Complete Guide to dplyr Functions

Syed Toushik Hossain

2025-06-14

Contents

Introduction to dplyr

The dplyr package is one of the most essential tools in the R ecosystem for data manipulation. Think of it as your Swiss Army knife for working with data frames and tibbles. Each function in dplyr serves a specific purpose, much like different tools serve different functions in a workshop. Let's explore each function with practical examples using a sample dataset.

```r
# Creating a sample dataset for demonstration
# This represents sales data for a fictional company
sales_data <- data.frame(
  employee_id = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5),
  employee_name = c("Alice", "Bob", "Charlie", "Diana", "Eve",
                    "Frank", "Grace", "Henry", "Iris", "Jack",
                    "Alice", "Bob", "Charlie", "Diana", "Eve"),
  department = c("Sales", "Sales", "Marketing", "Sales", "Marketing",
                 "IT", "Sales", "Marketing", "IT", "Sales",
```

```
                "Sales", "Sales", "Marketing", "Sales", "Marketing"),
  region = c("North", "South", "North", "East", "West",
             "North", "South", "East", "West", "North",
             "North", "South", "North", "East", "West"),
  sales_amount = c(15000, 22000, 18000, 25000, 12000,
                   8000, 30000, 16000, 9000, 21000,
                   17000, 24000, 19000, 27000, 13000),
  sales_date = as.Date(c("2024-01-15", "2024-01-20", "2024-02-10", "2024-02-15", "2024-03-05",
                         "2024-03-10", "2024-03-20", "2024-04-05", "2024-04-10", "2024-04-15",
                         "2024-05-01", "2024-05-05", "2024-05-10", "2024-05-15", "2024-05-20")),
  commission_rate = c(0.05, 0.06, 0.04, 0.07, 0.03,
                      0.02, 0.08, 0.04, 0.02, 0.06,
                      0.05, 0.06, 0.04, 0.07, 0.03)
)

# Display the first few rows to understand our data structure
head(sales_data)
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate
## 1            0.05
## 2            0.06
## 3            0.04
## 4            0.07
## 5            0.03
## 6            0.02
```

1. filter() - Select Rows Based on Conditions

The `filter()` function is like a bouncer at a club - it only lets through the rows that meet your specific criteria. This is incredibly useful when you want to focus on a subset of your data that meets certain conditions.

```
# Basic filtering: Show only sales from the North region
# This helps us focus on regional performance analysis
north_sales <- sales_data %>%
  filter(region == "North")

cat("Sales from North region:\n")
```

```
## Sales from North region:
```

```
print(north_sales)
```

```
##   employee_id employee_name department region sales_amount sales_date
```

```
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           3       Charlie  Marketing  North        18000 2024-02-10
## 3           6         Frank         IT  North         8000 2024-03-10
## 4          10          Jack      Sales  North        21000 2024-04-15
## 5           1         Alice      Sales  North        17000 2024-05-01
## 6           3       Charlie  Marketing  North        19000 2024-05-10
##   commission_rate
## 1            0.05
## 2            0.04
## 3            0.02
## 4            0.06
## 5            0.05
## 6            0.04
```

```
# Multiple conditions: High-value sales (>20000) in Sales department
# This identifies top performers in our sales team
high_value_sales <- sales_data %>%
  filter(sales_amount > 20000 & department == "Sales")

cat("\nHigh-value sales in Sales department:\n")
```

```
##
## High-value sales in Sales department:
```

```
print(high_value_sales)
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           2           Bob      Sales  South        22000 2024-01-20
## 2           4         Diana      Sales   East        25000 2024-02-15
## 3           7         Grace      Sales  South        30000 2024-03-20
## 4          10          Jack      Sales  North        21000 2024-04-15
## 5           2           Bob      Sales  South        24000 2024-05-05
## 6           4         Diana      Sales   East        27000 2024-05-15
##   commission_rate
## 1            0.06
## 2            0.07
## 3            0.08
## 4            0.06
## 5            0.06
## 6            0.07
```

```
# Using %in% operator: Sales from specific regions
# This is efficient when checking multiple values
selected_regions <- sales_data %>%
  filter(region %in% c("North", "East"))

cat("\nSales from North and East regions:\n")
```

```
##
## Sales from North and East regions:
```

```r
print(selected_regions)
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           3       Charlie  Marketing  North        18000 2024-02-10
## 3           4         Diana      Sales   East        25000 2024-02-15
## 4           6         Frank         IT  North         8000 2024-03-10
## 5           8         Henry  Marketing   East        16000 2024-04-05
## 6          10          Jack      Sales  North        21000 2024-04-15
## 7           1         Alice      Sales  North        17000 2024-05-01
## 8           3       Charlie  Marketing  North        19000 2024-05-10
## 9           4         Diana      Sales   East        27000 2024-05-15
##   commission_rate
## 1            0.05
## 2            0.04
## 3            0.07
## 4            0.02
## 5            0.04
## 6            0.06
## 7            0.05
## 8            0.04
## 9            0.07
```

```r
# Date filtering: Sales after March 2024
# Useful for analyzing recent performance trends
recent_sales <- sales_data %>%
  filter(sales_date > as.Date("2024-03-01"))

cat("\nSales after March 2024:\n")
```

```
##
## Sales after March 2024:
```

```r
print(recent_sales)
```

```
##    employee_id employee_name department region sales_amount sales_date
## 1            5           Eve  Marketing   West        12000 2024-03-05
## 2            6         Frank         IT  North         8000 2024-03-10
## 3            7         Grace      Sales  South        30000 2024-03-20
## 4            8         Henry  Marketing   East        16000 2024-04-05
## 5            9          Iris         IT   West         9000 2024-04-10
## 6           10          Jack      Sales  North        21000 2024-04-15
## 7            1         Alice      Sales  North        17000 2024-05-01
## 8            2           Bob      Sales  South        24000 2024-05-05
## 9            3       Charlie  Marketing  North        19000 2024-05-10
## 10           4         Diana      Sales   East        27000 2024-05-15
## 11           5           Eve  Marketing   West        13000 2024-05-20
##    commission_rate
## 1             0.03
## 2             0.02
## 3             0.08
## 4             0.04
```

```
## 5             0.02
## 6             0.06
## 7             0.05
## 8             0.06
## 9             0.04
## 10            0.07
## 11            0.03
```

2. select() - Choose Columns by Name or Pattern

Think of `select()` as choosing which columns to display on your screen. Just like you might choose specific columns in a spreadsheet, this function helps you focus on the variables that matter for your current analysis.

```
# Select specific columns by name
# This creates a focused view for basic employee information
basic_info <- sales_data %>%
  select(employee_name, department, sales_amount)

cat("Basic employee information:\n")
```

```
## Basic employee information:
```

```
print(head(basic_info))
```

```
##   employee_name department sales_amount
## 1         Alice      Sales        15000
## 2           Bob      Sales        22000
## 3       Charlie  Marketing        18000
## 4         Diana      Sales        25000
## 5           Eve  Marketing        12000
## 6         Frank         IT         8000
```

```
# Select columns by range
# Useful when you want consecutive columns
column_range <- sales_data %>%
  select(employee_id:department)

cat("\nColumns from employee_id to department:\n")
```

```
##
## Columns from employee_id to department:
```

```
print(head(column_range))
```

```
##   employee_id employee_name department
## 1           1         Alice      Sales
## 2           2           Bob      Sales
## 3           3       Charlie  Marketing
## 4           4         Diana      Sales
## 5           5           Eve  Marketing
## 6           6         Frank         IT
```

```r
# Select columns by pattern
# This is powerful for datasets with many similarly named columns
sales_columns <- sales_data %>%
  select(contains("sales"))

cat("\nColumns containing 'sales':\n")
```

```
##
## Columns containing 'sales':
```

```r
print(head(sales_columns))
```

```
##   sales_amount sales_date
## 1        15000 2024-01-15
## 2        22000 2024-01-20
## 3        18000 2024-02-10
## 4        25000 2024-02-15
## 5        12000 2024-03-05
## 6         8000 2024-03-10
```

```r
# Exclude specific columns using negative selection
# This keeps everything except what you don't need
without_id <- sales_data %>%
  select(-employee_id, -sales_date)

cat("\nData without ID and date columns:\n")
```

```
##
## Data without ID and date columns:
```

```r
print(head(without_id))
```

```
##   employee_name department region sales_amount commission_rate
## 1         Alice      Sales  North        15000             0.05
## 2           Bob      Sales  South        22000             0.06
## 3       Charlie  Marketing  North        18000             0.04
## 4         Diana      Sales   East        25000             0.07
## 5           Eve  Marketing   West        12000             0.03
## 6         Frank         IT  North         8000             0.02
```

```r
# Advanced selection: starts_with, ends_with
# These are helpful for structured column names
employee_columns <- sales_data %>%
  select(starts_with("employee"))

cat("\nColumns starting with 'employee':\n")
```

```
##
## Columns starting with 'employee':
```

```r
print(head(employee_columns))
```

```
##   employee_id employee_name
## 1           1         Alice
## 2           2           Bob
## 3           3       Charlie
## 4           4         Diana
## 5           5           Eve
## 6           6         Frank
```

3. mutate() - Add or Change Columns

The `mutate()` function is like a calculator that can create new columns or modify existing ones. It's essential for creating derived variables or transforming your data.

```r
# Add a new column with calculations
# This creates a commission column based on sales amount and rate
sales_with_commission <- sales_data %>%
  mutate(commission = sales_amount * commission_rate)

cat("Data with calculated commission:\n")
```

```
## Data with calculated commission:
```

```r
print(head(sales_with_commission))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate commission
## 1            0.05        750
## 2            0.06       1320
## 3            0.04        720
## 4            0.07       1750
## 5            0.03        360
## 6            0.02        160
```

```r
# Multiple new columns in one mutate call
# This is efficient and keeps related calculations together
sales_enhanced <- sales_data %>%
  mutate(
    commission = sales_amount * commission_rate,
    net_sales = sales_amount - commission,
    sales_category = case_when(
      sales_amount < 15000 ~ "Low",
      sales_amount < 25000 ~ "Medium",
      TRUE ~ "High"
```

```
    )
  )

cat("\nData with multiple new columns:\n")
```

```
##
## Data with multiple new columns:
```

```
print(head(sales_enhanced))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate commission net_sales sales_category
## 1            0.05        750     14250         Medium
## 2            0.06       1320     20680         Medium
## 3            0.04        720     17280         Medium
## 4            0.07       1750     23250           High
## 5            0.03        360     11640            Low
## 6            0.02        160      7840            Low
```

```
# Modify existing columns
# This transforms data in place, useful for data cleaning
sales_modified <- sales_data %>%
  mutate(
    sales_amount = sales_amount / 1000,  # Convert to thousands
    employee_name = toupper(employee_name)  # Convert names to uppercase
  )

cat("\nData with modified columns:\n")
```

```
##
## Data with modified columns:
```

```
print(head(sales_modified))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         ALICE      Sales  North           15 2024-01-15
## 2           2           BOB      Sales  South           22 2024-01-20
## 3           3       CHARLIE  Marketing  North           18 2024-02-10
## 4           4         DIANA      Sales   East           25 2024-02-15
## 5           5           EVE  Marketing   West           12 2024-03-05
## 6           6         FRANK         IT  North            8 2024-03-10
##   commission_rate
## 1            0.05
## 2            0.06
## 3            0.04
```

```
## 4            0.07
## 5            0.03
## 6            0.02
```

```r
# Using mutate with conditional logic
# This creates categories based on complex conditions
sales_categorized <- sales_data %>%
  mutate(
    performance_tier = case_when(
      sales_amount > 25000 ~ "Excellent",
      sales_amount > 20000 ~ "Good",
      sales_amount > 15000 ~ "Average",
      TRUE ~ "Needs Improvement"
    )
  )

cat("\nData with performance tiers:\n")
```

```
##
## Data with performance tiers:
```

```r
print(head(sales_categorized))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate  performance_tier
## 1            0.05 Needs Improvement
## 2            0.06             Good
## 3            0.04          Average
## 4            0.07             Good
## 5            0.03 Needs Improvement
## 6            0.02 Needs Improvement
```

4. summarise() - Collapse Rows to Summary Statistics

The `summarise()` function is like taking a step back to see the big picture. It collapses many rows into summary statistics, helping you understand patterns and trends in your data.

```r
# Basic summary statistics
# This gives us an overview of our sales performance
sales_summary <- sales_data %>%
  summarise(
    total_sales = sum(sales_amount),
    average_sales = mean(sales_amount),
    median_sales = median(sales_amount),
    min_sales = min(sales_amount),
    max_sales = max(sales_amount),
```

```
    count_records = n()
  )

cat("Overall sales summary:\n")
```

```
## Overall sales summary:
```

```
print(sales_summary)
```

```
##   total_sales average_sales median_sales min_sales max_sales count_records
## 1      276000         18400        18000      8000     30000            15
```

```
# Calculate multiple percentiles
# This helps understand the distribution of sales amounts
sales_percentiles <- sales_data %>%
  summarise(
    q25 = quantile(sales_amount, 0.25),
    q50 = quantile(sales_amount, 0.50),
    q75 = quantile(sales_amount, 0.75),
    iqr = IQR(sales_amount)
  )

cat("\nSales percentiles:\n")
```

```
##
## Sales percentiles:
```

```
print(sales_percentiles)
```

```
##     q25   q50   q75  iqr
## 1 14000 18000 23000 9000
```

```
# Summary with additional calculations
# This provides business-relevant metrics
business_metrics <- sales_data %>%
  summarise(
    total_revenue = sum(sales_amount),
    total_commission = sum(sales_amount * commission_rate),
    net_revenue = sum(sales_amount - (sales_amount * commission_rate)),
    average_commission_rate = mean(commission_rate),
    number_of_transactions = n(),
    unique_employees = n_distinct(employee_id)
  )

cat("\nBusiness metrics summary:\n")
```

```
##
## Business metrics summary:
```

```r
print(business_metrics)
```

```
##   total_revenue total_commission net_revenue average_commission_rate
## 1        276000            14870      261130                   0.048
##   number_of_transactions unique_employees
## 1                     15               10
```

5. group_by() - Group Data for Grouped Operations

The `group_by()` function is like organizing your data into separate piles before performing calculations. It's incredibly powerful when combined with `summarise()` because it allows you to calculate statistics for each group separately.

```r
# Group by department and calculate summary statistics
# This helps us compare performance across different departments
dept_summary <- sales_data %>%
  group_by(department) %>%
  summarise(
    total_sales = sum(sales_amount),
    average_sales = mean(sales_amount),
    employee_count = n_distinct(employee_id),
    transaction_count = n(),
    .groups = 'drop'  # This ungrouped the result
  )

cat("Summary by department:\n")
```

```
## Summary by department:
```

```r
print(dept_summary)
```

```
## # A tibble: 3 x 5
##   department total_sales average_sales employee_count transaction_count
##   <chr>            <dbl>         <dbl>          <int>             <int>
## 1 IT               17000          8500              2                 2
## 2 Marketing        78000         15600              3                 5
## 3 Sales           181000         22625              5                 8
```

```r
# Group by multiple variables
# This provides a more detailed breakdown
region_dept_summary <- sales_data %>%
  group_by(region, department) %>%
  summarise(
    total_sales = sum(sales_amount),
    avg_sales = mean(sales_amount),
    transactions = n(),
    .groups = 'drop'
  )

cat("\nSummary by region and department:\n")
```

```
##
## Summary by region and department:
```

```
print(region_dept_summary)
```

```
## # A tibble: 8 x 5
##   region department total_sales avg_sales transactions
##   <chr>  <chr>            <dbl>     <dbl>        <int>
## 1 East   Marketing        16000    16000             1
## 2 East   Sales            52000    26000             2
## 3 North  IT                8000     8000             1
## 4 North  Marketing        37000    18500             2
## 5 North  Sales            53000    17667.            3
## 6 South  Sales            76000    25333.            3
## 7 West   IT                9000     9000             1
## 8 West   Marketing        25000    12500             2
```

```
# Using group_by with mutate to create group-specific calculations
# This adds group statistics to each row while keeping all data
sales_with_group_stats <- sales_data %>%
  group_by(department) %>%
  mutate(
    dept_avg_sales = mean(sales_amount),
    dept_total_sales = sum(sales_amount),
    sales_vs_dept_avg = sales_amount - dept_avg_sales
  ) %>%
  ungroup()  # Always ungroup when done to avoid unexpected behavior

cat("\nData with department-level statistics:\n")
```

```
##
## Data with department-level statistics:
```

```
print(head(sales_with_group_stats))
```

```
## # A tibble: 6 x 10
##   employee_id employee_name department region sales_amount sales_date
##         <dbl> <chr>         <chr>      <chr>         <dbl> <date>
## 1           1 Alice         Sales      North         15000 2024-01-15
## 2           2 Bob           Sales      South         22000 2024-01-20
## 3           3 Charlie       Marketing  North         18000 2024-02-10
## 4           4 Diana         Sales      East          25000 2024-02-15
## 5           5 Eve           Marketing  West          12000 2024-03-05
## 6           6 Frank         IT         North          8000 2024-03-10
## # i 4 more variables: commission_rate <dbl>, dept_avg_sales <dbl>,
## #   dept_total_sales <dbl>, sales_vs_dept_avg <dbl>
```

6. arrange() - Sort Rows by Column Values

The `arrange()` function organizes your data in a specific order, much like sorting files in a filing cabinet. This is essential for identifying top performers, trends, or outliers.

```r
# Sort by sales amount in ascending order
# This helps identify the lowest performing sales
sales_ascending <- sales_data %>%
  arrange(sales_amount)

cat("Sales data sorted by amount (ascending):\n")
```

```
## Sales data sorted by amount (ascending):
```

```r
print(head(sales_ascending))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           6         Frank         IT  North         8000 2024-03-10
## 2           9          Iris         IT   West         9000 2024-04-10
## 3           5           Eve  Marketing   West        12000 2024-03-05
## 4           5           Eve  Marketing   West        13000 2024-05-20
## 5           1         Alice      Sales  North        15000 2024-01-15
## 6           8         Henry  Marketing   East        16000 2024-04-05
##   commission_rate
## 1            0.02
## 2            0.02
## 3            0.03
## 4            0.03
## 5            0.05
## 6            0.04
```

```r
# Sort by sales amount in descending order
# This identifies top performers
sales_descending <- sales_data %>%
  arrange(desc(sales_amount))

cat("\nTop sales performers:\n")
```

```
##
## Top sales performers:
```

```r
print(head(sales_descending))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           7         Grace      Sales  South        30000 2024-03-20
## 2           4         Diana      Sales   East        27000 2024-05-15
## 3           4         Diana      Sales   East        25000 2024-02-15
## 4           2           Bob      Sales  South        24000 2024-05-05
## 5           2           Bob      Sales  South        22000 2024-01-20
## 6          10          Jack      Sales  North        21000 2024-04-15
##   commission_rate
## 1            0.08
## 2            0.07
## 3            0.07
## 4            0.06
## 5            0.06
## 6            0.06
```

```r
# Sort by multiple columns
# This creates a hierarchical sorting system
sales_multi_sort <- sales_data %>%
  arrange(department, desc(sales_amount))

cat("\nSales sorted by department, then by amount (descending):\n")
```

```
##
## Sales sorted by department, then by amount (descending):
```

```r
print(head(sales_multi_sort, 10))
```

```
##    employee_id employee_name department region sales_amount sales_date
## 1            9          Iris         IT   West         9000 2024-04-10
## 2            6         Frank         IT  North         8000 2024-03-10
## 3            3       Charlie  Marketing  North        19000 2024-05-10
## 4            3       Charlie  Marketing  North        18000 2024-02-10
## 5            8         Henry  Marketing   East        16000 2024-04-05
## 6            5           Eve  Marketing   West        13000 2024-05-20
## 7            5           Eve  Marketing   West        12000 2024-03-05
## 8            7         Grace      Sales  South        30000 2024-03-20
## 9            4         Diana      Sales   East        27000 2024-05-15
## 10           4         Diana      Sales   East        25000 2024-02-15
##    commission_rate
## 1             0.02
## 2             0.02
## 3             0.04
## 4             0.04
## 5             0.04
## 6             0.03
## 7             0.03
## 8             0.08
## 9             0.07
## 10            0.07
```

```r
# Sort by date to see chronological order
# This helps identify trends over time
sales_chronological <- sales_data %>%
  arrange(sales_date)

cat("\nSales in chronological order:\n")
```

```
##
## Sales in chronological order:
```

```r
print(head(sales_chronological))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
```

```
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate
## 1            0.05
## 2            0.06
## 3            0.04
## 4            0.07
## 5            0.03
## 6            0.02
```

7. rename() - Rename Columns

The `rename()` function is like putting new labels on your file folders. It helps make your column names more descriptive or consistent with naming conventions.

```
# Rename columns for better clarity
# This makes the dataset more self-explanatory
sales_renamed <- sales_data %>%
  rename(
    emp_id = employee_id,
    emp_name = employee_name,
    dept = department,
    sales_revenue = sales_amount,
    commission_pct = commission_rate
  )

cat("Data with renamed columns:\n")
```

```
## Data with renamed columns:
```

```
print(head(sales_renamed))
```

```
##   emp_id emp_name      dept region sales_revenue sales_date commission_pct
## 1      1    Alice     Sales  North         15000 2024-01-15           0.05
## 2      2      Bob     Sales  South         22000 2024-01-20           0.06
## 3      3  Charlie Marketing  North         18000 2024-02-10           0.04
## 4      4    Diana     Sales   East         25000 2024-02-15           0.07
## 5      5      Eve Marketing   West         12000 2024-03-05           0.03
## 6      6    Frank        IT  North          8000 2024-03-10           0.02
```

```
# Rename multiple columns systematically
# This ensures consistent naming patterns
sales_consistent_names <- sales_data %>%
  rename(
    EmployeeID = employee_id,
    EmployeeName = employee_name,
    Department = department,
    Region = region,
    SalesAmount = sales_amount,
    SalesDate = sales_date,
    CommissionRate = commission_rate
```

```
  )

cat("\nData with consistent naming convention:\n")
```

```
##
## Data with consistent naming convention:
```

```
print(head(sales_consistent_names))
```

```
##   EmployeeID EmployeeName Department Region SalesAmount  SalesDate
## 1          1        Alice      Sales  North       15000 2024-01-15
## 2          2          Bob      Sales  South       22000 2024-01-20
## 3          3      Charlie  Marketing  North       18000 2024-02-10
## 4          4        Diana      Sales   East       25000 2024-02-15
## 5          5          Eve  Marketing   West       12000 2024-03-05
## 6          6        Frank         IT  North        8000 2024-03-10
##   CommissionRate
## 1           0.05
## 2           0.06
## 3           0.04
## 4           0.07
## 5           0.03
## 6           0.02
```

```
# Using rename with select for column reordering and renaming
# This combines column selection with renaming
focused_data <- sales_data %>%
  select(
    Name = employee_name,
    Dept = department,
    Revenue = sales_amount,
    Commission = commission_rate
  )

cat("\nFocused data with renamed columns:\n")
```

```
##
## Focused data with renamed columns:
```

```
print(head(focused_data))
```

```
##      Name      Dept Revenue Commission
## 1   Alice     Sales   15000       0.05
## 2     Bob     Sales   22000       0.06
## 3 Charlie Marketing   18000       0.04
## 4   Diana     Sales   25000       0.07
## 5     Eve Marketing   12000       0.03
## 6   Frank        IT    8000       0.02
```

8. distinct() - Remove Duplicate Rows

The `distinct()` function is like a duplicate detector that keeps only unique records. This is crucial for data cleaning and ensuring accurate analysis.

```
# Get unique employees
# This helps us understand how many unique individuals we have
unique_employees <- sales_data %>%
  distinct(employee_id, employee_name)

cat("Unique employees:\n")
```

```
## Unique employees:
```

```
print(unique_employees)
```

```
##    employee_id employee_name
## 1            1         Alice
## 2            2           Bob
## 3            3       Charlie
## 4            4         Diana
## 5            5           Eve
## 6            6         Frank
## 7            7         Grace
## 8            8         Henry
## 9            9          Iris
## 10          10          Jack
```

```
# Get unique department-region combinations
# This shows us all the department-region pairs in our data
unique_combinations <- sales_data %>%
  distinct(department, region)

cat("\nUnique department-region combinations:\n")
```

```
##
## Unique department-region combinations:
```

```
print(unique_combinations)
```

```
##   department region
## 1      Sales  North
## 2      Sales  South
## 3  Marketing  North
## 4      Sales   East
## 5  Marketing   West
## 6         IT  North
## 7  Marketing   East
## 8         IT   West
```

```r
# Remove complete duplicate rows
# This ensures each transaction is counted only once
unique_records <- sales_data %>%
  distinct()

cat("\nNumber of unique records:", nrow(unique_records), "\n")
```

```
##
## Number of unique records: 15
```

```r
cat("Original number of records:", nrow(sales_data), "\n")
```

```
## Original number of records: 15
```

```r
# Keep first occurrence of duplicates based on specific columns
# This is useful when you want to keep the first occurrence of each employee
first_occurrence <- sales_data %>%
  distinct(employee_id, .keep_all = TRUE)

cat("\nFirst occurrence of each employee:\n")
```

```
##
## First occurrence of each employee:
```

```r
print(first_occurrence)
```

```
##    employee_id employee_name department region sales_amount sales_date
## 1            1         Alice      Sales  North        15000 2024-01-15
## 2            2           Bob      Sales  South        22000 2024-01-20
## 3            3       Charlie  Marketing  North        18000 2024-02-10
## 4            4         Diana      Sales   East        25000 2024-02-15
## 5            5           Eve  Marketing   West        12000 2024-03-05
## 6            6         Frank         IT  North         8000 2024-03-10
## 7            7         Grace      Sales  South        30000 2024-03-20
## 8            8         Henry  Marketing   East        16000 2024-04-05
## 9            9          Iris         IT   West         9000 2024-04-10
## 10          10          Jack      Sales  North        21000 2024-04-15
##    commission_rate
## 1             0.05
## 2             0.06
## 3             0.04
## 4             0.07
## 5             0.03
## 6             0.02
## 7             0.08
## 8             0.04
## 9             0.02
## 10            0.06
```

9. count() - Count Occurrences of Values

The `count()` function is like a tally counter that helps you understand the frequency of different values in your data. It's essential for exploratory data analysis.

```
# Count occurrences of each department
# This shows the distribution of records across departments
dept_counts <- sales_data %>%
  count(department)

cat("Count by department:\n")
```

```
## Count by department:
```

```
print(dept_counts)
```

```
##   department n
## 1         IT 2
## 2  Marketing 5
## 3      Sales 8
```

```
# Count with sorting
# This shows the most common values first
dept_counts_sorted <- sales_data %>%
  count(department, sort = TRUE)

cat("\nDepartment counts (sorted):\n")
```

```
##
## Department counts (sorted):
```

```
print(dept_counts_sorted)
```

```
##   department n
## 1      Sales 8
## 2  Marketing 5
## 3         IT 2
```

```
# Count by multiple variables
# This provides a cross-tabulation of department and region
cross_count <- sales_data %>%
  count(department, region, sort = TRUE)

cat("\nCross-tabulation of department and region:\n")
```

```
##
## Cross-tabulation of department and region:
```

```r
print(cross_count)
```

```
##   department region n
## 1      Sales  North 3
## 2      Sales  South 3
## 3  Marketing  North 2
## 4  Marketing   West 2
## 5      Sales   East 2
## 6         IT  North 1
## 7         IT   West 1
## 8  Marketing   East 1
```

```r
# Count with additional calculations
# This adds percentages to understand proportions
dept_proportions <- sales_data %>%
  count(department) %>%
  mutate(
    percentage = round(n / sum(n) * 100, 1),
    proportion = n / sum(n)
  )

cat("\nDepartment counts with percentages:\n")
```

```
##
## Department counts with percentages:
```

```r
print(dept_proportions)
```

```
##   department n percentage proportion
## 1         IT 2       13.3  0.1333333
## 2  Marketing 5       33.3  0.3333333
## 3      Sales 8       53.3  0.5333333
```

```r
# Count unique values in a column
# This is equivalent to n_distinct()
unique_count <- sales_data %>%
  summarise(
    unique_employees = n_distinct(employee_id),
    unique_departments = n_distinct(department),
    unique_regions = n_distinct(region)
  )

cat("\nCount of unique values:\n")
```

```
##
## Count of unique values:
```

```r
print(unique_count)
```

```
##   unique_employees unique_departments unique_regions
## 1               10                 3              4
```

10. across() - Apply Functions Across Multiple Columns

The `across()` function is like having a magic wand that can apply the same operation to multiple columns at once. This is incredibly efficient for data cleaning and transformation tasks.

```
# Apply summary functions across multiple numeric columns
# This gives us summary statistics for all numeric variables
numeric_summary <- sales_data %>%
  summarise(across(where(is.numeric), list(
    mean = mean,
    median = median,
    sd = sd,
    min = min,
    max = max
  ), .names = "{.col}_{.fn}"))

cat("Summary statistics across numeric columns:\n")
```

```
## Summary statistics across numeric columns:
```

```
print(as.data.frame(numeric_summary))
```

```
##   employee_id_mean employee_id_median employee_id_sd employee_id_min
## 1         4.666667                  4       2.845213               1
##   employee_id_max sales_amount_mean sales_amount_median sales_amount_sd
## 1              10             18400                18000        6489.552
##   sales_amount_min sales_amount_max commission_rate_mean commission_rate_median
## 1             8000            30000                0.048                   0.05
##   commission_rate_sd commission_rate_min commission_rate_max
## 1         0.01859339                0.02                0.08
```

```
# Apply transformations to specific columns
# This standardizes multiple columns at once
sales_standardized <- sales_data %>%
  mutate(across(c(sales_amount, commission_rate), scale))

cat("\nFirst few rows of standardized data:\n")
```

```
##
## First few rows of standardized data:
```

```
print(head(sales_standardized))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North  -0.52391906 2024-01-15
## 2           2           Bob      Sales  South   0.55473783 2024-01-20
## 3           3       Charlie  Marketing  North  -0.06163754 2024-02-10
## 4           4         Diana      Sales   East   1.01701935 2024-02-15
## 5           5           Eve  Marketing   West  -0.98620058 2024-03-05
## 6           6         Frank         IT  North  -1.60257595 2024-03-10
##   commission_rate
```

```
## 1       0.1075651
## 2       0.6453905
## 3      -0.4302603
## 4       1.1832160
## 5      -0.9680858
## 6      -1.5059112
```

```
# Apply functions to columns matching a pattern
# This is useful for datasets with many similarly named columns
sales_rounded <- sales_data %>%
  mutate(across(contains("sales"), round, digits = 0))

cat("\nData with rounded sales values:\n")
```

```
##
## Data with rounded sales values:
```

```
print(head(sales_rounded))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate
## 1            0.05
## 2            0.06
## 3            0.04
## 4            0.07
## 5            0.03
## 6            0.02
```

```
# Apply different functions to different column groups
# This shows the flexibility of across()
sales_transformed <- sales_data %>%
  mutate(
    across(where(is.character), toupper),  # Convert text to uppercase
    across(where(is.numeric), round, digits = 2)  # Round numeric values
  )

cat("\nData with multiple transformations:\n")
```

```
##
## Data with multiple transformations:
```

```
print(head(sales_transformed))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         ALICE      SALES  NORTH        15000 2024-01-15
```

```
## 2           2           BOB      SALES  SOUTH        22000 2024-01-20
## 3           3       CHARLIE  MARKETING  NORTH        18000 2024-02-10
## 4           4         DIANA      SALES   EAST        25000 2024-02-15
## 5           5           EVE  MARKETING   WEST        12000 2024-03-05
## 6           6         FRANK         IT  NORTH         8000 2024-03-10
##   commission_rate
## 1            0.05
## 2            0.06
## 3            0.04
## 4            0.07
## 5            0.03
## 6            0.02
```

```
# Using across() with group_by() for grouped operations
# This calculates statistics for each group across multiple columns
grouped_summary <- sales_data %>%
  group_by(department) %>%
  summarise(across(c(sales_amount, commission_rate),
                   list(mean = mean, sd = sd),
                   .names = "{.col}_{.fn}"),
            .groups = 'drop')

cat("\nGrouped summary across multiple columns:\n")
```

```
##
## Grouped summary across multiple columns:
```

```
print(grouped_summary)
```

```
## # A tibble: 3 x 5
##   department sales_amount_mean sales_amount_sd commission_rate_mean
##   <chr>                  <dbl>           <dbl>                <dbl>
## 1 IT                      8500            707.                0.02
## 2 Marketing              15600           3050.                0.036
## 3 Sales                  22625           4984.                0.0625
## # i 1 more variable: commission_rate_sd <dbl>
```

11. case_when() - Vectorised if...else if...else

The `case_when()` function is like a sophisticated decision tree that can handle multiple conditions elegantly. It's much cleaner than nested if-else statements and is perfect for creating categorical variables.

```
# Create performance categories based on sales amount
# This helps classify employees into performance tiers
sales_performance <- sales_data %>%
  mutate(
    performance_level = case_when(
      sales_amount >= 25000 ~ "Outstanding",
      sales_amount >= 20000 ~ "Excellent",
      sales_amount >= 15000 ~ "Good",
      sales_amount >= 10000 ~ "Average",
```

```
      TRUE ~ "Needs Improvement"  # TRUE serves as the 'else' condition
    )
  )

cat("Sales data with performance levels:\n")
```

```
## Sales data with performance levels:
```

```
print(head(sales_performance))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate performance_level
## 1            0.05              Good
## 2            0.06         Excellent
## 3            0.04              Good
## 4            0.07       Outstanding
## 5            0.03           Average
## 6            0.02 Needs Improvement
```

```
# Complex conditions with multiple variables
# This creates sophisticated business rules
sales_categorized <- sales_data %>%
  mutate(
    sales_tier = case_when(
      department == "Sales" & sales_amount > 20000 ~ "Top Sales Performer",
      department == "Marketing" & sales_amount > 15000 ~ "Top Marketing Performer",
      department == "IT" & sales_amount > 8000 ~ "Top IT Performer",
      sales_amount > 18000 ~ "High Performer",
      sales_amount > 12000 ~ "Average Performer",
      TRUE ~ "Low Performer"
    )
  )

cat("\nSales data with complex categorization:\n")
```

```
##
## Sales data with complex categorization:
```

```
print(head(sales_categorized, 10))
```

```
##    employee_id employee_name department region sales_amount sales_date
## 1            1         Alice      Sales  North        15000 2024-01-15
## 2            2           Bob      Sales  South        22000 2024-01-20
## 3            3       Charlie  Marketing  North        18000 2024-02-10
## 4            4         Diana      Sales   East        25000 2024-02-15
```

```
## 5            5           Eve  Marketing   West        12000 2024-03-05
## 6            6         Frank         IT  North         8000 2024-03-10
## 7            7         Grace      Sales  South        30000 2024-03-20
## 8            8         Henry  Marketing   East        16000 2024-04-05
## 9            9          Iris         IT   West         9000 2024-04-10
## 10          10          Jack      Sales  North        21000 2024-04-15
##    commission_rate             sales_tier
## 1             0.05       Average Performer
## 2             0.06     Top Sales Performer
## 3             0.04 Top Marketing Performer
## 4             0.07     Top Sales Performer
## 5             0.03          Low Performer
## 6             0.02          Low Performer
## 7             0.08     Top Sales Performer
## 8             0.04 Top Marketing Performer
## 9             0.02        Top IT Performer
## 10            0.06     Top Sales Performer
```

```
# Using case_when with date conditions
# This creates time-based categories
sales_quarterly <- sales_data %>%
  mutate(
    quarter = case_when(
      sales_date >= as.Date("2024-01-01") & sales_date < as.Date("2024-04-01") ~ "Q1",
      sales_date >= as.Date("2024-04-01") & sales_date < as.Date("2024-07-01") ~ "Q2",
      sales_date >= as.Date("2024-07-01") & sales_date < as.Date("2024-10-01") ~ "Q3",
      TRUE ~ "Q4"
    )
  )

cat("\nSales data with quarterly classification:\n")
```

```
##
## Sales data with quarterly classification:
```

```
print(head(sales_quarterly))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate quarter
## 1            0.05      Q1
## 2            0.06      Q1
## 3            0.04      Q1
## 4            0.07      Q1
## 5            0.03      Q1
## 6            0.02      Q1
```

```r
# Using case_when for data cleaning
# This handles missing or problematic values
sales_cleaned <- sales_data %>%
  mutate(
    region_clean = case_when(
      is.na(region) ~ "Unknown",
      region == "" ~ "Unknown",
      region %in% c("North", "South", "East", "West") ~ region,
      TRUE ~ "Other"
    ),
    commission_category = case_when(
      commission_rate < 0.03 ~ "Low Commission",
      commission_rate < 0.06 ~ "Standard Commission",
      commission_rate >= 0.06 ~ "High Commission",
      is.na(commission_rate) ~ "No Commission Data",
      TRUE ~ "Other"
    )
  )

cat("\nCleaned data with case_when:\n")
```

```
##
## Cleaned data with case_when:
```

```r
print(head(sales_cleaned))
```

```
##   employee_id employee_name department region sales_amount sales_date
## 1           1         Alice      Sales  North        15000 2024-01-15
## 2           2           Bob      Sales  South        22000 2024-01-20
## 3           3       Charlie  Marketing  North        18000 2024-02-10
## 4           4         Diana      Sales   East        25000 2024-02-15
## 5           5           Eve  Marketing   West        12000 2024-03-05
## 6           6         Frank         IT  North         8000 2024-03-10
##   commission_rate region_clean commission_category
## 1            0.05        North Standard Commission
## 2            0.06        South     High Commission
## 3            0.04        North Standard Commission
## 4            0.07         East     High Commission
## 5            0.03         West Standard Commission
## 6            0.02        North      Low Commission
```

Combining dplyr Functions: The Power of the Pipe

The real magic of dplyr comes from combining these functions using the pipe operator (%>%). This allows you to create powerful data analysis pipelines that are both readable and efficient.

```r
# Complex analysis combining multiple dplyr functions
# This creates a comprehensive sales analysis pipeline
sales_analysis <- sales_data %>%
  # Step 1: Add calculated columns
  mutate(
```

```
    commission = sales_amount * commission_rate,
    net_sales = sales_amount - commission,
    performance_tier = case_when(
      sales_amount > 25000 ~ "Top",
      sales_amount > 20000 ~ "High",
      sales_amount > 15000 ~ "Medium",
      TRUE ~ "Low"
    )
  ) %>%
  # Step 2: Filter for relevant data
  filter(sales_amount > 10000) %>%
  # Step 3: Group by relevant categories
  group_by(department, performance_tier) %>%
  # Step 4: Calculate summary statistics
  summarise(
    total_sales = sum(sales_amount),
    avg_sales = mean(sales_amount),
    total_commission = sum(commission),
    employee_count = n_distinct(employee_id),
    transaction_count = n(),
    .groups = 'drop'
  ) %>%
  # Step 5: Sort results
  arrange(department, desc(total_sales)) %>%
  # Step 6: Add percentage calculations
  mutate(
    pct_of_total = round(total_sales / sum(total_sales) * 100, 2)
  )

cat("Comprehensive sales analysis:\n")
```

```
## Comprehensive sales analysis:
```

```
print(sales_analysis)
```

```
## # A tibble: 6 x 8
##   department performance_tier total_sales avg_sales total_commission
##   <chr>      <chr>                  <dbl>     <dbl>            <dbl>
## 1 Marketing  Medium                 53000    17667.             2120
## 2 Marketing  Low                    25000    12500               750
## 3 Sales      High                   92000    23000              5770
## 4 Sales      Top                    57000    28500              4290
## 5 Sales      Medium                 17000    17000               850
## 6 Sales      Low                    15000    15000               750
## # i 3 more variables: employee_count <int>, transaction_count <int>,
## #   pct_of_total <dbl>
```

```
# Top performing employees analysis
# This identifies and ranks top performers with detailed metrics
top_performers <- sales_data %>%
  # Add commission calculations
  mutate(commission = sales_amount * commission_rate) %>%
```

```r
  # Group by employee to get their totals
  group_by(employee_id, employee_name, department) %>%
  summarise(
    total_sales = sum(sales_amount),
    total_commission = sum(commission),
    avg_sale = mean(sales_amount),
    transaction_count = n(),
    .groups = 'drop'
  ) %>%
  # Filter for employees with significant sales
  filter(total_sales > 20000) %>%
  # Rank employees
  arrange(desc(total_sales)) %>%
  # Add rankings
  mutate(
    sales_rank = row_number(),
    performance_category = case_when(
      sales_rank <= 3 ~ "Top 3",
      sales_rank <= 5 ~ "Top 5",
      TRUE ~ "Other Top Performers"
    )
  ) %>%
  # Select and rename columns for final report
  select(
    Rank = sales_rank,
    Name = employee_name,
    Department = department,
    `Total Sales` = total_sales,
    `Avg Sale` = avg_sale,
    `Total Commission` = total_commission,
    Transactions = transaction_count,
    Category = performance_category
  )

cat("\nTop performers analysis:\n")
```

```
##
## Top performers analysis:
```

```r
print(top_performers)
```

```
## # A tibble: 7 x 8
##    Rank Name    Department `Total Sales` `Avg Sale` `Total Commission`
##   <int> <chr>   <chr>              <dbl>      <dbl>              <dbl>
## 1     1 Diana   Sales              52000      26000               3640
## 2     2 Bob     Sales              46000      23000               2760
## 3     3 Charlie Marketing          37000      18500               1480
## 4     4 Alice   Sales              32000      16000               1600
## 5     5 Grace   Sales              30000      30000               2400
## 6     6 Eve     Marketing          25000      12500                750
## 7     7 Jack    Sales              21000      21000               1260
## # i 2 more variables: Transactions <int>, Category <chr>
```

Conclusion

The dplyr functions work together like a well-orchestrated team, each serving a specific purpose in the data manipulation process. Understanding when and how to use each function is key to becoming proficient in R data analysis. The beauty of dplyr lies not just in individual functions, but in how they can be combined using the pipe operator to create powerful, readable data analysis workflows.

Remember that practice makes perfect. Try applying these functions to your own datasets, and don't be afraid to experiment with different combinations. The more you use dplyr, the more intuitive these operations will become, and you'll find yourself thinking in terms of data transformation pipelines rather than individual operations.

Each function serves as a building block in your data analysis toolkit. Just as a carpenter uses different tools for different purposes, you'll use different dplyr functions depending on what you need to accomplish with your data. The key is understanding what each tool does and when to use it effectively.