Data Types and Structures

1. Theoretical Aspects

Vectors

- A vector is a one-dimensional array that holds elements of the same data type.
- Types of vectors:
  - **Numeric:** Contains numbers.
  - **Character:** Contains strings.
  - **Logical:** Contains `TRUE` or `FALSE`.
- Created using the `c()` function.

Example:

```
numeric_vector <- c(1, 2, 3)
char_vector <- c("apple", "banana", "cherry")
logical_vector <- c(TRUE, FALSE, TRUE)
```

Matrices

- A matrix is a two-dimensional array where all elements must be of the same type.
- Created using the `matrix()` function, specifying the number of rows and columns.

Example:

```
my_matrix <- matrix(1:6, nrow = 2, ncol = 3)
```

Lists

- A list can hold elements of different types, including vectors, matrices, and other lists.
- Created using the `list()` function.

Example:

```
my_list <- list(name = "Alice", age = 25, scores = c(90, 85, 88))
```

Data Frames

- A data frame is a tabular structure where each column can hold data of a different type.
- Created using the `data.frame()` function.

Example:

```
my_df <- data.frame(
    name = c("Alice", "Bob"),
    age = c(25, 30),
    scores = c(85, 90)
)
```

Tibbles

- A tibble is a modern variant of a data frame provided by the `tibble` package in the tidyverse.
- Tibbles are designed to be more user-friendly, particularly for interactive data analysis.
- Key differences from data frames:
  - Printing: Tibbles only show the first 10 rows and as many columns as fit on the screen.
  - Column names: Non-syntactic column names are allowed and do not throw errors.
  - Subsetting: Returns a tibble instead of a vector unless explicitly extracted.
- Created using the `tibble()` function or by converting an existing data frame with `as_tibble()`.

Examples:

```
library(tibble)

# Create a tibble
my_tibble <- tibble(
    name = c("Alice", "Bob"),
    age = c(25, 30),
    score = c(85, 90)
)

# Convert a data frame to a tibble
df <- data.frame(
    name = c("Charlie", "Diana"),
    age = c(35, 40),
    score = c(88, 92)
)
```

```
tibble_from_df <- as_tibble(df)

# Access columns
names <- my_tibble$name

# Print tibble
print(my_tibble)
```

---

2. Comprehensive Examples

Vector Operations

```
x <- c(1, 2, 3, 4)
y <- c(5, 6, 7, 8)

sum_xy <- x + y  # Adds corresponding elements
filtered_x <- x[x > 2]  # Filters values greater than 2
```

Matrix Operations

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

row_sum <- rowSums(mat)  # Sums each row
col_mean <- colMeans(mat)  # Calculates mean of each column
```

Accessing List Elements

```
my_list <- list(name = "John", scores = c(95, 80, 85))

# Access by name
name <- my_list$name
# Access by index
scores <- my_list[[2]]
```

Data Frame and Tibble Operations

```
# Data frame example
df <- data.frame(
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    score = c(85, 90, 95)
)

# Access column
names <- df$name
```

```
# Tibble example
library(tibble)
tb <- tibble(
    name = c("Alice", "Bob"),
    age = c(28, 35),
    score = c(88, 93)
)

# Access column
scores <- tb$score
```

3. Best Practices

1. Use meaningful names for vectors, matrices, lists, and data frames.
2. Ensure consistent data types within vectors and matrices.
3. Use `tibble` (from the tidyverse) for improved data frame functionality.
4. Avoid manual indexing where possible—use functions like `apply()` for better readability.
5. Prefer `dplyr` for complex data frame and tibble manipulations.
6. Document the purpose of lists to avoid confusion during nested list usage.
7. Check data types (`class()`) and structures (`str()`) before performing operations.
8. Use `rowSums()` and `colMeans()` for matrices instead of manual loops.
9. Avoid excessive nesting in lists; flatten them when feasible.
10. Validate data before analysis to handle missing or inconsistent values.

4. Practical Exercises

1. Create a numeric vector with values from 1 to 10 and calculate the sum of its elements.
2. Filter a vector of numbers to include only those greater than 5.
3. Create a 3x3 matrix filled with numbers from 1 to 9.
4. Calculate the row sums and column means of a matrix.
5. Create a list containing a name, age, and a vector of scores.
6. Access and modify an element within a list.
7. Create a data frame with columns for `name`, `age`, and `city`.
8. Filter a data frame to include rows where `age` is greater than 25.

9. Add a new column to a data frame that contains the square of the `age` column.
10. Combine two vectors into a single data frame.
11. Create a logical vector indicating which elements of a numeric vector are even.
12. Create a matrix and transpose it using the `t()` function.
13. Extract a submatrix (specific rows and columns) from a larger matrix.
14. Combine three lists into a single list and access elements at different nesting levels.