Learn How to Count Unique Values in R Data Frames Using dplyr

Authored by

Mohammed looti

November 2, 2025

Introduction to Distinct Value Counting in R

Counting the number of unique, or **distinct**, values within a dataset is a fundamental step in exploratory data analysis. This process helps analysts understand the cardinality of variables, which is essential for tasks like identifying potential primary keys, normalizing data, or calculating frequency distributions. In the statistical programming environment of R, the most efficient and readable way to perform this operation is by utilizing the powerful functions provided by the dplyr package, a cornerstone of the **Tidyverse** ecosystem.

Specifically, dplyr provides the dedicated function `n_distinct()`, which is optimized for calculating unique counts across various scopes within a data frame. This article will explore three essential methodologies for applying `n_distinct()`: counting uniqueness in a single column, automating the count across all columns, and performing advanced counts grouped by specific categories. Mastery of these techniques ensures efficient and transparent data wrangling in **R**.

You can use one of the following methods to count the number of distinct values in an **R** data frame using the `n_distinct()` function from **dplyr**:

Method 1: Count Distinct Values in One Column

```
n_distinct(df$column_name)
```

Method 2: Count Distinct Values in All Columns

```
sapply(df, function(x) n_distinct(x))
```

Method 3: Count Distinct Values by Group

```
df %>%
group_by(grouping_column) %>%
summarize(count_distinct = n_distinct(values_column))
```

Setting the Stage: The `dplyr` Package and Sample Data

Before diving into the specific counting techniques, it is necessary to load the required library and establish a sample data frame for demonstration. The dplyr package must be loaded first, as it contains the crucial functions such as `n_distinct()`, `group_by()`, and `summarize()` that facilitate these operations. We will use a small dataset representing sports team statistics to illustrate the practical application of each method.

The following examples show how to use each of these methods in practice with the following data frame. Note how the structure includes character variables (`team`) and numeric variables (`points` and `assists`), which allows us to explore distinct counts across different data types.

library(dplyr)

#create data frame
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
points=c(6, 6, 8, 10, 9, 9, 12, 12),
assists=c(3, 6, 4, 2, 4, 5, 5, 9))

#view data frame
df

team points assists
1 A 6 3
2 A 6 6
3 A 8 4
4 A 10 2
5 B 9 4
6 B 9 5
7 B 12 5
8 B 12 9

This dataset, consisting of eight observations, serves as our foundation. Our subsequent analyses will determine the unique occurrences within the `team`, `points`, and `assists` variables, demonstrating the versatility of the distinct counting functions in R.

Method 1: Counting Distinct Values in a Single Column

The simplest and most direct application of the core function is counting distinct values within a single specified variable. This method is particularly useful when you need to quickly verify the number of categories in a nominal or ordinal column, such as geographical regions, product codes, or, in our case, team identifiers. We access the specific column using the standard **R** dollar-sign notation (`df$column_name`) and pass it directly to the `n_distinct()` function.

The following code demonstrates how to use `n_distinct()` to count the number of distinct values solely in the `team` column of our sample data frame. This answers the question: "How many unique teams are represented in this dataset?"

#count distinct values in 'team' column

n_distinct(df$team)

2

The output `2` confirms that there are only **2** distinct values in the `team` column (Team 'A' and Team 'B'), which is a quick and effective way to confirm the scope of categorical variables. This fundamental application is the building block for more complex distinct counting tasks.

Method 2: Comprehensive Distinct Counts Across All Columns

When conducting initial exploratory analysis, analysts often require a summary of distinct values for every column simultaneously. Manually applying `n_distinct()` to each column becomes tedious and inefficient, especially in wide datasets containing dozens or hundreds of variables. To automate this process, we combine the power of `n_distinct()` with **R**'s base function `sapply()`.

The `sapply()` function applies a specified function (in this case, `n_distinct()`) across all elements (columns) of a list or data frame. This combination allows for rapid assessment of variable cardinality across the entire dataset, providing a crucial overview of data structure and quality. The result is a named vector where each element corresponds to a column and its associated count of unique values.

The following code demonstrates how to use the `sapply()` and `n_distinct()` functions to count the number of distinct values in each column of the data frame:

#count distinct values in every column
sapply(df, function(x) n_distinct(x))

team points assists
2 5 6

The output vector provides a clear, concise summary of the cardinality for all variables within the data frame. From this comprehensive output, we can immediately deduce the following structural information:

There are **2** distinct values in the `team` column, confirming our previous finding.
There are **5** distinct values in the `points` column, indicating a moderate spread of scores.
There are **6** distinct values in the `assists` column, suggesting that most observations have unique assist totals.

Method 3: Grouped Distinct Counts (Advanced Analysis)

One of the most powerful features of the dplyr package is its ability to perform operations conditionally, based on groups defined within the data. Counting distinct values within specified groups is essential for comparative analysis--for instance, determining how many unique point scores were achieved by Team A versus Team B. This requires leveraging the pipe operator (`%>%`) in conjunction with `group_by()` and `summarize()`.

First, the `group_by()` function partitions the data frame based on the specified grouping variable (`team`). Then, `summarize()` applies an aggregation function (`n_distinct()`) to the desired value column (`points`) within each partition, generating a compact summary table. This workflow adheres to the principles of tidy data manipulation, leading to highly readable and maintainable code.

The following code demonstrates how to use the `n_distinct()` function within a summary pipeline to count the number of distinct `points` values for each unique team:

```
#count distinct 'points' values by 'team'
df %>%
group_by(team) %>%
summarize(distinct_points = n_distinct(points))
```

```
# A tibble: 2 x 2
team distinct_points
1 A 3
2 B 2
```

The resulting tibble clearly separates the unique counts by team. This powerful grouping mechanism allows for granular insight into data variation across different segments of the population.

From the output, we can accurately observe the differences in scoring variety between the two teams:

There are **3** distinct points values recorded for team A (6, 8, and 10).
There are **2** distinct points values recorded for team B (9 and 12).

Why Distinct Counts Matter in Data Analysis

Understanding the count of distinct values is not merely an exercise in counting; it is a critical diagnostic tool in data science. High cardinality (a large number of distinct values relative to the

total number of rows) in a categorical variable often suggests that the variable may not be suitable for certain machine learning models or may require specialized handling, such as target encoding or feature hashing. Conversely, low cardinality can indicate a variable suitable for immediate use in grouping or factor analysis.

Furthermore, distinct counting is crucial for identifying data quality issues. If a column that is expected to contain identifiers (like unique IDs) has a distinct count less than the total number of rows, it signals the presence of duplicate records, which must be addressed before analysis can proceed. This simple metric provides immediate feedback on data integrity, guiding the subsequent steps in the data cleaning pipeline.

By integrating the `n_distinct()` function--whether applied universally, individually, or conditionally via `group_by()`--analysts gain immediate and actionable insights into the underlying structure and variety present within their datasets, significantly improving the efficiency and reliability of downstream statistical modeling.

Additional Resources

To further enhance your proficiency in data manipulation using the **Tidyverse**, consider exploring related tutorials that delve into other common data wrangling operations. Mastering functions adjacent to `n_distinct()`, such as `mutate()`, `filter()`, and `arrange()`, will solidify your foundation in powerful and idiomatic **R** programming.

The following tutorials explain how to perform other common operations using **dplyr**: