Calculate Mode by Group in R (With Examples)

Authored by

Mohammed looti

October 28, 2025

The **mode** of a dataset is a fundamental concept in descriptive statistics, representing the value that occurs most frequently within a distribution. While the mean and median provide insights into the central tendency based on sums and positional order, the mode uniquely identifies the most common observation. This measure is particularly useful when analyzing categorical or discrete data where averages might be misleading or impossible to calculate. Understanding the distribution's mode is critical for data analysts seeking to identify prevalent trends or common categories within their variables. Unlike the mean, which is sensitive to outliers, the mode remains robust, offering a stable measure of centrality even in skewed distributions.

Despite its importance, the core R statistical software environment does not include a direct, built-in function to calculate the mode, unlike functions for calculating the mean (`mean()`) or median (`median()`). This omission stems partly from the complexity of handling datasets that might be multimodal (having two or more modes) or continuous data where no exact repeat values exist. Therefore, to effectively calculate the mode--especially when needing to determine the mode for subgroups within a larger dataset--users must rely on creating a custom function or utilizing specialized packages that handle frequency counts. This requirement necessitates a clear understanding of how R processes vectors and counts unique occurrences, forming the groundwork for accurate statistical reporting when central tendency is defined by frequency.

The goal of this tutorial is to present a robust, user-defined function specifically tailored for identifying the mode, which can then be seamlessly integrated with powerful data manipulation tools like the dplyr package. By combining a custom frequency calculation logic with the grouping capabilities inherent in modern R data workflows, we can efficiently determine the mode for subsets of data--a task often referred to as calculating the **mode by group**. The following sections will first introduce the necessary R function, explain its internal mechanics, and then provide comprehensive, practical examples demonstrating how to apply this logic to real-world data scenarios, including those involving both single and multiple modes.

Defining the Custom Mode Function in R

Since R lacks a standard `mode()` function for statistical calculation, we must define our own. The custom function, typically named `find_mode`, leverages several core R capabilities to efficiently determine the most frequent element in any given vector. The logic revolves around identifying all unique elements, counting their occurrences, and then selecting the elements corresponding to the maximum count. This approach ensures that the function is flexible enough to handle both numeric and character vectors, maintaining high utility across different data types encountered in data analysis projects.

The implementation relies heavily on two primary R base functions: `unique()` and `tabulate()`. The `unique(x)` function first extracts all distinct values from the input vector `x`, storing them in a

new vector, `u`. This step is crucial because it establishes the complete set of potential modes. Following this, the `match(x, u)` function creates a vector of integers indicating the position of each element of `x` within the `u` vector. This vector of indices is then passed to the tabulate() function, which is optimized for counting occurrences of positive integers, providing a highly efficient way to count how many times each unique value appears in the original data.

The final step in the function uses the resulting tabulation to identify the maximum frequency. The code snippet `tab == max(tab)` generates a logical vector (TRUE/FALSE) where TRUE indicates the position(s) corresponding to the highest frequency count. By using this logical vector to subset the unique values vector `u`, the function returns only those value(s) that achieved the maximum frequency. Crucially, this structure inherently handles situations where multiple values share the highest frequency (i.e., multimodal distributions), returning all of them.

Here is the definition of the custom function that will be utilized throughout our examples for calculating the mode:

```
find_mode <- function(x) {
u <- unique(x)
tab <- tabulate(match(x, u))
u
}
```

This function forms the cornerstone of our approach. The following examples will demonstrate how this user-defined function can be seamlessly integrated into powerful data manipulation pipelines, specifically using the **dplyr** package, to calculate the mode conditioned on grouping variables within a data frame.

Example 1: Calculating the Unimodal Mode by Group (Single Mode)

For our first demonstration, we will examine a scenario where each group in our dataset possesses a single, clearly defined mode--a **unimodal distribution**. Consider a common analytical task: determining the most typical score achieved by members of different teams. We begin by constructing a sample R data frame that tracks the points scored by various basketball players assigned to Team A and Team B. This data frame serves as the input structure for our grouped analysis, allowing us to illustrate the efficiency of combining the custom mode function with grouping operations.

The initial step involves defining and viewing our dataset. We create a data frame named `df` containing two columns: `team` (the categorical grouping variable) and `points` (the numeric variable for which we want to find the mode). Notice that in this specific setup, we have ensured that for

each team, only one 'points' value occurs most frequently. This setup allows us to confirm that the function correctly identifies the sole modal value for each subgroup.

#define data frame
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
points=c(5, 7, 7, 9, 12, 12, 10, 14))

#view data frame
df

team points
1 A 5
2 A 7
3 A 7
4 A 9
5 B 12
6 B 12
7 B 10
8 B 14

To perform the grouped mode calculation, we utilize the powerful `dplyr` library, which is essential for modern data manipulation in R. We pipe the data frame `df` into the `group_by()` function, specifying `team` as our grouping variable. Subsequently, the `summarize()` function is used to apply our custom `find_mode()` function specifically to the `points` column within each defined group. This sequence of operations effectively isolates the data for Team A, calculates its mode, isolates the data for Team B, and calculates its mode, returning a concise summary table. This approach drastically simplifies complex calculations by allowing analysts to treat subsets of data independently yet within a single, coherent pipeline command.

library(dplyr)

#define function to calculate mode (re-included for self-containment)
find_mode <- function(x) {
u <- unique(x)
tab <- tabulate(match(x, u))
u
}

#calculate mode of 'points' by 'team' using grouped operations
df %>%
group_by(team) %>%

summarize(mode_points = find_mode(points))

A tibble: 2 x 2
team mode_points

1 A 7
2 B 12

The resulting output clearly demonstrates the power of grouped summarization. We observe that for Team A, the points value of **7** occurs twice, making it the mode. For Team B, the points value of **12** occurs twice, establishing it as the mode. The structure of the output, an R tibble, is clean and readily interpretable, confirming that our custom function successfully identified the unique highest frequency value within each defined group. The ease of applying this calculation across groups highlights why the **dplyr** syntax is preferred for complex aggregation tasks in R.

Example 2: Handling Multimodal Data in R (Multiple Modes)

A critical challenge in mode calculation arises when a dataset exhibits a multimodal distribution, meaning two or more values share the highest frequency count. Unlike the mean or median, which always yield a single value, the mode must return all values that meet the criteria of maximum frequency. Fortunately, our custom `find_mode` function is inherently designed to handle this scenario gracefully, a feature we will now test by slightly altering our sample dataset. This example showcases the function's robustness when dealing with more complex data distributions, ensuring that no statistically relevant information is lost.

We define a new version of our data frame, `df`, where the scores for Team B have been adjusted such that two distinct point values occur with the same, highest frequency. Specifically, both 12 points and 10 points now appear twice for Team B. Team A remains unimodal for comparison. This revised dataset allows us to observe how the **dplyr** pipeline interacts with the `find_mode` function when it returns a vector containing multiple elements for a single group, which is a common occurrence in real-world data analysis.

#define data frame with multimodal distribution for Team B
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
points=c(5, 7, 7, 9, 12, 12, 10, 10))

#view data frame
df

team points
1 A 5

2 A 7
3 A 7
4 A 9
5 B 12
6 B 12
7 B 10
8 B 10

We apply the exact same R code structure used in Example 1. The custom function is called within the `summarize()` step after grouping by `team`. When the `find_mode` function executes on the data subset for Team B, it identifies that both 12 and 10 have the maximum frequency (two occurrences each), and thus, the function returns the vector `c(12, 10)`. The `dplyr::summarize` function automatically expands the output to accommodate the multiple results when a summarizing function returns more than one value per group, producing a highly informative output that correctly captures the bimodal nature of Team B's scores.

library(dplyr)

#define function to calculate mode
find_mode <- function(x) {
u <- unique(x)
tab <- tabulate(match(x, u))
u
}

#calculate mode of 'points' by 'team'
df %>%
group_by(team) %>%
summarize(mode_points = find_mode(points))

A tibble: 3 x 2
Groups: team
team mode_points

1 A 7
2 B 12
3 B 10

The results confirm successful handling of multimodal data. For Team A, the mode is still uniquely identified as **7**. However, for Team B, the output now spans two rows (rows 2 and 3), correctly

reporting both **12** and **10** as the modes. In this scenario, there were two point values that occurred most frequently for Team B, so each of these mode values is returned on a separate line for Team B in the output, demonstrating the necessary flexibility of the R environment when dealing with complex statistical measures.

Best Practices and Limitations of Mode Calculation

While the custom function `find_mode` provides a robust solution for calculating the mode in R, particularly when grouped, analysts must be mindful of its limitations and specific application contexts. The mode is most meaningful for discrete or categorical variables. Applying this function to truly continuous data (e.g., measurements taken with high precision, like temperature readings to several decimal places) will likely result in every value being unique, leading the function to potentially return the entire dataset as the "mode" or result in highly unstable mode estimates. In such continuous scenarios, it is generally preferable to use techniques like kernel density estimation to identify peaks in the distribution, rather than relying on exact frequency counts.

Furthermore, the computational efficiency of this function, while high due to the use of `tabulate`, can still be a consideration for extremely large datasets. The steps involve sorting and matching unique values, which scales reasonably well but can consume memory if the input vector contains millions of unique items. For performance optimization in massive datasets, developers might explore specialized packages or vectorized alternatives that bypass the need for explicit loops, though for typical corporate or academic data sizes, the provided `find_mode` function is more than adequate and highly readable.

It is also important to consider data preparation. Before calculating the mode, ensure that data types are consistent within the vector. For instance, if numerical scores are mistakenly stored as character strings, the mode calculation will treat '10' and '10.0' as distinct categories, potentially leading to incorrect statistical summaries. Adhering to strong data management practices, such as the Tidy Data principles--where each column is a variable and each row an observation--is paramount for ensuring that grouped operations via **dplyr** function correctly and yield meaningful results based on the defined grouping variable.

Conclusion and Further Resources

Calculating the mode by group in R, while requiring a user-defined function due to the lack of a built-in base R utility, is easily achieved through the combination of custom logic and the powerful grouping capabilities of the **dplyr** package. The `find_mode` function presented here offers a flexible and reliable method for identifying the most frequent value(s) in any vector, capably handling both unimodal and multimodal distributions. Integrating this function into the `group_by()` and `summarize()` pipeline allows data analysts to quickly extract essential descriptive statistics

broken down by key categorical variables, providing deeper insights into data concentration.

The examples demonstrated how clean, interpretable results are produced, regardless of whether a single mode or multiple modes exist within a group. This methodology is fundamental for deriving meaningful insights from data where frequency, rather than arithmetic average, is the most appropriate measure of central tendency. Mastering this technique empowers R users to move beyond standard arithmetic means and provide a more complete picture of their data distributions, which is especially vital in exploratory data analysis.

From the results of the multimodal example, we can clearly see:

The mode of points for team A is **7**.
The mode of points for team B is **12** and **10**.

The following tutorials explain how to calculate other descriptive statistics in R: