# How to Easily Filter Rows with NA Values in R

Authored by

**stats writer**

November 24, 2025

Working with real-world data invariably means encountering missing values, often represented as NA (Not Available) in R. Identifying and isolating rows containing these missing entries is a fundamental step in data cleaning and analysis. This guide provides an expert breakdown of the two primary, highly efficient techniques used in R for this purpose.

The core methodology relies on generating a logical vector using functions like is.na() or complete.cases(), which maps the presence of missing data to TRUE or FALSE flags. This vector is then utilized for advanced subsetting, allowing us to precisely extract only the desired rows containing NA values from the data frame.

## Understanding Missing Data (NA) in R

Missing data, designated as NA in the R environment, poses a significant challenge during statistical modeling and visualization. Unlike languages where NULL might represent a missing pointer, NA in R specifically denotes a value that is "Not Available" or undefined. Properly handling these values is crucial because most analytical functions, by default, will return NA if any input contains one, potentially masking important results and skewing aggregate metrics.

The first step in mitigation is always identification. Before imputation or removal, we must isolate the affected records. This process is highly dependent on subsetting, which is one of R's most powerful features. When we talk about selecting rows based on NA status, we are essentially asking R to evaluate a condition for every row and return only those rows where that condition (e.g., the presence of an NA) is TRUE. This technique forms the backbone of the methods discussed below, providing efficiency and control over the data selection process.

## The Foundation: Logical Vectors and R Subsetting

In R, the primary tool for detecting missingness is the built-in function is.na(). When applied to a vector or a data frame, it returns a structure of the same dimension containing only TRUE (if the corresponding element is NA) or FALSE (if it is a valid value). This resulting logical vector or matrix is the key mechanism for achieving precise row selection without manual iteration.

We use this logical vector directly inside the square brackets used for subsetting a data frame. For instance, the expression **df** instructs R to return all rows where the specified condition evaluates to TRUE, while retaining all columns (` , `). The methods outlined below leverage this principle, offering two distinct ways to define the condition--checking globally across all columns using complete.cases(), or checking locally within a single, specified column using is.na().

You can use the following methods to select rows with NA values in R:

**Method 1: Select Rows with NA Values in Any Column.** This approach is ideal for general data

cleaning where any missing value invalidates the observation. It uses the negation of complete.cases().

**df**

**Method 2: Select Rows with NA Values in Specific Column.** This method provides surgical precision, focusing only on missingness within a single, designated variable using is.na() applied directly to the column vector.

**df**

The following examples show how to use each method with the following sample data frame in R:

## Preparation: Setting Up the Sample Data Frame for Demonstration

To ensure clarity in the subsequent demonstrations, we initialize a sample data frame called **df**. This dataset simulates common statistical observations and includes intentional NA values placed strategically across multiple rows and columns. This structure allows us to differentiate clearly between the results of the global check (Method 1) and the specific check (Method 2).

The data frame consists of eight observations and three numerical variables: **points**, **rebounds**, and **assists**. We can observe that rows 1, 2, and 6 contain at least one NA value, confirming their status as incomplete cases, while the rest are complete observations.

**#create data frame**
**df <- data.frame(points=c(4, NA, 10, 14, 15, NA, 20, 22),**
**rebounds=c(NA, 3, 3, 7, 6, 8, 14, 10),**
**assists=c(NA, 9, 4, 4, 3, 7, 10, 11))**

#view data frame
df

points rebounds assists
1 4 NA NA
2 NA 3 9
3 10 3 4
4 14 7 4
5 15 6 3
6 NA 8 7
7 20 14 10
8 22 10 11

Our goal with Method 1 will be to select rows 1, 2, and 6. Our goal with Method 2, targeting only the **points** column, will be to select only rows 2 and 6.

## Example 1: Select Rows with NA Values in Any Column (Using !complete.cases)

This method utilizes the highly efficient complete.cases() function, which identifies rows that contain valid data across every variable. By negating the result using the **!** operator, we invert the logical vector to target precisely those rows containing at least one NA. This is the preferred method for standard removal of incomplete cases.

The following code shows how to select rows with NA values in any column of the data frame in R:

```
#select rows with NA values in any column
na_rows <- df

#view results
na_rows
```

```
points rebounds assists
1 4 NA NA
2 NA 3 9
6 NA 8 7
```

The resulting data frame, **na_rows**, successfully includes the three observations identified as having missing data. This confirms that the expression **!complete.cases(df)** generates a logical vector where the first, second, and sixth positions are TRUE, facilitating the exact subsetting required. This technique is computationally efficient and highly readable for general missing data screening.

Notice that the rows with NA values in any column are selected.

## Example 2: Select Rows with NA Values in Specific Column (Using is.na)

When the analysis demands a focus on a single variable, the application of is.na() directly to that column vector is necessary. This method isolates rows based only on missingness within the column specified, ignoring the completeness status of the remaining columns.

The following code shows how to select rows with NA values specifically in the **points** column of

the data frame in R:

**#select rows with NA values in the points column**
**na_rows <- df**

#view results
na_rows

points rebounds assists
2 NA 3 9
6 NA 8 7

The resulting data frame correctly contains only rows 2 and 6. Row 1, although containing NA values in other fields, was excluded because its **points** value (4) was valid. This demonstrates the precise control offered by applying is.na() to a specific column vector before performing the subsetting operation.

Notice that only the rows with NA values in the **points** column are selected.

## Advanced Considerations: Combining Conditions

In complex data analysis, it is often necessary to select rows that satisfy multiple missingness criteria. For instance, selecting rows where **points** is NA **OR rebounds** is NA. This is achieved by combining the is.na() results using the logical OR operator (`|`) or the logical AND operator (`&`).

For example, to select a row if either **points** or **rebounds** is missing, you would use:

**df**

Understanding how to combine these logical vector conditions allows for highly customized and nuanced data filtering, moving beyond simple global or single-column checks to address specific analytical needs.