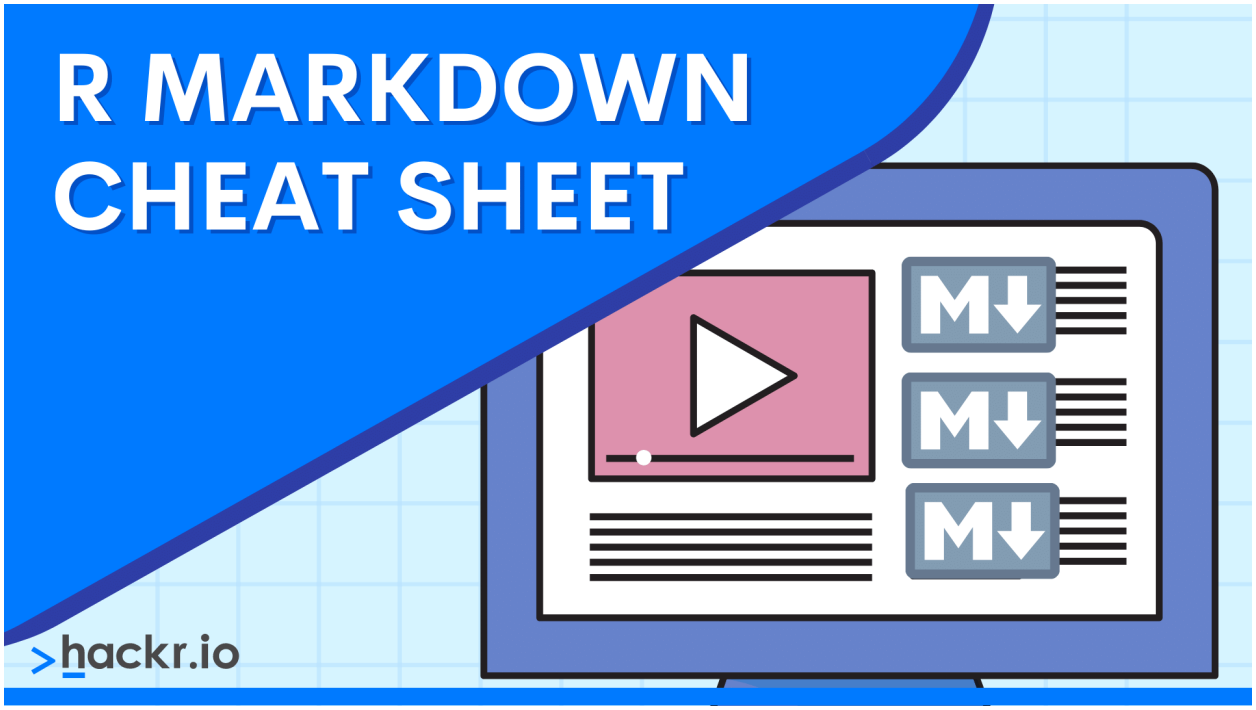


The Ultimate R Markdown Cheat Sheet

Installing R Markdown

R markdown is an open-source tool that's easy to install and use. All you need is to hit the following command from your terminal:

install.packages("rmarkdown")

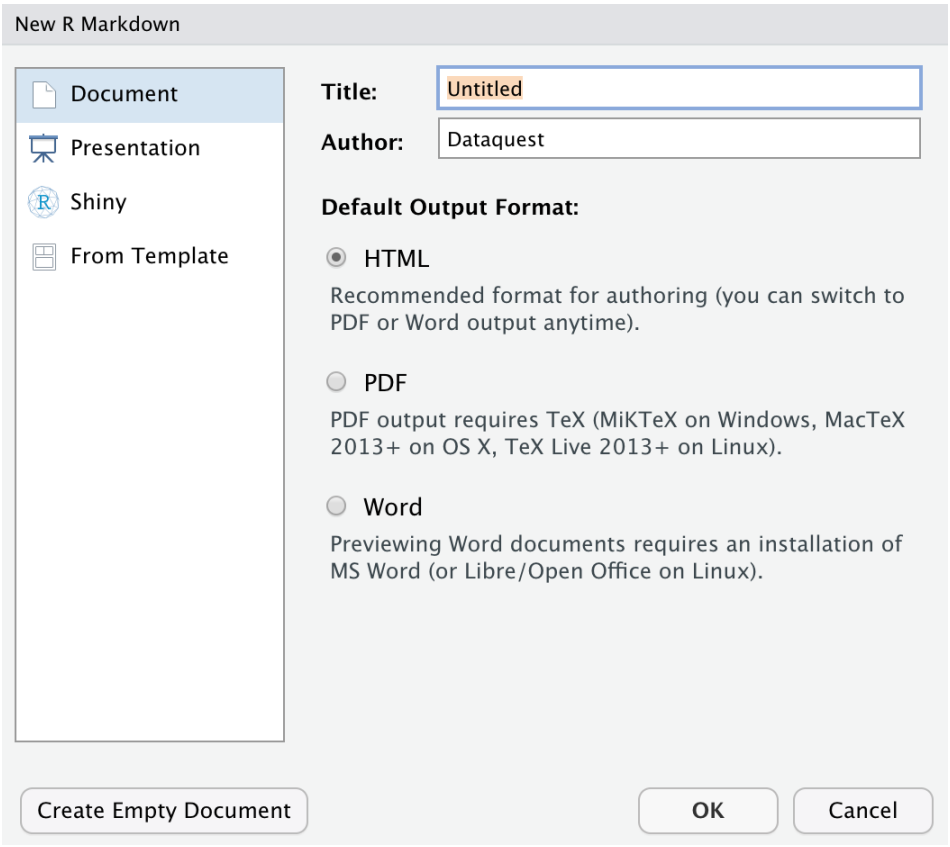
Once you install the R Markdown, open a new R Markdown file using RStudio. For that, go to File > New File > R Markdown. You can see that all the R Markdown file names have a file extension ".Rmd".

R Markdown Workflow

- Open a new .Rmd file at File ▸ New File ▸ R Markdown. Use the wizard that opens to prepopulate the file with a template.
- Write document by editing template.
- Knit document to create a report; use a knit button or render() to knit.
- Preview Output in IDE window.
- Publish (optional) to a web server.
- Examine the build log in the R Markdown console.
- Use the output file saved alongside .Rmd

Default Output Format

After opening a new R Markdown file in RStudio, you'll get a pop-up window to select a format:



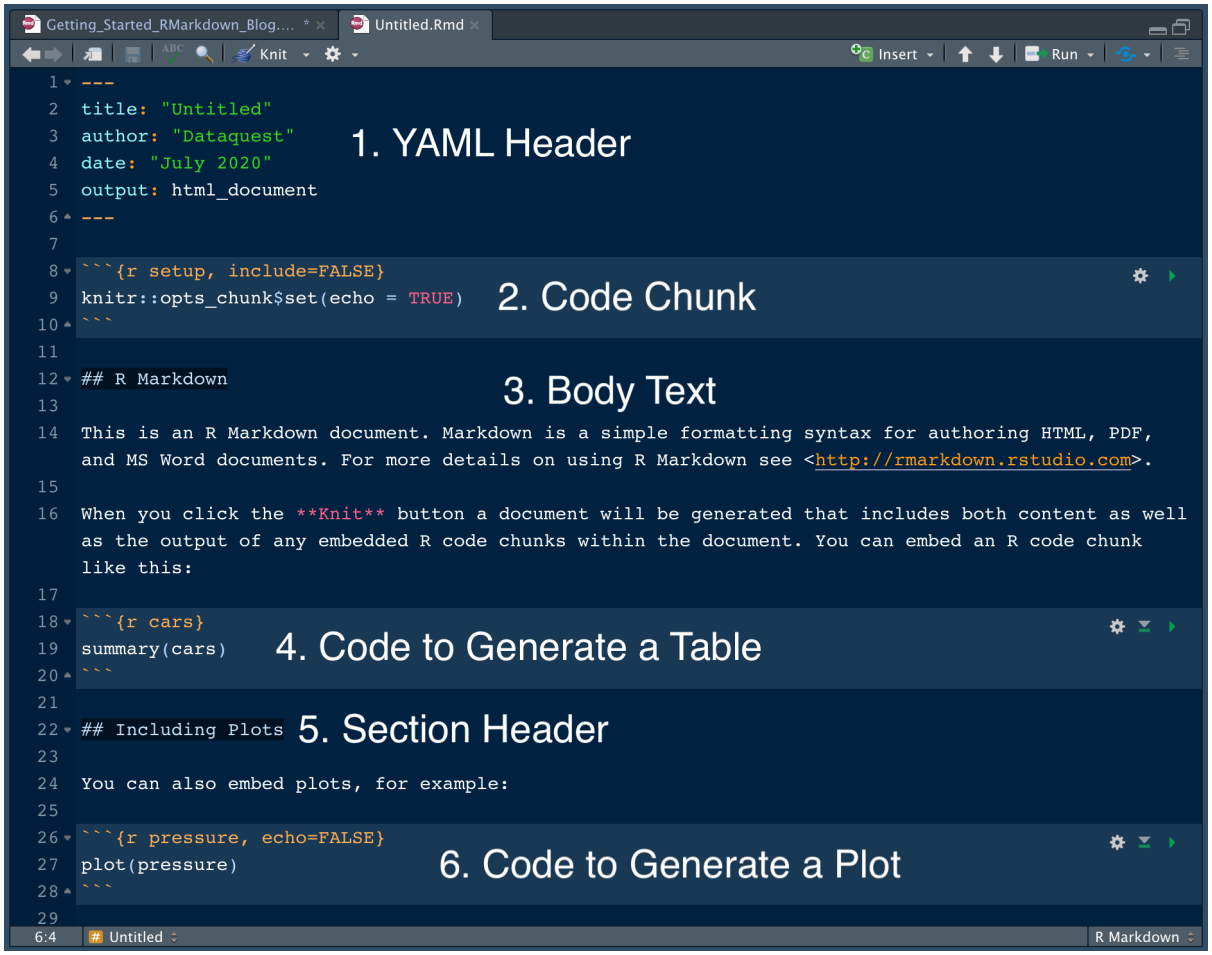
You will see an option to add a title, author, and default output format. Once you choose the right options, click OK. Then, you'll get a new window that explains more about the R Markdown files.

One available default output format is HTML, which helps you easily view the created file in a web browser. For this cheat sheet, we are using the default HTML setting — a faster alternative to a PDF or other format. When nearing completion, you can change the output to your preferred format and make final changes.

The title you specify for your document in the pop-up above is not the file name. Navigate to File > Save As.. to name, and save, the document.

R Markdown Document Format

After selecting your document's output format, you'll see an R Markdown document in your RStudio pane. Unlike an R script which is blank, the .Rmd document comes with some formatting.

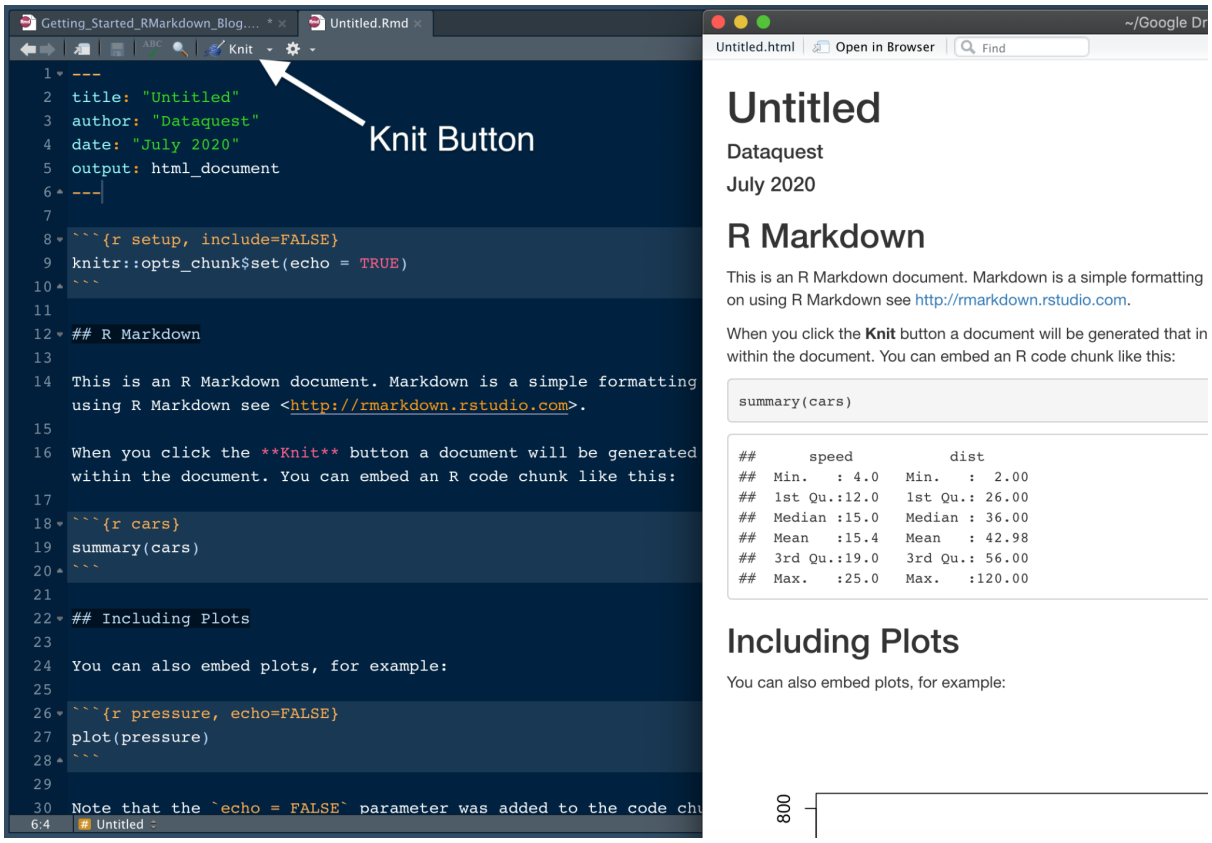


An R Markdown document has the following critical sections.

- **Header**: Bounded by the (---) dashes, where the user mentions all details like title, name, date, and document type. If you already filled out details in the previous window, they'll be automatically populated in the document.
- **Body text**: Starts with the "##". This section will render as text, but a PDF will be created along with the formatting being applied by the user.
- **Code chunk:** Bounded by "```" specifying the code that will run within the document to create a PDF.
- **Code to Generate a Plot:** Not included because the parameter echo=FALSE is specified. This is a chunk option.

Now the document is ready to generate the output. But for that, you need to knit the code and text to see preview formatting specifications. To knit your document, hit the "knit" button in your RStudio. This will generate the HTML document. You can also use this shortcut to knit the document:

Command + shift + K for Mac or Ctrl + Shift + K on Linux and Windows.



On the right pane above, you can preview your document in HTML format. The default ".Rmd" file comes with the guidance on formatting R Markdown documents.

Here, we'll save this document as RMarkdown_Guide.Rmd. The title "R Markdown Guide" is in the YAML header.

But, if you're working in R Markdown without RStudio, you can use the function rmarkdown::render() to compile your document. Just provide your document's name in quotes as the function argument:

`rmarkdown::render("RMarkdown_Guide.Rmd")`

Section Headers

An R Markdown file is a plain-text file written in Markdown, a formatting syntax. If you look closely at the default ".Rmd" file, you will see two sections in the document, R

Markdown and Including Plots. These are second-level headers. Now, we will create a new second-level header named Text Formatting Basics by entering:

Text Formatting Basics.

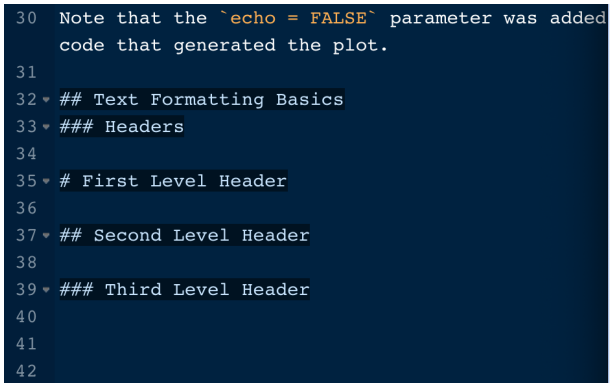
Now, follow this with a third-level header, called *Headers*:

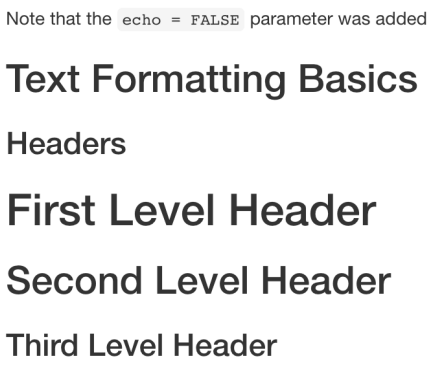
Text Formatting Basics
Headers

Now, we'll create syntax requirements for first, second, and third-level headers to show the header generating code:

First Level Header
Second Level Header
Third Level Header

Separate each line of code with a blank line at the output. And always have at least one blank line in-between adjacent and different format elements, such as section headers and body text. Once you make the changes, you will see the following output in your document.





You can see how different the second and third-level headers look when rendered, as well as the syntax for creating headers with #, ## or ###. If you don't want the headers to render as headers in your final output, you must wrap the code in backticks:

`# First Level Header`

Bulleted and Numbered Lists

Now, we'll create a new third-level header called *Bulleted and Numbered Lists*.

Type the following into your code:

```
* List element 1
* List element 2
* List element 3
 * List item 3a
 * List item 3b
```

To create unordered list items, use characters like *,-, and +.

Syntax:

```
1. Numbered list 1
1. Numbered list 2
1. Numbered list 3.
```

The numbers are auto-increment, so we have to enter 1. This makes adding and deleting items easy, since you don't have to renumber. You can also combine the ordered and unordered lists.

Hit the tab twice to indent the unordered bullets:

```
1. Numbered list 1
1. Numbered list 2
 * Item 1
 * Item 2
```

In the below example, you can see the side-by-side formatting view:

```
### Bulleted or Numbered Lists
* List element 1
* List element 2
* List element 3
  * List element 3a
  * List element 3b

Numbered lists:

1. Numbered list 1
1. Numbered list 2
1. Numbered list 3. The numbers auto-in
ever add or delete items, because we do

Numbered and unordered lists combined:

1. Numbered list 1
1. Numbered list 2
    * Item 1
    * Item 2
```

Bulleted or Numbered Lists

- List element 1
- List element 2
- List element 3
    - List element 3a
    - List element 3b

Numbered lists:

1. Numbered list 1
2. Numbered list 2
3. Numbered list 3. The numbers auto-increment, have to worry about renumbering!!!

Numbered and unordered lists combined:

1. Numbered list 1
2. Numbered list 2
    - Item 1
    - Item 2

Text Formatting

Let's create a new third-level header called *text formatting:*

** Make text italic like *this* or _this_.*
** Make text bold like **this** or __this__.*
** Use `backticks` for code.*
** Wrap a character to subscript in tildes (`~`). For example, `H~2~O` renders as H~2~O.*
** Wrap a character to superscript in carets (`^`), like this: `R^2^` renders as R^2^.*

You will get the below output after rendering the code:

```
### Text Formatting
* Make text italic like *this* or _this_.
* Make text bold like **this** or __this__.
* Use `backticks` for code.
* Wrap a character to subscript in tildes (`
* Wrap a character to superscript in carets
```

Text Formatting

- Make text italic like *this* or *this*.
- Make text bold like **this** or **this**.
- Use `backticks` for code.
- Wrap a character to subscript in tildes (`~`). For example, `H~2~O` renders as H_2O.
- Wrap a character to superscript in carets (`^`), like this: `R^2^` renders as R^2.

Links

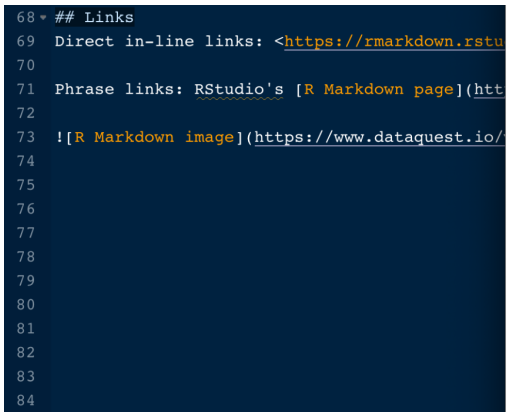
With R Markdown, you can simply link your text with different types of websites and images. Use the following code to link.

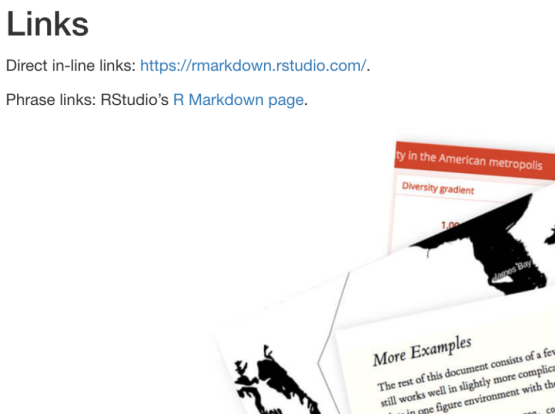
Direct inline links: <https://rmarkdown.rstudio.com/>.

Phrase links: RStudio's [R Markdown page](https://rmarkdown.rstudio.com/).

![R Markdown image](https://www.dataquest.io/wp-content/uploads/2020/06/r-markdown-1536x976.jpg)

After rendering, you'll get this output in HTML:





Code Chunks

To add new code to your R document, use the code chunks section. There are several ways to add a new code, such as.

- Command + Option + I on a Mac, or Ctrl + Alt + I on Linux and Windows.
- "Insert" drop-down Icon in the toolbar and select R.

Use the shortcuts below to save time:

- **cache** - cache results for future knits (default = FALSE)
- **cache.path** - directory to save cached results in (default = "cache/")
- **child - file(s)** to knit and then include (default = NULL)
- **collapse** - collapse all output into single block (default = FALSE)
- **comment** - prefix for each line of results (default = '##')
- **dependson** - chunk dependencies for caching (default = NULL)
- **echo** - Display code in output document (default = TRUE)
- **engine** - code language used in chunk (default = 'R')
- **error** - Display error messages in doc (TRUE) or stop render when errors occur (FALSE) (default = FALSE)
- **eval** - Run code in chunk (default = TRUE)
- **message** - display code messages in document (default = TRUE)
- **results (default = 'markup') 'asis'** - passthrough results 'hide' - do not display results 'hold' - put all results below all code

- **tidy** - tidy code for display (default = FALSE)
- **warning** - display code warnings in document (default = TRUE)
- **fig.align** - 'lef', 'right', or 'center' (default = 'default')
- **fig.cap** - figure caption as character string (default = NULL)
- **fig.height, fig.width** - Dimensions of plots in inches highlight - highlight source code (default = TRUE)
- **include** - Include chunk in doc afer running (default = TRUE)

Parameters

You can easily add parameters to your documents to reuse with different inputs (e.g., data, values, etc.). For example, you can **create and set parameters** in the header as sub-values of params:

--- params:
n: 100 d:
!r Sys.Date()
—

- **Calling parameters**

Call parameter values in code as params$. For example-
Today's date is `r params$d`

- **Set parameters**

Set values with Knit with parameters or the params argument of render():

render("doc.Rmd", params = list(n = 1, d = as.Date("2015-01-01"))

Render

You can use the rmarkdown::render() to render/knit at the command line. Here are some args you can use:

- **input - file** to render output_format
- **output_options:** List of render options (as in YAML)
- **output_file output_dir**
- **params**: list of params to use
- **envir**: environment to evaluate code chunks in
- **Encoding of input file**

Interactive Documents

Turn your report into an interactive document with these steps:

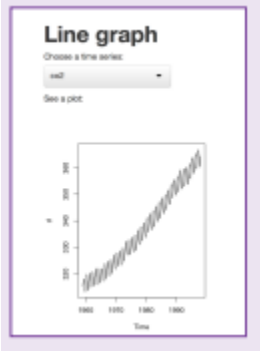
- **Add runtime: shiny to the YAML header.**

--- title: "Line graph" output: html_document runtime: shiny ---

- **Call Shiny input functions to embed input objects.**

--- title: "Line graph" output: html_document runtime: shiny ---
Choose a time series: ```{r echo = FALSE} selectInput("data", "", c("co2", "lh")) ```
See a plot: ```{r echo = FALSE} renderPlot({ d <- get(input$data) plot(d) }) ```

- **Call Shiny render functions to embed reactive output.**
- **Render with rmarkdown::run or click Run Document in RStudio IDE**



How to Run a Code

With R Markdown, you will get different options for running a new code. Go to the toolbar. Under the “Run” option, you’ll see your options:

Remember to restart your R session before running any new code by hitting the Command + Shift + F10 on a Mac, or Control + Shift + F10 on Linux and Windows.

Here are some other useful shortcuts:

- **Run all chunks above the current chunk:** Command + Option + P on a Mac, or Ctrl + Alt + P on Linux and Windows.
- **Run the current chunk:** Command + Option + C or Command + Shift + Enter on a Mac. On Linux and Windows, use Ctrl + Alt + C or Ctrl + Shift + Enter
- **Run the next chunk:** Command + Option + N on a Mac, or Ctrl + Alt + N on Linux and Windows.
- **Run all chunks:** Command + Option + R or Command + A + Enter on a Mac. On Linux and Windows, use Ctrl + Alt + R or Ctrl + A + Enter to run all chunks.

> install.packages("Dataquest")

Control Behavior with Code Chunk Options

You have complete freedom to control your code chunks, including their evaluation and presentation. You can use these options for creating presentations and valuable reports from scratch. You can include code, plots, tables, and even images. For example, you can mention a plot specifying the final results without including the code.

- **echo** = FALSE: you can use this option, if you do not want to show your code in the output, but run code and generate all outputs, plots, warnings and messages.
- **eval** = FALSE: you can use this option to display the code, but do not evaluate it.
- **fig.show** = "hide": this option is used to hide the plots.
- **include** = FALSE: this option will allow the code to run, but suppress all output. It is useful for setup code.
- **message** = FALSE: this option will prevent the packages from printing messages whenever they load. This will also suppress the messages generated by functions.
- **results** = "hide": this option will hide the printed output.
- **warning** = FALSE: this option will prevent the packages and functions from displaying warnings.

Inline Code

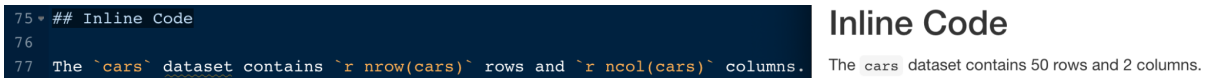
You can use inline code to embed the R code directly into the R Markdown document. This is ideal for including specific information about data in the text.

For example:

Use the inline code with r, then add the specific code for evaluation within the backticks. Here, we summarize the number of rows and columns in the cars dataset built-in to R.

Inline Code

The `cars` dataset contains `r nrow(cars)` rows and `r ncol(cars)` columns.



Suppose you want to make a small change to the dataset or change the number of rows and columns. In that case, rerun the code for an accurate result — a much easier solution than going through several documents to check where to update the results and manually altering the results.

Bottom line? R Markdown saves time, improves quality, and ensures report accuracy.

Navigating Sections and Code Chunks

As code length increases, so does complexity. To navigate lengthy code, name the code chunks carefully and purposely. For example: **{r my_boring_chunk_name}.**

Once you name the chunks, they will be available on the navigator included at the bottom of the R Markdown window pane. You can use them to identify plots easily using the name. You can also use the code in other sections of your document with ease, as this navigator allows you to jump to another section of your document quickly.

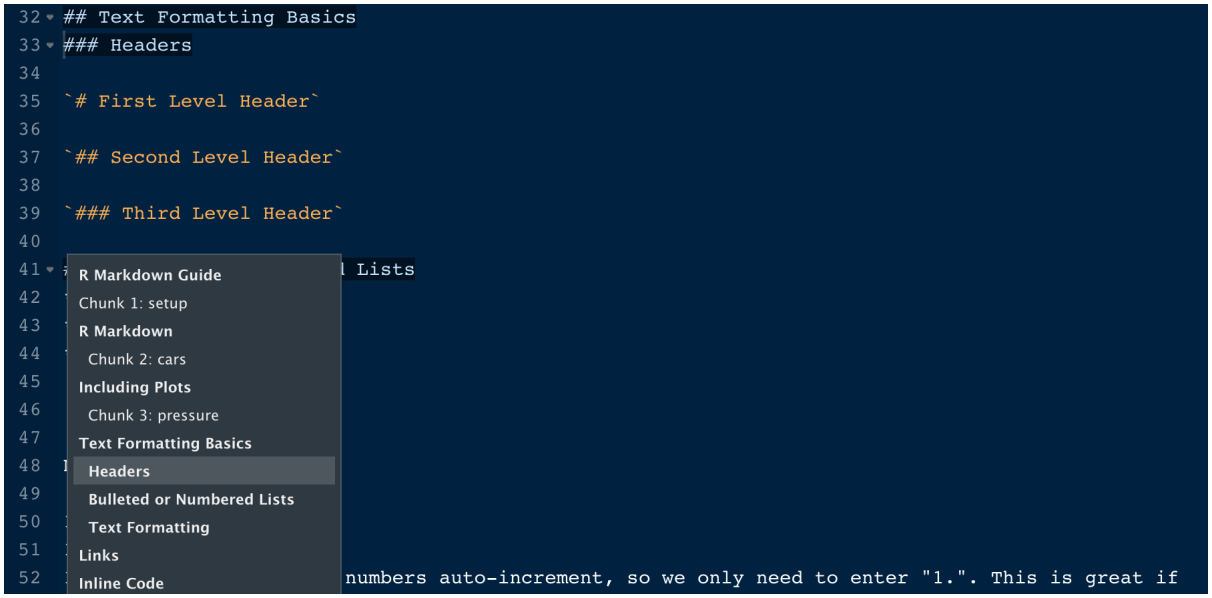


Table Formatting

In R Markdown, you will see the table as they're displayed in the console by default. To improve aesthetics and make changes to the table, use the function knitr::kable():

knitr::kable(head(cars), caption = "The First Few Rows of the Cars Dataset")

Here's the output after rendering:

Output Format Options

All the formatting options are generally specified in the YAML headers.

A single R Markdown document is capable of supporting different output formats. As mentioned, the HTML format is faster to render compared to PDF. For previewing your

document in HTML but will eventually output your document as a PDF, you need to comment out the PDF specifications until they are needed:

---
title: "R Markdown Guide"
author: "Dataquest"
date: "7/8/2020"
output: html_document
 # output: pdf_document
 # output: ioslides_presentation

Presentations

R Markdown has an R Markdown package that comes with the support for several presentation types. Some other useful R packages include revealjs, which expands R Markdown capabilities.

Here are some presentation formats built-into R Markdown:

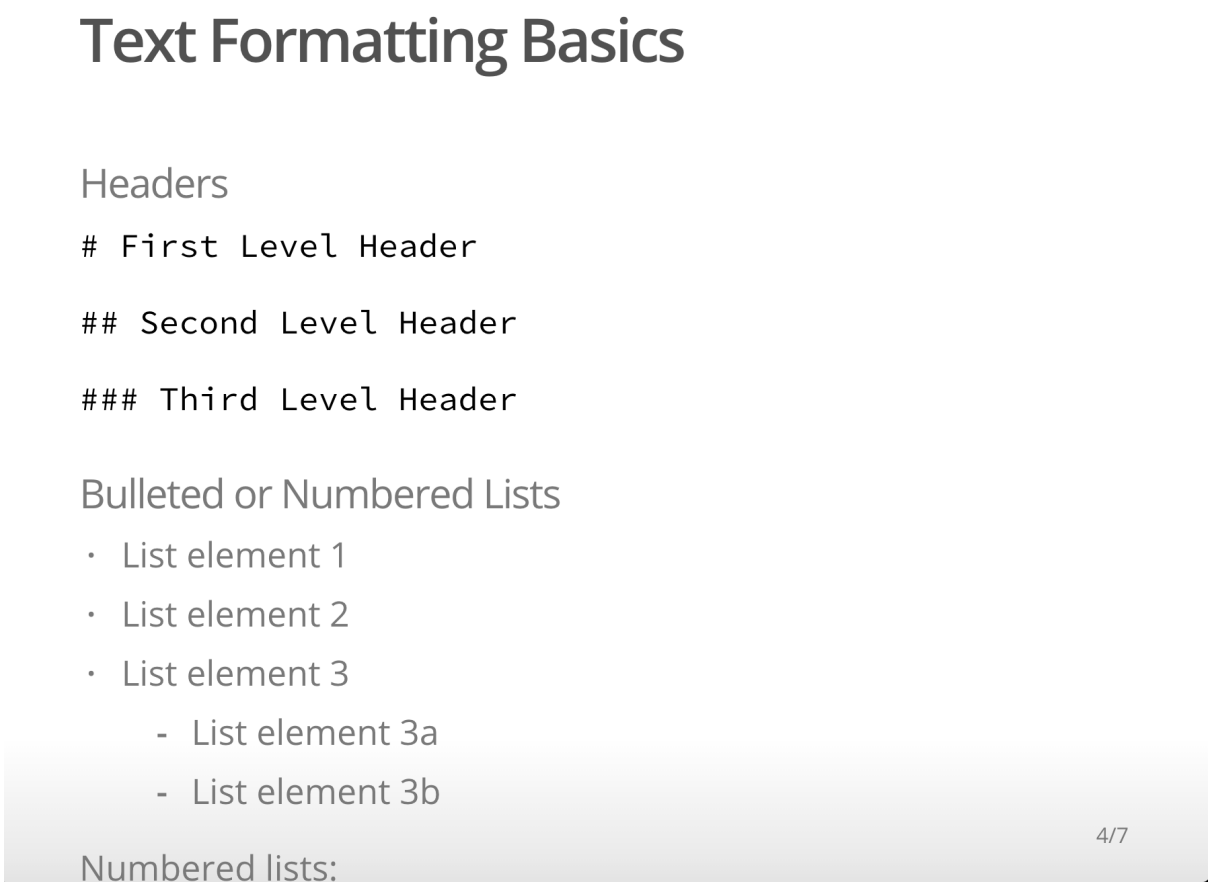
- html_document html
- pdf_document pdf (requires Tex)
- word_document Microsof Word (.docx)
- odt_document OpenDocument Text
- rtf_document Rich Text Format
- md_document Markdown
- github_document Github compatible markdown
- ioslides_presentation ioslides HTML slides
- slidy_presentation slidy HTML slides
- beamer_presentation Beamer pdf slides (requires Tex)

Now, let's convert the R Markdown document into an ioslides presentation (HTML). This is useful for delivering remote presentations with screen sharing, with output: ioslides_presentation.

---
title: "R Markdown Guide"
author: "Dataquest"
date: "7/8/2020"
 # output: html_document
 # output: pdf_document
output: ioslides_presentation

We have “commented-out” the HTML and PDF format options so they will not be considered when compiling the document.

Whenever we knit, the R Markdown Guide and HTML presentation appears with every second-level header to mark the beginning of a new slide. This works well except for our section “Text Formatting Basics” which has a few third-level header sections.

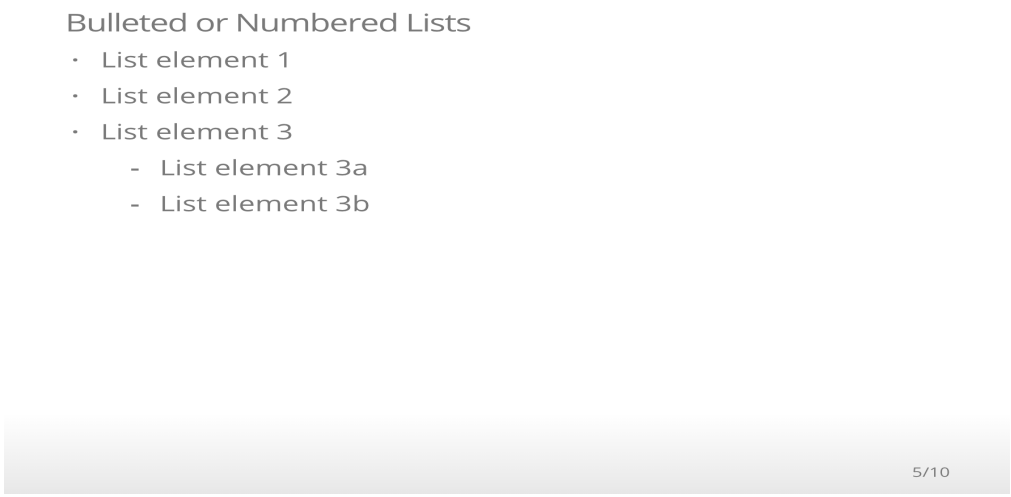


To avoid the manual line breaks, you need to insert *** as needed before each third-level header:

Text Formatting Basics
*### Headers`# First Level Header``## Second Level Header``### Third Level Header`****
Bulleted or Numbered Lists
*** List element 1*

** List element 2*
** List element 3*
** List element 3a*
** List element 3b*

Here's the generated output:



Adding a Table of Contents

A table of contents is important in all web developing, for easy navigation. All you have to do is add one line of code in your YAML header, toc: true:

output:
html_document:
toc: true

Make sure to add the : after html_document!

How to Create a Reusable Template

1. Create a new package with a inst/rmarkdown/templates directory.
2. In the directory, place a folder that contains: template.yaml (see below) skeleton.Rmd (template contents) any supporting files .
3. Install the package .
4. Access template in wizard at File ▸ New File ▸ R Markdown template.yaml.

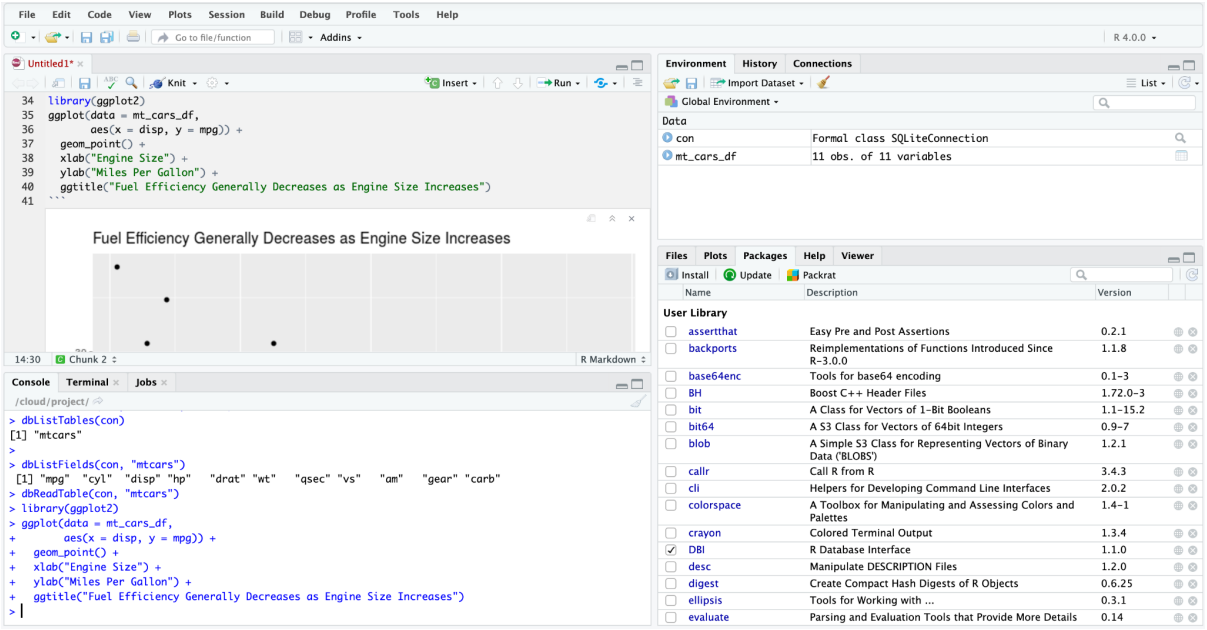
Reproducible Reports with RStudio Cloud

You can apply everything on a cloud-based version of RStudio Desktop called RStudio Cloud. RStudio Cloud allows you to generate reports and presentations in R Markdown without installing the software. All you need is a web browser.

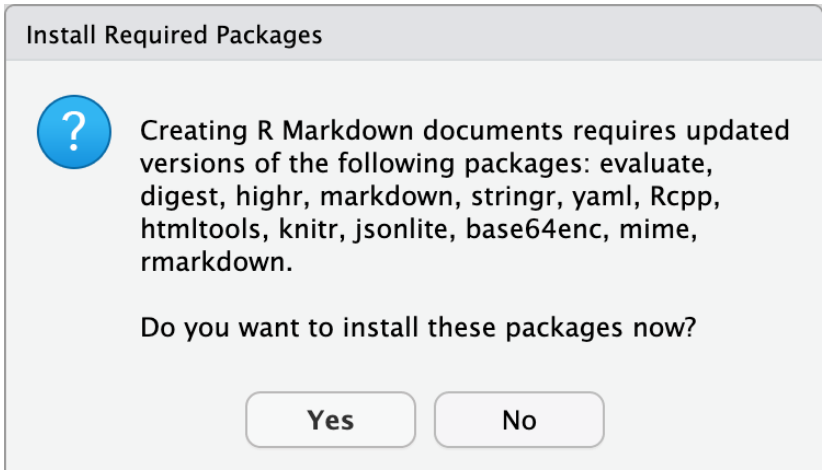
Work in RStudio Cloud is organized into projects similar to the desktop version, but RStudio Cloud requires you to specify the R version for each project.

The Cloud allows you to share projects securely with colleagues, and ensures the working environment is fully reproducible every time someone accesses a project.

As you can see, the RStudio Cloud layout is similar to RStudio Desktop.



When you open a new R Markdown document in RStudio Cloud for the first time, the program provides a prompt asking to install the required packages:



Install the required package — then, you'll be ready to create and knit R Markdown documents.

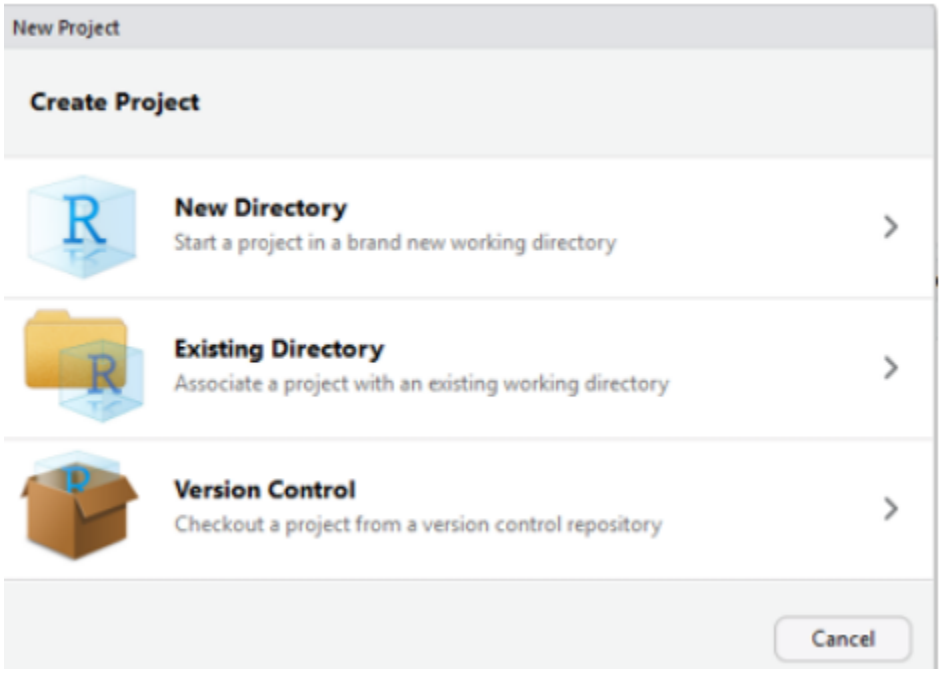
Package Development

Below are simple steps to create a new package.

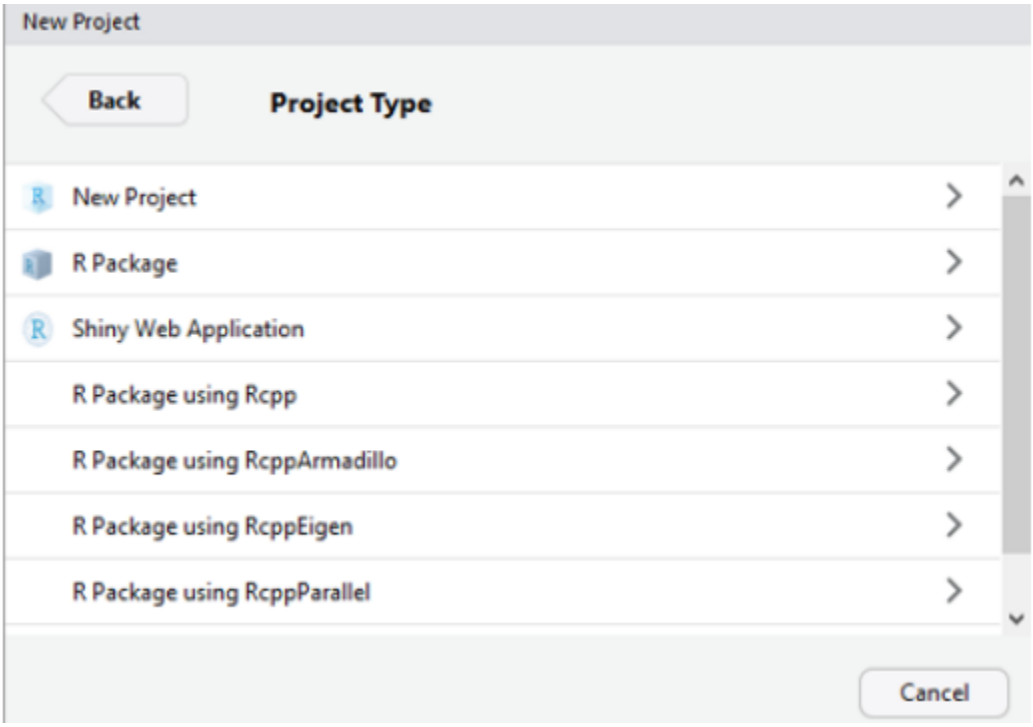
- Install two packages to create new packages.

> install.packages("roxygen2")
> install.packages("devtools")

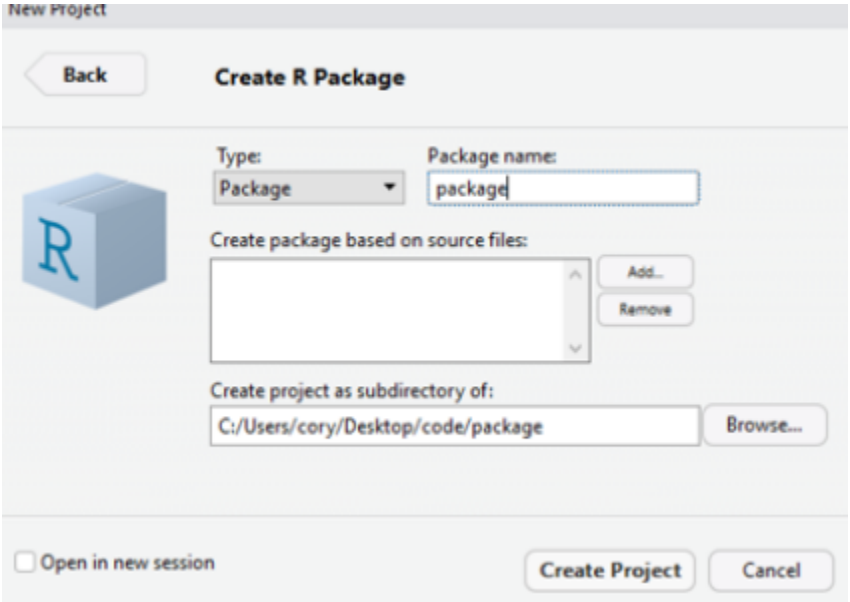
- Open File in RStudio and select New Project.



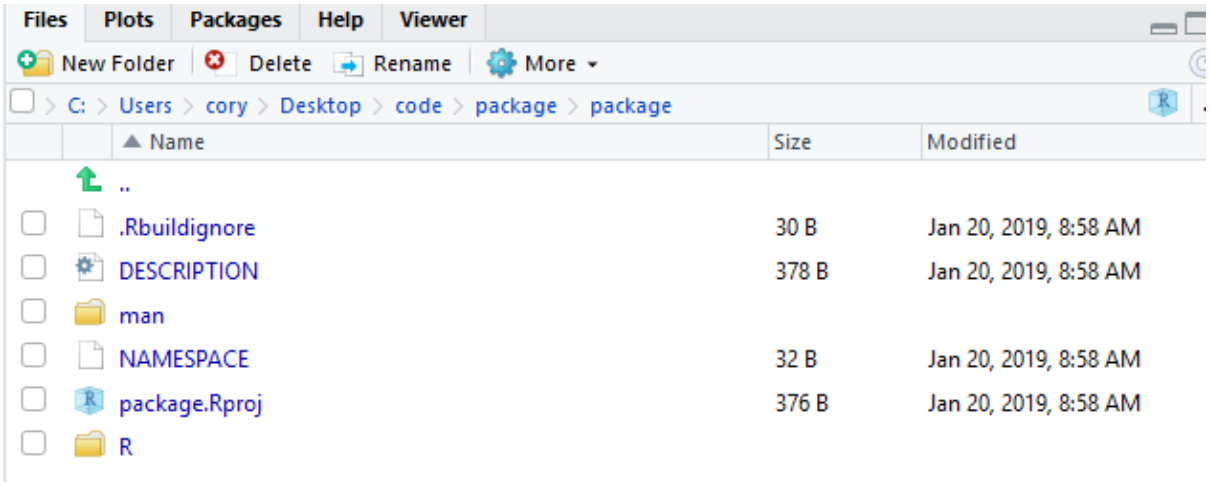
- Select a new directory as desired, and specify R Package:



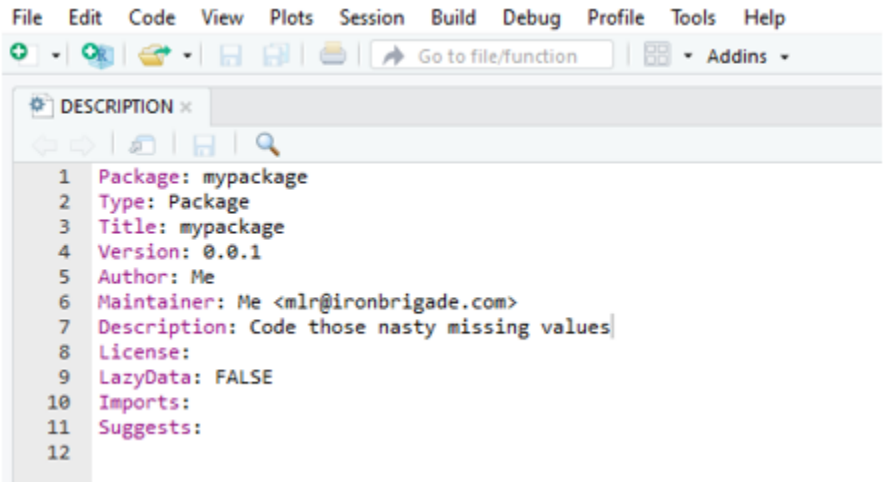
- Name your package:



- Go to your Files tab in RStudio, and you should see several files populated like this:

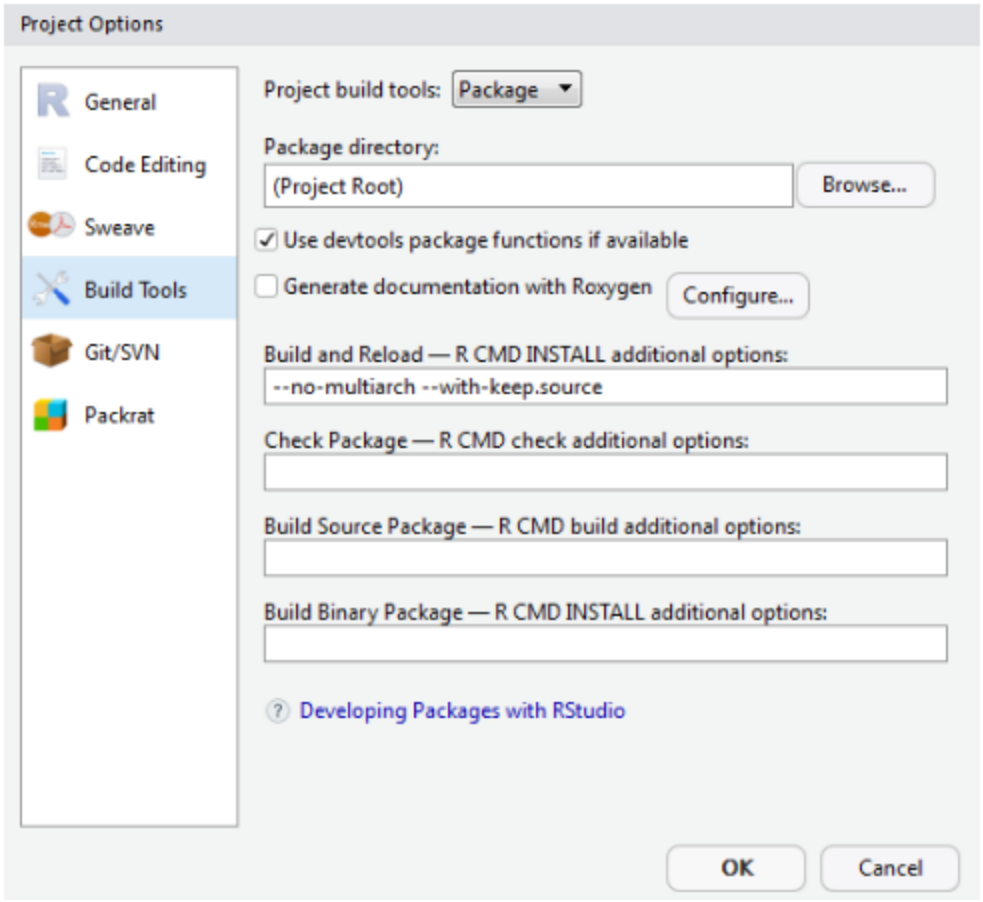


- We'll put our package's R functions under the R folder. Click on Description and fill it out accordingly, and save it.



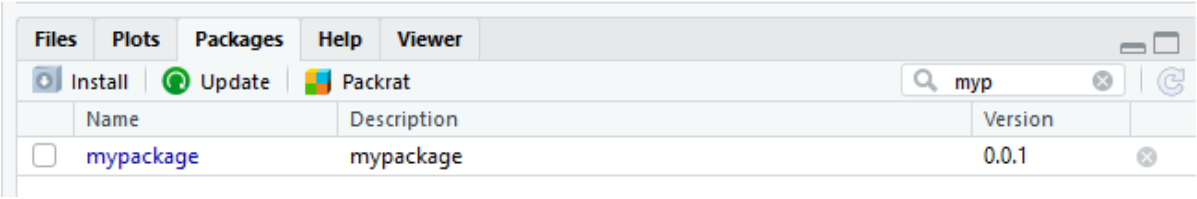
- **Title**: Your package title
- **Description**: A brief description
- **Param**: Parameters for that function; arguments
- **Return**: Values returned
- **Examples**: Examples of how to use the function
- **Export**: Desired function

Now, go to Build Tools:



Click the checkmark for Generate documentation with Roxygen. You probably want to rename your function now from hello.R to something relevant. Now, build your package by clicking Build – Clean and Rebuild.

Search for your package, and it should appear:



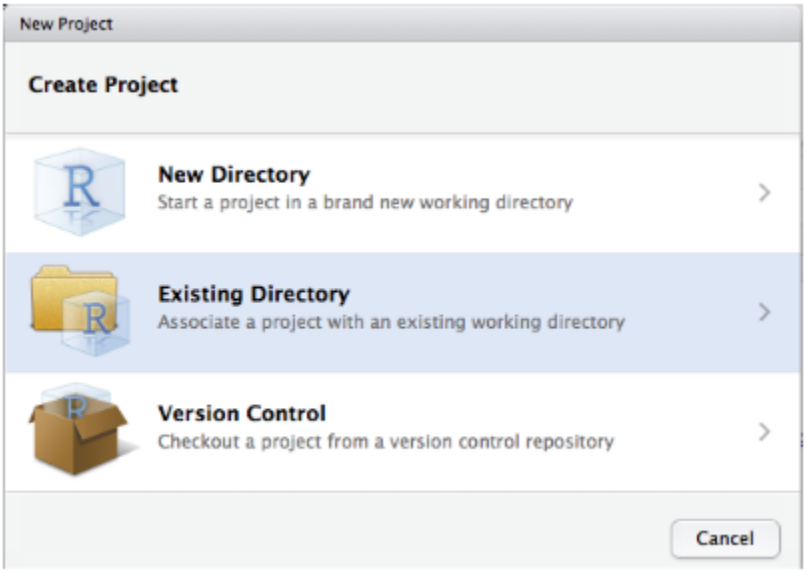
Version Control

We have two options for version control.

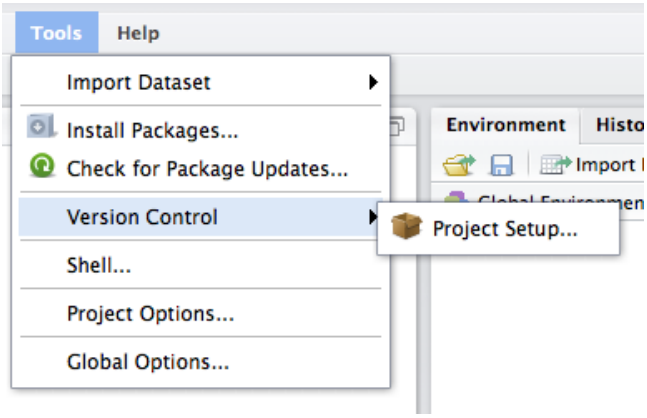
Either create the project based on an existing folder, or from scratch based on a new empty folder.

1. **Existing Work**

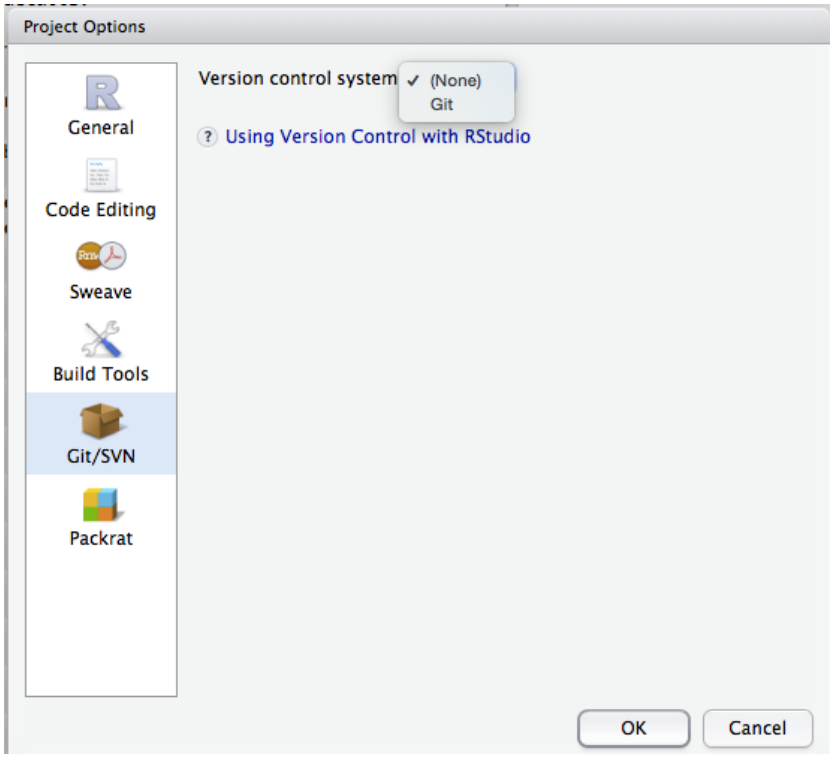
Choose Existing Directory from the wizard:



Browse to the main folder containing your data and enable version control. Go to the Tools menu -> Version Control -> Project Set:



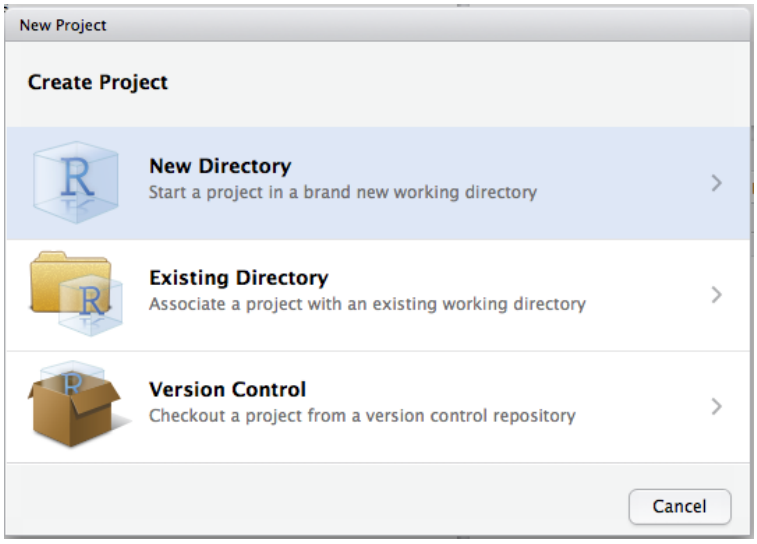
This will open a new window. Change the “None” setting by choosing Git from the drop-down menu:



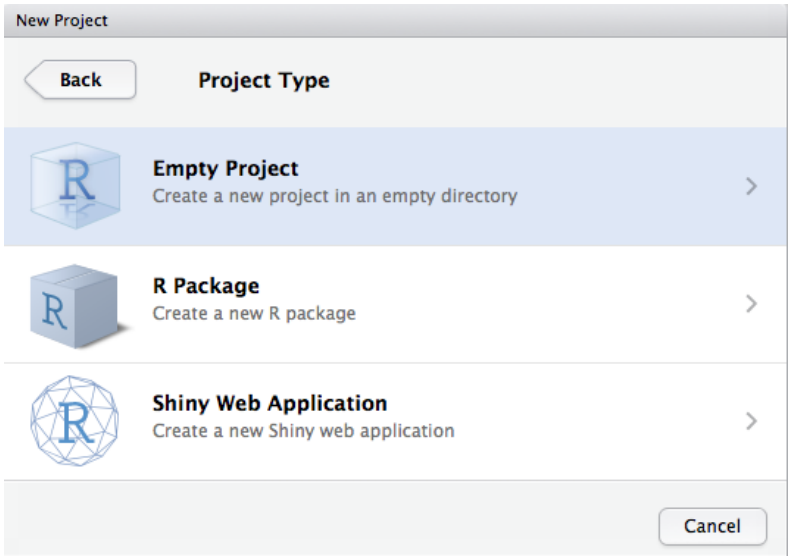
Click OK. Answer Yes to the next two pop-ups to restart RStudio and enable git on your project.

2. **New Work**

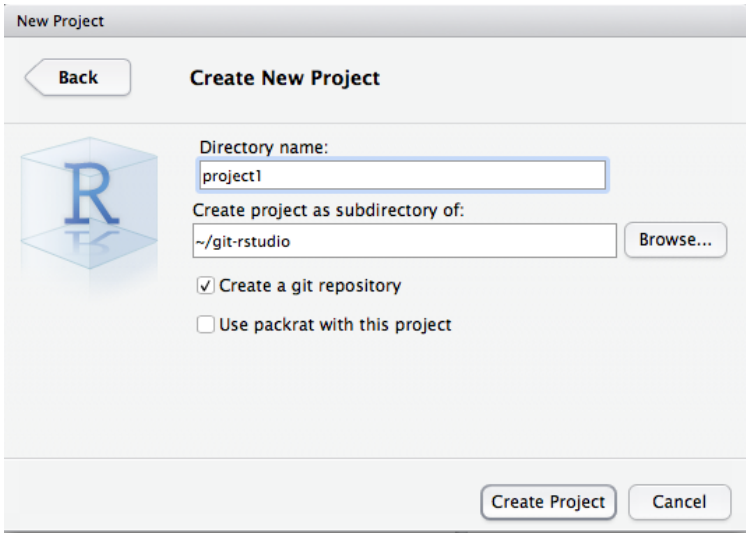
Create a new RStudio Project from the upper-right corner of the RStudio IDE window, choosing New Project. Choose New Directory:



Choose Empty Project:

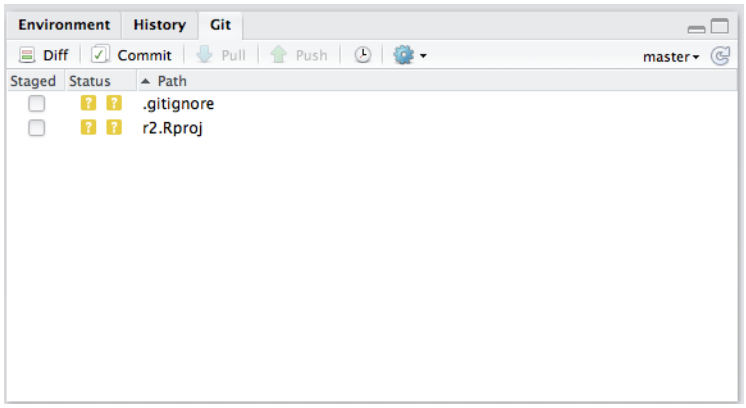


Name your folder, then browse for its location on your drive. Finally, check “Create a git repository” before clicking “Create Project.”



3. Using Git

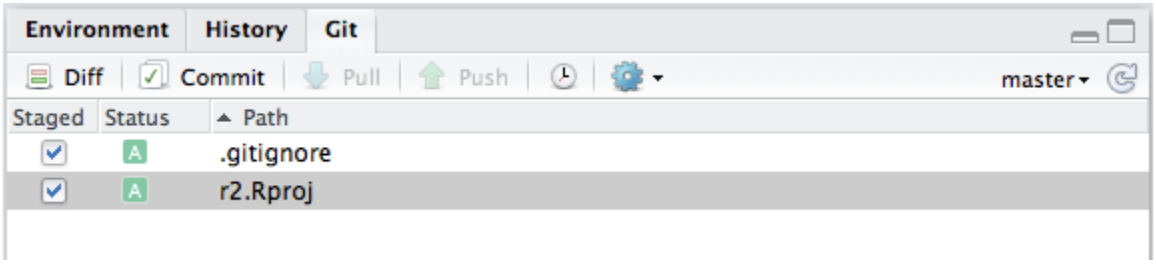
After enabling the git, you should see a “Git” tab on your interface next to the “Environment” and “History” tabs.



The .gitignore file is automatically generated by RStudio. It lists all file formats we do not want to track, here specifically the temporary files from R and RStudio. You can edit this document to add any file you don't want Git to track.

Staging

To specify which files you want to track (aka staging), check the boxes in front of the file names. This is the equivalent of the git add command:

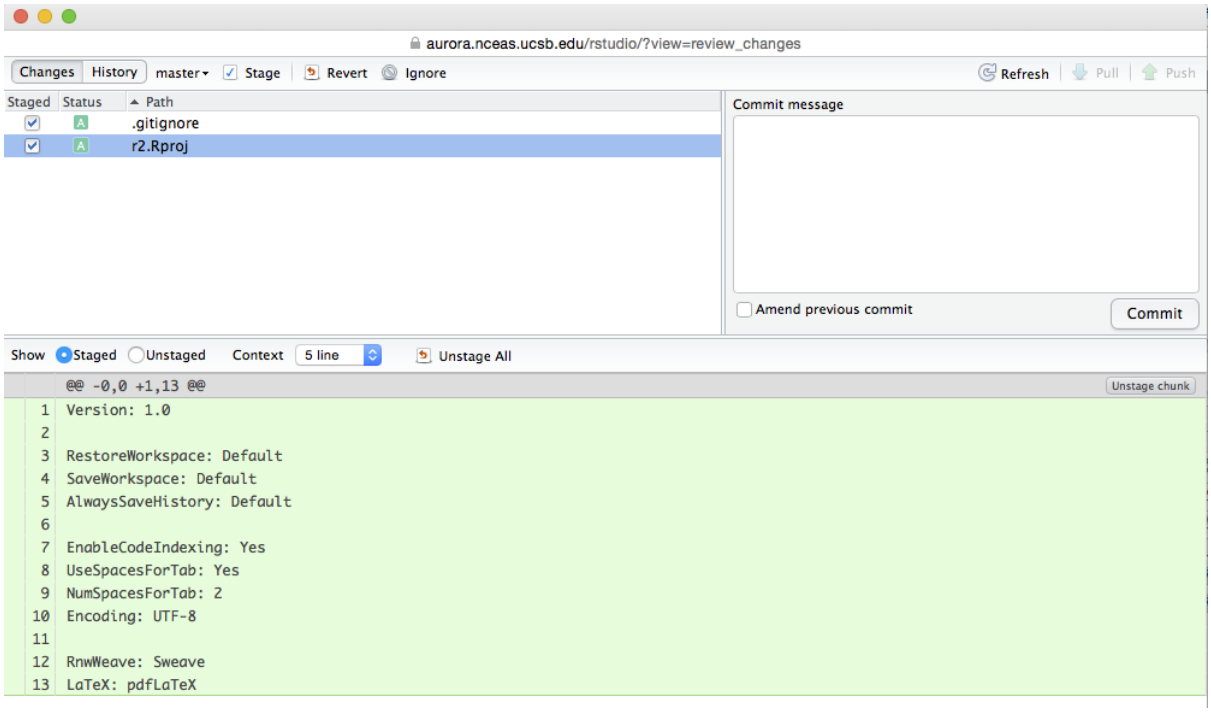


Committing

Now, you can do your first commit.

Take a snapshot of your current work status for tracked files by clicking on the Commit button (above your file names). This is the equivalent of the git commit command.

A new window should pop up:



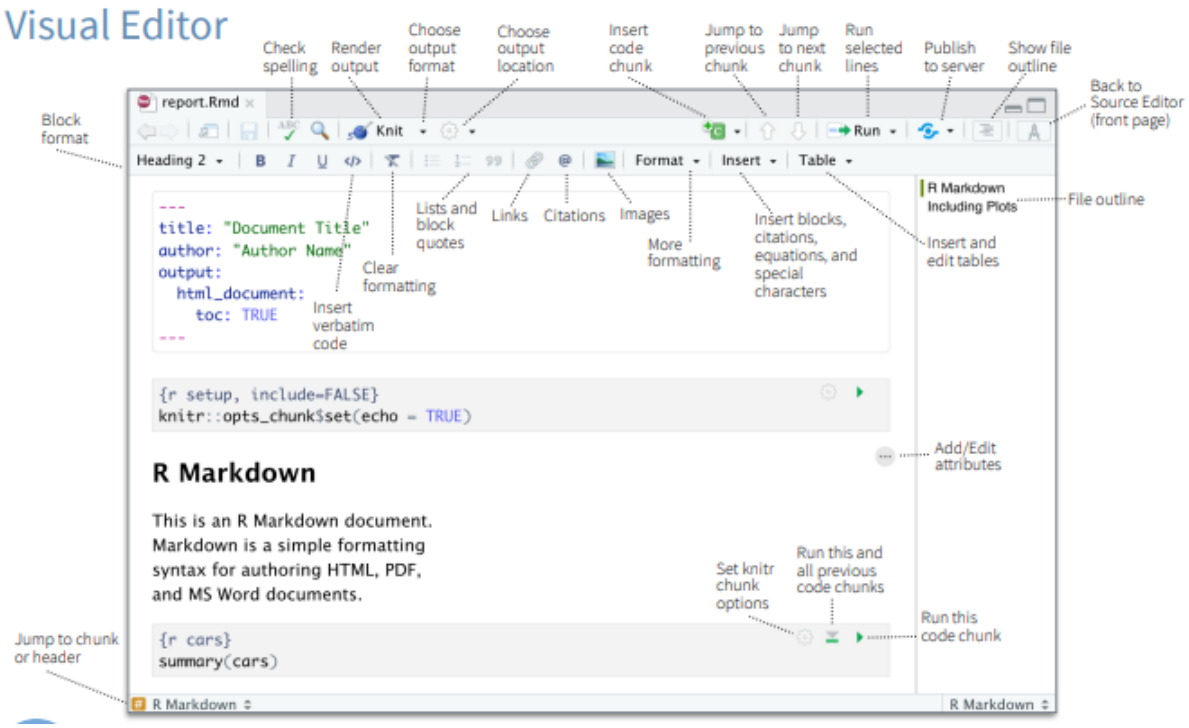
Pulling

After committing your changes, pull the latest repository version from GitHub by clicking on the pull button. If you come across any conflicts from multiple user adjustments, don't worry. Git will walk you through the necessary steps to fix the conflicts (see 3-git-advanced for more info). This is the equivalent of the git pull command.

Pushing

Once you have finished the pull process, you can click on the push button to upload your changes to GitHub and share your work with others. This is the equivalent of the git push command.

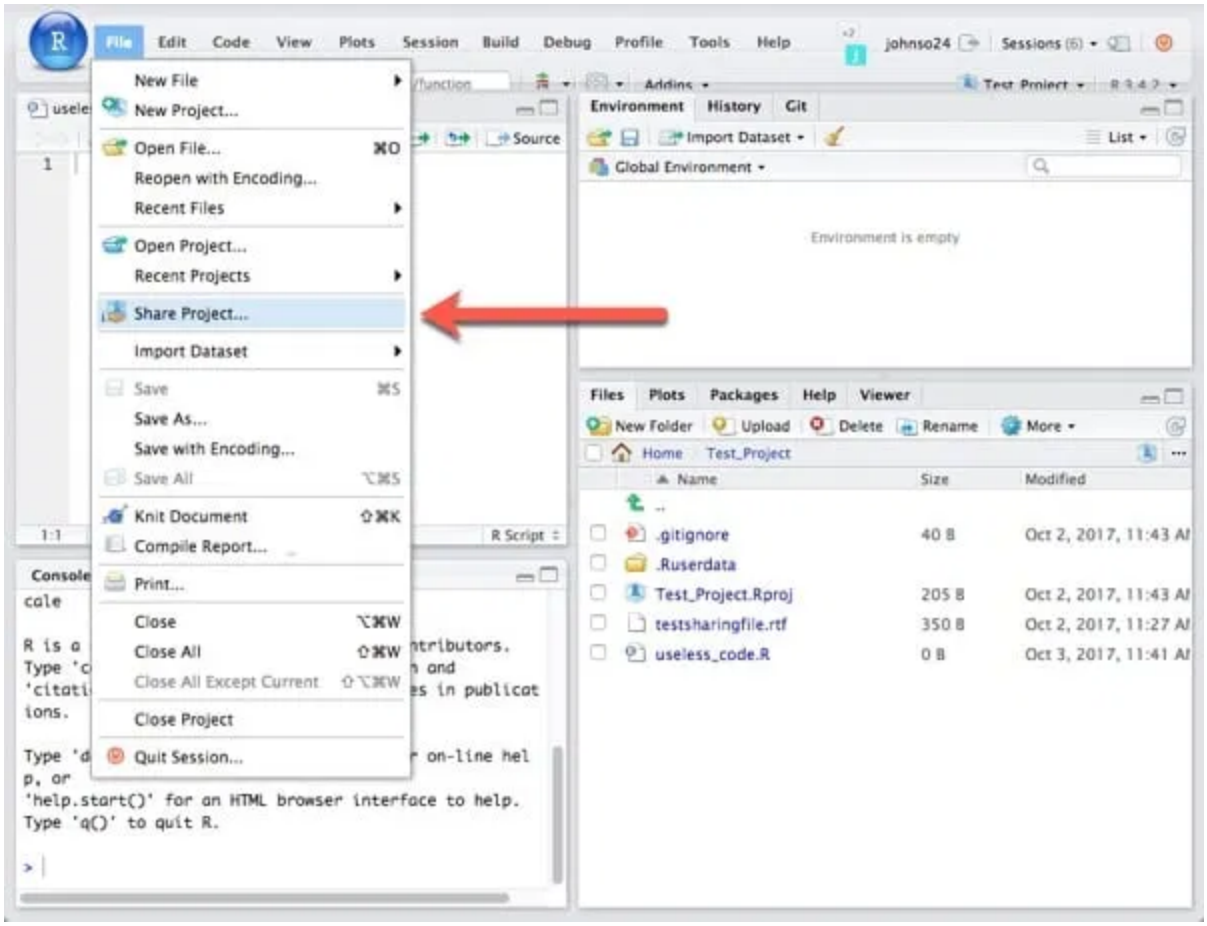
Visual Editor



Sharing Projects

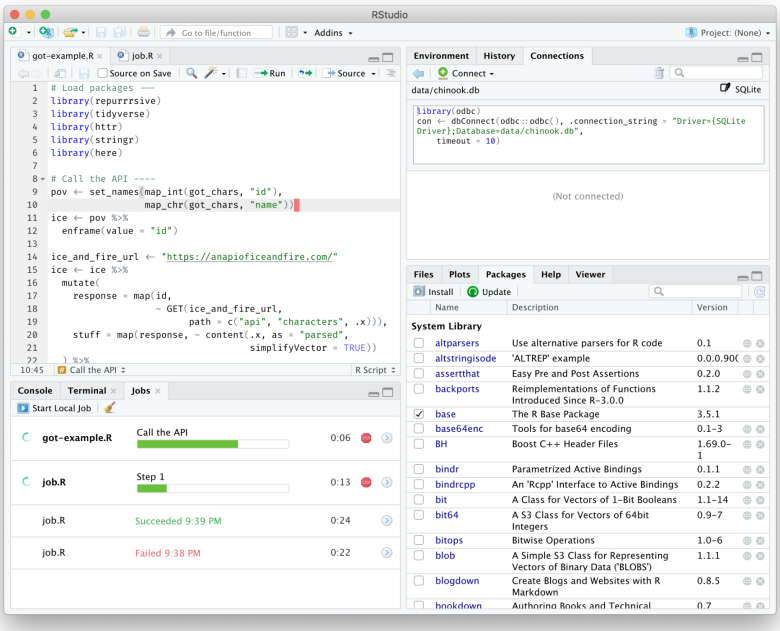
Go to File > New Project.

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each one when you reopen a project.



Running Remote Jobs

You can Run R on remote clusters (Kubernetes/Slurm) via the Job Launcher.



Keyboard Shortcuts

Description	Windows & Linux	Mac
Move cursor to Console	Ctrl+2	Ctrl+2
Clear console	Ctrl+L	Ctrl+L
Move cursor to beginning of line	Home	Cmd+Left
Move cursor to end of line	End	Cmd+Right
Navigate command history	Up/Down	Up/Down
Popup command history	Ctrl+Up	Cmd+Up
Interrupt currently executing command	Esc	Esc
Change working directory	Ctrl+Shift+H	Ctrl+Shift+H

Description	Windows & Linux	Mac
Go to File/Function	Ctrl+. [period]	Ctrl+. [period]
Move cursor to Source Editor	Ctrl+1	Ctrl+1

Toggle document outline	Ctrl+Shift+O	Cmd+Shift+O
Toggle Visual Editor	Ctrl+Shift+F4	Cmd+Shift+F4
New document (except on Chrome/Windows)	Ctrl+Shift+N	Cmd+Shift+N
New document (Chrome only)	Ctrl+Alt+Shift+N	Cmd+Shift+Alt+N
Open document	Ctrl+O	Cmd+O
Save active document	Ctrl+S	Cmd+S
Save all documents	Ctrl+Alt+S	Cmd+Option+S
Close active document (except on Chrome)	Ctrl+W	Cmd+W
Close active document (Chrome only)	Ctrl+Alt+W	Cmd+Option+W
Close all open documents	Ctrl+Shift+W	Cmd+Shift+W
Close other documents	Ctrl+Shift+Alt+W	Cmd+Option+Shift+W
Preview HTML (Markdown and HTML)	Ctrl+Shift+K	Cmd+Shift+K
Knit Document (knitr)	Ctrl+Shift+K	Cmd+Shift+K
Compile Notebook	Ctrl+Shift+K	Cmd+Shift+K
Compile PDF (TeX and Sweave)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave and Knitr)	Ctrl+Alt+I	Cmd+Option+I
Insert code section	Ctrl+Shift+R	Cmd+Shift+R
Run current line/selection	Ctrl+Enter	Cmd+Return
Run current line/selection (retain cursor position)	Alt+Enter	Option+Return
Re-run previous region	Ctrl+Alt+P	Cmd+Alt+P
Run current document	Ctrl+Alt+R	Cmd+Option+R
Run from document beginning to current line	Ctrl+Alt+B	Cmd+Option+B
Run from current line to document end	Ctrl+Alt+E	Cmd+Option+E
Run the current function definition	Ctrl+Alt+F	Cmd+Option+F
Run the current code section	Ctrl+Alt+T	Cmd+Option+T

Run previous Sweave/Rmd code	Ctrl+Shift+Alt+P	Cmd+Shift+Option+P
Run the current Sweave/Rmd chunk	Ctrl+Alt+C	Cmd+Option+C
Run the next Sweave/Rmd chunk	Ctrl+Alt+N	Cmd+Option+N
Source a file	Ctrl+Alt+G	Ctrl+Option+G
Source the current document	Ctrl+Shift+S	Cmd+Shift+S
Source the current document (with echo)	Ctrl+Shift+Enter	Cmd+Shift+Return
Send current line/selection to terminal	Ctrl+Alt+Enter	Cmd+Option+Return
Fold Selected	Alt+L	Cmd+Option+L
Unfold Selected	Shift+Alt+L	Cmd+Shift+Option+L
Fold All	Alt+O	Cmd+Option+O
Unfold All	Shift+Alt+O	Cmd+Shift+Option+O
Go to line	Shift+Alt+G	Cmd+Shift+Option+G
Jump to	Shift+Alt+J	Cmd+Shift+Option+J
Expand selection	Ctrl+Shift+Up	Ctrl+Option+Shift+Up
Shrink selection	Ctrl+Shift+Down	Ctrl+Option+Shift+Down
Next section	Ctrl+PgDn	Cmd+PgDn
Previous section	Ctrl+PgUp	Cmd+PgUp
Split into lines	Ctrl+Alt+A	Ctrl+Option+A
Edit lines from start	Ctrl+Alt+Shift+A	Ctrl+Shift+Option+A
Switch to tab	Ctrl+Shift+. [period]	Ctrl+Shift+. [period]
Previous tab	Ctrl+F11	Ctrl+F11
Previous tab (desktop)	Ctrl+Shift+Tab	Ctrl+Shift+Tab
Next tab	Ctrl+F12	Ctrl+F12
Next tab (desktop)	Ctrl+Tab	Ctrl+Tab
First tab	Ctrl+Shift+F11	Ctrl+Shift+F11
Last tab	Ctrl+Shift+F12	Ctrl+Shift+F12
Navigate back	Ctrl+F9	Cmd+F9
Navigate forward	Ctrl+F10	Cmd+F10
Extract function from selection	Ctrl+Alt+X	Cmd+Option+X

Extract variable from selection	Ctrl+Alt+V	Cmd+Option+V
Reindent lines	Ctrl+I	Cmd+I
Comment/uncomment current line/selection	Ctrl+Shift+C	Cmd+Shift+C
Reflow Comment	Ctrl+Shift+/	Cmd+Shift+/
Reformat Selection	Ctrl+Shift+A	Cmd+Shift+A
Show Diagnostics	Ctrl+Shift+Alt+D	Cmd+Shift+Option+D
Transpose Letters	No shortcut	Ctrl+T
Move Lines Up/Down	Alt+Up/Down	Option+Up/Down
Copy Lines Up/Down	Shift+Alt+Up/Down	Cmd+Option+Up/Down
Jump to Matching Brace/Paren	Ctrl+P	Ctrl+P
Expand to Matching Brace/Paren	Ctrl+Shift+Alt+E	Ctrl+Shift+E
Add Cursor Above Current Cursor	Ctrl+Alt+Up	Ctrl+Option+Up
Add Cursor Below Current Cursor	Ctrl+Alt+Down	Ctrl+Option+Down
Move Active Cursor Up	Ctrl+Alt+Shift+Up	Ctrl+Option+Shift+Up
Move Active Cursor Down	Ctrl+Alt+Shift+Down	Ctrl+Option+Shift+Down
Find and Replace	Ctrl+F	Cmd+F
Find Next	Win: F3, Linux: Ctrl+G	Cmd+G
Find Previous	Win: Shift+F3, Linux: Ctrl+Shift+G	Cmd+Shift+G
Use Selection for Find	Ctrl+F3	Cmd+E
Replace and Find	Ctrl+Shift+J	Cmd+Shift+J
Find in Files	Ctrl+Shift+F	Cmd+Shift+F
Check Spelling	F7	F7
Rename Symbol in Scope	Ctrl+Alt+Shift+M	Cmd+Option+Shift+M
Insert Roxygen Skeleton	Ctrl+Alt+Shift+R	Cmd+Option+Shift+R

Editing (Console and Source)

Description	Windows & Linux	Mac

Undo	Ctrl+Z	Cmd+Z
Redo	Ctrl+Shift+Z	Cmd+Shift+Z
Cut	Ctrl+X	Cmd+X
Copy	Ctrl+C	Cmd+C
Paste	Ctrl+V	Cmd+V
Select All	Ctrl+A	Cmd+A
Jump to Word	Ctrl+Left/Right	Option+Left/Right
Jump to Start/End	Ctrl+Home/End or Ctrl+Up/Down	Cmd+Home/End or Cmd+Up/Down
Delete Line	Ctrl+D	Cmd+D
Select	Shift+[Arrow]	Shift+[Arrow]
Select Word	Ctrl+Shift+Left/Right	Option+Shift+Left/Right
Select to Line Start	Alt+Shift+Left	Cmd+Shift+Left
Select to Line End	Alt+Shift+Right	Cmd+Shift+Right
Select Page Up/Down	Shift+PageUp/PageDown	Shift+PageUp/Down
Select to Start/End	Ctrl+Shift+Home/End or Shift+Alt+Up/Down	Cmd+Shift+Up/Down
Delete Word Left	Ctrl+Backspace	Option+Backspace or Ctrl+Option+Backspace
Delete Word Right	No shortcut	Option+Delete
Delete to Line End	No shortcut	Ctrl+K
Delete to Line Start	No shortcut	Option+Backspace
Indent	Tab (at beginning of line)	Tab (at beginning of line)
Outdent	Shift+Tab	Shift+Tab
Yank line up to cursor	Ctrl+U	Ctrl+U
Yank line after cursor	Ctrl+K	Ctrl+K
Insert currently yanked text	Ctrl+Y	Ctrl+Y
Insert assignment operator	Alt+-	Option+-
Insert pipe operator	Ctrl+Shift+M	Cmd+Shift+M

Show help for function at cursor	F1	F1
Show source code for function at cursor	F2	F2
Find usages for symbol at cursor (C++)	Ctrl+Alt+U	Cmd+Option+U

Completions (Console and Source)

Description	Windows & Linux	Mac
Attempt completion	Tab or Ctrl+Space	Tab or Cmd+Space
Navigate candidates	Up/Down	Up/Down
Accept selected candidate	Enter, Tab, or Right	Enter, Tab, or Right
Dismiss completion popup	Esc	Esc

Views

Description	Windows & Linux	Mac
Move focus to Source Editor	Ctrl+1	Ctrl+1
Zoom Source Editor	Ctrl+Shift+1	Ctrl+Shift+1
Add Source Column	Ctrl+F7	Cmd+F7
Move focus to Console	Ctrl+2	Ctrl+2
Zoom Console	Ctrl+Shift+2	Ctrl+Shift+2
Move focus to Help	Ctrl+3	Ctrl+3
Zoom Help	Ctrl+Shift+3	Ctrl+Shift+3
Move focus to Terminal	Alt+Shift+M	Shift+Option+M
Show History	Ctrl+4	Ctrl+4
Zoom History	Ctrl+Shift+4	Ctrl+Shift+4
Show Files	Ctrl+5	Ctrl+5
Zoom Files	Ctrl+Shift+5	Ctrl+Shift+5
Show Plots	Ctrl+6	Ctrl+6
Zoom Plots	Ctrl+Shift+6	Ctrl+Shift+6

Show Packages	Ctrl+7	Ctrl+7
Zoom Packages	Ctrl+Shift+7	Ctrl+Shift+7
Show Environment	Ctrl+8	Ctrl+8
Zoom Environment	Ctrl+Shift+8	Ctrl+Shift+8
Show Viewer	Ctrl+9	Ctrl+9
Zoom Viewer	Ctrl+Shift+9	Ctrl+Shift+9
Show Git/SVN	Ctrl+F1	Cmd+F1
Zoom Git/SVN	Ctrl+Shift+F1	Ctrl+Shift+F1
Show Build	Ctrl+F2	Cmd+F2
Zoom Build	Ctrl+Shift+F2	Ctrl+Shift+F2
Show Connections	Ctrl+F5	No shortcut
Zoom Connections	Ctrl+Shift+F5	Ctrl+Shift+F5
Show Find in Files Results	Ctrl+F6	Cmd+F6
Zoom Tutorial	Ctrl+Shift+F6	Ctrl+Shift+F6
Sync Editor & PDF Preview	Ctrl+F8	Cmd+F8
Global Options	No shortcut	Cmd+, [comma] (Chrome, Desktop), Option+Cmd+, [comma] (Safari, Firefox)
Project Options	No shortcut	Shift+Cmd+, [comma]

Build

Description	Windows & Linux	Mac
Build and Reload	Ctrl+Shift+B	Cmd+Shift+B
Load All (devtools)	Ctrl+Shift+L	Cmd+Shift+L
Test Package (Desktop)	Ctrl+Shift+T	Cmd+Shift+T
Test Package (Web)	Ctrl+Alt+F7	Cmd+Option+F7
Check Package	Ctrl+Shift+E	Cmd+Shift+E
Document Package	Ctrl+Shift+D	Cmd+Shift+D

Debug

Description	Windows & Linux	Mac

Toggle Breakpoint	Shift+F9	Shift+F9
Execute Next Line	F10	F10
Step Into Function	Shift+F4	Shift+F4
Finish Function/Loop	Shift+F7	Shift+F7
Continue	Shift+F5	Shift+F5
Stop Debugging	Shift+F8	Shift+F8

Plots

Description	Windows & Linux	Mac
Previous plot	Ctrl+Alt+F11	Cmd+Option+F11
Next plot	Ctrl+Alt+F12	Cmd+Option+F12

Git/SVN

Description	Windows & Linux	Mac
Diff active source document	Ctrl+Alt+D	Ctrl+Option+D
Commit changes	Ctrl+Alt+M	Ctrl+Option+M
Scroll diff view	Ctrl+Up/Down	Ctrl+Up/Down
Stage/Unstage (Git)	Spacebar	Spacebar
Stage/Unstage and move to next (Git)	Enter	Return

Session

Description	Windows & Linux	Mac
Quit Session (desktop only)	Ctrl+Q	Cmd+Q
Restart R Session	Ctrl+Shift+F10	Cmd+Shift+F10

Terminal

Description	Windows & Linux	Mac
New Terminal	Alt+Shift+R	Shift+Option+R
Move Focus to Terminal	Alt+Shift+M	Shift+Option+M
Previous Terminal	Alt+Shift+F11	Shift+Option+F11
Next Terminal	Alt+Shift+F12	Shift+Option+F12