How to Easily Filter Rows Between Two Values in R

Authored by

stats writer

November 20, 2025

Filtering rows in R is a fundamental operation in data analysis, allowing practitioners to focus on specific subsets of information. When working with numerical data, a recurring requirement is to select records where a particular column's value lies within a defined range--i.e., between a specified lower limit and an upper limit. This conditional selection is essential for tasks such as identifying outliers, isolating specific experimental groups, or preparing data for specialized statistical modeling. Understanding how to perform this operation efficiently using robust and readable code is key to effective data frame manipulation in R.

In this comprehensive guide, we detail two primary approaches to achieve range filtering. The first utilizes the built-in capabilities of Base R, relying on functions like `subset()` or standard logical indexing. The second, and often preferred method in modern workflows, employs the specialized functions provided by the widely popular Tidyverse package, dplyr. Both methods are powerful, but they offer distinct advantages in terms of syntax clarity and performance, which we will explore through practical, replicable examples.

To effectively filter a data frame in R, isolating rows where a specific column's value falls between two boundaries, you can utilize the following expertly detailed methods:

Method 1: Utilizing Base R Functions for Range Filtering

The Base R environment offers several mechanisms for subsetting data. The most straightforward method for range filtering involves using the dedicated `subset()` function, which is designed to return subsets of vectors, matrices, or data frames that satisfy specified conditions. The power of `subset()` lies in its concise syntax, allowing conditions to be expressed directly using column names without the need for the `data frame$` prefix.

Alternatively, for more complex scenarios, Base R supports Boolean indexing, combining logical operators (`&`, `|`) with standard comparison operators (`>=`, ``). However, for simple, inclusive numerical ranges, a highly efficient technique is to leverage the sequencing operator (`:`) in conjunction with the value matching operator (`%in%`). This approach verifies if the values in the target column are present within the sequence defined by the lower and upper bounds.

The following syntax demonstrates the concise use of the `subset()` function combined with the `%in%` operator to select rows where the column value is within the specified integer range (100 to 120, inclusive):

df_new <- subset(df, points %in% 100:120)

Method 2: Leveraging the Power of the Tidyverse and dplyr

The Tidyverse framework, particularly the dplyr package, provides a modern and highly readable alternative to Base R operations. The Tidyverse philosophy emphasizes consistent function syntax and the use of the pipe operator (`%>%`), leading to chains of operations that are easy to understand and maintain. For range filtering, dplyr offers two key functions: `filter()` and the specialized utility function `between()`.

The `filter()` function is the primary tool in dplyr for row selection based on logical conditions. It accepts a data frame (usually passed via the pipe) and one or more logical expressions. The `between(x, left, right)` function is a specific helper designed precisely for our task; it tests whether a value `x` falls inclusively within the range defined by `left` and `right`. This combination drastically improves code clarity compared to manually chaining multiple logical operators (e.g., `points >= 100 & points <= 120`).

The syntax below illustrates the use of the Tidyverse approach. Note the dependency on loading the dplyr library first, followed by the piping structure which directs the original data frame (`df`) into the filtering operation:

library(dplyr)

df_new <- df %>% filter(between(points, 100, 120))

In both methodologies presented above, the objective is identical: to create a new data frame (`df_new`) that contains only the rows where the value in the **points** column is inclusively situated between the values of **100** and **120**. The choice between these methods often depends on the overall programming context and adherence to either Base R conventions or the Tidyverse style.

Setting Up the Demonstration Data Frame

To demonstrate these filtering techniques in a practical setting, we will first create a sample data frame named `df`. This data frame simulates basic sports statistics, including team names, points scored, and assists recorded. Having a concrete dataset allows us to visually verify the results of our filtering operations and confirm that only the rows meeting the specified criteria are retained.

The construction of this sample data frame uses the standard Base R function `data.frame()`, explicitly defining three columns: `team` (character vector), `points` (numeric vector), and `assists` (numeric vector). The values chosen for the `points` column are intentionally spread across the 100-135 range to ensure we have both qualifying and non-qualifying rows for our subsequent filtering examples.

The following R code chunk creates and then displays the structure and contents of our demonstration data:

```
#create data frame
df <- data.frame(team=c('Mavs', 'Pacers', 'Mavs', 'Celtics', 'Nets', 'Pacers'),
points=c(104, 110, 134, 125, 114, 124),
assists=c(22, 30, 35, 35, 20, 27))

#view data frame
df
```

```
team points assists
1 Mavs 104 22
2 Pacers 110 30
3 Mavs 134 35
4 Celtics 125 35
5 Nets 114 20
6 Pacers 124 27
```

Upon reviewing the original data frame, we can anticipate which rows should be retained when filtering for `points` between 100 and 120: rows 1 (104 points), 2 (110 points), and 5 (114 points) are expected to remain in the resultant data frame. Rows 3 (134 points), 4 (125 points), and 6 (124 points) should be filtered out as they exceed the upper limit of 120.

Example 1: Filtering Where Column is Between Two Values Using Base R

We now apply the Base R filtering approach using the powerful **subset()** function. The advantage of using the `%in%` operator with the sequence notation (`100:120`) is that it creates a vector of all integers from the lower bound to the upper bound, and then checks for an exact match for each value in the `points` column. This is generally cleaner than writing `subset(df, points >= 100 & points <= 120)`, particularly for inclusive integer ranges.

The following implementation demonstrates the Base R syntax. We assign the resulting subset to a new object, `df_new`, preserving the original data frame integrity. The subsequent display of `df_new` confirms the successful application of the filter logic, reducing the data set to only the qualifying rows.

```
#filter for rows where value in points column is between 100 and 120 (inclusive)
df_new <- subset(df, points %in% 100:120)

#view updated data frame
```

df_new

team points assists
1 Mavs 104 22
2 Pacers 110 30
3 Nets 114 20

As expected, the resulting data frame, `df_new`, contains only three rows. Notice that the index numbers (1, 2, 3) correspond to the original row indices (1, 2, 5). This result confirms that only the rows where the value in the **points** column falls inclusively between 100 and 120 were retained, while all other rows with a value outside of this precise range were successfully dropped by the filtering mechanism.

Example 2: Filtering Where Column is Between Two Values Using dplyr

For analysts operating within the Tidyverse, the combination of the `filter()` and `between()` functions from the dplyr package provides an exceptionally clear and intuitive syntax for this task. The explicit use of the `between()` function immediately communicates the intent of the operation, enhancing code readability, particularly when dealing with complex data manipulation pipelines that involve multiple steps chained together using the pipe operator (`%>%`).

Before executing the filtering, it is mandatory to load the dplyr library into the R session. Once loaded, the data frame (`df`) is piped to the `filter()` function, where the `between()` utility handles the range check for the `points` column, taking the lower limit (100) and the upper limit (120) as its subsequent arguments.

library(dplyr)

#filter for rows where value in points column is between 100 and 120
df_new <- df %>% filter(between(points, 100, 120))

#view updated data frame
df_new

team points assists
1 Mavs 104 22
2 Pacers 110 30
3 Nets 114 20

The output from the dplyr method is identical to the Base R method, reinforcing that both achieve the same robust result. The use of `between()` internally translates to the inclusive condition

`points >= 100 & points <= 120`. This method is highly favored in modern R workflows due to its streamlined structure and clear demarcation of the filtering logic.

Handling Edge Cases and Boundary Conditions

When filtering numerical ranges, understanding how boundary conditions are handled is critical. By default, both the Base R `%in% 100:120` approach (for integers) and the dplyr `between(points, 100, 120)` function are **inclusive**. This means that if a row has a value exactly equal to the lower limit (100) or the upper limit (120), that row will be included in the resulting subset.

If, however, you require an **exclusive** filter (where the bounds themselves are not included), you must use explicit logical indexing, even within the dplyr framework. For example, to filter for values strictly greater than 100 and strictly less than 120, the syntax shifts slightly.

Exclusive Filter (Base R Syntax): `subset(df, points > 100 & points < 120)`

Exclusive Filter (dplyr Syntax): `df %>% filter(points > 100, points < 120)`

Furthermore, when dealing with real-world data, the presence of missing values (NA) must be considered. In R, comparisons involving NA typically result in NA, which is treated as `FALSE` by filtering functions. Therefore, rows containing `NA` in the target column (`points`) will automatically be excluded from the subset unless specific logic is added to handle them, such as using `is.na()` within the filter condition if you wished to retain them.

Comparing Performance: Base R vs. dplyr

While both methods yield correct results, their underlying implementation and performance characteristics can differ, particularly when processing very large datasets (millions of rows). The choice between **Base R** and **dplyr** often involves a trade-off between execution speed and code readability.

Generally, Base R operations, especially those utilizing logical vector indexing (e.g., `df`), tend to be highly optimized because they rely on fundamental R primitives. For moderate-sized data frames, the Base R approach is often marginally faster due to less overhead from function calls and environment management.

However, the `dplyr` package is highly optimized, particularly when combined with efficient backend systems like the C++ implementation provided by the `Rcpp` package. For large-scale data manipulation and complex pipelines, the increase in development efficiency and code clarity offered by the `filter()` function often outweighs minor differences in raw processing speed. In the context of the Tidyverse, the consistent, readable syntax makes `df %>% filter(between(...))` the recommended best practice for most analytical tasks.

Note: You can find the complete documentation for the **filter** function in dplyr, along with detailed explanations of its various uses in row selection.