Here’s the logical order of using key dplyr functions — filter(), select(), mutate(), summarise(), and group\_by() — along with ten simple examples to make it easy to understand.

**Order of Functions:**

1. **filter()**: Subset the rows based on certain conditions.
2. **select()**: Pick specific columns to keep.
3. **mutate()**: Create or transform variables.
4. **group\_by()**: Group the data by one or more columns to perform operations by group.
5. **summarise()**: Aggregate data to compute summary statistics, often used with group\_by().

**Examples**

**Example 1: Filtering Rows**

Filter the rows where mpg (miles per gallon) is greater than 20 from the mtcars dataset.

r

mtcars %>%

filter(mpg > 20)

This selects cars with an mpg value greater than 20.

**Example 2: Selecting Columns**

Select only the mpg, cyl, and gear columns.

r

mtcars %>%

select(mpg, cyl, gear)

This keeps only the mpg, cyl, and gear columns in the output.

**Example 3: Combining filter() and select()**

Filter cars with more than 4 cylinders and select only the mpg and hp (horsepower) columns.

r

mtcars %>%

filter(cyl > 4) %>%

select(mpg, hp)

This filters the data and then keeps only two columns.

**Example 4: Using mutate() to Add New Columns**

Create a new column wt\_kg by converting the wt (weight) column from 1000 lbs to kilograms.

r

mtcars %>%

mutate(wt\_kg = wt \* 0.453592)

This adds a new column wt\_kg by converting weight from pounds to kilograms.

**Example 5: Combining mutate() and select()**

Create a new column hp\_per\_wt for horsepower per unit weight and then select only mpg, hp, and the new column.

r

mtcars %>%

mutate(hp\_per\_wt = hp / wt) %>%

select(mpg, hp, hp\_per\_wt)

This creates the new column and keeps only the necessary columns.

**Example 6: Grouping and Summarising Data**

Group the cars by the number of cylinders and compute the average horsepower for each group.

r

mtcars %>%

group\_by(cyl) %>%

summarise(avg\_hp = mean(hp))

This groups the cars by the number of cylinders and calculates the average horsepower for each group.

**Example 7: Filtering, Mutating, and Grouping**

Filter cars with an mpg greater than 20, create a new column for weight in kilograms, and then group by the number of cylinders to find the average weight in kilograms.

r

mtcars %>%

filter(mpg > 20) %>%

mutate(wt\_kg = wt \* 0.453592) %>%

group\_by(cyl) %>%

summarise(avg\_wt\_kg = mean(wt\_kg))

This filters cars with mpg > 20, creates a new column for weight in kilograms, and calculates the average weight per cylinder group.

**Example 8: Multiple Summaries**

Group by cylinders and calculate both the average horsepower and average weight.

r

mtcars %>%

group\_by(cyl) %>%

summarise(avg\_hp = mean(hp), avg\_wt = mean(wt))

This groups the data by the number of cylinders and computes both the average horsepower and weight.

**Example 9: Arranging Results After Summarising**

Group cars by the number of cylinders, compute the average miles per gallon, and sort the result in descending order of avg\_mpg.

r

mtcars %>%

group\_by(cyl) %>%

summarise(avg\_mpg = mean(mpg)) %>%

arrange(desc(avg\_mpg))

This groups the data by the number of cylinders, calculates the average mpg, and sorts the result.

**Example 10: Combining Multiple Functions**

Filter cars with more than 4 cylinders, select mpg, hp, and wt, create a new column for horsepower per weight, group by the number of cylinders, and then calculate the average horsepower per weight for each group.

r

mtcars %>%

filter(cyl > 4) %>%

select(mpg, hp, wt) %>%

mutate(hp\_per\_wt = hp / wt) %>%

group\_by(cyl) %>%

summarise(avg\_hp\_per\_wt = mean(hp\_per\_wt))

This example combines filtering, selecting, mutating, grouping, and summarising to derive a new insight.

**Summary of the Logical Workflow:**

1. **filter()**: Narrow down the data by conditions.
2. **select()**: Choose only the columns you need.
3. **mutate()**: Create new variables or modify existing ones.
4. **group\_by()**: Organize data into groups based on one or more variables.
5. **summarise()**: Compute summary statistics for each group.

This workflow, using the pipe operator (%>%), allows you to perform complex transformations in a simple, readable manner.