How to Check if a String Contains Multiple Substrings in R

Authored by

stats writer

November 19, 2025

When working with text data in R, a common requirement is determining whether a specific character string contains one or more patterns or substrings. While the function str_detect() from the popular `stringr` package is highly effective for checking single substring presence, addressing the requirement for **multiple substrings** often necessitates a more complex combination of base R functions or specialized techniques. This approach typically involves iterating through a vector of desired patterns and applying a logical aggregation to the results.

The core challenge lies not just in finding a single pattern, but in efficiently determining the relationship between multiple potential patterns within a target string. We are usually interested in two distinct logical outcomes: does the string contain **at least one** of the specified substrings (the "OR" condition), or does the string contain **all** of the specified substrings (the "AND" condition)? Achieving this requires leveraging functions that facilitate vectorized operations and logical reduction, such as grepl, sapply(), and apply().

This comprehensive guide explores the robust methods available in base R for performing advanced multiple substring checks. We will detail the structure and logic necessary to generate a definitive logical vector that accurately reflects the presence or absence of complex substring combinations within columns of a data frame, ensuring high performance and clarity in data analysis tasks. We specifically focus on methods that utilize the `grepl` function for pattern matching combined with logical aggregation across the resulting matrix.

Understanding the Core R Components for String Matching

To successfully check for multiple substrings without relying on external packages, we must combine several base R functions that handle iterative application and logical reduction. The primary function for pattern matching is `grepl`, which stands for "grep logical." This function determines whether a pattern is found within a character vector, returning a simple `TRUE` or `FALSE` for each element. However, `grepl` is designed to check one pattern against many strings, or many patterns against one string sequentially. To check many patterns against many strings simultaneously, we must introduce the `*apply` family of functions.

The `sapply()` function is instrumental here because it applies a function (in our case, `grepl`) over a list or vector and attempts to simplify the result into an array or vector. By iterating `sapply()` over our vector of desired substrings (`find_strings`), we create an intermediate matrix where each column corresponds to one substring check, and each row corresponds to a specific entry in the target data column (e.g., `df$team`). This matrix contains the logical results of every single pattern match.

Once this matrix of logical outcomes is generated, we utilize the `apply()` function. The `apply()` function is critical for performing operations across the margins of an array or matrix. By specifying the margin `1`, we tell `apply()` to operate row-wise. Since each row represents a single string from

our original data frame, operating row-wise allows us to aggregate the logical results for that specific string. This structure is the foundational basis for differentiating between the "OR" and "AND" conditions in multiple substring detection.

Implementing the "OR" Condition: Checking for Any Substring

The "OR" condition is met when the target string contains **at least one** of the defined substrings. In the context of the logical matrix created by `sapply` and `grepl`, this means we need to check if any of the logical values in a given row (representing a single string) are `TRUE`. This aggregation is efficiently handled by the built-in R function `any()`, which returns `TRUE` if at least one element of a logical vector is `TRUE`.

When implementing this logic, the syntax encapsulates the entire process: running the individual checks via `sapply`, feeding the resulting matrix into `apply()`, and using `any()` as the aggregating function across the rows (margin 1). This method ensures that the final output is a single logical vector that can be directly added back to the original data frame, effectively marking whether the "OR" criterion was satisfied for each observation.

This approach provides a powerful and readable way to implement complex conditional filtering based on multiple patterns. The resulting syntax for this "OR" check is concise and highly effective:

Method 1: Check if String Contains One of Several Substrings

df$contains_any <- apply(sapply(find_strings, grepl, df$team), 1, any)

This particular syntax checks if each string in the **team** column contains *any* of the strings specified in the vector of strings called **find_strings**. The use of `any` dictates that even if only one pattern matches, the overall result for that specific row will be `TRUE`.

Implementing the "AND" Condition: Checking for All Substrings

In contrast to the "OR" condition, the "AND" condition requires that the target string contains **every single** substring defined in the search vector. Logically, this means that for a given row in our intermediate logical matrix, all values must be `TRUE`. If even one substring is missing, the entire result for that row must be `FALSE`.

To achieve this aggregation, we replace the `any()` function with the `all()` function within the `apply()` structure. The `all()` function returns `TRUE` only if all elements of the input logical vector are `TRUE`. If one or more elements are `FALSE`, `all()` immediately returns `FALSE`, perfectly executing the "AND" logic.

This subtle but crucial change in the aggregation function transforms the query from an inclusive search (finding at least one match) to a restrictive search (requiring all matches). This is particularly useful in data processing scenarios where composite criteria must be met, such as identifying records that contain multiple specific labels or keywords simultaneously.

Method 2: Check if String Contains Several Substrings

df$contains_any <- apply(sapply(find_strings, grepl, df$team), 1, all)

This particular syntax checks if each string in the **team** column contains all of the strings specified in the vector of strings called **find_strings**. The stringent requirement imposed by `all` means that only rows satisfying every condition will yield a `TRUE` result.

Establishing the Dataset for Demonstration

To illustrate these two distinct methods--the inclusive "OR" check and the restrictive "AND" check--we will utilize a simple, predefined data frame in R. This dataset contains team names and their corresponding points, providing a practical context for applying our string detection logic. The variation in the `team` column ensures that we have examples that satisfy both criteria, only one criterion, or neither criterion.

The following example code block demonstrates the creation and structure of our example data frame, which will be the basis for subsequent practical applications:

#create data frame
df = data.frame(team=c('Good East Team', 'Good West Team', 'Great East Team',
'Great West Team', 'Bad East Team', 'Bad West Team'),
points=c(93, 99, 105, 110, 85, 88))

#view data frame
df

team points
1 Good East Team 93
2 Good West Team 99
3 Great East Team 105
4 Great West Team 110
5 Bad East Team 85
6 Bad West Team 88

As observed in the output, the `df` data frame consists of six records. The `team` column contains

varying combinations of the words "Good," "Great," "Bad," "East," and "West." Our goal in the upcoming examples will be to search this column for the presence of the strings "Good" and "East," first using the lenient "OR" logic, and then using the strict "AND" logic.

Practical Application 1: Detecting "Good" OR "East" (The "Any" Logic)

In our first practical example, we aim to identify all teams that are either classified as "Good" **or** are designated as "East." This is a common scenario in data filtering where we want to cast a wide net, including records that satisfy any one of several conditions. We define our search vector `find_strings` to contain the two patterns we are looking for: "Good" and "East".

The core of the operation remains the combined use of sapply() and apply(), crucially terminating with the `any` function. The `sapply` step first runs the grepl check for "Good" against all teams, and then separately for "East" against all teams. These two results are compiled into a logical matrix. The subsequent `apply(..., 1, any)` step then looks across each row of this matrix--that is, across the results for each individual team name--and returns `TRUE` if at least one of the two checks passed.

The syntax below executes this conditional check and adds the resulting logical vector as a new column, `good_or_east`, to our data frame:

#define substrings to look for
find_strings <- c('Good', 'East')

#check if each string in team column contains either substring
df$good_or_east <- apply(sapply(find_strings , grepl, df$team), 1, any)

#view updated data frame
df

team points good_or_east
1 Good East Team 93 TRUE
2 Good West Team 99 TRUE
3 Great East Team 105 TRUE
4 Great West Team 110 FALSE
5 Bad East Team 85 TRUE
6 Bad West Team 88 FALSE

Decoding the "Any" Logic Output

The resultant column `good_or_east` clearly demonstrates the inclusive nature of the `any` logic. For

instance, Row 2 ("Good West Team") returns `TRUE` because it contains "Good", even though it misses "East". Similarly, Row 3 ("Great East Team") returns `TRUE` because it contains "East", despite missing "Good". Row 1 ("Good East Team") returns `TRUE` as it satisfies both conditions.

Conversely, the rows that returned `FALSE` are those that contain neither of the specified substrings. Row 4 ("Great West Team") and Row 6 ("Bad West Team") are both designated `FALSE` because they lack both "Good" and "East."

The interpretation of the new `good_or_east` column is summarized as follows:

TRUE if team contains "Good" **or** "East" (or both).
FALSE if team contains neither "Good" nor "East".

This result validates that the `apply(..., 1, any)` structure successfully implemented the disjunctive logical operation required for checking if a string contains one of several substrings.

Practical Application 2: Detecting "Good" AND "East" (The "All" Logic)

Our second application focuses on the restrictive "AND" condition. Here, we only want to identify teams that are classified as "Good" **and** are designated as "East." This approach is used when highly specific data points must be isolated based on the simultaneous presence of multiple key characteristics. We use the exact same definition for `find_strings`: the patterns "Good" and "East".

The procedure follows the same pattern as before, using `sapply` and `grepl` to generate the intermediate logical matrix. The critical difference is the final aggregation step: we substitute `any` with `all`. The function apply(), when instructed to use `all` across margin 1 (rows), ensures that only if the checks for both "Good" **and** "East" return `TRUE` for a specific team name will the final result for that record be `TRUE`.

This strict logical requirement drastically reduces the number of matches compared to the inclusive "OR" search, providing a targeted subset of the data. The following code block demonstrates this implementation and the resulting output:

```
#define substrings to look for
find_strings <- c('Good', 'East')

#check if each string in team column contains either substring
df$good_and_east <- apply(sapply(find_strings , grepl, df$team), 1, all)

#view updated data frame
df
```

team points good_and_east
1 Good East Team 93 TRUE
2 Good West Team 99 FALSE
3 Great East Team 105 FALSE
4 Great West Team 110 FALSE
5 Bad East Team 85 FALSE
6 Bad West Team 88 FALSE

Analyzing the Results of the "All" Logic

Reviewing the output column `good_and_east` confirms the strictness of the "AND" condition. Only Row 1, corresponding to "Good East Team," returns `TRUE`. This is the sole record in the entire data frame where both the substring "Good" **and** the substring "East" are simultaneously present.

All other rows return `FALSE` because they fail at least one of the conditions. For example, "Good West Team" fails the "East" check, and "Great East Team" fails the "Good" check. The resulting vector accurately reflects the confluence of all required patterns within a single string.

The interpretation of the new `good_and_east` column is:

TRUE if team contains "Good" **and** "East".
FALSE if team does not contain both "Good" and "East".

Notice that only one **TRUE** value is returned since there is only one team name that contains the substring "Good" **and** the substring "East." This demonstrates the powerful filtering capability achieved by leveraging the combination of sapply(), grepl, and `apply(..., 1, all)` for multi-criteria substring analysis in R.