Learning to Rename Multiple Columns in R with dplyr

Authored by

Mohammed looti

October 28, 2025

1. Introduction to Efficient Column Renaming with dplyr

Effective data management often requires precise **data wrangling**, and one of the most common tasks analysts face is renaming columns within a **data frame**. While base R offers methods for this purpose, the **dplyr** package, a core component of the Tidyverse, provides streamlined and highly readable functions specifically designed for this operation. This tutorial explores two powerful and flexible functions from **dplyr**--`rename()` and `rename_with()`--demonstrating how to efficiently handle the renaming of multiple columns simultaneously.

Choosing the correct function depends largely on the scale and complexity of the renaming task. For simple, one-off changes where you know both the old and new names explicitly, `rename()` offers a concise syntax. However, when dealing with larger datasets requiring programmatic renaming or the application of specific naming conventions across several columns, `rename_with()` provides unparalleled flexibility. Understanding both approaches ensures that you can select the most efficient method for any given scenario, significantly improving the clarity and maintainability of your R scripts.

2. Quick Overview of dplyr Renaming Syntax

The **dplyr** package is integral to modern data analysis in R, providing a consistent grammar for common data manipulation verbs. Both renaming methods utilize the forward **pipe operator (%>%)**, which sequentially passes the data frame to the renaming function, promoting a clean, step-by-step workflow. Here is a brief look at the fundamental syntax for renaming multiple columns using these two functions.

The `rename()` function is perhaps the most straightforward. It operates by specifying the new column name followed by an equals sign and then the existing column name, allowing you to define multiple pairs within a single function call. This is ideal when you have a handful of specific columns whose names need manual correction or adjustment.

Method 1: Use rename() for Direct Mapping

df %>% rename(new1 = old1, new2 = old2)

Conversely, `rename_with()` excels when you need to rename columns based on a vector of replacement names or when applying a function to the existing names (e.g., converting names to lowercase, adding prefixes). When mapping specific old names to specific new names, as shown below, you supply the vector of new names and use the `all_of()` helper function to identify the columns to be replaced.

Method 2: Use rename_with() for Programmatic Control

```
new <- c('new1', 'new2')
old <- c('old1', 'old2')

df %>% rename_with(~ new, all_of(old))
```

It is important to note that, despite their distinct syntax structures, both `rename()` and `rename_with()` are capable of producing the exact same result when applied correctly. The choice between them often comes down to personal preference or the complexity of the task at hand.

3. Setting Up the Example Data Frame

To demonstrate these powerful renaming techniques, we will work with a simple sports statistics **data frame**. This example dataset, named `df`, tracks the performance metrics--team identifier, points scored, and assists--for several fictional teams. Before proceeding with the renaming examples, you must ensure the **dplyr** package is installed and loaded into your R environment.

The creation of the sample data frame is straightforward using the base R `data.frame()` function. This initial structure provides clear column names (`team`, `points`, `assists`) which we will subsequently modify to demonstrate how `rename()` and `rename_with()` handle specific column selections.

The following code snippet shows the creation and structure of our initial dataset. We aim to rename the `team` and `points` columns to reflect a revised naming convention, perhaps adding a suffix to indicate they are raw scores.

```
#create data frame
df <- data.frame(team=c('A', 'B', 'C', 'D', 'E'),
points=c(22, 34, 30, 12, 18),
assists=c(7, 9, 9, 12, 14))

#view data frame
df

team points assists
1 A 22 7
2 B 34 9
3 C 30 9
4 D 12 12
5 E 18 14
```

4. Example 1: Renaming Multiple Columns Using `rename()`

The `rename()` function is the preferred tool when you are renaming a small, specific number of columns. Its syntax is highly readable and mimics a simple assignment operation, mapping the desired new name directly to the existing old name. This method is particularly efficient for correcting typos or standardizing names when the mapping is static and manually defined.

In this specific example, we intend to change `team` to `team_new` and `points` to `points_new`. Notice how the structure places the desired output name on the left side of the equals sign and the current column name on the right. This convention ensures that the analyst clearly defines the transformation outcome. We use the **pipe operator (%>%)** to feed `df` directly into the `rename()` function, creating the new data frame `df2` without modifying the original.

The following code demonstrates the practical application of `rename()`. Observe that the `assists` column, which was not explicitly mentioned in the function call, is automatically retained and remains unchanged in the resulting data frame, illustrating that `rename()` only affects the specified columns.

library(dplyr)

#rename team and points columns
df2 <- df %>% rename(team_new = team, points_new = points)

#view updated data frame
df2

team_new points_new assists
1 A 22 7
2 B 34 9
3 C 30 9
4 D 12 12
5 E 18 14

As expected, the **team** and **points** columns have been successfully renamed to `team_new` and `points_new`, respectively, while the **assists** column has remained precisely the same, confirming the targeted nature of the `rename()` operation.

5. Example 2: Renaming Multiple Columns Using `rename_with()`

While `rename()` is excellent for manual mapping, the `rename_with()` function offers a more dynamic, programmatic approach to column renaming. This function is particularly valuable when

the number of columns to be renamed is extensive or when the new names are generated based on existing data or external lists. It requires defining the list of new names and the list of old names separately, which enhances code organization when dealing with complex **data wrangling** tasks.

To achieve the same result as in Example 1--renaming `team` and `points` to `team_new` and `points_new`--we first create two character vectors: `new`, containing the desired names, and `old`, containing the current names. It is critical that the order of names in both vectors aligns perfectly, as `rename_with()` relies on this positional mapping to correctly apply the transformations. We then use `all_of(old)` within `rename_with()` to select exactly those columns listed in the `old` vector. The `~ new` syntax specifies that the column names selected by `all_of(old)` should be replaced by the values contained within the `new` vector, in order.

This method is incredibly scalable. Imagine having fifty columns to rename; defining the mappings in two external vectors is far cleaner and easier to manage than listing fifty pairs within a single `rename()` call. The following code illustrates this programmatic method:

library(dplyr)

#define new names
new <- c('team_new', 'points_new')

#define old names to replace
old <- c('team', 'points')

#rename old names with new names
df2 <- df %>% rename_with(~ new, all_of(old))

#view updated data frame
df2

team_new points_new assists
1 A 22 7
2 B 34 9
3 C 30 9
4 D 12 12
5 E 18 14

Just like with the `rename()` function, the **team** and **points** columns have been successfully updated, and the **assists** column remains untouched. The primary takeaway here is the increased organizational capacity that `rename_with()` provides, particularly beneficial for complex or routine renaming tasks where lists of names are defined dynamically.

6. Summary: When to Use `rename()` vs. `rename_with()`

Both `rename()` and `rename_with()` are invaluable tools in the **dplyr** toolkit for column renaming, but they serve slightly different niches based on the required level of specificity and automation. Analysts should choose the function that maximizes code clarity and efficiency for the specific task at hand.

Use `rename()` when: The mapping from old names to new names is explicit, manual, and involves only a few columns. This function is perfect for simple standardization tasks or correcting errors discovered during initial **data wrangling**. Its syntax is direct: `new_name = old_name`.

Use `rename_with()` when: You need to apply a function to column names (e.g., lowercase conversion, adding a suffix to all columns matching a pattern) or when the list of columns to be renamed is extensive and is managed via external vectors. This function allows for sophisticated, programmatic control over column names, making large-scale data preparation more manageable.

Ultimately, mastering both functions ensures that you are equipped to handle any column renaming challenge within R using the efficient and consistent grammar provided by the **dplyr** package.

7. Additional Resources for dplyr Mastery

Renaming columns is just one aspect of effective data manipulation using the **dplyr** package. To further enhance your skills in data preparation and analysis within R, exploring other core functions is highly recommended. The following tutorials explain how to perform other common tasks using **dplyr**, such as filtering rows, selecting columns, and summarizing data.