Standardizing Column Names in R: A Tutorial Using the clean_names() Function

Authored by

Mohammed looti

November 13, 2025

In the advanced world of **R programming** and statistical computing, the foundational requirement for efficient analysis is the presence of standardized, consistent variable names. Data frequently arrives in its raw form from sources like spreadsheets, legacy systems, or messy APIs, often featuring column headers riddled with inconsistencies, special characters, embedded spaces, and mixed capitalization. These irregularities are not merely aesthetic issues; they introduce significant friction into the data workflow, complicating downstream analysis, breaking packages that rely on non-standard evaluation, and hindering code reproducibility. To address these universal challenges, data professionals require robust, automated solutions capable of transforming chaotic column headers into the highly desirable structure known as **tidy data**.

The most authoritative and efficient tool available to **R data frame** users for this purpose is the `clean_names()` function. This function is the centerpiece of the highly-regarded **janitor package**, a specialized library explicitly engineered to streamline common **data cleaning** operations. The **janitor** package offers a fast, opinionated, and highly reliable method for preparing data structures for immediate analysis. The `clean_names()` function excels at converting diverse, complex column names into a consistent format, typically defaulting to the lowercased, underscore-separated convention known as **snake_case**, while automatically resolving complex issues such as punctuation, leading numerical characters, and the creation of unique names for duplicates.

The Critical Importance of Standardized Column Names in R

While column name structure might initially appear to be a minor stylistic consideration, its impact on code efficiency, maintainability, and collaboration is profound. When column identifiers include spaces (e.g., "Customer ID"), referencing them in R requires cumbersome quoting using backticks (`` `Customer ID` ``). This practice rapidly deteriorates code readability and increases the likelihood of scripting errors. Similarly, inconsistent capitalization--such as mixing "transactionDate" and "Transaction_Date"--imposes unnecessary cognitive load on the analyst and introduces inherent risk of typographical mistakes during scripting. Standardization eliminates this ambiguity, ensuring every variable can be referenced easily and consistently, often without the need for quotation marks.

Adopting a single, defined naming convention is a recognized best practice across modern programming and **data science** communities. The preferred standard in the R ecosystem is often **snake_case**, where words are lowercased and separated by underscores. This uniformity is essential when integrating custom functions or working collaboratively within a team, as it removes all uncertainty regarding how variables should be spelled and accessed. The `clean_names()` function automatically enforces these robust conventions, saving the analyst substantial manual effort that would otherwise be spent writing laborious, custom renaming scripts.

The core functionality of `clean_names()` extends far beyond simple lowercasing. It performs

several critical data hygiene operations to ensure compliance with R's stringent rules for valid variable names. Specifically, it meticulously removes all non-alphanumeric characters (except underscores), replaces spaces with underscores, and critically, guarantees that all resulting names are unique by appending sequential numbers if any duplicates are detected during the transformation process. This comprehensive approach ensures that the output data frame is immediately compatible with powerful analytical packages, such as those within the **tidyverse** ecosystem, which rely heavily on clean, syntactic column identifiers for smooth operation.

Installation and Prerequisites for Using the janitor Package

Before leveraging the powerful capabilities of the `clean_names()` function, the analyst must first ensure that the **janitor package** is correctly installed and loaded into the current R session. Unlike base R packages, **janitor** is a third-party extension. This means it must be explicitly installed once onto your system using the package manager, and subsequently loaded into every new R session where its functions are required using the `library()` command.

The initial installation procedure is straightforward and utilizes R's built-in package management capabilities. The following command instructs R to fetch the package files from the Comprehensive R Archive Network (CRAN) and install them locally onto your computing environment.

To install the necessary package, execute the following syntax in your R console:

install.packages('janitor')

Once the installation is complete, the final prerequisite is loading the library into your current working session. This loading step is absolutely crucial, as the functions within **janitor** will not be accessible until this command is executed. After the **janitor package** has been successfully installed and loaded, you can proceed directly to applying the `clean_names()` function to any **R data frame** object.

Understanding the Syntax and Core Arguments of clean_names()

The design philosophy behind the `clean_names()` function prioritizes simplicity and ease of use, requiring only the data object itself as the primary input. Nevertheless, it includes an important optional argument that allows users to override the default naming convention (which is **snake_case**). Grasping these basic components is essential for utilizing the function effectively across diverse project requirements and stylistic needs.

The fundamental syntax for calling `clean_names()` is defined concisely:

clean_names(dat, case)

The function accepts two main arguments that determine both the target data structure and the desired output style. The first argument, `dat`, is mandatory and refers simply to the name of the data structure--typically an **R data frame** or tibble--whose column names require standardization. The second argument, `case`, is optional but highly flexible. It specifies the desired target case convention for the resulting column names, providing an escape from the default **snake_case** setting when organizational or project standards dictate an alternative format.

dat: This mandatory argument accepts the rectangular data object (such as a **data frame**) whose column headers will be modified. It is important to note that the function only alters the names of this object; the underlying data values remain unchanged.
case: This optional string argument dictates the final naming convention style. If this argument is omitted, `clean_names()` defaults to **snake_case** (e.g., `total_score`), which is widely recommended for general **R programming** practices. However, users can specify alternatives based on their specific organizational or project style guides.

Exploring Flexible Case Convention Options (Snake, Camel, All Caps)

While **snake_case** is the most common and often preferred convention within the R community, the `case` argument offers significant flexibility, enabling users to conform to various style guides used across different languages or computing platforms. The **janitor package** recognizes that certain environments--such as those interacting with specific databases, Java, or JavaScript--might require conventions like camel case. Thus, the package provides a comprehensive list of supported transformations. Selecting the appropriate case convention ensures seamless integration of R-processed data with external software architectures.

The `case` argument is capable of accepting any of the following string values, each producing a distinct, standardized output format:

"snake" produces the standard **snake_case** (e.g., `variable_name`). This is the default setting and highly recommended for general R use.
"lower_camel" produces **lowerCamel** (e.g., `variableName`). This style is frequently encountered in JavaScript and certain API communication protocols.
"upper_camel" produces **UpperCamel** (e.g., `VariableName`). Also referred to as PascalCase, this is often utilized in contexts involving object-oriented programming.
"all_caps" produces **ALL_CAPS** (e.g., `VARIABLE_NAME`). This convention is sometimes required for constants or specific legacy database schemas.
"lower_upper" produces **lowerUPPER** (e.g., `variableNAME`). This is less common but supported for specialized stylistic requirements.
"upper_lower" produces **UPPERlower** (e.g., `VARIABLEname`). Another specialized case convention provided for comprehensive coverage.

In practice, analysts typically deploy **`clean_names()`** during the very first stages of the **data cleaning** pipeline. The primary objective is to enforce a uniform pattern throughout the entire data frame. This uniformity ensures that regardless of the original data source--be it a poorly formatted Excel spreadsheet, a CSV file, or a complex database query--all column headers adhere strictly to the chosen standard, thereby maximizing compatibility and minimizing operational friction during subsequent data manipulation and statistical modeling steps.

Practical Implementation: Cleaning a Messy Data Frame in R

To fully appreciate the power and streamlined simplicity of **`clean_names()`**, let us walk through a common data preparation scenario. We will simulate importing raw data into R, where the data frame has problematic column names characterized by inconsistent casing, missing word separators, and mixed capitalizations--all of which make programmatic access difficult.

First, we create a sample data frame detailing hypothetical basketball player statistics. Notice the problematic column names intentionally included: `TEAM` (all caps), `pointsscored` (all lowercase, no separator), `assists` (lowercase), and `Totalrebounds` (mixed case).

 # Create data frame with messy column names
df <- data.frame(TEAM=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
pointsscored=c(99, 68, 86, 88, 95, 74, 78, 93),
assists=c(22, 28, 31, 35, 34, 45, 28, 31),
Totalrebounds=c(30, 28, 24, 24, 30, 36, 30, 29))

View the data frame before cleaning
df

TEAM pointsscored assists Totalrebounds
1 A 99 22 30
2 A 68 28 28
3 A 86 31 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 the output clearly demonstrates, the column names lack uniformity in pattern and style. We now apply the **`clean_names()`** function without specifying the `case` argument, allowing it to utilize the default **snake_case** convention. Crucially, the function's internal logic automatically detects where

word separation should occur (often by identifying case changes), inserting underscores and resolving issues like `Totalrebounds` becoming `total_rebounds`.

We must first load the **janitor package**, and then apply the cleaning function to the data frame:

library(janitor)

# Clean names of data frame using the default (snake_case)
clean_names(df)

team pointsscored assists total_rebounds
1 A 99 22 30
2 A 68 28 28
3 A 86 31 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 data frame maintains the exact same underlying data but now features column names that are entirely standardized. Notice the transformation: `TEAM` was successfully converted to `team`, and the problematic `Totalrebounds` was intelligently corrected to `total_rebounds`. This showcases the sophisticated logic embedded within the **janitor** package, ensuring the data is immediately usable for complex analysis.

Utilizing Specific Case Conventions for External Requirements

The true versatility of the **`clean_names()`** function becomes apparent when project specifications mandate a specific case convention that deviates from the R standard. For example, if the data must be prepared for seamless integration with a legacy system or database that requires all uppercase identifiers, we can easily enforce this transformation using the flexible `case` argument.

To convert every column name exclusively to uppercase characters, we utilize the **`all_caps`** option:

library(janitor)

# Clean names of data frame using all_caps
clean_names(df, case='all_caps')

TEAM POINTSSCORED ASSISTS TOTAL_REBOUNDS
1 A 99 22 30
2 A 68 28 28
3 A 86 31 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 execution returns the data frame structure with all column names uniformly converted to uppercase characters. Furthermore, the function correctly inserts underscores to separate detected words (e.g., transforming the conceptual `TOTALREBOUNDS` into the syntactic `TOTAL_REBOUNDS`).

Alternatively, if the project workflow demands a lower camel case convention--a format often preferred in environments like SQL stored procedures or certain API communication formats--we simply specify the **`lower_camel`** argument. This setting converts the column names into a format where the first letter is lowercase and subsequent word starts are capitalized, eliminating underscores entirely.

library(janitor)

# Clean names of data frame using lower_camel
clean_names(df, case='lower_camel')

team pointsscored assists totalRebounds
1 A 99 22 30
2 A 68 28 28
3 A 86 31 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

In this final demonstration, every column name adheres perfectly to the lower camel case standard. Analysts are empowered to select the case convention that optimally satisfies their project's requirements, relying on the robustness and flexibility provided by the **`case`** argument of the **`clean_names()`** function to maintain data integrity while ensuring maximum code readability

and system compatibility.

Additional Resources

For further exploration into advanced data manipulation, cleaning techniques, and best practices in **R**, consider reviewing the official documentation for the **tidyverse** and the comprehensive documentation provided by the **janitor package** developers.

<!--

Featured Posts

-->