Combine Two Columns into One in R (With Examples)

Authored by

Mohammed looti

November 7, 2025

In the vast landscape of data science and statistical computation, the ability to meticulously prepare and structure data is often the most critical step toward meaningful analysis. Within the powerful **R programming environment**, data analysts frequently encounter situations where crucial information is distributed across several distinct columns. This segmentation, while sometimes necessary for initial data acquisition, often proves inefficient for subsequent modeling, visualization, or creating unique identifiers. The requirement to merge two or more existing variables into a single, cohesive column is therefore a fundamental data wrangling task, essential for standardization and simplifying input structures.

Consider a standard business or scientific application involving **time-series data** where components of the date--specifically the month and the year--have been stored separately within an **R data frame**. Maintaining these components separately complicates tasks like sorting or indexing by chronological order using a single field. The initial structure might present itself in a format similar to the following table, where temporal context must be derived from two independent columns:

month year value
10 2019 15
10 2020 13
11 2020 13
11 2021 19
12 2021 22

The core objective of this transformation is to consolidate these fragmented temporal identifiers (`month` and `year`) into a singular column, perhaps labeled `date_key`, typically utilizing a clear, consistent delimiter such as an underscore or a hyphen. This consolidation yields a data structure that is substantially more efficient for subsequent analytical operations, including filtering, aggregation, and the creation of unique keys for database joining. The resulting desired structure elegantly represents the combined information:

date value
2019_10 15
2020_10 13
2020_11 13
2021_11 19
2021_12 22

This comprehensive tutorial will guide you through two primary, established methodologies available in R for executing this column combination. We will first explore the foundational

capabilities provided by **Base R**, which relies solely on core R functions without external package dependencies. Following this, we will demonstrate the highly streamlined and modern approach offered by the **Tidyverse** suite, specifically leveraging the specialized functions in the `tidyr` package.

Initial Setup and Data Requirements

Before implementing the column combination techniques, it is necessary to confirm that the R environment is properly configured. Ensuring a current and working installation of **R** is the fundamental prerequisite. For analysts intending to utilize the second, Tidyverse-based method, it is beneficial to have familiarity with the core concepts of the **Tidyverse**, particularly the data manipulation philosophy championed by packages like `dplyr` and the data reshaping tools provided by `tidyr`.

The methods detailed here are primarily concerned with combining vectors that contain or can be easily converted into character strings. When working with numeric columns, such as our `month` and `year` fields, R automatically coerces these into character strings during the concatenation process to facilitate the merging of text elements. It is important to distinguish this simple string combination from the more complex manipulation of true date-time objects, which often requires specialized libraries like `lubridate` to handle time zones, formatting, and arithmetic operations. However, for the straightforward task of creating a delimited identifier, the string combination tools are perfectly adequate.

To ensure reproducibility and consistency throughout this demonstration, we will begin by initializing the sample data frame that will serve as the input for both the Base R and Tidyverse approaches. This initialization uses standard **Base R** commands to define the structure of the `data` object, replicating the initial segmented structure we aim to consolidate.

```
# Initialize the example data frame named 'data'
data <- data.frame(month=c(10, 10, 11, 11, 12),
year=c(2019, 2020, 2020, 2021, 2021),
value=c(15, 13, 13, 19, 22))

# Verify the structure and data types
str(data)
```

Once the data frame is created, we can proceed directly to the first method, which leverages the fundamental tools available in the core R distribution, proving reliable regardless of the user's package environment.

Method 1: The Robust Base R Approach using paste() Function

The most foundational and universally applicable technique for concatenating vectors, whether they contain text or numeric values, within R is through the use of the built-in **`paste`** function. As a core function of **Base R**, **`paste`** requires no installation of supplementary packages, cementing its position as an efficient, lightweight choice for rapid data manipulation, especially in scripts where minimizing dependencies is crucial.

The **`paste`** function operates by converting all input arguments--which are typically vectors or columns from a data frame--into character strings, and then joining these strings element-wise. Its utility is largely governed by the `sep` argument, which defines the separator that will be inserted between the concatenated elements. For our specific objective of generating a 'date' column formatted as "YYYY_MM", it is essential that we explicitly define the underscore character (_) as the separator within the function call. Failure to specify `sep` results in the default separator, which is a single space.

Implementation involves a standard R assignment: we create a new column, `data$date`, and assign the output of the **`paste`** operation to it. Crucially, the order in which the columns are listed within the **`paste`** function (e.g., `data$year` followed by `data$month`) directly determines the resulting sequence of the combined string. It is vital to maintain the correct chronological or logical order required for the new key.

#create data frame (re-initializing for clarity)
data <- data.frame(month=c(10, 10, 11, 11, 12),
year=c(2019, 2020, 2020, 2021, 2021),
value=c(15, 13, 13, 19, 22))

#combine year and month into one column using '_' as the separator
data$date <- paste(data$year, data$month, sep="_")

#view new data frame structure and content
data

month year value date
1 10 2019 15 2019_10
2 10 2020 13 2020_10
3 11 2020 13 2020_11
4 11 2021 19 2021_11
5 12 2021 22 2021_12

Essential Post-Combination Step: Removing Source Columns

A significant operational distinction between the **`paste`** function and the Tidyverse's `unite()` function (Method 2) is the handling of the source columns. When using **`paste`**, the newly created column is simply appended to the existing **data frame**, meaning the original columns (`month` and `year`) are preserved. While preservation offers flexibility, these original columns often become redundant once the combination is successful, cluttering the dataset and potentially leading to confusion or errors in subsequent analysis.

To maintain a tidy and efficient dataset--a critical aspect of effective data wrangling--these redundant columns must be explicitly removed. In **Base R**, this is accomplished using standard data frame subsetting techniques. The most efficient way to achieve this involves explicitly selecting only the columns designated for retention and assigning the result to a new object, which we call `data_new` in this scenario.

The method relies on providing a vector of column names (e.g., `c("date", "value")`) within the square bracket notation used for subsetting the data frame. This technique ensures that only the relevant, derived information is retained, effectively dropping the unnecessary source variables in a single, clear operation. This two-step structure--combining first, then subsetting--is highly characteristic of many data manipulation pipelines executed exclusively using **Base R** functionality.

 # Select only the desired columns (date and value)
data_new <- data

View the refined data frame structure
data_new

date value
1 2019_10 15
2 2020_10 13
3 2020_11 13
4 2021_11 19
5 2021_12 22

This process ensures that the resulting data frame is lean, focused, and ready for analytical routines where redundant columns might pose difficulties or increase memory consumption.

Method 2: Streamlining Data Wrangling with tidyr::unite()

For R users who are deeply integrated into the **Tidyverse** ecosystem, the **`tidyr`** package offers a highly efficient and syntactically clean alternative for reshaping data: the **`unite()`** function. This

method is the preferred choice in modern R workflows because it encapsulates the entire column combination and cleanup process into a single, highly readable function call, aligning perfectly with the Tidyverse philosophy of chainable, declarative operations.

The primary benefit of **`unite()`** over the two-step **`paste`** approach is its native handling of column removal. By default, **`unite()`** is engineered to automatically drop the original source columns after they have been successfully merged into the new destination column. This single-step execution significantly streamlines the data cleaning process, reducing the risk of errors associated with manual subsetting or column selection.

To employ this technique, the **`tidyr`** package must be loaded into the session using `library(tidyr)`. The **`unite()`** function requires three essential pieces of information: first, the data frame object (often passed via the pipe operator, though here we use direct input); second, the desired name of the new combined column (e.g., `date`); and third, a vector listing the names of the source columns to be merged (e.g., `c(year, month)`). Importantly, the default separator used by **`unite()`** is the underscore (_), which conveniently matches the format established in our Base R example. If a different delimiter were required, the `sep` argument could be explicitly provided.

```
#load tidyr package (part of the Tidyverse)
library(tidyr)

#create data frame (re-initializing for demonstration)
data <- data.frame(month=c(10, 10, 11, 11, 12),
year=c(2019, 2020, 2020, 2021, 2021),
value=c(15, 13, 13, 19, 22))

#combine year and month into one column and automatically remove source columns
unite(data, date, c(year, month))

date value
1 2019_10 15
2 2020_10 13
3 2020_11 13
4 2021_11 19
5 2021_12 22
```

The resulting output is functionally identical to the refined data frame achieved through the two-step Base R process, yet it is accomplished with demonstrably greater conciseness and clarity, highlighting the benefits of utilizing specialized Tidyverse tools for dedicated data wrangling tasks.

Advanced Considerations and Use Cases

While this tutorial focuses on merging two numeric columns (coerced to strings) into a single key, the principles of **`paste()`** and **`unite()`** are broadly applicable to various combination tasks. For instance, combining first name and last name into a `full_name` field, or merging geographical identifiers like `state_code` and `county_id` to create a unique identifier, relies on the exact same methodology.

A key consideration for advanced users is the difference between simple string concatenation and true data type conversion. Since both **`paste()`** and **`unite()`** yield character vectors, the resulting `date` column cannot be directly used for chronological calculations. If the analytical goal requires date arithmetic (e.g., calculating time differences or sequencing data), an additional step is necessary to convert the resulting string column into a proper R date object using functions like `as.Date()` or the dedicated tools provided by the `lubridate` package. However, for creating unique keys or labels for visualization, the character vector output is sufficient.

Performance considerations, particularly with very large **R data frame** objects (millions of rows), might sometimes favor Base R operations, as they bypass the overhead associated with package loading and Tidyverse object handling. However, for the vast majority of medium-sized datasets encountered in standard data analysis, the difference in execution speed between the optimized **`unite()`** function from **tidyr** and the Base R **`paste()`** function is negligible, making code readability and maintainability the overriding factors in method selection.

Comparative Analysis: paste() vs. unite()

Both the **`paste`** function in Base R and the **`unite`** function from `tidyr` are highly effective solutions for merging multiple columns into a single variable. The definitive choice between these two methods often rests on the analyst's existing coding philosophy, project constraints, and specific priorities regarding package usage and code structure.

The primary advantage of **`paste()`** lies in its zero-dependency requirement. It is a robust tool, guaranteed to work in any R environment without needing to load external libraries, making it ideal for streamlined scripts or environments with strict package restrictions. Conversely, **`unite()`** demands the loading of the `tidyr` package, but in return, it provides significantly enhanced syntax and functionality. It adheres to the consistent, highly readable syntax of the **Tidyverse**, facilitating cleaner code when dealing with complex, multi-step data manipulation pipelines.

The key differentiator remains column handling: **`unite()`** performs the two necessary steps--combination and removal of source columns--in one operation, which generally leads to less verbose and more reliable code. While **`paste()`** requires a manual, separate subsetting step, this method offers finer granular control over the data frame structure at every stage. Ultimately, both

methods deliver identical, reliable results. Analysts who prioritize minimal dependencies and core R functionality should embrace **`paste()`**, while those who value efficiency, chaining, and modern syntax will find **`unite()`** to be the superior and more elegant solution.

Dependency: **`paste()`** requires no external packages, ensuring stability. **`unite()`** necessitates loading the `tidyr` package.
Syntax: **`unite()`** offers a declarative, intuitive syntax focused purely on reshaping data, which integrates smoothly into Tidyverse pipelines.
Column Handling: **`unite()`** automatically removes the source columns by default, simplifying the data cleaning process. **`paste()`** requires an explicit, subsequent subsetting operation.

*For detailed usage and additional arguments concerning the **`unite`** function, including options for handling missing values and controlling column removal, please refer to the official documentation.*