# Learning R: How to Select Rows Based on Values in Any Column

Authored by

**Mohammed looti**

November 3, 2025

Efficiently querying and subsetting data is a foundational skill for any data analysis project, particularly within the R programming environment. A frequent and often tricky challenge faced by analysts involves identifying specific rows within a data frame where a target value--or a defined set of values--exists in **any** column, rather than being confined to a single, predefined variable. Successfully executing this task requires a highly specialized approach, which is elegantly and powerfully solved using the functionality provided by the widely adopted dplyr package.

This comprehensive guide will demonstrate how to leverage the dplyr package to perform generalized searches across entire datasets. We will specifically focus on the potent combination of the `filter_all()` and `any_vars()` functions. This syntax not only provides highly flexible data manipulation capabilities but also results in exceptionally readable and concise code, crucial for maintaining complex data pipelines.

To immediately address this common requirement, analysts can utilize the following standard syntax structure to locate and select rows within an R data frame where a specific value appears in one or more columns:

**library(dplyr)**

df %>% filter_all(any_vars(. %in% c('value1', 'value2', ...)))

The subsequent sections will provide detailed, practical examples, illustrating how this syntax is applied effectively in real-world data science scenarios, covering both numeric and character data types, and concluding with a note on modern `dplyr` best practices.

## The Core Mechanism: Deconstructing filter_all and any_vars

Before implementing the solution, it is vital to establish a clear understanding of the roles played by the two primary functions in this operation: `filter_all` and `any_vars`. Both functions are integral components of the dplyr package, which forms the backbone of the Tidyverse ecosystem--a collection of R packages specifically designed for intuitive and efficient data wrangling.

The function `filter_all` is designed to generalize filtering criteria across every single variable in the input data frame. By utilizing `filter_all`, the user avoids the tedious requirement of manually specifying column names. This is especially useful when working with wide datasets that may contain dozens or even hundreds of columns where a generalized check is required. The function automatically iterates over all variables, significantly reducing code complexity and improving efficiency.

In contrast, `any_vars` functions as a crucial logical aggregator. Its role is to ensure that the conditional statement provided within its arguments must evaluate to true for at least one column

for the row in question to be retained. If the search value is found in column A **OR** column B **OR** column C, the row is included in the output. The powerful synergy between `filter_all` (checking all columns) and `any_vars` (requiring just one match) creates a highly generalized search utility that executes universal OR logic across the entire dataset.

## Practical Application: Searching for a Single Numeric Value

To effectively illustrate this technique, let us define a basic numeric data frame in R, which contains hypothetical sports statistics for five different players. This dataset serves as our starting point for all subsequent numeric filtering examples:

**#define data frame**
**df = data.frame(points=c(25, 12, 15, 14, 19),**
**assists=c(5, 7, 7, 9, 12),**
**rebounds=c(11, 8, 10, 6, 6))**

#view data frame
df

points assists rebounds
1 25 5 11
2 12 7 8
3 15 7 10
4 14 9 6
5 19 12 6

Our initial objective is straightforward: we aim to isolate and select every row in this data frame that contains the specific numeric value **25** in any of its columns, regardless of whether that value appears in the `points`, `assists`, or `rebounds` column. This requires a generalized check across the entire structure.

The following syntax executes this precise filtering operation using the combined functionality of the dplyr package. Note the use of the binary operator `%in%`, which checks for inclusion within the specified vector--in this instance, the vector contains only the target value, 25.

**library(dplyr)**

#select rows where 25 appears in any column
df %>% filter_all(any_vars(. %in% c(25)))

points assists rebounds

1 25 5 11

The resulting output clearly shows that only the first row is retained, as it is the only one where the value **25** appears (specifically, within the `points` column). This confirms the successful application of the generalized row selection mechanism.

## Scaling the Search: Filtering Based on Multiple Numeric Criteria

The true advantage of this methodology becomes apparent when the search criteria must encompass multiple values simultaneously. Instead of being restricted to checking for a single data point, we can pass an entire vector of target values into the `c()` function nested within the `any_vars` call. This capability is exceptionally useful for data subsetting tasks that involve criteria spanning across several columns, such as identifying all rows associated with a defined list of outlier scores or specific threshold values.

For this example, we will significantly expand our search criteria to include the values **25, 9, or 6**. Our goal is now to select all rows where at least one of these three numbers is present in any of the available columns (`points`, `assists`, or `rebounds`). This operation requires powerful OR logic applied both across the criteria set and across the column set.

**library(dplyr)**

#select rows where 25, 9, or 6 appears in any column
df %>% filter_all(any_vars(. %in% c(25, 9, 6)))

points assists rebounds
1 25 5 11
2 14 9 6
3 19 12 6

The resulting output effectively demonstrates the selection process: Row 1 is kept because it contains 25. Row 4 is kept because it contains both 9 (in `assists`) and 6 (in `rebounds`). Finally, Row 5 is kept because it contains 6 (in `rebounds`). The `any_vars` function successfully ensures that as long as the filtering condition is met even once within a row, that row is fully preserved in the resulting filtered data frame.

This streamlined approach drastically simplifies complex logical operations that would otherwise demand extensive, manually constructed chains of the OR operator (`|`) in base R or require numerous explicit column name references within standard `dplyr::filter()` calls. By generalizing the search, code becomes cleaner and far less prone to errors.

## Handling Character Data in Mixed Type Data Frames

The `filter_all` and `any_vars` pattern is highly versatile and is not restricted solely to numeric inputs; it operates equally well when searching for specific character strings. This methodology proves particularly robust when applied to mixed data types--for instance, data frames containing both quantitative (numeric) and qualitative (character) columns--provided that the search value correctly aligns with the column's underlying data type.

Let us introduce a revised data frame that incorporates a new character column, `position`, which represents the player's role, alongside the existing numeric statistics. This change allows us to demonstrate how the filtering mechanism handles non-numeric data:

**#define data frame**
**df = data.frame(points=c(25, 12, 15, 14, 19),**
**assists=c(5, 7, 7, 9, 12),**
**position=c('G', 'G', 'F', 'F', 'C'))**

#view data frame
df

points assists position
1 25 5 G
2 12 7 G
3 15 7 F
4 14 9 F
5 19 12 C

Our current task is to select all rows of this data frame that contain the character string **G** in any column. It is critical to remember that when searching for strings, the values must be correctly enclosed in single or double quotes within the `c()` vector passed to `any_vars()`.

**library(dplyr)**

df %>% filter_all(any_vars(. %in% c('G')))

points assists position
1 25 5 G
2 12 7 G

The resulting filtered output accurately returns the two rows where the player's `position` is 'G'. Crucially, the filtering operation correctly handles the disparate data types, effectively bypassing

the numeric columns (`points` and `assists`) where the character string 'G' would logically not be found, ensuring that only meaningful matches are considered.

## Advanced String Filtering: Combining Multiple Character Criteria

Mirroring the flexibility demonstrated with numeric data, we can easily define a vector containing multiple character strings to search for simultaneously. Suppose a researcher is interested in isolating players categorized as either a Guard ('G') or a Center ('C').

By including both 'G' and 'C' in the search vector, the `any_vars` function executes its powerful OR logic across both the set of target values and the set of columns. It checks if a row contains either 'G' OR 'C' in any column, providing an exceptionally efficient method for subsetting large datasets based on complex categorical criteria.

**library(dplyr)**

df %>% filter_all(any_vars(. %in% c('G', 'C')))

points assists position
1 25 5 G
2 12 7 G
3 19 12 C

The final result successfully isolates the two rows corresponding to position 'G' and the single row corresponding to position 'C'. This outcome powerfully validates the robustness of using `filter_all` for complex, multi-criteria searches, even within R data frames that contain mixed data types.

## Modern dplyr Practices: Transitioning to across() and if_any()

While the `filter_all()` function is highly effective and remains fully functional for backward compatibility, users of modern dplyr should be aware that it has been superseded in recent updates. The Tidyverse community is actively transitioning toward a more flexible and unified approach utilizing the `across()` methodology for all column-wise operations.

The recommended modern approach combines the standard `filter()` function with `across()` and the helper function `if_any()` to achieve the same generalized filtering results with enhanced control and clarity. A key advantage of using `across()` is the ability to explicitly control which columns are selected before the filtering logic is applied, often utilizing selection helpers such as `starts_with()`, `is.numeric()`, or `everything()`.

To replicate the result achieved earlier (checking if the values 25, 9, or 6 appear in any column), the modern syntax using `if_any` would be structured as follows:

**library(dplyr)**

# Modern approach using across()
df %>% filter(if_any(everything(), ~ . %in% c(25, 9, 6)))

Despite the introduction of these newer functions, the legacy tools `filter_all` and `any_vars` are still valuable for rapid data exploration, especially in straightforward cases where checking every single column is genuinely required, due to their concise and highly readable syntax.

## Summary of Techniques and Further Resources

The combination of `filter_all` and `any_vars` within the dplyr package offers an exceptionally elegant and efficient strategy for subsetting an R data frame based on the presence of specific values across multiple columns. This methodology significantly improves coding practices by eliminating the necessity for tedious, manual column enumeration and complex, error-prone conditional statements.

Achieving mastery over these column-wise operations is indispensable for undertaking advanced data cleaning, preparation, and exploration tasks in R. For those who wish to deepen their understanding of related data manipulation functions and modern techniques, the following curated resources are highly recommended for continued learning.

## Additional Learning Resources

The following tutorials explain how to perform other common functions crucial for efficient R programming:

A comprehensive tutorial on selecting specific columns based on their data type using newer functions like `select_if()` or the versatile `across()`.

A detailed guide demonstrating the use of `mutate_all()` and `summarise_all()` for streamlined, non-selective column transformations.

A thorough explanation of the Tidyverse's modern `across()` and `if_any()`/`if_all()` functions, promoting robust contemporary programming practices.

Essential learning on how to effectively handle missing values (NA) during generalized filtering and data manipulation operations in R.