# How to Round Values in Specific Columns Using dplyr

Authored by

**stats writer**

November 20, 2025

In the world of data analysis and statistical reporting, precise control over numerical values is paramount. Often, raw data contains excessive decimal places which can clutter reports, reduce readability, and sometimes even lead to minor computational instability or confusion. To address this, data professionals rely heavily on rounding techniques. In the R ecosystem, especially when working with structured tabular data, the process of rounding specific column values is a frequent requirement.

Using the highly efficient **dplyr** package, rounding values in specific columns is achieved through the `round()` function applied via transformation helpers. This procedure requires specifying both the target column(s) and the desired number of decimal places. The basic syntax involves using `mutate()` to create or modify columns. For instance, a simple approach for a single column would be `df %>% mutate(column_name = round(column_name, 2))`, which rounds the specified values to two decimal places. However, for multiple columns, the modern **dplyr** approach utilizes the powerful `across()` function, enabling uniform application across many variables simultaneously.

## The Power of `dplyr` and the `mutate`/`across` Functions

The **dplyr** package, central to the Tidyverse, offers a consistent and highly readable grammar for complex data manipulation tasks. When dealing with column transformations like rounding, the combination of `mutate()` and `across()` is the recommended standard. The `mutate()` function modifies or adds variables, while `across()` allows us to apply the same transformation function--in this case, `round()`--to a dynamically selected or explicitly named set of columns. This framework significantly streamlines code compared to performing individual column operations.

We will examine two primary methods for rounding numerical data within a data frame using the **dplyr** library. The first method focuses on surgical precision, targeting only a specific vector of named columns. The second method provides broad coverage, identifying and rounding all columns based on their data type, ensuring every numeric variable adheres to the specified precision standard.

You can use the following methods to round values in specific columns of a data frame using the **dplyr** package in R:

## Technique 1: Targeting Specific Columns for Rounding

This technique is employed when an analyst needs absolute control over which columns receive the rounding treatment. By providing a character vector of column names to the `across()` function, we explicitly limit the scope of the `round()` operation. This is critical for datasets containing mixed numeric types where some must retain high precision (e.g., technical measurements) while others need simplification (e.g., financial reporting figures). The code below demonstrates how to target

'sales' and 'returns' specifically.

**Method 1: Round Values in Specific Columns**

```
library(dplyr)

#round values in 'sales' and 'returns' columns to 2 decimal places
df_new <- df %>% mutate(across(c('sales', 'returns'), round, 2))
```

## Technique 2: Applying Rounding Across All Numeric Data Types

When consistency is required across all measured variables in a dataset, applying the rounding transformation dynamically is far more efficient than listing every column name. This second technique leverages the `where()` selection helper within `across()`, using the predicate function `is.numeric` to identify all suitable columns automatically. This guarantees that columns like identifiers or dates, which are often stored as non-numeric types, are safely ignored, preventing data corruption.

This dynamic approach ensures scalability; if new numeric variables are added to the source data frame, the rounding script remains valid without modification. This is essential for robust data pipelines that process incoming data streams where schema changes might occur. We instruct **dplyr** to find all columns that satisfy the `is.numeric` condition and apply the `round` function to them, setting the desired precision level (e.g., 2 decimal places).

**Method 2: Round Values in All Numeric Columns**

```
library(dplyr)

#round values in all numeric columns to 2 decimal places
df_new <- df %>% mutate(across(where(is.numeric), round, 2))
```

## Setting Up the Demonstration Data Frame

To provide a clear, practical contrast between these two methods, we initialize a sample dataset using the base R function `data.frame()`. This data frame, named `df`, includes both a categorical variable ('store') and three numeric variables ('sales', 'returns', 'promos'), all containing raw values with varying, high precision decimal places. This setup allows us to rigorously test the selectivity and breadth of the **dplyr** operations.

The 'store' column, being character data, serves as a crucial control--it should remain entirely

unaffected by any of the numeric rounding operations. The 'sales', 'returns', and 'promos' columns, on the other hand, require attention. By viewing the initial state of `df`, we establish the baseline precision against which we will compare the results of the rounding scripts, ensuring that the transformations are applied correctly according to the specified decimal precision.

The following examples show how to use each method in practice with the following data frame in R:

**#create data frame**
**df <- data.frame(store=c('A', 'A', 'A', 'B', 'B', 'C', 'C', 'C'),**
**sales=c(4.352, 6.5543, 7.5423, 9.22111, 4.332, 9.55, 8.0094, 7.2),**
**returns=c(1.2324, 2.6654, 3.442, 6.545, 8.11, 8.004, 7.545, 6.0),**
**promos=c(12.11, 14.455, 10.277, 23.51, 20.099, 29.343, 30.1, 45.6))**

#view data frame
df

store sales returns promos
1 A 4.35200 1.2324 12.110
2 A 6.55430 2.6654 14.455
3 A 7.54230 3.4420 10.277
4 B 9.22111 6.5450 23.510
5 B 4.33200 8.1100 20.099
6 C 9.55000 8.0040 29.343
7 C 8.00940 7.5450 30.100
8 C 7.20000 6.0000 45.600

## Example 1: Round Values in Specific Columns Using dplyr

### Practical Application: Rounding Specific Columns

This first detailed example applies Technique 1, focusing solely on rounding the 'sales' and 'returns' columns to two decimal places. The explicit definition of the target columns ensures that the 'promos' column, despite being numeric, is intentionally excluded from the transformation. This level of selectivity is paramount in scenarios where different numeric variables in the same dataset demand varied precision levels.

Executing the `mutate(across(c('sales', 'returns'), round, 2))` command generates the `df_new` data frame. A detailed examination of the output confirms that all original values in the

'sales' and 'returns' columns have been correctly adjusted to two decimal places, adhering to standard rounding rules (e.g., 6.5543 is rounded down to 6.55, while 8.0094 is rounded up to 8.01). This demonstrates the effective implementation of the targeted transformation.

By comparing the 'promos' column in the original data frame `df` to the updated data frame `df_new`, we can visually confirm that its values--such as 14.455 and 10.277--have retained their initial high precision. This validates the effectiveness of the `across()` function when used with an explicit column vector, providing precise control necessary for advanced data handling.

The following code shows how to round the values in the **sales** and **returns** columns to 2 decimal places:

**library(dplyr)**

#round values in 'sales' and 'returns' columns to 2 decimal places
df_new <- df %>% mutate(across(c('sales', 'returns'), round, 2))

#view updated data frame
df_new

store sales returns promos
1 A 4.35 1.23 12.110
2 A 6.55 2.67 14.455
3 A 7.54 3.44 10.277
4 B 9.22 6.54 23.510
5 B 4.33 8.11 20.099
6 C 9.55 8.00 29.343
7 C 8.01 7.54 30.100
8 C 7.20 6.00 45.600

Notice that the values in the **sales** and **returns** columns are rounded to 2 decimal places while all other columns have remain unchanged.

## Example 2: Round Values in All Numeric Columns Using dplyr

### Practical Application: Rounding All Numeric Columns

This second detailed example showcases the utility of dynamic column selection using `where(is.numeric)`. The goal here is to enforce a consistent two-decimal-place precision across every numeric column, including 'sales', 'returns', and 'promos'. This method is invaluable when

preparing data for publications or standardized reports where all continuous variables must adhere to a predefined visual standard.

By executing `mutate(across(where(is.numeric), round, 2))`, the transformation is applied efficiently across the entire subset of numeric data. The resulting `df_new` confirms the comprehensive nature of this approach. While 'sales' and 'returns' are rounded as before, the 'promos' column is now also affected: for example, 14.455 is rounded to 14.46, and 20.099 becomes 20.10. This illustrates how the selection helper correctly identified all appropriate columns.

The successful transformation of all three numeric columns using a single line of concise **dplyr** code highlights the benefits of using predicate functions. This not only ensures data consistency but significantly reduces the manual overhead associated with maintaining lists of column names, especially in environments where the source data structure may frequently change. The 'store' column, being non-numeric, remains correctly isolated from the rounding operation.

The following code shows how to round the values in all of the numeric columns to 2 decimal places:

**library(dplyr)**

#round values in all numeric columns 2 decimal places
df_new <- df %>% mutate(across(where(is.numeric), round, 2))

#view updated data frame
df_new

store sales returns promos
1 A 4.35 1.23 12.11
2 A 6.55 2.67 14.46
3 A 7.54 3.44 10.28
4 B 9.22 6.54 23.51
5 B 4.33 8.11 20.10
6 C 9.55 8.00 29.34
7 C 8.01 7.54 30.10
8 C 7.20 6.00 45.60

Notice that the values in all three numeric columns of the data frame have been rounded to 2 decimal places.

## Summary and Best Practices for Rounding

Mastering column-wise operations in **dplyr** is essential for efficient data preparation in R. The `mutate()` and `across()` functions provide powerful tools for enforcing numerical precision, whether through explicit naming or dynamic selection based on data type.

When choosing between the specific column targeting method (Technique 1) and the broad numeric column application (Technique 2), analysts should prioritize data integrity and future scalability. Technique 1 guarantees that only intended columns are modified, providing maximum safety for complex datasets. Technique 2 ensures uniform standards and minimizes maintenance in automated pipelines that handle variable schema inputs.

Ultimately, the ability to selectively or broadly apply functions like `round()` using the **dplyr** framework significantly streamlines data cleaning and preparation workflows, leading to more readable code and more trustworthy analytical outputs. Utilizing these modern R techniques is a cornerstone of effective data science practice.