Statistical Computing Techniques using R - Unit 2

First steps in graphics, The basics of R syntax, The R workspace, Matrices and lists, Subsetting, System-defined functions; the help system, Errors and warnings; coherence of the workspace. Data input and output; interface with other software packages, Writing your own code; R script. Good programming practice, R syntax -- further steps The parentheses and brackets; =, == and <-. Apply-type functions Compiling and applying functions

1. Basics of R Syntax and Workspace

Assignment and Comparison Operators

R offers flexible operators that can sometimes confuse beginners.

- <- is the traditional assignment operator in R and is widely used in the R community. It clearly separates assignment from comparison, improving readability.
- = also works for assignment, but it is commonly used for function arguments. Using it for variable assignment can sometimes cause ambiguity.
- == is strictly a comparison operator. It checks equality and always returns a logical (TRUE/FALSE).

```
x <- 10 # Assign
y = 20 # Assign
x == 10 # TRUE (comparison)</pre>
```

f Good practice: Use <- for assignment, = for arguments in functions, and == for comparisons.

Workspace in R

The workspace is like R's **working memory**. Everything you create—variables, functions, data frames, models—resides here until you close R.

- Checking current objects: ls()
- Removing specific objects: rm(x)
- Removing all objects (cleaning memory): rm(list=ls())
- Saving the workspace: save.image("project.RData")
- Loading a saved workspace: load("project.RData")

f Importance:

- Keeps your work intact between sessions.
- Essential for large projects where re-running code every time is inefficient.
- Helps organize projects into reusable components.

2. Data Structures in R

Matrices

Matrices are widely used in **statistics**, **linear algebra**, **and scientific computing**. They are strictly homogeneous, meaning all elements must be of the same type.

Creating a matrix:

mat <- matrix(1:6, nrow=2, ncol=3, byrow=TRUE) mat

👉 Use Cases:

- Representing numeric datasets in tabular form.
- Performing operations like matrix multiplication, eigenvalues, and solving linear equations.

Lists

Lists are R's most flexible data structure. They allow combining different object types. A list can even contain other lists, making it hierarchical.

mylist <- list(Name="Alice", Age=22, Scores=c(90,85,88)) mylist\$Scores

- **b** Use Cases:
 - Storing regression results (coefficients, residuals, fitted values).
 - Combining unrelated data like text, numbers, and models.

Subsetting

Subsetting is the core technique for extracting meaningful parts of data.

- Vectors: Access by index or condition.
- Matrices: Access by row and column positions.
- Lists: Access by \$, double brackets, or index.
- *†* Why Important?
 - Allows data cleaning and analysis on selected portions.
 - Essential for handling large datasets efficiently.

3. Functions in R

System-Defined Functions

R comes with hundreds of pre-built functions for mathematical, statistical, and text operations.

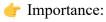
Figurificance: These functions save time and reduce the need to "reinvent the wheel."

• Examples:

User-Defined Functions

You can create your own functions when system functions are insufficient.

```
square <- function(x) {
  return(x^2)
}
square(5) # 25</pre>
```



- Encourages modular programming.
- Makes code reusable and easier to debug.
- Supports documentation for clarity.

4. R Help System

R includes a **comprehensive help system**, making it self-contained for learners.

- ?function \rightarrow Opens manual page.
- help.search("keyword") → Finds related functions.
- example(mean) \rightarrow Runs examples.

- *b Significance:*
 - Reduces dependence on external resources.
 - Encourages self-learning.
 - Provides official documentation and examples.

5. Errors and Warnings

- Errors: Fatal issues; stop program execution. Example: using a string where a number is expected.
- Warnings: Non-fatal; execution continues but output may be unreliable.

sqrt(-9) # Warning: NaNs produced

log("abc") # Error

- *†* Importance:
 - Errors must be fixed immediately.
 - Warnings alert the user about potential issues but allow flexibility.

6. Data Input and Output

Data I/O is crucial for real-world use of R, since most data is external.

- Input methods:
 - o $scan() \rightarrow For quick entry of vectors.$
 - o read.table() \rightarrow Reads tabular data.
 - \circ read.csv() \rightarrow Reads CSV files.
- Output methods:
 - \circ write.table() \rightarrow Saves data to a text file.
 - o write.csv() \rightarrow Saves data to CSV format.

7. R Scripts

Scripts are the backbone of automation and reproducibility in R.

- Saved as .R files.
- Allow running an entire project at once.
- Used in collaboration to share analysis steps.

source("myscript.R")

- *†* Importance:
 - Ensures that work can be replicated.
 - Allows team-based workflows.
 - Reduces errors from typing commands repeatedly.

8. Apply-Type Functions

Apply-type functions are R's alternative to explicit loops. They make code **shorter**, **cleaner**, **and faster**.

• apply(): Works on rows or columns of matrices.

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

apply(mat, 1, sum) # Row sums

f Importance: Essential in data analysis workflows where vectorization increases speed and efficiency.

9. Programming Constructs in R

Parentheses

- Used for grouping operations: (5+3)*2
- Required in function calls: print("Hello")
- Used in control flow conditions: if $(x > 0) \{ ... \}$

Conditional Statements

- If-Else: Basic decision-making.
- Nested If-Else: Handles multiple conditions.
- **Switch:** Efficient for multi-choice conditions.
- **f** Importance: Provides flexibility in writing decision-based programs.

Loops and Iterations

R provides three main loop types:

- For loop: Repeats for a fixed number of iterations.
- While loop: Continues until a condition becomes false.
- **Repeat loop:** Runs indefinitely until break is used.
- **/** Importance: Automates repetitive tasks like simulations, transformations, or iterative calculations.

10. Good Programming Practice in R

- Use meaningful and consistent variable names.
- Add comments (#) for clarity.
- Break code into reusable functions.
- Indent and format code consistently.
- Save and document scripts properly.
- Following best practices ensures code is readable, maintainable, and professional.