Handling Missing Data in R: Replacing NA Values with the Mean using dplyr

Authored by

Mohammed looti

October 28, 2025

Introduction to Handling Missing Data in R

In the realm of data analysis, encountering missing values, often denoted as **NA values** in the R programming language, is a common challenge. These missing data points can significantly impact the reliability and validity of analyses if not handled appropriately. One widely adopted strategy for dealing with numerical missing data is **mean imputation**, where NA values are replaced by the mean of their respective columns. This article will guide you through effective methods for performing mean imputation using the powerful dplyr and tidyr packages in R.

The Tidyverse, a collection of R packages designed for data science, provides an intuitive and efficient framework for data manipulation. Specifically, **dplyr** offers a consistent set of verbs that help you solve the most common data manipulation challenges, while **tidyr** focuses on tidying data, including functions for managing missing values. Together, these packages streamline the process of data cleaning and preparation.

Understanding how to systematically address missing data is a fundamental skill for any data professional. This tutorial will demonstrate three distinct approaches to replace NA values with the column mean: targeting a single column, specifying multiple columns, and applying the imputation across all numeric columns in a data frame.

Core Methodologies for Mean Imputation with `dplyr` and `tidyr`

The dplyr and tidyr packages provide a highly flexible and readable syntax for data transformation. The central idea involves chaining operations using the pipeline operator (`%>%`), applying `mutate()` to create or modify columns, utilizing `across()` for applying functions to multiple columns, and finally using `replace_na()` from `tidyr` to perform the imputation. This combination allows for selective or broad imputation based on your specific data requirements.

Here are the primary syntaxes for replacing NA values with the column mean, catering to different imputation scopes:

Method 1: Replace NA values with Mean in One Column

df %>% mutate(across(col1, ~replace_na(., mean(., na.rm=TRUE))))

This method is ideal when you need to apply mean imputation to a specific variable within your dataset. You simply specify the target column's name, such as `col1`, directly within the `across()` function. The lambda function `~replace_na(., mean(., na.rm=TRUE))` then calculates the mean of that column, ignoring existing NAs, and uses it to fill any missing entries.

Method 2: Replace NA values with Mean in Several Columns

df %>% mutate(across(c(col1, col2), ~replace_na(., mean(., na.rm=TRUE))))

For scenarios requiring imputation across a predefined set of columns, this syntax offers efficiency. You provide a vector of column names (e.g., `c(col1, col2)`) to the `across()` function, and the imputation is performed independently for each column within that vector. This ensures that each column's missing values are replaced by its own mean, not a global mean.

Method 3: Replace NA values with Mean in All Numeric Columns

df %>% mutate(across(where(is.numeric), ~replace_na(., mean(., na.rm=TRUE))))

When your goal is to apply mean imputation broadly to all quantitative variables in your dataset, the `where(is.numeric)` helper function becomes incredibly useful. This predicate selects all columns that are of a numeric type, ensuring that non-numeric columns (like character or factor variables) remain untouched. This is particularly helpful in larger datasets with many numerical features, automating the selection process.

In all these methods, the core of the imputation logic lies within `~replace_na(., mean(., na.rm=TRUE))`. The `.` symbol represents the current column being processed by `across()`. The `mean(., na.rm=TRUE)` component calculates the mean of that column, explicitly ignoring any existing NA values (`na.rm=TRUE`) to ensure a valid mean is computed. This calculated mean is then passed to `replace_na()` to fill in the missing entries within that specific column.

Setting Up the Illustrative Dataset

To practically demonstrate these imputation techniques, we will work with a simple data frame in R. This dataset simulates real-world scenarios where missing observations are present across different columns. Before we proceed with the examples, ensure you have the `dplyr` and `tidyr` packages loaded in your R session.

#create data frame
df <- data.frame(player=c('A', 'B', 'C', 'D', 'E'),
points=c(17, 13, NA, 9, 25),
rebounds=c(3, 4, NA, NA, 8),
blocks=c(1, 1, 2, 4, NA))

#view data frame
df

player points rebounds blocks
1 A 17 3 1
2 B 13 4 1
3 C NA NA 2
4 D 9 NA 4
5 E 25 8 NA

As observed in the output, our `df` data frame contains missing values (NA) in the `points`, `rebounds`, and `blocks` columns. The `player` column, being categorical, has no missing values. Our subsequent examples will show how to systematically address these missing entries using the methods outlined above, starting with the simplest case.

Example 1: Replacing NA Values with Mean in One Column

Let's begin by focusing on a single column, `points`. Our objective is to replace the NA value in this column with the calculated mean of the existing non-missing values in `points`. This method provides fine-grained control over which specific variables undergo imputation.

library(dplyr)
library(tidyr)

#replace NA values in points column with mean of points column
df <- df %>% mutate(across(points, ~replace_na(., mean(., na.rm=TRUE))))

#view updated data frame
df

player points rebounds blocks
1 A 17 3 1
2 B 13 4 1
3 C 16 NA 2
4 D 9 NA 4
5 E 25 8 NA

Before executing the code, let's manually calculate the mean of the `points` column's non-missing values: (17 + 13 + 9 + 25) / 4 = 64 / 4 = 16. The code successfully identifies the single NA value in the `points` column (for player 'C') and replaces it with 16.

It is important to note that this operation specifically targets the `points` column. All other columns, namely `rebounds` and `blocks`, retain their original NA values and other entries, demonstrating

the precise control offered by this method. This approach is ideal for focused data cleaning tasks.

Example 2: Replacing NA Values with Mean in Several Columns

Next, we will extend our imputation efforts to multiple selected columns. This scenario is common when certain variables are known to require similar handling for missing data. Here, we aim to impute the mean for both the `points` and `blocks` columns simultaneously.

library(dplyr)
library(tidyr)

#replace NA values in points and blocks columns with their respective means
df <- df %>% mutate(across(c(points, blocks), ~replace_na(., mean(., na.rm=TRUE))))

#view updated data frame
df

player points rebounds blocks
1 A 17 3 1
2 B 13 4 1
3 C 16 NA 2
4 D 9 NA 4
5 E 25 8 2

In this example, the NA value in `points` is replaced by 16 (as calculated previously), and the NA value in `blocks` is imputed. Let's calculate the mean of the `blocks` column: (1 + 1 + 2 + 4) / 4 = 8 / 4 = 2. Consequently, the missing value in `blocks` for player 'E' is replaced by 2.

This method showcases the power of applying transformations to a vector of column names, allowing for efficient and targeted data cleaning when multiple variables require similar imputation logic. The `rebounds` column, not specified in `across()`, remains unchanged, preserving its original NA values.

Example 3: Replacing NA Values with Mean in All Numeric Columns

For comprehensive data preparation, you might need to apply mean imputation to every numeric column in your dataset. This approach is particularly useful in larger datasets where manually listing all numeric columns would be cumbersome and error-prone. We will now apply this powerful technique to our example data frame.

library(dplyr)

```
library(tidyr)

#replace NA values in all numeric columns with their respective means
df <- df %>% mutate(across(where(is.numeric), ~replace_na(., mean(., na.rm=TRUE))))

#view updated data frame
df

player points rebounds blocks
1 A 17 3 1
2 B 13 4 1
3 C 16 5 2
4 D 9 5 4
5 E 25 8 2
```

Upon executing this code, you will observe that all NA values within the `points`, `rebounds`, and `blocks` columns have been replaced by their respective column means. We've already confirmed the means for `points` (16) and `blocks` (2). Let's verify the mean for the `rebounds` column: (3 + 4 + 8) / 3 = 15 / 3 = 5. Both NA values in the `rebounds` column (for players 'C' and 'D') are correctly imputed with 5.

Crucially, the `player` column, being a character variable, remains entirely untouched by this operation, as it is not identified as `numeric` by the `where(is.numeric)` helper function. This highlights the robustness and intelligence of `across()` when combined with selection helpers, ensuring that only appropriate columns are modified.

Conclusion and Further Reading

Handling missing data is a critical step in any data preparation workflow. The dplyr and tidyr packages in R offer powerful and flexible tools to perform mean imputation, whether for a single column, a selection of columns, or all numeric variables within a data frame. By leveraging functions like `mutate()`, `across()`, `where()`, and `replace_na()`, you can efficiently and effectively clean your datasets, making them ready for more advanced analysis.

While mean imputation is a straightforward and commonly used technique, it is essential to understand its limitations. It can reduce variance and distort relationships between variables, especially if the proportion of missing data is high or if the data are not missing completely at random. For more sophisticated approaches to missing data, consider exploring other imputation methods such as median imputation, mode imputation, hot-deck imputation, or model-based imputation techniques like K-Nearest Neighbors (KNN) or multiple imputation.

Mastering these fundamental data wrangling skills will significantly enhance your capabilities as a data analyst or scientist, enabling you to produce more robust and reliable analytical results.

Additional Resources

To deepen your understanding of data manipulation in R and explore other common tasks, consider the following tutorials and documentation:

`dplyr` Official Documentation
`tidyr` Official Documentation
A Review of Missing Data Methods (PDF)
Handling Missing Data with `tidyr` in R for Data Science