0 Getting Started with R

AUTHOR
Jeremy Mikecz

PUBLISHED
February 24, 2025

Getting Started with R

For more help for beginners to R, please visit:

- Alex Douglas, Deon Roos, Francesca Mancina, Ana Couto, and David Lusseau, *An Introduction to R*(2024).
- Garret Grolemund, *Hands-On Programming with R*(2014).
- For a list of additional R resources see the *Big Book of R*

1. Download and Install

Download and Install R and R Studio from the Posit webpage (Posit is the company that manages open-source R).

1.1 Download R

```
a. Select `Download and Install R`
b. Select `Download R for {your operating system`
c. Follow the instructions for downloading and installing R
```

1.2 Download and Install R Studio

```
a. Select `Download Rstudio Desktop for {your operating system}`
b. Find the installation software you just downloaded (probably in your "Downloads" folder)
c. Follow instructions for installation
```

1.3 Folder Setup & Organization

```
a. One project, one folder
b. Folder and file naming
    i. Keep simple
    ii. Replace whitespace with underscore ("_").
    iii. Add dates for different file versions using iso format: YYYY-MM-DD

c. Subfolders (aka. Subdirectories) for separate components of a project
    i. I.e. research study of spiny anteaters:


                spiny_anteaters/
                    data/
```

```
                    field_notes/
                    Images/
                    notebooks/
                    results/
                    README.txt
```

ii. Or for a class on data visualization

```
Dataviz_2026/
    week1/
        notebooks/
        notes/
        presentations/
    week2/
    week3/
    week4/
    week5/
    project/
        code/
        data/
        notes/
        results/
```

4. Downloading or Cloning Data from a Repository

<!–For those of you with experience working with git and GitHub repositories, you may:

1. download the code for this class from: https://github.com/Dartmouth-Libraries/R-Data-Analysis-and-Visualization.
2. If you have your Github SSH key set up, select `Code` –> `SSH` tab –> `Copy url to clipboard`.
3. Navigate to the folder on your computer where you would like to place your local version of this class

–>

REVIEW LINK ABOVE & BELOW - CHANGE FOR SPECIFIC SESSIONS!

4b. Everyone else: downloading data

1. Visit our repository to download code for this course: https://github.com/Dartmouth-Libraries/R-Data-Analysis-and-Visualization.
2. Select the green `Code` button –> `Download Zip`.
3. Unzip the folder and place it in a permanent location on your local drive (or cloud drive if you prefer).

5. Coding in R (from the terminal)

Basic operations: numbers, strings, and lists

1. Within R Studio, take a look at the **Console**. If you can't find the Console, ???.

2. Try typing in some basic mathematical operations in the console. Note:

```
+ addition
- subtraction
* multiplication
/ division
^ exponents (i.e. 2^2 is 2 squared)
%% modulus (remainder from division)
%/% quotient for integer division (excluding remainder)
```

   You can use parentheses `()` to place operations into groups, such as:

```
(3 + 8) / 4  # contrast to:
3 + 8 / 4
```

6. Coding in R (from the console)

1. We can store values (numbers, strings, lists, and far more complicated data types) in variables.
2. In the console the right assign your favorite number to a variable like this:

```
fav_num <- 11
```

3. Notice, R uses two characters "<" with "-" to assign a value to a variable. Many other languages, like Python, simply use `=`. We can also use the equal sign (`=`) sign to assign values to variables in R, but the "<-" is convention.

4. Note, spaces help for example:

```
x<-2  #is this assigning 2 to x, or is it checking if x is less than -2?
x <- 2 #better to make assignment clear with spaces
x < -2 #and comparisons like this
```

   For example, try typing the following in the console:

```
x <- 2
x < -2
```

   What results do you get?

5. Now trying doing an assignment in reverse order:

```
6 -> z
```

Note: view the Environment Pane (usually top right of R Studio). There you will see a table of values saved as variables. This is very useful to remind you of what you have saved into memory.

6a. Vectors & Lists

If you want to create a "list" of similar items in R, usually you will store such values in a vector denoted by `c(item1, item2, item3)`. For example, you can create a vector of some of your favorite numbers in the console like this (replace with your favorite numbers:

```
fav_nums <- c(3, 7, 11, 38, 83)   #replace with your favorite numbers
```

You can then perform math operations on a list, such as:

```
fav_nums * 2
fav_nums / 3
fav_nums ^ 2
```

You can also create a vector with a series of text strings:

```
fict_detectives <- c("Sherlock Holmes", "Jessica Fletcher", "Marti MacAlister", "Hercule Poirot",
"Nancy Drew",  "Columbo", "Lt. Leaphorn")

fict_detectives * 2
```

By convention, in R, most one-dimensional arrays of data (like the above) are stored in vectors. However, you can also create a **list** using `list()`. These lists work similarly to lists in Python in that you may store more heterogenous data in a list than in a vector. I.e. try the following:

```
nested_vec <- c(1, 2, c(3, 4, 5))
nested_list <- list(1, 2, c(3, 4, 5))
```

6b. working with functions

Tasks to be repeated are commonly saved as **functions**.

In R, we regularly work with three main types of functions:

1. core R functions (like `print()` or `seq()` which are available to any R setup.
2. additional functions downloaded with specific packages. Only the most basic functions come with base R. More advanced functionality is unlocked by importing external libraries that include their own customs.
3. writing our own custom functions: want to create a function that reads in a person's age and calculates how many days, hours, and minutes until their next birthday? You can create a function that does just that.

We will focus on core R functions here. Later lessons will include instructions how to import and use external functions and how to create your own.

```
print("Hello world!")
#vs.
```

```
greeting <- "Hello"
name <- "Jeremy"
print(c(greeting, name))
```

To learn more about a function use:

```
help(print)
```

Most (but not all) functions allow you to pass in multiple arguments. For example, the help documentation for **print()** indicates you can pass in a boolean argument to the parameter `quote` . So, let's try:

```
print(c(greeting, name), quote=FALSE)
```

```
seq()
```

```
seq(2)
```

```
seq(from = 2, to = 10)
```

```
seq(2, 10)
```

```
seq(2, 10, 2)
```

```
seq(from = 2, to = 10, by = 1.5)
```

6c. Create a basic plot

We can also use the `plot()` function to create a basic dot plot.

```
x <- c(3, -4, 6, 1)
y <- c(0, 2, -4, 5)
plot(x, y)

#then try:
plot(x, y, "l")
```

7. Coding in R (from an R script `.r`)

Now that we've had some practice with the console, let's create our first R script.

The console allows some degree of interactivity: you can enter some code and receive an immediate response. However, for more complicated tasks as well as any task you hope to re-run in the future (one of the main purposes for programming) you will want to save your code for re-use. The simplest way to save your code is save it in a simple **R script** (ending in `.R`), the subject of this section. In the next section, we will learn how to mix R code with human instructions and notes using a **notebook**.

7.1. Testing a script

Writing a short program

Running the Code from the Script

1. **Open RStudio** and create a new R script:
   - Go to the top-left corner and click on `File` > `New File` > `R Script`.
2. **Copy and paste the provided R script** into the new R script file:

```
# A simple R script to greet the user

# Ask for the user's name
user_name <- readline(prompt = "What is your name? ")

hometown <- readline(prompt = "What is the name of your hometown and state/country?")

# Create a customized greeting message
greeting <- paste("Hello,", user_name, "from", hometown, "! Welcome to the world of R!")

# Print the greeting message
print(greeting)
```

3. Make whatever changes you would like to customize the above script.
4. **Save the script** :
   - Click on `File` > `Save`, and choose a name for your script (e.g., `greeting_script.R`).
5. **Run the script** :
   - To execute the entire script, click the `Source` button (usually at the top-right of the script editor). This will run the script in the console, and the prompt will appear asking for the user's name.
   - To run portions of the script you can highlight those lines and select `Run`.
6. **Input your name and hometown** when prompted in the console and hit `Enter`. The personalized greeting will be displayed.

To run the above script from the console (assuming it is saved), you will use the `source()` function like this:

```
source([absolute or relative path to script])
```

Using relative paths is usually preferred. A relative path points to a file's relative location compared to the directory where the computer is in now. For example if you are in the folder:

`C:/Users/abby/documents/R_projects/notebooks`

and the code you want to call is in:

`C:/Users/abby/documents/R_projects/code/script.R`

its relative path is: `../code/script.R` .

To run the script from the console, we will need to use the `source()` function as mentioned above. But, to use relative paths with this function, we need to import the **here** library. Running the following code in the console will install this library and run your script:

```
install.packages("here")    #installs here, only need to do this once
source(here::here("top-level-folder", "folder", "script-name.R"))

# this works the same as above:
# source(here::here("top-level-folder/folder/script-name.R"))
```

7.2. Running an entire script from the terminal

We can also run scripts from the terminal.

1. Within the terminal, navigate to the location of your custom script.
2. Try entering the following:

   ```
   Rscript script-name.R
   ```

   Did this work?

   You can also run R from within the terminal by:

   ```
   R
   # R will open in the terminal
   source("script-name.R")

   #to exit R:
   q
   ```

7.3 Why use the terminal

Beginning programmers often find using their computer's terminal or command line intimidating. Fortunately, you can get started with R and R Studio without using the command line.

That said, as you advance your skills you may find various reasons to use it. Some examples include:

1. Using the terminal found within R Studio (or using your computer's native terminal) to perform version control using git For example, the materials in this workshop's directory were 'pushed' to our Github repositoryusing the terminal.
2. It allows you to call completed R scripts quickly, without the need to open the script itself.
3. Using the terminal allows you to interact with your computer's system and administration in ways not possible through the graphical user interface (GUI).
4. You can quickly and more efficiently navigate through your folders and files.
5. Command History - the terminal keeps track of your past commands, making it easier to repeat them.

8. Coding in R (from a notebook)

Coding scripts in R are saved in `.R` files (as we saw above). These R scripts are the most common file type for software apps and packages.

However, if you are teaching, learning, sharing your code with collaborators (with varying knowledge of the code), or simply performing data analysis and visualization, you may prefer using a **coding notebook**. Coding notebooks include a mixture of:

- metadata for the notebook itself (this includes instructions telling the computer how to "publish" your notebook as a website, slideshow, or report)
- code cells with machine-readable code
- text cells with instructions intended for humans (i.e. instructions, explanations, tutorials, links to resources, etc.)
    - these cells are often called *markdown* cells because they are written following markdown format that allows the users to encode styling instructions using relatively basic set of characters (see the difference between unrendered and the rendered markdown below):

```
# Header 1
## Header 2
### Header 3

*italics*

**bold**

~~strikethrough~~

numbered list:
1. first item
2. second item
2. third item

unordered list:
+ item
    + sub-item
+ item
+ item

links & images:

[CNN](www.cnn.com)

![Dartmouth Library](https://live.staticflickr.com/65535/67023922_0dfbf259a9_b.jpg)
```

Header 1

Header 2

Header 3

italics

bold

~~strikethrough~~

numbered list: 1. first item 2. second item 2. third item

unordered list: + item + sub-item + item + item

links & images:

CNN

Dartmouth Library

For more on markdown syntax see Markdown Basics.

8.1. R Markdown notebooks (`.Rmd`) vs. Quarto notebooks (`.qmd`)

For a long time, the primary type of notebook used in R Studio was the **R Notebook** which was saved using the R Markdown `.Rmd` extension. However, recently Posit has developed **Quarto notebooks** (`.qmd`). Some advantages of Quarto:

1. Quarto automatically renders markdown live as you type it whereas R notebooks only render the markdown when you publish them. Thus what looks like this

   ```
   [link](https://r-graph-gallery.com/) to R Graph Gallery
   ```

   looks like this in Quarto:

   link to R Graph Gallery

2. its language-agnostic. Sure you can run R code with them. But, you can also run Python, Julia, Observable, etc.

3. Quarto notebooks offer multiple publication options in one tool. You can convert and publish your Quarto notebooks into slideshow presentations, webpages, or online books.

4. unlike R Notebooks, Quarto is a command line interface, meaning you can more easily work with Quarto outside R Studio.

5. for more on the differences see jumpingrivers, Posit's FAQs, the Quarto documentation, and the Quarto chapter in *R for Data Science*.

8.2. Working with Quarto notebooks (`.qmd`)

We will demonstrate how to work with Quarto notebooks in subsequent lessons.