SPEC REU: Introduction to R

Alix Ziff, Gaea Morales, Zachary Johnson

Summer 2024

Welcome to the first module of your Data Science training in R! In this module, we'll dive into the essentials of R. We'll cover everything from setting up your workspace in RStudio to performing basic operations and creating objects in R. We'll also introduce you to vectors and variables, and guide you through using arithmetic and logical operators.

By the time you finish this module, you'll be well-equipped to navigate R and ready to explore its capabilities in more depth in the modules to come.

What is R?

R is a programming language widely used for statistical computing and data visualization. Unlike other software like Stata or SPSS, R is open source (free and frequently updated), and supported by a vibrant community of programmers and statisticians.

R has become the go-to programming language for political science and international relations scholars, enabling everything from regression analysis and data management to creating compelling visualizations.

Let's get started!

Before you begin, make sure you've downloaded both R and RStudio. If you haven't done so yet, you can download R from Download and Install R and RStudio from RStudio Desktop.

R is the core statistical computing software that powers the analysis and data processing. RStudio, on the other hand, provides a user-friendly interface that makes it easier to write, run, and manage your R code.

Navigating the RStudio Interface

As you begin working with R, it's important to familiarize yourself with the RStudio interface, as each section plays a crucial role in your data analysis workflow.

- **R Script:** This is generally where you'll write and save your code. You can create a new script by navigating to the menu: `File > New File > R Script`. It's best practice to type your code here and save it as a `.R` file to keep track of your work.
- **Console:** Located by default below the R Script window, the Console is where your code is executed. While you can type commands directly into the Console, using the R Script allows you to save your commands for future reference.
- **Global Environment:** Positioned to the right of the R Script, the Global Environment keeps track of the data and objects you create or manipulate during your session.
- **Files, Plots, Packages, Help, and Viewer:** These tabs are located below the Global Environment. They allow you to manage your files, view plots and visualizations, load and manage packages, access help files, and preview data or visual content.

Running Commands

In R, executing commands (also known as functions) is straightforward. You can run a command by highlighting the code and either clicking `Run` at the top of the R script window or pressing `Ctrl` (or `Cmd` on a Mac) and `Enter` simultaneously. This will execute the selected code in the Console.

Annotating your Code!

When writing code in R, you can add comments that won't be executed by preceding any word or line with a `#`. Annotating your code is crucial for understanding what the code does when you revisit the file later and is particularly important when collaborating with others. Clear, well-commented code helps ensure that your logic and processes are easily followed by anyone who may work with or review your scripts.

Storing Objects in R

In R, you can assign values to an **object**, which act as containers for the values you create, use, or manipulate. You can name objects almost anything you like (it's recommended to choose names that clearly reference the data or purpose of the object). However, there are some rules to keep in mind: object names cannot begin with numbers, contain special characters, or use words that R reserves for specific functions or operators (known as "reserved words"). Examples of reserved words include `function`, `false`, `if`, `else`, `repeat`, `for`, and `NA`. For the full list, type `help(reserved)` or `?reserved` and run the code in your R script.

You can assign values to an object using the assignment operator `<-`. For example, to store the number `10102020` as an object called `luckyno`, you would write:

```
# Assigning the value 10102020 to the object named 'luckyno'
luckyno <- 10102020
```

You can then view the value of `luckyno` by using the `print()` function or simply typing the object's name:

```
# Printing the value stored in the object 'luckyno' to the Console
## Display the value of 'luckyno' using print() function
print(luckyno)
```

```
## [1] 10102020
```

```
## Display the value of 'luckyno' by typing 'luckyno'
luckyno
```

```
## [1] 10102020
```

Arithmetic in R

R is a powerful tool capable of handling complex calculations, but it can also be used as a basic calculator for simple arithmetic operations. You can perform calculations directly in the Console or within your scripts.

For example, multiplying two numbers:

```
# Multiply two numbers
5*2
```

```
## [1] 10
```

```
# This will return 10 in the Console
```

You can also use numbers stored in objects, which allows you to reuse values without typing them out again:

```
# Divide object 'luckyno' by 2
luckyno/2
```

```
## [1] 5051010
```

```
# This will divide the value 10102020 stored in 'luckyno' by 2
```

R also allows you to create new objects from the results of calculations. This is useful for storing the outcomes of operations for later use:

```
# Save the results of calculations as 'result'
result <- 1601.6 * 5

# The result of the multiplication is stored in the object 'result'
```

Additionally, you can update or overwrite the value of existing objects. By assigning a new value to an object with the same name, you replace its previous content:

```
# Replace the value of the object 'result'
result <- 8007393-742

# The value of 'result' is updated with the result of this subtraction
```

Using Arithmetic Operators

R provides a range of arithmetic operators to perform basic mathematical calculations:

	Operator
Addition	`x + y`
Subtraction	`x - y`
Multiplication	`x * y`
Division	`x / y`
Exponentiation	`x ^ y` (or `x**y`)
Remainder of division (modulus)	`x %% y`
Division rounded down to the nearest integer	`x % / % y`

In addition to arithmetic operations, R also supports logical operators that return either `TRUE` or `FALSE`. These are particularly useful for making comparisons:

	Operator
Less than	`<`
Less than or equal to	`<=`
Greater than	`>`
Greater than or equal to	`>=`
Exactly equal to	`==`
Not equal to	`!=`
Not `x`	`!x`
`x` or `y` (TRUE if either `x` or `y` is TRUE)	`x | y`
`x` and `y` (TRUE only if both `x` and `y` are TRUE)	`x & y`
Is `x` in `y` (checks if `x` is an element of the vector `y`)	`x %in% y`

For example, check if the number 8 is in the vector `c(2,3,4,5)`.

```
# Check if the number 8 is in the vector c(2,3,4,5)
8 %in% c(2,3,4,5)
```

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

```
# This expression will return FALSE, as 8 is not included in the vector
```

Using these operators, you can perform a wide range of calculations and logical checks, which are fundamental to data analysis in R.

Vectors & Variables

In R, a vector is the most basic type of data structure and serves as a fundamental building block for data analysis. A vector is essentially a sequence of elements that are all of the same data type. R supports several types of variables that can be stored in vectors:

- **Numerical**: This includes any numeric values, such as integers and doubles (which refer to how R stores and represents these numbers in memory).
- **Character**: Known as strings in other programming languages, these are sequences of text or alphanumeric data. Character vectors store data that is not inherently numerical or ordered.
- **Logical**: This type of vector contains Boolean values, represented as `TRUE` or `FALSE`.
- **Factor**: Factors are used to represent categorical data, often in an ordered manner. They are useful for data that falls into a limited number of categories or levels, such as ordinal variables.

To create a vector in R, you use the `c()` function, which stands for "concatenate." This function combines individual elements into a single vector. The basic syntax for creating a vector is as follows:

```
name_of_vector <- c("element1", "element2", "element3")
```

For example, if you wanted to create a vector of numbers, you could do:

```
numbers <- c(1, 2, 3, 4, 5)
```

Or, if you wanted to create a character vector:

```
words <- c("apple", "banana", "cherry")
```

Common Pitfalls to Avoid When Coding in R

When coding in R, there are a few common mistakes that can trip you up. Here's a quick guide to help you avoid those "Oh No!" moments:

- Make sure your assignment operator `<-` is correctly oriented.
- Remember that any line beginning with `#` is treated as a comment and won't be executed as code.
- R is case-sensitive, so make sure you're using the correct capitalization. For example, `data` and `Data` would be recognized as two different objects.
- Always ensure that quotation marks, parentheses, and brackets are properly paired.
- Check for typos. Typos are a frequent source of errors.
- Assign unique names to your objects to avoid accidentally overwriting existing data. Use the `ls()` command to list all objects you've defined in your session. This can be especially helpful if you're unsure whether you've already used a name for an object.

Getting Help

If you run into a problem and aren't sure how to fix it, don't worry—it's completely normal when you're just starting to learn how to code. Fortunately, there are plenty of resources available to help you:

- **Google It**. No seriously. If you are struggling, chances are someone else has faced the same issue. A quick search often leads to helpful answers on forums like Stack Overflow or dedicated R websites.
- **R Studio Help Menu.** The 'Help' tab in RStudio, located between 'Packages' and 'Viewer,' provides detailed documentation on functions, packages, and commands. You can also access help directly in your script by typing a question mark before a function, like `?c()`.
- **SPEC Statistics Consulting Hours.** There's at least an hour every single day with a brilliant SPEC student or doctoral affiliate. Take advantage.