Learning Guide: Selecting Columns by String Content in R

Authored by

Mohammed looti

October 28, 2025

Introduction to Advanced Column Selection in R

Selecting specific columns from a data frame based on patterns in their names is a fundamental task for data preparation and analysis in R. When dealing with large datasets where manual column naming is impractical or inefficient, leveraging pattern matching becomes essential. The most efficient and readable way to accomplish this sophisticated selection is by utilizing the functions provided within the powerful dplyr package, which forms the cornerstone of the Tidyverse suite of tools.

This tutorial focuses on how to use the `select()` function in conjunction with the `matches()` helper to employ regular expressions (regex) for dynamic column subsetting. This method allows analysts to target columns based on specific substrings they contain, offering far greater flexibility than traditional indexing methods. We will explore two primary scenarios: selecting columns that contain a single, specific string, and expanding that capability to select columns that match any one of several defined strings using the logical **OR** operator.

Understanding the Dplyr Framework and Pattern Matching

Effective column selection relies heavily on the structure provided by dplyr. The core of the Tidyverse methodology is the pipe operator (`%>%`), which allows operations to be chained together sequentially, making complex data manipulation steps highly legible. The `select()` function is specifically designed to work within this pipeline, providing a consistent interface for managing columns based on various criteria.

The key to dynamic string selection lies in the `matches()` helper function, which is designed to interpret its argument as a regular expression pattern. Regular expressions are sequences of characters that define a search pattern, and they provide incredible power for finding, replacing, or validating text strings. When `select()` encounters `matches()`, it iterates through every column name in the input data frame and returns `TRUE` only if the column name satisfies the complex pattern defined within the regex.

Method 1: Selecting Columns that Contain One Specific String

This is the simplest application of the pattern matching workflow. When you need to extract all columns that share a common identifiable element, such as a project identifier, a unit of measurement, or a specific label, you define that element as the regex pattern. Since the standard regex engine in R defaults to searching for the pattern anywhere within the string, simply providing the substring is sufficient to identify all relevant column headers.

The syntax below shows how to implement this method, piping the target data frame into the `select()` function and specifying the exact string we are searching for within `matches()`. This

approach is highly efficient and eliminates the need for manual indexing or long lists of column names.

```
df %>%
select(matches("string1"))
```

In this construction, `"string1"` represents the literal substring that must be present in the column name for it to be included in the resulting output. This method ensures consistency when working across different versions of a dataset where column order might change, but the naming conventions remain reliable.

Method 2: Selecting Columns that Contain One of Several Strings

A more advanced requirement involves selecting columns that satisfy one of several possible naming criteria. For example, a user might need all columns related to "sales" or all columns related to "profit." To handle these inclusive, multi-criteria searches, we must utilize the logical **OR** operator provided by regular expressions, which is represented by the vertical bar (**|**).

By placing the vertical bar between multiple search strings within the `matches()` function, we instruct the regex engine to return a match if the column name contains the string before the bar **OR** the string after the bar. This allows for the construction of a single, powerful command that replaces what might otherwise require multiple filtering steps or complex logical combinations outside of the Tidyverse framework.

```
df %>%
select(matches("string1|string2|string3"))
```

This technique is indispensable for comprehensive data wrangling, particularly when dealing with variables that must be grouped together for analysis but do not share a single, unified prefix or suffix. The ability to specify multiple patterns within one function call is a key advantage of using dplyr and regex together.

Setting Up the Sample Data Frame

To provide concrete illustrations of these two methods, we will create and utilize a simple data frame in R. This sample data, named `df`, contains several columns representing basketball team abbreviations, some of which share common letter patterns. Observing how the selection methods interact with these varied column names helps solidify the understanding of regex pattern behavior.

The data frame is constructed using the base R `data.frame()` function. Note the specific column

names: `mavs`, `cavs`, `hornets`, `spurs`, and `nets`. The goal of the upcoming examples is to programmatically isolate subsets of these columns using the pattern matching techniques described above, proving the efficacy of the `matches()` function.

#create data frame
df <- data.frame(mavs=c(12, 10, 14, 19, 22, 25, 29),
cavs=c(22, 41, 14, 15, 15, 19, 22),
hornets=c(8, 8, 12, 14, 15, 13, 12),
spurs=c(10, 12, 12, 16, 22, 28, 30),
nets=c(9, 7, 10, 22, 28, 23, 25))

#view data frame
df

mavs cavs hornets spurs nets
1 12 22 8 10 9
2 10 41 8 12 7
3 14 14 12 12 10
4 19 15 14 16 22
5 22 15 15 22 28
6 25 19 13 28 23
7 29 22 12 30 25

Example 1: Isolating Columns Containing "avs"

For our first practical demonstration, we aim to select only those columns that contain the string `"avs"`. This pattern matches both `mavs` and `cavs`. We begin by loading the necessary dplyr library to ensure access to the `select()` and `matches()` functions, and then apply the single-string pattern matching technique to our sample data frame `df`.

The following code shows how to use the **matches()** function to select only the columns that contain the string "avs" somewhere in their name. The concise nature of the pipe operator (`%>%`) and the clarity of the `select(matches())` syntax make the intent of the code immediately apparent, a hallmark of the Tidyverse design philosophy.

library(dplyr)

#select all columns that contain "avs" in the name
df %>%
select(matches("avs"))

mavs cavs
1 12 22
2 10 41
3 14 14
4 19 15
5 22 15
6 25 19
7 29 22

As demonstrated by the output, only the columns that contain `"avs"` in the name are returned. Specifically, `mavs` and `cavs` are the only columns that satisfy this criterion. This simple example confirms that the `matches()` function successfully identifies the provided substring regardless of its position within the column header.

Example 2: Isolating Columns Containing "avs" OR "ets"

In our second example, we apply the multi-criteria selection method. We want to retrieve columns that match the pattern `"avs"` or the pattern `"ets"`. This requires the use of the vertical bar (**|**) to combine the two search criteria into a single, unified regular expression.

The following code shows how to use the **matches()** function to select only the columns that contain "avs" or "ets" somewhere in their name. This powerful technique is crucial for selecting related variables that may have slightly different organizational naming structures. For instance, `nets` and `hornets` are both selected because they contain the `"ets"` substring, while `mavs` and `cavs` are selected based on the `"avs"` substring.

library(dplyr)

#select all columns that contain "avs" or "ets" in the name
df %>%
select(matches("avs|ets"))

mavs cavs hornets nets
1 12 22 8 9
2 10 41 8 7
3 14 14 12 10
4 19 15 14 22
5 22 15 15 28
6 25 19 13 23
7 29 22 12 25

The output includes `mavs`, `cavs`, `nets`, and `hornets`, confirming that the pattern `"avs|ets"` successfully used the logical **OR** operator to identify columns matching either of the specified criteria. It is important to emphasize that the vertical bar (**|**) is the standardized **OR** operator within R's regular expression syntax, and this pattern matching method provides a flexible and maintainable solution for complex column selection tasks.

Additional Resources for Dplyr and R Proficiency

While the `matches()` function offers the highest degree of flexibility through regular expressions, the dplyr package includes several other specialized helpers within `select()` that are useful for simpler pattern matching. These include `starts_with()`, `ends_with()`, and `contains()`, all of which streamline common selection tasks without requiring complex regex knowledge. Mastering the various selectors available within select() is critical for efficient data manipulation in R.

We encourage users to look beyond column selection and explore how these methods integrate seamlessly with other key dplyr functions, such as `filter()` for row subsetting, `mutate()` for variable creation, and `group_by()` for aggregation. The combination of these functions allows users to build highly robust and reproducible data transformation pipelines.

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