Introduction to R

Tan Sein Jone

University of British Columbia

July 26, 2024

Table of contents

Table of Contents

Functional Programming

- Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- It is a declarative type of programming style.
- It focuses on what to solve rather than how to solve.
- It uses expressions instead of statements.
- It is based on mathematical functions.

Pure Functions

- A pure function is a function where the output value is determined by its input values, without observable side effects.
- This is how functions in math work: Math.cos(x) will, for the same value of x, always return the same result.
- Pure functions are easier to reason about and test.

```
pure_function <- function(x, y){
    return(x + y)
}

impure_function <- function(x, y){
    print(x)
    return(x + y)
}
```

First Class Functions

- In functional programming, functions are first-class citizens.
- This means that functions can be assigned to variables, passed as arguments, and returned from other functions.
- This allows for the creation of higher-order functions.

```
add <- function(x, y){
    return(x + y)
}

subtract <- function(x, y){
    return(x - y)
}

operate <- function(func, x, y){
    return(func(x, y))
}

operate(add, 5, 3)
operate(subtract, 5, 3)
```

Higher Order Functions

- Higher-order functions are functions that can either take other functions as arguments or return them as results.
- This is possible because functions are first-class citizens.
- Higher-order functions allow us to abstract over actions, not just values.

```
add <- function(x, y){
    return(x + y)
}

subtract <- function(x, y){
    return(x - y)
}

create_operator <- function(op){
    if(op == "add"){
        return(add)
    } else if(op == "subtract"){
        return(subtract)
    }
}

operator <- create_operator("add")
operator(5, 3)
```

Table of Contents

Data Handling

- Data handling is a crucial part of data analysis.
- R has a wide range of functions and packages that make data handling easier.

Data Structures

- R has several data structures that are used to store data.
- The most common data structures are vectors, matrices, data frames, and lists.

Vectors

- A vector is a one-dimensional array that can hold numeric, character, or logical data.
- Vectors are created using the c() function.
- Vectors can be of two types: atomic vectors and lists.

```
numeric_vector <- c(1, 2, 3, 4, 5)
character_vector <- c("a", "b", "c", "d", "e")
logical_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
```

Matrices

- A matrix is a two-dimensional array that can hold numeric, character, or logical data.
- Matrices are created using the matrix() function.
- Matrices are created by combining vectors.

```
matrix_1 <- matrix(1:9, nrow = 3, ncol = 3)
matrix_2 <- matrix(letters[1:9], nrow = 3, ncol = 3)
matrix_3 <- matrix(
    c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE),
    nrow = 2, ncol = 3
)
```

Data Frames

- A data frame is a two-dimensional array that can hold numeric, character, or logical data.
- Data frames are created using the data.frame() function.
- Data frames are similar to matrices, but they can hold different types of data in each column.

```
data_frame <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)
```

File IO

- R has functions that allow you to read and write data from and to files.
- The most common file formats are CSV, Excel, and text files.
- R has functions that allow you to read and write data in these formats.

```
data <- read.csv("data_frame.csv")
write.csv(data, "data_frame.csv")
```

Data Manipulation

- Data manipulation is the process of transforming data to make it more useful for analysis.
- R has functions and packages that make data manipulation easier.
- The most common data manipulation tasks are filtering, sorting, and aggregating data.

```
data <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

# Filter data
filtered_data <- data[data$age > 30, ]

# Sort data
sorted_data <- data[order(data$age), ]

# Aggregate data
aggregated_data <- aggregate(
    data$age,
    by = list(data$married),
    FUN = mean
)
```

Merge and Join

- Merge and join are two common data manipulation tasks.
- Merge is used to combine two data frames based on a common column.
- Join is used to combine two data frames based on a common column.

```
data_1 <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    married = c(TRUE, FALSE, TRUE)
)

data_2 <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    salary = c(50000, 60000, 70000)
)

merged_data <- merge(data_1, data_2, by = "name")
joined_data <- merge(
    data_1, data_2, by = "name",
    all = TRUE
)
```