filter() Method in R

In R Programming Language the filter() method is a powerful tool for subsetting data frames based on specified conditions. It allows you to extract rows that meet specific criteria, providing a flexible and efficient way to manipulate data. This comprehensive guide aims to demystify the filter() method, covering its syntax, functionality, and practical examples.

The filter() function is used to subset a data frame, retaining all rows that satisfy your conditions. To be retained, the row must produce a value of TRUE for all conditions. Note that when a condition evaluates to NA the row will be dropped, unlike base subsetting with [.

Understanding the filter() Method

The filter() method is part of the dplyr package, a popular package in the R ecosystem for data manipulation. It is designed to work with data frames and Tibbles, enabling users to extract subsets of data based on logical conditions.

The basic syntax of the filter() method is as follows:

Syntax: filter(data_frame, condition)

Parameters:

- **data_frame:** The input data frame or tibble.
- **condition:** The logical condition used to filter rows.

filter(.data, ..., .by = NULL, .preserve = FALSE)

data - A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).

... - <data-masking> Expressions that return a logical value, and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.

by - <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by().

preserve - Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Useful filter functions: There are many functions and operators that are useful when constructing the expressions used to filter the data:

- ==, >, >= etc
- &, |, !, xor()
- is.na()
- between(), near()

Filtering Rows Based on a Single Condition:

```
# Load necessary library

library(dplyr)
```

```
# Create a simple dataset
employees <- data.frame(
  ID = 1:10,
  Name = c("John", "Jane", "Bill", "Anna", "Tom", "Sue", "Mike", "Sara", "Alex","Nina"),
  Department = c("HR", "Finance", "IT", "Finance", "IT", "HR", "IT", "Finance", "HR", "Finance"),
  Salary = c(50000, 60000, 70000, 65000, 72000, 48000, 75000, 67000, 52000, 69000)
)
# Print the dataset
print(employees)
# Filter employees in the IT department
it <- filter(employees,Department == "IT")
# Print the result
print(it)
```

Filter by Multiple Conditions:

Filter employees in the Finance department with a salary greater than 65000.

```
# Filter employees in the Finance department with salary greater than 65000
high_paid_finance_employees <-filter(employees,Department == "Finance" & Salary > 65000)
# Print the result
print(high_paid_finance_employees)
```

Filter Using the or and %in% Operator:

With the help of or operator we Filter employees in either the HR or IT department and with the help of in operator Filter employees in specific departments (HR and Finance).

```
# Filter employees in either HR or IT department
hr_it_employees <- employees %>% filter(Department == "HR" | Department == "IT")
# Print the result
print(hr_it_employees)
# Filter employees in HR or Finance department using %in% operator
hr_finance_employees <- employees %>% filter(Department %in% c("HR", "Finance"))
# Print the result
```

print(hr_finance_employees)

Example:

Using the filter() function from the dplyr package.

library(**dplyr**)

Dataset – **starwars**

#view first six rows of starwars dataset

head(starwars)

Example 1: Filter Rows Equal to Some Value

Filter the dataset for rows where the variable 'species' is equal to Droid.

starwars %>% filter(species == 'Droid')

Example 2: Filter Rows Using 'And'

filter for rows where the species is Droid and the eye color is red

starwars %>% filter(species == 'Droid' & eye_color == 'red')

Example 3: Filter Rows Using 'Or'

filter for rows where the species is Droid or the eye color is red

starwars %>% filter(species == 'Droid' | eye_color == 'red')

A tibble: 7 x 13

Example 4: Filter Rows with Values in a List

filter for rows where the eye color is in a list of colors

starwars %>% filter(eye_color %in% c('blue', 'yellow', 'red'))

Example 5: Filter Rows Using Less Than or Greater Than

filter rows using less than or greater than operations on numeric variables

#find rows where height is greater than 250

```
starwars %>% filter(height > 250)

#find rows where height is between 200 and 220
starwars %>% filter(height > 200 & height < 220)

#find rows where height is above the average height
starwars %>% filter(height > mean(height, na.rm = TRUE))
```

Examples:

```
# Filtering by one criterion
filter(starwars, species == "Human")

filter(starwars, mass > 1000)

# Filtering by multiple criteria within a single logical expression
filter(starwars, hair_color == "none" & eye_color == "black")
filter(starwars, hair_color == "none" | eye_color == "black")

# When multiple expressions are used, they are combined using &
filter(starwars, hair_color == "none", eye_color == "black")

#The following filters rows where `mass` is greater than the global average:
starwars %>% filter(mass > mean(mass, na.rm = TRUE))

# Whereas this keeps rows with `mass` greater than the gender average:
starwars %>% group_by(gender) %>% filter(mass > mean(mass, na.rm = TRUE))
```

Example:

```
#create data frame
df <- data.frame(player = c('a', 'b', 'c', 'd', 'e'),
                 position = c('G', 'F', 'F', 'G', 'G'),
```

```
                points = c(12, 15, 19, 22, 32),
                rebounds = c(5, 7, 7, 12, 11))

#view data frame
Df
```

Example 1: Remove Columns by Name

```
#remove column named 'points'
df %>% select(-points)
```

Example 2: Remove Columns in List

```
#remove columns named 'points' or 'rebounds'
df %>% select(-one_of('points', 'rebounds'))
```

Example 3: Remove Columns in Range

Remove all columns in the range from ‘position’ to ‘rebounds’:

```
#remove columns in range from 'position' to 'rebounds'
df %>% select(-(position:rebounds))
```

Example 4: Remove Columns that Contain a Phrase

Remove all columns that contain the word ‘points’

```
#remove columns that contain the word 'points'
df %>% select(-contains('points'))
```

Example 5: Remove Columns that Start with Certain Letters

Remove all columns that start with the letters ‘po’

```
#remove columns that start with 'po'
df %>% select(-starts_with('po'))
```

Example 6: Remove Columns that End with Certain Letters

Remove all columns that end with the letter ‘s’

```
#remove columns that end with 's'
```

```
df %>% select(-ends_with('s'))
```

Example 6: Remove Columns that End with Certain Letters

Remove all columns that end with the letter ‘s’

```
#remove columns that end with 's'
df %>% select(-ends_with('s'))
```

Example 7: Remove Columns by Position

remove columns in specific positions

```
#remove columns in position 1 and 4
df %>% select(-1, -4)
```