

# Workshop: Basic R and More, Part 1 of 3

Jackson Heart Study Training and Education Center  
Summer Research Institute II  
University of Mississippi Medical Center (UMMC)  
Jackson, MS

Philip Turk, MS, PhD<sup>1</sup>

Professor of Data Science  
Chair, Department of Data Science  
John D. Bower School of Population Health

07/14/2023 - 07/15/2023



---

<sup>1</sup> <[pturk@umc.edu](mailto:pturk@umc.edu)>

# Introduction

- This is an intense “get started” workshop on R and more. I assume you know nothing about it.
- R is an open-source (free!) program that is commonly used in statistics and data science. Most of the leading statistical research is first available on R.
  - <<https://www.r-project.org>>
- Specifically, R is a command-line/script-based language, so there is no point-and-click interface. The initial learning curve will be steep, but there are several advantages. Two important ones are:
  - You have to understand what you are doing and why you are doing it.
  - You have a record of what steps you performed in your data analysis.



# Introduction

- History of R:
  - Was created by Ross Ihaka and Robert Gentleman in 1992 as an implementation of the S programming language
  - Language is based on C, Fortran, and R itself
  - Initial Windows version released in 1995
  - Comprehensive R Archive Network (CRAN) started in 1997
  - Considered stable enough for production use in 2000
  - Mac OS X version released in 2001
  - Now over 19,800 available contributed packages
- For any credible poll on the Internet regarding the popularity of statistics and data science software, R will be at or near the top.
- I have been using R ever since it was first released. Today, I use it extensively, followed by Python. When I first started doing serious statistics back in the early 90s, I did everything in SAS. Now, I rarely boot it up. How times have changed . . .

# Introduction

- Finding help:
  - Will show you how to do this in R (e.g., `> ?mean`)
  - Google
  - [<https://stackoverflow.com>](https://stackoverflow.com)
- You should run R through the program *RStudio* ([<https://posit.co>](https://posit.co)). This is a free Integrated Development Environment (IDE) that works on Macs, Windows and Linux. It has many advantages we will demonstrate.
- Many people find RStudio Cheat Sheets to be very useful when they are first starting out:
  - [<https://posit.co/resources/cheatsheets/>](https://posit.co/resources/cheatsheets/)



# Introduction

-   (means we will be switching over to RStudio)
- When you open up RStudio, you will see four windows. My default *starting* configuration starting from the upper-left window moving clockwise is as follows:
  - Source (Code-Edit): This is where you write your R code.
  - Environment: You can see what you have created or view your previous commands. This window can also act as an interface to GitHub (and Azure Repos).
  - Files (Viewing): You can see file directories, view plots, see your packages, and access R help.
  - Console: This is where your code from the Source is evaluated by R or you can also perform quick calculations and operations.
- We will begin our workshop coding in the Console window. Later on, we will see more efficient ways to work.

# Introduction

- When you start learning how to write R code, you do not type command prompts `>` nor line spans `+`. My way of programming in R is to use the symbol `#` to 'mute' code in the testing phase and to use `##` to leave comments in the script. In this workshop, you do not need to type muted code and comments to reproduce the results.
- Online courses from Udemy, Coursera, DataCamp, etc., are good, but they will never replace nor be as effective as lots of practice and the subsequent **normal** discomfort of learning, and being able to ask questions "face-to-face" in real-time.
- So let's get started. This workshop is designed for you to be an active participant, and for me to be an active facilitator. You'll want to follow along and run R on your laptop. Dr. Walker and Dr. Zhu are also here to help you. For this workshop, I will pause regularly to answer any questions.

# R as a Calculator

- In this workshop, the command prompt and code are multi-colored for clarity and decoration. Your command prompt and code will be black.
- Go to the Console window, type each of the following operations at the command prompt `>`, hitting Return after each time  

```
> 2 + 3 ## Add comments; be kind to your future self!
```

```
[1] 5
```

```
> 2 - 3 ## R is insensitive to spacing
```

```
[1] -1
```



```
> 2*3
```

```
[1] 6
```

```
> 2/3
```

```
[1] 0.6666667
```

# Introduction to Functions

- Honorable mentions: trig (e.g., `sin()`), exponential (`exp()`), log (`log()`), absolute value (`abs()`), and many other functions
- When you call a function, there will be some arguments that are mandatory, some that are optional, and the arguments are separated by a comma.
  - Type `?log` in your console  

```
> log(100) ## R will default to natural base e
```

```
[1] 4.60517
```

```
> log(100, base = exp(1)) ## Redundant and unnecessary
```

```
[1] 4.60517
```

```
> log(100, base = 10) ## Override option default to base 10
```

```
[1] 2
```



# Introduction to Functions

- Arguments can be specified via the order in which they are specified or by naming the arguments. Be careful when you are first starting out learning R!

```
> log(x = 100, base = 10) ## Long version
```

```
[1] 2
```

```
> log(base = 10, x = 100) ## Weird but it works
```

```
[1] 2
```



```
> log(100, 10) ## R decides x is first, base second
```

```
[1] 2
```

```
> log(10, 100) ## Bad news?
```

```
[1] 0.5
```

# Assignment

- Simply typing something into R does not mean R remembers it.
- We need to save, or *assign*, a value or values to a variable, or *object*, to be able to use it later. R does this by using an arrow `<-` or an equal sign `=` (I prefer the arrow)  

```
> a <- 2*pi  
> a ## Hit return to see it
```

```
[1] 6.283185
```

- If you want R to save any objects (e.g., `a`) you created once your session is over, then answer “Save” when you quit the R session.

# Assignment

Here are some rules for naming variables in R:

- Variable names may not include spaces, and are **case sensitive**. *This last one is a very common mistake.*
- The variable name can be a combination of letters, numbers, period (.), and underscore (\_) (e.g., can't have %).
- It must start with a letter or a period (e.g., can't start with a number, can't start with underscore).
- If it starts with a period, then it can't be followed by a number.
- Reserved words in R can't be used (e.g., TRUE).

# Assignment



```
> b <- 5  ## Look at the Environment tab in RStudio  
> b
```

```
[1] 5
```

```
> c <- a*b ## Create a third variable from first two  
> c
```

```
[1] 31.41593
```

# Assignment

- As you work with R you'll often want to re-execute a command which you previously entered. The RStudio console supports the ability to recall previous commands using the arrow keys:
  -  
  - Up will recall previous command(s)
  - Down is the reverse of Up
- You could also go to the History tab in what I called the Environment window and send selected commands to the R console.

# Vectors

- R operates on vectors where a vector is a collection of elements, usually numbers. To do this, we use the `c()` function which 'combines' elements into a vector.

```
> c(1, 2, 3, 4)
```

```
[1] 1 2 3 4
```

- There are other ways to define vectors.

```
> rep(2, 5) ## Repeat 2 five times
```

```
[1] 2 2 2 2 2
```

```
> rep(c("A", "B"), 3) ## Repeat A B three times
```

```
[1] "A" "B" "A" "B" "A" "B"
```

- Note that single quotes `'` would work as well when defining character values (e.g., `'A'`).

# Vectors

- We can also define a *sequence* of numbers using the `seq(from, to, by, length.out)` function which expects the user to supply any 3 out of 4 possible arguments. 'From' is the starting point of the sequence, 'to' is the end point, 'by' is the difference between any two successive elements, and 'length.out', or just 'length', is the total number of elements in the vector.

```
> seq(1, 4) ## 'by' default is 1
```

```
[1] 1 2 3 4
```

```
> ## 1:4 by itself will also work
```

```
> seq(1, 4, by = 0.5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
```

```
> seq(6, 12, length.out = 5) ## Doesn't have to be integers
```

```
[1] 6.0 7.5 9.0 10.5 12.0
```

- If we have two vectors and we wish to combine them, we can again use the `c()` function.

```
> vec1 <- rep(2, 5)
> vec2 <- seq(6, 10, length.out = 3)
> vec3 <- c(vec1, vec2)
> vec3
```

```
[1]  2  2  2  2  2  6  8 10
```



# Accessing Vector Elements

- To pull out a certain elements or several elements from a vector, we use `[]` notation.

```
> vec3[7] ## Seventh element in vec3
```

```
[1] 8
```

```
> vec3[c(5, 7)] ## Elements 5 and 7 in vec3
```

```
[1] 2 8
```



```
> vec3[5:7] ## Elements 5 through 7 in vec3
```

```
[1] 2 6 8
```

```
> vec3[-7] ## All of vec3 except element 7
```

```
[1] 2 2 2 2 2 6 10
```

# Accessing Vector Elements

- Now the [1] notation you've been seeing all along in the output should make sense. Because vectors are often very long and might span multiple lines, R will help you by telling you the index number of the left most value. If we have a very long vector, the second line of values will start with the index of the first value on the second line.
- For example, submit the following and see what happens. Your answers will vary depending on the size of your Console window  

```
> letters
```

# Scalar Functions Applied to Vectors

- It is very common to want to perform some operation on all the elements of a vector simultaneously. Scalar functions that are defined on single values will almost always apply the function to each element of the vector (“element-wise”) if given a vector.

```
> x <- -5:5
```

```
> x
```

```
[1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```
> abs(x) ## Take absolute value of each element
```

```
[1] 5 4 3 2 1 0 1 2 3 4 5
```

```
> x^2 ## Square each element
```

```
[1] 25 16  9  4  1  0  1  4  9 16 25
```

# Vector Algebra

- All algebra done with vectors will be done element-wise by default.
  - For matrix and vector multiplication as usually defined by mathematicians, do **not** use this approach.

```
> x <- 1:4
```

```
> x
```

```
[1] 1 2 3 4
```

```
> y <- 5:8
```

```
> y
```

```
[1] 5 6 7 8
```

```
> x + y
```

```
[1] 6 8 10 12
```

```
> x*y
```

```
[1] 5 12 21 32
```

# Commonly Used Scalar Functions

- R has many common scalar functions already in the base version of R (base R). For example, the sample variance  $S^2$  is easily computed in R's `var()` function. Nevertheless, we can easily write our own functions if we have to. To show how scalars, vectors, and functions of them work together, we will show how to use R to compute the sample variance  $S^2$  by hand. Recall:

$$S^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n - 1}$$

- Suppose we have the sample  $\{2, 4, 6, 8, 10\}$ . What is the sample variance?

# Commonly Used Scalar Functions

```
> x <- c(2, 4, 6, 8, 10)
> xbar <- mean(x) ## Compute the mean
> xbar
```

```
[1] 6
```

```
> x - xbar ## Compute the 'errors'
```

```
[1] -4 -2  0  2  4
```

```
> (x - xbar)^2 ## Square the errors
```

```
[1] 16  4  0  4 16
```

```
> sum((x - xbar)^2) ## Sum of the squared errors
```

```
[1] 40
```

# Commonly Used Scalar Functions

```
> n <- length(x) ## How many data do we have?  
> n
```

```
[1] 5
```

```
> sum((x - xbar)^2)/(n - 1)
```

```
[1] 10
```

```
> var(x) ## Same as R's built-in function
```

```
[1] 10
```

# Matrices

- We can store numerical data in *matrices*. These will have two dimensions, rows and columns.

```
> W <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
> W
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

- The order that R fills the matrix is to fill in the first column, then the second, and then the third (aka “column-major ordering”). If we wanted to fill the matrix in order of the rows first, then we’d use the optional `byrow = TRUE` argument (default is set to `FALSE`).
- Technically, either `nrow` or `ncol` could be left off in this example.



# Matrices

- Accessing a part of a matrix is done in a similar manner as with vectors, using the `[]` notation, but this time we must specify which row and which column using `[row, col]`.

```
> W[1, 2] ## Row 1, column 2
```

```
[1] 3
```

```
> W[1, 1:2] ## Row 1, columns 1 and 2; c(1, 2) also works
```

```
[1] 1 3
```

```
> W[1, ] ## All of row 1
```

```
[1] 1 3 5
```

```
> W[, 2] ## All of column 2
```

```
[1] 3 4
```

# Matrices

- We can also assign names for the columns and rows.

```
> colnames(W) <- c("T1", "T2", "T3") ## Set column names
> rownames(W) <- c("C0", "NC") ## Set row names
> W
```

	T1	T2	T3
C0	1	3	5
NC	2	4	6

- Note: This is actually an odd and outdated way of setting the attributes of the object W because it looks like we are evaluating a function and assigning some value to the function output.