National Institute of Diabetes and Digestive and Kidney Diseases

Data organization and manipulation (Intermediate & Advanced Levels)

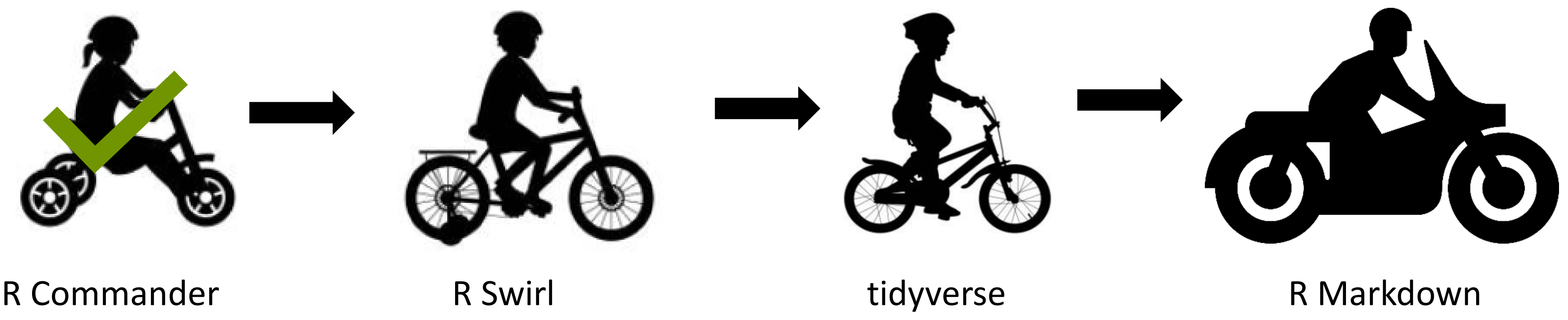
Emily Leary, PhD

Chief, Biostatistics Program Office
NIDDK

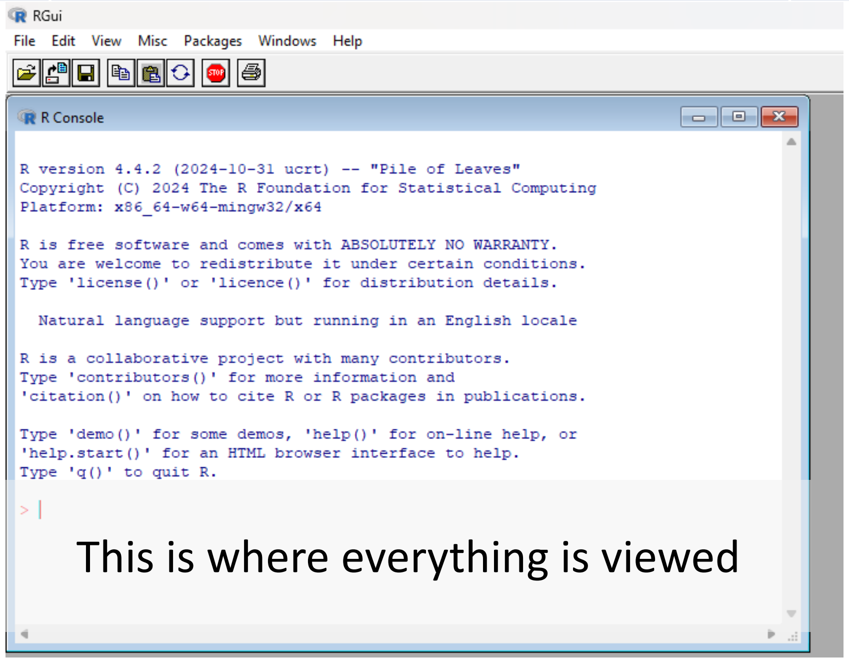
National Institute of Diabetes and Digestive and Kidney Diseases

Learning Objectives

NIH Library Intro to R and RStudio - how to load and basics

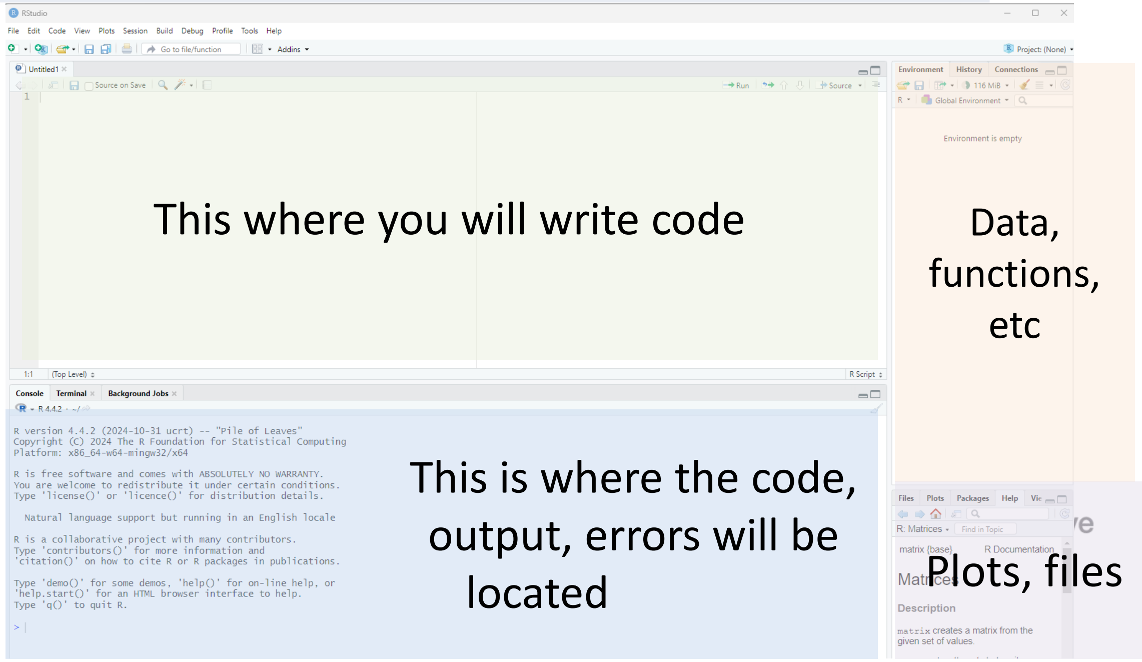


Things to Consider	Base R	R Studio
Nature	Programming language	Integrated Development Environment
Usage	Statistical computing, analysis	Enhancing the use of R through tools
User Interface	Command-line	GUI with additional features
Functionality	Core statistical analysis & graphing	Code editing, debugging, version control integration



```
R version 4.4.2 (2024-10-31 ucrt) -- "Pile of Leaves"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

This is where everything is viewed



```
R version 4.4.2 (2024-10-31 ucrt) -- "Pile of Leaves"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

matrix creates a matrix from the given set of values.

This where you will write code

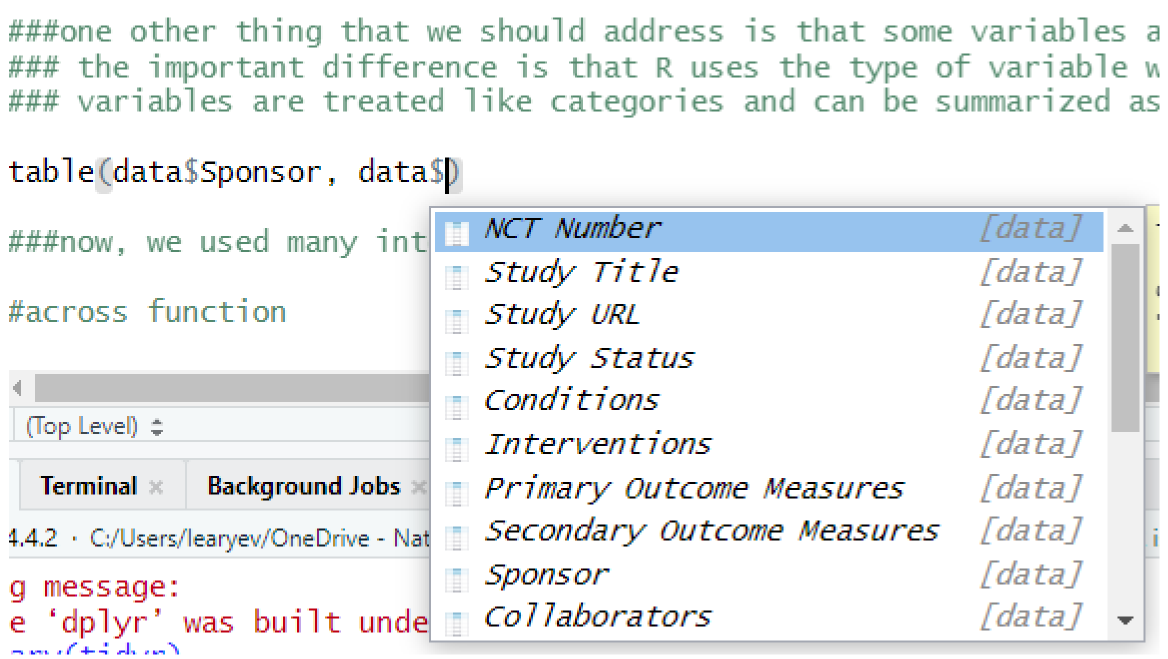
Data, functions, etc

This is where the code, output, errors will be located

Plots, files

Things to notice in RStudio

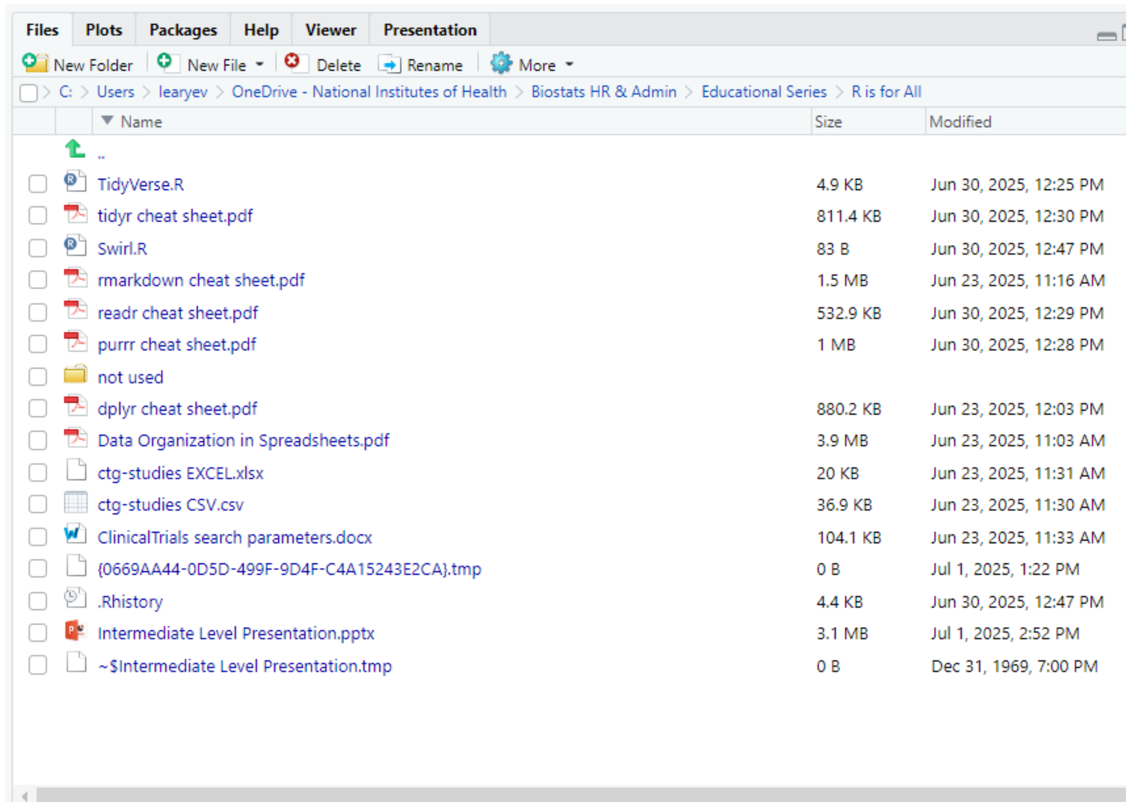
Here, we see some advantages to Rstudio over base R. You get helpful pop-ups which can make coding faster/easier.
Or, you have a handy location to find other files (other coding files, plots, etc) as well as tabs to look at packages, etc.



```
###one other thing that we should address is that some variables a
### the important difference is that R uses the type of variable w
### variables are treated like categories and can be summarized as

table(data$Sponsor, data$)

###now, we used many int

#across function
```

```
NCT Number                    [data]
Study Title                   [data]
Study URL                     [data]
Study Status                  [data]
Conditions                    [data]
Interventions                 [data]
Primary Outcome Measures      [data]
Secondary Outcome Measures    [data]
Sponsor                       [data]
Collaborators                 [data]
```

```
4.4.2 · C:/Users/learyev/OneDrive - Nat
g message:
e 'dplyr' was built unde
```

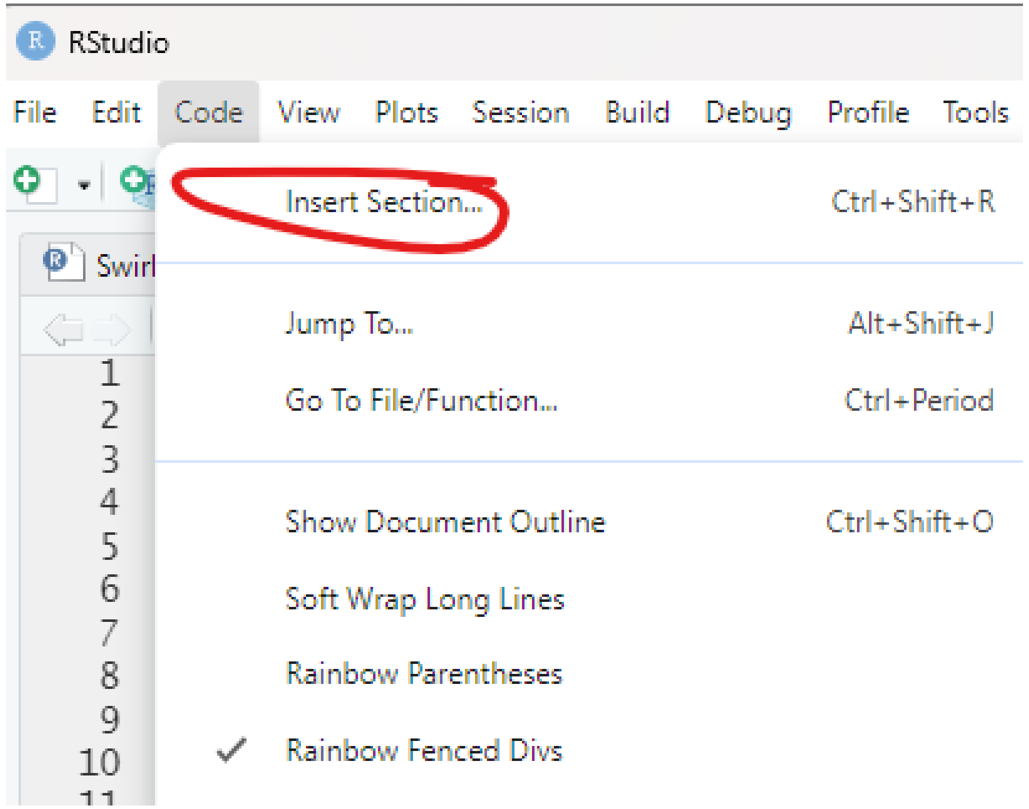


C: > Users > learyev > OneDrive - National Institutes of Health > Biostats HR & Admin > Educational Series > R is for All

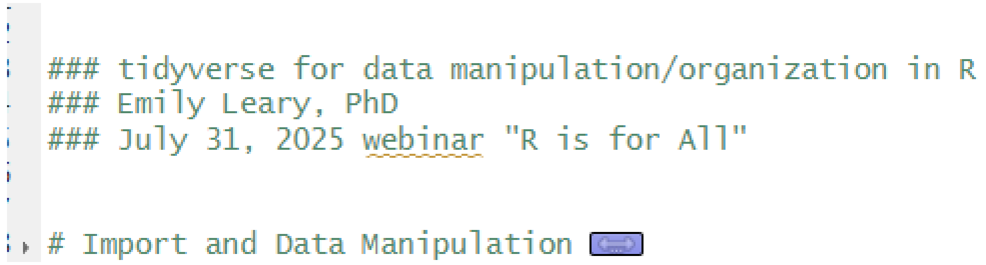
Name	Size	Modified
..		
TidyVerse.R	4.9 KB	Jun 30, 2025, 12:25 PM
tidyr cheat sheet.pdf	811.4 KB	Jun 30, 2025, 12:30 PM
Swirl.R	83 B	Jun 30, 2025, 12:47 PM
rmarkdown cheat sheet.pdf	1.5 MB	Jun 23, 2025, 11:16 AM
readr cheat sheet.pdf	532.9 KB	Jun 30, 2025, 12:29 PM
purrr cheat sheet.pdf	1 MB	Jun 30, 2025, 12:28 PM
not used		
dplyr cheat sheet.pdf	880.2 KB	Jun 23, 2025, 12:03 PM
Data Organization in Spreadsheets.pdf	3.9 MB	Jun 23, 2025, 11:03 AM
ctg-studies EXCEL.xlsx	20 KB	Jun 23, 2025, 11:31 AM
ctg-studies CSV.csv	36.9 KB	Jun 23, 2025, 11:30 AM
ClinicalTrials search parameters.docx	104.1 KB	Jun 23, 2025, 11:33 AM
{0669AA44-0D5D-499F-9D4F-C4A15243E2CA}.tmp	0 B	Jul 1, 2025, 1:22 PM
.Rhistory	4.4 KB	Jun 30, 2025, 12:47 PM
Intermediate Level Presentation.pptx	3.1 MB	Jul 1, 2025, 2:52 PM
~$Intermediate Level Presentation.tmp	0 B	Dec 31, 1969, 7:00 PM

Things to notice in RStudio

We'll talk about more advanced options, but using sections
Is a good start to better organize your R code.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools
Swirl
Insert Section... Ctrl+Shift+R
Jump To... Alt+Shift+J
Go To File/Function... Ctrl+Period
Show Document Outline Ctrl+Shift+O
Soft Wrap Long Lines
Rainbow Parentheses
✓ Rainbow Fenced Divs

Can you see how I've organized my code for this webinar?



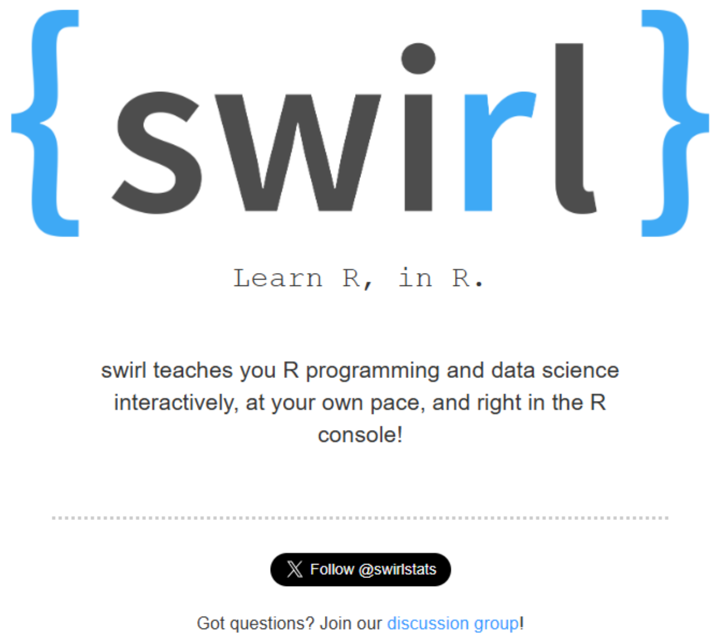
```
### tidyverse for data manipulation/organization in R
### Emily Leary, PhD
### July 31, 2025 webinar "R is for All"


# Import and Data Manipulation
```

What terminology is needed for data cleaning in R?

- Objects - to store data
  - Data frame – 2-dimensional data storage of all types; this is basically like an excel file with rows and columns
  - Matrix – 2-dimensional data storage for numerical data only
  - Vector – 1-dimensional data storage
- Data Types
  - Character/string – storing text
  - Factor –character variables with limited number of options for text
  - Substring – characters extracted from a character variable or string
  - Numerical – storing numbers
- # is the way to comment code in R

R Swirl

- We’ll start with R Swirl which is an interactive package in R to help self-direct learning for more general tools, often called **functional programming tools** because they are built around functions that take other functions as inputs.
- This can quickly get abstract, but R Swirl helps learners stay focused



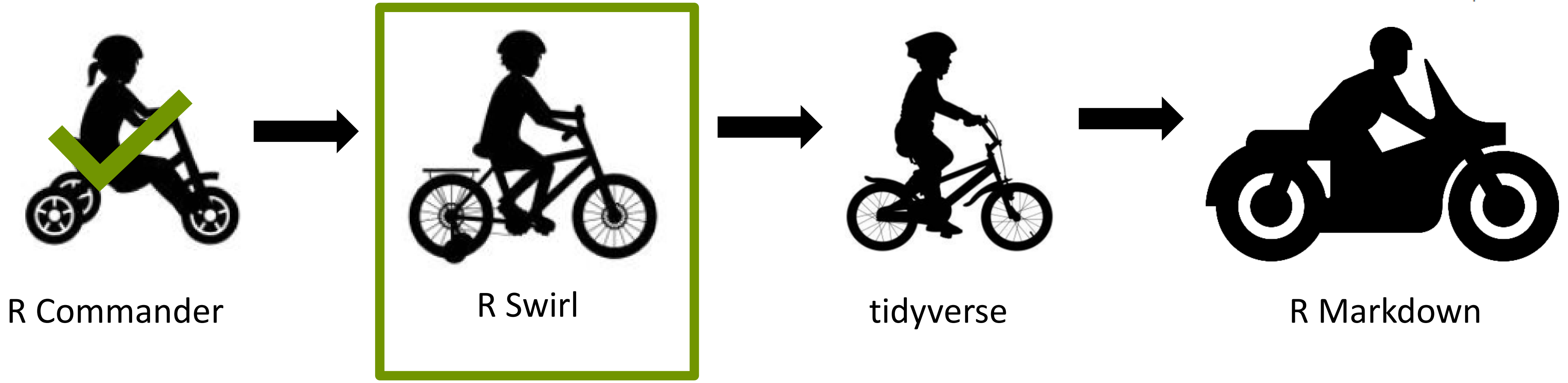
{swirl}

Learn R, in R.

swirl teaches you R programming and data science interactively, at your own pace, and right in the R console!

Follow @swirlstats

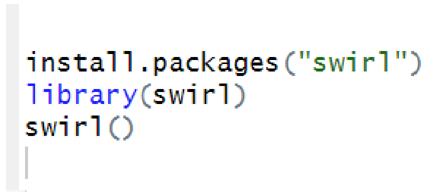
Got questions? Join our discussion group!



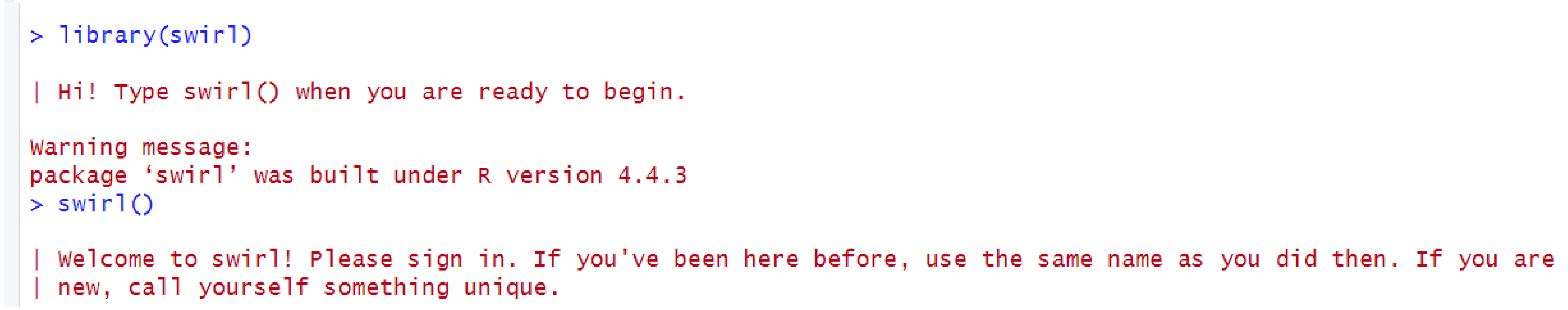
R Swirl

To start using swirl, use the following code in RStudio

Note that this is all happening in the log portion of RStudio, follow the prompts until you get the course selection screen below



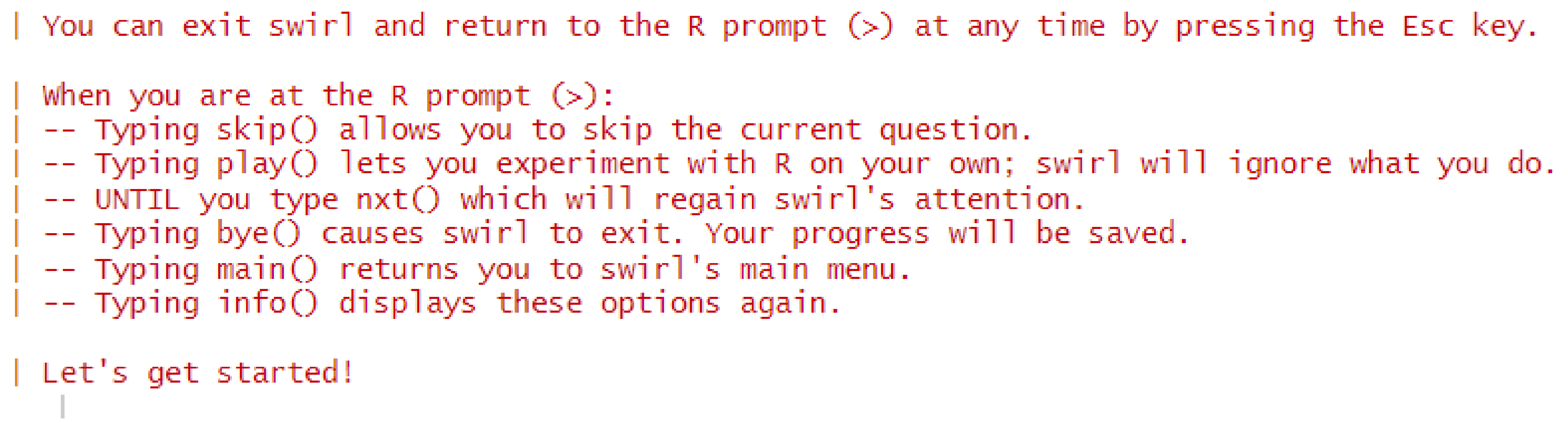
```
install.packages("swirl")
library(swirl)
swirl()
```



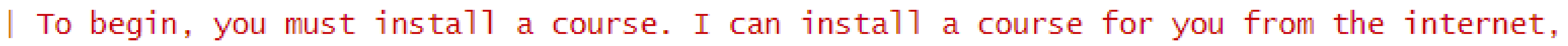
```
> library(swirl)

| Hi! Type swirl() when you are ready to begin.

Warning message:
package ‘swirl’ was built under R version 4.4.3
> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same name as you did then. If you are
| new, call yourself something unique.
```

```
If you are already at the prompt, type bye() to exit and save your progress.

When you exit properly, you'll see a short message letting you know you've done so.
```

R Swirl

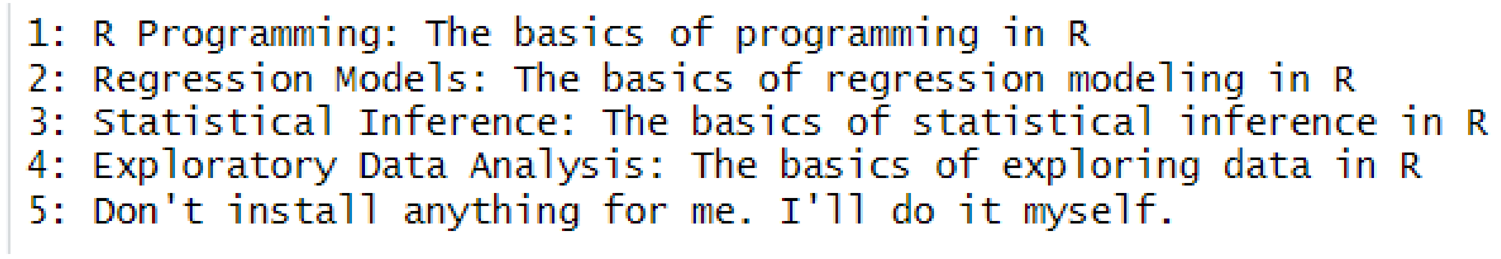


```
| You can exit swirl and return to the R prompt (>) at any time by pressing the Esc key.

| When you are at the R prompt (>):
| -- Typing skip() allows you to skip the current question.
| -- Typing play() lets you experiment with R on your own; swirl will ignore what you do.
| -- UNTIL you type nxt() which will regain swirl's attention.
| -- Typing bye() causes swirl to exit. Your progress will be saved.
| -- Typing main() returns you to swirl's main menu.
| -- Typing info() displays these options again.

| Let's get started!
```



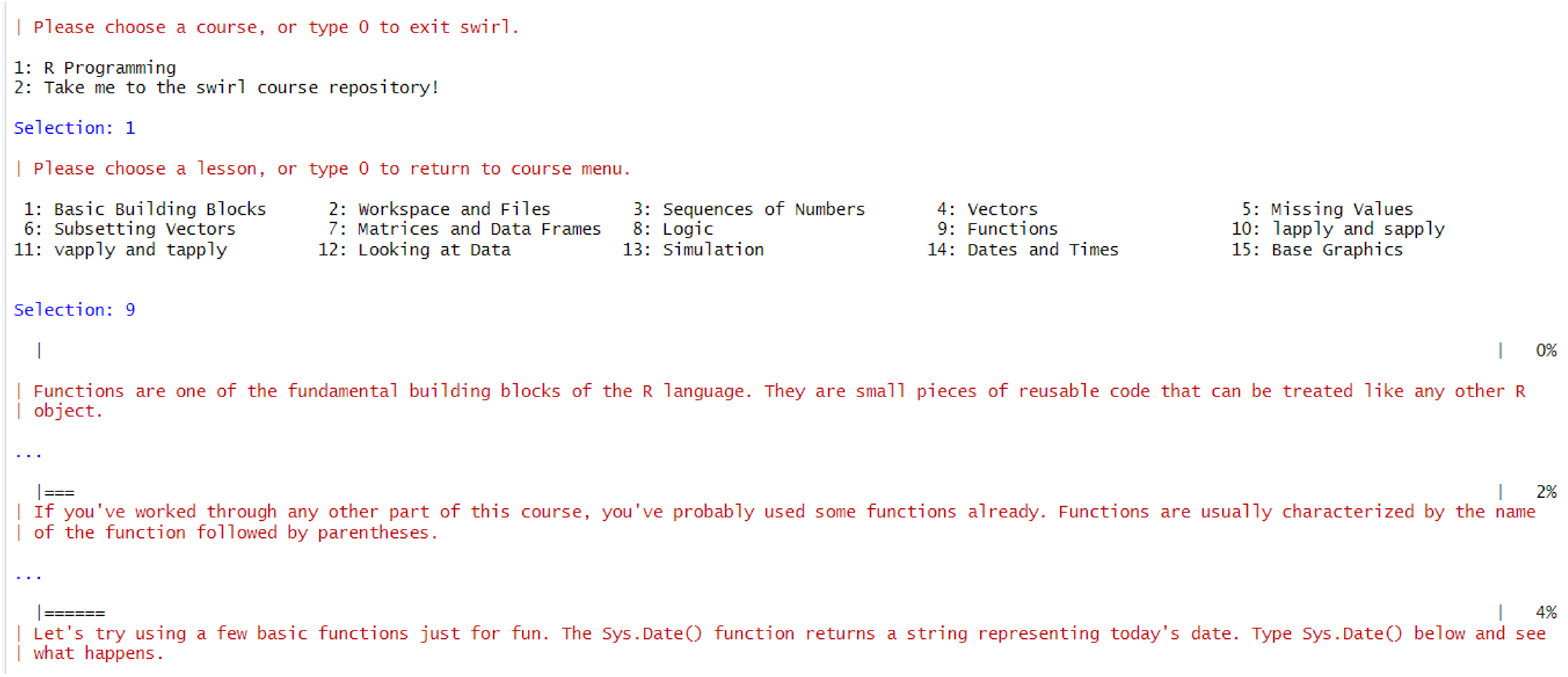
```
| To begin, you must install a course. I can install a course for you from the internet,
```

Note that you MUST be connected to the internet to use RSwirl



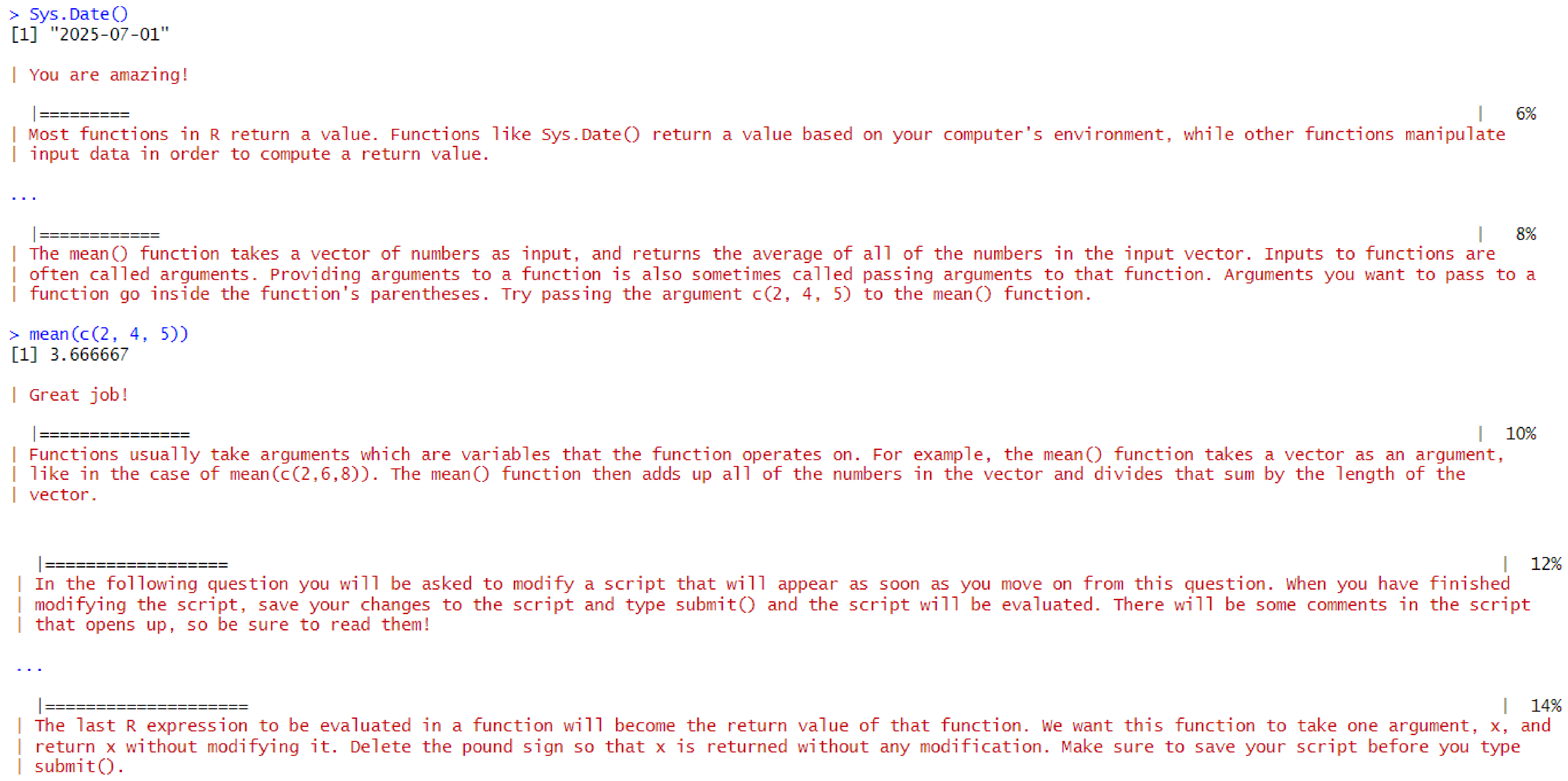
```
1: R Programming: The basics of programming in R
2: Regression Models: The basics of regression modeling in R
3: Statistical Inference: The basics of statistical inference in R
4: Exploratory Data Analysis: The basics of exploring data in R
5: Don't install anything for me. I'll do it myself.
```

Follow along for R Programming: Functions



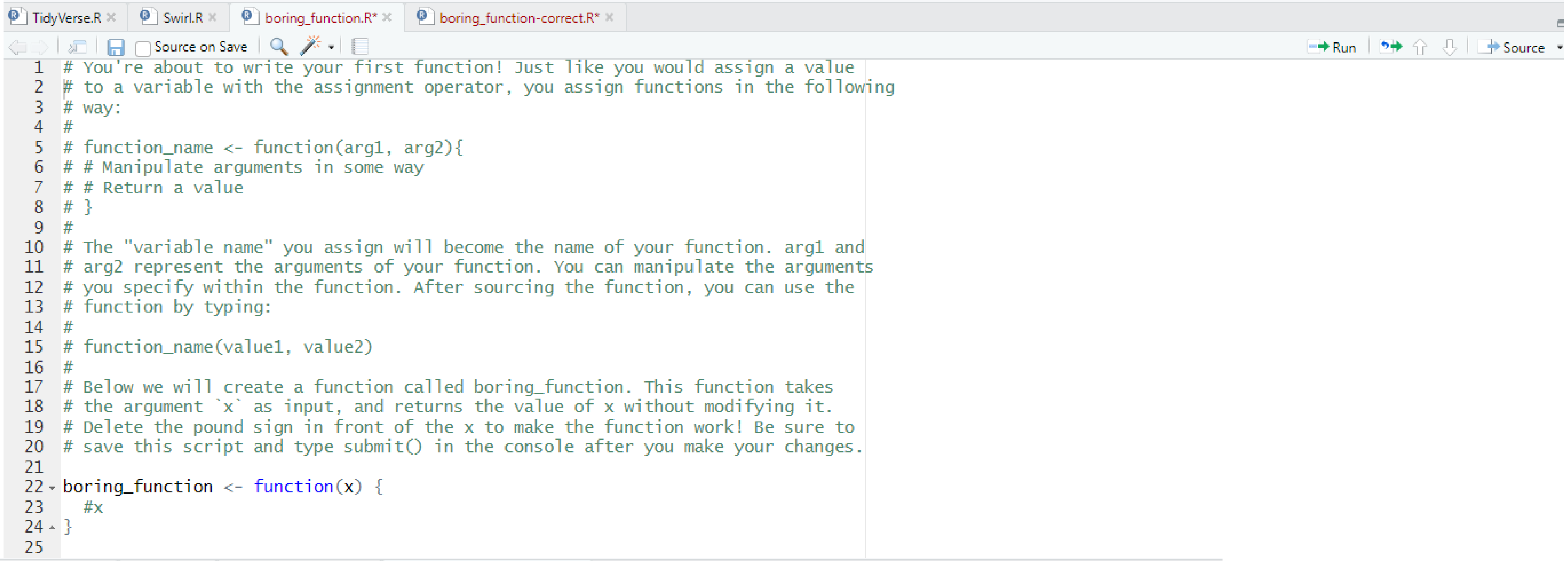
```
| Please choose a course, or type 0 to exit swirl.

1: R Programming
2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

 1: Basic Building Blocks      2: Workspace and Files        3: Sequences of Numbers       4: Vectors                    5: Missing Values
 6: Subsetting Vectors         7: Matrices and Data Frames   8: Logic                      9: Functions                 10: lapply and sapply
11: vapply and tapply         12: Looking at Data           13: Simulation                14: Dates and Times           15: Base Graphics


Selection: 9

  |                                                                                                                             |   0%

| Functions are one of the fundamental building blocks of the R language. They are small pieces of reusable code that can be treated like any other R
| object.

...

  |===                                                                                                                          |   2%
| If you've worked through any other part of this course, you've probably used some functions already. Functions are usually characterized by the name
| of the function followed by parentheses.

...

  |======                                                                                                                       |   4%
| Let's try using a few basic functions just for fun. The Sys.Date() function returns a string representing today's date. Type Sys.Date() below and see
| what happens.
```

Follow along for R Programming: Functions

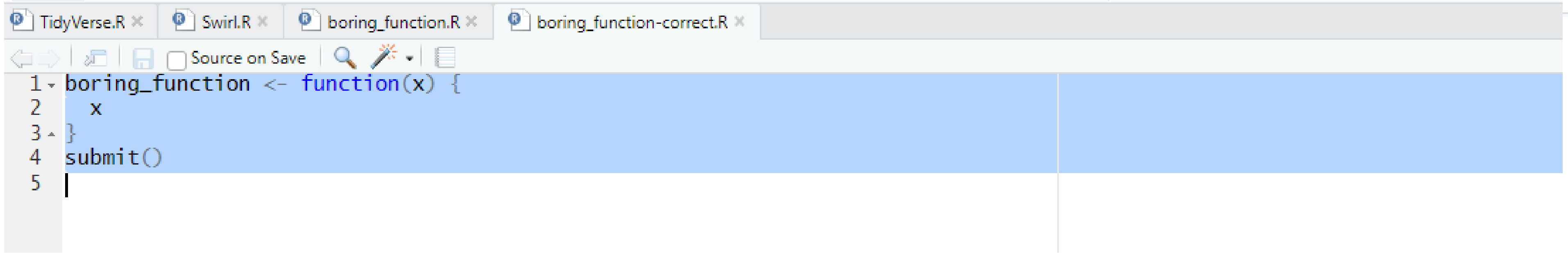


```
> Sys.Date()
[1] "2025-07-01"

| You are amazing!

  |=========                                                                                                      |   6%
| Most functions in R return a value. Functions like Sys.Date() return a value based on your computer's environment, while other functions manipulate
| input data in order to compute a return value.

...

  |============                                                                                                   |   8%
| The mean() function takes a vector of numbers as input, and returns the average of all of the numbers in the input vector. Inputs to functions are
| often called arguments. Providing arguments to a function is also sometimes called passing arguments to that function. Arguments you want to pass to a
| function go inside the function's parentheses. Try passing the argument c(2, 4, 5) to the mean() function.

> mean(c(2, 4, 5))
[1] 3.666667

| Great job!

  |===============                                                                                                |  10%
| Functions usually take arguments which are variables that the function operates on. For example, the mean() function takes a vector as an argument,
| like in the case of mean(c(2,6,8)). The mean() function then adds up all of the numbers in the vector and divides that sum by the length of the
| vector.


  |==================                                                                                             |  12%
| In the following question you will be asked to modify a script that will appear as soon as you move on from this question. When you have finished
| modifying the script, save your changes to the script and type submit() and the script will be evaluated. There will be some comments in the script
| that opens up, so be sure to read them!

...

  |====================                                                                                           |  14%
| The last R expression to be evaluated in a function will become the return value of that function. We want this function to take one argument, x, and
| return x without modifying it. Delete the pound sign so that x is returned without any modification. Make sure to save your script before you type
| submit().
```

Follow along for R Programming: Functions

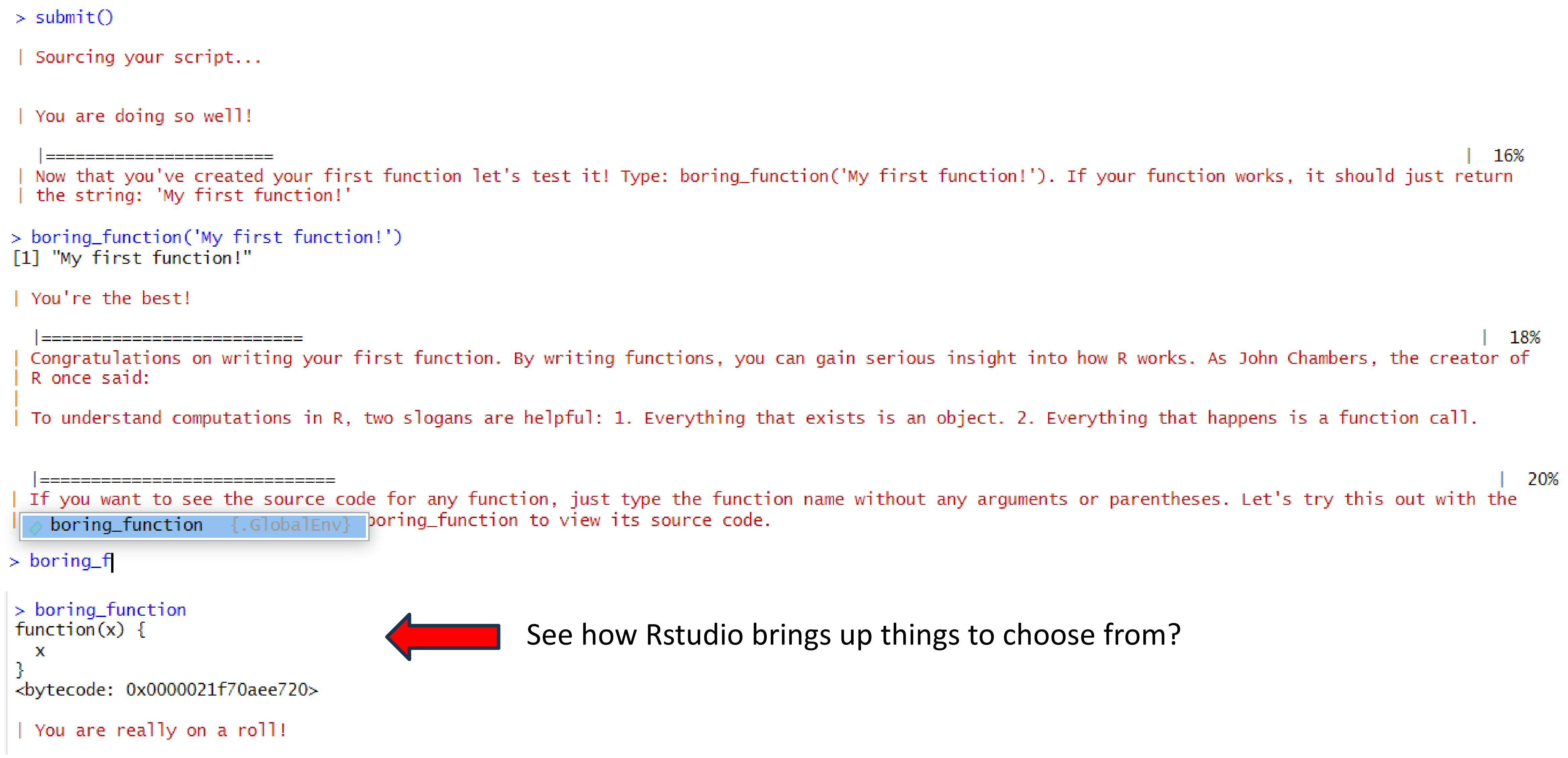


```r
# You're about to write your first function! Just like you would assign a value
# to a variable with the assignment operator, you assign functions in the following
# way:
#
# function_name <- function(arg1, arg2){
# # Manipulate arguments in some way
# # Return a value
# }
#
# The "variable name" you assign will become the name of your function. arg1 and
# arg2 represent the arguments of your function. You can manipulate the arguments
# you specify within the function. After sourcing the function, you can use the
# function by typing:
#
# function_name(value1, value2)
#
# Below we will create a function called boring_function. This function takes
# the argument `x` as input, and returns the value of x without modifying it.
# Delete the pound sign in front of the x to make the function work! Be sure to
# save this script and type submit() in the console after you make your changes.

boring_function <- function(x) {
  #x
}

```



```r
boring_function <- function(x) {
  x
}
submit()

```

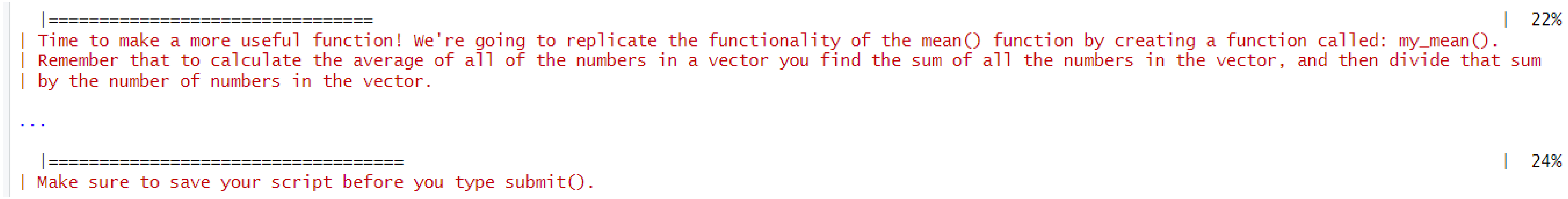
Follow along for R Programming: Functions



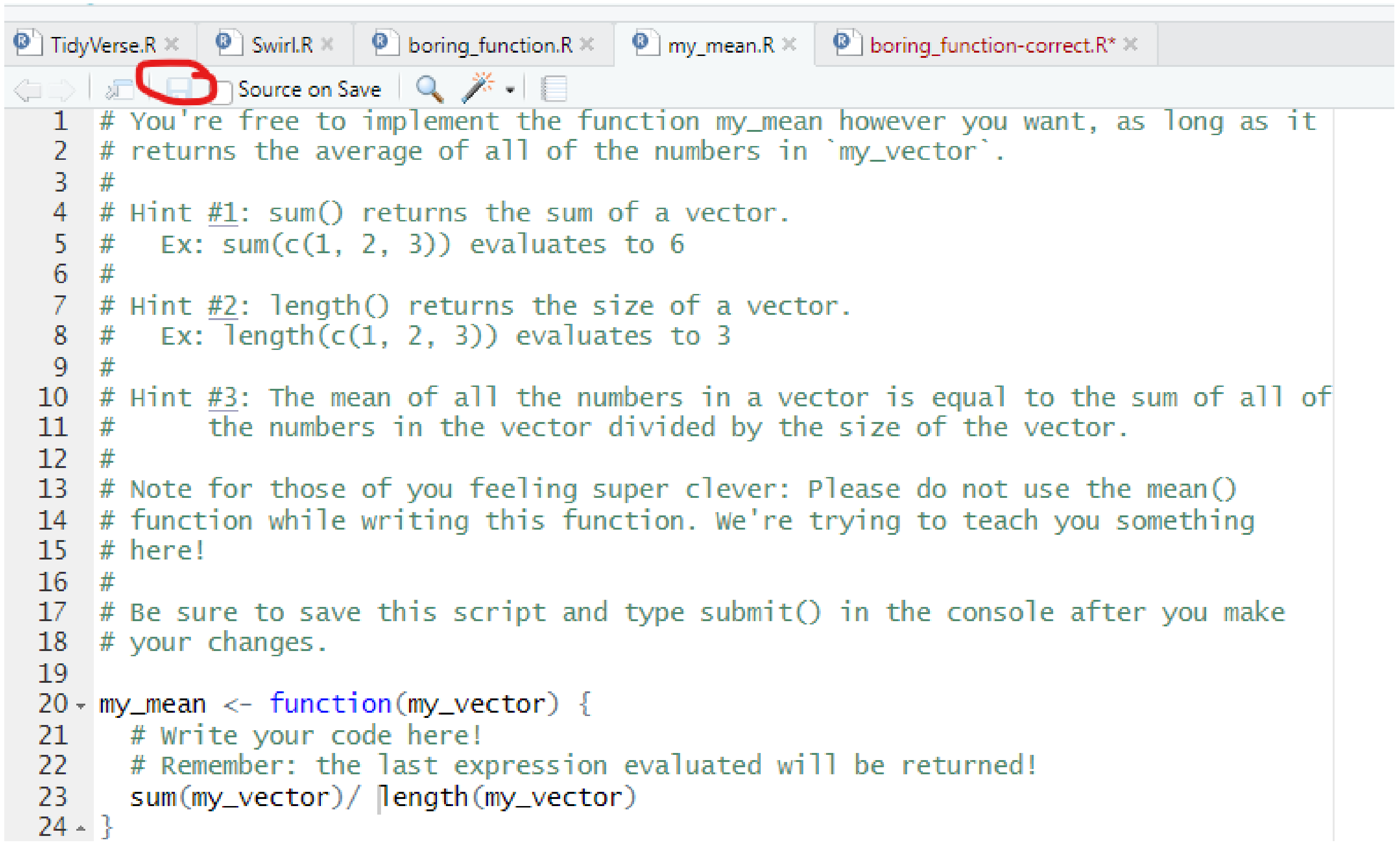
```
> submit()

| Sourcing your script...


| You are doing so well!

  |======================                                                                                           |  16%
| Now that you've created your first function let's test it! Type: boring_function('My first function!'). If your function works, it should just return
| the string: 'My first function!'

> boring_function('My first function!')
[1] "My first function!"

| You're the best!

  |==========================                                                                                       |  18%
| Congratulations on writing your first function. By writing functions, you can gain serious insight into how R works. As John Chambers, the creator of
| R once said:
|
| To understand computations in R, two slogans are helpful: 1. Everything that exists is an object. 2. Everything that happens is a function call.


  |=============================                                                                                    |  20%
| If you want to see the source code for any function, just type the function name without any arguments or parentheses. Let's try this out with the
|  boring_function   {.GlobalEnv}  boring_function to view its source code.

> boring_f

> boring_function
function(x) {
  x
}
<bytecode: 0x0000021f70aee720>

| You are really on a roll!
```

See how Rstudio brings up things to choose from?

Follow along for R Programming: Functions

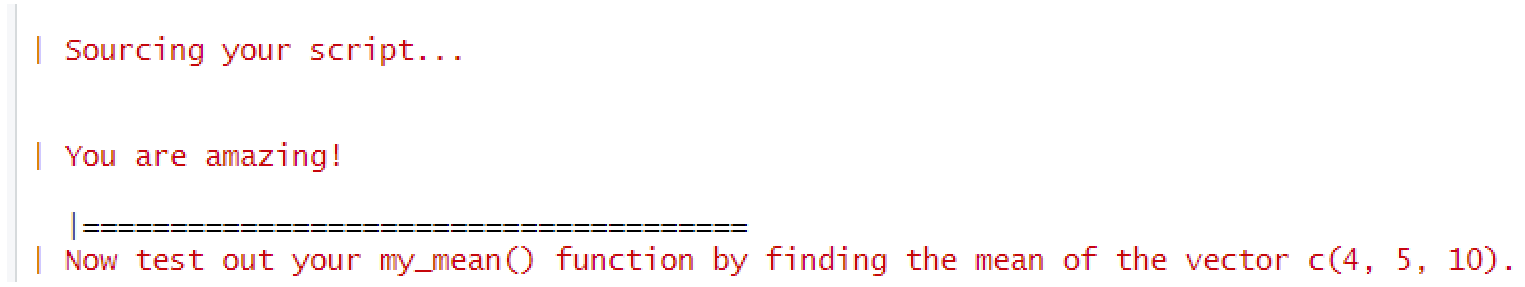


```
  |================================                                                                    |  22%
| Time to make a more useful function! We're going to replicate the functionality of the mean() function by creating a function called: my_mean().
| Remember that to calculate the average of all of the numbers in a vector you find the sum of all the numbers in the vector, and then divide that sum
| by the number of numbers in the vector.

...

  |==================================                                                                  |  24%
| Make sure to save your script before you type submit().
```

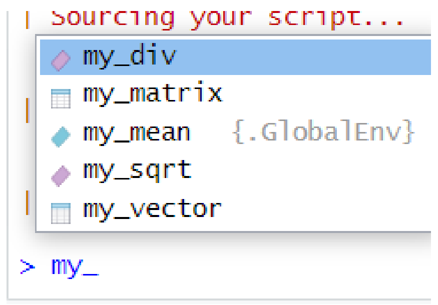


```r
# You're free to implement the function my_mean however you want, as long as it
# returns the average of all of the numbers in `my_vector`.
#
# Hint #1: sum() returns the sum of a vector.
#   Ex: sum(c(1, 2, 3)) evaluates to 6
#
# Hint #2: length() returns the size of a vector.
#   Ex: length(c(1, 2, 3)) evaluates to 3
#
# Hint #3: The mean of all the numbers in a vector is equal to the sum of all of
#      the numbers in the vector divided by the size of the vector.
#
# Note for those of you feeling super clever: Please do not use the mean()
# function while writing this function. We're trying to teach you something
# here!
#
# Be sure to save this script and type submit() in the console after you make
# your changes.

my_mean <- function(my_vector) {
  # Write your code here!
  # Remember: the last expression evaluated will be returned!
  sum(my_vector)/ length(my_vector)
}
```

Follow along for R Programming: Functions

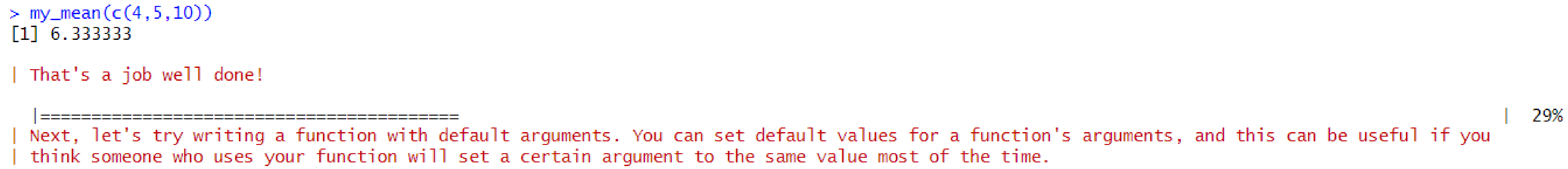


```
| Sourcing your script...


| You are amazing!

  |=======================================
| Now test out your my_mean() function by finding the mean of the vector c(4, 5, 10).
```



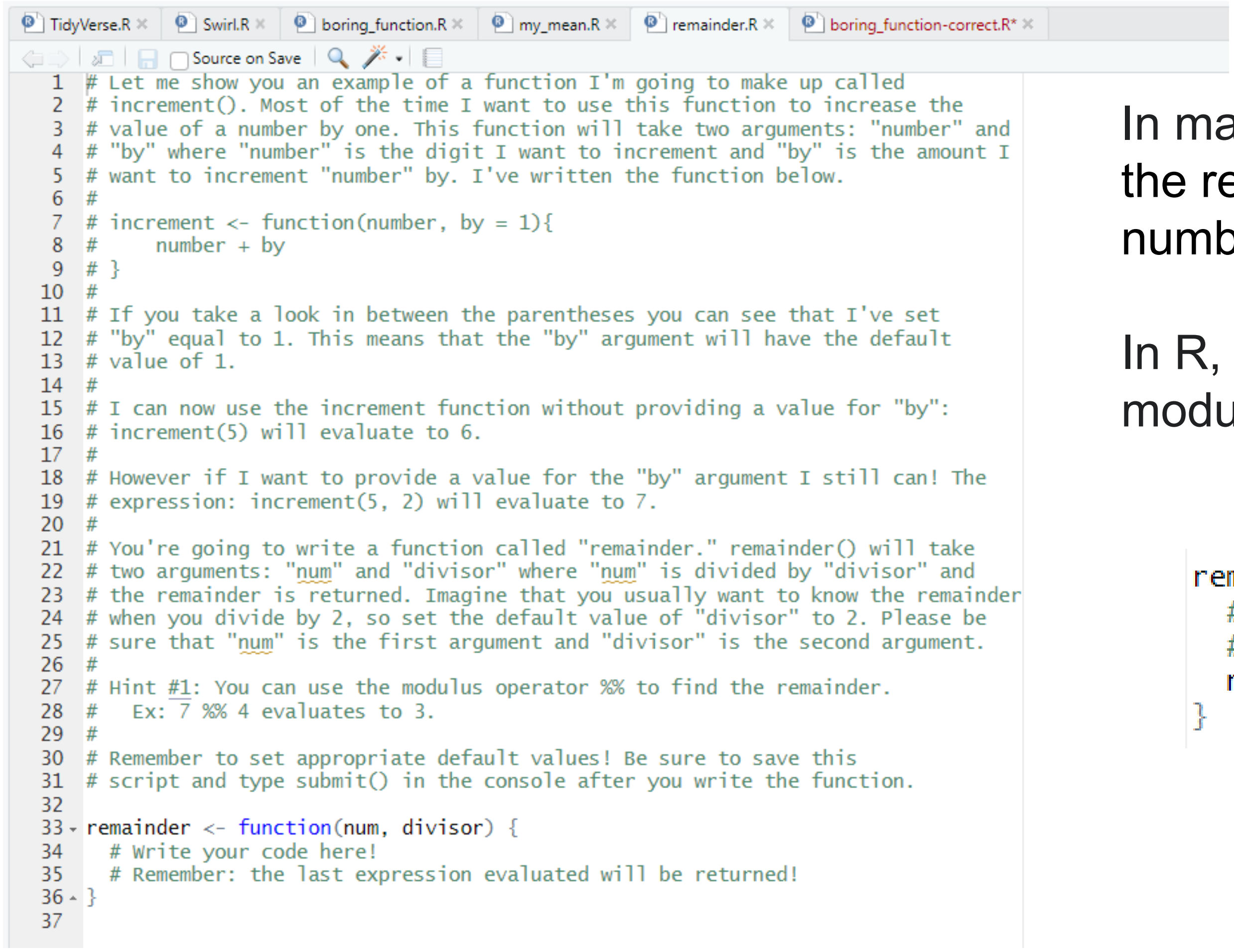
```
| Sourcing your script...
   my_div
   my_matrix
   my_mean    {.GlobalEnv}
   my_sqrt
   my_vector
> my_
```

Notice how Rstudio helps you by providing a menu to autofill your command?



```
> my_mean(c(4,5,10))
[1] 6.333333

| That's a job well done!

  |=====================================                                                         |  29%
| Next, let's try writing a function with default arguments. You can set default values for a function's arguments, and this can be useful if you
| think someone who uses your function will set a certain argument to the same value most of the time.
```

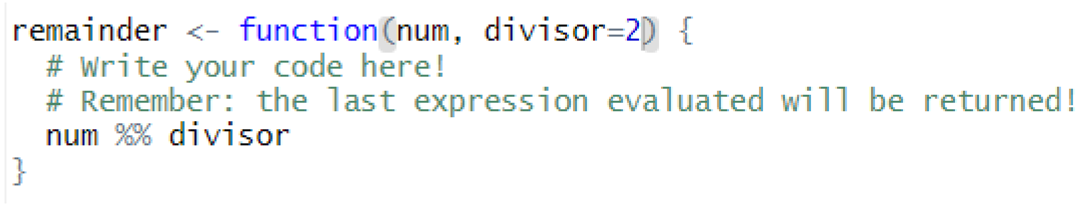
Follow along for R Programming: Functions



```r
# Let me show you an example of a function I'm going to make up called
# increment(). Most of the time I want to use this function to increase the
# value of a number by one. This function will take two arguments: "number" and
# "by" where "number" is the digit I want to increment and "by" is the amount I
# want to increment "number" by. I've written the function below.
#
# increment <- function(number, by = 1){
#     number + by
# }
#
# If you take a look in between the parentheses you can see that I've set
# "by" equal to 1. This means that the "by" argument will have the default
# value of 1.
#
# I can now use the increment function without providing a value for "by":
# increment(5) will evaluate to 6.
#
# However if I want to provide a value for the "by" argument I still can! The
# expression: increment(5, 2) will evaluate to 7.
#
# You're going to write a function called "remainder." remainder() will take
# two arguments: "num" and "divisor" where "num" is divided by "divisor" and
# the remainder is returned. Imagine that you usually want to know the remainder
# when you divide by 2, so set the default value of "divisor" to 2. Please be
# sure that "num" is the first argument and "divisor" is the second argument.
#
# Hint #1: You can use the modulus operator %% to find the remainder.
#   Ex: 7 %% 4 evaluates to 3.
#
# Remember to set appropriate default values! Be sure to save this
# script and type submit() in the console after you write the function.

remainder <- function(num, divisor) {
  # Write your code here!
  # Remember: the last expression evaluated will be returned!
}

```

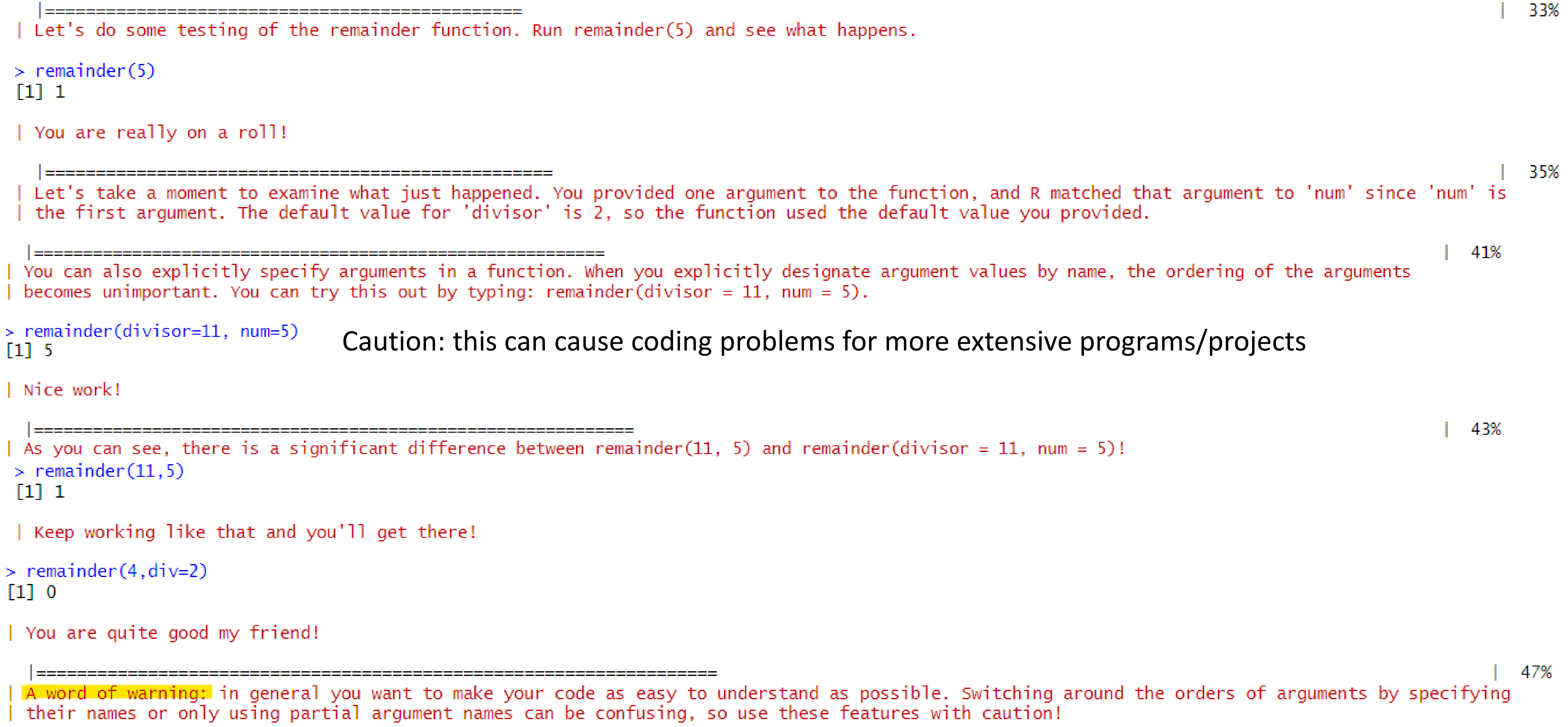
In mathematics, the **modulo operation** returns the remainder or signed remainder of a division, after one number is divided by another

In R, we can use “%%” to do this OR “mod(n, m)” is the modulo operator and returns n mod m.



```r
remainder <- function(num, divisor=2) {
  # Write your code here!
  # Remember: the last expression evaluated will be returned!
  num %% divisor
}
```

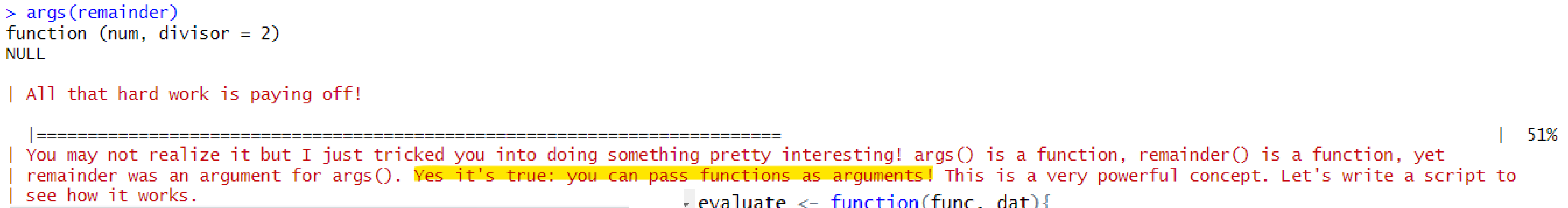
Follow along for R Programming: Functions



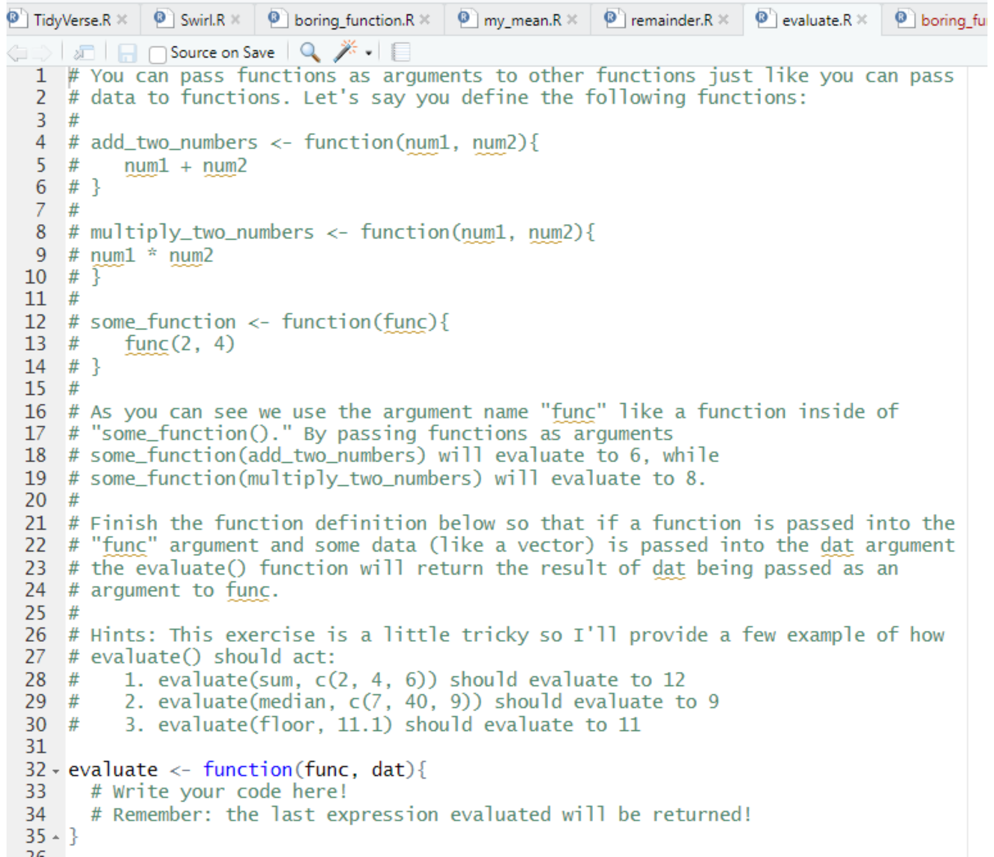
```
  |=============================================                         |  33%
| Let's do some testing of the remainder function. Run remainder(5) and see what happens.

> remainder(5)
[1] 1

| You are really on a roll!

  |================================================                      |  35%
| Let's take a moment to examine what just happened. You provided one argument to the function, and R matched that argument to 'num' since 'num' is
| the first argument. The default value for 'divisor' is 2, so the function used the default value you provided.

  |=======================================================               |  41%
| You can also explicitly specify arguments in a function. When you explicitly designate argument values by name, the ordering of the arguments
| becomes unimportant. You can try this out by typing: remainder(divisor = 11, num = 5).

> remainder(divisor=11, num=5)
[1] 5

| Nice work!

  |==========================================================            |  43%
| As you can see, there is a significant difference between remainder(11, 5) and remainder(divisor = 11, num = 5)!
> remainder(11,5)
[1] 1

| Keep working like that and you'll get there!

> remainder(4,div=2)
[1] 0

| You are quite good my friend!

  |==================================================================    |  47%
| A word of warning: in general you want to make your code as easy to understand as possible. Switching around the orders of arguments by specifying
| their names or only using partial argument names can be confusing, so use these features with caution!
```

Caution: this can cause coding problems for more extensive programs/projects

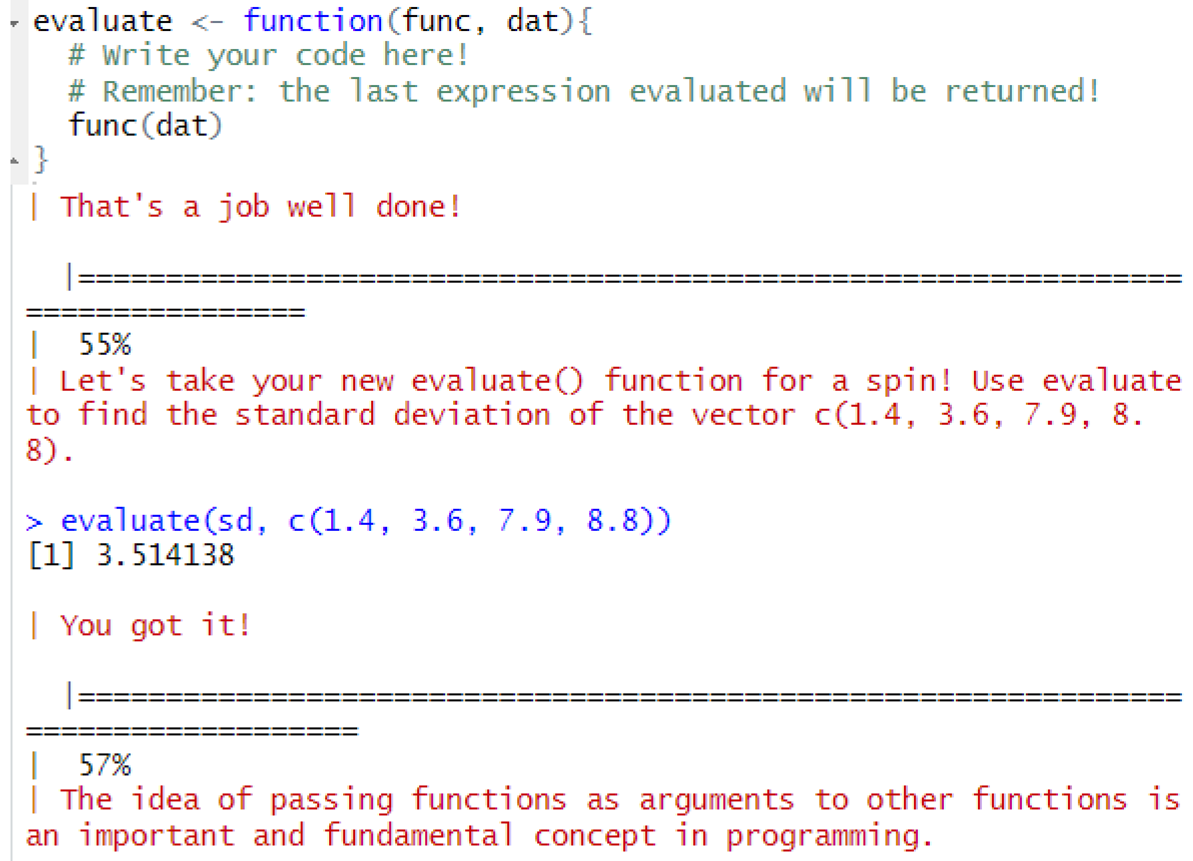
THE KEY TO CODING FREEDOM!



```
> args(remainder)
function (num, divisor = 2)
NULL

| All that hard work is paying off!

  |===========================================================================                               |  51%
| You may not realize it but I just tricked you into doing something pretty interesting! args() is a function, remainder() is a function, yet
| remainder was an argument for args(). Yes it's true: you can pass functions as arguments! This is a very powerful concept. Let's write a script to
| see how it works.
```

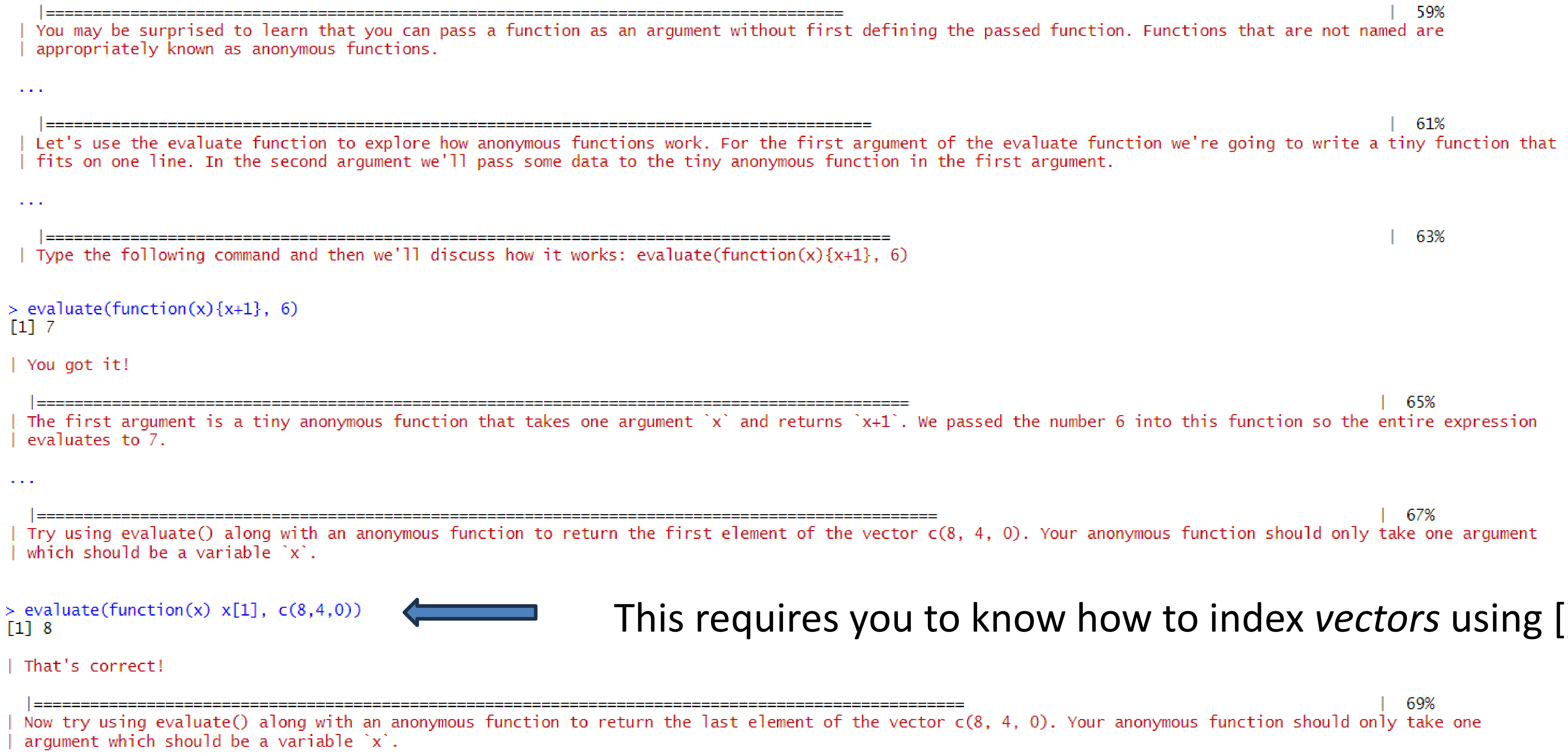


```
# You can pass functions as arguments to other functions just like you can pass
# data to functions. Let's say you define the following functions:
#
# add_two_numbers <- function(num1, num2){
#    num1 + num2
# }
#
# multiply_two_numbers <- function(num1, num2){
# num1 * num2
# }
#
# some_function <- function(func){
#    func(2, 4)
# }
#
# As you can see we use the argument name "func" like a function inside of
# "some_function()." By passing functions as arguments
# some_function(add_two_numbers) will evaluate to 6, while
# some_function(multiply_two_numbers) will evaluate to 8.
#
# Finish the function definition below so that if a function is passed into the
# "func" argument and some data (like a vector) is passed into the dat argument
# the evaluate() function will return the result of dat being passed as an
# argument to func.
#
# Hints: This exercise is a little tricky so I'll provide a few example of how
# evaluate() should act:
#    1. evaluate(sum, c(2, 4, 6)) should evaluate to 12
#    2. evaluate(median, c(7, 40, 9)) should evaluate to 9
#    3. evaluate(floor, 11.1) should evaluate to 11

evaluate <- function(func, dat){
  # Write your code here!
  # Remember: the last expression evaluated will be returned!
}
```



```
evaluate <- function(func, dat){
  # Write your code here!
  # Remember: the last expression evaluated will be returned!
  func(dat)
}
| That's a job well done!

  |===================================================================
================
|  55%
| Let's take your new evaluate() function for a spin! Use evaluate
to find the standard deviation of the vector c(1.4, 3.6, 7.9, 8.
8).

> evaluate(sd, c(1.4, 3.6, 7.9, 8.8))
[1] 3.514138

| You got it!

  |===================================================================
===================
|  57%
| The idea of passing functions as arguments to other functions is
an important and fundamental concept in programming.
```

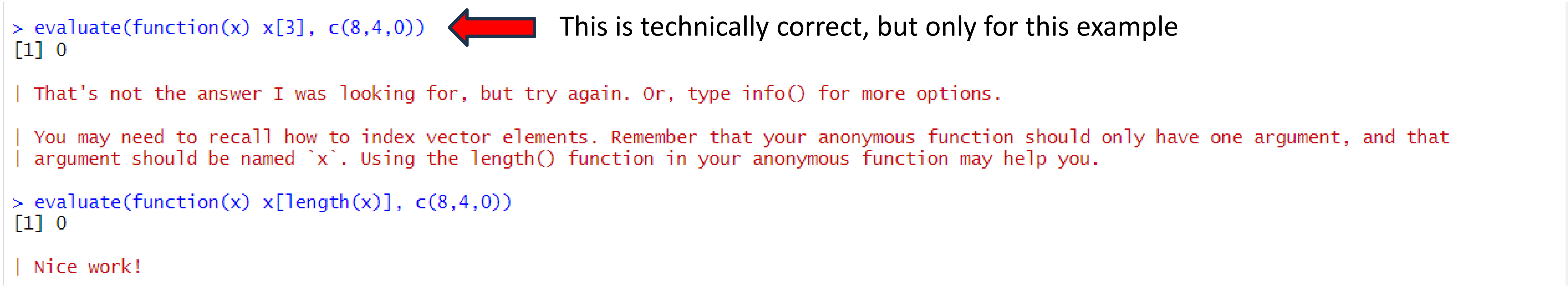
Follow along for R Programming: Functions



```
  |=====================================================================================                                  |  59%
| You may be surprised to learn that you can pass a function as an argument without first defining the passed function. Functions that are not named are
| appropriately known as anonymous functions.

...

  |========================================================================================                               |  61%
| Let's use the evaluate function to explore how anonymous functions work. For the first argument of the evaluate function we're going to write a tiny function that
| fits on one line. In the second argument we'll pass some data to the tiny anonymous function in the first argument.

...

  |==========================================================================================                             |  63%
| Type the following command and then we'll discuss how it works: evaluate(function(x){x+1}, 6)

> evaluate(function(x){x+1}, 6)
[1] 7

| You got it!

  |============================================================================================                           |  65%
| The first argument is a tiny anonymous function that takes one argument `x` and returns `x+1`. We passed the number 6 into this function so the entire expression
| evaluates to 7.

...

  |===============================================================================================                        |  67%
| Try using evaluate() along with an anonymous function to return the first element of the vector c(8, 4, 0). Your anonymous function should only take one argument
| which should be a variable `x`.

> evaluate(function(x) x[1], c(8,4,0))
[1] 8

| That's correct!

  |=================================================================================================                      |  69%
| Now try using evaluate() along with an anonymous function to return the last element of the vector c(8, 4, 0). Your anonymous function should only take one
| argument which should be a variable `x`.
```

This requires you to know how to index *vectors* using []

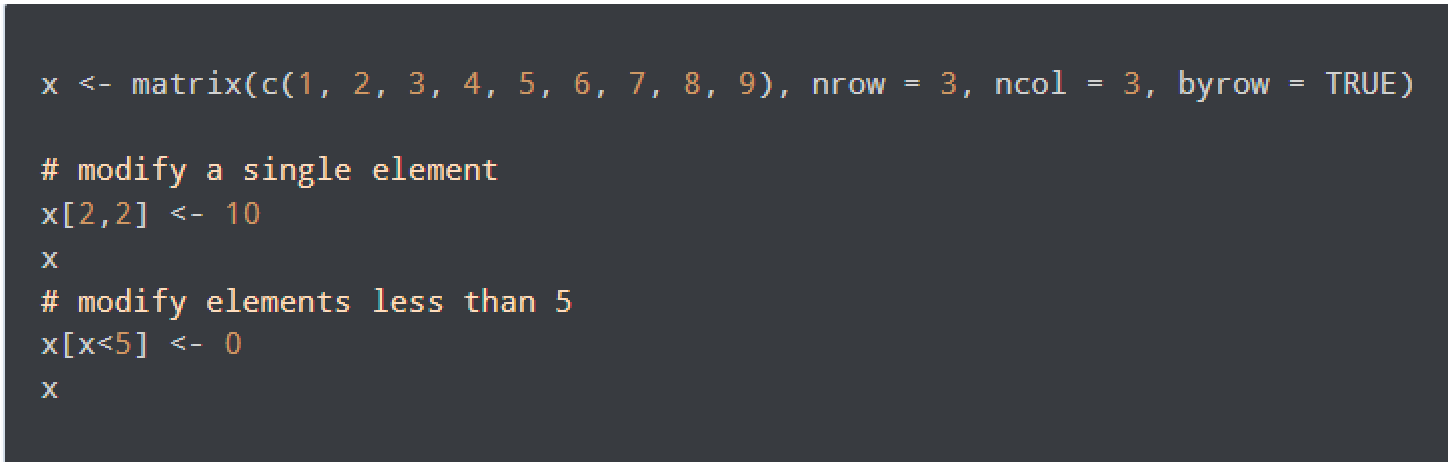
Follow along for R Programming: Functions



```
> evaluate(function(x) x[3], c(8,4,0))
[1] 0

| That's not the answer I was looking for, but try again. Or, type info() for more options.

| You may need to recall how to index vector elements. Remember that your anonymous function should only have one argument, and that
| argument should be named `x`. Using the length() function in your anonymous function may help you.

> evaluate(function(x) x[length(x)], c(8,4,0))
[1] 0

| Nice work!
```

This is technically correct, but only for this example

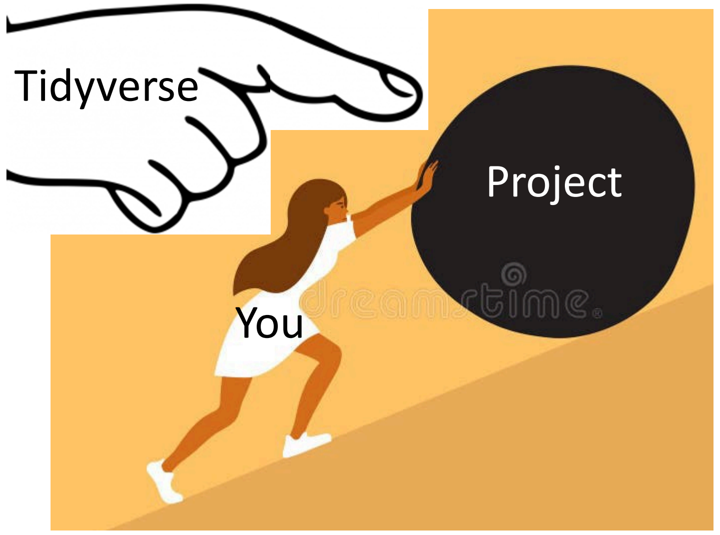
Indices for matrices are like for vectors are accessed as var[row, column]. Here row and column are vectors and we can select elements in the same way as for vectors.

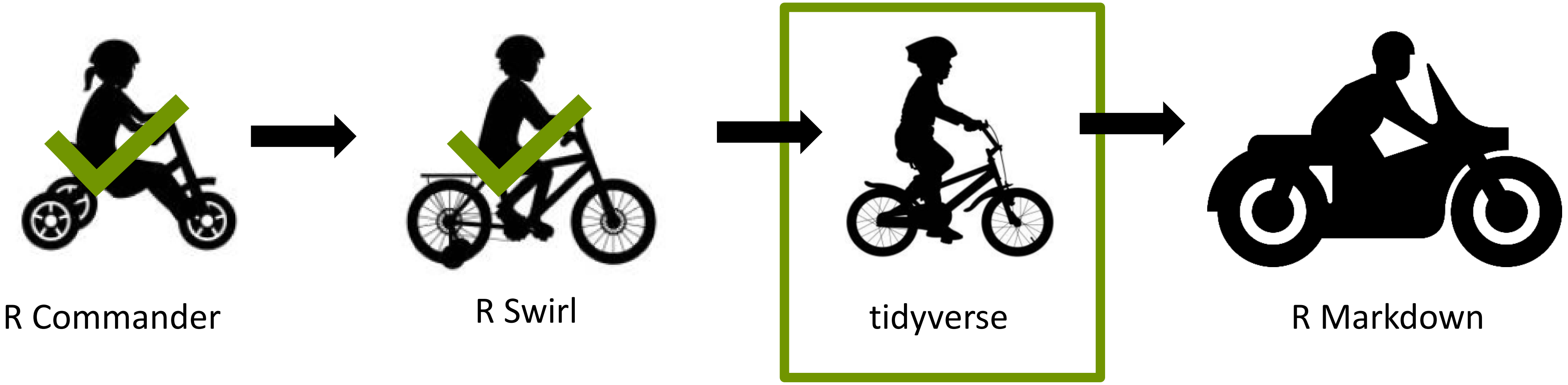


```
x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE)

# modify a single element
x[2,2] <- 10
x
# modify elements less than 5
x[x<5] <- 0
x
```

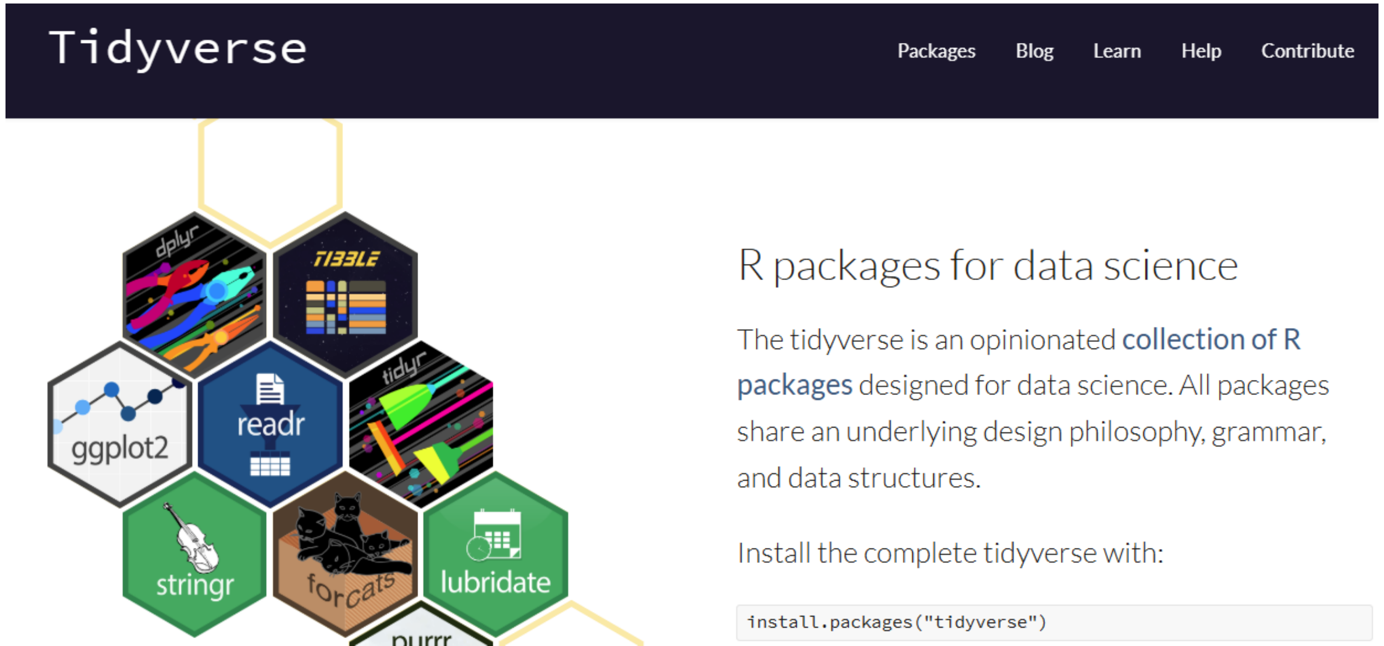
Level up: Tidyverse

- R swirl is great for learning functions, but it is not helpful to guide you for specific things when working with your own data.
- We’ll use Tidyverse to continue to build skills to investigate and understand our data.
- However, we’ll leverage functions that we learned using R swirl and expand upon them





What is Tidyverse?



R packages for data science

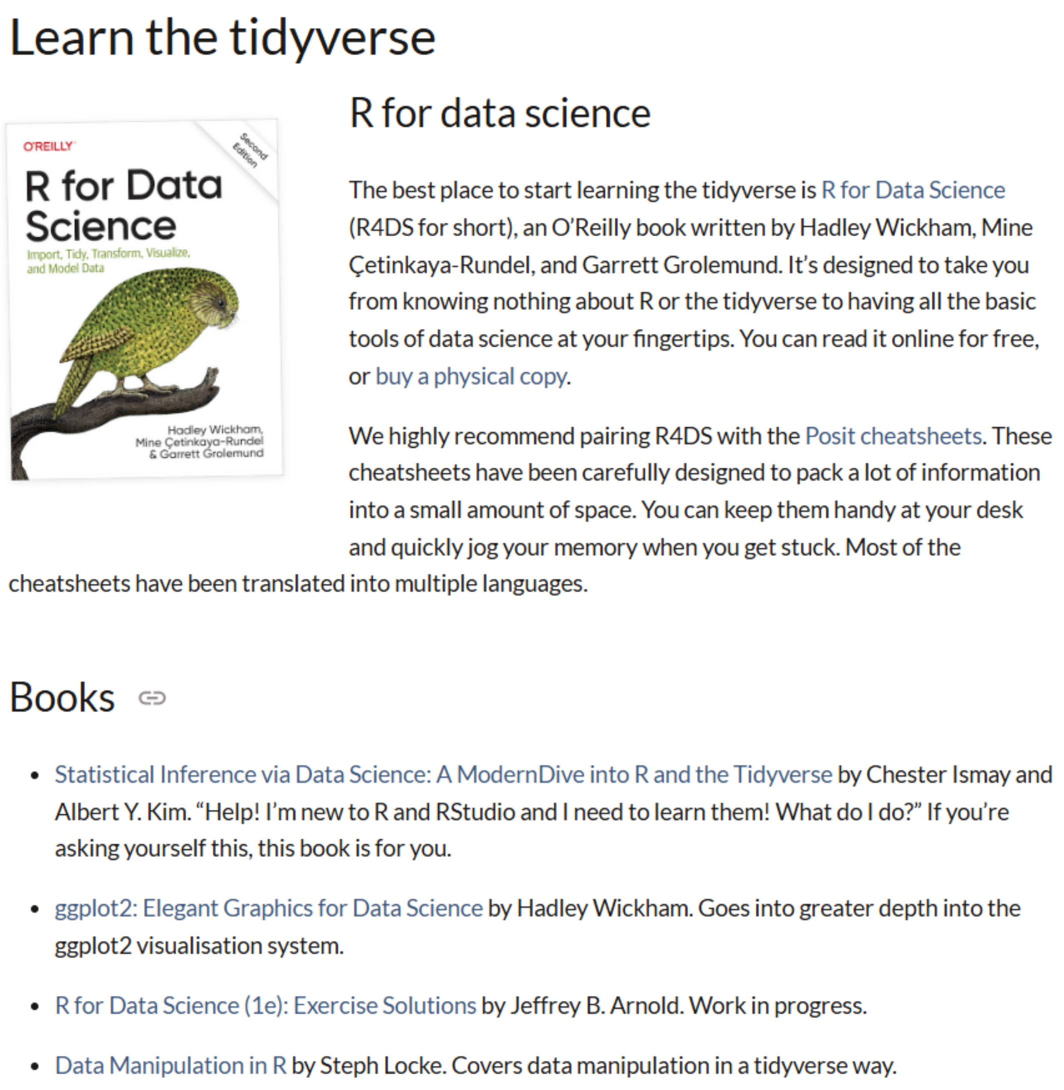
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

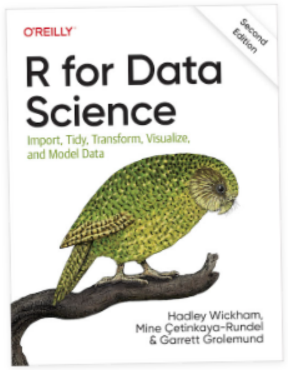
https://www.tidyverse.org/

Version 2024.12.1+563
Copyright (C) 2025 Posit Software, PBC



Learn the tidyverse

R for data science


O'REILLY
Second Edition
R for Data Science
Import, Tidy, Transform, Visualize, and Model Data
Hadley Wickham, Mine Çetinkaya-Rundel & Garrett Grolemund

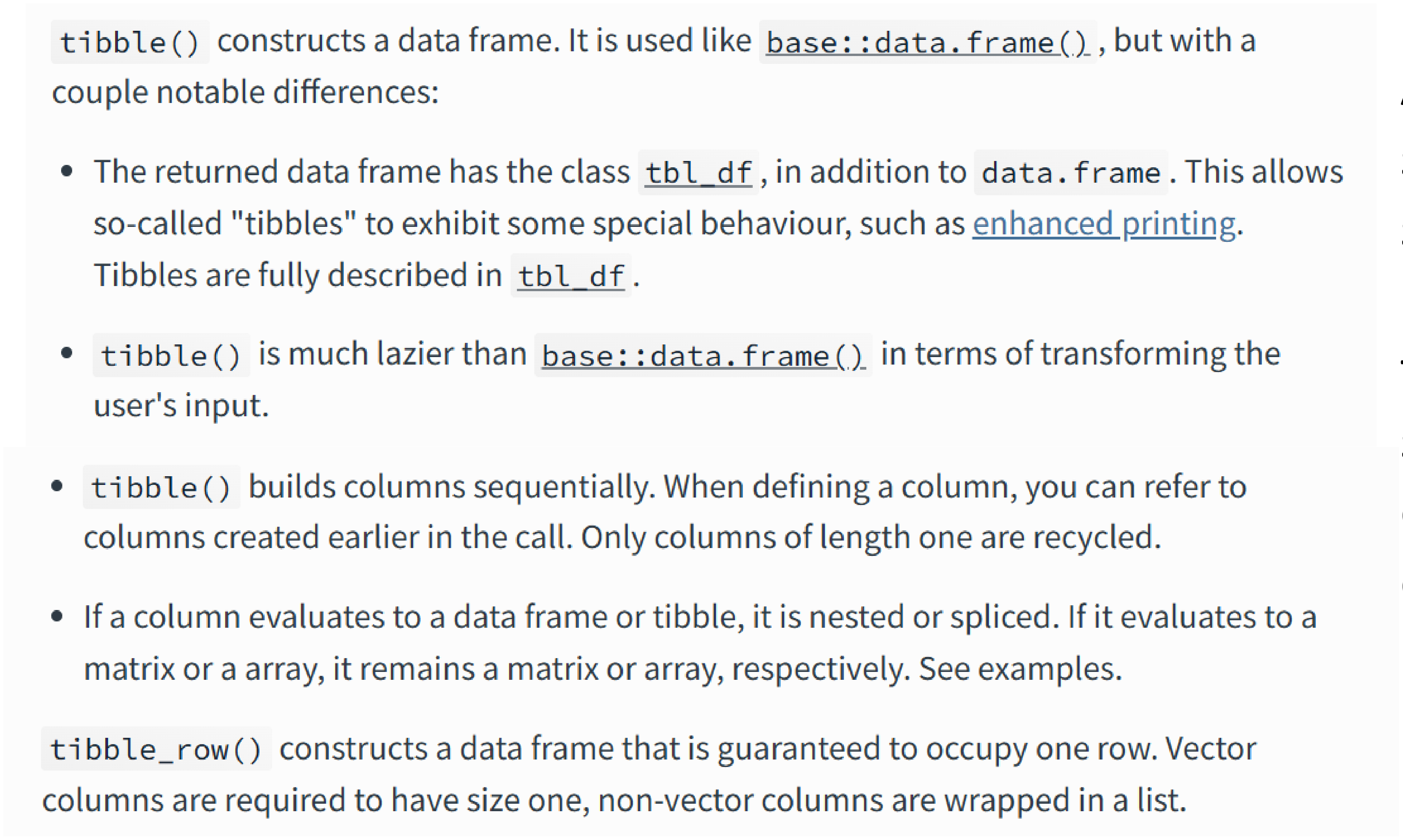
The best place to start learning the tidyverse is R for Data Science (R4DS for short), an O'Reilly book written by Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund. It's designed to take you from knowing nothing about R or the tidyverse to having all the basic tools of data science at your fingertips. You can read it online for free, or buy a physical copy.

We highly recommend pairing R4DS with the Posit cheatsheets. These cheatsheets have been carefully designed to pack a lot of information into a small amount of space. You can keep them handy at your desk and quickly jog your memory when you get stuck. Most of the cheatsheets have been translated into multiple languages.

Books

- Statistical Inference via Data Science: A ModernDive into R and the Tidyverse by Chester Ismay and Albert Y. Kim. "Help! I'm new to R and RStudio and I need to learn them! What do I do?" If you're asking yourself this, this book is for you.
- ggplot2: Elegant Graphics for Data Science by Hadley Wickham. Goes into greater depth into the ggplot2 visualisation system.
- R for Data Science (1e): Exercise Solutions by Jeffrey B. Arnold. Work in progress.
- Data Manipulation in R by Steph Locke. Covers data manipulation in a tidyverse way.

What terminology is needed for data cleaning with tidyverse?



`tibble()` constructs a data frame. It is used like `base::data.frame()`, but with a couple notable differences:

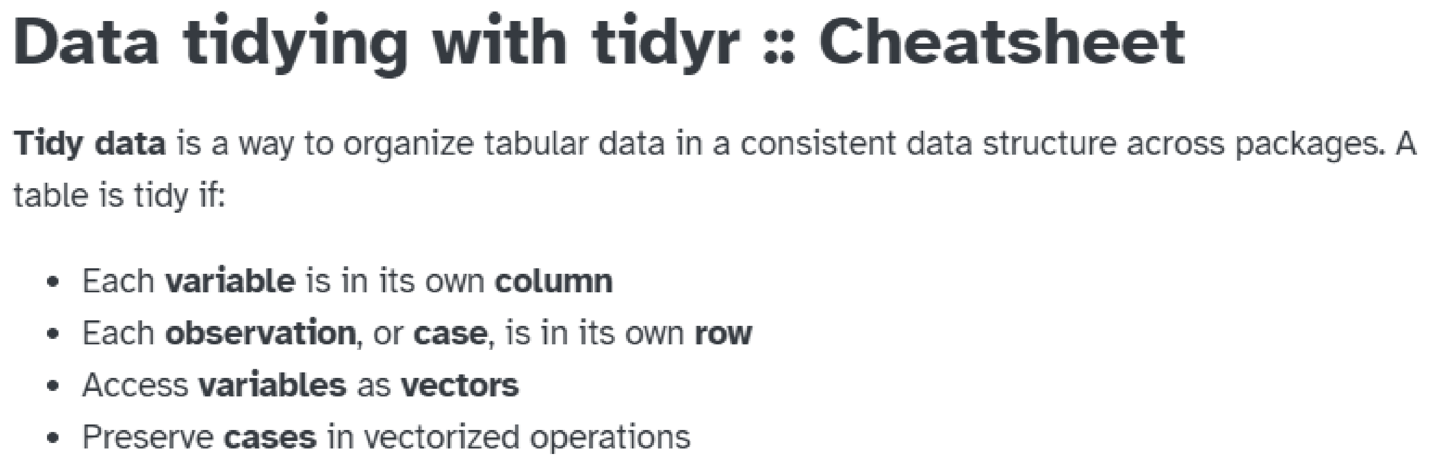
- The returned data frame has the class `tbl_df`, in addition to `data.frame`. This allows so-called "tibbles" to exhibit some special behaviour, such as enhanced printing. Tibbles are fully described in `tbl_df`.
- `tibble()` is much lazier than `base::data.frame()` in terms of transforming the user's input.
- `tibble()` builds columns sequentially. When defining a column, you can refer to columns created earlier in the call. Only columns of length one are recycled.
- If a column evaluates to a data frame or tibble, it is nested or spliced. If it evaluates to a matrix or a array, it remains a matrix or array, respectively. See examples.

`tibble_row()` constructs a data frame that is guaranteed to occupy one row. Vector columns are required to have size one, non-vector columns are wrapped in a list.

A tibble() is a special kind of data structure that is used in the tidyverse & has special properties, different from dataframes

Tibbles vs Data Frames vs Matrices

- A **matrix** can only contain numbers but can be used more efficiently if your data are only numerical
- A **data frame** is like excel file with a list of equal length vectors – good for data storage of all types
- A **tibble** is a special type of data frame with special functions for printing and subsetting; some functions don't work with tibbles. Use "as.data.frame()" to turn the tibble into a data frame for use.



Data tidying with tidyr :: Cheatsheet

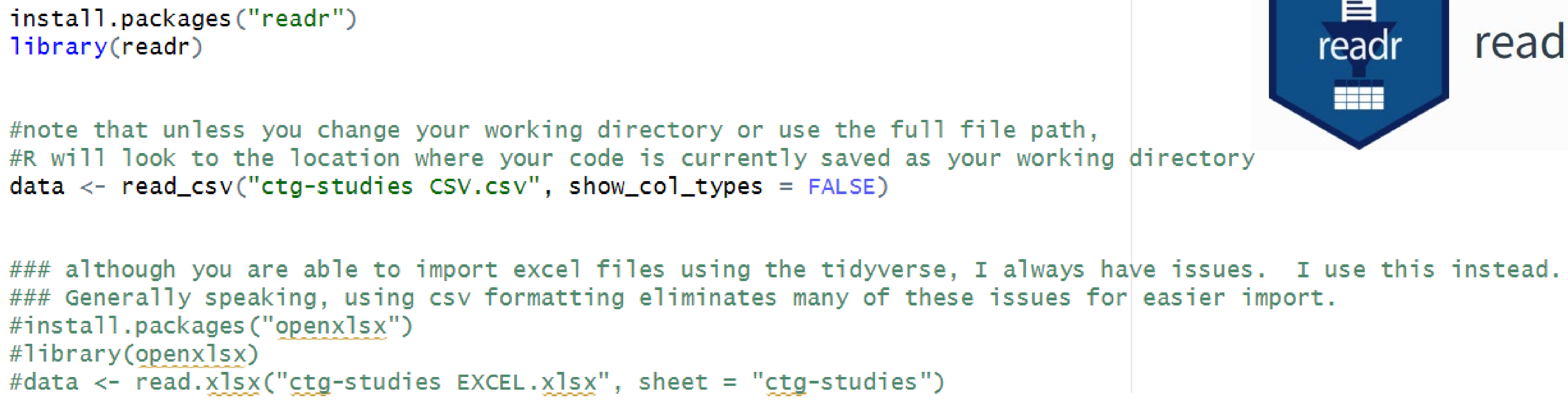
Tidy data is a way to organize tabular data in a consistent data structure across packages. A table is tidy if:

- Each **variable** is in its own **column**
- Each **observation**, or **case**, is in its own **row**
- Access **variables** as **vectors**
- Preserve **cases** in vectorized operations

How to Import Data?

We’ll use the *readr* package to import our csv dataset.

I always recommend using .csv files vs .xlsx files as embedded formatting can cause import issues.



```
install.packages("readr")
library(readr)


#note that unless you change your working directory or use the full file path,
#R will look to the location where your code is currently saved as your working directory
data <- read_csv("ctg-studies CSV.csv", show_col_types = FALSE)


### although you are able to import excel files using the tidyverse, I always have issues.  I use this instead.
### Generally speaking, using csv formatting eliminates many of these issues for easier import.
#install.packages("openxlsx")
#library(openxlsx)
#data <- read.xlsx("ctg-studies EXCEL.xlsx", sheet = "ctg-studies")
```

A note on file paths and common issues…

Note that R expects forward slashes (/) in your filepath

If you have a PC, the default file path uses back slashes (\) and you will get the following error

```
Error: '\U' used without hex digits in character string (<input>:1:23)
```

Output from PC => C:\Users\learyev\R is for All

Corrected for R => C:/Users/learyev/R is for All

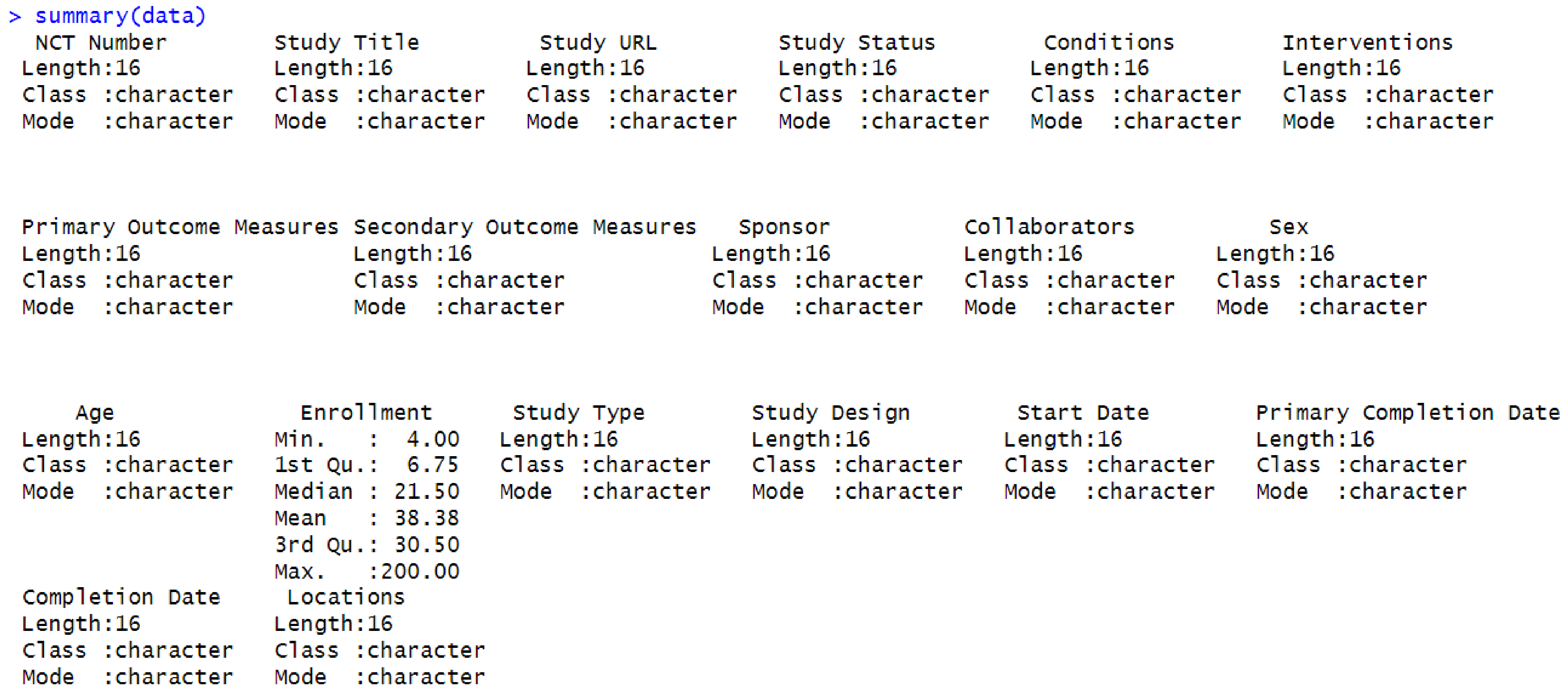
Also correct for R => C:\\Users\\learyev\\R is for all

This is also important to note if you switch between a PC and Mac computer as Macs have different formatting for file paths

So, you may want to create and label the different file

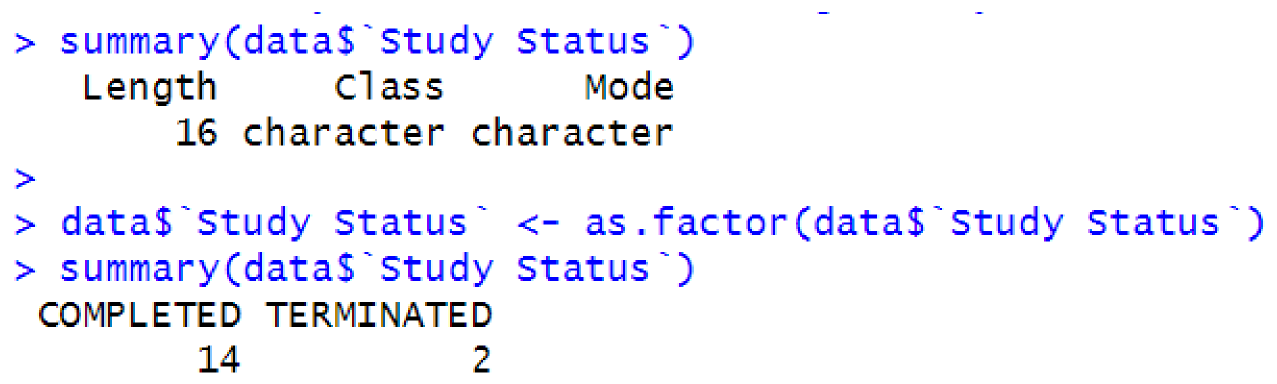
 paths for what computer you are using!

How to summarize data?



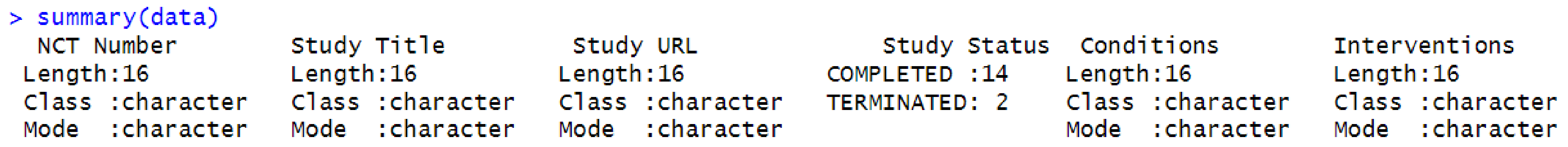
```
> summary(data)
  NCT Number        Study Title          Study URL         Study Status        Conditions        Interventions
 Length:16          Length:16          Length:16          Length:16          Length:16          Length:16
 Class :character   Class :character   Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character



 Primary Outcome Measures Secondary Outcome Measures   Sponsor          Collaborators          Sex
 Length:16                Length:16                  Length:16          Length:16          Length:16
 Class :character         Class :character           Class :character   Class :character   Class :character
 Mode  :character         Mode  :character           Mode  :character   Mode  :character   Mode  :character



     Age              Enrollment      Study Type        Study Design        Start Date        Primary Completion Date
 Length:16          Min.   :  4.00   Length:16          Length:16          Length:16          Length:16
 Class :character   1st Qu.:  6.75   Class :character   Class :character   Class :character   Class :character
 Mode  :character   Median : 21.50   Mode  :character   Mode  :character   Mode  :character   Mode  :character
                    Mean   : 38.38
                    3rd Qu.: 30.50
                    Max.   :200.00
 Completion Date     Locations
 Length:16          Length:16
 Class :character   Class :character
 Mode  :character   Mode  :character
```

Change variable type: character -> factor

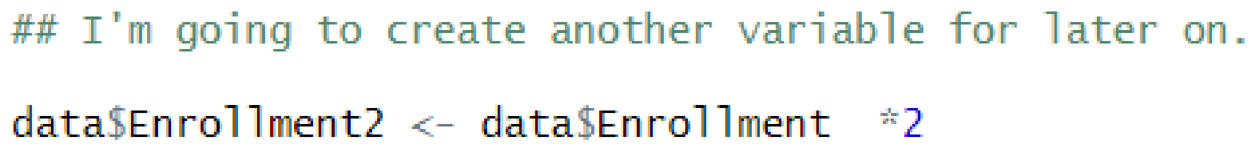


```
> summary(data$`Study Status`)
   Length     Class      Mode 
       16 character character 
>
> data$`Study Status` <- as.factor(data$`Study Status`)
> summary(data$`Study Status`)
 COMPLETED TERMINATED 
        14          2 
```

This allows us to summarize our data by study status



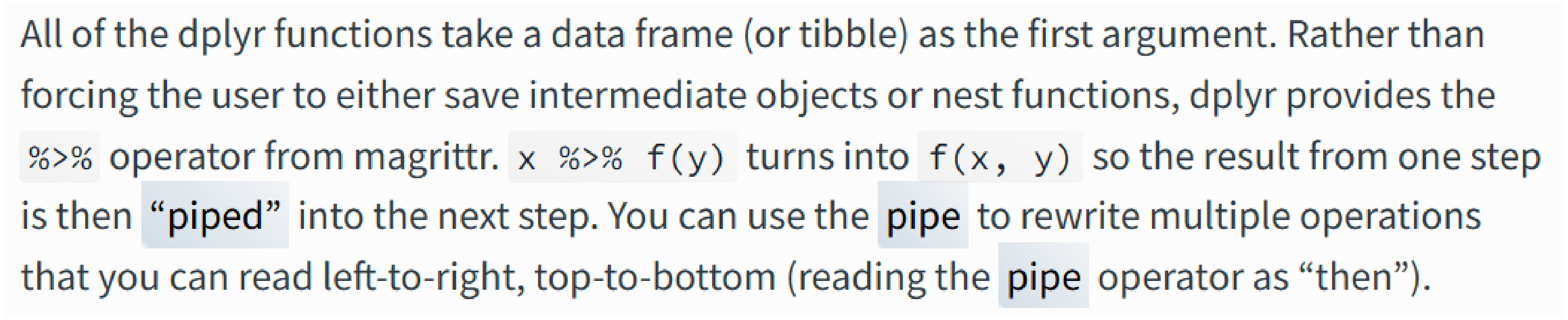
```
> summary(data)
  NCT Number         Study Title         Study URL              Study Status   Conditions         Interventions     
 Length:16          Length:16          Length:16          COMPLETED :14    Length:16          Length:16         
 Class :character   Class :character   Class :character   TERMINATED: 2    Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character                    Mode  :character   Mode  :character  
```



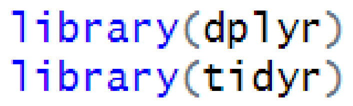
```
## I'm going to create another variable for later on.

data$Enrollment2 <- data$Enrollment  *2
```

Using tidyverse: summarize your data

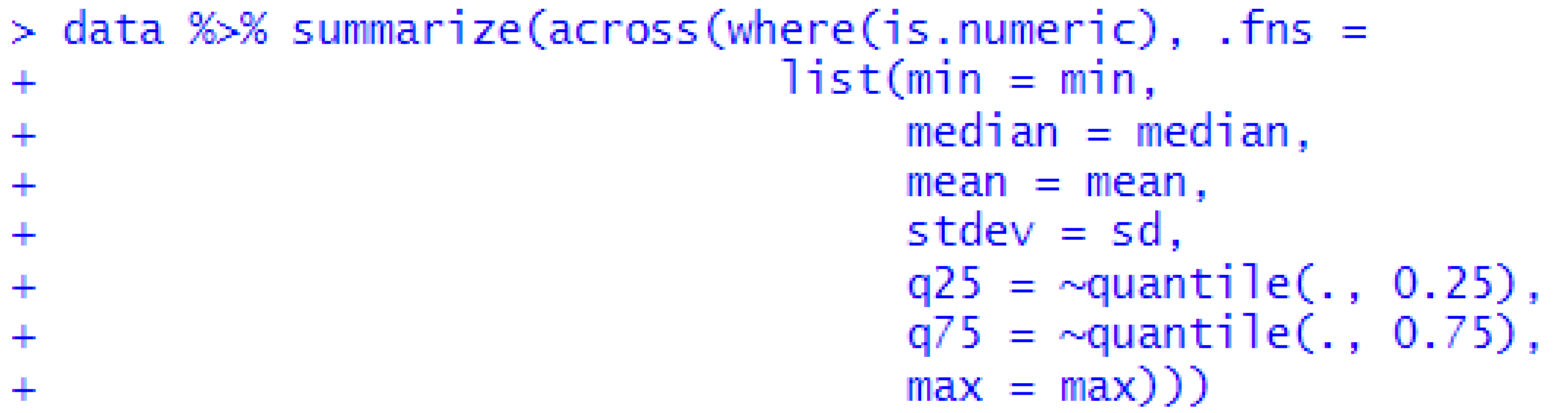
The summary() function provides useful information, but additional functionality would help with data cleaning and management.



All of the dplyr functions take a data frame (or tibble) as the first argument. Rather than forcing the user to either save intermediate objects or nest functions, dplyr provides the `%>%` operator from magrittr. `x %>% f(y)` turns into `f(x, y)` so the result from one step is then "piped" into the next step. You can use the pipe to rewrite multiple operations that you can read left-to-right, top-to-bottom (reading the pipe operator as "then").



```
library(dplyr)
library(tidyr)
```



```
> data %>% summarize(across(where(is.numeric), .fns =
+                             list(min = min,
+                                  median = median,
+                                  mean = mean,
+                                  stdev = sd,
+                                  q25 = ~quantile(., 0.25),
+                                  q75 = ~quantile(., 0.75),
+                                  max = max)))
```

Functions from tidyverse to create summary functions

`summarise()` creates a new data frame. It returns one row for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarising all observations in the input. It will contain one column for each grouping variable and one column for each of the summary statistics that you have specified.

`summarise()` and `summarize()` are synonyms.

purrr

We recommend you use the base `|>` pipe instead of magrittr's `%>%`.

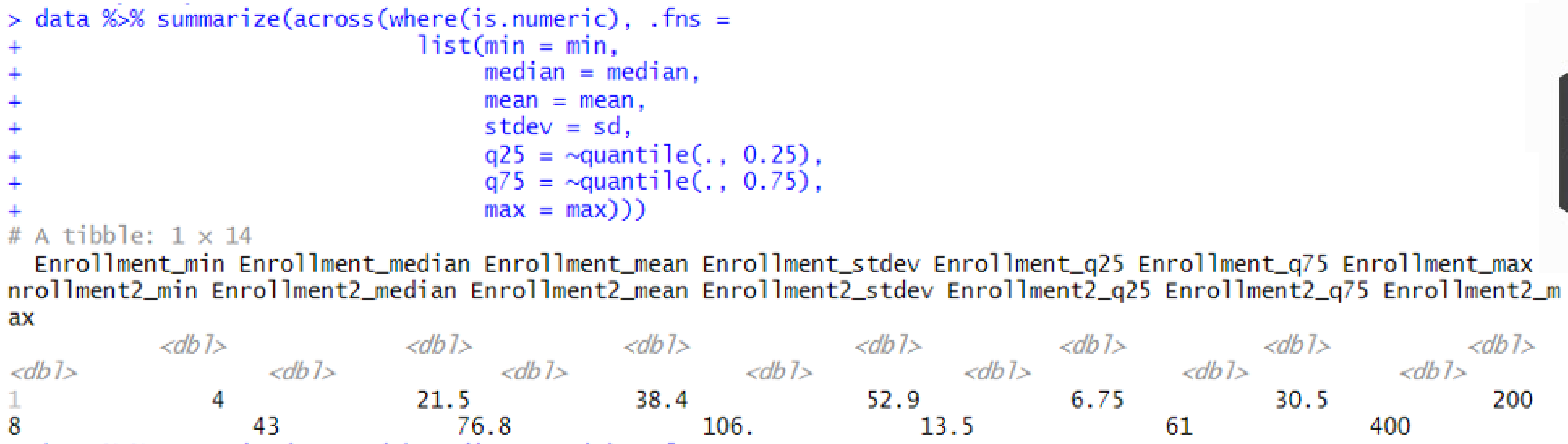
`across()` makes it easy to apply the same transformation to multiple columns, allowing you to use `select()` semantics inside in "data-masking" functions like `summarise()` and `mutate()`. See `vignette("colwise")` for more details.

You can use purrr-like formulas as a shortcut for creating a function on the spot. These expressions are equivalent:

```
iris %>% select(where(is.numeric))

iris %>% select(where(function(x) is.numeric(x)))
```

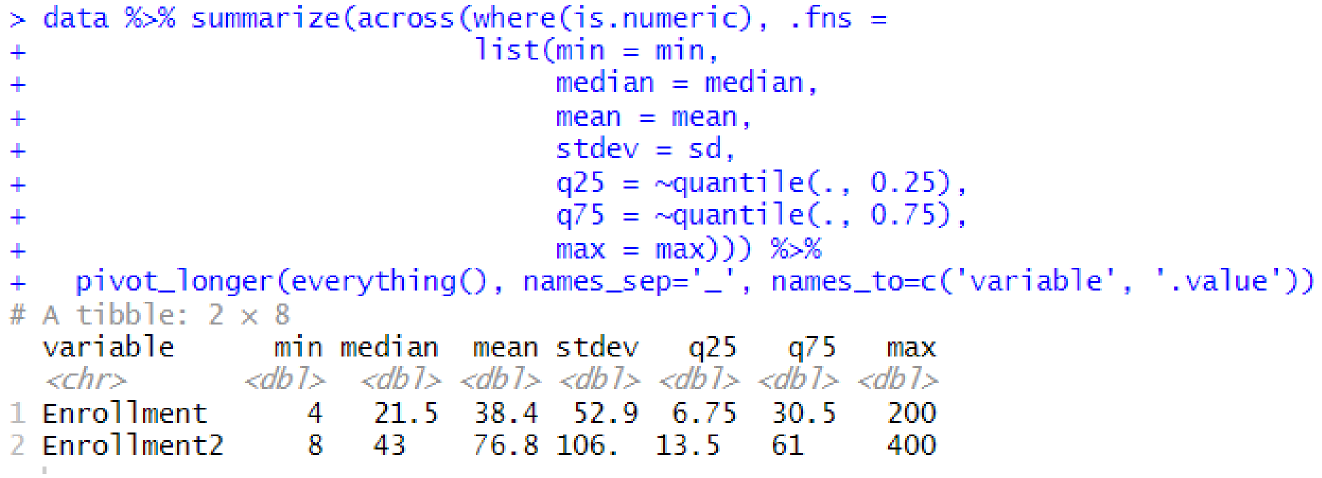
```
> data %>% summarize(across(where(is.numeric), .fns =
+                               list(min = min,
+                                    median = median,
+                                    mean = mean,
+                                    stdev = sd,
+                                    q25 = ~quantile(., 0.25),
+                                    q75 = ~quantile(., 0.75),
+                                    max = max)))
```

Functions from tidyverse to make data cleaning easier



```
> data %>% summarize(across(where(is.numeric), .fns =
+                            list(min = min,
+                                 median = median,
+                                 mean = mean,
+                                 stdev = sd,
+                                 q25 = ~quantile(., 0.25),
+                                 q75 = ~quantile(., 0.75),
+                                 max = max)))
# A tibble: 1 × 14
  Enrollment_min Enrollment_median Enrollment_mean Enrollment_stdev Enrollment_q25 Enrollment_q75 Enrollment_max
nrollment2_min Enrollment2_median Enrollment2_mean Enrollment2_stdev Enrollment2_q25 Enrollment2_q75 Enrollment2_m
ax
           <dbl>             <dbl>           <dbl>            <dbl>          <dbl>          <dbl>          <dbl>
<dbl>              <dbl>            <dbl>             <dbl>           <dbl>           <dbl>           <dbl>
1              4              21.5            38.4             52.9           6.75           30.5            200
8                43               76.8            106.              13.5              61             400
```

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`



```
> data %>% summarize(across(where(is.numeric), .fns =
+                            list(min = min,
+                                 median = median,
+                                 mean = mean,
+                                 stdev = sd,
+                                 q25 = ~quantile(., 0.25),
+                                 q75 = ~quantile(., 0.75),
+                                 max = max))) %>%
+   pivot_longer(everything(), names_sep='_', names_to=c('variable', '.value'))
# A tibble: 2 × 8
  variable      min median  mean stdev   q25   q75   max
  <chr>       <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Enrollment      4   21.5  38.4  52.9  6.75  30.5   200
2 Enrollment2     8   43    76.8 106.  13.5  61      400
```

This tells R to make the wide output, long and to split the names at the underscore

This is very helpful if you have multiple continuous variables

Functions from tidyverse to make data cleaning easier

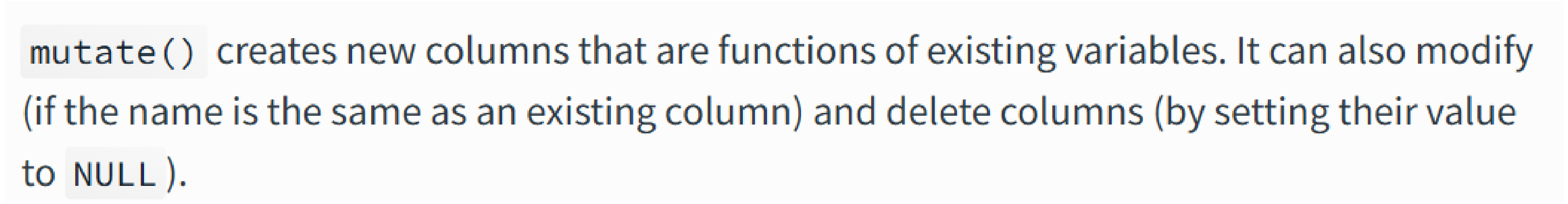
`mutate()` creates new columns that are functions of existing variables. It can also modify (if the name is the same as an existing column) and delete columns (by setting their value to `NULL`).

Let's create a new variable from the numerical values of another using the mutate() function from dplyr

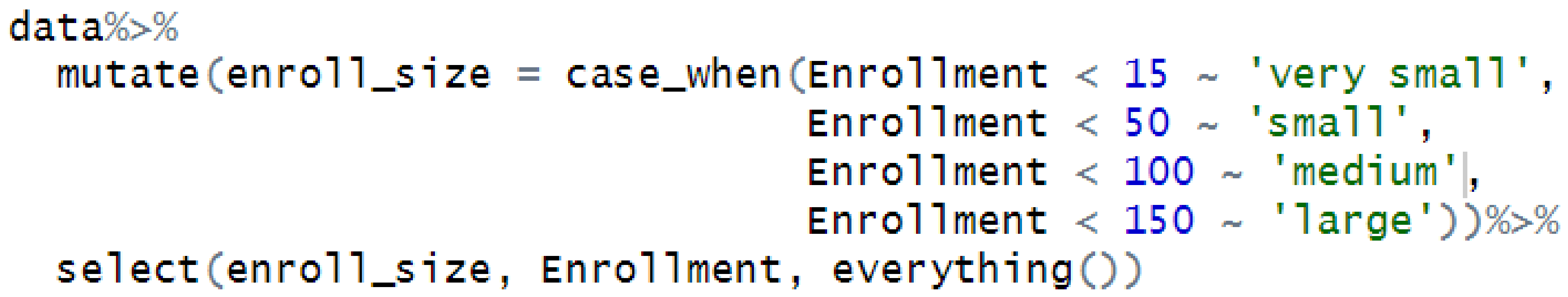
```
data%>%
  mutate(enroll_size = case_when(Enrollment < 15 ~ 'very small',
                                 Enrollment < 50 ~ 'small',
                                 Enrollment < 100 ~ 'medium',
                                 Enrollment < 150 ~ 'large'))%>%
  select(enroll_size, Enrollment, everything())
```

Question: What would happen if you had enrollments with values larger than 150?

Functions from tidyverse to make data cleaning easier

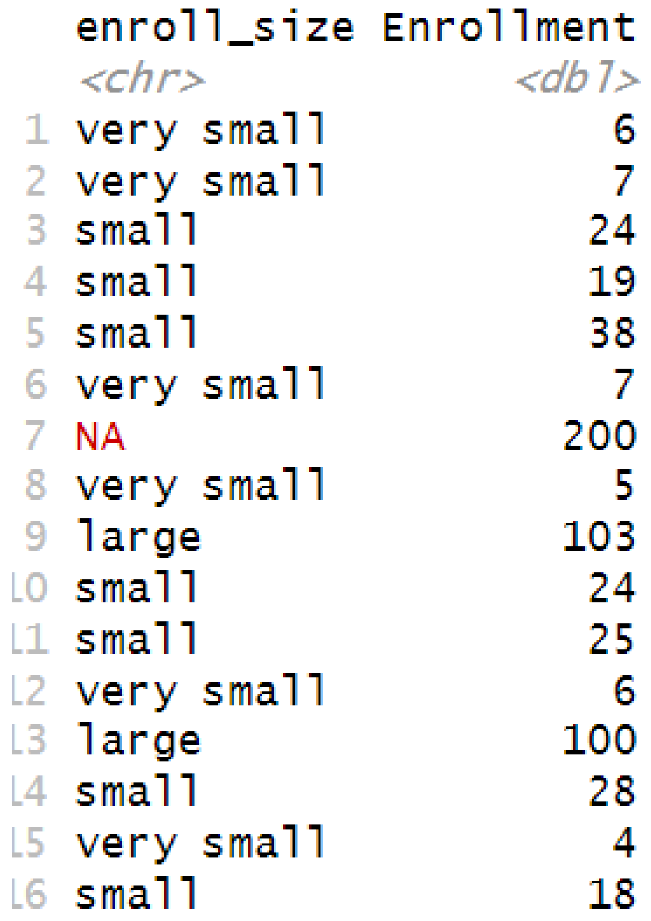

`mutate()` creates new columns that are functions of existing variables. It can also modify (if the name is the same as an existing column) and delete columns (by setting their value to `NULL`).

Let's create a new variable from the numerical values of another using mutate from dplyr


```
data%>%
  mutate(enroll_size = case_when(Enrollment < 15 ~ 'very small',
                                 Enrollment < 50 ~ 'small',
                                 Enrollment < 100 ~ 'medium',
                                 Enrollment < 150 ~ 'large'))%>%
  select(enroll_size, Enrollment, everything())
```

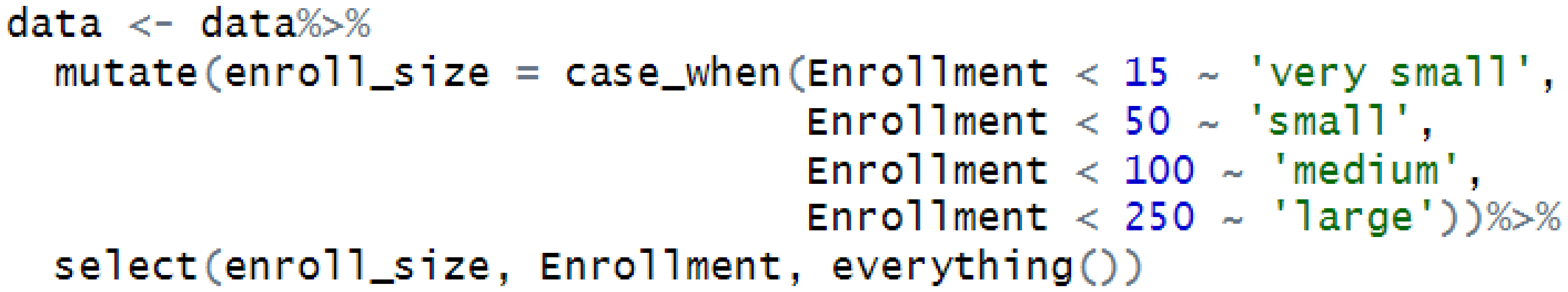
Question: What would happen if you had enrollments with values larger than 150?

Answer: you'd have NA's and may not realize it!

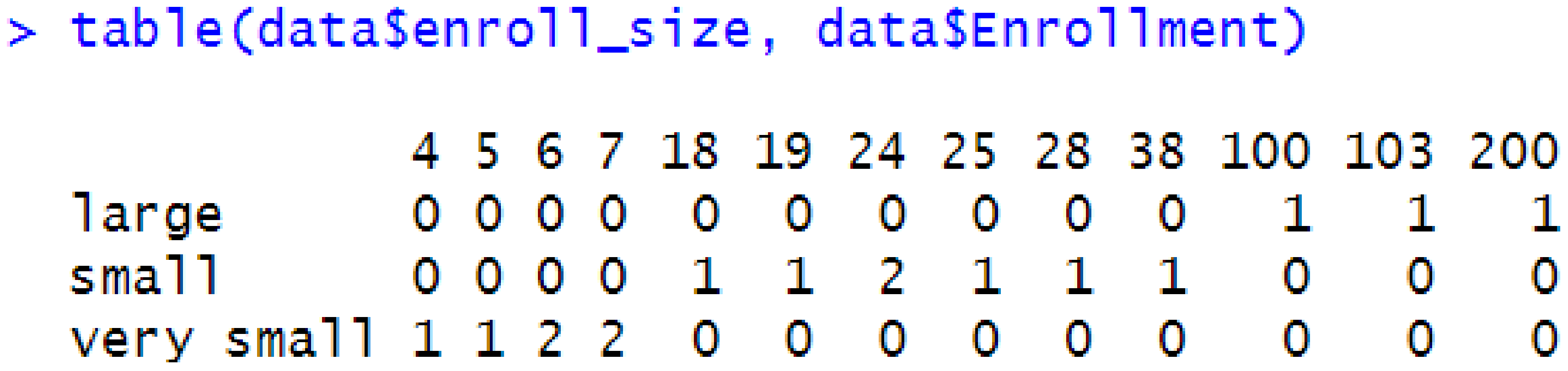

```
   enroll_size Enrollment
   <chr>            <dbl>
 1 very small           6
 2 very small           7
 3 small               24
 4 small               19
 5 small               38
 6 very small           7
 7 NA                 200
 8 very small           5
 9 large              103
10 small               24
11 small               25
12 very small           6
13 large              100
14 small               28
15 very small           4
16 small               18
```

Functions from tidyverse to make data cleaning easier

Here we do the same but create a dataframe from the resulting tibble, which is useful for checking



```
data <- data%>%
  mutate(enroll_size = case_when(Enrollment < 15 ~ 'very small',
                                 Enrollment < 50 ~ 'small',
                                 Enrollment < 100 ~ 'medium',
                                 Enrollment < 250 ~ 'large'))%>%
  select(enroll_size, Enrollment, everything())
```



```
> table(data$enroll_size, data$Enrollment)

             4 5 6 7 18 19 24 25 28 38 100 103 200
  large      0 0 0 0  0  0  0  0  0  0   1   1   1
  small      0 0 0 0  1  1  2  1  1  1   0   0   0
  very small 1 1 2 2  0  0  0  0  0  0   0   0   0
```

Functions from tidyverse to make data cleaning easier

Now, what if we wanted to look at summary data by Study Status?

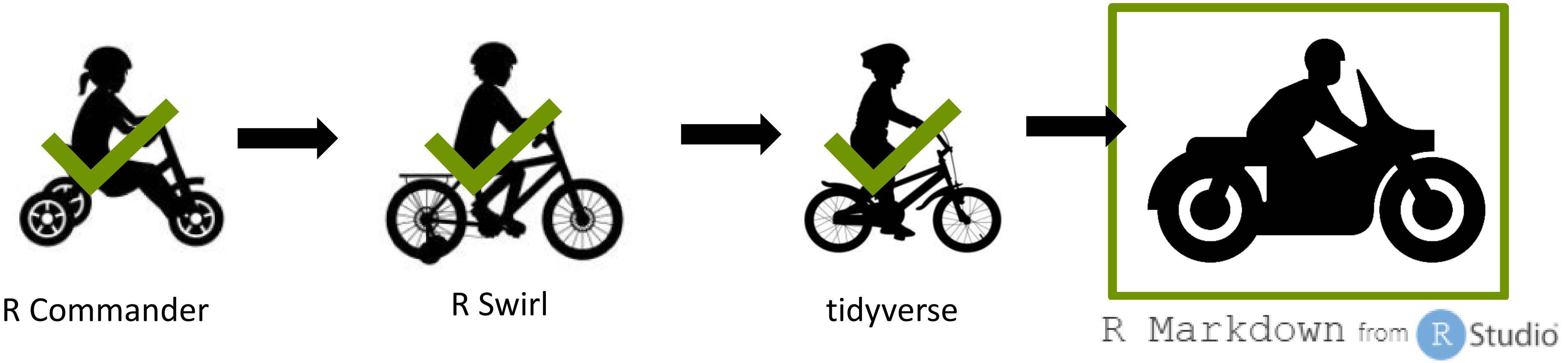
```
data %>%
 group_by(`Study Status`) %>%
 summarize(mean_enroll = mean(Enrollment, na.rm = TRUE),
           std_enroll = sd(Enrollment, na.rm=TRUE),
           iqr_enroll = IQR(Enrollment, na.rm = TRUE),
           n_enroll = n())
```

We can use the group_by() function from tidyverse to group output

```
# A tibble: 2 × 5
  `Study Status` mean_enroll std_enroll iqr_enroll n_enroll
  <fct>                <dbl>      <dbl>      <dbl>    <int>
1 COMPLETED             43.1      55.1        25.8       14
2 TERMINATED             5.5       2.12        1.5        2
```

Level up: R Markdown

- R Markdown is a *file format* for making dynamic documents with R and uses the *rmarkdown* package
- An R Markdown document is written in markdown (an easy-to-write plain text format) and contains chunks of embedded R code and output
- The most important advantage is *project management:* R Markdown files are the source code for rich, reproducible documents – place code, figures, written explanations, decisions with reasoning – all in ONE place

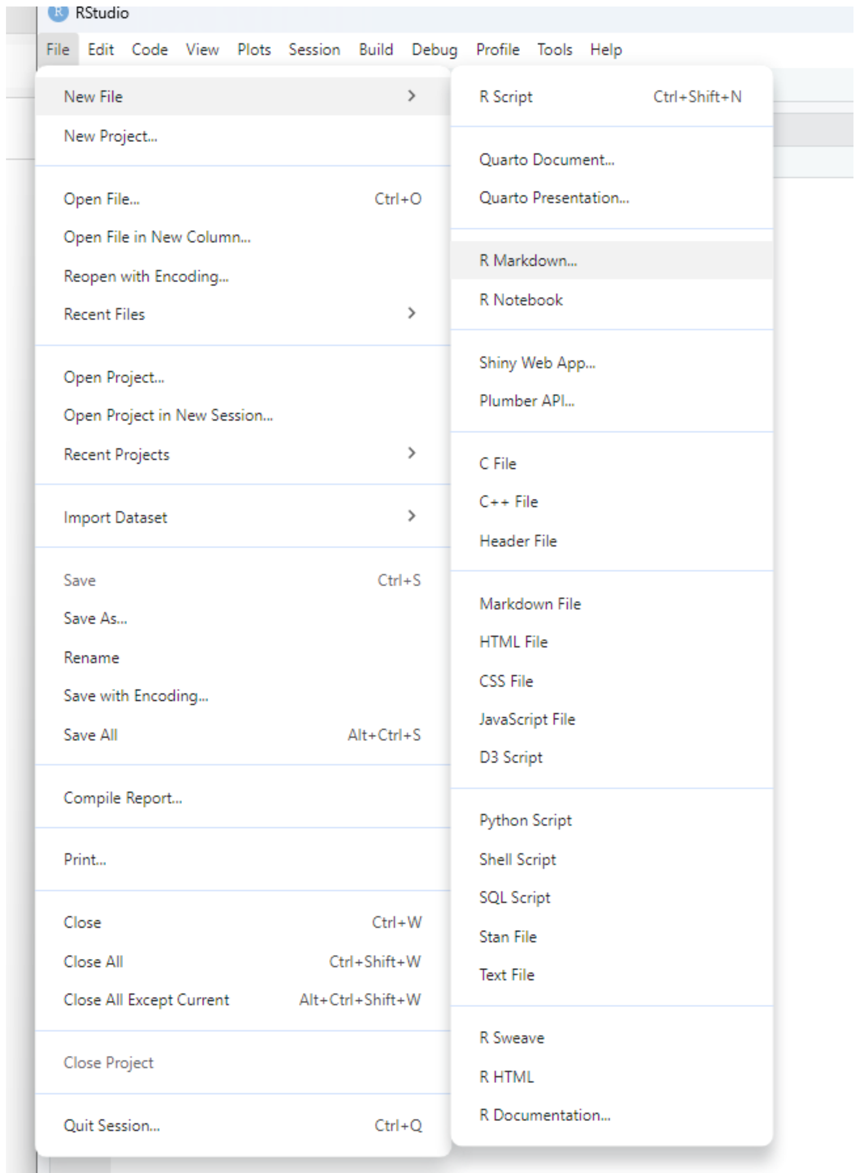


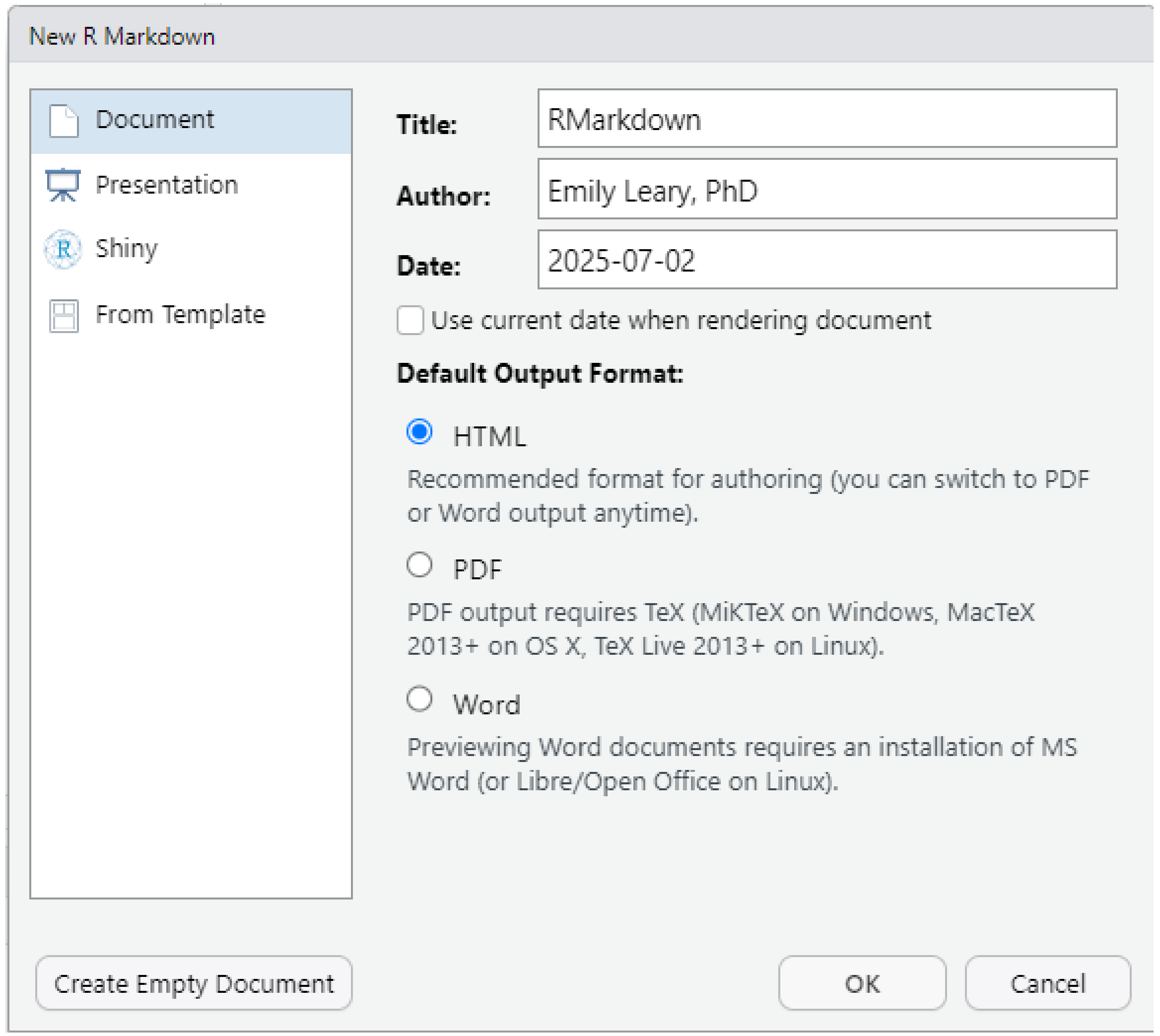
R Markdown Definitions

- **Knit**: Function of the *knitr* package and runs each chunk of R code in the Markdown document and appends the results of the code in the document, next to the code chunk
- **Convert**: changes the file into a new file format: HTML, PDF, .docx
- **Render**: Refers to the two-step process of knitting and converting an R Markdown file

- Tip: Your R Markdown file should not have any spaces in the file name!

Creating a blank R Markdown File





Title: RMarkdown

Author: Emily Leary, PhD

Date: 2025-07-02

☐ Use current date when rendering document

Default Output Format:

● HTML

Recommended format for authoring (you can switch to PDF or Word output anytime).

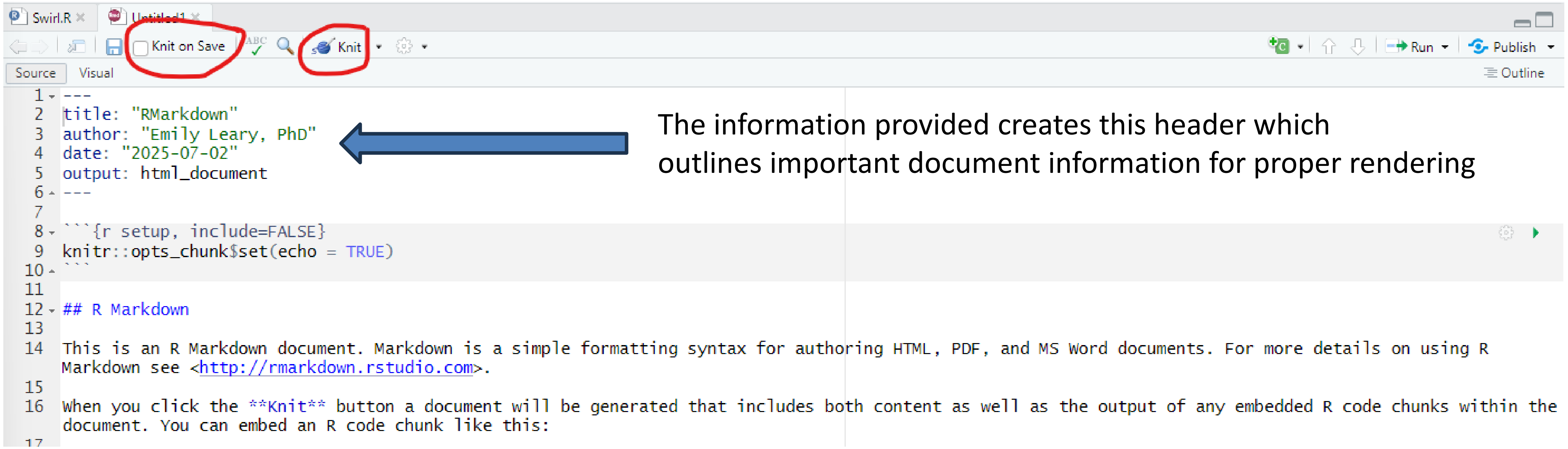
○ PDF

PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

○ Word

Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

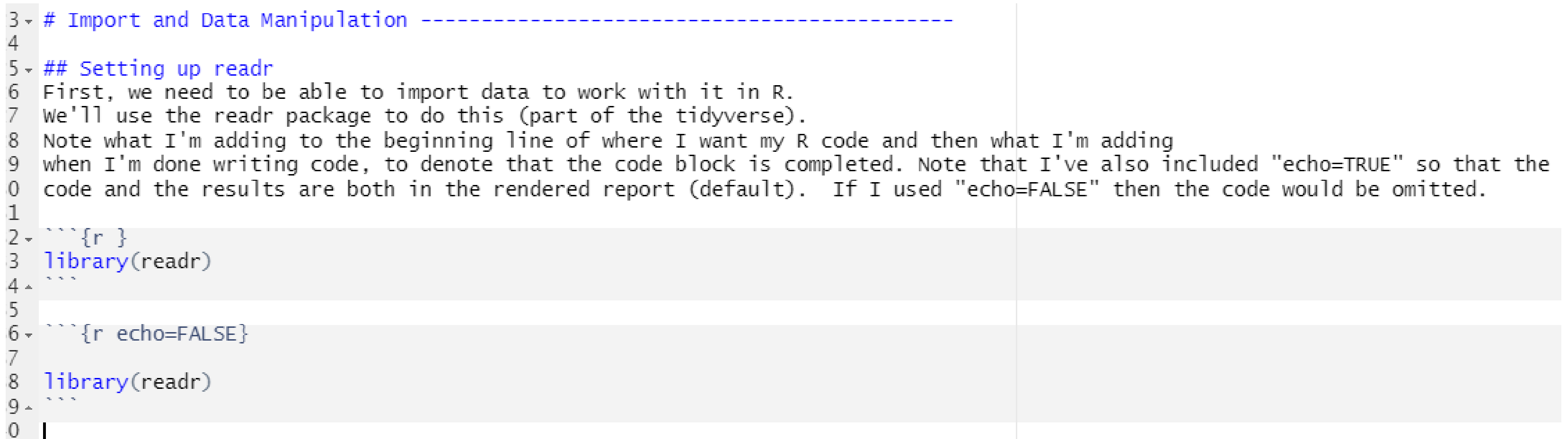
Blank R Markdown File Created



~~~
---
title: "RMarkdown"
author: "Emily Leary, PhD"
date: "2025-07-02"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R
Markdown see <http://rmarkdown.rstudio.com>.

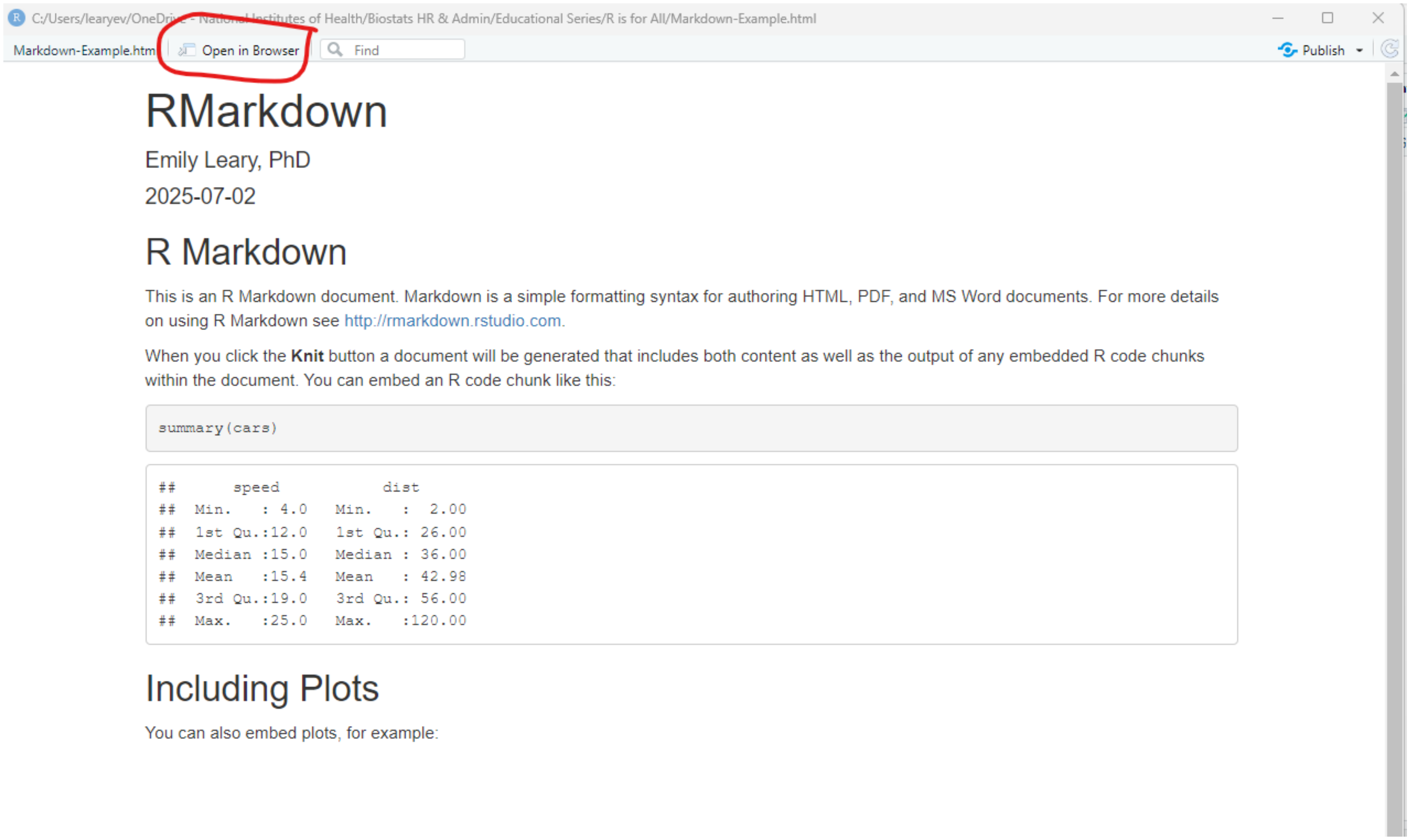
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the
document. You can embed an R code chunk like this:
```

The information provided creates this header which outlines important document information for proper rendering



```
# Import and Data Manipulation ---------------------------------------------

## Setting up readr
First, we need to be able to import data to work with it in R.
We'll use the readr package to do this (part of the tidyverse).
Note what I'm adding to the beginning line of where I want my R code and then what I'm adding
when I'm done writing code, to denote that the code block is completed. Note that I've also included "echo=TRUE" so that the
code and the results are both in the rendered report (default).  If I used "echo=FALSE" then the code would be omitted.

```{r }
library(readr)
```

```{r echo=FALSE}

library(readr)
```
```
```

Blank R Markdown File Rendered


C:/Users/learyev/OneDrive - National Institutes of Health/Biostats HR & Admin/Educational Series/R is for All/Markdown-Example.html
Markdown-Example.html
Open in Browser
Find
Publish

RMarkdown

Emily Leary, PhD

2025-07-02

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

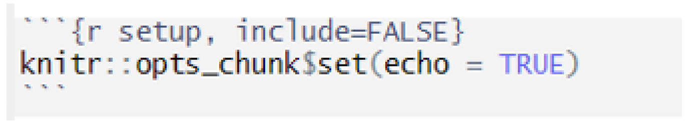
```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:

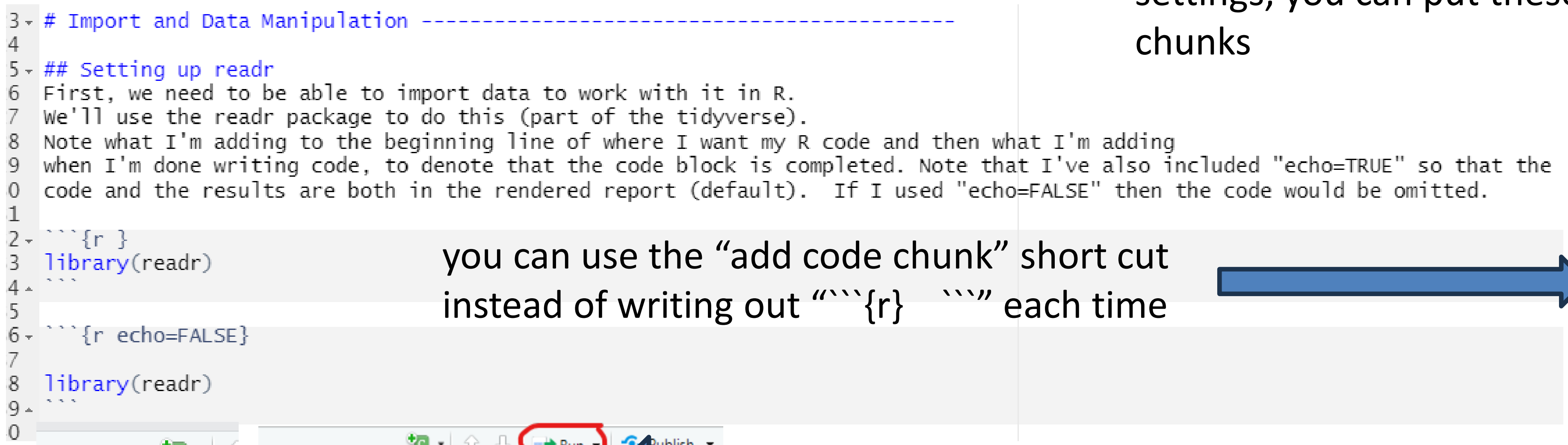
- Opens up in a new R Studio window
- Can also open in browser
- Can also “Publish” but won’t cover this

Editing an R Markdown file & Tips



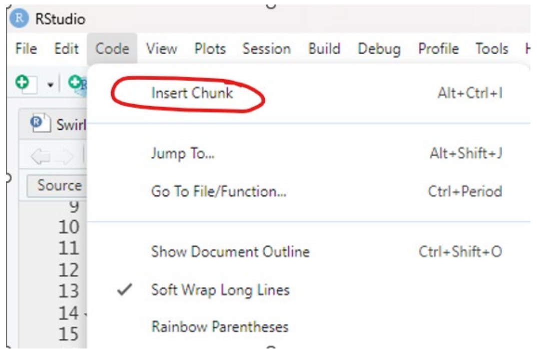
~~~
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
~~~

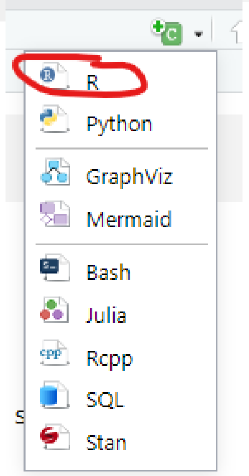
The *r setup* code chunk sets the settings for **all** R code chunks in the Markdown file - if you need different settings, you can put these in the individual code chunks

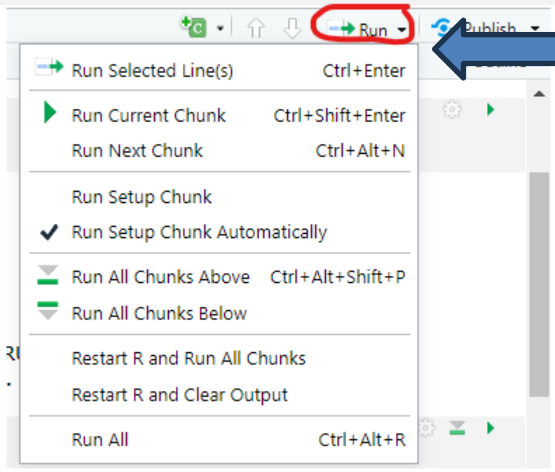


~~~
# Import and Data Manipulation ---------------------------------------------

## Setting up readr
First, we need to be able to import data to work with it in R.
We'll use the readr package to do this (part of the tidyverse).
Note what I'm adding to the beginning line of where I want my R code and then what I'm adding
when I'm done writing code, to denote that the code block is completed. Note that I've also included "echo=TRUE" so that the
code and the results are both in the rendered report (default).  If I used "echo=FALSE" then the code would be omitted.

```{r }
library(readr)
```

```{r echo=FALSE}

library(readr)
```
~~~

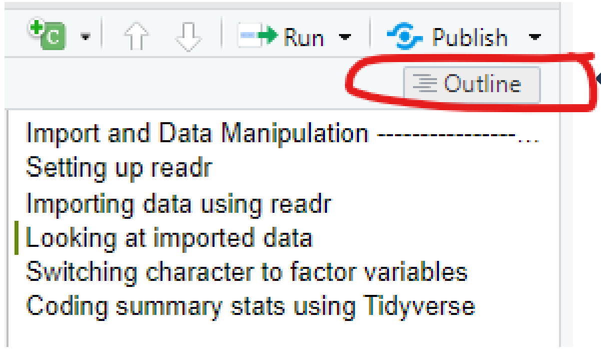
you can use the “add code chunk” short cut instead of writing out “```{r}   ```” each time


RStudio
File Edit Code View Plots Session Build Debug Profile Tools
Insert Chunk Alt+Ctrl+I
Jump To... Alt+Shift+J
Go To File/Function... Ctrl+Period
Show Document Outline Ctrl+Shift+O
Soft Wrap Long Lines
Rainbow Parentheses
Source



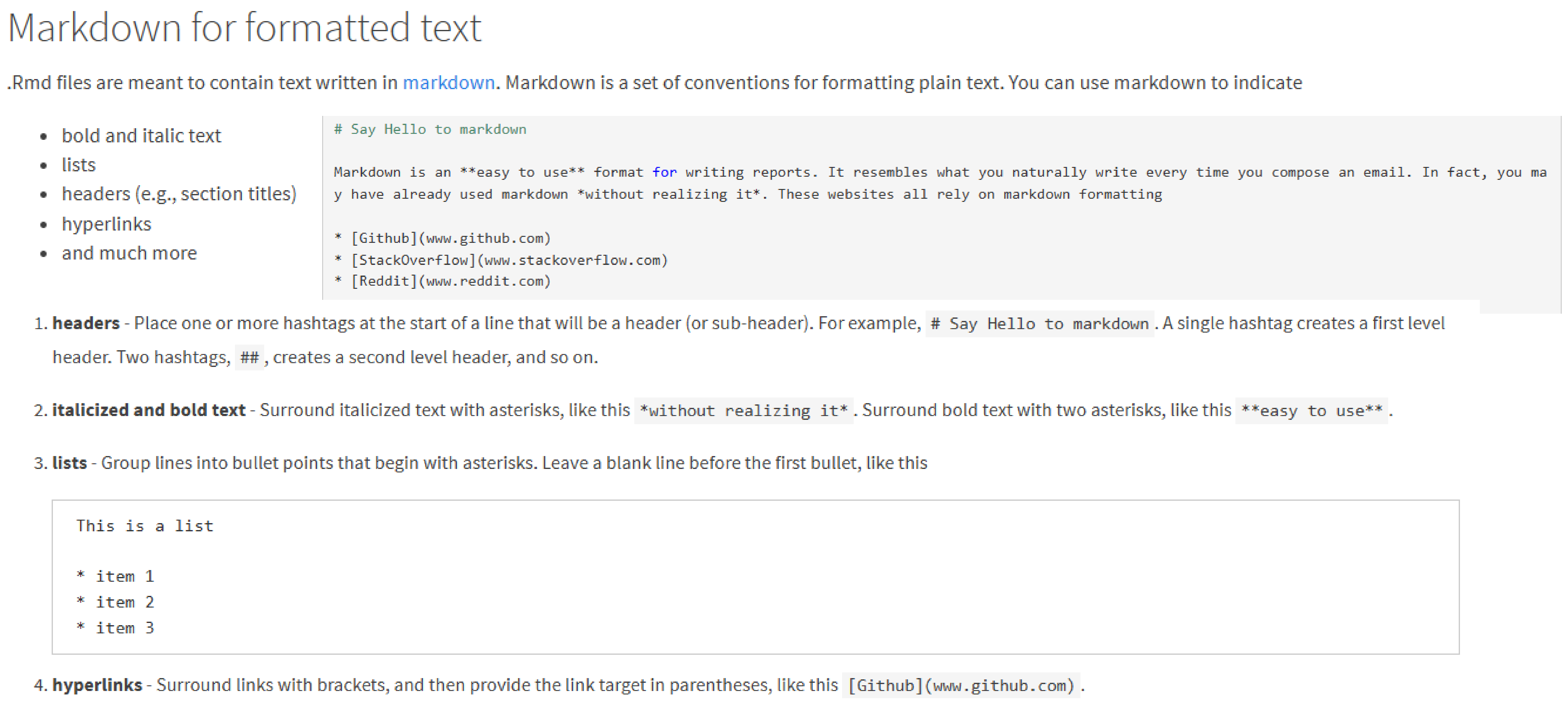

Run
Publish
Run Selected Line(s) Ctrl+Enter
Run Current Chunk Ctrl+Shift+Enter
Run Next Chunk Ctrl+Alt+N
Run Setup Chunk
Run Setup Chunk Automatically
Run All Chunks Above Ctrl+Alt+Shift+P
Run All Chunks Below
Restart R and Run All Chunks
Restart R and Clear Output
Run All Ctrl+Alt+R

You can run specific things using the Run menu


Run
Publish
Outline
Import and Data Manipulation ----------------...
Setting up readr
Importing data using readr
Looking at imported data
Switching character to factor variables
Coding summary stats using Tidyverse

You can navigate your file using the outline button

Add formatting and text to R Markdown files



Markdown for formatted text

.Rmd files are meant to contain text written in markdown. Markdown is a set of conventions for formatting plain text. You can use markdown to indicate

- bold and italic text
- lists
- headers (e.g., section titles)
- hyperlinks
- and much more

```
# Say Hello to markdown

Markdown is an **easy to use** format for writing reports. It resembles what you naturally write every time you compose an email. In fact, you ma
y have already used markdown *without realizing it*. These websites all rely on markdown formatting

* [Github](www.github.com)
* [StackOverflow](www.stackoverflow.com)
* [Reddit](www.reddit.com)
```

1. **headers** - Place one or more hashtags at the start of a line that will be a header (or sub-header). For example, `# Say Hello to markdown`. A single hashtag creates a first level header. Two hashtags, `##`, creates a second level header, and so on.

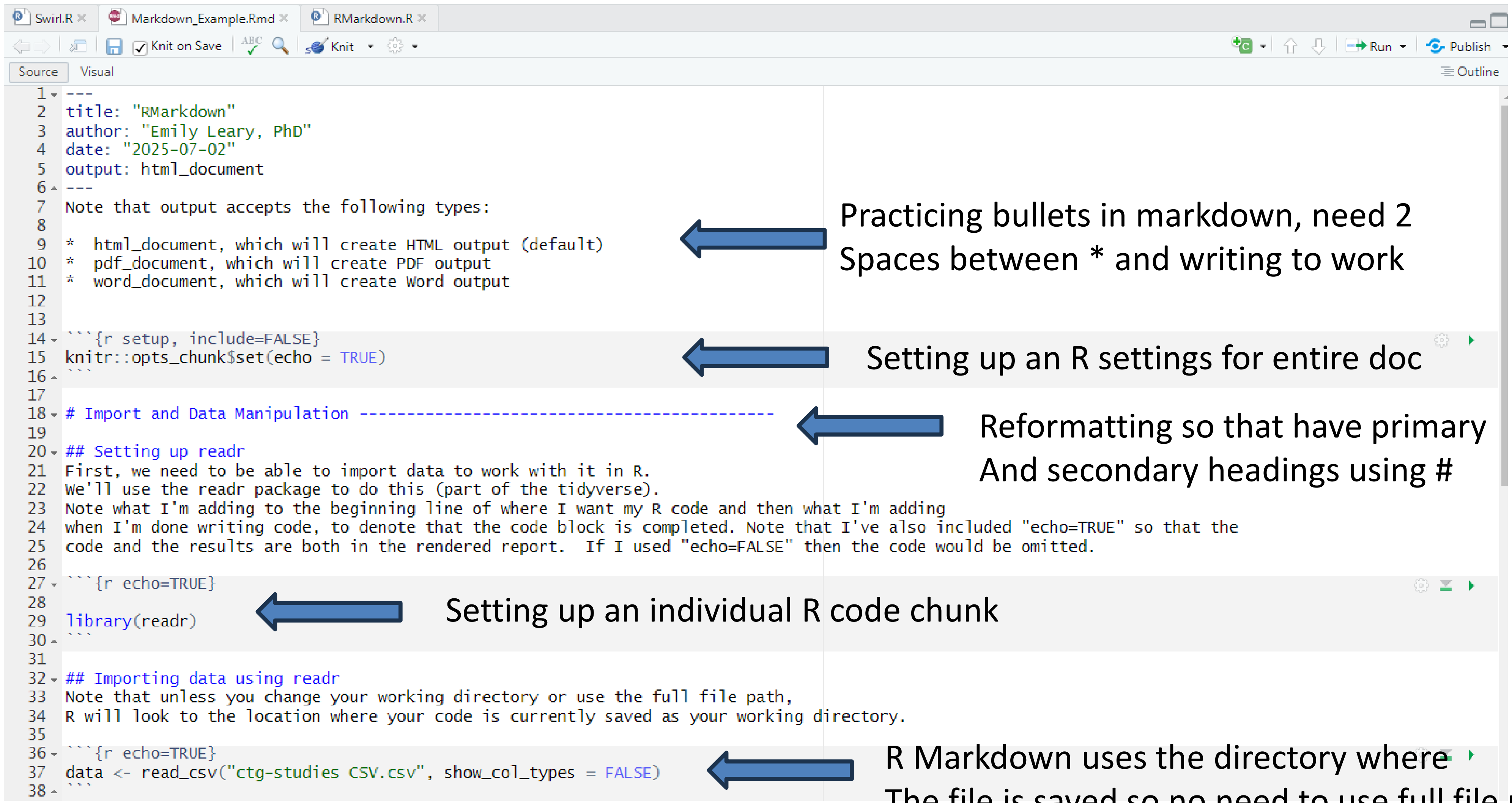
2. **italicized and bold text** - Surround italicized text with asterisks, like this `*without realizing it*`. Surround bold text with two asterisks, like this `**easy to use**`.

3. **lists** - Group lines into bullet points that begin with asterisks. Leave a blank line before the first bullet, like this

```
This is a list

* item 1
* item 2
* item 3
```

4. **hyperlinks** - Surround links with brackets, and then provide the link target in parentheses, like this `[Github](www.github.com)`.

Let's reformat our tidyverse file to R Markdown



~~~
---
title: "RMarkdown"
author: "Emily Leary, PhD"
date: "2025-07-02"
output: html_document
---
Note that output accepts the following types:

*  html_document, which will create HTML output (default)
*  pdf_document, which will create PDF output
*  word_document, which will create Word output


```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

# Import and Data Manipulation --------------------------------------------

## Setting up readr
First, we need to be able to import data to work with it in R.
We'll use the readr package to do this (part of the tidyverse).
Note what I'm adding to the beginning line of where I want my R code and then what I'm adding
when I'm done writing code, to denote that the code block is completed. Note that I've also included "echo=TRUE" so that the
code and the results are both in the rendered report.  If I used "echo=FALSE" then the code would be omitted.

```{r echo=TRUE}

library(readr)
```

## Importing data using readr
Note that unless you change your working directory or use the full file path,
R will look to the location where your code is currently saved as your working directory.

```{r echo=TRUE}
data <- read_csv("ctg-studies CSV.csv", show_col_types = FALSE)
```
~~~

Practicing bullets in markdown, need 2 Spaces between * and writing to work

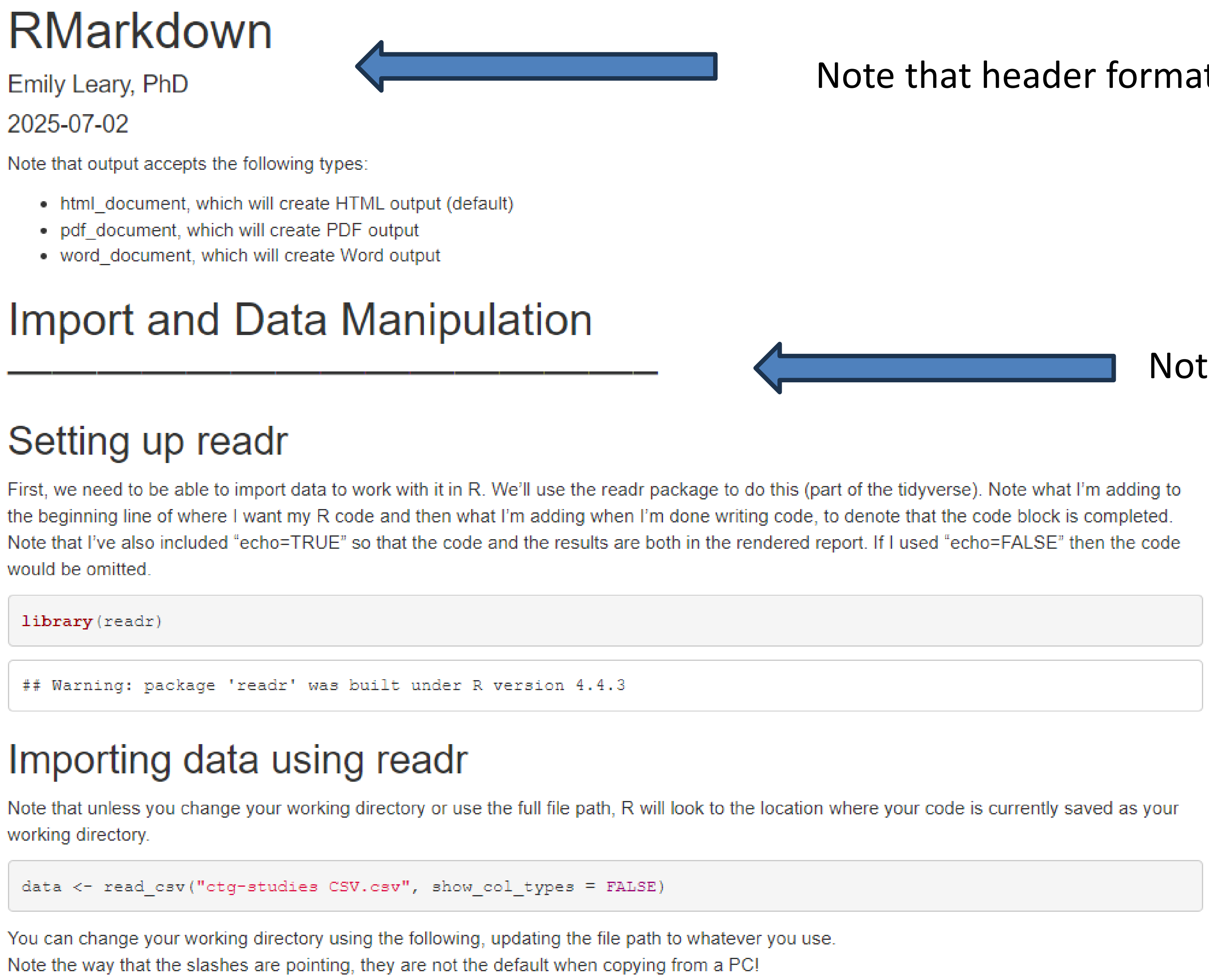
Setting up an R settings for entire doc

Reformatting so that have primary And secondary headings using #

Setting up an individual R code chunk

R Markdown uses the directory where The file is saved so no need to use full file paths

Reformatted & rendered output



RMarkdown

Emily Leary, PhD

2025-07-02

Note that output accepts the following types:

- html_document, which will create HTML output (default)
- pdf_document, which will create PDF output
- word_document, which will create Word output

Import and Data Manipulation

Setting up readr

First, we need to be able to import data to work with it in R. We'll use the readr package to do this (part of the tidyverse). Note what I'm adding to the beginning line of where I want my R code and then what I'm adding when I'm done writing code, to denote that the code block is completed. Note that I've also included "echo=TRUE" so that the code and the results are both in the rendered report. If I used "echo=FALSE" then the code would be omitted.

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.4.3
```

Importing data using readr

Note that unless you change your working directory or use the full file path, R will look to the location where your code is currently saved as your working directory.

```
data <- read_csv("ctg-studies CSV.csv", show_col_types = FALSE)
```

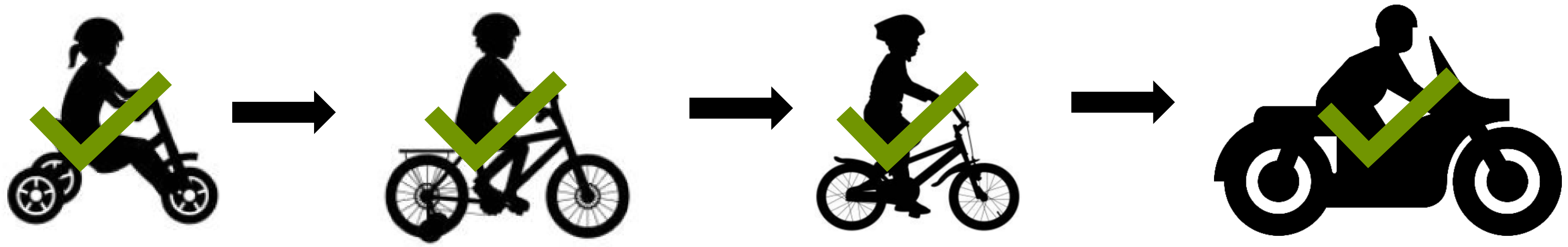
You can change your working directory using the following, updating the file path to whatever you use.
Note the way that the slashes are pointing, they are not the default when copying from a PC!

Note that header formats the information

Note that we have both primary and secondary headings

Note that we have both code and output blocks which makes things very readable!

Review



R Commander

- Data import
- GUI for analysis
- Low bar to start

R Swirl

- Coding foundations
- Self-directed
- More for implementation

tidyverse

- Get to know data
- Flexible for analysis
- Need coding basics

R Markdown

- Project management
- Already know R
- Dynamic document

CHALLENGE!

R Commander

→

R Swirl

→

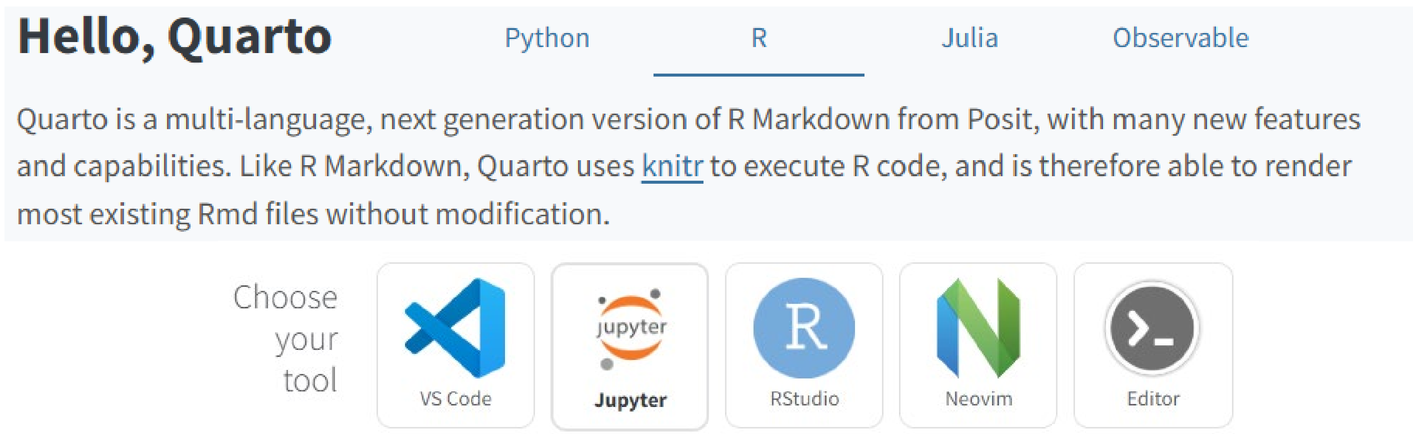
tidyverse

→

R Markdown

↓

Quarto

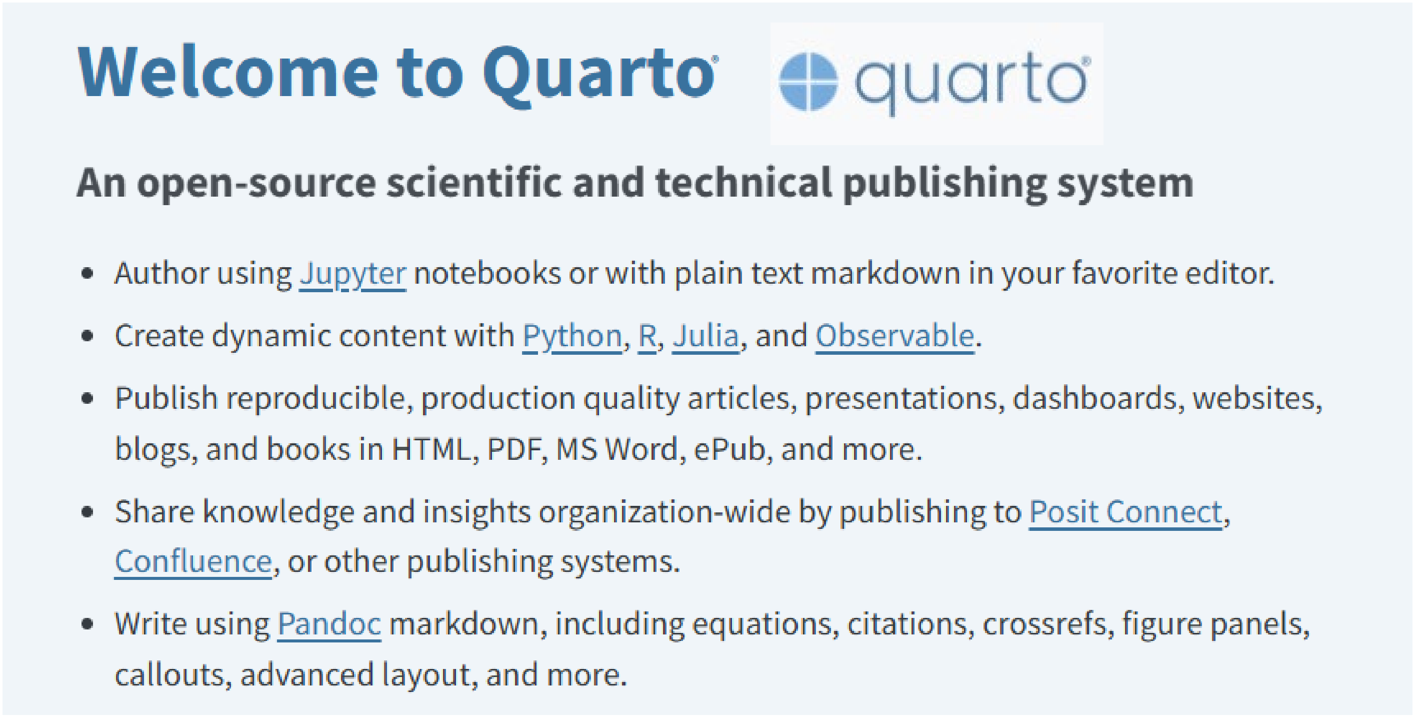

Hello, Quarto Python R Julia Observable

Quarto is a multi-language, next generation version of R Markdown from Posit, with many new features and capabilities. Like R Markdown, Quarto uses knitr to execute R code, and is therefore able to render most existing Rmd files without modification.

Choose your tool

VS Code Jupyter RStudio Neovim Editor

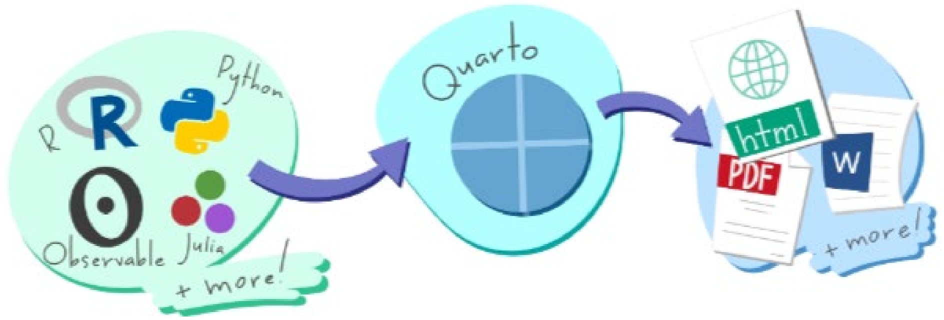
CHALLENGE: Quarto



Welcome to Quarto

An open-source scientific and technical publishing system

- Author using Jupyter notebooks or with plain text markdown in your favorite editor.
- Create dynamic content with Python, R, Julia, and Observable.
- Publish reproducible, production quality articles, presentations, dashboards, websites, blogs, and books in HTML, PDF, MS Word, ePub, and more.
- Share knowledge and insights organization-wide by publishing to Posit Connect, Confluence, or other publishing systems.
- Write using Pandoc markdown, including equations, citations, crossrefs, figure panels, callouts, advanced layout, and more.



Quarto Websites in 20min by Dr. Charlotte Wickham

Quarto for Reproducible Medical Manuscripts by Dr. Mine Çetinkaya-Rundel

Other Resources & FYIs

NIH Library Trainings

- How to Write a Research Paper: Part 1, August 5, 10:00 am–11:00 am
- Copyright and Plagiarism: What Authors Need to Know, August 6, 1:00 pm–2:00 pm
- Introduction to EndNote Desktop, August 7, 1:00 pm–2:00 pm
- How to Write a Research Paper: Part 2, August 12, 10:00 am–11:00 am
- How to Create an Effective ORCiD Profile, August 13, 1:00 pm–2:00 pm
- Tips for Creating Scientific Posters, August 19, 1:00 pm–2:00 pm
- Introduction to Quarto for Scientific Writing, August 25, 1:00 pm–2:30 pm
- Medical Imaging Workflows in MATLAB, August 26, 2:00 pm–3:00 pm

- **Writing and Publishing Services**
- No time to attend a class? Take advantage of NIH Library writing and publishing services and resources.
- Editing Service: Get free, light and medium manuscript editing, suggestions for responding to peer review comments, and more. Submit an Editing Request to get started.
- Plagiarism Checking Service: Identify missed citations or paraphrased wording that is too similar to a published source with an *iThenticate* report. Free and confidential for requesters who are the first, last, or corresponding author of NIH or HHS work-related, unpublished manuscripts.
- Journal Selection Consultation: Consult with our experts on selecting the most appropriate journal for your manuscript based on your audience, type of article and research, and available open access options.
- Author Profile Consultations: Collaborate with our experts on updating your ORCiD, Web of Science, Scopus, or Google Scholar Author Profiles so your research output is curated for maximum visibility on a variety of platforms.

- **Writing and Publishing Resources**
- Writing and Publishing Videos: Watch short writing and publishing tutorials in our new (but growing), NIH Library-created On Demand Video collection.
- EndNote: Download EndNote for free from the NIH Library and use it to manage the references in your manuscript.
- AMA Manual of Style: Ensure your manuscript follows the styles established to support the medical and scientific publishing community.
- Associated Press (AP) Stylebook: Access the style manual used primarily for writing NIH web content, fact sheets, brochures, newsletters, and other promotional materials.
- Chicago Manual of Style: Use the style manual employed by those working in literature, history, and the arts.
- NIH Style Guide: Refer to the style guide developed by NIH for NIH staff.
-

QUESTIONS?

NIH National Institute of Diabetes and Digestive and Kidney Diseases

NIH National Institute of Diabetes and Digestive and Kidney Diseases