How do I count the unique values in a column in R?

Authored by

stats writer

November 21, 2025

Determining the number of unique entries within a specific column is a fundamental operation in R for data analysis and preparation. Whether you are checking the cardinality of a categorical variable or verifying data integrity, counting distinct values is essential. Fortunately, the R environment offers several highly efficient methods to accomplish this task, primarily leveraging functionalities from either Base R or the powerful dplyr package. The most straightforward approach involves the n_distinct() function provided by dplyr, which is highly optimized for performance and returns the count of distinct values in a column. To utilize this function, the dplyr library must first be loaded into your current session. Alternatively, you can achieve the same result using a combination of traditional Base R functions, providing flexibility depending on your specific coding environment and dependencies.

Introduction: The Importance of Counting Distinct Values in R

In data science, understanding the composition and quality of your dataset is paramount. One key metric is the count of unique values, often referred to as **cardinality**, within a column of an R data frame. High cardinality suggests many unique entries, while low cardinality implies repetition, which is critical information when preparing data for modeling or visualization. R provides robust tools that allow analysts to quickly determine this count without manual iteration, making the exploration phase significantly more efficient. This guide explores the two most common and reliable methods--one relying purely on Base R structures and another utilizing the streamlined functionality of the dplyr package.

Although both approaches yield identical results for the unique count, they differ significantly in syntax, required dependencies, and sometimes, performance, especially when dealing with extremely large datasets. The Base R method is ideal for environments where external packages might be restricted, or if you prefer minimizing dependencies. Conversely, the dplyr approach, featuring the dedicated n_distinct() function, is often preferred in modern R workflows due to its readability and seamless integration with the Tidyverse ecosystem. Choosing the right method depends on project requirements and personal preference, but familiarity with both ensures maximum adaptability across various analytical contexts.

Understanding the Core Concepts: Distinctness and Data Frames

A data frame in R is fundamentally a list of vectors of equal length, organized into columns. When we talk about finding the **unique values** in a column, we are identifying the set of elements that appear at least once, discarding all duplicates. For example, if a column contains the sequence (A, A, B, C, C, C, D), the set of unique values is (A, B, C, D), and the unique count is four. This initial step of identifying distinct values is prerequisite to counting them, and both major methodologies follow this two-step logic, though they abstract the process differently for user convenience.

The standard methods we will explore are structured around this concept: first, isolating the unique elements, and second, determining the total number of items in that resulting set. The two primary methods available to analysts in R are highly robust and widely utilized, providing flexibility for different programming styles and requirements, especially when balancing performance and code transparency.

You can use the following methods to count the number of unique values in a column of a data frame in R:

Method 1: Leveraging Base R Functions for Uniqueness Counting

Method 1: Using Base R

The Base R approach is highly reliable and relies on a simple yet effective chaining of two core functions: unique() and **length()**. The unique() function, available in Base R, takes a vector as input (a column from your data frame) and returns a new vector containing only the distinct elements. Subsequently, the **length()** function is applied to this newly created vector to calculate the total number of elements it contains, which is precisely the count of unique values.

This method is computationally efficient for most common data sizes and does not require loading any external packages, which can slightly improve startup time in scripts and reduce external dependencies. The syntax is concise and immediately communicates the intent: "find the unique elements, then count how many there are." This is particularly useful for maintaining backward compatibility or working in environments with strict package limitations.

length(unique(df$my_column))

Method 2: Utilizing the Powerful dplyr Package

Method 2: Using dplyr

For those working within the Tidyverse framework, the dplyr package offers a dedicated and highly optimized function for this task: n_distinct(). This function provides a more idiomatic and readable solution within the dplyr ecosystem, as it directly solves the problem of counting distinct elements without requiring explicit chaining of **unique()** and **length()**. This unified approach abstracts away the underlying separation of finding and counting, offering superior code clarity.

The primary functional difference is that n_distinct() handles the entire operation internally, often utilizing highly optimized C++ code (via the underlying `Rcpp` integration) for improved speed and resource management. This often translates to slightly better performance and reduced memory overhead for very large vectors compared to the two-step Base R method, especially when dealing

with high cardinality. Before using n_distinct(), the dplyr library must be explicitly loaded into the session using the **library()** command.

library(dplyr)

n_distinct(df$my_column)

Practical Demonstration: Setting Up the Sample Data Frame

To illustrate both counting methodologies effectively, we will utilize a small, representative data frame. This example dataset simulates a scenario involving team performance metrics, allowing us to accurately calculate the unique counts for both categorical (`team`) and numerical (`points`) variables. Understanding how these functions behave on different data types is crucial for reliable analysis in R, as the underlying mechanisms handle character and numeric vectors slightly differently, though the results are equivalent.

The data frame, named `df`, contains nine total entries. Notice that both columns include repeated values: Team 'A' appears four times, 'C' appears twice, and points 14 and 20 both appear twice. We need to use distinct value counting techniques to determine the actual number of unique teams participating and the variety of scores achieved, providing essential summary statistics for the dataset.

The following code creates and displays our sample data structure in the R console, which serves as the foundation for the subsequent method applications:

#create data frame
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'C', 'C', 'D'),
points=c(10, 13, 14, 14, 18, 19, 20, 20, 22))

#view data frame
df

team points
1 A 10
2 A 13
3 A 14
4 A 14
5 B 18
6 B 19
7 C 20
8 C 20

9 D 22

Method 1: Detailed Application of Base R for Specific Columns

Once the data frame is loaded, we can immediately apply the Base R method. We will focus initially on the `points` column. Since the column has repeated scores, our objective is to determine how many different score values exist within the dataset, ignoring the frequency with which each score appears. This is achieved by extracting the column as a vector, processing it with unique(), and counting the result with **length()**.

The implementation involves extracting the column using the `$` operator (i.e., `df$points`), passing this vector to the unique() function to filter out duplicates, and finally wrapping the entire expression in **length()**. This sequence of operations is the standard procedure for count distinct operations using core R functions, yielding a numeric scalar result.

The following code shows how to count the number of unique values in the **points** column of the data frame using functions from Base R:

#count unique values in points column
length(unique(df$points))

7

The output confirms that there are **7 unique values** recorded in the **points** column. This result is crucial for understanding the variability within the numerical scores and provides a quick summary statistic about this feature.

Base R Implementation: Counting Across All Columns Using sapply()

A common requirement in data auditing is generating a comprehensive summary of the cardinality for every column within a data frame. Base R facilitates this through the use of the powerful iteration function sapply(). The sapply() function is specifically designed to apply a specified function to every column of a data frame or element of a list, consolidating the results into a simplified vector or matrix output.

By combining sapply() with an anonymous function that executes our established `length(unique(x))` logic, we can efficiently calculate the unique count for both the `team` and `points` columns simultaneously. This approach is highly flexible and scalable for data frames containing numerous features, providing an instantaneous overview of data diversity.

To count the number of unique values in each column of the data frame, we use the sapply() function as shown below:

#count unique values in each column
sapply(df, function(x) length(unique(x)))

team points
4 7

From the output, we can derive the following conclusions regarding the dataset's unique composition:

There are **7 unique values** in the **points** column.
There are **4 unique values** in the **team** columm (A, B, C, D).

Method 2: Implementing dplyr's n_distinct() for Targeted Counting

The dplyr package provides the specialized function n_distinct(), which streamlines the unique counting process into a single, highly readable operation. While the result is identical to the Base R method, the syntax is often preferred in modern R scripting for its clarity and efficiency when integrated into piped operations (%>%).

This method requires loading the library first, but once loaded, we can apply n_distinct() directly to the specified column (`df$points`). The function handles the necessary steps of identifying and counting distinct elements in an optimized manner, providing a clean solution for calculating column cardinality. We apply this function to the `points` column to verify consistency with the Base R results.

The following code shows how to count the number of distinct values in the **points** column using the n_distinct() function from the dplyr package:

library(dplyr)

#count unique values in points column
n_distinct(df$points)

7

The result confirms that there are **7 unique values** in the **points** column. This successful validation demonstrates that, while the underlying implementation differs, both the Base R and dplyr methods are reliable tools for accurately calculating unique counts.

Advanced dplyr Application: Counting Across All Columns Using sapply() and n_distinct()

To obtain a column-wise unique count summary utilizing the optimization benefits of n_distinct(), we can once again leverage the iteration capabilities of sapply(). While dplyr offers specialized Tidyverse functions for this type of operation (such as `summarise` combined with `across`), using sapply() remains a foundational and cross-compatible technique for applying a single function across an entire data frame structure efficiently.

By replacing the internal `length(unique(x))` call from the previous section with `n_distinct(x)`, we seamlessly integrate the optimized counting mechanism provided by dplyr within the sapply() framework. This hybrid approach demonstrates the versatility of R, allowing users to combine high-performance package functions with traditional Base R iteration utilities to achieve comprehensive dataset summaries.

To count the number of unique values in each column of the data frame using n_distinct() within sapply(), we execute the following code:

library(dplyr)

#count unique values in each column
sapply(df, function(x) n_distinct(x))

team points
4 7

The output provides a clear summary of the cardinality for both features:

There are **7 unique values** in the **points** column.
There are **4 unique values** in the **team** columm.

Notice that these results match the ones derived from the pure Base R method, confirming the accuracy and interoperability of the two primary techniques for distinct value counting.