How to Easily Count Words in a String Using R

Authored by

stats writer

November 20, 2025

In R, you can count the number of words in a string by using the str_count() function from the stringr package. This function takes a character vector and counts the number of words in each string. The result is returned as an integer vector with the same length as the input vector.

When dealing with textual data analysis or natural language processing (NLP) tasks within the R programming environment, accurately determining the length of text segments is a fundamental requirement. Whether you are analyzing document complexity, preparing features for machine learning models, or performing basic data cleaning, counting words efficiently is essential. Fortunately, R offers several robust methods--ranging from fundamental native functions to specialized packages--to accomplish this task with high precision.

Why Word Counting Matters in Text Analysis

The ability to quantify the number of words in a string is not just a trivial exercise; it forms the basis for several analytical metrics. Word counts are crucial for calculating readability scores, defining document length constraints, and establishing features used in advanced text modeling. Understanding the various methodologies available in R allows users to select the most appropriate and performant solution based on the scale and complexity of their data processing needs.

While the fundamental goal is simple--to count delimited tokens--the approach chosen significantly impacts execution speed, especially when processing large corpora. We will explore three primary ways to achieve word counts in R, highlighting their syntax, efficiency, and suitability for different scenarios. These methods cover the core capabilities of Base R, the highly optimized **stringi** package, and the user-friendly **stringr** package, which adheres to the Tidyverse principles.

For all subsequent examples, we assume the string to be analyzed is stored in a variable named **my_string**. Regardless of the method chosen, the output will consistently be an integer or integer vector representing the total count of words identified in the input string or strings.

There are three highly reliable and widely used methods you can employ to count the number of words in a string in R, each offering a distinct balance of simplicity and performance:

Understanding the Fundamentals of String Manipulation

Before diving into specific functions, it is helpful to grasp the underlying mechanism: counting words usually involves splitting the input string wherever a delimiter (typically a space) occurs, resulting in a list or vector of individual word tokens. The final count is then derived from the length of this resultant list.

The three methods below demonstrate how this tokenization and counting process is streamlined

using different R libraries. Method 1 (Base R) executes these steps explicitly, while Methods 2 and 3 encapsulate them into single, highly efficient functions.

Method 1: Use Base R for Explicit Tokenization

This approach utilizes functions native to R and is excellent for understanding the core logic. It involves using the **strsplit()** function to break the string into a list of words based on the space delimiter, followed by **lengths()** to determine the size of that list.

lengths(strsplit(my_string, ' '))

Method 2: Optimize Performance with the stringi Package

The stringi package is known for its incredible speed and reliability, often offering the fastest execution times for complex string operations in R. It leverages the International Components for Unicode (ICU) library, making it ideal for high-volume text processing and cross-language analysis.

library(stringi)

stri_count_words(my_string)

Method 3: Adopt the Tidyverse Standard with stringr

The **stringr** package, part of the Tidyverse ecosystem, provides consistent function naming and syntax, making string manipulation intuitive and readable. When counting words, **str_count()** typically utilizes regular expression patterns to identify and count word boundaries efficiently.

library(stringr)

str_count(my_string, 'w+')

Each of these methods will return a numerical value that represents the number of words in the string called **my_string**. The consistency in the final result across these diverse methods provides flexibility for developers to choose based on project dependencies or performance requirements.

The following sections delve into practical implementations, showing how to use each of these methods successfully in practice and interpreting the resulting output.

Detailed Example 1: Implementing Base R Functions

Using Base R functions to count words is a transparent and highly portable method, as it requires

no external package installations. This approach explicitly highlights the steps of tokenization and length calculation, making the process easy to debug and understand for R novices.

The following code shows how to count the number of words in a string using the **lengths** and **strsplit** functions from Base R. The core idea is to split the string wherever a space character (' ') appears, converting the string into a list of individual words. The **lengths()** function then calculates the size of the resulting character vector, providing the word count.

```
# Create string for demonstration
my_string <- 'this is a string with seven words'

# Count number of words in string by splitting on space delimiter
lengths(strsplit(my_string, ' '))
```

```
7
```

From the output, we can clearly see that there are seven words in the string. While effective, it is important to note a slight limitation of the Base R approach: if the string contains multiple consecutive spaces, **strsplit()** might introduce empty strings into the resulting list, which could complicate the count if not handled carefully. However, for well-formed strings, this method is straightforward and reliable.

Detailed Example 2: Utilizing the stringi Function

For applications where speed is paramount--such as analyzing massive datasets or performing real-time text processing--the **stringi** package is the optimal choice. Its function, **stri_count_words()**, is highly optimized and often outperforms methods using Base R or **stringr** for pure word counting tasks.

The following code demonstrates how to count the number of words in a string using the highly efficient **stri_count_words** function from the stringi package in R. This function inherently handles common complexities like multiple spaces, punctuation, and varying locale settings, ensuring an accurate and fast count.

```
library(stringi)

# Define the sample string
my_string <- 'this is a string with seven words'

# Count number of words using the specialized stringi function
stri_count_words(my_string)
```

7

The **stri_count_words()** function automatically tokenizes the input based on standard word boundary definitions, making the code clean and requiring minimal input beyond the string itself. For R users focused on maximizing processing efficiency and adhering to advanced international standards for text handling, **stringi** is the preferred library.

Detailed Example 3: Mastering Regular Expressions with stringr

The **stringr** package provides a unified and readable interface for string manipulation, making it a favorite among Tidyverse users. Its primary word-counting function, **str_count()**, requires a regular expression pattern to define what constitutes a "word," offering great flexibility in defining word boundaries.

The following code shows how to count the number of words in a string using the **str_count** function from the stringr package in R. Crucially, we use the pattern `w+` to define a word, which enhances the function's precision and applicability.

library(stringr)

# Define the sample string
my_string <- 'this is a string with seven words'

# Count number of words using the str_count function and regex pattern
str_count(my_string, 'w+')

7

From the output, we confirm that there are seven words in the string. The power of this method lies in its use of the regular expression **w+**. The metacharacter `w` matches any "word" character (alphanumeric characters and the underscore), and the quantifier `+` indicates that the match should include one or more of these characters in a row. This ensures that punctuation or multiple spaces between words do not disrupt the accurate counting of actual word tokens.

Note that we used the regular expression **w+** to match non-word characters with the **+** sign to indicate one or more in a row. This specific regex pattern is the standard, robust way to define a word when using regular expression-based counting methods in R.

Handling Vectors of Strings

An important consideration for practical data analysis is that text data rarely comes as a single

string; it is usually structured as a vector of strings or a column within a data frame.

Note: In each of these preceding examples, we counted the number of words in a single string, but critically, each method--**lengths(strsplit())**, **stri_count_words()**, and **str_count()**--will also work seamlessly with a vector of strings, returning an integer vector where each element corresponds to the word count of the respective string in the input vector. This vectorized performance is fundamental to R's efficiency in data analysis.

Performance Comparison and Use Case Scenarios

While all three methods yield the same correct result for standard inputs, their performance characteristics and suitability for different tasks vary considerably. Choosing the right tool depends on whether readability, performance, or dependency requirements are the highest priority.

Base R (lengths(strsplit())): Pros: No external dependencies required; excellent for simple, non-production tasks; conceptually transparent. **Cons:** Generally the slowest method, especially for large vectors; less robust handling of edge cases (e.g., leading/trailing spaces).

stringi (stri_count_words()): Pros: Exceptionally fast and optimized; built upon industrial-strength Unicode libraries; best choice for high-volume NLP tasks. **Cons:** Requires installing an external package.

stringr (str_count()): Pros: Excellent integration with the Tidyverse workflow; highly readable syntax; flexible use of regular expression for nuanced definitions of "word." **Cons:** Slightly slower than **stringi**, though still very fast; requires package installation.

For most data analysis tasks within the Tidyverse framework, the **stringr** method provides the best balance of performance and readability. However, if dealing with massive text corpora where every millisecond counts, leveraging the power of **stringi** is highly recommended.