

Introduction to R and RStudio

Table of Contents

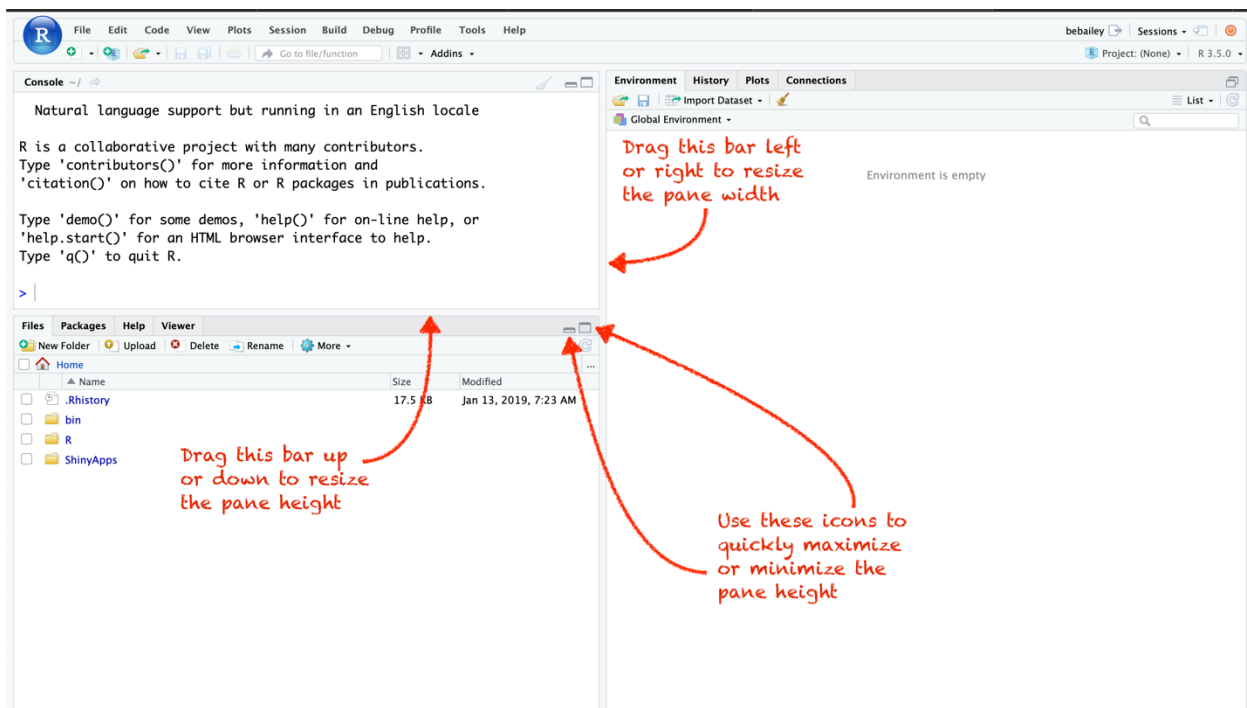
UNDERSTANDING RSTUDIO PANES.....	2
SETTING PREFERRED GLOBAL OPTIONS.....	3
1. <i>General options</i>	3
2. <i>Code options</i>	3
3. <i>Appearance options</i>	4
4. <i>Pane Layout</i>	4
WORKING IN THE CONSOLE.....	5
WORKING WITH R SCRIPTS.....	7
WORKING WITH R MARKDOWN DOCUMENTS.....	8
<i>The YAML header</i>	9
<i>The “setup” code chunk</i>	9
<i>Installing and loading R packages</i>	10
<i>Word processing and formatting text with Markdown</i>	11
<i>Working with code chunks and output</i>	12
<i>Coding and coding style</i>	13
<i>Answering questions in provided Rmd documents</i>	14

Understanding RStudio Panes

When you first open RStudio, you will see three or four panes in some order:

- ▶ **Console/Jobs:** where we actively code in R
- ▶ **Environment/History/Connections/Tutorial:** shows objects currently loaded in the environment and console history
- ▶ **Files/Plots/Packages/Help/Viewer:** navigate directories, view plots created in console, and get help with R code
- ▶ **Source** (may not be visible): where we work on reproducible R or R Markdown files.

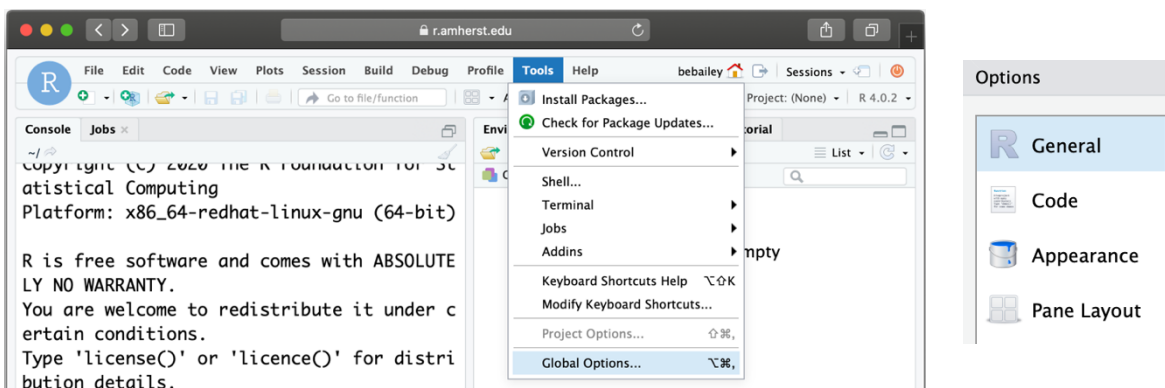
You can resize the panes to suit your needs as you work. See the next section for rearranging the panes and setting other useful defaults for RStudio.



Setting preferred global options

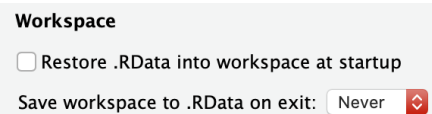
The default settings in RStudio are not ideal. This section will get you set up with global options that will make it easier to work in RStudio.

Go to the **Tools** menu at the top of the screen. Select **Global Options** at the bottom of the list. We will focus on the first four options listed: General, Code, Appearance, and Pane Layout.



1. General options

Under General options, go to the **Workspaces** section and **uncheck** the box next to the option that reads "Restore .RData into workspace at startup."

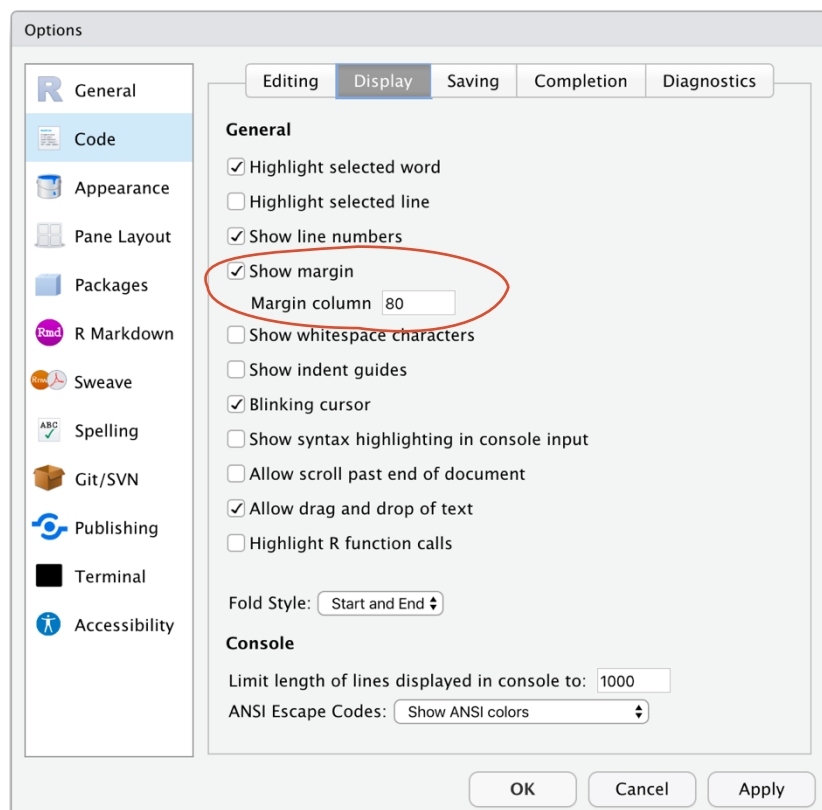


Then change the option next to "Save workspace to .RData on exit" to **Never**.

2. Code options

Under code options, click on the **Display** tab. **Check** the box next to **Show margin**, and set the **Margin Column** somewhere between 65 and 80 (max).

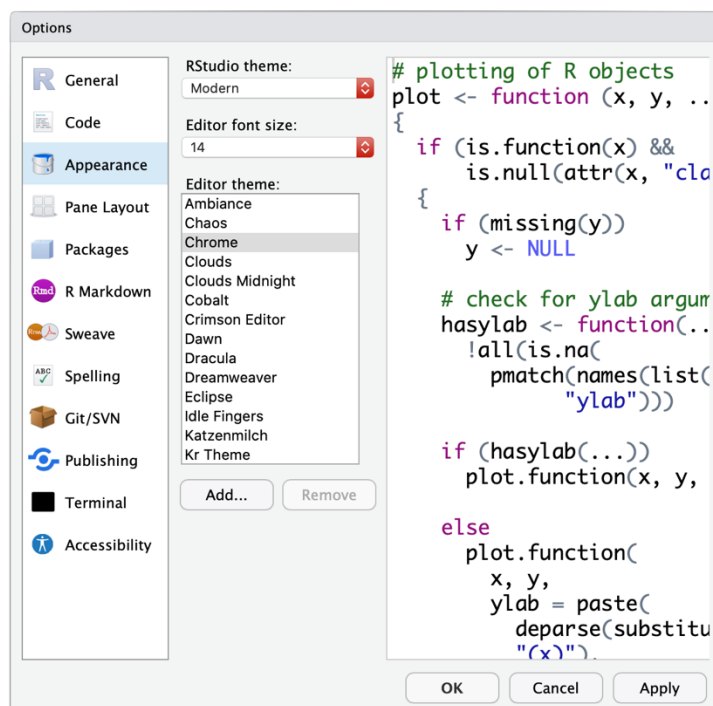
Show Margin adds a vertical line to R Markdown files indicating where code will start to fall outside of the grey box when you knit to pdf. The margin column may need to be adjusted slightly depending on the font and style settings of the Markdown document.



3. Appearance options

There are no important default settings here, but this is where you can change the font size and the editor theme (background and syntax highlighting colors). Feel free to play around with the options here and set them to something that pleases you! I recommend one of the following:

Chrome	Tomorrow Night
Cobal	Tomorrow Night 80s
Dreamweaver	Tomorrow Night Blue
SQL Server	Tomorrow Night Bright
Textmate	Twilight
Tomorrow	



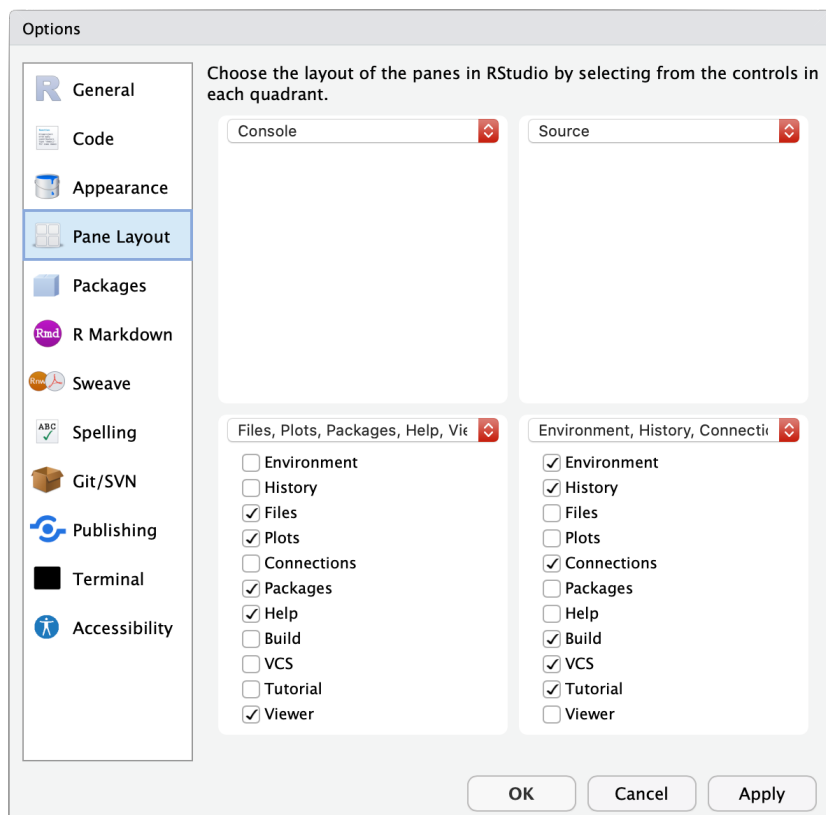
4. Pane Layout

We spend most of our time bouncing back and forth between the Console and the Source panes, so it is much easier to work with these panes located side by side.

My preferred setup is shown here. Use the drop-down menus in each of the four panels to select the pane layout that works for you.

Use the drop-down menus in each of the four panels to select the pane layout that works for you.

When you are done making changes, click **Apply** and then **OK** to exit the Options window.



Working in the console

This document contains material modified from Amelia McNamera's *Introduction to R & RStudio* short course.

R is a programming language that uses operators (e.g., +, -, /, *, ^, <-, %>%) and functions (e.g., sqrt(), mean(), lm(), t.test(), gf_histogram()) to perform calculations, conduct statistical procedures, display data, and more!

The console works a lot like an advanced calculator (think TI-89). You enter commands on the line that begins with a > (known as the prompt). When you hit *Enter*, R will run, or *evaluate*, your command and display output below it. When R is done running your command, another prompt will appear and R will be ready for you to enter another command.

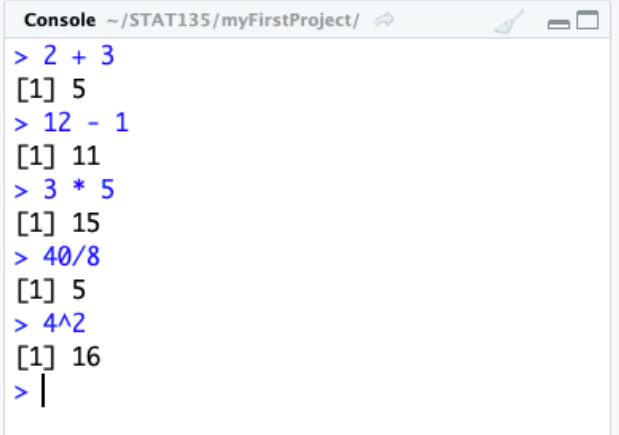
Note that R often displays an index (e.g., [1]) next to the output. It is not so useful in the case of one line of output, but becomes more useful when you have *vectors*, *matrices*, *dataframes*, *lists* or other objects. More on that later.

R can do all the things that your calculator can do. It even has a memory like a calculator: press the *up* arrow on your keyboard and you should get the last line of code you entered. Press it again and it will go another line back. You can go up or down through memory (and hit *Escape* if you want to return to the blank prompt line).

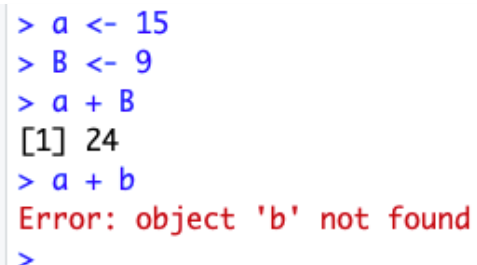
You can save objects to the environment using the arrow operator (<-), for example the number 9 is saved as the object B in the image on the right. Note that R is picky about capitalization, though (i.e., b is different from B).

R does not usually care about spacing (i.e., 3^2 is the same as 3 ^ 2).

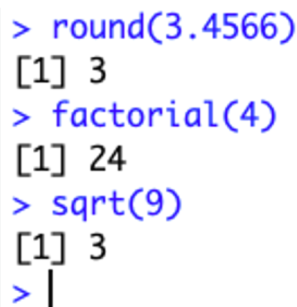
Base R has built in functions that allow you to do more sophisticated manipulations. Functions are composed of *function names* (e.g., round, sqrt, sum, factorial), followed by comma-separated *arguments* that are enclosed within parentheses.



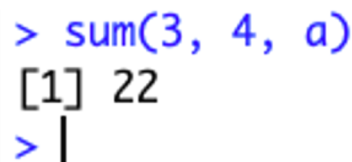
```
Console ~/STAT135/myFirstProject/
> 2 + 3
[1] 5
> 12 - 1
[1] 11
> 3 * 5
[1] 15
> 40/8
[1] 5
> 4^2
[1] 16
> |
```



```
> a <- 15
> B <- 9
> a + B
[1] 24
> a + b
Error: object 'b' not found
>
```



```
> round(3.4566)
[1] 3
> factorial(4)
[1] 24
> sqrt(9)
[1] 3
> |
```



```
> sum(3, 4, a)
[1] 22
> |
```

R always works from the innermost parentheses to the outermost, and R will prompt you with a `+` instead of a `>` if you don't finish your command (this happens when you miss a closing parenthesis or you end a line with an operator). When the `+` prompt appears, either finish your command on the next line, or hit *Escape* to bail on the command and return to a blank prompt.

```
> factorial(round(3.564) + 1
+
> factorial(round(3.564) + 1
+ )
[1] 120
> |
```

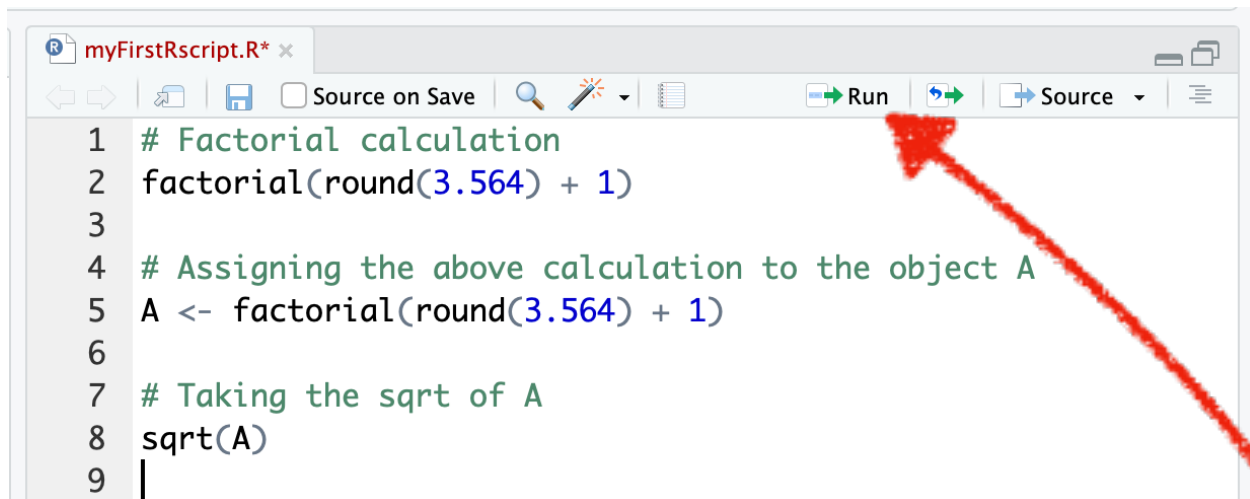
The R Console is useful for quick calculations, testing code, or using dynamic functions that do not work in R Markdown files (e.g., `View(dataset)`). However, you cannot save the work in the console for later use. This is why we use *R Scripts* or *R Markdown documents*!

Working with R Scripts

You can create an R Script by going to **File → New File → R Script**.

R scripts contain only *R code* (commands that you would like to execute) and *R comments*.

Comments in R code begin with a **#** symbol, and they tell R to ignore everything that follows the **#** symbol on that line. Comments are a great way of *documenting* your code to remind your future self (or another user) of what you were doing.



```
1 # Factorial calculation
2 factorial(round(3.564) + 1)
3
4 # Assigning the above calculation to the object A
5 A <- factorial(round(3.564) + 1)
6
7 # Taking the sqrt of A
8 sqrt(A)
9 |
```

Some things to note:

- The R Script is not interactive like the console—it's just a place to store and document code. You might use an R Script to workshop some code or for storing longer sets of commands that you load (or *source*) into a reproducible R Markdown file.
- You can run a line (or highlighted selection) of R code by putting your cursor on the line (or highlighting the code you want to run) and either clicking the Run button or by pressing *Cmd + Enter* (or *Ctrl + Enter* on Windows) on your keyboard. RStudio copies the line or selection to the console and evaluates the code there.
- The filename will display in **red with a * at the end** if the file has not been saved since changes have been made. Click the floppy disk or press *Cmd + S* (or *Ctrl + S*) to save regularly.

Working with R Markdown Documents

Nearly everything we work on in RStudio will be via R Markdown documents. Markdown documents weave together formatted text (like a Word document), R code (like an R script), and the output of the evaluated R code (like what gets displayed in the console or plot pane after a command is run).

The filename will display in **red with a * at the end** if the file has not been saved since changes have been made. Click the floppy disk or press *Cmd + S* (or *Ctrl + S*) to **save regularly**.

We **Knit** the Rmd file to translate and compile the contents into a beautiful **pdf**. You can Knit the document by clicking Knit or pressing *Cmd + Shift + K* (or *Ctrl + Shift + K*).

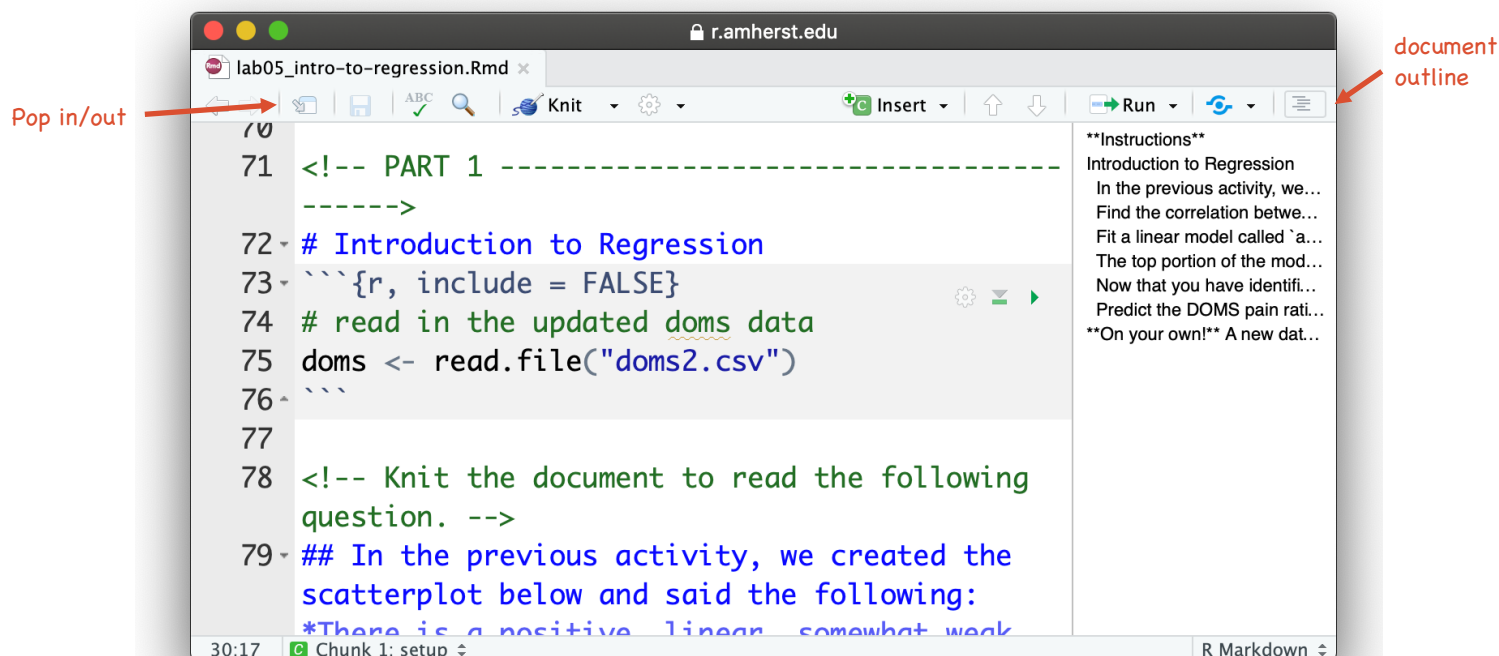
Get used to saving and knitting obsessively (hence the keyboard shortcuts)!

The server crashes often and you don't want to lose any unsaved work. You should knit the document after any coding or Markdown formatting to make sure the document compiles correctly and without error. Waiting til the very end to knit makes it harder to find and correct those errors.

Some R Markdown documents can get long and unwieldy. In addition to syntax highlighting, RStudio has a number of features that make it easier to navigate.

One is the **document outline**, which can be accessed by clicking on the outline icon (📄) in the top right corner of the Source pane. You can then click on an item in the outline to jump down to that section of the document.

It's also useful sometimes to **pop out** just the source pane in a window of its own (maybe to have the Source and the knitted pdf side by side while you edit formatting). You can pop out the Source pane by clicking the pop out icon (📄), just make sure to pop it back in (📄) before ending your work session.



The YAML header

The top of the document always contains the **YAML header**, which starts and ends with three dashes (---) as shown in Lines 1 and 11. I tend to have longer YAML headers to customize the final look of the knitted pdf.

In most cases, you will only need to change the author line (circled on Line 3 in this image) and perhaps the default figure height and figure width (in inches; shown on Lines 8 and 9, respectively). **You should ignore everything else unless you know what you are doing.**

```

1 ---
2 title: "Problem Set 7"
3 author: "Your Name"
4 date: "October 31"
5 classoption: fleqn
6 output:
7   pdf_document:
8     fig_height: 2
9     fig_width: 3.5
10    number_section: true
11 ---
12

```

The “setup” code chunk

The first code chunk we come to in a Markdown document typically sets up defaults for the code in the document and loads all packages necessary to run the code used in the document. Many of the functions we use in STAT 135 and STAT 230 come from the **mosaic** package (which also loads the **ggformula** package for plots).

Unless you are loading additional packages (or know what you are doing), **there should be no need to edit the first code chunk.**

This first code chunk usually has the code chunk option `include = FALSE`. This

single option tells RStudio to evaluate the code in the code chunk (`eval = TRUE`) but not to display in the knitted document the code chunk or any output, messages, or warnings associated with evaluating the code.

```

20
21- ```{r setup, include=FALSE}
22 # load packages
23 require(mosaic)
24 require(kableExtra)
25
26 # set some code chunk defaults
27 knitr::opts_chunk$set(
28   # display code as typed
29   tidy = F,
30   # slightly smaller font for code
31   size = "small",
32   # indent plain R output
33   comment = "\t",
34   # center figures
35   fig.align = "center",
36   # suppress warnings and messages
37   warning = F, message = F)
38 # default to black & white plot theme
39 theme_set(theme_classic())
40 # slower to display scientific notation; "NA"s knit as blank spaces
41 options(scipen = 1, knitr.kable.NA = '')
42- ```
43

```

Installing and loading R packages

An R package is a bundle of R functions, data, and documentation. Packages allow us to do more sophisticated tasks in a more efficient manner.

Most R packages are available for download from the Comprehensive R Archive Network ([CRAN](https://cran.r-project.org/)). Packages only need to be installed once to be used indefinitely and then updated occasionally thereafter. **You should not install any packages if you are working on the RStudio Server** unless explicitly told otherwise, but you will need to install packages on your own computer if you have downloaded R and RStudio on your device.

You must load packages into R (usually at the top of each Markdown file or at the start of each R session) using `library(packageName)` or `require(packageName)` in order to use any of their corresponding functions, datasets, or other content. Here are two metaphors to describe how packages work.

Library metaphor:

Imagine the packages on CRAN as books in a bookstore. If we like the functions available in a particular package, we first need to buy the book from the bookstore and add it to our personal library (install packages using `install.packages('packageName')`). Then anytime we want to use the function, we need to grab the book off the shelf (load packages using `library(packageName)` or `require(packageName)`).

App metaphor:

Imagine the packages on CRAN as apps in an app store. If we like the functions available in a particular package, we first need to download the app from the app store so that it's available on our device (install packages using `install.packages('packageName')`). Then anytime we want to use the functions in a particular package, we need to first open the app (load packages using `library(packageName)` or `require(packageName)`).

Again, unless a package gets deleted from the system, it only needs to be downloaded and installed once ever and then occasionally updated (`update.packages()`).

Once you have a package installed on your system, you need to load (or *require*) it any time you want to use the functions or datasets it contains. There is no harm in loading packages that you don't end up using in your document, but your code won't work if you try to use functions from packages that you didn't first load from your library.

Word processing and formatting text with Markdown

If you are not in the YAML header or a code chunk, then you are working with plain text (like a Word document or Google Doc) that can be formatted or stylized by starting and ending the formatted text with the appropriate symbols. See the [R Markdown Reference Guide \(Simple\)](#) below for help.

Markdown is a simple way to format text that looks great on any device. It doesn't do anything fancy like change the font size, color, or type — just the essentials, using keyboard symbols you already know.

[TRY OUR 10 MINUTE MARKDOWN TUTORIAL](#)

Type	Or	... to Get
<i>*Italic*</i>	<code>_Italic_</code>	<i>Italic</i>
Bold	<code>__Bold__</code>	Bold
# Heading 1	Heading 1 =====	Heading1
## Heading 2	Heading 2 -----	Heading2
[Link](http://a.com)	[Link][1] : [1]: http://b.org	Link
![Image](http://url/a.png)	![Image][1] : [1]: http://url/b.jpg	
> Blockquote		Blockquote
* List * List * List	- List - List - List	• List • List • List
1. One 2. Two 3. Three	1) One 2) Two 3) Three	1. One 2. Two 3. Three
Horizontal Rule ---	Horizontal Rule ***	Horizontal Rule
`Inline code` with backticks		Inline code with backticks
``` # code block print '3 backticks or' print 'indent 4 spaces' ```	```# code block ...print '3 backticks or' ...print 'indent 4 spaces' ```	# code block print '3 backticks or' print 'indent 4 spaces'

## Working with code chunks and output

Code chunks in R Markdown start and end with three back ticks (```). You can insert a code chunk manually by typing the ticks yourself, or you can click the **Insert** button (📄 Insert) at the top of the Source pane. We will code only in R, but there are other programming languages you can include in code chunks.

When you insert a code chunk, you must indicate the code language (r). After that, you can specify a unique descriptive label and then list any desired chunk options separated by commas. Common chunk options are listed below (additional options available by clicking the gear icon):

- ▶ **echo** display code in knitted document (default: TRUE),
- ▶ **eval** evaluate code (default: TRUE),
- ▶ **fig.width** and **fig.height** in inches (only specify if changing from default)

When you are working with a particular chunk, you can quickly run all the code in that chunk by pressing the **Run** icon (▶). If you need to re-run all previous code chunks before running the current one, you can press the **Run above** icon (⬆▶).

To declutter your workspace, you can also pop out, minimize, or close any output in the document.

The screenshot shows the RStudio Source Editor with a code chunk for a scatter plot. The code is as follows:

```

86
87
88
89 `r domsAgePlot, echo = FALSE, fig.width = 6, fig.height = 3.5
90 # graph of DOMS vs age
91 gf_point(data = doms, DOMS ~ age,
92 alpha = 0.65,
93 size = 3) %>%
94 # custom labels
95 gf_labs(x = "Age (years)",
96 y = "DOMS Pain Index") %>%
97 # custom axes lengths
98 gf_lims(x = c(15, 50))
99 `

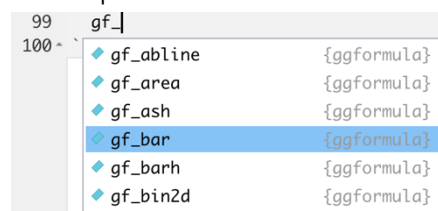
```

The plot shows DOMS Pain Index (y-axis, 0.0 to 10.0) versus Age (years) (x-axis, 15 to 50). The plot area has window control icons (pop out, minimize, close) at the bottom right.

## Coding and coding style

You can run a line (or highlighted selection) of R code by putting your cursor on the line (or highlighting the code you want to run) and either clicking the Run button or by pressing *Cmd + Enter* (or *Ctrl + Enter* on Windows) on your keyboard. RStudio copies the line or selection to the console and evaluates the code there.

RStudio makes coding and formatting code easier: it automatically indents code when we start a new line, and it will suggest code that you can choose to complete by scrolling through the list and hitting *Tab*.



When it comes to style, there are three things to pay attention to for now:

1. **Commenting your code:** Comments in R code begin with a *#* symbol, and they tell R to ignore everything that follows the *#* symbol on that line. Comments are a great way of *documenting* your code to remind your future self (or someone else) what you were attempting to do with the code. They can also be used to keep non-working code from running without deleting it entirely.
2. **Spacing for readability:** Good practice is to include space around each word, operator, and after each comma. (See [tidy syntax principles](#) if you want to learn more.)
3. **Starting new lines for the knitted pdf:** If a line of code is too long, it will run outside of the code box in the knitted pdf (and possibly off the page). You can change the Global Options to **Show a Margin** line in the .Rmd file to indicate the point at which the paper margin starts (see the preferred Global Options). Any text beyond the vertical line will go outside the grey box in the knitted document (and potentially run off the page entirely).

### PROBLEMATIC STYLE:

 A screenshot of the RStudio code editor showing a code block with line numbers 61 to 67. The code is:
 

```
61
62 ~ ````{r}
63 # This is a long comment that runs off the page if I let the text go beyond the vertical line.
64 # A histogram
65 gf_histogram(~ mpg, data = mtcars)
66 ````
67
```

 A red arrow points to a vertical line on the right side of the code block, labeled "Margin line". The comment on line 63 is too long and wraps around the margin line. A preview window on the right shows how the code would look in a knitted PDF, where the long comment is cut off and runs off the page.

### BETTER STYLE:

 A screenshot of the RStudio code editor showing a code block with line numbers 61 to 69. The code is:
 

```
61
62 ~ ````{r}
63 # This is a long comment that runs off the page if I let the text go beyond the
64 # vertical line.
65
66 # A histogram
67 gf_histogram(~ mpg, data = mtcars)
68 ````
69
```

 The code is formatted with proper spacing and line wrapping. The comment on line 63 is split across two lines to fit within the margin. A preview window on the right shows how the code would look in a knitted PDF, where the comment is properly wrapped and fits within the page margin.

---

## Answering questions in provided Rmd documents

I have redefined the format of the headings in most documents so that each heading level is a either an actual heading (e.g. “Instructions” or “Part 1: Exploring Data”), a question, or a part of a question (e.g., in line 123 below). Any non-heading text, then, is either additional description of the problem or your answers (e.g., in line 125 below). This makes it easier to find your answers when you knit the pdf, and easier to jump around to different questions using the outline feature.

```
123 • ## Predict the DOMS pain rating for someone who is 28 years old.
 Use as many decimal places as possible when you are doing the
 calculation, but round to the nearest hundredth in your final
 answer.
124
125 We estimate the average DOMS Pain score for someone who is 28
 years old is about 3.84.
126
```

It may often be easier to read the questions in the knitted pdf rather than the Rmd file.

If code and output are involved in your answer, I typically prefer that any code precedes the answer you provide in the text. So, if the question involves creating and interpreting a plot, I prefer that you insert a code chunk with your code for the plot, and then interpret the plot in space after the code chunk.