Statistical Computing Techniques using R - Unit 1

General introduction to computing, Using R as a calculator, Numbers, words and logicals; missing values (NA), Vectors and their attributes (names, length, type), System- and user-defined objects, Accessing data (data()). Data in the system and date outside the system (read.table, scan)

Using R as a Calculator

R is not just a programming language but also an interactive environment that can be used as a simple or advanced calculator. This makes it extremely powerful for quick calculations, testing, and prototyping statistical formulas.

Arithmetic Operations

R supports the basic arithmetic operations of addition, subtraction, multiplication, division, exponents, and modulus. These operations follow the standard rules of arithmetic (BODMAS precedence).

Examples:

```
2 + 3     # 5
10 - 4    # 6
6 * 7     # 42
20 / 4    # 5
2^3       # 8 (power)
10 %% 3   # 1 (modulus)
```

Logical Operations

In addition to arithmetic, R also supports logical operators, which return **TRUE** or **FALSE** values. Logical operations are crucial in conditions, decision-making, and filtering datasets.

```
5 > 3         # TRUE
4 == 4        # TRUE
7 < 2         # FALSE
TRUE & FALSE  # FALSE
TRUE | FALSE  # TRUE
```

👉 R's ability to combine **numeric** and **logical** operations makes it unique for statistical modeling, where decisions are often based on conditions.

Data Types in R: R provides flexible data types to handle both numerical and categorical information.

- **Numeric:** Used for integers and decimals. These are the most common type in statistical analysis (e.g., income, age, weight).
- **Character (words):** Used for textual data like names, categories, and labels. Always enclosed in quotes.
- **Logical:** Represents binary decisions (TRUE/FALSE). Very useful for comparisons and data filtering.

Example:

```
num <- 25
word <- "R programming"
logical <- TRUE
```

👉 Understanding data types is important because R automatically chooses the best type when creating objects. If mixed types are stored in the same object, R performs **type coercion** (e.g., numbers + words → everything becomes character).

Missing Values (NA) in R

In real-world datasets, it is common to encounter missing information. R represents this as **NA (Not Available)**.

- **Impact:** Missing values affect calculations because most functions return NA if they encounter missing data.
- **Identification:** The function is.na() can be used to locate missing values.
- **Handling Strategies:**
  - **Removal:** Using na.omit() or na.exclude().
  - **Replacement:** Replace NA with zero, mean, median, or another estimated value.

Example:

```
x <- c(10, 20, NA, 40)
mean(x)            # NA
mean(x, na.rm=TRUE)  # 23.33
x[is.na(x)] <- 0     # Replace NA with 0
```

👉 Handling missing values carefully ensures the accuracy of statistical analysis and prevents misleading results.

Vectors in R

Vectors are the **foundation of all data structures** in R. They represent ordered collections of values of the same type (numeric, character, or logical). Almost all data manipulation in R begins with vectors.

Types of Vectors:

1. **Numeric vectors:** Hold numbers.
2. **Character vectors:** Hold text strings.
3. **Logical vectors:** Hold TRUE/FALSE values.

Examples:

```
num_vec <- c(1, 2, 3)
char_vec <- c("apple", "banana")
log_vec <- c(TRUE, FALSE, TRUE)
```

👉 Vectors are essential for storing single-variable datasets and performing vectorized operations, which are faster and more efficient than loops.

Attributes of Vectors

Every vector has attributes that describe its properties.

- **Length:** Total number of elements.
- **Names:** Labels assigned to each element.
- **Type (class):** Defines the type of elements (numeric, character, logical).

Example:

```
marks <- c(85, 90, 78)
names(marks) <- c("Math", "Science", "English")
length(marks)   # 3
class(marks)    # "numeric"
```

👉 Adding names makes data self-descriptive, which is important for clear reporting and analysis.

Role of Vectors in Data Handling

Vectors play a central role in R because they are used to build larger structures such as matrices, arrays, data frames, and lists.

- They support **element-wise operations** (e.g., addition, multiplication).
- They can be **filtered using logicals**, making them powerful for data cleaning and analysis.

Example - Assigning Names and Accessing:

```
sales <- c(200, 350, 400)
names(sales) <- c("Jan", "Feb", "Mar")
sales["Feb"]   # 350
```

👉 Vectors simplify working with datasets by allowing easy referencing and manipulation.

System-Defined vs User-Defined Objects

R comes with many built-in objects that save time and effort. Users can also define their own objects.

- **System-defined objects:** Predefined in R.
  - letters → "a" to "z"
  - LETTERS → "A" to "Z"
  - pi → 3.141593
- **User-defined objects:** Created by programmers for their own analysis.

```
myvec <- c(10, 20, 30)
myvec[2]   # 20
```

👉 System objects are helpful shortcuts, while user-defined objects provide customization and control.

Accessing Datasets in R

R provides two main methods to access datasets:

1. **Built-in datasets (data()):** R includes sample datasets for practice and demonstration.
2. data(mtcars)
3. head(mtcars)
4. **External data (read.table()):** Import data from text files, CSV files, etc.
5. mydata <- read.table("data.txt", header=TRUE)
6. head(mydata)

👉 data() is mainly for learning and examples, while read.table() is used in real-world projects.

Comparison - data() vs read.table()

- **data():** Loads internal datasets, no file required.
- **read.table():** Reads from external sources, requires file path.

Example:

```
data(iris)     # Built-in dataset
iris[1:3, ]

mydata <- read.table("marks.txt", header=TRUE)  # External file
```

---

scan() Function in R

The scan() function is a quick method to read numeric or character input directly from the console.

Example:

```
x <- scan()
1 2 3 4 5
# Enter numbers and press Enter
```

Difference from read.table():

- scan() is best for simple, unstructured input (like a list of numbers).
- read.table() is best for structured tabular data.

Handling Vectors with Missing Values

Real-world data often has missing entries. R provides flexible ways to handle them.

- **Identify missing values:** is.na(x)
- **Remove them:** na.omit(x)
- **Replace them:** x[is.na(x)] <- mean(x, na.rm=TRUE)

👉 Choosing whether to **remove** or **replace** depends on the analysis context.

Assigning Names to Vector Elements

Assigning names makes vectors easier to read and interpret.

Example:

```
sales <- c(200, 350, 400)
names(sales) <- c("Jan", "Feb", "Mar")
sales
# Jan Feb Mar
# 200 350 400
```

👉 Names act like column headers in a table.

Importing External Data into R

R allows data import from many formats:

1. **Text/CSV files:**
2. data1 <- read.table("data.txt", header=TRUE)
3. data2 <- read.csv("data.csv")
4. **Excel files:** (with packages)
5. library(readxl)
6. data3 <- read_excel("data.xlsx")
7. **Quick input:**
8. x <- scan()

👉 Importing data correctly is the first step in any statistical project.

Logical Operations in R

Logical values are TRUE and FALSE. They are used for comparisons, filtering, and conditions.

Example:

```
x <- c(10, 20, 30, 40)
x > 25
# FALSE FALSE TRUE TRUE
```

👉 Logical operations allow quick extraction of subsets that satisfy specific conditions.

R Program Examples

Vector with numbers, words, and logicals:

```
vec <- c(10, "apple", TRUE)
vec
length(vec)   # 3
class(vec)    # "character" (coercion happens)
```

Filtering using logicals:

```
x <- c(5, 10, 15, 20)
x[x > 10]
# 15 20
```