# Learning Programmatic Column Renaming with rename_with() in R

Authored by

**Mohammed looti**

November 13, 2025

## The Essential Role of Programmatic Column Renaming

In the dynamic field of R data analysis, the process of data cleaning and preparation is paramount, often demanding the standardization of variable names. While manually adjusting column headers might be feasible for small, bespoke datasets, managing large-scale data--which frequently involves dozens or even hundreds of variables--requires a robust, efficient, and **programmatic** approach. Establishing and maintaining consistent naming conventions is fundamental for ensuring compliance, maximizing code readability, and facilitating seamless integration with advanced statistical models or external data engineering systems.

This critical need for programmatic efficiency is expertly met by the **dplyr** package, which stands as a foundational element within the broader Tidyverse ecosystem in R. The **dplyr** package provides a powerful and intuitive collection of verbs designed specifically to manipulate and transform data structures with exceptional ease. For the particular task of renaming multiple columns simultaneously based on a defined rule or function, **dplyr** introduces the highly specialized and invaluable function: **rename_with()**.

The central utility of **rename_with()** lies in its ability to apply a specified renaming function uniformly across a selection of columns. This function completely eliminates the repetitive and error-prone process of writing individual renaming operations for every single variable. By automating string transformations, **rename_with()** significantly streamlines complex data workflows. It is explicitly engineered to handle scenarios where column names require uniform modification, such as converting all names to lowercase, scrubbing special characters, or implementing complex string manipulation rules defined by the user.

## Deconstructing the rename_with() Syntax

To utilize this powerful tool effectively, data analysts must possess a clear understanding of its structure and the specific purpose of its arguments. The **rename_with()** function adheres strictly to the standard Tidyverse syntax, ensuring smooth integration into existing data pipelines. The basic signature defining the function's operation is structured as follows:

**rename_with(.data, .fn, .cols = everything(), ...)**

This signature highlights the three primary arguments that govern the renaming process. The first argument, **.data**, accepts the input data frame or tibble upon which the transformation will be performed. The second argument, **.fn**, is arguably the most crucial component: it requires a function that accepts a character vector (representing the current column names) and returns a corresponding modified character vector (the new column names). This exceptional flexibility allows users to employ any base R function or a custom-defined function for the renaming logic.

The optional third argument, **.cols**, defines precisely which columns within the data frame should be targeted by the renaming function. By default, **.cols** is set to **everything()**, which means the function will attempt to rename all columns. However, **rename_with()** is designed to work seamlessly with the powerful dplyr selection helpers, such as **starts_with()**, **ends_with()**, **contains()**, or **num_range()**. Leveraging these helpers provides precise control, ensuring that only the relevant metric or descriptive columns are transformed, while critical identifier or grouping columns remain untouched, thereby preserving data integrity.

**.data**: The input data frame or tibble being manipulated by the operation.
**.fn**: The renaming function (e.g., **toupper**, **tolower**, **gsub**, or a custom lambda function) applied sequentially to the selected column names.
**.cols**: A specification that defines the subset of columns to rename; the default and broadest value is **everything()**.

## Setting Up the Demonstration Data Frame in R

To practically illustrate the application and benefits of **rename_with()**, we must first establish a sample data frame. This dataset will contain basic performance statistics for a small group of basketball players. This example is intentionally structured with column names that require immediate standardization, specifically due to mixed casing and the use of **snake_case** (e.g., `total_points`).

The following code snippet generates our initial sample data frame. Notice the inconsistent naming conventions, particularly the underscores and the lowercase team identifier, which we intend to address using the versatility of **rename_with()** in the subsequent examples. This setup provides a clear starting point for demonstrating targeted and global transformations.

```
#create data frame
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
total_points=c(99, 68, 86, 88, 95, 74, 78, 93),
total_assists=c(22, 28, 45, 35, 34, 45, 28, 31),
total_rebounds=c(30, 28, 24, 24, 30, 36, 30, 29))

#view data frame
df

team total_points total_assists total_rebounds
1 A 99 22 30
2 A 68 28 28
3 A 86 45 24
4 A 88 35 24
```

5 B 95 34 30
6 B 74 45 36
7 B 78 28 30
8 B 93 31 29

This initial structure clearly presents our challenge: we have one categorical variable (`team`) and three quantitative metric variables (`total_points`, `total_assists`, `total_rebounds`). Our overarching goal is to apply various transformations to these names to achieve a unified standard, thereby showcasing the power, versatility, and ease of use inherent in the **rename_with()** function across different operational requirements.

## Global Transformations: Applying Uniform Case Changes (Using toupper)

A very frequent requirement in data processing, especially when preparing data for integration into SQL databases or specific data warehousing environments, is converting all column headers to a uniform case, typically uppercase, to mitigate potential case-sensitivity conflicts. The base R function **toupper()** is the perfect tool for this task, and when combined with **rename_with()**, the transformation becomes instantaneous and global across the entire dataset.

Because the **.cols** argument defaults intelligently to **everything()**, we need only to supply the input data frame and the desired renaming function (**toupper**). The **rename_with()** function handles the complex iteration internally, passing each existing column name to **toupper** and efficiently collecting the results to construct the set of new column headers. This process ensures maximal speed and efficiency for broad modifications.

**library(dplyr)**

#rename all columns to uppercase
rename_with(df, toupper)

TEAM TOTAL_POINTS TOTAL_ASSISTS TOTAL_REBOUNDS
1 A 99 22 30
2 A 68 28 28
3 A 86 45 24
4 A 88 35 24
5 B 95 34 30
6 B 74 45 36
7 B 78 28 30
8 B 93 31 29

As clearly demonstrated by the resulting output, every column name in the data frame, including the `team` identifier, has been successfully converted to all uppercase letters. This comprehensive, global transformation is highly efficient, proving especially valuable when managing wide datasets where hundreds of variables require identical case changes for consistency. This showcases the immediate speed and operational simplicity provided by **rename_with()** for uniform data structure modifications.

## Precision Renaming: Targeting Specific Column Subsets

While global renaming is highly useful, data analysts often encounter scenarios where transformations must be applied selectively. For example, we might need to standardize only the quantitative metric columns (those beginning with "total") while ensuring the case of categorical identifier columns (like "team") is preserved. This is precisely where the robust functionality of the **.cols** argument, when combined with dplyr selection helpers, demonstrates its true power.

By accurately specifying **starts_with("total")** within the **.cols** argument, we explicitly instruct **rename_with()** to strictly limit the application of the **toupper** function only to those column names that commence with the string "total". This highly targeted approach ensures extreme precision in data preparation and prevents any unintended alteration to variables that already comply with required standards or serve unique structural purposes within the dataset.

**library(dplyr)**

#rename specific columns to uppercase
rename_with(df, toupper, starts_with("total"))

team TOTAL_POINTS TOTAL_ASSISTS TOTAL_REBOUNDS
1 A 99 22 30
2 A 68 28 28
3 A 86 45 24
4 A 88 35 24
5 B 95 34 30
6 B 74 45 36
7 B 78 28 30
8 B 93 31 29

The results clearly illustrate the intended outcome: the `team` column maintains its original lowercase format, while the metric columns--`TOTAL_POINTS`, `TOTAL_ASSISTS`, and `TOTAL_REBOUNDS`--have all been correctly converted to uppercase. This demonstrates the superior level of control offered by **rename_with()**, which allows users to apply specific functions only to

subsets of variables defined by sophisticated and powerful logical criteria.

## Advanced String Manipulation: Replacing Characters with gsub

The true versatility of **rename_with()** becomes apparent when analysts move beyond simple case changes and integrate more advanced string manipulation functions. A highly common data preparation requirement involves replacing specific characters, such as transitioning from **snake_case** (using underscores) to **kebab-case** (using dashes), or systematically removing unwanted prefixes or suffixes. We can achieve this complex task seamlessly by utilizing powerful base R string functions like **gsub()** within the **.fn** argument.

When integrating a function such as **gsub()**, which requires multiple arguments (pattern, replacement, and string), we must use R's concise formula interface or an anonymous function. In the context of **rename_with()**, the column name string itself is implicitly passed as the variable **.x**. The formula expression `~ gsub("_", "-", .x, fixed = TRUE)` instructs R to execute a global substitution, replacing every instance of an underscore (`"_"`) with a dash (`"-"`) within the column name string (`.x`). The `fixed = TRUE` argument ensures that the pattern is treated literally.

**library(dplyr)**

#replace underscores with dashes in column names
rename_with(df, ~ gsub("_", "-", .x, fixed = TRUE))

team total-points total-assists total-rebounds
1 A 99 22 30
2 A 68 28 28
3 A 86 45 24
4 A 88 35 24
5 B 95 34 30
6 B 74 45 36
7 B 78 28 30
8 B 93 31 29

The resulting output clearly demonstrates the successful conversion of all underscores in the metric columns into dashes, achieving the kebab-case format. This operation is exponentially more efficient than manually executing **gsub()** for each column individually. This unique ability to integrate highly complex, multi-argument functions via the elegant formula interface is what establishes **rename_with()** as an indispensable tool for advanced R data manipulation and standardization.

## Conclusion: The Efficiency of Programmatic Naming

The **rename_with()** function, a core component of the **dplyr** package, provides analysts with a robust, flexible, and highly efficient solution for the bulk renaming of columns within an R data frame. By leveraging its carefully designed architecture, users can instantly apply any standard or custom function to their column names, achieving instantaneous standardization, enhanced code clarity, and superior data quality across datasets of any size.

Regardless of the required task--whether it is simple case conversion using **toupper**, precise, targeted renaming using advanced selection helpers, or intricate character replacement via **gsub**--**rename_with()** dramatically simplifies the entire process. This critical functionality is essential for maintaining code reproducibility and consistency in modern data science projects, firmly cementing its status as an essential and frequently used tool in the Tidyverse toolkit.

We strongly encourage users to further explore the comprehensive **dplyr** documentation to uncover other powerful data manipulation functions that integrate seamlessly with **rename_with()**, enabling the development of even more sophisticated and fully automated data preparation workflows.

# Additional Resources

The following tutorials explain how to perform other common tasks in R: