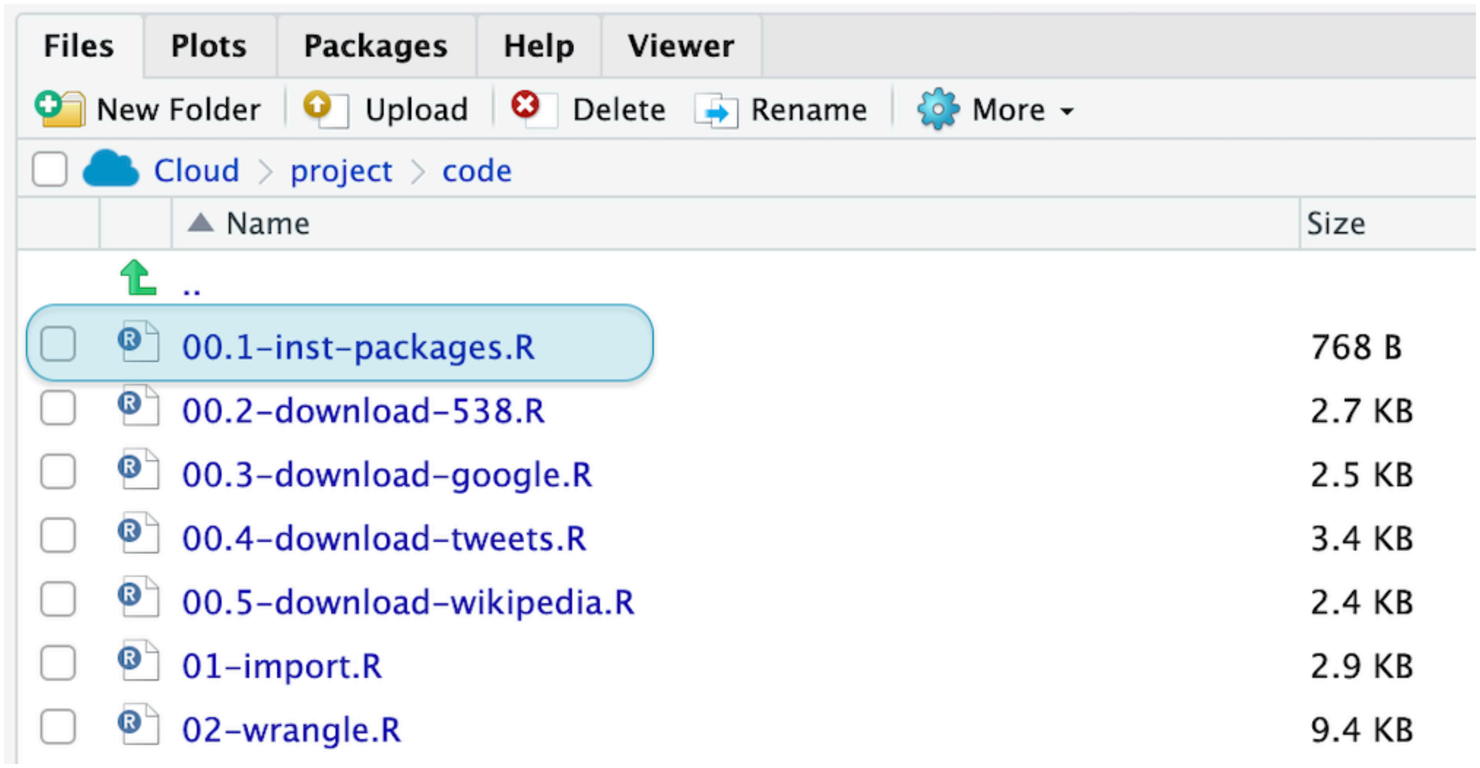
R is open-source software, so users can write packages to expand and enhance its functionality. The tidyverse is a collection of R packages from RStudio for doing data science. All `tidyverse` packages share a few similar underlying principles that allow them to work well together.

Unlike base R, the `tidyverse` also a consistent grammar and syntax, which makes it easier to read and write. You can learn more about this syntax in the R for Data Science text or on the tidyverse webpage.

So far, the code we've run comes from base R. Going forward; we're going to use various packages from the `tidyverse.`

## Files

The **Files** pane displays the files and folders in this project. The file path is visible in the top portion of the pane, beneath the options for *New Folder*, *Upload*, *Delete*, *Rename*, and *More*. An image of this folder's contents is below:



| Name | Size |
|---|---|
| 00.1-inst-packages.R | 768 B |
| 00.2-download-538.R | 2.7 KB |
| 00.3-download-google.R | 2.5 KB |
| 00.4-download-tweets.R | 3.4 KB |
| 00.5-download-wikipedia.R | 2.4 KB |
| 01-import.R | 2.9 KB |
| 02-wrangle.R | 9.4 KB |

The script to install the packages for this project (`00.1-inst-packages.R`) is in the `code` folder. Navigate to this folder either by clicking on the file path (`Cloud/project/code`).

We can see `00.1-inst-packages.R` and other scripts are in the **Files** pane. We now know that because the `.R` files are plain text files, so they have code for the computer to execute, and comments for a human to tread.

In R, we can create comments with a preceding # on any line.

```
# this is a comment
```

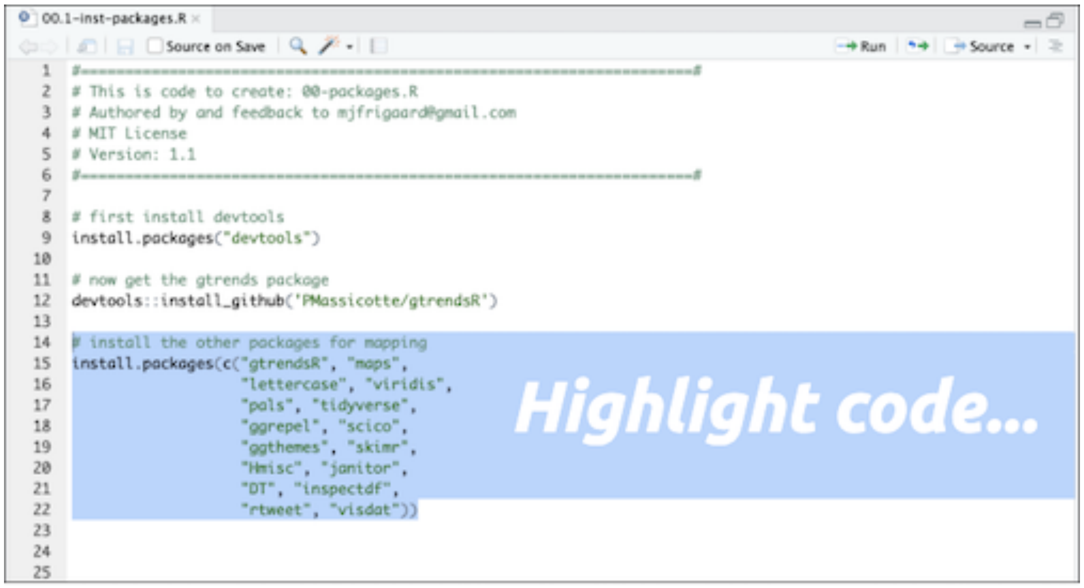
When we click on the `00.1-inst-packages.R` file, we can see it open in the **Source** pane. RStudio.Cloud is giving us a few hints about the packages referenced in the .R file. We're being told we need to install a package before moving forward:

> ***Package devtools required but is not installed.***

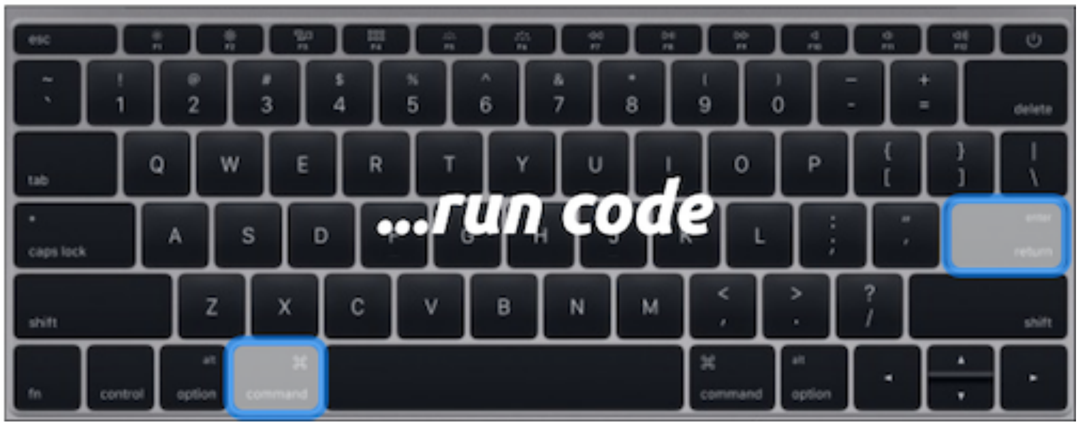
RStudio gives us a choice to ***Install*** or ***Don't Show Again***, and we'll click on the ***Install*** option. Packages vary in the length they take to install, but we'll wait patiently for `devtools` package finish downloading. After the install has finished, you should see the > prompt in the **Console** pane.

## Source

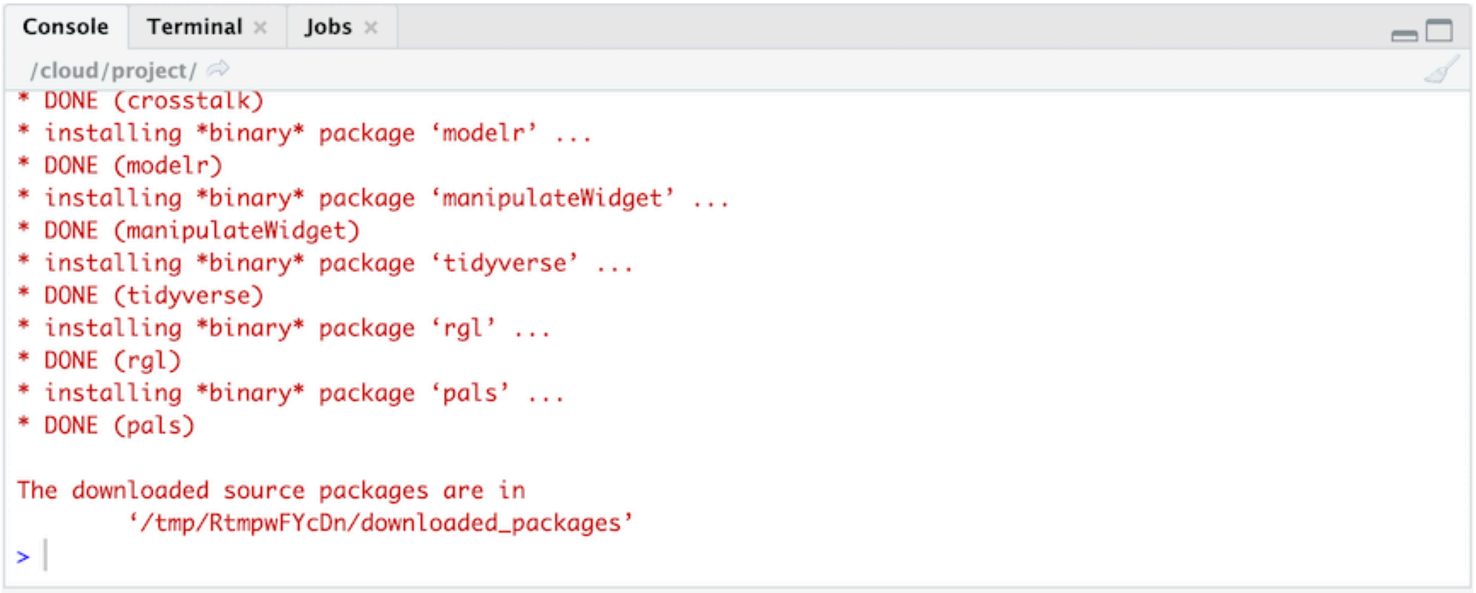
When working in RStudio, sometimes we'll write some code, and want to run that code and see if it works. To do this, we can highlight the rest of the script with the mouse cursor (lines 14–22), then hold down `ctrl` or `cmd` and hit `enter` or `return`.



```
#======================================================================#
# This is code to create: 00-packages.R
# Authored by and feedback to mjfrigaard@gmail.com
# MIT License
# Version: 1.1
#======================================================================#

# first install devtools
install.packages("devtools")

# now get the gtrends package
devtools::install_github('PMassicotte/gtrendsR')

# install the other packages for mapping
install.packages(c("gtrendsR", "maps",
                   "lettercase", "viridis",
                   "pals", "tidyverse",
                   "ggrepel", "scico",
                   "ggthemes", "skimr",
                   "Hmisc", "janitor",
                   "DT", "inspectdf",
                   "rtweet", "visdat"))
```

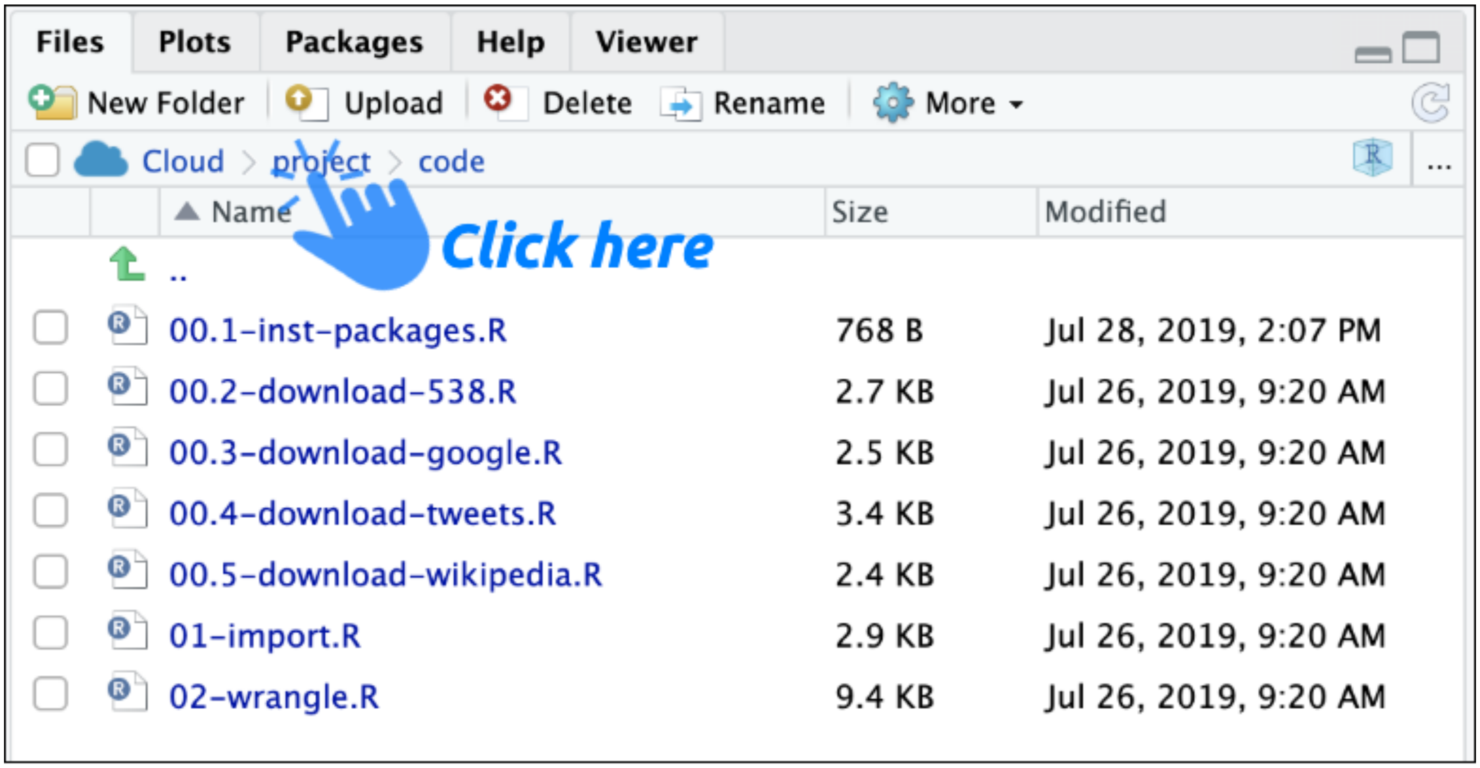


After the installation has completed for all the packages, we'll see the following in the **Console** pane.



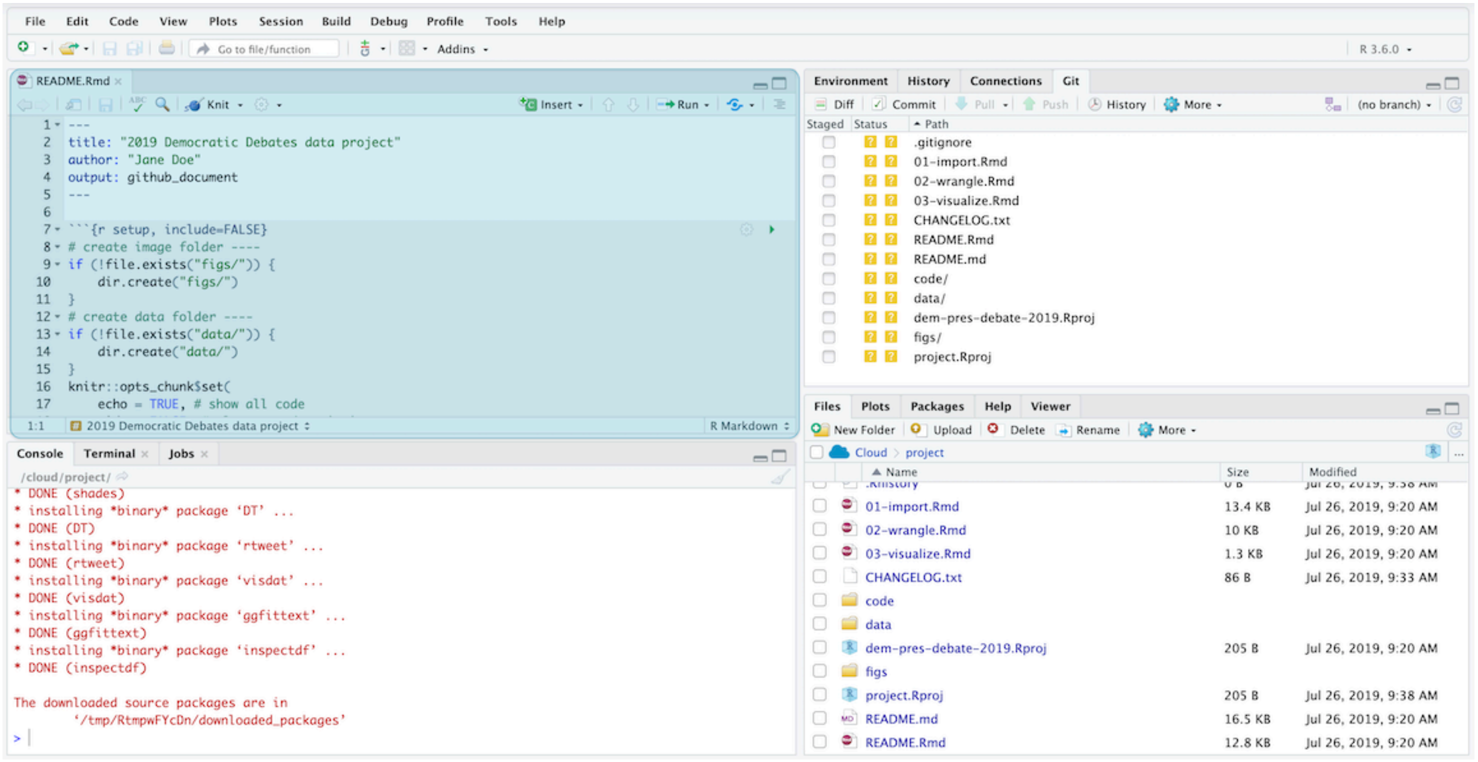
```
* DONE (crosstalk)
* installing *binary* package 'modelr' ...
* DONE (modelr)
* installing *binary* package 'manipulateWidget' ...
* DONE (manipulateWidget)
* installing *binary* package 'tidyverse' ...
* DONE (tidyverse)
* installing *binary* package 'rgl' ...
* DONE (rgl)
* installing *binary* package 'pals' ...
* DONE (pals)

The downloaded source packages are in
        '/tmp/RtmpwFYcDn/downloaded_packages'
>
```

Back in the **Files** pane, we can navigate back to the `project/` folder by clicking on the path above the files.



| Name | Size | Modified |
|---|---|---|
| .. | | |
| 00.1-inst-packages.R | 768 B | Jul 28, 2019, 2:07 PM |
| 00.2-download-538.R | 2.7 KB | Jul 26, 2019, 9:20 AM |
| 00.3-download-google.R | 2.5 KB | Jul 26, 2019, 9:20 AM |
| 00.4-download-tweets.R | 3.4 KB | Jul 26, 2019, 9:20 AM |
| 00.5-download-wikipedia.R | 2.4 KB | Jul 26, 2019, 9:20 AM |
| 01-import.R | 2.9 KB | Jul 26, 2019, 9:20 AM |
| 02-wrangle.R | 9.4 KB | Jul 26, 2019, 9:20 AM |

We can see the `README.Rmd` file in the **Files** pane in the lower right corner. We'll click on it, so the file opens in the **Source** pane (see image below).



~~~
---
title: "2019 Democratic Debates data project"
author: "Jane Doe"
output: github_document
---

```{r setup, include=FALSE}
# create image folder ----
if (!file.exists("figs/")) {
    dir.create("figs/")
}
# create data folder ----
if (!file.exists("data/")) {
    dir.create("data/")
}
knitr::opts_chunk$set(
    echo = TRUE, # show all code
~~~

```
* DONE (shades)
* installing *binary* package 'DT' ...
* DONE (DT)
* installing *binary* package 'rtweet' ...
* DONE (rtweet)
* installing *binary* package 'visdat' ...
* DONE (visdat)
* installing *binary* package 'ggfittext' ...
* DONE (ggfittext)
* installing *binary* package 'inspectdf' ...
* DONE (inspectdf)

The downloaded source packages are in
        '/tmp/RtmpwFYcDn/downloaded_packages'
>
```

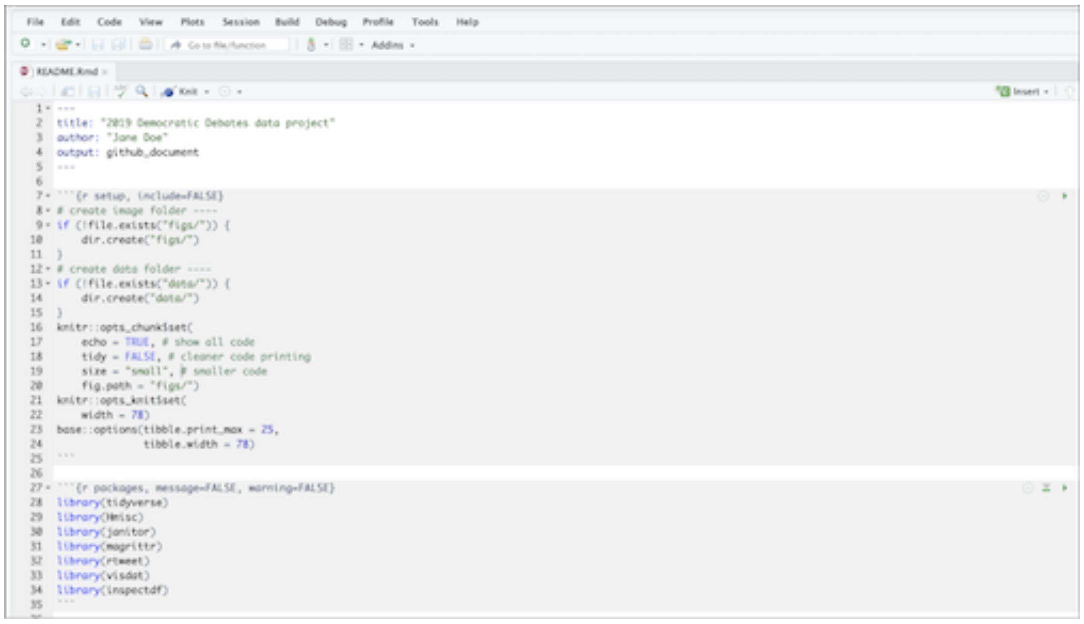
| Staged | Status | Path |
|---|---|---|
| ☐ | ? ? | .gitignore |
| ☐ | ? ? | 01-import.Rmd |
| ☐ | ? ? | 02-wrangle.Rmd |
| ☐ | ? ? | 03-visualize.Rmd |
| ☐ | ? ? | CHANGELOG.txt |
| ☐ | ? ? | README.Rmd |
| ☐ | ? ? | README.md |
| ☐ | ? ? | code/ |
| ☐ | ? ? | data/ |
| ☐ | ? ? | dem-pres-debate-2019.Rproj |
| ☐ | ? ? | figs/ |
| ☐ | ? ? | project.Rproj |

| Name | Size | Modified |
|---|---|---|
| .Rhistory | 0 B | Jul 26, 2019, 9:38 AM |
| 01-import.Rmd | 13.4 KB | Jul 26, 2019, 9:20 AM |
| 02-wrangle.Rmd | 10 KB | Jul 26, 2019, 9:20 AM |
| 03-visualize.Rmd | 1.3 KB | Jul 26, 2019, 9:20 AM |
| CHANGELOG.txt | 86 B | Jul 26, 2019, 9:33 AM |
| code | | |
| data | | |
| dem-pres-debate-2019.Rproj | 205 B | Jul 26, 2019, 9:20 AM |
| figs | | |
| project.Rproj | 205 B | Jul 26, 2019, 9:38 AM |
| README.md | 16.5 KB | Jul 26, 2019, 9:20 AM |
| README.Rmd | 12.8 KB | Jul 26, 2019, 9:20 AM |

We've highlighted the pane of RStudio that's displaying the `README.Rmd` file. As you can see, this is a relatively small area in the IDE. Working in a tiny corner of the screen can be hard, but fortunately, RStudio gives us the ability to expand any pane into a fullscreen view. In this case, we'll zoom in the `README.Rmd` file.

If we want to focus on the **Source** pane, and can zoom in using `shift+control+1`.

As soon as you click `1`, your screen should expand to look like the image below:



~~~
---
title: "2019 Democratic Debates data project"
author: "Jane Doe"
output: github_document
---

```{r setup, include=FALSE}
# create image folder ----
if (!file.exists("figs/")) {
    dir.create("figs/")
}
# create data folder ----
if (!file.exists("data/")) {
    dir.create("data/")
}
knitr::opts_chunk$set(
    echo = TRUE, # show all code
    tidy = FALSE, # cleaner code printing
    size = "small", # smaller code
    fig.path = "figs/")
knitr::opts_knit$set(
    width = 78)
base::options(tibble.print_max = 25,
              tibble.width = 78)
~~~

```{r packages, message=FALSE, warning=FALSE}
library(tidyverse)
library(Hmisc)
library(janitor)
library(magrittr)
library(rtweet)
library(visdat)
library(inspectdf)
```
```

Since the **Source** pane is where we'll write most of our code, it's also the pane we are spending most of our time. We've seen that's where our files open, so being able to focus on the area quickly is helpful.

## Console

The **Console** is probably the second most used area in RStudio (it displays most of the output), so we also should know how to zoom in on this pane. To focus on this pane, we'll use `shift+control+2`.
```

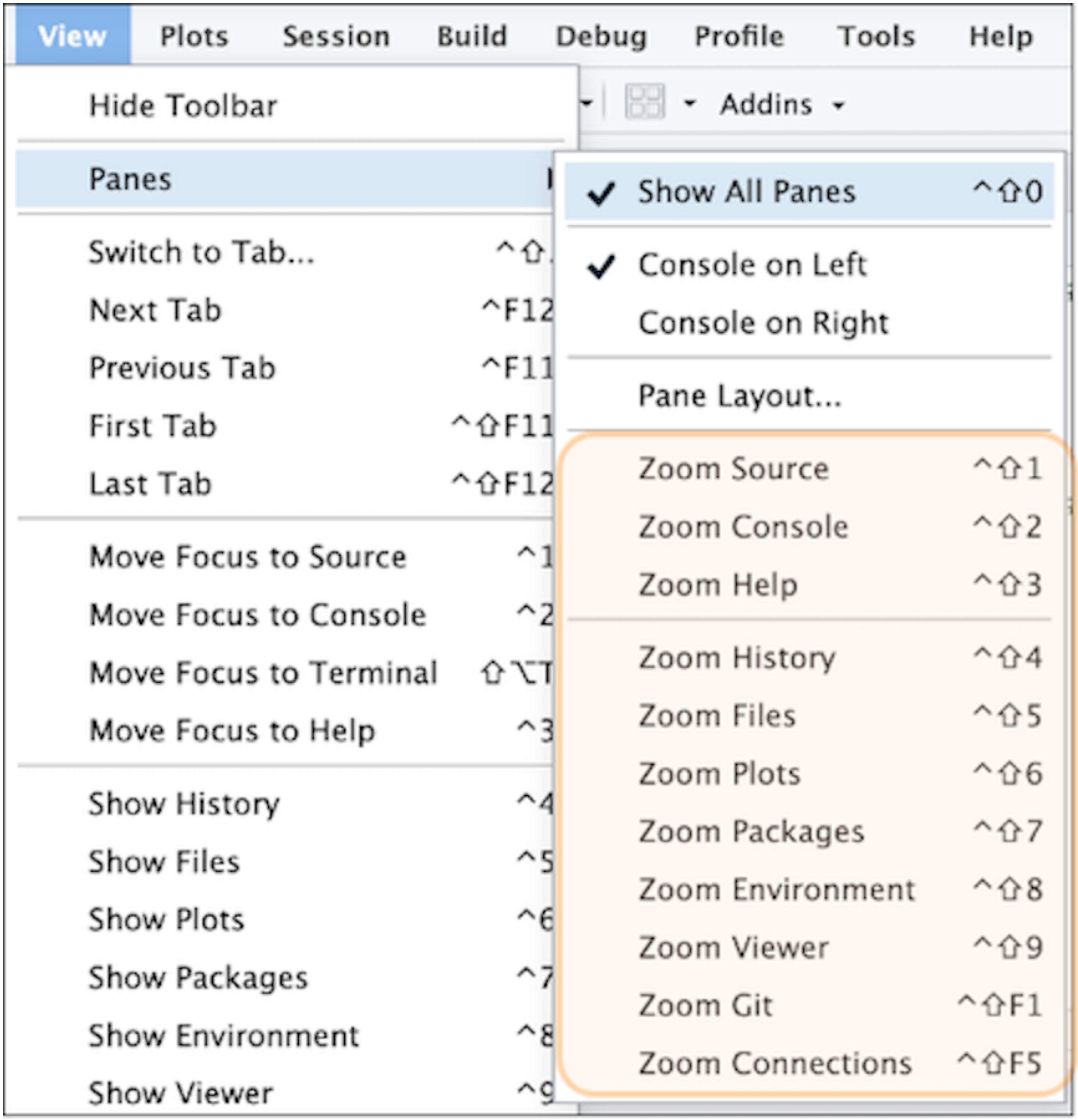
Zoom Console Pane

What if we accidentally zoom into the wrong pane? We can easily resize the IDE to its original position by holding down the `shift+control` buttons, then clicking either number again. Combining those keys should return the IDE to its default arrangement.

## Working in one pane

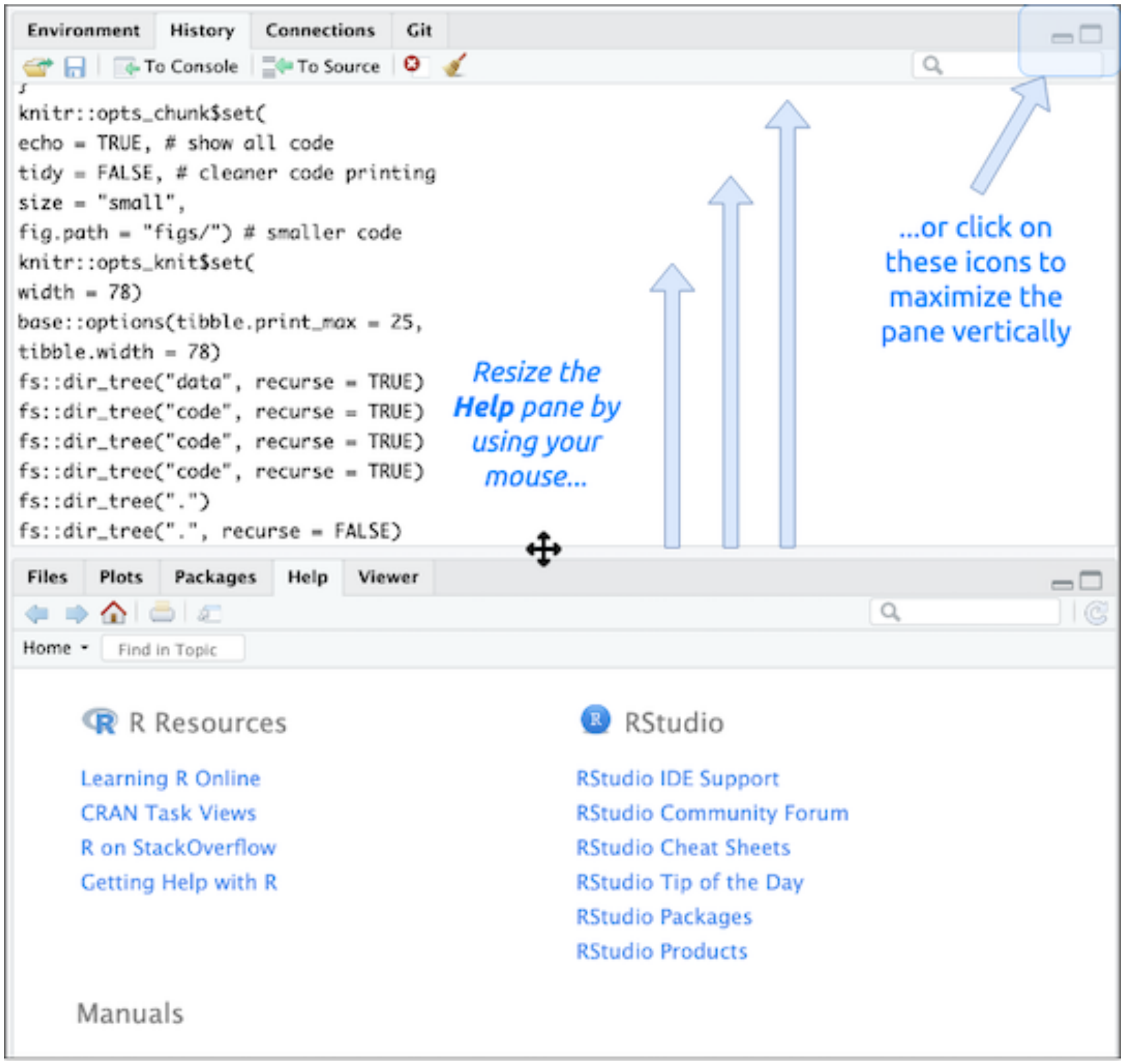
We can move through every pane in RStudio by holding down `shift+control` and clicking on numbers `1-9`. Go ahead and do that now.

If you ever forget which number corresponds to which pane, you can always find them under *View* > *Panes* (see image below)



## Help

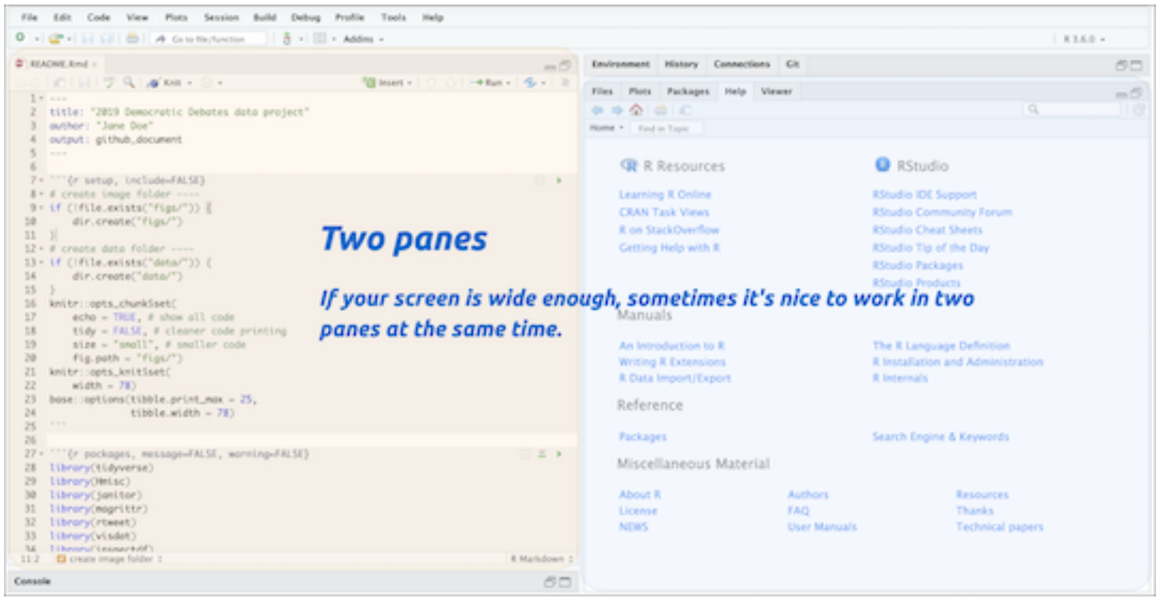
Inevitably, we'll write something that doesn't work. When things aren't working (try to remain calm), you're going to want a place to start looking for solutions. The **Help** pane is accessible in the lower right corner of the IDE.



```
knitr::opts_chunk$set(
echo = TRUE, # show all code
tidy = FALSE, # cleaner code printing
size = "small",
fig.path = "figs/") # smaller code
knitr::opts_knit$set(
width = 78)
base::options(tibble.print_max = 25,
tibble.width = 78)
fs::dir_tree("data", recurse = TRUE)
fs::dir_tree("code", recurse = TRUE)
fs::dir_tree("code", recurse = TRUE)
fs::dir_tree("code", recurse = TRUE)
fs::dir_tree(".")
fs::dir_tree(".", recurse = FALSE)
```

## Working in two panes

Sometimes we'll want to do our work in more than one pane at a time. For example, what if we're working in the **Source** pane, but have a question about a function? We can get answers to a lot of problems using RStudio's internal **Help** pane. After resizing the **Help** pane, we should see the following layout in our IDE.



~~~
---
title: "2019 Democratic Debates data project"
author: "Jane Doe"
output: github_document
---

```{r setup, include=FALSE}
# create image folder ----
if (!file.exists("figs/")) {
    dir.create("figs/")
}
# create data folder ----
if (!file.exists("data/")) {
    dir.create("data/")
}
knitr::opts_chunk$set(
    echo = TRUE, # show all code
    tidy = FALSE, # cleaner code printing
    size = "small", # smaller code
    fig.path = "figs/")
knitr::opts_knit$set(
    width = 78)
base::options(tibble.print_max = 25,
              tibble.width = 78)
~~~

```{r packages, message=FALSE, warning=FALSE}
library(tidyverse)
library(Hmisc)
library(janitor)
library(magrittr)
library(rtweet)
library(visdat)
```

**Two panes**

*If your screen is wide enough, sometimes it's nice to work in two panes at the same time.*

## Recap on RStudio panes

As we pointed out earlier, RStudio is like a workbench built around different panes. Each pane serves a specific purpose and being able to move between them allow us to work quickly and efficiently.

**Keyboard shortcuts are time-savers**. Not having to switch from keyboard to mouse over and over again keeps us focused on the task at hand.

---

## Where do we write code?

You might be wondering why we have provided you with both `.R` scripts and `.Rmd` files. Well, we want to show you a few options for documenting your work in RStudio. The two sections below outline two standard options (but these are by no means the only way to work with these tools!)

### Option 1) everything goes in scripts

`rmarkdown` is a relatively new package, so early `R` users didn't have an option to put everything in `.Rmd` files (we fit into this group). We have found this approach is excellent when you're reasonably confident about the project. If you're familiar with the data files, wrangling techniques, visualizations, and products, you can quickly outline the scripts and predetermine their names and contents.

For example, the folder tree of our code folder tells us what we should expect from this project.

```
code
    ├── 00.1-inst-packages.R
    ├── 00.2-download-538.R
    ├── 00.3-download-google.R
    ├── 00.4-download-tweets.R
    ├── 00.5-download-wikipedia.R
    ├── 01-import.R
```

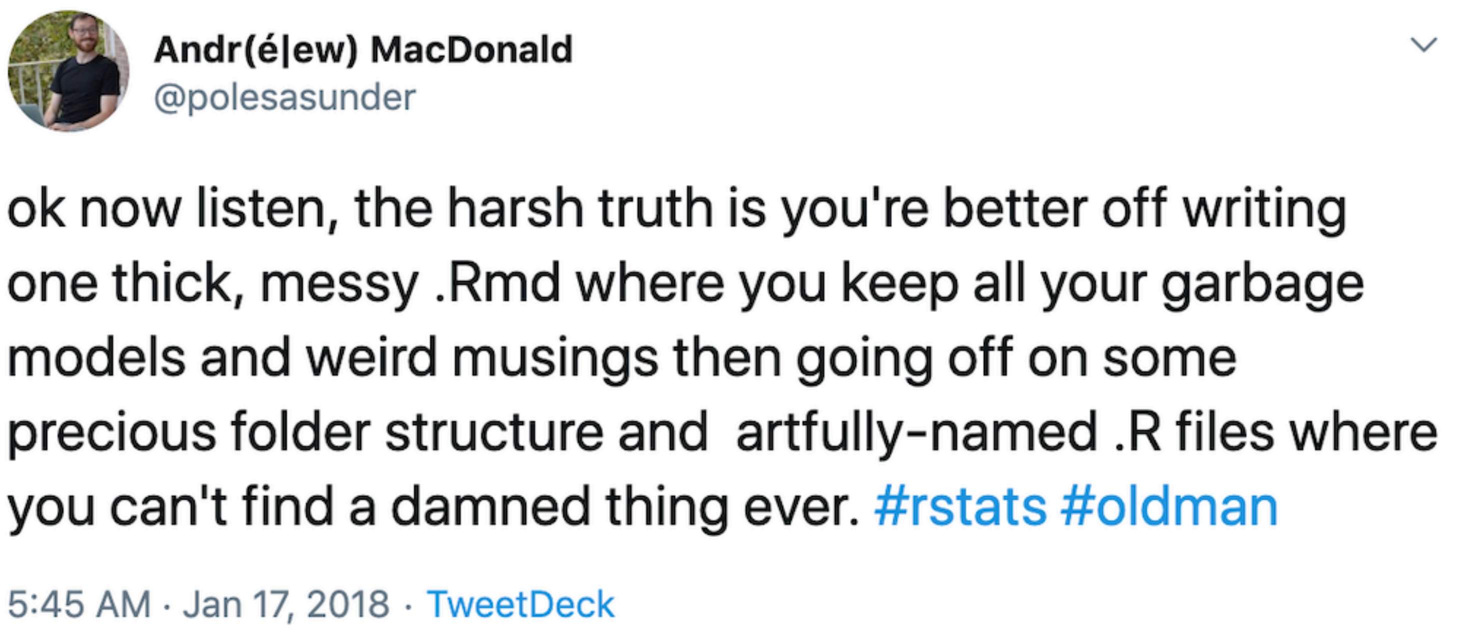
```
    └── 02-wrangle.R
```

We can see there are seven script files, each one with a numerical prefix. These prefixes tell us the order to run the code files, too. The file names also tell us what each script does (`download`, `import`, `wrangle`, etc.).

The layout above is an example of how to organize a set of `.R` scripts that follow the guidelines mentioned in previous chapters. These naming conventions make it easier for newcomers to the project to get oriented to its contents.

## Option 2) everything goes in the Rmarkdown

The second option is well illustrated in the tweet below by Andrew MacDonald:



Andr(é|ew) MacDonald
@polesasunder

ok now listen, the harsh truth is you're better off writing one thick, messy .Rmd where you keep all your garbage models and weird musings then going off on some precious folder structure and  artfully-named .R files where you can't find a damned thing ever. #rstats #oldman

5:45 AM · Jan 17, 2018 · TweetDeck

*https://twitter.com/polesasunder/status/953624238266646529?s=20*

In our experience, this accurately captures the reality of data projects (especially in the beginning stages). Throwing everything into the Rmarkdown file gives us a lot more flexibility by allowing us to add multiple types of content.

All that being said, it's always good practice to come back and revise the `README.Rmd` document, so it's contents outline all steps in the analysis thoroughly. We consider documentation to be a form of communication with our future selves, and this typically involves clearly describing each step. During the revision, we might also break down each of the chunks into individual scripts. We can also add more details to the text portions, with links to external content, etc.

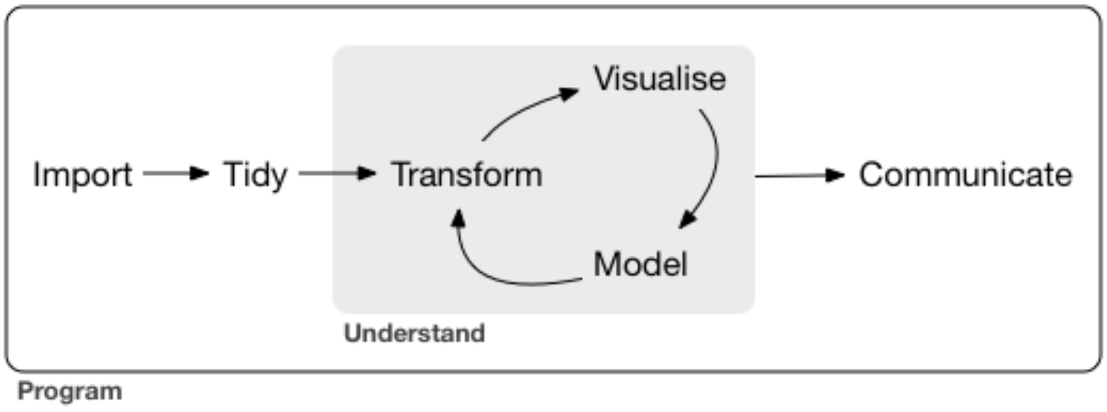
We've found each project usually starts at a bit of a sprint, so we try to capture as much code and content in `.Rmd` files early. We can focus on trimming it down later, but the flexibility of markdown combined with functional code is better than limiting ourselves to code and comments `.R` script files.

---

# Rmarkdown

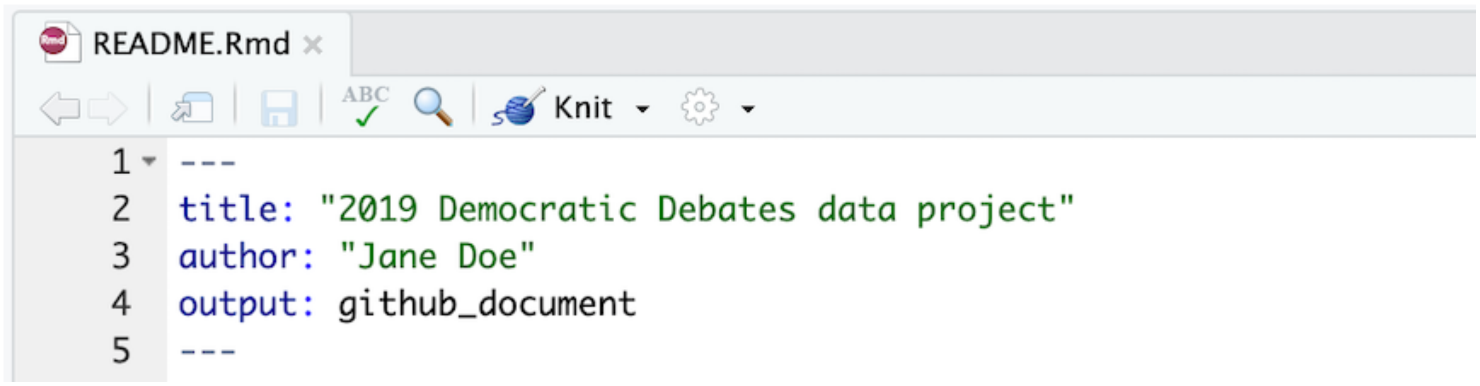
We're going to move forward assuming we're documenting everything in a `README.Rmd` file. In the next few sections, we'll add various components to the `README.Rmd` we've placed in the project folder.

If you're ever wondering how to outline your `README.Rmd` file, the figure from R for Data Science isn't a wrong place to start.



## .Rmd step 1: the YAML header

At the top of the `README.Rmd` document, the first thing we see is what's called the "`YAML` header", and it's going to tell RStudio.Cloud the file's `title`, the `author`, and what the `output` file will be.



```
---
title: "2019 Democratic Debates data project"
author: "Jane Doe"
output: github_document
---
```

The `YAML` header always goes at the top of the `README.Rmd` file, between two sets of three dashes:

```
---
title: "2019 Democratic Debates data project"
author: "Jane Doe"
output: github_document
---
```

People typically use `YAML` in configuration files, which makes it perfect for setting some default options in our `README.Rmd` document. Read more about YAML headers in the RMarkdown book and on the YAML website. `YAML` actually stands for "YAML Ain't Markup Language."

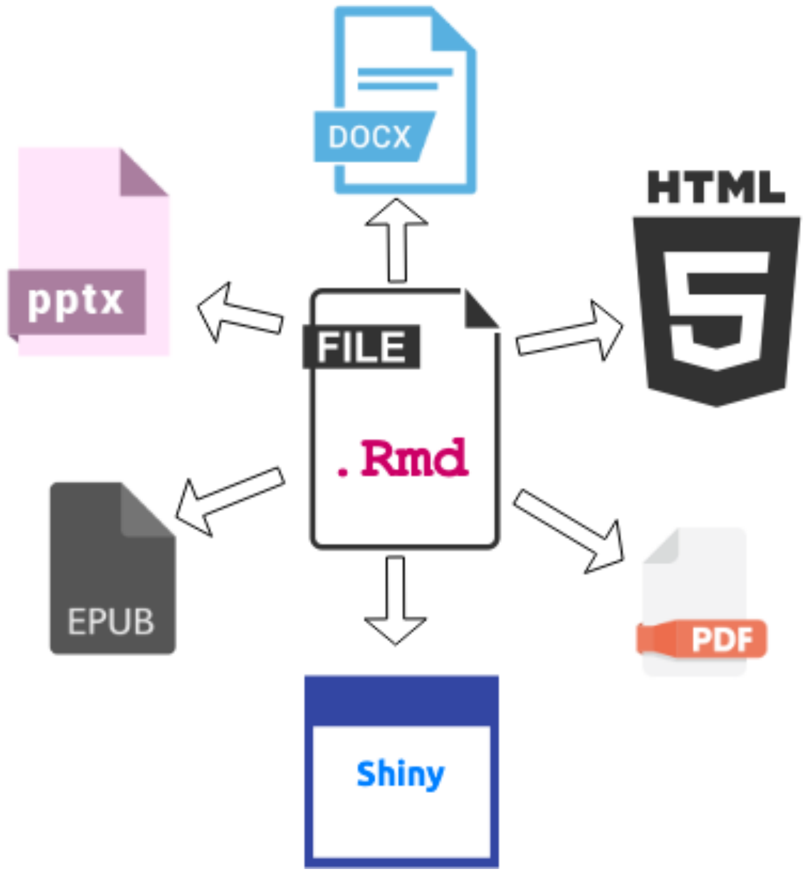
We can change the `output` argument to `html_document`, `word_document`, or `pdf_document` and create a different file from the plain text we are going to be working in. For now, we are going to focus on the `github_document` output.

## .Rmd step 2: Knit output options

Another benefit of working in the `README.Rmd` document is that when we combine it with the powerful `knitr` package, we drastically extend the kind of files we can produce from our analysis. `Knitr` follows a principle of literate programming put forth by Donald E. Knuth.

> *"Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do."*
> *- Donald E. Knuth, Literate Programming, 1984*

The **Knit** button will render the plain text `.Rmd` document into a variety of different output options.



The `.Rmd` files also give us the ability to document our intentions, write and execute code, and interpret and explain the results. After we've outlined (and revised) our thought process, we can go about organizing the code in more efficient ways to carry out our intentions.

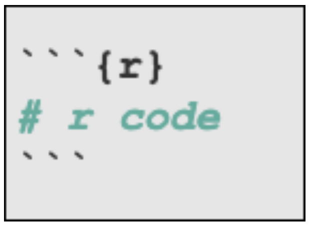
## Functional code chunks

We like to think of the `.Rmd` document like a stack of two sheets of paper, and each piece representing a different file type (`.R` and `.md`). When you lay the markdown file on top of the .R script, you get an Rmarkdown file.

Rmarkdown
Lorem ipsum dolor sit amet, consectetur adipisicing
Lorem ipsum dolor sit amet, consectetur adipisicing

*R code chunk*

This combination of files gives us a syntax for formatting our text (stuff like *italic*, **bold**, `code`, etc.), and a functional code script we can get access to by inserting code chunks. For example, in the `README.Rmd` file we can type directly onto the paper using the markdown syntax. But if we want to run some R code, imagine tearing a little hole in the markdown paper, and revealing an R script underneath.



~~~
```{r}
# r code
```
~~~

*R code chunk*

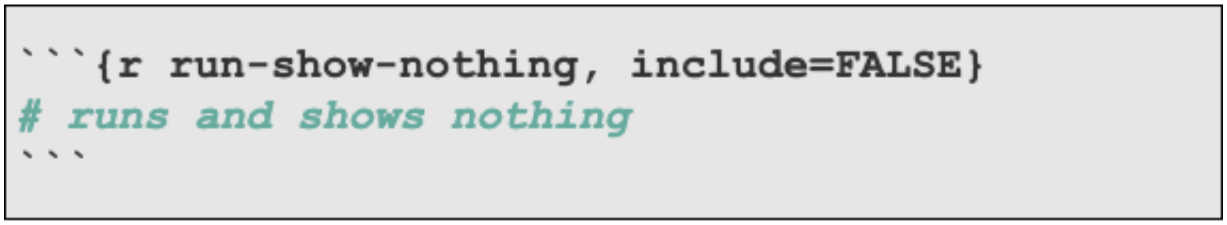
These code chunks allow us to run R code in-between the markdown text.

## R Code chunk options & labels

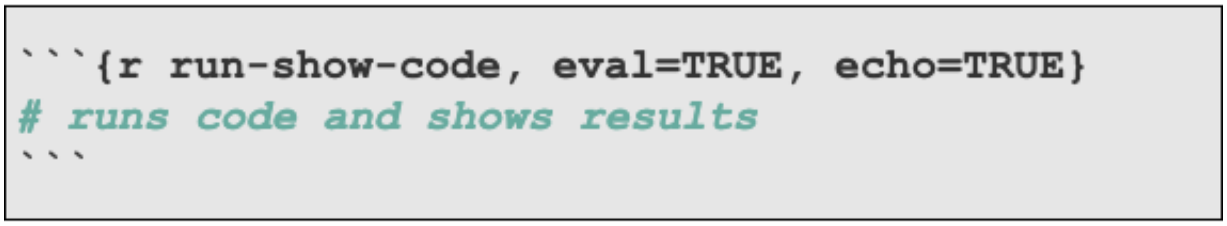
Code chunks come with a long list of options (feel free to experiment with all possible combinations found here and here). The most common are `echo`, `eval`, and `include`.

- `echo` = *show the code in the output?*
- `eval` = *run the code chunk?*
- `include` = *put the results from the code in the output?*

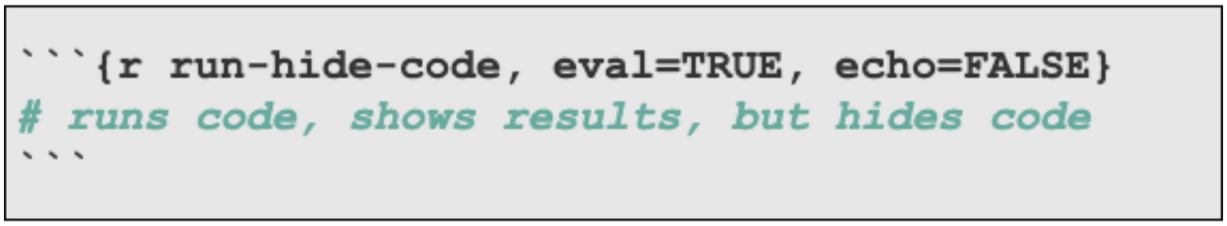
We will show a few examples of these with labels below:



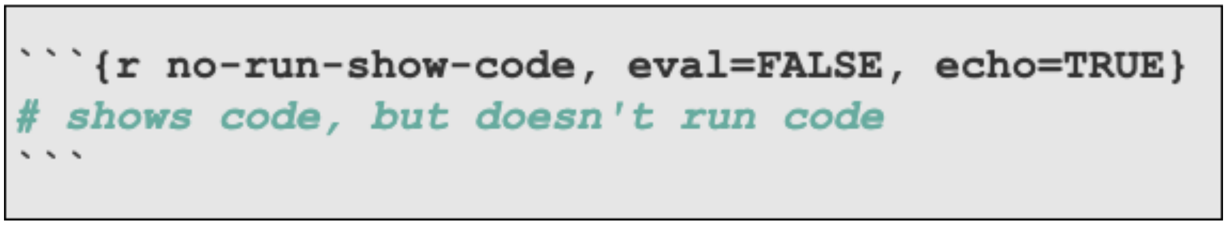
~~~
```{r run-show-nothing, include=FALSE}
# runs and shows nothing
```
~~~



~~~
```{r run-show-code, eval=TRUE, echo=TRUE}
# runs code and shows results
```
~~~



~~~
```{r run-hide-code, eval=TRUE, echo=FALSE}
# runs code, shows results, but hides code
```
~~~



~~~
```{r no-run-show-code, eval=FALSE, echo=TRUE}
# shows code, but doesn't run code
```
~~~

The code chunks above are labeled, and we recommend *labeling all code chunks*. We've found it forces us to think through the analysis as a series of operations, and reduces random calculations spread throughout the document.

## The setup code chunk

After the `YAML` header, our `setup` chunk tells us what we're going to be doing with the code, text, figures, and output in this `README.Rmd` file.

Our `setup` chunk does the following:

**Project folders:** We check to see if the project has image or data folders, and if not, it creates them.

```
# create image folder ----
if (!file.exists("figs/")) {
    dir.create("figs/")
}
# create data folder ----
if (!file.exists("data/")) {
    dir.create("data/")
}
```

**Chunk options:** We've included `echo=TRUE`, which means the code will print in the output file. The `tidy=FALSE` makes sure the code doesn't get reformatted when the document gets rendered. We set both options to their default values, but we've included them below so that you can see more examples.

The `size = "small"` and `fig.path = "figs/"` are used to change the size of the printed code, and the location of any output visualizations.

```
knitr::opts_chunk$set(
    echo = TRUE, # show all code
    tidy = FALSE, # cleaner code printing
    size = "small", # smaller code
    fig.path = "figs/") # where the figures will end up
```

**Knit options:** The `knitr::opts_knit$set()` function gives us the ability to stipulate options for what happens when we render the document (knitted and rendered are somewhat synonymous).

```
knitr::opts_knit$set(
    width = 78)
```

**Base options:** the final setting tells RStudio.Cloud how we want the tables to print (`25` rows, and `78` columns).

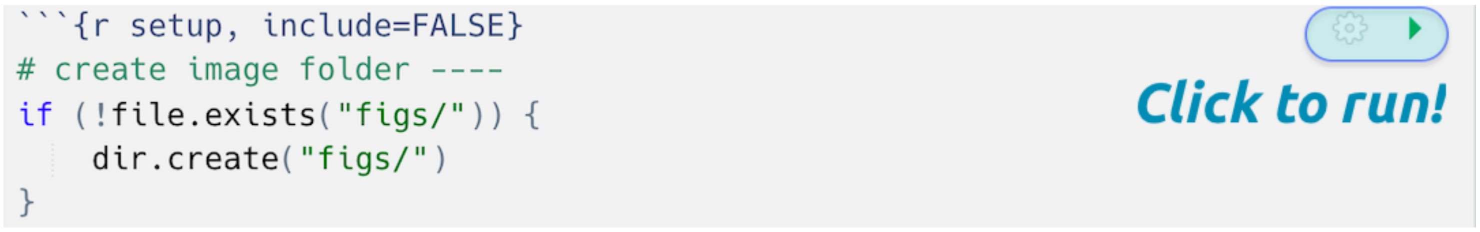
```
base::options(tibble.print_max = 25,
              tibble.width = 78)
```

Why `78`? The standard code file is 80 columns across, which is the length of a punch card (read more here). We go two columns less than 80 ( to give a little wiggle room).

## Running code chunks

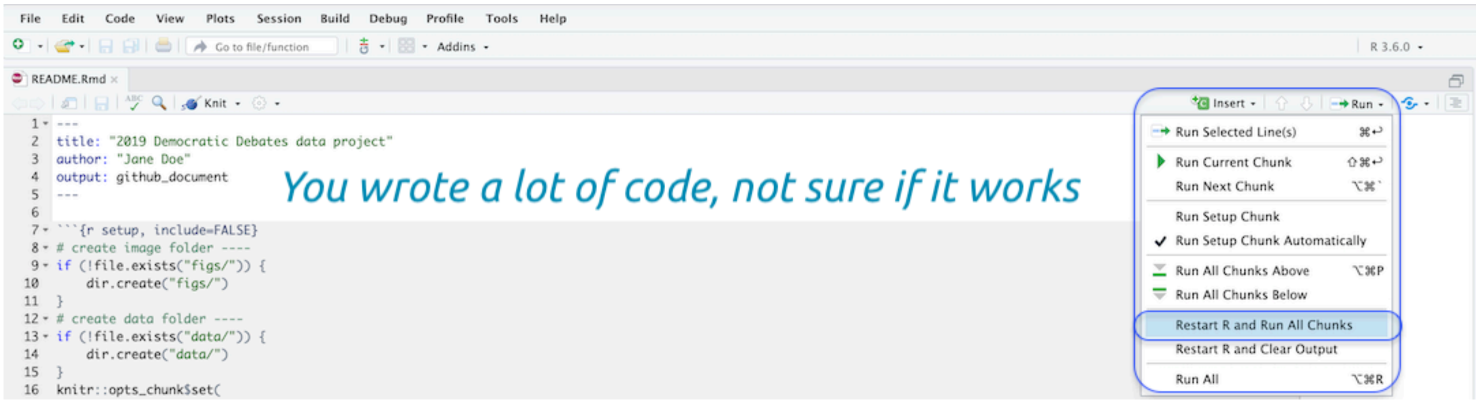
To run the code in your document, we have a few options. First, we can use the same keyboard shortcuts we use for executing code in the `.R` scripts (`ctrl+enter` or `cmd+return`).

The second option is the small green play icon on the far right-hand side of the code chunk. Running code one chunk at a time is a great way to test what the output of a graph looks like, or to see the data table from a join.



~~~
```{r setup, include=FALSE}
# create image folder ----
if (!file.exists("figs/")) {
    dir.create("figs/")
}
~~~

The third option is we run a series of chunks (or the entire document) using the drop-downs at the top of the `.Rmd` file. We use this option if we've added a few chunks, and we want to make sure everything works together.


You wrote a lot of code, not sure if it works
Insert
Run
Run Selected Line(s)
Run Current Chunk
Run Next Chunk
Run Setup Chunk
Run Setup Chunk Automatically
Run All Chunks Above
Run All Chunks Below
Restart R and Run All Chunks
Restart R and Clear Output
Run All

~~~
---
title: "2019 Democratic Debates data project"
author: "Jane Doe"
output: github_document
---

```{r setup, include=FALSE}
# create image folder ----
if (!file.exists("figs/")) {
    dir.create("figs/")
}
# create data folder ----
if (!file.exists("data/")) {
    dir.create("data/")
}
knitr::opts_chunk$set(
~~~

Don't worry if all of this is confusing! We will be creating our very own `README.Rmd` for this project, so you'll get some practice!

---

## Start with a README.Rmd

The `README.Rmd` file we've provided for this project is in the **Source** pane, and we can start by looking at the contents of the file (look at the `YAML` header, `setup` chunk, and `packages`).

As we've stated before, we won't be going through the code collects and downloads the data for this project. We'll provide links to resources where you can get all that information, bu we're going to focus on the *workflow for using data to create tables, images, and products*.

Hit the enter/return key until we a few lines of padding beneath the `packages` code chunk. Our cursor should be around line 37.

We will begin this document with the following text under our first line header (#).

```
# Motivation

We read an interesting article on `fivethirtyeight` about the democratic debates in June of 2019. In p
articular, the image below displays how voters had changed their minds after watching the candidates.
```

As the text above states, we saw something cool, and we're going to see if we can recreate or verify some of the information. The source of this inspiration comes from this fivethirtyeight article.

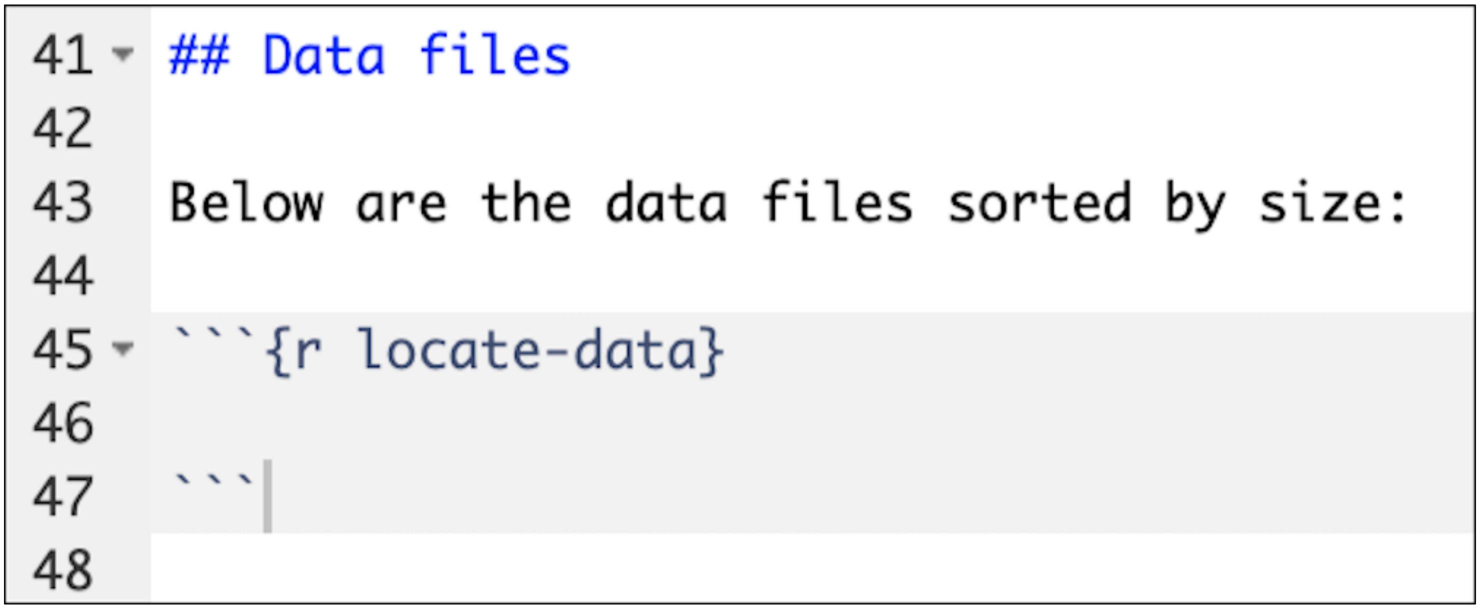
> ***Including an image in the .Rmd file:*** *If you want to include an image in your `README.Rmd` file, read about the `knitr::include_graphics()` function by entering `??include_graphics` in the **Console** pane.*

## Data sources

Now that we have an idea why we're here, we want to start exploring the data. We're going to document all of our steps in the `README.Rmd` file.

We will start by importing all of the data for this project. We've put the raw data files in the `data/raw/` folder. **Always leave raw data in its own folder, and never alter the raw data files (read more here).**

In our `README.Rmd` file, we'll create a new level-two header (##) called 'Data files' and a code chunk labeled "`locate-data`" (it should look like the image below).



~~~
## Data files

Below are the data files sorted by size:

```{r locate-data}

```

~~~

### File management with the `fs` package

To locate the files in `data/raw/` folder, we like to use the `fs` package. `fs` stands for 'file system', and this package gives us the ability to navigate our project folders and files.

Check out the package website here for a full description and examples. `fs` is loaded as part of the `tidyverse`.

Start with a folder tree of the data folder.

```r
# where are the data?
fs::dir_tree("data")
# data
# ├── processed
# └── raw
#     ├── 538
#     │   └── 2019-07-06-Cand538Fav.csv
#     ├── google-trends
#     │   ├── 2019-07-10-Dems2020Night1Group1.rds
#     │   └── 2019-07-10-Dems2020Night1Group2.rds
#     ├── twitter
#     │   ├── 2019-07-06-Night01Tweets.rds
#     │   ├── 2019-07-06-Night01TweetsRaw.rds
#     │   ├── 2019-07-06-Night01TweetsUsers.rds
#     │   ├── 2019-07-06-Night02Tweets.rds
#     │   ├── 2019-07-06-Night02TweetsRaw.rds
#     │   └── 2019-07-06-Night02TweetsUsers.rds
#     └── wikipedia
#         ├── 2019-07-10-WikiDemAirTime01Raw.csv
#         ├── 2019-07-10-WikiDemAirTime02Raw.csv
#         └── 2019-07-25-PollingCriterionRaw.csv
```

The raw data are in four separate folders, each representing the data source (`538`, `google-trends`, `twitter`, `Wikipedia`).

## Determine the size of the data files

Size can be a significant impediment to getting your work done quickly, so it's best to determine the size of the raw data files before importing.

The code chunk below tells us how big each file is and the folder in which its located.

```
# how big are each of these files?
fs::dir_info(path = "data", recurse = TRUE) %>%
    # only files
    filter(type == "file") %>%
    group_by(path) %>%
    # sort by size
    tally(wt = size, sort = TRUE)
# A tibble: 12 x 2
#    path                                                               n
#    <fs::path>                                                <fs::bytes>
#  1 data/raw/twitter/2019-07-06-Night02Tweets.rds                    7.99M
#  2 data/raw/twitter/2019-07-06-Night02TweetsRaw.rds                 7.87M
#  3 data/raw/twitter/2019-07-06-Night01Tweets.rds                    6.92M
#  4 data/raw/twitter/2019-07-06-Night01TweetsRaw.rds                 6.83M
#  5 data/raw/twitter/2019-07-06-Night02TweetsUsers.rds                1.9M
#  6 data/raw/twitter/2019-07-06-Night01TweetsUsers.rds               1.65M
#  7 data/raw/google-trends/2019-07-10-Dems2020Night1Group2.rds     111.47K
#  8 data/raw/google-trends/2019-07-10-Dems2020Night1Group1.rds     109.52K
#  9 data/raw/wikipedia/2019-07-25-PollingCriterionRaw.csv              722
# 10 data/raw/538/2019-07-06-Cand538Fav.csv                             581
# 11 data/raw/wikipedia/2019-07-10-WikiDemAirTime02Raw.csv              202
# 12 data/raw/wikipedia/2019-07-10-WikiDemAirTime01Raw.csv              191
```

The output tells us the twitter data files are the largest (`7.99M`). All of these files are small enough for RStudio.Cloud to handle, though.

## Loading the data into R

We can import the data using the `01-import.R` file in the `code` folder. Feel free to open this file and examine its contents in the **Source** pane. We will use the `base::source()` function to run all of the code in the `01-import.R` file.

```
# import data using 01-import.R file
source("code/01-import.R")
```

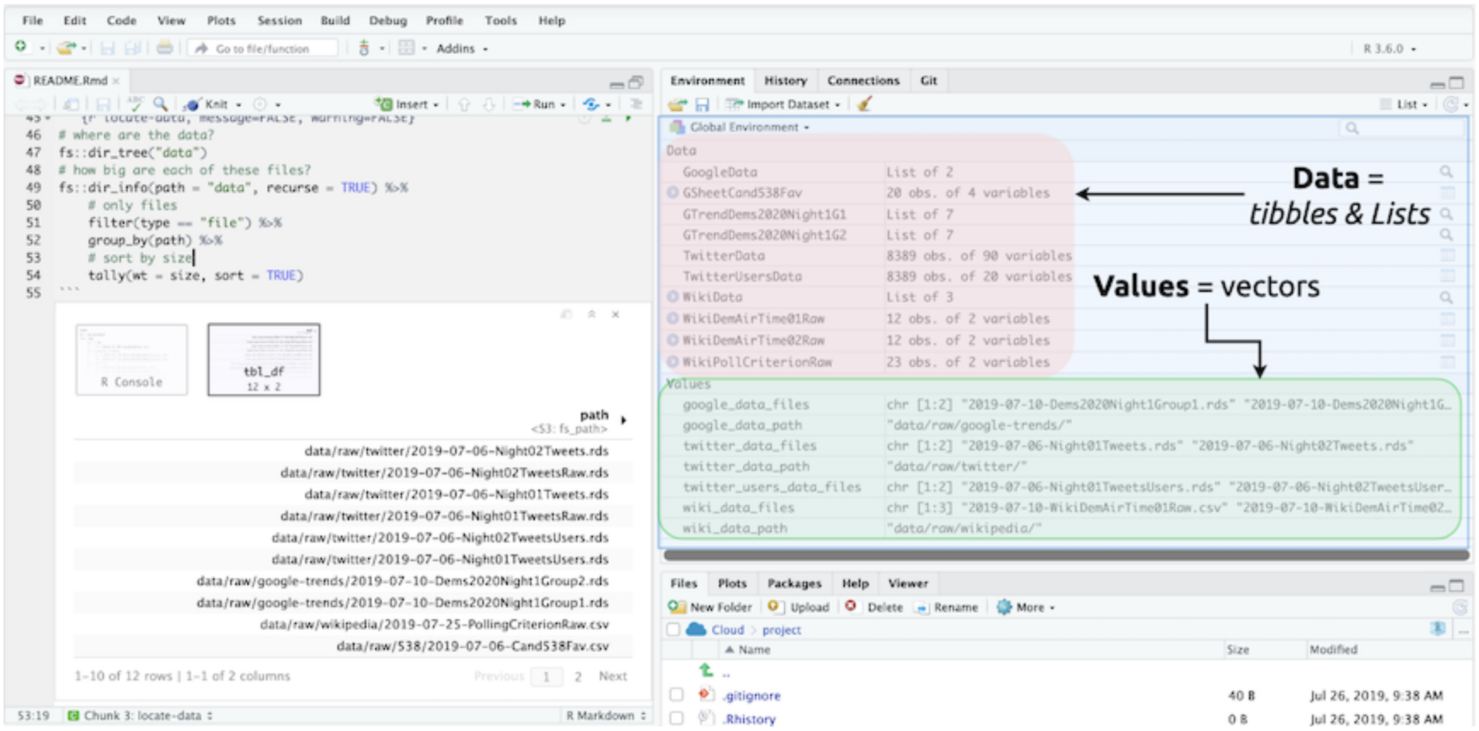
The `01-import.R` file loads all of the data files into the RStudio.Cloud session. We can verify this by examining the contents of the environment with `base::ls()`.

```
base::ls()
```

The output tells us the following objects are in our working environment.

```
#  [1] "google_data_files"
#  [2] "google_data_path"
#  [3] "GoogleData"
#  [4] "GSheetCand538Fav"
#  [5] "GTrendDems2020Night1G1"
#  [6] "GTrendDems2020Night1G2"
#  [7] "twitter_data_files"
#  [8] "twitter_data_path"
#  [9] "twitter_users_data_files"
# [10] "TwitterData"
# [11] "TwitterUsersData"
# [12] "wiki_data_files"
# [13] "wiki_data_path"
# [14] "WikiData"
# [15] "WikiDemAirTime01Raw"
# [16] "WikiDemAirTime02Raw"
# [17] "WikiPollCriterionRaw"
```

We can also see these objects in the **Environment** pane.



```
{r locate-data, message=FALSE, warning=FALSE}
# where are the data?
fs::dir_tree("data")
# how big are each of these files?
fs::dir_info(path = "data", recurse = TRUE) %>%
    # only files
    filter(type == "file") %>%
    group_by(path) %>%
    # sort by size
    tally(wt = size, sort = TRUE)
```
```

R Console
tbl_df 12 x 2

| path <S3: fs_path> |
|---|
| data/raw/twitter/2019-07-06-Night02Tweets.rds |
| data/raw/twitter/2019-07-06-Night02TweetsRaw.rds |
| data/raw/twitter/2019-07-06-Night01Tweets.rds |
| data/raw/twitter/2019-07-06-Night01TweetsRaw.rds |
| data/raw/twitter/2019-07-06-Night02TweetsUsers.rds |
| data/raw/twitter/2019-07-06-Night01TweetsUsers.rds |
| data/raw/google-trends/2019-07-10-Dems2020Night1Group2.rds |
| data/raw/google-trends/2019-07-10-Dems2020Night1Group1.rds |
| data/raw/wikipedia/2019-07-25-PollingCriterionRaw.csv |
| data/raw/538/2019-07-06-Cand538Fav.csv |

1-10 of 12 rows | 1-1 of 2 columns    Previous 1 2 Next

```
Data
  GoogleData              List of 2
  GSheetCand538Fav        20 obs. of 4 variables
  GTrendDems2020Night1G1  List of 7
  GTrendDems2020Night1G2  List of 7
  TwitterData             8389 obs. of 90 variables
  TwitterUsersData        8389 obs. of 20 variables
  WikiData                List of 3
  WikiDemAirTime01Raw     12 obs. of 2 variables
  WikiDemAirTime02Raw     12 obs. of 2 variables
  WikiPollCriterionRaw    23 obs. of 2 variables
Values
  google_data_files       chr [1:2] "2019-07-10-Dems2020Night1Group1.rds" "2019-07-10-Dems2020Night1G…
  google_data_path        "data/raw/google-trends/"
  twitter_data_files      chr [1:2] "2019-07-06-Night01Tweets.rds" "2019-07-06-Night02Tweets.rds"
  twitter_data_path       "data/raw/twitter/"
  twitter_users_data_files chr [1:2] "2019-07-06-Night01TweetsUsers.rds" "2019-07-06-Night02TweetsUser…
  wiki_data_files         chr [1:3] "2019-07-10-WikiDemAirTime01Raw.csv" "2019-07-10-WikiDemAirTime02…
  wiki_data_path          "data/raw/wikipedia/"
```

| Name | Size | Modified |
|---|---|---|
| .. | | |
| .gitignore | 40 B | Jul 26, 2019, 9:38 AM |
| .Rhistory | 0 B | Jul 26, 2019, 9:38 AM |

The *Global Environment* holds all of our imported data.

## Documenting changes to our project with Git

So, we've imported some data into the RStudio.Cloud session, but we need to make sure we're keeping track of the changes to our files.

We can do this by checking our `git status` in the **Terminal** pane.

```
$ git status
On branch master

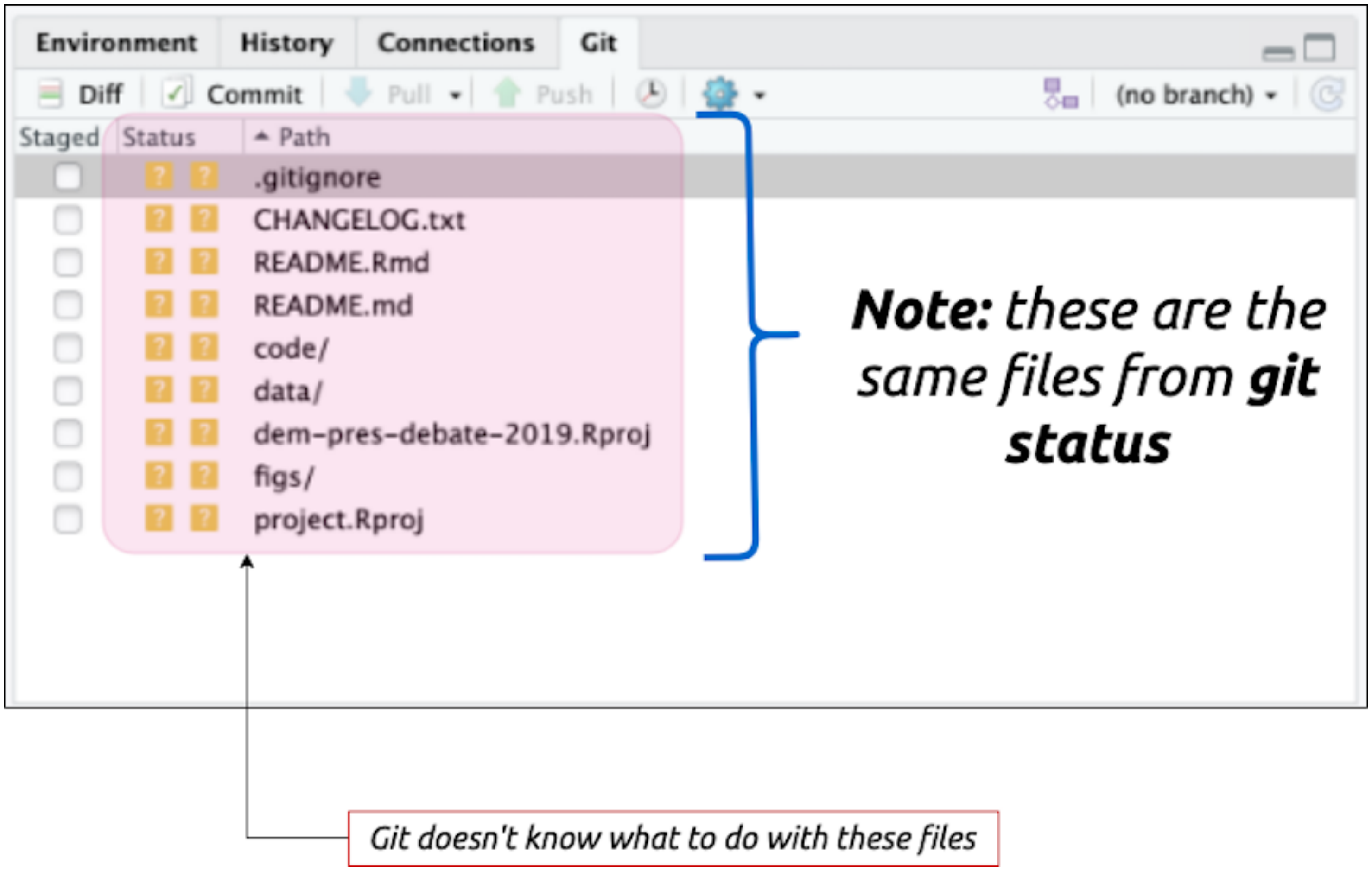
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        CHANGELOG.txt
        README.Rmd
```
```

```
        README.md
        code/
        data/
        dem-pres-debate-2019.Rproj
        figs/
        project.Rproj

nothing added to commit but untracked files present (use "git add" to track)
```

Git is telling us we have quite a few new files in our project (which we expected), but that to git to pay attention to them, we need to `add` them. We can do this in the **Git** pane RStudio provides (it's by the **Environment** pane).

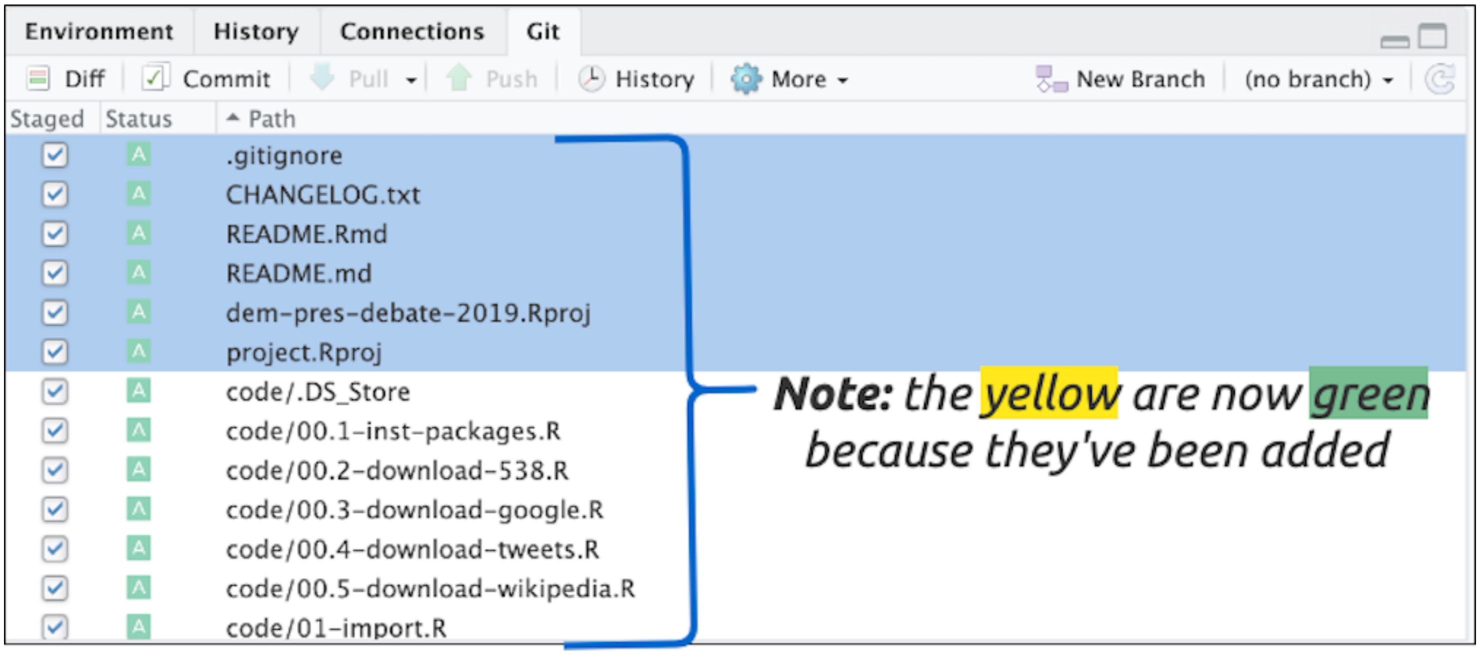


```
.gitignore
CHANGELOG.txt
README.Rmd
README.md
code/
data/
dem-pres-debate-2019.Rproj
figs/
project.Rproj
```

**Note:** these are the same files from **git status**

Git doesn't know what to do with these files

We will click on each checkbox next to the files in the **Git** pane. Don't be alarmed if this list expands to include all the subfiles and folders.



| Staged | Status | Path |
|---|---|---|
| ✓ | A | .gitignore |
| ✓ | A | CHANGELOG.txt |
| ✓ | A | README.Rmd |
| ✓ | A | README.md |
| ✓ | A | dem-pres-debate-2019.Rproj |
| ✓ | A | project.Rproj |
| ✓ | A | code/.DS_Store |
| ✓ | A | code/00.1-inst-packages.R |
| ✓ | A | code/00.2-download-538.R |
| ✓ | A | code/00.3-download-google.R |
| ✓ | A | code/00.4-download-tweets.R |
| ✓ | A | code/00.5-download-wikipedia.R |
| ✓ | A | code/01-import.R |

***Note:*** *the yellow are now green because they've been added*

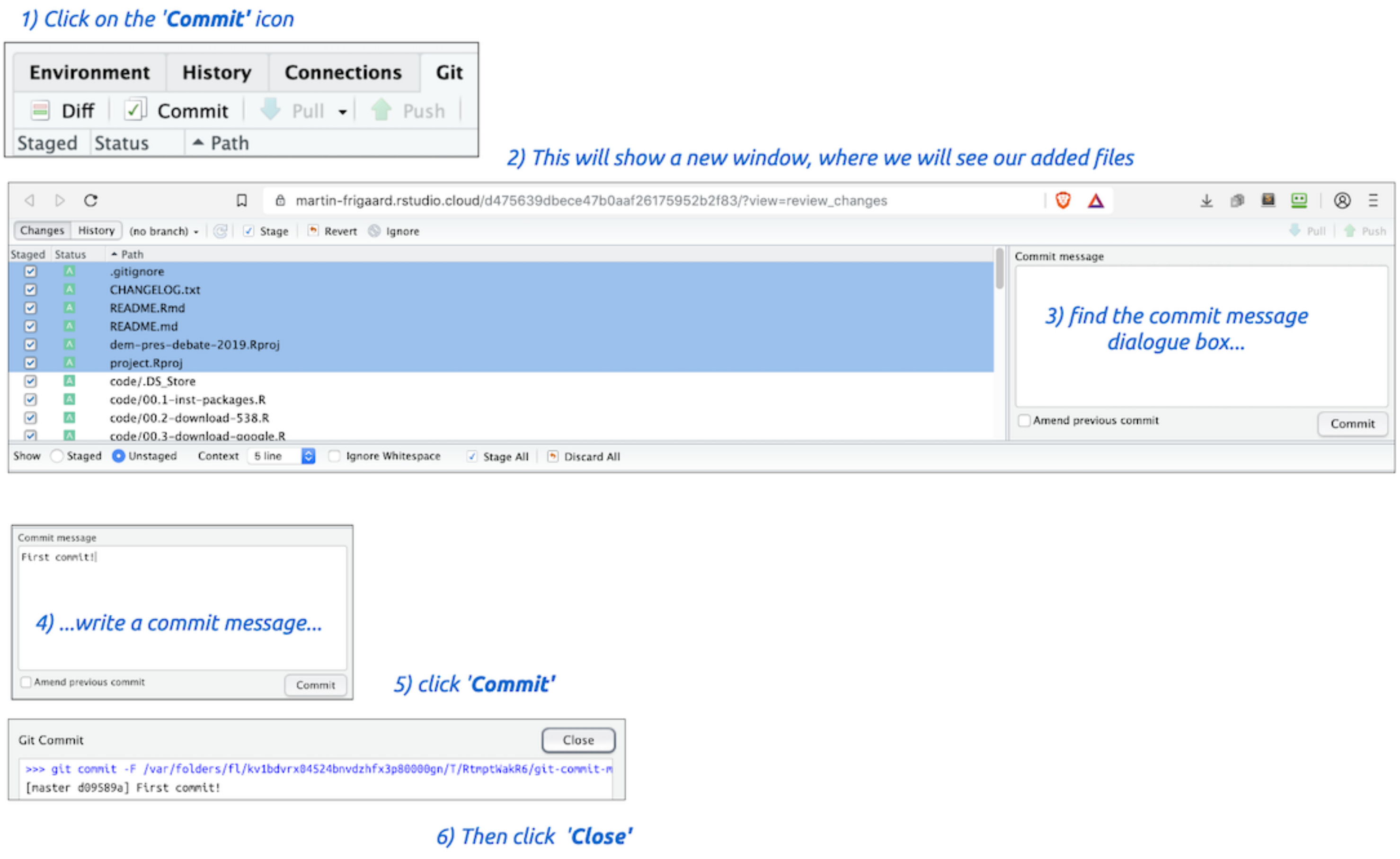
Now we can re-check `git status` to see what just happened.

```
$ git status

On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   CHANGELOG.txt
        new file:   README.Rmd
        new file:   README.md
        new file:   code/.DS_Store
        new file:   code/00.1-inst-packages.R
        new file:   code/00.2-download-538.R
        new file:   code/00.3-download-google.R
        new file:   code/00.4-download-tweets.R
        new file:   code/00.5-download-wikipedia.R
        new file:   code/01-import.R
        new file:   code/02-wrangle.R
        new file:   data/.DS_Store
        new file:   data/processed/.DS_Store
        new file:   data/raw/.DS_Store
        new file:   data/raw/538/2019-07-06-Cand538Fav.csv
        # omitted
```

This long list of files shows all the contents have been added and are ready to be committed.

## Committing the changes

If we click on the *Commit* icon in the **Git** pane, this will open a new window in RStudio.Cloud. In this window, we will enter a commit message ("First commit!"), and click the *Commit* button. We've outlined this entire process in the schematic below:


1) Click on the 'Commit' icon
Environment
History
Connections
Git
Diff
Commit
Pull
Push
2) This will show a new window, where we will see our added files
martin-frigaard.rstudio.cloud/d475639dbece47b0aaf26175952b2f83/?view=review_changes
.gitignore
CHANGELOG.txt
README.Rmd
README.md
dem-pres-debate-2019.Rproj
project.Rproj
code/.DS_Store
code/00.1-inst-packages.R
code/00.2-download-538.R
code/00.3-download-google.R
3) find the commit message dialogue box...
First commit!
4) ...write a commit message...
5) click 'Commit'
6) Then click 'Close'

```
>>> git commit -F /var/folders/fl/kv1bdvrx84524bnvdzhfx3p80000gn/T/RtmptWakR6/git-commit-m
[master d09589a] First commit!
```

## A quick git terminology overview

Below are some commonly used terms/commands associated with Git and Github.

***init*** - the command `git init` is used to initialize a new git repository (it tells Git to start tracking changes in this directory).

***status*** - Git status will tell you what you've done and what is happening. You can check the status of a git repository with `git status` (use this liberally).

***add*** - In order for Git to keep track of the changes we make to files, we have to tell it which files to pay attention to. We can do this using `git add`. The `git add -A` tells git to stage *ALL* the files that are in an initialized repo.

***commits*** - commits are the staple in Git/Github the workflow. Commits are what Git uses to track the changes you've made to files or folders. Commits are confusing because they can be nouns ("I'm creating a commit with these changes") or verbs (I am going to commit these changes to my project").

To quote David Demaree,

> *"Semantically, each commit represents a complete snapshot of the state of your project at a given moment in time; its unique identifier serves to distinguish that state from the way the files in your project looked at any other moment in time."*

***repository*** - repos are the files and folders in your project and all the changes you make while working on them. On your local computer, a repository can exist in a folder you initialize a repository in (see below). On Github, a repo has the following structure: `https://github.com/<username>/<repository_name>`.

***clone*** - this command copies all files and changes into a new working directory from a remote, initializes (`init`) a new Git repository, and it adds a remote called `origin`.

***diff*** - this is how Git shows differences between files. Read more about how changes are formatted/displayed here.