Practical No. 9

Title: Data Manipulation with `data.table` Package in R

Introduction:

In R, efficient data manipulation is key to handling large datasets and performing quick operations. The `data.table` package is a high-performance version of base R's `data.frame`, optimized for speed, especially with large datasets. It allows for concise syntax and blazing-fast operations such as subsetting, grouping, updating, and joining data.

Objective:

- To understand the structure and functionality of the `data.table` package.
- To learn how to perform key data manipulation tasks such as subsetting, filtering, aggregating, updating, and ordering data using `data.table`.
- To compare and appreciate the performance benefits of `data.table` over `data.frame`.

Tools Required:

- **Software:** R (version 4.0 or higher)
- **IDE (Recommended):** RStudio
- **Libraries:** `data.table`

Theory:

1. Introduction to `data.table`

The `data.table` package is an enhanced version of `data.frame`. It is fast and memory efficient, especially for big data applications. It uses a simple syntax:

```
DT[i, j, by]
```

Where:

- `i` = row operations (filtering)
- `j` = column operations (selecting, modifying)
- `by` = grouping operations (similar to SQL's GROUP BY)

2. Creating a `data.table`

```
library(data.table)

# Create sample data.table
employee_data <- data.table(
  ID = 1:10,
  Name = paste("Emp", 1:10),
```

```
  Age = sample(25:50, 10, replace = TRUE),
  Department = sample(c("HR", "Finance", "IT", "Sales"), 10, replace = TRUE),
  Salary = sample(seq(30000, 70000, by = 5000), 10, replace = TRUE)
)
```

3. Basic Operations with `data.table`

Operation	Syntax	Example
View Data	`employee_data[]`	`employee_data[]`
Select Columns	`employee_data[, .(Name, Salary)]`	Select Name and Salary columns
Filter Rows	`employee_data[Salary > 50000]`	Filter employees with salary > 50,000
Add New Column	`employee_data[, Bonus := Salary * 0.10]`	Add Bonus column as 10% of Salary
Update Column	`employee_data[Age > 40, Age_Group := "Senior"]`	Add conditional column
Sort Rows	`employee_data[order(-Salary)]`	Sort by Salary descending
Group & Summarise	`employee_data[, .(Avg_Salary = mean(Salary)), by = Department]`	Group by department and calculate average salary

4. Detailed Examples

a. Selecting Columns

```
employee_data[, .(Name, Salary)]
```

b. Filtering Rows

```
employee_data[Salary > 50000]
```

c. Adding or Updating Columns

```
employee_data[, Bonus := Salary * 0.10]
```

d. Conditional Column Creation

```
employee_data[, Age_Group := ifelse(Age < 35, "Young", "Experienced")]
```

e. Sorting Rows

```
employee_data[order(-Salary)]
```

f. Grouping and Summarising

```
employee_data[, .(Average_Salary = mean(Salary)), by = Department]
```

5. Advantages of `data.table`:

- **Speed:** Very fast with large datasets.

- **Memory Efficiency:** Performs operations in-place.
- **Concise Syntax:** Easy to chain filtering, selection, and aggregation.
- **SQL-Like Grouping:** Allows grouping and summarising in one line.
- **Built-in joins:** Like SQL joins (not covered here, but powerful).

Conclusion:

The `data.table` package in R is a powerful and efficient tool for data manipulation, particularly suitable for large-scale datasets. With its compact syntax and high performance, `data.table` is widely used in production-level data science and analytics projects. By learning how to filter, select, update, and summarize data using `data.table`, users can significantly boost the speed and efficiency of their R programming tasks