How to Easily Remove Columns and Rows from Data Frames in R

Authored by

stats writer

November 19, 2025

The **drop()** function in the R programming environment is a crucial tool for simplifying the structure of multi-dimensional objects such as arrays and matrices. When subsetting these structures, it allows the removal of dimensions that have only a single level (singleton dimensions), streamlining the object for further analysis. While dimension reduction often occurs during subsetting of multi-dimensional structures, calling **drop()** explicitly ensures this simplification is applied immediately to the object.

It is important to clarify that **drop()**, in its base R form, operates fundamentally on the dimensions of arrays and matrices. Its core purpose is to prevent objects that contain essentially one-dimensional data from retaining higher-dimensional attributes, such as converting a 1xN matrix into a standard vector. This capability is essential for managing the object class consistency required by various analytical routines.

The Core Mechanism of the drop() Function

The **drop()** function in base R is primarily designed to delete dimensions of an array or matrix that only have one level. This functionality is critical when subsetting operations inadvertently preserve dimensions that hold no meaningful structural information. For instance, if you define a data structure that is theoretically three-dimensional but the extent of one dimension is restricted to 1, **drop()** eliminates this redundant dimension, resulting in a more concise, two-dimensional structure.

The underlying principle behind **drop()** relates to R's generalized method for handling multi-dimensional structures. When we extract data, R must decide whether to retain the original object class (e.g., keep the result a matrix) or simplify it (e.g., turn it into a vector). Explicitly calling **drop()** enforces this simplification. This is particularly useful when dealing with custom functions or legacy code where structural assumptions about the data type (vector vs. matrix) are strict.

While the effect of **drop()** is often achieved by setting the `drop = TRUE` argument within square bracket subsetting (e.g., `x`), using the standalone function is necessary when the object creation or modification process has already occurred, leaving the object with unwanted singleton dimensions that need retrospective correction. The following examples demonstrate how this function structurally cleans up both arrays and matrices.

Example 1: Using drop() to Simplify a Multi-Dimensional Array

Suppose we have a 3-dimensional array in R where the dimensions are set to (1, 2, 5). This means the first dimension (row index) has only one level, making it a singleton dimension suitable for removal. We initialize the array with 10 sequential values and assign these specific dimensions to illustrate the starting state of the data structure.

#create 3-dimensional array
my_array <- c(1:10)
dim(my_array) <- c(1,2,5)

#view array
my_array

, , 1

1 2

, , 2

3 4

, , 3

5 6

, , 4

7 8

, , 5

9 10

As shown in the output, the object retains three dimensions, even though the first dimension is always indexed at . This is computationally inefficient and structurally misleading. Our objective is to reduce this object to a two-dimensional structure (a 2x5 matrix) while maintaining the original data sequence. This simplification is performed easily using the **drop()** function.

Analyzing the Array Dimension Reduction

We apply the **drop()** function directly to `my_array`. The function internally checks the size of each dimension and removes any dimension whose extent is 1, resulting in `new_array`, a structurally sound, lower-dimensional object.

#drop dimensions with only one level
new_array <- drop(my_array)

#view new array
new_array

1 3 5 7 9
2 4 6 8 10

Notice the immediate structural change: the data is now laid out in a conventional two-dimensional grid (2 rows by 5 columns). We can confirm the success of the dimension reduction using the **dim()** function, which retrieves the current dimension attributes of the object.

#view dimensions of new array
dim(new_array)

2 5

The output `2 5` confirms that the object has successfully transitioned from a 3D structure to a 2D structure, achieving the intended simplification by removing the singleton dimension. This is a powerful demonstration of how **drop()** cleans up object structures derived from complex initializations or intermediate steps in data processing.

Example 2: Applying drop() to Convert a Matrix to a Vector

A frequent use case for **drop()** involves converting a matrix that contains only a single row or column into a standard one-dimensional vector. We begin by creating a simple matrix with one row and seven columns, and confirm its dimensions.

#create matrix
my_matrix <- matrix(1:7, ncol=7)

#view matrix
my_matrix

1 2 3 4 5 6 7

#view dimensions of matrix
dim(my_matrix)

1 7

The matrix structure (1 row, 7 columns) clearly shows a singleton dimension (the row dimension). Applying **drop()** here will enforce the dimensional reduction, removing the row dimension and coercing the matrix class object into a standard numeric vector.

#drop dimensions with only one level

new_matrix <- drop(my_matrix)

#view new matrix (now a vector)
new_matrix

1 2 3 4 5 6 7

The resulting object, `new_matrix`, is now displayed as a list of elements, confirming its transformation into a vector. This change of class is vital in R, as functions expecting a vector might fail or produce unexpected results if fed a 1xN matrix instead.

Understanding the Null Dimension Result

After the application of **drop()** in the matrix example, if we try to inspect the dimensions of the resulting vector using **dim()**, it returns `NULL`. This occurs because the object is no longer an array or matrix; it has been reduced to a simple vector, which inherently lacks the `dim` attribute.

#view dimensions of new matrix
dim(new_matrix)

NULL

Since we cannot use **dim()** to determine the size of the new object, we rely on **length()**. The **length()** function accurately reports the count of elements within any one-dimensional object in R, including vectors, providing the correct size information without requiring dimensional attributes.

#view length
length(new_matrix)

7

This result confirms that, while the structure and class of the data have changed (from matrix to vector), the integrity of the data--seven elements--is preserved. This distinction between **dim()** (for arrays/matrices) and **length()** (for vectors) is fundamental when dealing with dimension manipulation in R.

When to Use drop() vs. Subsetting Options

While **drop()** can be applied explicitly, its behavior is most frequently encountered when subsetting data structures. When extracting data from a data frame or matrix, R allows the user to specify

whether singleton dimensions should be retained or dropped using the `drop` argument within the subsetting brackets.

Matrix Subsetting: If you extract a single column from a matrix using `M`, R usually defaults to `drop = TRUE`, returning a vector. Setting `drop = FALSE` (e.g., `M`) preserves the dimension, resulting in a N x 1 matrix.
Data Frame Subsetting: When extracting a single column from a data frame using single brackets (`df`), R defaults to `drop = TRUE`, which returns a vector. Using `drop = FALSE` (or using double brackets `df]`) preserves the column as a single-column data frame, which is essential if downstream functions require a data frame input.

Using the standalone **drop()** function is recommended when you need to ensure dimension reduction on an object that has already been created or passed as an argument, and you need to verify its lowest possible dimension regardless of how it was originally generated. It acts as a safety measure for structural normalization.