Understanding the PRXMATCH Function in SAS: A Comprehensive Guide with Syntax and Examples

Authored by

Mohammed looti

November 14, 2025

The **PRXMATCH function** is recognized as an indispensable utility within the **SAS** programming environment, specifically engineered for sophisticated text manipulation and pattern recognition. This function dramatically enhances an analyst's ability to handle unstructured or semi-structured data by enabling precise searches for complex patterns within character variables. Unlike simple string comparison tools, **PRXMATCH** leverages the power of regular expressions to locate structural matches.

The fundamental operation of this function involves scanning a designated source string for the first occurrence of a defined pattern. If a match is successfully identified, the function returns an integer representing the starting character position of that pattern within the source string (with position 1 being the first character). Conversely, if the search criteria are not met anywhere in the string, the function returns a value of 0. This binary outcome makes it exceptionally useful for creating conditional logic.

Integrating **PRXMATCH** into the **DATA step** is critical for numerous data management practices, including robust data cleaning, ensuring data validation rules are met, and executing high-level conditional processing. By mastering this function, SAS programmers gain precise control over text data that is often messy or inconsistent, leading to cleaner and more reliable analytical datasets.

Deciphering the Core Syntax and Regular Expression Power

At the heart of the **PRXMATCH function** lies the implementation of **regular expressions** (regex). Regex provides a specialized sequence of characters that defines a powerful search pattern. It is this reliance on regex that elevates **PRXMATCH** far beyond standard SAS string functions like `FIND` or `INDEX`, enabling searches based on complex structural criteria, character classes (e.g., numbers, letters, whitespace), or sequences, rather than being limited to fixed literal strings.

To execute the search, the function requires two positional arguments, following a precise structure:

PRXMATCH(regular expression, source)

Understanding the role of each argument is essential for successful implementation:

regular expression: This mandatory argument defines the exact pattern to be sought. It is critical that the pattern string is correctly enclosed within delimiters (commonly forward slashes, e.g., `/pattern/`). Furthermore, optional trailing modifiers, such as `i` for initiating a case-insensitive search, can be appended immediately after the closing delimiter to fine-tune the matching behavior.
source: This specifies the target variable or literal character string against which the compiled regular expression pattern will be evaluated. This is the data that SAS scans to find the position of

the pattern.

The next sections will transition from theoretical syntax to practical application, demonstrating the versatility of this function across different data analysis needs, including simple position finding, conditional flagging, and subsetting datasets based on pattern presence.

Establishing the Foundation: Sample Dataset Creation

To effectively demonstrate the capabilities and nuances of the PRXMATCH function, we will first establish a small, representative sample dataset. This dataset contains fictional basketball team names and their corresponding scores. A key characteristic of this data, which will be leveraged in the following examples, is the intentional inconsistency in the capitalization of the team names. This variability allows us to clearly differentiate between case-sensitive and case-insensitive pattern matching.

The following **SAS** code utilizes the **DATA step** and the `PROC PRINT` procedure to create and display our foundational dataset, named `my_data`. Observe the differing capitalization of "Mavs" and "Warriors" across the rows.

```
/*create dataset*/
data my_data;
input team $ points;
datalines;
Mavs 22
mavs 14
Warriors 23
Mavs 19
warriors 34
MAVS 40
WARRIORS 39
;
run;

/*view dataset*/
proc print data=my_data;
```

Obs	team	points
1	Mavs	22
2	mavs	14
3	Warriors	23
4	Mavs	19
5	warriors	34
6	MAVS	40
7	WARRIORS	39

Application 1: Locating the Exact Starting Position of a Pattern (Case-Sensitive)

One of the most straightforward and essential uses of PRXMATCH is pinpointing the precise starting index of a pattern within a text string. This first example illustrates a standard case-sensitive search. We aim to identify where the lowercase substring "avs" begins in the `team` variable and store this index in a new variable named `position`. Since we use the expression `/avs/` without any modifiers, the search maintains strict sensitivity to letter casing.

The execution of PRXMATCH yields two possible outcomes: if the pattern is successfully located, the function returns a positive integer corresponding to the character position where the match starts (remembering that SAS strings are 1-indexed). If the pattern is absent from the source string, the function returns the value 0. This integer result is invaluable for subsequent string manipulation or data quality checks.

```
/*create new dataset*/
data new_data;
set my_data;
position = prxmatch("/avs/", team);
run;

/*view new dataset*/
proc print data=new_data;
```

Obs	team	points	position
1	Mavs	22	2
2	mavs	14	2
3	Warriors	23	0
4	Mavs	19	2
5	warriors	34	0
6	MAVS	40	0
7	WARRIORS	39	0

The resulting output clearly demonstrates the case-sensitive nature of the search. Although rows containing "Mavs" (e.g., Row 1) contain the letters 'a', 'v', and 's', they match the lowercase pattern `/avs/` starting at position 2. Rows 3 and 7, which contain "Warriors" and "WARRIORS," do not contain the precise sequence of lowercase letters 'a', 'v', 's' and therefore correctly result in a **0**. This example highlights how the absence of a modifier enforces exact textual matching.

Application 2: Conditional Logic and Creating Binary Flags (Case-Insensitive)

A more powerful application of PRXMATCH involves leveraging its returned value (position index or 0) directly within conditional logic to create binary flags. This method streamlines data categorization and preparation for statistical modeling. In this scenario, we aim to generate a new variable, `mavs_flag`, which serves as a flag, assigning a value of **1** to any record where the team name contains "Mavs," irrespective of the capitalization used, and **0** otherwise.

To override the default case-sensitive behavior, we employ the powerful `i` modifier appended to the regular expression pattern, resulting in `/Mavs/i`. The inclusion of this modifier instructs the pattern matching engine to treat uppercase and lowercase letters identically. Within the DATA step, we wrap the PRXMATCH call inside an `IF/THEN` statement, assigning the flag value based on whether the function returns a positive integer (i.e., a match was found, indicating the position is greater than 0).

```
/*create new dataset*/
data new_data;
set my_data;
if prxmatch("/Mavs/i", team) > 0 then mavs_flag = 1;
else mavs_flag = 0;
run;

/*view new dataset*/
```

proc print data=new_data;

Obs	team	points	mavs_flag
1	Mavs	22	1
2	mavs	14	1
3	Warriors	23	0
4	Mavs	19	1
5	warriors	34	0
6	MAVS	40	1
7	WARRIORS	39	0

As demonstrated in the resulting table, every row containing any variant of "Mavs" (e.g., "Mavs," "mavs," or "MAVS") now possesses a `mavs_flag` value of 1. The rows containing "Warriors" are correctly assigned 0. This confirms the efficacy of the `i` modifier in enabling a robust **case-insensitive search**, a necessity when dealing with user-entered or inconsistently formatted text data in **SAS**. Should the analytical requirement demand strict capitalization, the modifier must be omitted, reverting to the default matching behavior.

Application 3: Efficient Data Subsetting and Filtering

Beyond data transformation and flagging, PRXMATCH is a highly efficient mechanism for filtering and subsetting large datasets based on the presence of a specific pattern. This example illustrates how to utilize the function within the **DATA step** to create a new dataset that includes only the records where the `team` variable contains the string "Mavs," maintaining the case-insensitive search logic established in the previous example.

In SAS programming, placing a conditional statement (like `IF condition;`) without an explicit `THEN OUTPUT` command causes SAS to output only those records where the condition evaluates to true. By evaluating whether `prxmatch("/Mavs/i", team)` returns a value greater than 0, we instruct the DATA step to effectively keep only the matching records. This technique is often cleaner and more direct than creating a temporary flag variable.

```
/*create filtered dataset*/
data mavs_only;
set my_data;
if prxmatch("/Mavs/i", team) > 0;
run;
```

```
/*view filtered dataset*/
proc print data=mavs_only;
```

Obs	team	points
1	Mavs	22
2	mavs	14
3	Mavs	19
4	MAVS	40

The resulting dataset, visualized above, confirms that the filtering operation was successful, retaining only the five original records that contained "Mavs" regardless of capitalization. This clearly illustrates the utility of PRXMATCH as a powerful, pattern-based tool for targeted data subsetting, significantly improving efficiency when dealing with specific textual criteria.

Conclusion and Next Steps for SAS Text Processing

The **regular expressions**-based **PRXMATCH** function stands out as an indispensable tool for advanced text processing within the **SAS** environment. Its ability to return the precise starting position of a match, or a definitive zero if no match is found, makes it uniquely suited for robust data quality checks, conditional logic, and the creation of highly targeted data subsets.

Throughout these examples, we demonstrated how PRXMATCH handles complexity, from simple case-sensitive substring location to more flexible, case-insensitive conditional flagging using modifiers. Mastering the syntax and the application of regular expression modifiers is key to unlocking maximum efficiency when handling real-world, often messy, text data.

To further expand proficiency in SAS text functions and data management, analysts are encouraged to explore related functions and techniques that complement PRXMATCH, such as PRXPARSE (for compiling patterns) and PRXCHANGE (for replacing matched patterns). These functions form a powerful suite for comprehensive text manipulation.

For users seeking to expand their proficiency with SAS text functions and data management techniques, the following resources provide guidance on other common programming challenges:

The following tutorials explain how to perform other common tasks in SAS: