Use str_split in R (With Examples)

Authored by

Mohammed looti

November 4, 2025

Introduction to String Splitting in R: The stringr Package

String manipulation is an absolutely fundamental skill required for effective data cleaning and preparation within the R programming environment. Raw datasets frequently contain concatenated information--such as full addresses, combined names, or mixed codes--that must be precisely parsed and separated into distinct, manageable components for analysis. Failing to properly isolate these components leads to inaccurate modeling and flawed reporting.

To address this crucial need, the stringr package, a cornerstone utility within the popular Tidyverse collection, offers a highly consistent and intuitive framework for working with character strings. Unlike base R functions, **stringr** streamlines complex text operations, making the process of extracting, replacing, and splitting data far more approachable and less error-prone.

This comprehensive guide dives deep into the two primary functions provided by **stringr** for dividing character vectors based on a specified delimiter or pattern: **str_split()** and **str_split_fixed()**. While both achieve the goal of separating strings, they yield drastically different output formats--one a flexible list and the other a fixed-dimension matrix. Mastering this distinction is paramount for designing efficient and robust data workflows in R.

Preparing the Data for Demonstration

To effectively illustrate the capabilities of these string splitting tools, we will begin by establishing a practical working environment. We create a sample data frame named `df`, which accurately simulates common real-world scenarios. In this setup, the `team` column contains multiple pieces of information (player names) bundled together in a single character vector, separated by the clear delimiter " & ".

The objective is simple yet essential: to use the splitting functions to isolate the individual names from the `team` column, transforming a single compound variable into two distinct variables ready for further processing or relational mapping.

 # Create the sample data frame
df <- data.frame(team=c('andy & bob', 'carl & doug', 'eric & frank'),
points=c(14, 17, 19))

View the resulting data frame structure
df

team points
1 andy & bob 14
2 carl & doug 17

3 eric & frank 19

Our subsequent steps will demonstrate how **str_split()** and **str_split_fixed()** handle the separation based on the " & " delimiter, showcasing why choosing the correct function is crucial depending on the intended output structure.

The Flexible Approach: Utilizing str_split() for List Output

The str_split() function is specifically designed for scenarios where the number of resulting pieces following the split operation may vary across different rows or observations. For instance, if some entries contained two names and others contained three, **str_split()** would handle this irregularity gracefully.

Because of this inherent flexibility in output length, **str_split()** returns the result as a nested `list` structure. This list contains character vectors, where each individual vector corresponds to the separated parts of a single original input string. While powerful for handling non-uniform data, this list format often necessitates additional steps, such as unlisting or iterative application, if the data needs to be integrated back into a rectangular data frame.

The primary syntax for the function requires only the input string and the separating pattern:

str_split(string, pattern)

The arguments are defined as follows:

string: This is the input character vector (e.g., `df$team`) that you intend to divide.
pattern: This defines the delimiter--the specific sequence of characters or regular expression--upon which the split should occur.

Practical Application 1: Decomposing Strings into a List Structure

In this first practical example, we apply the **str_split()** function to the `team` column, specifying the " & " sequence as our splitting criterion. We must first ensure the `stringr` library, which manages these text operations, is loaded into the R session.

library(stringr)

# Split the string in the *team* column on " & "
str_split(df$team, " & ")

]
"andy" "bob"

]
"carl" "doug"

]
"eric" "frank"

The resulting output confirms that the function returns a `list` containing three elements. Crucially, each element corresponds directly to a row from the original data frame and now holds a character vector comprising the separated player names. This list format, while preserving all data pieces regardless of their count, generally requires subsequent processing steps--such as using functions like `unlist()` or iterating through the list--if the goal is to align the results into new columns within the data frame.

The Structured Approach: Leveraging str_split_fixed() for Matrix Output

For data processing goals where the output must be immediately rectangular and ready for integration into a data frame, the str_split_fixed() function is the superior choice. This function rigorously enforces a fixed, two-dimensional structure--specifically, an R matrix--with a consistent, user-defined number of columns.

The power of **str_split_fixed()** lies in its guarantee of output dimensions, which is controlled by the critical `n` parameter. If an input string results in fewer pieces than specified by `n`, the function automatically pads the resulting matrix columns with empty strings (`""`). Conversely, if an input string contains too many pieces, the excess results are silently discarded. This mechanism ensures that the output matrix always maintains the exact dimension defined by `n`, eliminating structural inconsistencies.

The syntax for **str_split_fixed()** explicitly incorporates the dimension constraint:

str_split_fixed(string, pattern, n)

The required arguments for this structured split are:

string: The input character vector to be processed.
pattern: The delimiter or regular expression defining the split point.
n: This integer specifies the exact number of pieces (columns) the resulting matrix must contain.

Practical Application 2: Generating a Consistent Fixed Matrix

We now apply the **str_split_fixed()** function to our `team` column. Since we have prior knowledge that each string contains exactly two names separated by the delimiter, we set the fixed number of

desired pieces, `n`, to 2. This mandates that the output must be a two-column matrix.

library(stringr)

# Split the string and force the output into a 2-column matrix
str_split_fixed(df$team, " & ", 2)

"andy" "bob"
"carl" "doug"
"eric" "frank"

The outcome is a clean, rectangular matrix that is immediately useful. The first column () reliably holds the first player's name, and the second column () holds the second player's name. This fixed output structure bypasses the need for complex list manipulation, making **str_split_fixed()** a highly efficient component in common data preparation pipelines.

Advanced Workflow: Seamlessly Integrating Results into a Data Frame

The most common application of **str_split_fixed()** is to decompose compound variables and append the results as new columns to the original data frame. Because the output is a perfectly structured matrix, R allows for direct column assignment using subsetting notation. We assign the results generated by **str_split_fixed()** to the next available column indices (in this case, columns 3 and 4) of our data frame `df`.

library(stringr)

# Split the string and append the resulting matrix directly to the data frame
df <- str_split_fixed(df$team, " & ", 2)

View the updated data frame
df
team points V3 V4
1 andy & bob 14 andy bob
2 carl & doug 17 carl doug
3 eric & frank 19 eric frank

The final data frame, `df`, is successfully augmented with two new columns, automatically labeled `V3` and `V4` by R. Column `V3` now isolates the first player's name, and `V4` isolates the second player's name. This workflow demonstrates the immense efficiency gained by using the structured output of str_split_fixed() for transforming data within the stringr package ecosystem.

Conclusion and Further Resources

Choosing between **str_split()** and **str_split_fixed()** fundamentally depends on whether you prioritize flexibility (handling variable output lengths via a list) or structure (guaranteeing fixed column dimensions via a matrix). For most data cleaning tasks involving integration into a data frame, **str_split_fixed()** provides the most straightforward and efficient path.

We highly recommend exploring the broader capabilities of the **stringr** package and delving into the intricacies of regular expressions for those committed to mastering advanced text manipulation within the R programming environment.

How to Perform Partial String Matching in R