Practical No. 8

Title: Data Manipulation with `dplyr` Package in R

Introduction:

Data manipulation is an essential task in data analysis. The `dplyr` package in R provides a powerful, intuitive, and efficient set of tools for manipulating and transforming data. It simplifies the process of cleaning and restructuring data, which is crucial before performing any analysis or visualization. With functions like `filter()`, `select()`, `mutate()`, `summarise()`, and `arrange()`, `dplyr` enables users to quickly filter, sort, summarize, and transform datasets.

Objective:

- To understand the core functions in the `dplyr` package for data manipulation.
- To learn how to perform common operations like filtering, selecting, mutating, summarizing, and sorting data.
- To explore how to chain operations using the pipe operator (`%>%`) for cleaner, more readable code.

Tools Required:

- **Software:** R (version 4.0 or higher)
- **IDE (Recommended):** RStudio
- **Libraries:** `dplyr`, `tidyverse`

Theory:

1. Introduction to `dplyr`:

`dplyr` is a package within the `tidyverse` collection, and it's specifically designed for data manipulation. It provides a set of functions, each designed for a specific task, which makes it easier to perform common data manipulation tasks like selecting columns, filtering rows, adding or modifying columns, summarizing data, and sorting.

The key advantage of `dplyr` is its **simplicity and readability**, especially when used with the pipe operator (`%>%`), which allows for chaining multiple operations together.

2. Key Functions in `dplyr`:

Here are the core functions used in `dplyr` for data manipulation:

`select()` – Select Columns

The `select()` function is used to choose specific columns from a dataset. It is a fast and efficient way to subset your data.

Syntax:

```
select(data, column1, column2, ...)
```

Example:

```
# Select only Name and Salary columns
employee_data <- select(employee_data, Name, Salary)
```

You can also use helper functions to select columns by patterns:

- `starts_with("prefix")`
- `ends_with("suffix")`
- `contains("substring")`

```
# Select columns starting with 'Dept'
select(employee_data, starts_with("Dept"))
```

`filter()` – Filter Rows

The `filter()` function is used to filter rows based on certain conditions or logical statements.

Syntax:

```
filter(data, condition)
```

Example:

```
# Filter employees with Salary greater than 50,000
filtered_data <- filter(employee_data, Salary > 50000)
```

You can also combine multiple conditions using `&` (AND) or `|` (OR).

```
# Filter employees in IT department with Salary > 50,000
filtered_data <- filter(employee_data, Department == "IT" & Salary > 50000)
```

`mutate()` – Add or Modify Columns

The `mutate()` function allows you to add new columns or modify existing ones. This is useful for creating new variables derived from others.

Syntax:

```
mutate(data, new_column = expression)
```

Example:

```
# Add a new column for Bonus (10% of Salary)
employee_data <- mutate(employee_data, Bonus = Salary * 0.10)
```

`arrange()` – Sort Data

The `arrange()` function is used to reorder rows based on the values of one or more columns.

Syntax:

```
arrange(data, column)
```

Example:

```
# Arrange employees by Salary in descending order
arranged_data <- arrange(employee_data, desc(Salary))
```

`summarise()` – Summarize Data

The `summarise()` function is used to compute summary statistics for your data. It is often combined with `group_by()` to calculate statistics by groups.

Syntax:

```
summarise(data, new_column = summary_function(column))
```

Example:

```
# Calculate the average salary of all employees
avg_salary <- summarise(employee_data, Avg_Salary = mean(Salary))
```

`group_by()` – Group Data by Variables

The `group_by()` function is used to group data based on one or more variables, often in preparation for summary statistics.

Syntax:

```
group_by(data, column)
```

Example:

```
# Group data by Department and calculate average salary
avg_salary_by_dept <- employee_data %>%
  group_by(Department) %>%
  summarise(Average_Salary = mean(Salary))
```

3. Using the Pipe Operator (`%>%`)

The pipe operator (`%>%`) is a key feature of `dplyr` and the `tidyverse`. It allows you to chain multiple functions together, which makes your code more readable and efficient.

Syntax:

```
data %>% function1() %>% function2() %>% function3()
```

Example:

```
# Chain operations: Filter, select, and arrange
result <- employee_data %>%
  filter(Salary > 50000) %>%
  select(Name, Salary, Department) %>%
  arrange(desc(Salary))
```

Conclusion:

The `dplyr` package in R is a powerful tool for data manipulation. It provides an intuitive set of functions for selecting, filtering, modifying, summarizing, and sorting data, all of which are essential steps in data cleaning and analysis. These tools are essential for any data scientist or analyst working with R, enabling efficient data wrangling and preparation for analysis.