How to delete multiple columns in R?

Authored by

stats writer

December 20, 2025

The process of manipulating data frame objects is central to data analysis in R. While adding or modifying columns is straightforward, removing one or more columns efficiently is a common task that requires specific knowledge of R's indexing capabilities. This guide provides a comprehensive, expert-level overview of the most robust methods available in R for deleting multiple columns simultaneously, focusing on both traditional **Base R** approaches and modern **Tidyverse** solutions. We will ensure that the resulting code is clean, efficient, and highly reproducible, covering methods based on column names, numeric indices, and ranges.

Historically, R provides several fundamental ways to achieve rapid column deletion. One of the cleanest and most direct methods involves assigning the value `NULL` to the desired columns using list-style assignment. This technique is highly optimized for performance and is particularly useful when working within existing **Base R** scripts where minimizing external package dependency is crucial. Understanding these core Base R mechanisms ensures that you can handle complex data manipulation tasks reliably, regardless of the environment or available package set.

When dealing with large datasets or complex data preparation pipelines, the ability to quickly and accurately remove irrelevant variables is paramount to maintaining computational efficiency and script clarity. Whether you choose to delete columns based on their descriptive names, which is the preferred method for stability, or based on their numerical position, mastering these fundamental deletion mechanisms is essential for any proficient R user. The following sections will provide step-by-step guidance and detailed code examples demonstrating column removal using various techniques, emphasizing the importance of choosing the right method for the job.

The Principle of Column Deletion in Base R

In many analytical workflows, data analysts frequently encounter the need to streamline their data frame structures by eliminating columns that contain redundant information, excessive missing values, or variables that are simply not relevant for the current model or visualization. While using traditional subsetting functions might create a new data frame excluding the unwanted columns, the most idiomatic and memory-efficient method in **Base R** for truly deleting columns in place is the strategic assignment of `NULL`. This mechanism effectively removes the column reference and associated data from the object, saving memory and speeding up subsequent processing.

The standard convention for deleting multiple columns relies on array indexing combined with the `list(NULL)` assignment operator. When working with a two-dimensional object like a **data frame**, R expects two primary arguments within the square brackets: . To target specific columns for deletion, we deliberately leave the row argument empty (e.g., `df`), which indicates that the operation should apply across all rows. We then supply a vector of column identifiers--either names or indices--in the column slot. By setting this targeted column subset equal to `list(NULL)`, R handles the necessary internal memory adjustments to detach those columns permanently from

the data object.

It is important to understand why `list(NULL)` is used instead of just `NULL`. When deleting a single column, `df$column <- NULL` or `df <- NULL` works perfectly. However, when attempting to delete multiple columns simultaneously, the assignment mechanism changes. If you attempt to assign a single `NULL` value to a subset of multiple columns, R throws an error because it expects the replacement value to match the structure of the subset. By wrapping `NULL` within `list()`, we signal to R that we are performing a list-style assignment of replacement items, where `NULL` signifies removal, allowing the operation to execute correctly across the vectorized column names or indices.

Method 1: Deleting Columns by Name using Base R Indexing

The most robust and highly recommended method for removing columns involves referencing them directly by their descriptive names. This approach offers stability, guaranteeing that you are deleting the correct variable regardless of its dynamic position within the data structure. Using column names ensures the stability and long-term readability of your scripts, particularly when handling data sources that might introduce column reordering or minor structural updates over time. The fundamental requirement for this technique is supplying a character vector formatted as `c('name1', 'name2')` within the column index position and assigning `list(NULL)` to trigger deletion.

The generalized syntax for removing multiple columns based on their unique identifier names is concise and powerful:

df <- list(NULL)

This syntax explicitly directs R to target the columns specified within the character vector across all rows (signified by the empty space before the comma) and eliminate them through the assignment of `list(NULL)`. This method bypasses the need for knowledge of column order, making the code highly resilient to changes in the underlying data structure. This is critical for ensuring that data pipelines operate consistently over time, even as data schema evolves.

To demonstrate this in practice, consider a scenario where we initialize a simple four-variable data frame and subsequently need to remove two specific columns, `var2` and `var3`. The following example illustrates the entire process, including the creation of the sample data and the immediate, effective removal of the unwanted variables using their names:

#create data frame
df <- data.frame(var1=c(1, 3, 2, 9, 5),
var2=c(7, 7, 8, 3, 2),

var3=c(3, 3, 6, 6, 8),
var4=c(1, 1, 2, 8, 7))

#delete columns 2 and 3 from data frame (using names)
df <- list(NULL)

#view data frame
df

var1 var4
1 1 1
2 3 1
3 2 2
4 9 8
5 5 7

As confirmed by the output, the columns `var2` and `var3` are successfully purged from the data frame, leaving only `var1` and `var4`. This technique forms the cornerstone of reliable and maintainable data cleaning practices within Base R, providing unparalleled clarity and positional independence to data manipulation code.

Method 2: Deleting Columns by Numeric Index (Positional Removal)

Although using column names is generally preferable for scripting stability, there are specialized contexts, particularly in dynamic data preparation or brief exploratory analysis, where deleting columns based on their numerical position (index) is necessary. This approach capitalizes on R's positional indexing system, where the first column is index 1, the second is 2, and so on. To remove multiple columns by position, we substitute the character vector of names with a numeric vector of positions within the column index slot.

It is crucial to highlight the potential pitfalls of relying solely on numeric indexing. Should the data structure change--for instance, if an unexpected column is inserted at the beginning of the data frame--all subsequent indices will shift. This structural dependency means that your script could inadvertently delete the wrong variables without warning, leading to severe data integrity issues. Consequently, this method should only be used when the data structure is guaranteed to be static or when indices are calculated dynamically based on content.

To demonstrate the mechanics of positional deletion, we apply the same **Base R** syntax structure but input a numeric vector `c(2, 3)` to target columns 2 and 3:

#create data frame

df <- data.frame(var1=c(1, 3, 2, 9, 5),
var2=c(7, 7, 8, 3, 2),
var3=c(3, 3, 6, 6, 8),
var4=c(1, 1, 2, 8, 7))

#delete columns in position 2 and 3
df <- list(NULL)

#view data frame
df

var1 var4
1 1 1
2 3 1
3 2 2
4 9 8
5 5 7

The resulting data frame is identical to the one produced using the named method, confirming that the underlying assignment functionality is consistent across both indexing paradigms. Regardless of whether you use names or indices, the crucial and consistent element is the assignment of `list(NULL)` to the selected subset of columns, which signals to the R interpreter the intention of permanent removal from the data object.

Method 3: Efficiently Removing Columns in a Sequential Range

A frequent requirement in data processing is the removal of a contiguous block of columns, often representing intermediate calculation steps or metadata grouped together. Instead of laboriously listing every individual column index, R provides a highly efficient mechanism using the colon operator (`:`) to specify a range of column positions slated for deletion. This approach significantly simplifies the code when bulk removal of adjacent columns is required.

The range operator is particularly useful when dealing with data structures where columns are naturally ordered, such as time series data where several features follow one another sequentially. For example, if a data frame has thirty variables and you need to discard variables 10 through 25, the syntax `10:25` is vastly superior in terms of efficiency and code footprint compared to writing out sixteen separate indices. This method leverages R's vectorization capabilities to apply the deletion across the entire specified sequence.

To illustrate the range removal technique, we will use our existing four-variable data frame and demonstrate how to delete the first three columns--indices 1 through 3--using the range operator

within the column indexing slot:

```
#create data frame
df <- data.frame(var1=c(1, 3, 2, 9, 5),
var2=c(7, 7, 8, 3, 2),
var3=c(3, 3, 6, 6, 8),
var4=c(1, 1, 2, 8, 7))

#delete columns in range 1 through 3
df <- list(NULL)

#view data frame
df

var4
1 1
2 1
3 2
4 8
5 7
```

In this final example, `var1`, `var2`, and `var3` were all deleted because they collectively occupied the range 1:3, leaving only `var4`. While highly efficient, this range-based removal method is the most sensitive to data reordering, as it relies completely on static positional information. It should be reserved for situations where the column order is guaranteed not to change or when performing temporary, localized data cleaning.

Alternative Approach: Utilizing the Tidyverse Package dplyr

For users engaged in modern R development, particularly within data science contexts, the use of packages from the Tidyverse, such as dplyr, offers a profoundly readable and powerful framework for column manipulation. The `select()` function within dplyr is the recommended tool for managing columns. Unlike the **Base R** method of assignment, `select()` operates by creating a new data frame that includes only the specified columns, or conversely, excludes those marked for dropping.

The primary advantage of using `dplyr::select()` lies in its expressive syntax and rich collection of selection helpers. To achieve column deletion (exclusion), we use the concise minus sign (`-`) preceding the column name or index. This approach is often considered more intuitive and semantically clear than the **Base R** assignment method for exclusion tasks, and it integrates seamlessly into modern data pipelines using the pipe operator (`%>%`).

To delete columns `var2` and `var3` using dplyr, the code is highly succinct:

```
# Assuming dplyr is loaded: library(dplyr)

df_new <- df %>%
select(-var2, -var3)
```

Beyond simple named exclusion, `select()` provides powerful utilities for dynamic column management. Functions like `starts_with()`, `ends_with()`, `contains()`, and `matches()` allow analysts to delete columns based on regular expression patterns or partial name matches. For example, to remove all columns starting with "ID_", one would simply use `select(-starts_with("ID_"))`. This level of flexibility and readability makes **dplyr** the preferred choice for complex and reproducible data wrangling operations.

Summary and Key Recommendations

In conclusion, the effective deletion of multiple columns from an R **data frame** can be achieved through highly optimized methods in both the foundational **Base R** environment and the popular Tidyverse ecosystem. The traditional **Base R** approach relies on the assignment of `list(NULL)` to the subset of columns identified by either name, specific index positions, or a continuous index range. While all methods are functional, the deletion by column name is fundamentally superior for ensuring long-term code stability and maximizing reproducibility.

For analysts prioritizing clarity, speed, and integration into existing data pipelines, the `select()` function from dplyr offers an extremely expressive alternative, utilizing the clean minus sign (`-`) syntax for column exclusion. Regardless of the framework chosen, mastering these column deletion techniques is a mandatory skill for any practitioner performing serious data preparation in R. Always default to column names for deletions unless there is a compelling, specific reason to utilize positional indices.

Mastering column deletion is just one component of effective data manipulation. For those looking to expand their R toolkit and explore related advanced techniques, here are resources covering common data wrangling tasks:

How to Loop Through Column Names in R
How to Combine Two Columns into One in R
How to Remove Outliers from Multiple Columns in R