Introduction to R Programming

Red Rock Data Science

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School
w.evan.johnson@rutgers.edu

2024-05-24

R Tutorials (GitHub and YouTube)

This is an abbreviated version of Dr. Johnson's online R tutorial on GitHub: https://github.com/wevanjohnson/2024_04_R_tutorial

R Tutorials (GitHub and YouTube)

Lecture	Topics
Lecture 1	Installing R, RStudio, and R packages
Lecture 2	Introduction to R/RStudio
Lecture 3	R basics, Part 1
Lecture 4	R basics, Part 2
Lecture 5	R basics, Part 3
Lecture 6	Programming Basics
Lecture 7	R Markdown
Lecture 8	Input/output data, Data structures
Lecture 9	The tidyverse
Lecture 10	Visualization with ggplot2, Part 1
Lecture 11	Visualization with ggplot2, Part 2
Lecture 12	Visualization with ggplot2, Part 3
Lecture 13	Creating R Packages
Lecture 14	Shiny Programming, Part 1
Lecture 15	Shiny Programming, Part 2

Section 1

Installation Details

Important installations

You will need to install the following:

Mac Users

- R and R Studio
- Know how to access a terminal (Rstudio or Terminal)
- git (type `git --version` in the terminal)

Windows Users:

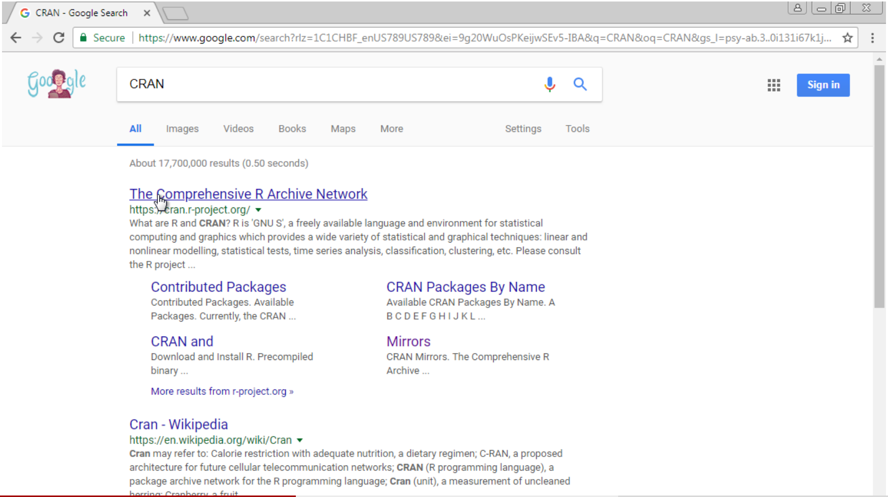
- R and R Studio
- A terminal app (Git Bash, MobaXterm, Putty)
- Git for Windows

R and Rstudio

R is a language for statistical computing and graphics. **RStudio** is an interactive desktop environment (IDE), but it is not R, nor does it include R when you download and install it. Therefore, to use RStudio, we first need to install R.

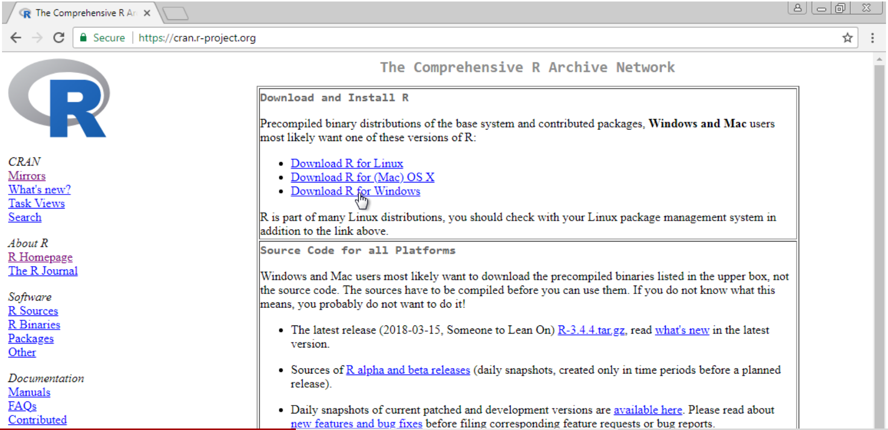
Installing R (Windows and Mac)

You can download R from the Comprehensive R Archive Network (CRAN)[1]. Search for CRAN on your browser:



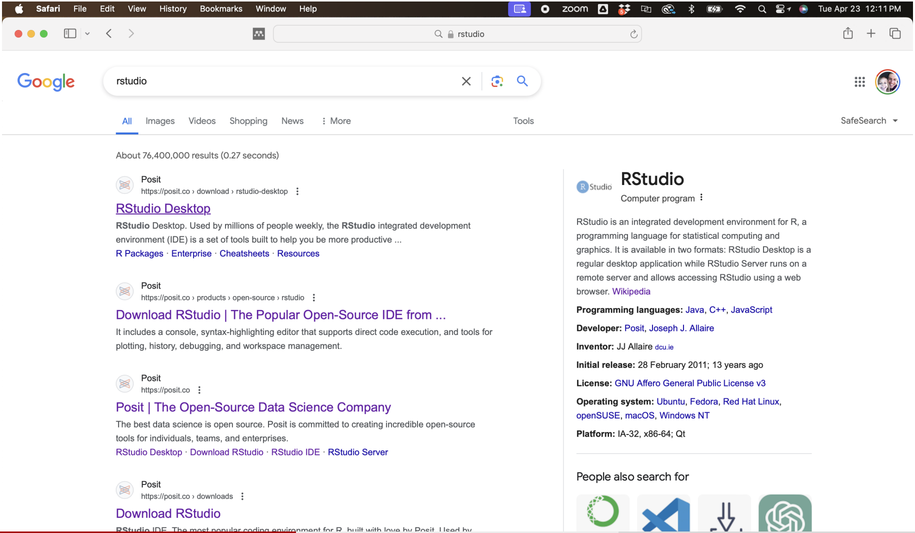
Installing R (Windows and Mac)

Once on the CRAN page, select the version for your operating system: Linux, Mac OS X, or Windows. Here we show screenshots for Windows, but the process is similar for the other platforms. When they differ, we will also show screenshots for Mac OS X.



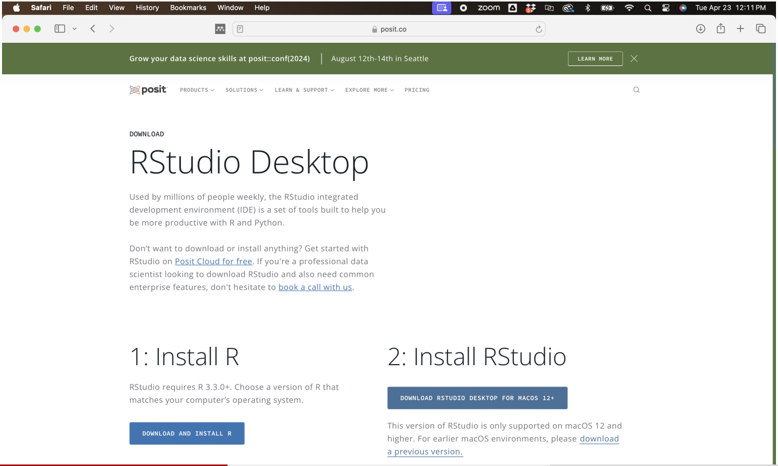
Installing RStudio (Windows and Mac)

To install RStudio, start by searching for “RStudio” on your browser:



Installing RStudio (Mac)

You should find the Posit/RStudio website as shown above. Once there, click on "Download RStudio Desktop for Mac OS 12+" below the *2: Install RStudio* header.



More on R and Rstudio

See more detailed instructions at in Lecture 1 at:
https://github.com/wevanjohnson/2024_04_R_tutorial

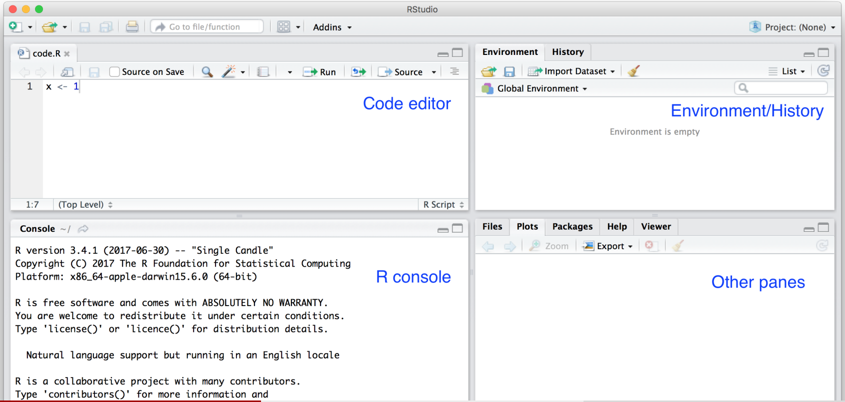
Why R?

R is not a programming language for software development like C or Java. It was created by statisticians as an environment for data analysis. A history of R is summarized here: A Brief History of S.

The **interactivity** of R (more later), is an indispensable feature in data science because, as you will learn, the ability to quickly explore data is a necessity for success in this field.

RStudio

One of the great advantages of R over point-and-click analysis software is that you can save your work as scripts. You can edit and save these scripts using a text editor. We will use the interactive *Integrated Development Environment* (IDE) RStudio.



Objects

Suppose we are asked to solve the quadratic equation $x^2 + x - 1 = 0$, we can define:

```
a <- 1
b <- 1
c <- -1
```

which stores the values for later use. We use <- to assign values to the variables. We can also assign values using = instead of <-, but we recommend against using = to avoid confusion.

Objects

To see the value stored in a variable, we simply ask R to evaluate a and it shows the stored value:

```
a
```

```
## [1] 1
```

A more explicit way to ask R to show us the value stored in a is using print like this:

```
print(a)
```

```
## [1] 1
```

The Workspace

Now since these values are saved in variables, to obtain a solution to our equation, we use the quadratic formula:

```
(-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
```

```
## [1] 0.618034
```

```
(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
```

```
## [1] -1.618034
```

Scripts

To solve another equation such as $3x^2 + 2x - 1$, we can copy and paste the code above and then redefine the variables and recompute the solution:

```
a <- 3
b <- 2
c <- -5
(-b + sqrt(b^2 - 4*a*c)) / (2*a)
(-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

By creating and saving a script with the code above, we would not need to retype everything each time and, instead, simply change the variable names. Try writing the script above into an editor and notice how easy it is to change the variables and receive an answer.

Functions

Once you define variables, the data analysis process can usually be described as a series of **functions** applied to the data. R includes several predefined functions and most of the analysis pipelines we construct make extensive use of these.

Functions

Most functions require one or more **arguments**. Below is an example of how we assign an object to the argument of the function `log`. Remember that we earlier defined `a` to be `1`:

```
log(8)
```

```
## [1] 2.079442
```

```
log(a)
```

```
## [1] 0
```

Functions

You can change the default values by simply assigning another object:

```
log(8, base = 2)
```

```
## [1] 3
```

Note that we have not been specifying the argument `x` as such:

```
log(x = 8, base = 2)
```

```
## [1] 3
```

Installing R packages

The functionality provided by a fresh install of R is only a small fraction of what is possible. In fact, we refer to what you get after your first install as **base R**. The extra functionality comes from add-ons available from developers.

There are currently hundreds of these available from CRAN and many others shared via other repositories such as GitHub. However, because not everybody needs all available functionality, R instead makes different components available via **packages**.

Installing R packages

R makes it very easy to install packages from within R. For example, to install the **dslabs** package, which we use to share datasets and code related to this book, you would type:

```
install.packages("dslabs")
```

We can install more than one package at once by feeding a character vector to this function:

```
install.packages(c("tidyverse", "dslabs"))
```

Vectors

In R, the most basic objects available to store data are **vectors**. As we have seen, complex datasets can usually be broken down into components that are vectors. For example, in a data frame, each column is a vector. Here we learn more about this important class.

Creating Vectors

We can create vectors using the function c, which stands for **concatenate**. We use c to concatenate entries in the following way:

```
codes <- c(380, 124, 818)
codes
```

```
## [1] 380 124 818
```

Creating Vectors

We can also create character vectors. We use the quotes to denote that the entries are characters rather than variable names.

```r
country <- c("italy", "canada", "egypt")
```

Names

Sometimes it is useful to name the entries of a vector. For example, when defining a vector of country codes, we can use the names to connect the two:

```
codes <- c("italy" = 380, "canada" = 124, "egypt" = 818)
codes
```

```
##  italy canada  egypt
##    380    124    818
```

Subsetting elements

The elements of vectors can be obtained using the following:

```
codes[1]
```

```
## italy
##   380
```

```
codes["italy"]
```

```
## italy
##   380
```

```
codes[2:3]
```

```
## canada  egypt
##    124    818
```

Data Frames

The most common way of storing a dataset in R is in a **data frame**, which is a combination of several (columns of) vectors.

Data frames are particularly useful for datasets because we can combine different data types into one object.

Data Frames

A large proportion of data analysis challenges start with data stored in a data frame. For example, we stored the data for our motivating example in a data frame. You can access this dataset by loading the **dslabs** library and loading the `murders` dataset using the `data` function:

```
library(dslabs)
data(murders)
```

Data Frames

We can show the first six lines using the function `head`:

```r
head(murders)
```

```
##        state abb region population total
## 1    Alabama  AL  South    4779736   135
## 2     Alaska  AK   West     710231    19
## 3    Arizona  AZ   West    6392017   232
## 4   Arkansas  AR  South    2915918    93
## 5 California  CA   West   37253956  1257
## 6   Colorado  CO   West    5029196    65
```

The Accessor: $

For our analysis, we will need to access the different variables represented by columns included in this data frame. To do this, we use the accessor operator $ in the following way:

```
murders$population
```

```
##  [1]  4779736   710231  6392017  2915918 37253956  5029196  3574097    89
##  [9]   601723 19687653  9920000  1360301  1567582 12830632  6483802   304
## [17]  2853118  4339367  4533372  1328361  5773552  6547629  9883640   530
## [25]  2967297  5988927   989415  1826341  2700551  1316470  8791894   205
## [33] 19378102  9535483   672591 11536504  3751351  3831074 12702379   105
## [41]  4625364   814180  6346105 25145561  2763885   625741  8001024   672
## [49]  1852994  5686986   563626
```

Subsetting Columns and Rows

In addtion the columns and rows of a data frame can be subsetted using the following syntax:

```
murders[1,1]
```

```
## [1] "Alabama"
```

```
murders[1,]
```

```
##     state abb region population total
## 1 Alabama  AL  South    4779736   135
```

```
murders[,4]
```

```
##  [1]  4779736   710231  6392017  2915918 37253956  5029196  3574097   89
##  [9]   601723 19687653  9920000  1360301  1567582 12830632  6483802  304
## [17]  2853118  4339367  4533372  1328361  5773552  6547629  9883640  530
## [25]  2967297  5988927   989415  1826341  2700551  1316470  8791894  205
## [33] 19378102  9535483   672591 11536504  3751351  3831074 12702379  105
## [41]  4625364   814180  6346105 25145561  2763885   625741  8001024  672
## [49]  1852994  5686986   563626
```

Programming Basics

By coding in R, we can efficiently perform exploratory data analysis, build data analysis pipelines, and prepare data visualization to communicate results. However, R is not just a data analysis environment but a programming language.

You should also understdn the following three key programming concepts: **conditional expressions**, **for-loops**, and **functions**. These are not just key building blocks for advanced programming, but are sometimes useful during data analysis.

Session info

```r
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sonoma 14.2.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] dslabs_0.8.0
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.4.0    fastmap_1.1.1     cli_3.6.2         tools_4.4.0
##  [5] htmltools_0.5.8.1 rstudioapi_0.16.0 yaml_2.3.8        rmarkdown_2.26
##  [9] knitr_1.46        xfun_0.43         digest_0.6.35     rlang_1.1.3
## [13] evaluate_0.23
```