R Refresher

Actuarial Data Manipulation with R – CAS Spring Meeting 2024

Denis Dreano

2024-05-05

Comments, Variables, Data Types, and Packages

Comments

Comments in R are preceded by the # symbol. They are used to annotate code for readability and documentation purposes.

```
# This is a comment in R
```

Variables

Variables in R are used to store data values. Variable names can consist of letters, numbers, and periods (.), but cannot start with a number or contain spaces. Variable assignment is done using the <- or = operator.

```
x <- 10         # Assign 10 to variable x
y <- "Hello"    # Assign "Hello" to variable y
```

Data Types

R supports various data types, including numeric, character, logical, factor, and more. Use the class() function to determine the data type of a variable.

```
x <- 10         # Numeric
y <- "Hello"    # Character
z <- TRUE       # Logical
```

Packages:

```
# Install package
install.packages("package_name")

# Load package
library(package_name)
```

Vectors

Vectors are one-dimensional arrays that can hold multiple values of the same data type. They are created using the c() function.

```
# Create a numeric vector
numeric_vector <- c(1, 2, 3, 4, 5)
# Create a character vector
character_vector <- c("a", "b", "c", "d", "e")
# Create a logical vector
logical_vector <- c(TRUE, FALSE, TRUE, FALSE)
```

Other ways to create vectors

```
# Create a sequence of numbers
seq_vector <- 1:10
# Create a sequence with a specific increment
seq_vector <- seq(1, 10, by = 2)
# Create a vector with repeated values
rep_vector <- rep(1, times = 5)
```

Vectors can be indexed using square brackets [ ].

```
# Access the first element of a vector
numeric_vector[1]
# Access the last element of a vector
numeric_vector[length(numeric_vector)]
```

Vectors of the same type can be combined using the c() function.

```
combined_vector <- c(1:10, rep(0, 5))
```

Missing Values (`NA`)

Missing values in R are represented by `NA`. They can be included in vectors.

```
missing_vector <- c(1, 2, NA, 4, 5)
```

Operators

R supports arithmetic, relational, logical, and assignment operators. Here are some examples:

```r
# Arithmetic operators
x + y    # Addition
x - y    # Subtraction
x * y    # Multiplication
x / y    # Division
x %% y   # Modulus
x ^ y    # Exponentiation

# Relational operators
x == y   # Equal to
x != y   # Not equal to
x > y    # Greater than
x < y    # Less than
x >= y   # Greater than or equal to
x <= y   # Less than or equal to
```

```r
# Logical operators
x && y   # Logical AND (scalar)
x & y    # Logical AND (vector)
x || y   # Logical OR (scalar)
x | y    # Logical OR (vector)
!x       # Logical NOT
```

Most operators in R are vectorized, meaning they operate element-wise on vectors.

Control Structures

R supports various control structures for flow control, including if-else statements, for loops, while loops, and more.

If-else statement

```
if (condition) {
  # Code to execute if condition is TRUE
} else if (another_condition) {
  # Code to execute if another_condition is TRUE
} else {
  # Code to execute if condition is FALSE
}
```

For loop (we will see better ways to do loops later)

```
for (i in 1:5) {
  # Code to execute for each iteration
}
```

Functions

Functions in R are defined using the function() keyword. They can take arguments and return values.

Function definition

```r
my_function <- function(arg1, arg2) {
  # Code to execute
  result <- arg1 + arg2
  result
  # implicitly returns the last evaluated value
  # use the return(result) to return explicitly
}
```

Function call

```r
my_result <- my_function(10, 20)
```

Functions are first-class objects in R

Functions can be assigned to variables, passed as arguments to other functions, and returned from functions.

```r
call_function <- function(f, x) {
  f(x)
}
call_function(function(x) x^2, 5) # Returns 25

build_cubic_function <- function() {
  function(x) x^3
}
```