How to Easily Replace NA Values with the Median in R

Authored by

stats writer

November 28, 2025

Dealing with NA (Not Available) values is a fundamental step in data cleaning and analysis using R. One of the most robust and commonly employed techniques for addressing missing numerical data is imputation using the median. Replacing missing values with the central tendency ensures that the overall distribution of your dataset remains minimally affected by outliers, unlike mean imputation.

While base R offers methods for handling missing data, utilizing the powerful functions within the **tidyverse** ecosystem, specifically the dplyr and tidyr packages, provides a highly efficient and readable workflow for this task. This guide demonstrates three distinct methods for replacing **NA** values with the column median across various scenarios in your data frame.

Introduction to Handling Missing Data in R

In statistical computing, particularly within the **R** environment, the presence of missing data (represented as **NA**) can severely skew analytical results and restrict the use of various modeling techniques. Effective handling of these gaps is paramount for maintaining data integrity and ensuring the reliability of statistical inferences.

When working with numerical variables, median imputation is generally preferred over mean imputation because the **median** is resistant to extreme values or outliers. Unlike the mean, which can be heavily pulled by a single high or low value, the median is the middle value, making it a more stable and representative measure of central tendency for skewed distributions.

Essential Tidyverse Functions for Imputation

The standard workflow for performing median imputation leverages the functionality of the **tidyverse** packages. This process involves combining data manipulation functions from **dplyr** with data cleaning functions from **tidyr**. Understanding the role of each function is key to mastering these techniques:

`mutate()`: This **dplyr** function is used to modify or create new columns within a dataset. We use it here to update the existing columns containing **NA** values.

`across()`: A core **dplyr** helper function that allows us to apply the same operation (in this case, median calculation and replacement) to multiple columns simultaneously, streamlining the code significantly.

`median(..., na.rm=TRUE)`: This calculates the median of the specified vector. The crucial argument `na.rm=TRUE` ensures that existing **NA** values are ignored during the calculation, providing an accurate median based only on available data points.

`replace_na()`: Sourced from **tidyr**, this function takes the result of the calculated median and uses it to fill the previously identified **NA** locations within the column(s) being processed.

Core Implementation Strategies

The implementation of median imputation can be tailored to the specific needs of the analysis by controlling which columns are affected. We outline three powerful strategies, moving from the most selective to the most generalized approach, allowing for maximum control over your data frame structure.

Method 1: Replacing NA Values with Median in One Column

This syntax is used when you need to strictly target one specific column for imputation. By passing a single column name to the `across()` function, we ensure that only the missing values within that variable are replaced by its corresponding median.

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

Method 2: Replacing NA Values with Median in Several Specific Columns

If a predefined set of columns requires cleaning, you can provide a character vector of column names using the `c()` function within `across()`. This approach maintains precision by imputing only the columns explicitly listed, while leaving all other variables untouched.

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

Method 3: Replacing NA Values with Median in All Numeric Columns

For scenarios where all numerical variables must be cleaned, the most efficient method is to utilize the selection helper `where(is.numeric)`. This robust command automatically identifies all columns containing numeric data types and applies the median imputation to each one independently, saving significant time in large datasets.

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

Preparing the Sample Data for Demonstration

To clearly illustrate how these three methods operate in practice, we will use a small sample **R data frame**, named `df`. This dataset simulates real-world conditions by including missing

observations (represented by **NA**) across the numeric performance columns (points, rebounds, and blocks). The `player` column, being categorical, is intentionally free of missing values.

```
#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
```

Our subsequent examples will show how the respective column medians are calculated from the non-missing entries and then systematically used to fill the observed NA values, transforming this sparse dataset into a complete one.

Example 1: Targeted Imputation in One Column

This example utilizes Method 1 to address the missing data in only the **points** column. We first load the necessary **dplyr** and **tidyr** libraries. The code then isolates the **points** column, calculates its median (which is 15, derived from 17, 13, 9, and 25), and replaces the single **NA** entry in that column with the calculated median value.

```
library(dplyr)
library(tidyr)
```

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

```
#view updated data frame
df
```

```
player points rebounds blocks
1 A 17 3 1
```

2 B 13 4 1
3 C 15 NA 2
4 D 9 NA 4
5 E 25 8 NA

The output clearly shows that the missing score for Player C in the **points** column has been replaced with 15. Crucially, the remaining columns, **rebounds** and **blocks**, retain their original NA values, demonstrating the precision of this targeted, single-column approach.

Example 2: Imputing Several Specific Columns

Building on Method 2, we now extend the imputation to two specific variables: **points** and **blocks**. The procedure calculates the median for each column independently (Median of **points** = 15; Median of **blocks** = 1.5). The replacement is then executed simultaneously across both selected columns using the column-specific median.

```
library(dplyr)
library(tidyr)

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

#view updated data frame
df
```

player points rebounds blocks
1 A 17 3 1.0
2 B 13 4 1.0
3 C 15 NA 2.0
4 D 9 NA 4.0
5 E 25 8 1.5

The output confirms that **points** is imputed with 15 and **blocks** with 1.5. Since the **rebounds** column was intentionally excluded from the list of target columns passed to `across()`, its **NA** entries are still present. This demonstrates how to precisely manage imputation tasks when you only want a subset of numeric variables affected.

Example 3: Batch Imputation Across All Numeric Columns

Finally, we apply Method 3, the most automated approach, to complete the dataset. By using

`where(is.numeric)`, we instruct **R** to automatically iterate through all columns that can hold numeric data--**points**, **rebounds**, and **blocks**--and apply the median imputation tailored to each column's unique median (Median of **rebounds** = 4).

library(dplyr)
library(tidyr)

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

#view updated data frame
df

player points rebounds blocks
1 A 17 3 1.0
2 B 13 4 1.0
3 C 15 4 2.0
4 D 9 4 4.0
5 E 25 8 1.5

The final data frame is now fully imputed. Notice that the NA values in **rebounds** have been replaced by the median value of 4, alongside the updates to **points** and **blocks**. Crucially, the non-numeric **player** column correctly remained unchanged, demonstrating the power and safety of using conditional selection with `where(is.numeric)`.

Summary and Further Learning

Median imputation provides a stable and reliable method for handling missing numerical data in **R**, especially when using the efficient piping syntax of the **tidyverse**. By leveraging the combined power of `mutate`, `across`, and `replace_na`, data analysts can quickly and precisely clean their datasets, preparing them for robust statistical analysis.

Mastering these three methods--imputing a single column, selecting multiple specific columns, or processing all numeric columns--equips you with the necessary flexibility to tackle virtually any data cleaning challenge involving missing values.

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