1. Practical Introduction to R

Bernd Wurth

Table of contents

1 Introduction

This introduction covers the basics of R programming. As you progress, you'll discover more advanced features and packages that extend R's capabilities even further. Remember to use the help function (`?function_name`) to learn more about specific functions and their usage.

2 Executing R Scripts: Full vs. Partial Execution

2.1 Full Execution of an R Script

Running a script from top to bottom is useful when you want to execute all the code at once. This is typically done after you've written and verified the entire script.

Steps:

1. Save your script with a `.R` extension, such as `my_script.R`.
2. Use the `source()` function in R to execute the entire script:

```
source("my_script.R")
```

3. Alternatively, in RStudio:
   - Click the Source button at the top of the script editor.
   - Use the shortcut `Ctrl + Shift + S` (Windows) or `Cmd + Shift + S` (Mac).

Example script:

```r
# Define a function
add_numbers <- function(x, y) {
  return(x + y)
}

# Perform calculations
result <- add_numbers(5, 3)
print(result)

# Generate a sequence
seq_data <- seq(1, 10, by = 2)
print(seq_data)
```

2.2 Partial Execution of an R Script

Running parts of your script manually is useful during the development and debugging process. This allows you to test specific sections without executing the entire script.

Steps in RStudio:

1. Highlight the portion of the script you want to run.
2. Press `Ctrl + Enter` (Windows) or `Cmd + Enter` (Mac) to run the selected lines.
3. If you want to run only the current line, place the cursor on that line and press the same shortcut.

```r
# Define a function
add_numbers <- function(x, y) {
  return(x + y)
}

# Highlight and run the following line:
# result <- add_numbers(5, 3)

# Debugging this line separately:
print(result)

# Highlight and run this block to test sequences:
seq_data <- seq(1, 10, by = 2)
print(seq_data)
```

2.3 Best Practices for Script Organization

Divide your script into sections using comments:

```r
# Section 1: Load libraries
library(ggplot2)

# Section 2: Load data
data <- read.csv("data.csv")

# Section 3: Analysis
summary(data)
```

Use descriptive variable and function names to make scripts easier to understand.

Keep your script modular—write functions for reusable code blocks.

Include comments to explain complex logic or calculations.

3 Basic R Syntax

R is case-sensitive and uses the `<-` operator for assignment (though `=` can also be used). Comments start with `#`.

```
# This is a comment
x <- 5  # Assign the value 5 to x
y = 10  # This also works, but <- is more common in R
```

4 Objects and Variable Assignment

In R, you can assign values to variables using the assignment operator `<-`:

```
my_variable <- 42
my_name <- "Alice"
```

You can view the contents of a variable by typing its name:

```
my_variable
my_name
```

5 Data Types in R

R has several basic data types:

1. Numeric (real numbers)
2. Integer
3. Character (string)
4. Logical (boolean)
5. Complex

```
num_var <- 3.14
int_var <- 42L  # The 'L' suffix creates an integer
char_var <- "Hello, R!"
log_var <- TRUE
comp_var <- 3 + 2i
```

You can check the type of a variable using the `class()` function:

```
class(num_var)
class(char_var)
```

6 Operators in R

R supports various types of operators:

1. Arithmetic: `+`, `-`, `*`, `/`, `^` (exponent), `%%` (modulus)
2. Relational: `<`, `>`, `<=`, `>=`, `==`, `!=`
3. Logical: `&` (and), `|` (or), `!` (not)

```
x <- 10
y <- 3

x + y
x > y
(x > 5) & (y < 5)
```

7 Basic Data Structures

R has several important data structures:

1. Vectors: One-dimensional arrays that can hold data of the same type
2. Lists: Can hold elements of different types
3. Matrices: Two-dimensional arrays with data of the same type
4. Data Frames: Two-dimensional arrays that can hold different types of data

```r
# Vector
vec <- c(1, 2, 3, 4, 5)

# List
my_list <- list(name = "Alice", age = 30, scores = c(95, 87, 91))

# Matrix
mat <- matrix(1:9, nrow = 3, ncol = 3)

# Data Frame
df <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  age = c(25, 30, 35),
  city = c("New York", "London", "Paris")
)
```

8 Functions in R

R has many built-in functions, and you can also create your own:

```r
# Using a built-in function
mean(c(1, 2, 3, 4, 5))

# Creating a custom function
square <- function(x) {
  return(x^2)
}

square(4)
```

9 Control Structures

R supports common control structures like if-else statements and loops:

```r
# If-else statement
x <- 10
if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is not greater than 5")
}

# For loop
for (i in 1:5) {
  print(i^2)
}

# While loop
i <- 1
while (i <= 5) {
  print(i^2)
  i <- i + 1
}
```

10 Reading and Writing Data

R can read data from various file formats. Here's an example with CSV:

```r
# Reading a CSV file
# Assuming you have a file named "data.csv" in your working directory
data <- read.csv("data.csv")

# Writing a CSV file
write.csv(df, "output.csv", row.names = FALSE)
```

For this example, you'll need to create a "data.csv" file in your working directory or adjust the file path accordingly.

11 Basic Data Manipulation

R provides many functions for manipulating data:

```r
# Assuming we're using the 'df' data frame from earlier

# Selecting a column
df$name

# Filtering rows
df[df$age > 25, ]

# Adding a new column
df$is_adult <- df$age >= 18

# Summarizing data
summary(df)
```

12 Introduction to Basic Plotting

R has powerful plotting capabilities. Here's a simple example:

```r
# Create some data
x <- 1:10
y <- x^2

# Create a scatter plot
plot(x, y, main = "Square Function", xlab = "x", ylab = "y")

# Add a line
lines(x, y, col = "red")
```

We will explore more advanced plotting with the `ggplot` package later.