How to Easily Calculate Summary Statistics in R with dplyr

Authored by

stats writer

November 21, 2025

Analyzing and understanding large datasets often hinges on the ability to efficiently calculate summary statistics. These fundamental measures--including central tendency metrics like the mean and median, and variability metrics such as the standard deviation--provide critical insights into data distributions. In the statistical programming environment of R, the process of data manipulation and summarization is revolutionized by the functionalities offered by the dplyr package.

The dplyr package, a core component of the Tidyverse, delivers a high-performance, consistent, and readable grammar for data wrangling. Its primary advantage lies in its intuitive structure, which allows analysts to formulate complex data operations using a clear sequence of steps, often chained together using the pipe operator (**%>%**). This standardized approach drastically simplifies tasks that were traditionally verbose or complex in base R, making data cleaning and statistical calculation accessible and efficient for both novice and expert users.

This comprehensive guide will detail the exact methodology required to calculate a full suite of summary metrics across all numeric variables within a data frame, leveraging the combined power of dplyr and tidyr. We will focus specifically on utilizing the flexible across() function within summarise() to achieve streamlined, comprehensive statistical outputs, concluding with a practical, step-by-step example.

Essential Syntax for Comprehensive Summary Statistics

To generate a wide array of descriptive statistics simultaneously across multiple variables, a specialized syntax is required. This approach ensures that we automatically target only the appropriate columns--specifically those containing numeric data--and then apply a custom list of statistical functions to each. This methodology is particularly useful when dealing with data frames containing numerous features, where manual selection would be time-consuming and error-prone.

The following code block demonstrates the elegant solution offered by combining the **summarise()**, **across()**, and **where()** functions from **dplyr**, alongside the necessary reshaping provided by **tidyr**. Notice how the use of the pipe operator (**%>%**) creates a logical workflow, moving the data frame sequentially through the summarization and reshaping stages.

library(dplyr)
library(tidyr)

df %>% summarise(across(where(is.numeric), .fns =
list(min = min,
median = median,
mean = mean,
stdev = sd,

```
q25 = ~quantile(., 0.25),
q75 = ~quantile(., 0.75),
max = max))) %>%
pivot_longer(everything(), names_sep='_', names_to=c('variable', '.value'))
```

This sequence of operations first ensures that the calculation step is executed robustly, regardless of the column names or number of numeric variables present in the data frame (represented here as **df**). Subsequently, the output, which initially presents the statistics in a wide format, is converted into a more readable long format, making variable comparison and reporting significantly easier. This combination represents a best practice for automated descriptive statistics generation in modern R programming.

Understanding the Core summarise() and across() Functions

The foundation of this operation rests upon the summarise() function, which is the essential tool provided by dplyr for reducing a data frame down to a single row (or one row per group, if grouping is applied). In its traditional usage, one would explicitly define the statistic and the variable, such as `summarise(avg_points = mean(points))`. However, when dealing with dozens of variables and multiple statistics, this manual approach becomes cumbersome and prone to transcription errors.

To overcome this limitation, we utilize the powerful across() function. The `across()` function allows us to apply the same set of operations to multiple columns simultaneously. Within our syntax, `across(where(is.numeric), ...)` instructs summarise() to identify all columns that satisfy the condition of being numeric (`where(is.numeric)`) and then proceed to apply the defined list of functions (`.fns`) to each of those selected columns. This automated selection mechanism ensures robustness, as the code will execute correctly even if the structure of the input data frame changes, provided the relevant columns remain numeric.

The list of functions passed to the `.fns` argument calculates a set of widely accepted summary statistics, providing a complete descriptive overview of the data distribution for each numeric variable:

Minimum value: Identifies the smallest observation in the column.
Median value: Represents the 50th percentile, often a more robust measure of central tendency than the mean.
Mean value: Calculates the arithmetic average of the observations.
Standard deviation (stdev): Measures the amount of variation or dispersion of a set of values, using the `sd()` function.
25th percentile (q25): The first quartile, indicating the value below which 25% of the data falls.
75th percentile (q75): The third quartile, indicating the value below which 75% of the data falls.

Maximum value: Identifies the largest observation in the column.

It is important to note the use of lambda functions (`~quantile(., 0.25)`) for calculating quartiles, as this syntax permits passing custom arguments (like the probability value 0.25 or 0.75) directly within the across() structure.

Restructuring Output with tidyr::pivot_longer()

When summarise() executes the summarized calculation using `across()`, the resulting data frame is typically in a wide format. For example, if we summarize variables A and B, the output will have columns such as `A_min`, `A_mean`, `B_min`, and `B_mean`. While this format is computationally useful, it is often difficult to read, print, and compare metrics efficiently, especially when the number of metrics and variables is high. The solution lies in reshaping the output using the pivot_longer() function from the **tidyr** package, which transforms the data into a long, or tidy, format.

The arguments used in the pivot_longer() call are highly specific and powerful. `pivot_longer(everything(), names_sep='_', names_to=c('variable', '.value'))` instructs R to take all columns (`everything()`) and condense them. Crucially, `names_sep='_'` tells R that the original column names (like `points_mean`) are separated by an underscore. The resulting pieces of the name are then mapped to new column names defined by `names_to=c('variable', '.value')`.

In this mapping, the first part of the original column name (e.g., 'points') is stored under the new column `variable`, signifying the original column being summarized. The special marker `.value` tells **tidyr** that the second part of the original column name (e.g., 'mean' or 'stdev') should actually become the names of new columns, effectively creating a column for each statistic. This transformation results in an output where each row corresponds to a single variable, and columns define the metrics (min, mean, median, etc.), providing maximum readability and facilitating immediate comparison of summary statistics across different features.

Practical Example: Setting Up the Data Frame

To fully illustrate the efficacy of this dplyr and **tidyr** syntax, we will utilize a sample dataset. Imagine we are working with performance metrics for basketball players, tracking their performance across several key metrics: points scored, assists recorded, and rebounds collected. The data frame also includes a categorical variable, `team`, which will be ignored by the `where(is.numeric)` selection criterion.

We begin by constructing the data frame in R. This setup ensures we have a mix of data types, demonstrating the utility of the `across()` function in automatically identifying the columns suitable for statistical calculation.

#create data frame
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
points=c(12, 15, 19, 14, 24, 25, 39, 34),
assists=c(6, 8, 8, 9, 12, 6, 8, 10),
rebounds=c(9, 9, 8, 10, 8, 4, 3, 3))

#view data frame
df

team points assists rebounds
1 A 12 6 9
2 A 15 8 9
3 A 19 8 8
4 A 14 9 10
5 B 24 12 8
6 B 25 6 4
7 B 39 8 3
8 B 34 10 3

The resulting data frame, **df**, contains eight observations and four variables. Our objective is to calculate the minimum, maximum, mean, median, standard deviation, and both the 25th and 75th percentiles exclusively for the `points`, `assists`, and `rebounds` columns. Since the `team` column is character-based, it will be automatically excluded from the statistical calculations by the `where(is.numeric)` selector.

Executing the Comprehensive Summarization

With the data frame established, we now proceed to execute the powerful Tidyverse pipeline. We must first ensure that both the **dplyr** and **tidyr** packages are loaded into the R session using the `library()` function. The subsequent piped command chain calculates all seven desired summary statistics, applies them across the numeric columns via across(), and finally reshapes the output into a clean, long table format using pivot_longer().

Running the code block below demonstrates how efficiently this complex summarization is handled. The intermediate result from the summarise() and across() combination is transformed immediately by pivot_longer(), yielding a final output that is structured specifically for reporting and analysis.

library(dplyr)
library(tidyr)

```
#calculate summary statistics for each numeric variable in data frame
df %>% summarise(across(where(is.numeric), .fns =
list(min = min,
median = median,
mean = mean,
stdev = sd,
q25 = ~quantile(., 0.25),
q75 = ~quantile(., 0.75),
max = max))) %>%
pivot_longer(everything(), names_sep='_', names_to=c('variable', '.value'))
```

```
# A tibble: 3 x 8
variable min median mean stdev q25 q75 max

1 points 12 21.5 22.8 9.74 14.8 27.2 39
2 assists 6 8 8.38 2.00 7.5 9.25 12
3 rebounds 3 8 6.75 2.92 3.75 9 10
```

Interpreting the Descriptive Statistics Output

The resulting output is a clean tibble (a modern R data frame) where each row corresponds to a specific variable, and the columns represent the calculated summary statistics. This structure facilitates immediate comparative analysis of the player performance metrics. By examining the `points` row, for instance, we can quickly grasp the distribution characteristics.

For the `points` variable, we observe significant variation: the minimum score recorded is **12**, while the maximum is **39**. The central tendency measures, mean (**22.8**) and median (**21.5**), are relatively close, suggesting a distribution that is not severely skewed. Furthermore, the standard deviation of **9.74** indicates a wide spread around the mean. The quartile values (q25 = **14.8** and q75 = **27.2**) establish the Interquartile Range (IQR), defining the central 50% of the scores.

Similarly, analyzing the `assists` column shows a much tighter distribution. With a standard deviation of just **2.00**, the observations are clustered closely around the mean of **8.38**. This robust, automated statistical output provides the analyst with high-quality descriptive metrics instantly, eliminating the need for repetitive manual calculation across numerous variables. This approach is highly recommended for exploratory data analysis (EDA) in any R project utilizing the Tidyverse principles.

For users seeking more in-depth functionality or specialized applications of the functions discussed, consulting the official documentation is essential. The capabilities of across(), for

example, extend far beyond simple summarization, allowing for conditional data transformation and modification across selected columns. Mastery of these Tidyverse tools is crucial for modern, efficient data processing in R.