Lunch Hour Learning Guide, Session 1, Spring 2025

Basics of R and RStudio

Sean Morey Smith

2025-01-15

What You Will Learn

- What R is and why you should use it
- Finding your way around RStudio
- Creating an R project
- Interacting with R
- Installing and loading packages
- Getting help in R
- Incorporating best practices into your scripts

Before Starting

Ensure R and RStudio are installed and up-to-date. Instructions can be found at https://library.rice.edu/sites/default/files/materials/R_installation_instructions_2024.pdf

Intro to R - Pros and Cons

R is an open-source programming language that was invented explicitly to deal with data tasks. Like any data analysis tools, R has pros and cons. For example, some of the benefits of R are that it is:

- Open source and free to use
- Extremely flexible and great for customized analysis and visualization
- Supported by a robust community of users
- Enhanced by thousands of packages (collections of code) that extend its capabilities.

The main challenge of R is the learning curve associated with learning a new programming language. But Research Data Services can help with this!

Tour of RStudio

RStudio is a free, open-source Integrated Development Environment. It provides lots of neat features:

- built-in editor
- platform neutral
- integration with version control and project management
- GUI features and shortcuts that make your life easier without reducing reproducibility

Overview of panels:

- Left: interactive R console/Terminal
- Upper right: Environment/History/Connections
- Lower right: "Miscellaneous" (Files/Plots/Packages/Help/Viewer)

You can also open a fourth panel for a script editor; we will do that shortly.

R Projects

To create a project:

1. Click File then New Project.
2. Click New Directory.
3. Click New Project.
4. Type the name of a directory where you'll store the project (e.g., my_project).
5. Click Create a git repository if this is an option.
6. Click the Create Project button.

This creates a project (.Rproj file) within the directory you created or selected.
All files that you store within this directory will be contained within the project.
This also gives the advantage of using relative file paths.

To open a project:

1. Navigate to the directory.
2. Double-click on the .Rproj file.

Interacting with R

Look at Console first:

- Good for trying things out before coding to a script file. It keeps a history but isn't saved.
- Greater than sign is the prompt to type something in.

Calculator

R can work like a calculator. Enter

```
1 + 100
```

```
## [1] 101
```

Note that the answer is preceded by a number in brackets [1]. This is the index of the first element of the line being printed in the console.
We will return to the concept of the index later. For now, just know that the output (101) is a single value, indexed as [1].

What happens when we type `1 +`? We get a + in the console, indicating that R expects additional code.

Options:

- complete the command
- cancel the commend with *Esc* within RStudio

Note that order of operations (PEMDAS - Parentheses, Exponents, Multiplication/Division, Addition/Subtraction) applies. Practice:

```
3 + 5 *2
```

```
## [1] 13
```

```
(3 + 5) * 2
```

```
## [1] 16
```

```
2/10000
```

```
## [1] 2e-04
```

Very small or large numbers are printed in scientific notation. "`e`" means "times 10 to the power of" or "`x10^`".

Open a Script File

Most of the time, we want to code in a script file instead of the console, as a script file can be edited, saved, and shared.
To open the file, click File -> New File -> R Script, or click the page/plus icon and select R Script. As a shortcut, you can click Ctrl + Shift + N.

Functions

A function is code that performs a specific operation. In R, functions are called by using their name, followed by open and closing parentheses.
For example, we might want to know what our current working directory is. We can use `getwd()` to check.

```
getwd()
```

```
## [1] "G:/Shared drives/Research_Data_Services/Workshops_2024_2025/lunch_hour_series_spring_2025"
```

Mathematical functions perform a mathematical operation on the value or values we provide.
These values constitute the input to the function and are called "arguments" in R lingo. Arguments are placed inside the parentheses and are separated by commas.
Test a couple of these:

```
log(1)
```

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

```
sum(2, 4, 7)
```

```
## [1] 13
```

```
mean(1:10) # Note the use of a semi-colon to indicate a series
```

```
## [1] 5.5
```

Logical operators are used to compare things in R. These are written like a math equation and return `TRUE` or `FALSE`. `==` tests for equality.

```
1 == 1
```

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

```
1 == 2
```

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

An exclamation point means "not", so "not equals" is `!=`.

```
1 != 1
```

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

```
1 != 2
```

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

R can also do less than, greater than, less than or equal, and greater than or equal:

```
2 > 3
```

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

```
-5 <= 9
```

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

R can also combine logical values with the "&" (AND - TRUE if both sides are TRUE) and "|" (OR - TRUE if at least one side is TRUE):

```
TRUE & FALSE
```

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

```
x <- 3
x < 5 | x > 10
```

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

Practice to become comfortable with logical operators; try three more comparisons of your own choosing.

Variable Assignment

We can store values that we enter or that are the outputs of functions using the `<-` assignment operator.

```
x <- 1/40
x
```

```
## [1] 0.025
```

We can reuse this variable, such as by passing it to a function:

```
log(x)
```

```
## [1] -3.688879
```

We can reassign new values to existing variables. This will overwrite the existing values:

```
x <- 100
x
```

```
## [1] 100
```

Practice: Create a variable that contains a number. Add another number to that variable, and save the result as a new variable. Then add the first and second variable. What are the results? Example:

```
x <- 10
y <- x + 5
x + y
```

```
## [1] 25
```

Introduction to Vectors

We will cover this in more depth in the next session, but let's look at one type of R object: a vector. A vector is a set of one or more values in a particular order. The order of each value is captured by its numeric index, beginning with the number 1.

We can create vectors in various ways:

```r
my_vector <- 1:5
my_vector
```

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

Note that the colon stands for all values between the first and the last.

Also of note: R doesn't automatically print the result of variable assignment; to see the result, you must type in the name of the variable (in this case, my_vector).

Alternatively, you can wrap the entire command with parentheses!

```r
(my_sequence <- seq(1, 10, by = 0.1))
```

```
##  [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4
## [16]  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9
## [31]  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1  5.2  5.3  5.4
## [46]  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9
## [61]  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.0  8.1  8.2  8.3  8.4
## [76]  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9
## [91] 10.0
```

Here, we use the function `seq()` to create a vector of values between 1 and 10, in increments of 0.1 (as specified with the `by` argument).

Practice: Create a vector of even numbers between 1 and 100. *Solution:*

```r
(even_numbers <- seq(2, 100, by = 2))
```

```
##  [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38
## [20]  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72  74  76
## [39]  78  80  82  84  86  88  90  92  94  96  98 100
```

We can also create a vector with the `c()` function, which stands for concatenate or combine. We string together the values, separated by a comma. Note that character data like these letters require quotation marks around them.

```r
grade_vector <- c("a", "b", "c", "d", "f")
grade_vector
```

```
## [1] "a" "b" "c" "d" "f"
```

All data in a vector must be the same data type; we will cover data types more in the next session, but for now, an example of two different types would be a number and a word. If you try to put two different types into the same vector, R will convert one of the types to the other type. Here is an example:

```r
mixed_vector <- c("a", "b", 1, 2)
mixed_vector
```

```
## [1] "a" "b" "1" "2"
```

What can we figure out by looking at the output?
R converts numeric data to character data if both types appear in the vector. This is called coercion. In a later session, we will discuss how to coerce variables into different data types!

Installing Packages

Functions to know:
`install.packages("packagename")` installs a package,
`update.packages()` updates a package, and
`library(packagename)` loads a package into the current session.

Also see the Packages tab in the bottom-right Miscellaneous area to see available and loaded packages.

Practice: `install.packages("tidyverse")` then `library(tidyverse)`

Getting Help

Get help on a function with ? or help(). Example: `?median` or `help(median)`

Coding Best Practices

- Use a hash tag (#) to comment on code.
- Break code into small, discrete chunks.
- Begin scripts with the packages you have included in the code.
- Save functions in a separate script.
- Store data files in a data sub-directory (folder).
- Store scripts in a separate folder from your data.
- Treat data as read-only (don't modify the data file itself).
- Treat generated output as disposable (everything should be reproducible through code).
- Variable names:
  - Spaces aren't allowed; use _ instead.
  - Must start with a letter.
  - Avoid using the names of existing functions.

Bonus practice:

1. Create a variable that contains a vector of numeric values.
2. Find the mean of that variable and save it as a new variable.
3. Find the natural log of the new variable.