How to Filter Data Frames by Factor Levels Using dplyr in R

Authored by

stats writer

January 16, 2026

The dplyr package stands as a cornerstone in the modern R ecosystem, providing a powerful, consistent, and highly optimized set of tools for manipulating structured data. Designed with performance and readability in mind, dplyr simplifies common data wrangling tasks that often prove complex using base R functions, especially when dealing with large datasets. At its core, dplyr provides a clear syntax for common operations such as selecting, grouping, summarizing, arranging, and crucially, subsetting data. Mastering this package is essential for any professional working with data analysis or statistical modeling in R, as it dramatically increases productivity and ensures reproducible research outcomes.

One of the most frequent requirements in data analysis is the ability to subset or isolate specific observations based on categorical attributes. These attributes are typically stored in data frame columns as a special type of variable known as a factor. A factor is a categorical variable in R, meaning it holds a fixed, limited number of unique levels or labels. For example, a column representing 'Gender' might have levels 'Male', 'Female', and 'Other', or a 'Region' column might contain 'North', 'South', 'East', and 'West'. The efficient handling and manipulation of these factors is paramount, as misinterpretation of their underlying structure can lead to erroneous results during analysis.

The challenge often lies in correctly isolating rows within a data frame that correspond to one or more specific factor levels. Whether you are preparing data for visualization, running regression models, or generating summary statistics for specific sub-populations, the ability to filter based on these categorical constraints is fundamental. This tutorial focuses on how dplyr provides elegant and concise solutions for this task, utilizing the powerful combination of the `filter()` function and various logical operators tailored for factor variables. We will explore two distinct methodologies: filtering based on the human-readable factor labels, and filtering based on the underlying numeric factor levels.

1. The Critical Role of the filter() Function in Data Manipulation

The `filter()` function is arguably one of the most frequently used verbs in the dplyr toolkit. Its purpose is straightforward yet incredibly powerful: it allows the user to subset rows based on logical conditions applied to column values. When dealing with character or numeric variables, the conditions might involve simple comparisons like equality (==), inequality (!=), or range checks (>, <). However, when working with factor variables, the syntax must accommodate the unique dual nature of these objects--having both a visible label and an underlying integer representation.

Effective use of `filter()` necessitates a deep understanding of R's standard evaluation rules and how dplyr leverages non-standard evaluation (NSE). In simple terms, inside the `filter()` function, you write code as if you were defining a new vector based on existing column names, without needing to reference the data frame name repeatedly. This streamlined approach, combined with

the piping operator (`%>%`) which passes the output of one function as the input to the next, creates highly readable and maintainable data pipelines. When filtering factors, the choice of operator--whether the standard equality check or the specialized inclusion operator `%in%`--determines the efficiency and clarity of the resultant code block.

Filtering based on factors is essential because it allows analysts to move beyond simple binary selection. Instead of just selecting 'A' versus 'not A', we can select 'A', 'C', and 'D' simultaneously, or select everything **except** 'B'. This multi-criteria subsetting is where `filter()` truly shines, especially when combined with logical operators like `|` (OR) or `&` (AND). Furthermore, unlike manual subsetting using base R square bracket notation, the dplyr approach integrates seamlessly into complex transformations, ensuring that the filtered output is immediately ready for subsequent steps, such as grouping or aggregation, without creating unnecessary intermediate variables.

2. Understanding R Factors: Labels vs. Levels

To effectively filter using a factor variable in R, it is critical to distinguish between the factor's labels and its underlying levels. The **labels** are the human-readable strings associated with each category--for instance, 'High', 'Medium', or 'Low'. These are the values you see when you print the data frame or inspect the column contents. The **levels**, however, are the internal integer representations that R uses for storage and comparison. By default, R assigns integers starting from 1 to the levels alphabetically or in the order they were first encountered (unless explicitly specified otherwise).

This duality means that a single categorical entry ('A', for example) holds two types of values: the string 'A' (the label) and an integer (e.g., 1, 2, 3, or 4) that corresponds to its position in the stored list of levels. When you compare factor values using standard string operations, R typically uses the labels. However, R allows coercion of factors to integers using functions like `as.integer()`, which exposes the underlying numeric levels. This capability provides two distinct, powerful methods for filtering, depending on whether the analyst requires string matching or numerical comparison.

Understanding the ordering of these internal levels is particularly important when dealing with ordinal factors--categories that inherently have a rank or sequence (like 'Small', 'Medium', 'Large'). If the factor is defined such that 'Small' maps to 1, 'Medium' to 2, and 'Large' to 3, then filtering for levels greater than 2 would logically select only 'Large'. If, however, the levels are ordered alphabetically (e.g., 'Large' = 1, 'Medium' = 2, 'Small' = 3), then filtering for levels greater than 2 would yield only 'Small'. This distinction highlights why Method 2 (filtering by level) requires careful consideration of the factor definition, whereas Method 1 (filtering by label) is generally safer for nominal (unordered) factor data.

3. Method 1: Filtering Based on Explicit Factor Labels

You can use the following methods in dplyr to filter the rows of a data frame in R based on a factor variable:

The most intuitive and frequently used approach for subsetting data based on categorical variables involves referencing the explicit factor labels. This method is highly transparent and minimizes the risk of error, as the criteria used in the code directly correspond to the visible data values. To accomplish this using dplyr, we utilize the `filter()` function in conjunction with the `%in%` operator. The `%in%` operator efficiently checks if the value in the specified column matches any item within a provided vector of desired values, making it ideal for selecting multiple categories simultaneously.

To implement this method, the user supplies the column name (which must be a factor or character variable) on the left side of the `%in%` operator, and a character vector containing the desired labels on the right side. The syntax remains clean and readable, clearly stating the intention: "keep rows where the value in this factor column is one of these listed labels." This approach is robust because it relies purely on string matching, circumventing any potential issues related to the underlying integer mapping or level ordering, which can sometimes be non-obvious to an external user or during automated processing.

Method 1: Filter Based on Factor Labels

library(dplyr)

#filter rows where team column is equal to factor label 'A' or 'C'
df %>%
filter(team %in% c('A', 'C'))

4. Method 2: Filtering Based on Underlying Factor Levels (Integers)

While filtering by factor labels (Method 1) is generally recommended for clarity, there are specific scenarios where filtering based on the underlying numeric levels (integers) becomes necessary or highly efficient. This method is particularly relevant when the factor is inherently **ordinal**, meaning the categories possess a meaningful sequence or rank (e.g., severity ratings: 1=Low, 2=Medium, 3=High). In such cases, numerical comparison operators (`>`, `<`, `>=`, `<=`) can be used to select entire ranges of categories without having to list every single label explicitly.

To leverage the numeric levels in R, we must explicitly coerce the factor variable into an integer type within the `filter()` function. This is achieved using the `as.integer()` function, which transforms the categorical column into a vector of integers corresponding to the factor's established order. The key consideration here is that the comparison is performed not against the

labels ('A', 'B', 'C'), but against their positional indices (1, 2, 3). For this technique to produce accurate results, the analyst must be certain of the order of the factor levels, as defined during the data frame creation or later manipulation.

For example, if we wish to select all categories that fall above a certain threshold in an ordinal scale, filtering by levels is far more concise than listing all subsequent labels. We might want to select all teams whose factor level is greater than 2. Assuming the default alphabetical ordering ('A'=1, 'B'=2, 'C'=3, 'D'=4), filtering for levels greater than 2 will automatically select teams 'C' and 'D'. This method simplifies range selection considerably, offering an alternative when the numerical ranking of the categories is the primary concern, rather than the categorical names themselves.

Method 2: Filter Based on Factor Levels

library(dplyr)

#filter rows where factor level of team column is greater than 2
df %>%
filter(as.integer(team)>2)

5. Initializing the Data Frame for Demonstration

To demonstrate both filtering methodologies in a practical context, we must first establish a working data frame in R that contains a factor variable. For this demonstration, we create a small dataset detailing basketball player information, featuring the `team` column defined explicitly as a factor. This setup ensures that we are properly dealing with the categorical structure inherent in R's factor type, which allows us to showcase both filtering methods accurately. The initial data creation step is crucial for establishing the context of the subsequent filtering operations.

The dataset below contains information about various basketball players, where the `team` column serves as our categorical variable. Note that the default factor creation in R typically assigns levels in alphabetical order ('A', 'B', 'C', 'D'), which will be essential for understanding the level-based filtering method later on.

#create data frame
df <- data.frame(team=as.factor(c('A', 'A', 'A', 'B', 'B', 'C', 'C', 'D')),
points=c(12, 34, 20, 25, 22, 28, 34, 19))

#view data frame
df

team points

1 A 12
2 A 34
3 A 20
4 B 25
5 B 22
6 C 28
7 C 34
8 D 19

6. Example 1: Filtering Based on Factor Labels (Using %in%)

This first example implements Method 1, filtering the data frame based on the explicit, human-readable labels of the `team` factor. We aim to subset the data to include only observations where the team membership is either 'A' or 'C'. This approach is highly readable and is generally the safest way to filter nominal categorical data, as it is independent of the underlying numerical order of the factor levels.

We use the `%in%` operator within the `filter()` function to achieve this multi-criteria selection efficiently. This operator checks for membership, allowing us to specify a vector of desired labels, providing an elegant alternative to chaining multiple OR (`|`) conditions. The resulting output will strictly conform to the criteria defined by the listed factor labels.

library(dplyr)

#filter rows where team column is equal to factor label 'A' or 'C'
df %>%
filter(team %in% c('A', 'C'))

team points
1 A 12
2 A 34
3 A 20
4 C 28
5 C 34

Notice that the resulting data frame only contains rows where the value in the **team** column is equal to either A or C. This outcome confirms the successful application of the label-based filtering technique, demonstrating how efficiently dplyr handles categorical subsetting based on explicit label inclusion.

7. Example 2: Filtering Based on Factor Levels (Using as.integer())

This second example implements Method 2, filtering the data frame based on the numerical rank of the factor levels. Our goal is to select all rows where the factor level of the `team` column is strictly greater than 2. Since the teams were created in alphabetical order ('A'=1, 'B'=2, 'C'=3, 'D'=4), this condition should isolate teams 'C' and 'D'. This method is particularly useful when dealing with ordinal data where numerical comparison provides a concise way to select ranges of categories.

The key function here is `as.integer()`, which temporarily coerces the factor variable into its numerical level representation, allowing us to use standard numerical comparison operators (>) within the `filter()` function. This technique provides a powerful way to leverage the underlying data structure, provided the analyst is aware of the level ordering.

library(dplyr)

#filter rows where factor level of team column is greater than 2
df %>%
filter(as.integer(team)>2)

team points
1 C 28
2 C 34
3 D 19

In this particular example, the **as.integer** function converts the factor labels of the **team** column to their corresponding internal integer values. The resulting data frame only includes teams 'C' and 'D' because their assigned integer levels (3 and 4, respectively) satisfy the condition of being greater than 2. This confirms the successful application of level-based filtering for range selection.

For example, the conversion process used by R for this factor variable is as follows:

Factor level 'A' becomes **1**.
Factor level 'B' becomes **2**.
Factor level 'C' becomes **3**.
Factor level 'D' becomes **4**.

8. Choosing the Right Method for Factor Filtering

The decision between using factor labels (Method 1) and factor levels (Method 2) hinges entirely on the nature of the categorical variable and the specific analytical requirement. For **nominal factors**--categories without inherent ordering, such as 'Color' or 'Country'--filtering by label is

overwhelmingly the recommended practice. Using labels (`team %in% c('A', 'C')`) results in code that is robust, self-explanatory, and insensitive to changes in factor level ordering, which might occur if new data is introduced or if the factor definition is accidentally reordered.

Conversely, filtering by factor levels (`as.integer(team) > 2`) should be reserved for scenarios involving **ordinal factors**--categories where the order matters significantly, such as 'Satisfaction Rating' (Poor, Fair, Good, Excellent). If an analyst is confident in the established order of the levels, this method provides a shortcut for range selection (e.g., selecting 'Good' and 'Excellent' by targeting levels greater than 2). However, analysts must exercise extreme caution: if the underlying level order is modified without updating the filtering criteria, the code will silently produce incorrect results, as the numeric value 3 might suddenly correspond to a different label.

A key advantage of the label-based approach (Method 1) is its resilience to data integrity issues. If a new, unlisted factor level (say, 'E') were introduced to the variable, the label-based filter would continue to work correctly, selecting only 'A' and 'C'. However, the level-based filter would shift its integer mappings, potentially causing the filter `> 2` to now select unexpected categories if the ordering shifted unexpectedly. Therefore, unless a numerical comparison is fundamentally required for ordinal data, sticking to explicit labels ensures maximum code stability and readability across different analytical environments and data snapshots.

9. Conclusion and Resources

The dplyr package offers flexible and intuitive methods for subsetting data based on factor variables within the R environment. By leveraging the `filter()` function alongside appropriate comparison operators, analysts can precisely isolate the observations necessary for specific downstream analyses. Whether choosing the label-based approach using `%in%` or the level-based approach using `as.integer()`, the resulting code is significantly cleaner and more efficient than traditional base R indexing methods, aligning perfectly with the principles of tidy data analysis.

As a final recommendation, adopt the label-based filtering (Method 1) as the default choice for all nominal factor variables due to its unparalleled clarity and robustness against unexpected changes in the factor level structure. Reserve the level-based filtering (Method 2) strictly for variables that possess a clear, non-negotiable ordinal structure where numerical range selection is the primary analytical goal. Always verify the factor levels using functions like `levels()` before applying `as.integer()` to prevent misinterpretation of the numerical rankings.

The following tutorials explain how to perform other common functions in dplyr: