Learning to Combine Data Frames in R with dplyr’s bind_rows()

Authored by

Mohammed looti

November 13, 2025

Introduction to Combining Data Structures in R

In the realm of data analysis and manipulation using R, it is a frequent requirement to consolidate information from multiple sources. Data is rarely available in a single, perfectly structured file; instead, analysts often encounter scenarios where they must merge two or more disparate datasets, typically stored as distinct data frame objects, into one cohesive structure for subsequent processing. Efficiently managing this process is critical for maintaining data integrity and streamlining analytical workflows. The **tidyverse** ecosystem, and specifically the powerful **dplyr** package, provides a suite of intuitive and high-performance functions designed precisely for these combination tasks, offering significant advantages in readability and speed over base R methods, especially when dealing with large datasets.

The choice of combination method hinges entirely on the desired output structure. Fundamentally, combining data frames involves two primary approaches: stacking them vertically, which results in a "long" data frame by appending rows, or joining them horizontally, which results in a "wide" data frame by binding columns. The **dplyr** package simplifies these complex operations into two primary functions: `bind_rows()` and `bind_cols()`. Understanding the distinctions between these functions--how they handle mismatched columns or rows, and the necessary prerequisites for successful execution--is paramount for any serious data practitioner utilizing R for manipulation and statistical modeling.

Before diving into the practical examples, it is important to ensure that the foundational tools are in place. The **dplyr** package is not automatically loaded with base R, although it is a core component of the **tidyverse** meta-package. Therefore, a necessary preliminary step involves verifying its installation and ensuring it is properly loaded into the current R session. For users who have not yet installed this utility, the installation process is straightforward and typically only needs to be performed once, providing access to a vast array of data wrangling functions that significantly enhance the capabilities of the R environment.

Prerequisites: Installing and Loading the dplyr Package

For data manipulation tasks requiring the robust functionality provided by `bind_rows()` or `bind_cols()`, the **dplyr** package must be accessible. If this package is not yet installed on your system, you must execute the following command within the R console. This command initiates the process of downloading the package and its dependencies from the Comprehensive R Archive Network (CRAN) and installing them locally, making the functions available for use in future sessions.

install.package('dplyr')

Once the installation is complete, or if the package was already installed, the next critical step is to load the library into the current working environment. Loading a package makes its functions callable directly without needing to specify the package namespace every time. The standard function for this operation is `library()`. By running this command, all functions, including the row-binding and column-binding utilities, become immediately available for execution against your data frame objects, allowing you to proceed with the data combination tasks efficiently.

library(dplyr)

new_df <- bind_rows(df1, df2)

It is important to recognize that while loading the **dplyr** package is a necessary prerequisite for using its specialized functions, the syntax shown above demonstrates the structure for combining data frames by rows, which is one of the two primary methods we will explore in detail. This preliminary setup ensures a smooth transition into the core manipulation steps, regardless of whether you are combining data vertically or horizontally, provided your source data frames are ready for integration.

Method 1: Combining Data Frames by Rows using `bind_rows()`

The `bind_rows()` function is specifically engineered to combine two or more data frames vertically. This operation, often termed row-binding or concatenation, takes the rows from the second data frame (and any subsequent data frames) and appends them to the end of the first data frame. The fundamental requirement for this method is that the data frames share common column names, although **dplyr** is smart enough to handle slight variations. If a column exists in one data frame but not the other, `bind_rows()` automatically fills the missing entries with `NA` values in the resulting combined data frame, ensuring no data is lost and the resulting structure remains consistent.

Executing `bind_rows()` results in a "long" data frame. Imagine you have monthly sales records stored in separate files, **sales_jan** and **sales_feb**. Using `bind_rows()` combines these records sequentially, creating a unified dataset spanning both months. This is invaluable when the underlying schema (the column structure) is identical or highly similar, and the goal is simply to aggregate observations over different time periods or segments. The syntax is straightforward, requiring only the names of the data frames you wish to combine, separated by commas, and the assignment of the result to a new variable.

Consider the following practical illustration, where we define two distinct data frames, **df1** and **df2**, both containing information about basketball teams and their scores, maintaining an identical column structure. This setup ensures a seamless integration when using the row-binding

approach, as demonstrated by the subsequent code block. The resulting data frame, **new_df**, will possess a total number of rows equal to the sum of the rows in the contributing data frames, while the column count remains unchanged.

Example 1: Using `bind_rows()` to Combine Data Frames by Rows

We begin by defining the two source data frames, **df1** and **df2**, ensuring they represent structured data with consistent column headings. Notice that both structures contain the variables `team` and `points`, which is the necessary prerequisite for a clean row-binding operation.

#create first data frame
df1 <- data.frame(team=c('A', 'A', 'B', 'B', 'C'),
points=c(22, 25, 30, 43, 19))

df1

team points
1 A 22
2 A 25
3 B 30
4 B 43
5 C 19

#create second data frame
df2 <- data.frame(team=c('D', 'D', 'E', 'F', 'F'),
points=c(11, 36, 29, 22, 30))

df2

team points
1 D 11
2 D 36
3 E 29
4 F 22
5 F 30

As observed in the output above, both **df1** and **df2** possess five rows and two columns. The immediate goal is to vertically stack these datasets to form one comprehensive data structure. We utilize the `bind_rows()` function from the loaded **dplyr** library to perform this aggregation, resulting in a single data frame that logically sequences the entries from **df1** followed by the entries from **df2**.

```
library(dplyr)

#row bind two data frames together
new_df <- bind_rows(df1, df2)

#view new data frame
new_df

team points
1 A 22
2 A 25
3 B 30
4 B 43
5 C 19
6 D 11
7 D 36
8 E 29
9 F 22
10 F 30
```

This returns one "long" data frame named **new_df** that has combined the data frames named **df1** and **df2** into one. The result is a consolidated structure containing 10 rows, which successfully integrates all observations sequentially. This method is exceptionally useful when pooling data collected using the same methodology or structure, such as combining experimental results from different cohorts or adding newly acquired data points to an existing dataset without altering the original variables.

Method 2: Combining Data Frames by Columns using `bind_cols()`

In contrast to row-binding, the `bind_cols()` function is utilized for combining data frames horizontally, attaching the columns of one data frame alongside the columns of another. This operation, known as column-binding, is typically employed when you have supplementary information about the same set of observations that is stored separately. For instance, if **df1** contains demographic data (e.g., team name and points scored) and **df2** contains performance metrics (e.g., position and assists) for the identical list of players, column-binding is the appropriate technique to merge these parallel attributes into a unified record for each player.

The critical condition for using `bind_cols()` successfully is that all input data frames must contain the exact same number of rows. Unlike row-binding, which handles column mismatches gracefully by inserting `NA`, column-binding strictly requires a one-to-one correspondence between the

observations being linked. If the row counts differ, **dplyr** will issue an error or warning, as it cannot determine how to align the supplemental data correctly. When the row counts are equal, the function simply concatenates the columns side-by-side, preserving the original row order, thereby creating a "wide" data frame.

This approach is particularly valuable in situations where data collection was compartmentalized. Perhaps one researcher recorded primary variables while another recorded secondary variables on the same subjects, resulting in two data frames keyed implicitly by the order of observation. The resulting wide data frame will contain a total number of columns equal to the sum of the columns in the contributing data frames, while the row count remains identical to the original data frames.

Example 2: Using `bind_cols()` to Combine Data Frames by Columns

Suppose we create the following two data frames that contain information about various basketball players. We now define two data frames, **df1** and **df2**, specifically constructed to have the same number of rows (five rows each) but different columns, representing distinct attributes of the same five players.

#create first data frame
df1 <- data.frame(team=c('A', 'A', 'B', 'B', 'C'),
points=c(22, 25, 30, 43, 19))

df1

team points
1 A 22
2 A 25
3 B 30
4 B 43
5 C 19

#create second data frame
df2 <- data.frame(position=c('D', 'D', 'E', 'F', 'F'),
assists=c(11, 36, 29, 22, 30))

df2

position assists
1 D 11
2 D 36
3 E 29
4 F 22

5 F 30

Notice that these two data frames contain the same number of rows. Since the number of rows is identical, we can confidently apply the column-binding operation. This procedure assumes that the first row of **df1** corresponds conceptually to the first row of **df2**, and so forth, linking the supplementary data based purely on their positional index.

library(dplyr)

#column bind two data frames together
new_df <- bind_cols(df1, df2)

#view new data frame
new_df

team points position assists
1 A 22 D 11
2 A 25 D 36
3 B 30 E 29
4 B 43 F 22
5 C 19 F 30

This returns one "wide" data frame named **new_df** that has combined the data frames named **df1** and **df2** into one. The execution of `bind_cols()` results in the creation of this structure, successfully integrating all four variables across the original five observations. This outcome provides a complete profile for each observation, combining previously separate metrics into a single row, which is essential for comprehensive statistical analysis.

Additional Resources

The following tutorials explain how to perform other common tasks in R, providing supplementary knowledge for data manipulation and analysis within the **tidyverse** framework:

Performing Relational Joins in R
Filtering and Subsetting Data Frames
Grouping and Summarizing Data with group_by()
Reshaping Data using tidyr

<!--

Featured Posts

-->