Learning How to Combine Data Frames with dplyr’s union() Function in R

Authored by

Mohammed looti

November 13, 2025

In the realm of data preparation and analysis using R, a common requirement is the consolidation of information spread across multiple datasets. Specifically, analysts frequently encounter situations where they need to combine all unique rows from two or more separate data frames into a single, comprehensive structure. This operation, often termed a full outer join or, more precisely, a set union, is fundamental to robust data aggregation.

Fortunately, the powerful capabilities of the dplyr package--a core component of the Tidyverse--provide an elegant and highly optimized solution for this task. The primary tool for this purpose is the **union()** function, which is expertly designed to merge the rows of two data frames while meticulously ensuring that every resulting row is unique. Understanding how to leverage this function effectively is paramount for efficient data wrangling in R.

Understanding Set Operations in Data Manipulation

The concept behind combining data frames using **union()** is rooted deeply in standard **set theory**. In mathematics, the union of two sets, A and B, is defined as the set containing all elements that are in A, or in B, or in both. Crucially, if an element exists in both A and B, it is only listed once in the resulting union. This principle of automatic de-duplication is the defining characteristic of the **union()** function within dplyr.

When applying set operations to data frames, it is essential to recognize that each row in the data frame is treated as a distinct element or observation. For the union operation to execute successfully, the input data frames must possess an identical structure. Specifically, they must have the same number of columns, and the corresponding columns must share compatible data types. If these structural prerequisites are not met, the **union()** function will typically return an error or produce an unexpected result, emphasizing the need for meticulous data cleaning prior to combination.

While standard joins (like inner or left joins) merge columns based on shared key variables, the **union()** function merges rows, focusing solely on stacking the datasets on top of one another. This distinction is critical for users transitioning from SQL environments, as set operations offer a different, but equally necessary, tool for holistic data preparation.

Introducing the union() Function in dplyr

The primary purpose of the union() function is to return all rows that appear in either the first data frame (x) or the second data frame (y), eliminating any redundancy. This function is streamlined and efficient, making it the preferred method for set union operations within the modern R ecosystem.

The syntax for invoking this powerful operation is remarkably straightforward, maintaining the clear

and intuitive style characteristic of the dplyr package. The function requires two primary arguments, representing the two data frames intended for combination:

```
union(x, y)
```

The parameters used in this structure are defined as follows:

x: The name or object reference of the first data frame. This is treated as the initial set of observations.
y: The name or object reference of the second data frame. This contains the rows to be added to the result.

The output of the **union()** function is always a new, single data frame. Importantly, the resulting data frame preserves the column names and order established in the first input data frame, **x**. This consistency ensures predictability in the merged dataset, which is vital when performing subsequent analytical steps.

The Critical Distinction: union() vs. union_all()

While the goal of merging two datasets may seem singular, the treatment of duplicate rows introduces a crucial bifurcation in methodology. The standard **union()** function adheres strictly to mathematical set theory, meaning it automatically filters out any rows that appear identically in both data frames, or any rows that are repeated within a single data frame prior to the combination. This is ideal when the objective is a list of unique observations.

However, there are many scenarios in data analysis, particularly when dealing with transactional records or log data, where the preservation of every single record, including duplicates, is necessary. For these situations, the union_all() function provides the perfect alternative. This function uses the exact same syntax as **union(x, y)** but fundamentally alters the output behavior.

The core difference lies in their approach to row management: **union()** performs a distinct operation before combining, ensuring uniqueness, whereas **union_all()** simply stacks the rows of the second data frame (y) directly beneath the rows of the first data frame (x), irrespective of whether those rows are identical. Choosing between these two functions must be driven entirely by the specific analytical objective: choose **union()** for unique records (de-duplication) and **union_all()** for complete record retention (including duplicates).

Preparing the Environment: Installing and Loading dplyr

Before any data manipulation functions from the Tidyverse can be utilized, the required packages must be installed and loaded into the current R session. The **dplyr** package does not come pre-installed with base R, so a prerequisite step is necessary for first-time users. Ensuring the

environment is correctly configured prevents errors and guarantees access to the **union()** function.

If the **dplyr** package is not yet available on your system, you must first install it using the standard package installation command. This process typically requires an active internet connection to download the package files from CRAN (Comprehensive R Archive Network):

install.packages('dplyr')

Once the installation is complete, or if the package has been installed previously, it must be loaded into the current R session using the **library()** function. Loading the package makes all its contained functions, including **union()**, accessible for immediate use. This step is mandatory every time a new R session is started that intends to use dplyr functionalities.

Practical Application: Merging Two Data Frames

To illustrate the practical application of the union() function, let us construct a representative scenario involving two small data frames, **df1** and **df2**, which might represent partial records from two different data collection periods. Note that **df1** is deliberately designed to contain a duplicate row to demonstrate the de-duplication feature of **union()**.

We begin by defining the two sample data frames, **df1** and **df2**, ensuring they share the exact same column names and data types (`team` as character/factor and `points` as numeric). The creation and initial view of these data frames are shown below:

#create first data frame
df1 <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
points=c(14, 14, 19, 25, 40, 34, 38, 17))

df1

team points
1 A 14
2 A 14
3 A 19
4 A 25
5 B 40
6 B 34
7 B 38
8 B 17

#create second data frame

```
df2 <- data.frame(team=c('C', 'C', 'D', 'D', 'D', 'E', 'E', 'E'),
points=c(14, 10, 11, 15, 10, 32, 28, 27))

df2

team points
1 C 14
2 C 10
3 D 11
4 D 15
5 D 10
6 E 32
7 E 28
8 E 27
```

Data frame **df1** contains 8 rows, and data frame **df2** also contains 8 rows, yielding a potential total of 16 rows if combined without any de-duplication. Notice specifically that the first two rows of **df1** are identical: (Team A, Points 14). This duplicate pair will serve as the test case for the **union()** operation.

Now, we execute the union operation. After loading the necessary library, we call the **union()** function, passing **df1** and **df2** as arguments, and assign the resultant combined data frame to a new object, **df_all**:

```
library(dplyr)

#return all rows that occur in either data frame
df_all <- union(df1, df2)

#view resulting data frame
df_all

team points
1 A 14
2 A 19
3 A 25
4 B 40
5 B 34
6 B 38
7 B 17
8 C 14
```

9 C 10
10 D 11
11 D 15
12 D 10
13 E 32
14 E 28
15 E 27

Upon reviewing the resulting data frame, **df_all**, we observe that it contains a total of 15 rows. This count is highly informative. Given that the input data frames collectively contained 16 rows (8 + 8), the resulting 15 rows confirm that exactly one duplicate row was identified and removed during the union process. This removed row corresponds precisely to the redundant (Team A, Points 14) entry found in the second row of the original **df1**. Every other unique row from both **df1** and **df2** has been successfully included, demonstrating the core utility of the **union()** function for achieving a true set union.

Advanced Considerations and Best Practices

While the **union()** function is highly effective, its successful implementation relies on strict structural equivalence between the input data frames. Beyond the simple requirement of having the same number of columns, users must pay close attention to the consistency of column names and data types, as inconsistencies can lead to errors or silent data corruption.

One common pitfall is differing column names. If **df1** uses "Team_ID" and **df2** uses "teamID", the **union()** function will fail because it requires an exact match for column names. A necessary best practice before performing the union is to standardize the column nomenclature across all input data frames, perhaps using functions like **rename()** from **dplyr** or base R's column indexing. Furthermore, ensuring that corresponding columns have compatible data types (e.g., merging a numeric column with a character column is likely to coerce the entire resulting column to character, potentially complicating later calculations) is paramount for data integrity.

For operations involving extremely large datasets, performance considerations become relevant. While R is optimized, the process of identifying and removing duplicates across millions of rows can be computationally intensive. In such scenarios, if the user is absolutely certain that one data frame (e.g., **x**) contains only unique records and the second data frame (**y**) contains no overlap with **x**, using the faster **union_all()** combined with an external de-duplication step (if required later) might be a performance optimization. However, for most standard analytical tasks, relying on the built-in efficiency and safety of **union()** is the recommended approach.

Summary and Further Resources

The **union()** function within the **dplyr** package is an indispensable tool for combining data frames in R, specifically designed to produce a comprehensive set of unique rows from two source datasets. By adhering to the principles of set theory, it guarantees that redundant observations are efficiently eliminated, resulting in a clean, consolidated data structure ready for analysis.

When the inclusion of duplicate rows is necessary, the alternative function, **union_all()**, provides the stacking capability without the overhead of de-duplication. Mastering the choice between these two functions based on whether uniqueness is desired or all records must be retained is key to advanced data manipulation in R.

For users seeking to deepen their understanding of data frame manipulation, or to explore additional set operations like **intersect()** (which returns only shared rows) or **setdiff()** (which returns rows present in the first set but not the second), the complete official documentation for the **dplyr** package provides exhaustive details on these powerful functions.

Note: You can find the complete documentation for the union() function from the **dplyr** package online.

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

<!--

Featured Posts

-->