Kalinga University

Faculty of Computer Science & Information Technology

Course- BCAAIML
Subject- R Programming
Subject Code – BCAAIML401 **Sem- IV**

Unit 3

Creating Data Frames – Matrix-like operations in frames – merging Data frames – Applying functions to Data Frames – Factors and Tables – Factors and levels – Common Functions used with factors – Working with tables – Other factors and table related functions – Control statements – Arithmetic and Boolean operators and values – Default Values for arguments – Returning Boolean Values – Functions are objects – Environment and scope issues – Writing Upstairs – Recursion – Replacement functions – Tools for Composing function code – Math and Simulation in R.

Creating Data Frames in R

A **data frame** in R is a two-dimensional, tabular data structure, which can hold data of different types (numeric, character, factor, etc.) across columns. It is similar to a spreadsheet or SQL table and is one of the most commonly used data structures for data manipulation in R.

Creating a Data Frame:

- Use the data.frame() function to create a data frame:

```
df <- data.frame(name = c("Alice", "Bob", "Carol"),
                 age = c(25, 30, 35),
                 gender = c("F", "M", "F"))
```

- Each column can be of a different data type (e.g., character, numeric, or logical).

Accessing Data Frame:

- To access a specific column, use df$column_name or df[, "column_name"].
- To access a specific row or subset, use indexing df[row, column].

Matrix-like Operations in Data Frames

Although data frames can hold different types of data, matrix-like operations such as arithmetic and logical operations can still be performed, but they require homogeneous data types for specific operations.

- **Column-wise operations**: You can perform operations on entire columns, for example:

```
df$age <- df$age + 5  # Adds 5 to every value in the 'age' column
```

- **Matrix Operations**: You can convert a data frame into a matrix using as.matrix() and then perform matrix operations.

```
mat <- as.matrix(df[ ,c("age")])  # Convert 'age' column to a matrix
mat + 10  # Perform matrix operation
```

Merging Data Frames

Merging data frames in R is similar to SQL joins, allowing you to combine multiple data frames based on shared column(s).

- **merge() function**:

```
df1 <- data.frame(ID = c(1, 2, 3), name = c("A", "B", "C"))
df2 <- data.frame(ID = c(2, 3, 4), age = c(25, 30, 35))

merged_df <- merge(df1, df2, by = "ID")  # Inner join by 'ID'
```

- You can specify the type of join: all.x = TRUE for left join, all.y = TRUE for right join, and all = TRUE for full outer join.

Applying Functions to Data Frames

You can apply functions across data frames using the apply(), lapply(), sapply(), or tapply() functions.

- **apply()**: Applies a function to rows or columns of a matrix or data frame.

```
apply(df, 2, mean)  # Apply mean function to each column (2 means columns)
```

- **lapply() and sapply()**: Used to apply a function to each element of a list or vector.

```
lapply(df, mean)  # Apply mean to each column of the data frame
```

- **tapply()**: Apply a function to subsets of a vector, grouped by a factor.

```
tapply(df$age, df$gender, mean)  # Mean age by gender
```

Factors and Tables

A **factor** is an R data type used to represent categorical data. Factors are useful when the data represent a fixed number of unique values, such as gender, education level, or rating scores.

- **Levels**: The possible values of a factor are called **levels**.

```
gender <- factor(c("M", "F", "F", "M"))
levels(gender)  # Returns the levels: "M", "F"
```

- **Common Functions with Factors**:
  - factor(): Converts a variable to a factor.
  - levels(): Retrieves or sets the levels of a factor.
  - table(): Creates a contingency table of counts for a factor or categorical data.

```
table(gender)  # Returns a frequency table for gender
```

Control Statements

Control statements allow you to control the flow of your program.

- **if and else**:

```
if (x > 0) {
  print("Positive")
} else {
  print("Non-positive")
}
```

- **for loop**: Iterates over a sequence of elements.

```
for (i in 1:5) {
  print(i)
}
```

- **while loop**: Repeats as long as a condition is TRUE.

```
while (x < 10) {
  x <- x + 1
}
```

- **repeat loop**: Loops indefinitely until a break condition is met.

Arithmetic and Boolean Operators

R supports both **arithmetic operators** and **boolean operators**:

- **Arithmetic operators**:
    - +, -, *, /, ^ for addition, subtraction, multiplication, division, and exponentiation.
- **Boolean operators**:
    - &: Element-wise logical AND
    - &&: Logical AND (only evaluates the first element)
    - |: Element-wise logical OR
    - ||: Logical OR (only evaluates the first element)
    - ==: Equality comparison
    - !=: Inequality comparison

Default Values for Arguments

In R, functions can have default values for their arguments. If no argument is passed, the default value is used.

```
my_function <- function(x = 10) {
  print(x)
}

my_function()  # Prints 10 (default value)
my_function(20)  # Prints 20
```

Returning Boolean Values

Functions in R can return boolean values (TRUE or FALSE) based on conditions.

```
is_even <- function(x) {
  return(x %% 2 == 0)
}

is_even(4)  # TRUE
is_even(5)  # FALSE
```

Functions as Objects

In R, functions are objects, meaning you can assign a function to a variable and pass it around like any other object.

```
f <- function(x) { x^2 }
f(4)  # 16
g <- f
g(4)  # 16
```

Environment and Scope Issues

R has different environments where variables are stored (e.g., global environment, local function environment). Scope refers to where a variable can be accessed.

- **Global environment**: Variables are accessible throughout the script.
- **Local environment**: Variables defined inside a function are local to that function.

Writing Functions (Recursion)

A **recursive function** is a function that calls itself. Recursion is useful for problems that can be broken down into smaller sub-problems of the same type.

```
factorial <- function(n) {
  if (n == 0) {
    return(1)
  } else {
    return(n * factorial(n - 1))
  }
}

factorial(5)  # 120
```

Replacement Functions

Replacement functions are used to modify existing data structures (e.g., data frames, vectors, lists).

- Example: Replace elements of a vector using [:

  ```
  x <- c(1, 2, 3)
  x[2] <- 10  # Replaces the second element with 10
  ```

Tools for Composing Function Code

R provides various tools to help compose and manage function code effectively:

- **debug()**: To debug a function.
- **traceback()**: To show the sequence of function calls leading to an error.
- **RStudio**: A powerful IDE for R programming with debugging and profiling tools.

Math and Simulation in R

R is particularly strong in mathematical and statistical computing. It has built-in functions for simulations:

- **Random Number Generation**: Functions like runif(), rnorm(), and sample() allow you to simulate data.

  ```
  sample(1:10, 5)  # Random sample from 1 to 10
  ```

- **Mathematical Functions**: R supports a wide range of mathematical functions like log(), exp(), sin(), cos(), etc.