How to Use “If Contains” in Power BI to Filter Data Easily

Authored by

stats writer

January 29, 2026

The ability to efficiently search for specific textual patterns within large volumes of data is fundamental to modern data analysis. In the context of Power BI, mastering the "If Contains" logic allows analysts to categorize records, flag exceptions, and filter information based on specific keywords or partial strings. This powerful conditional functionality is executed using DAX (Data Analysis Expressions), the formula language native to Power BI, Analysis Services, and Power Pivot in Excel.

The "If Contains" formula is not a single, dedicated function but rather a combination of functions--most notably `IF` and CONTAINSSTRING--that work together to evaluate a logical test. This structure provides a simple and robust way to search within a selected column for a specified substring, returning a user-defined result if the match is found. This mechanism is crucial for data preparation steps, enabling users to transform raw text into meaningful categorical metrics necessary for visualization and reporting.

By leveraging this methodology, users can significantly enhance the utility and navigability of their dataset. Whether the goal is to identify all customer notes containing the word "refund," or to flag product names that include a specific model code, the calculated column approach detailed below offers a highly performant and scalable solution within the Power BI environment. We will explore the precise syntax required in DAX to implement this conditional text search effectively.

Power BI: A Simple Formula for "If Contains"

The Core DAX Syntax for String Containment

To implement the "If Contains" logic within Power BI, we utilize the DAX language to create a new calculated column. A calculated column evaluates the formula row by row, providing a result for every entry in the table. The foundational function for this operation is CONTAINSSTRING, which efficiently checks if one string is contained within another. This function returns a boolean value (TRUE or FALSE), which is then processed by the `IF` function to return customized text or numerical outputs.

The standard syntax for checking string containment and returning a binary categorical result (like "Yes" or "No") requires wrapping the CONTAINSSTRING test inside an `IF` statement. This combination provides clear, human-readable labels for the resulting analysis. The structure allows you to define exactly what happens when the specified text is found (the "If True" result) and what happens when it is not found (the "If False" result).

Below is the standard syntax pattern used to create a new column that analyzes the content of an existing column, such as a team identifier or product code, returning a simple categorical flag based on the presence of a target substring.

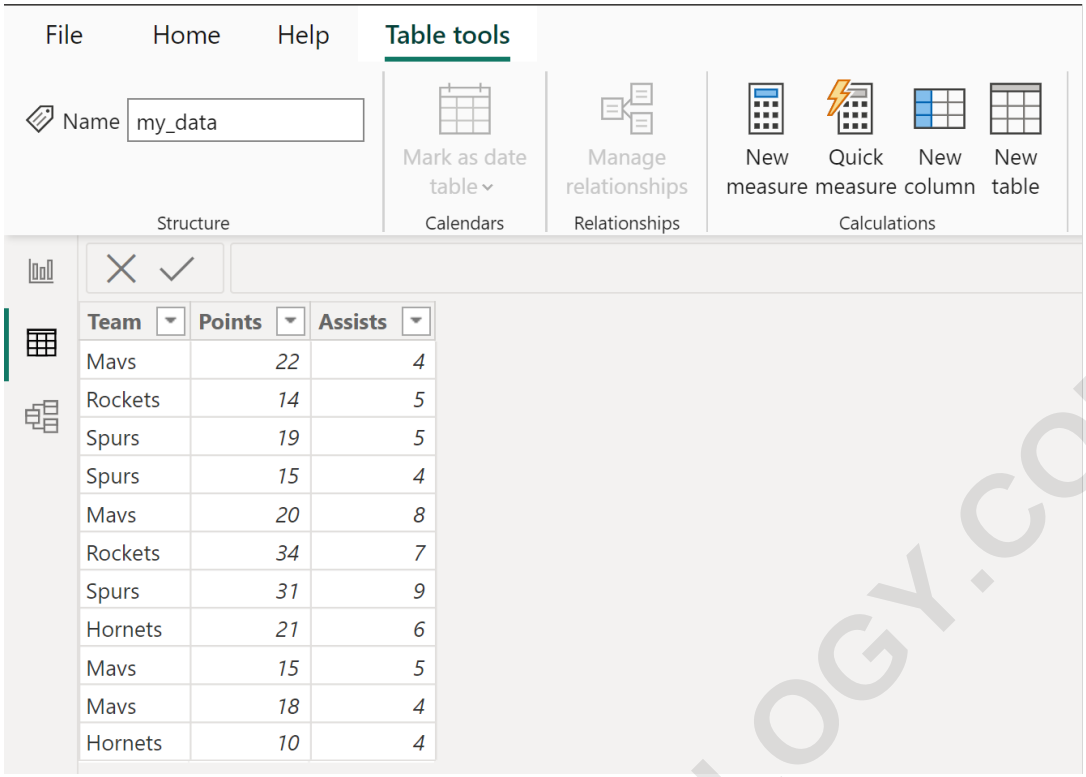
```
contains_ets =
IF(
CONTAINSSTRING('my_data', "ets"),
"Yes",
"No"
)
```

This specific formula creates a new calculated column named `contains_ets`. It performs a logical test on every row in the `Team` column within the table named `my_data`. If the text in that row contains the literal substring "ets," the column returns "Yes"; otherwise, it returns "No." It is crucial to remember that CONTAINSSTRING is case-insensitive, meaning it treats "Ets," "ets," and "ETS" as identical matches. This makes it ideal for general-purpose text scanning where case variation is common.

Step-by-Step Example: Implementing "If Contains" in Power BI

Understanding the theory is important, but practical application solidifies the concept. We will walk through a common scenario where you need to classify records based on partial matches in a dataset. Suppose we are working with sales data for a sports apparel company, and our data includes a table named **my_data**, which logs various basketball teams and player information. Our goal is to identify which teams belong to a specific division or contain a particular naming convention, represented here by the substring "ets."

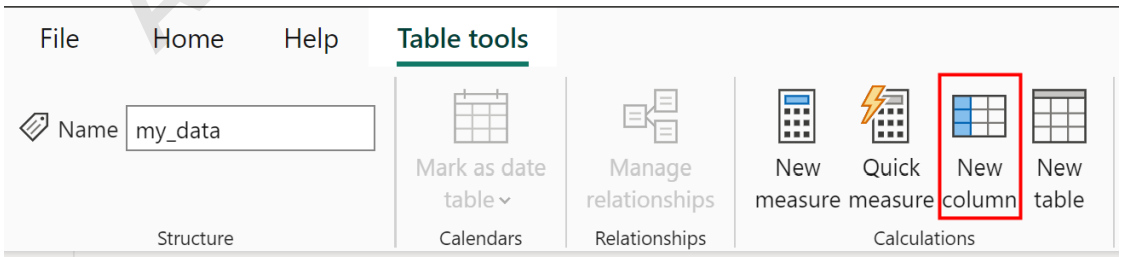
Consider the following sample table structure. This table, **my_data**, contains essential player details, including their associated team. We want to add a new column to facilitate easy filtering based on the team name.



Team	Points	Assists
Mavs	22	4
Rockets	14	5
Spurs	19	5
Spurs	15	4
Mavs	20	8
Rockets	34	7
Spurs	31	9
Hornets	21	6
Mavs	15	5
Mavs	18	4
Hornets	10	4

To begin the process of applying the "If Contains" formula, you must first navigate to the Data view or the Report view in Power BI Desktop. Since we are adding a permanent, row-level classification, the correct approach is to create a **New column** directly within the table definition. This action ensures that the calculation is performed once during data refresh and stored alongside the rest of the data, optimizing performance.

Once you have selected the target table (`my_data`), locate and click the **New column** option in the Table tools ribbon. This action opens the DAX formula bar, prompting you to define the calculation for the new column. The visual cue for this step is critical for users new to Power BI's data modeling capabilities.



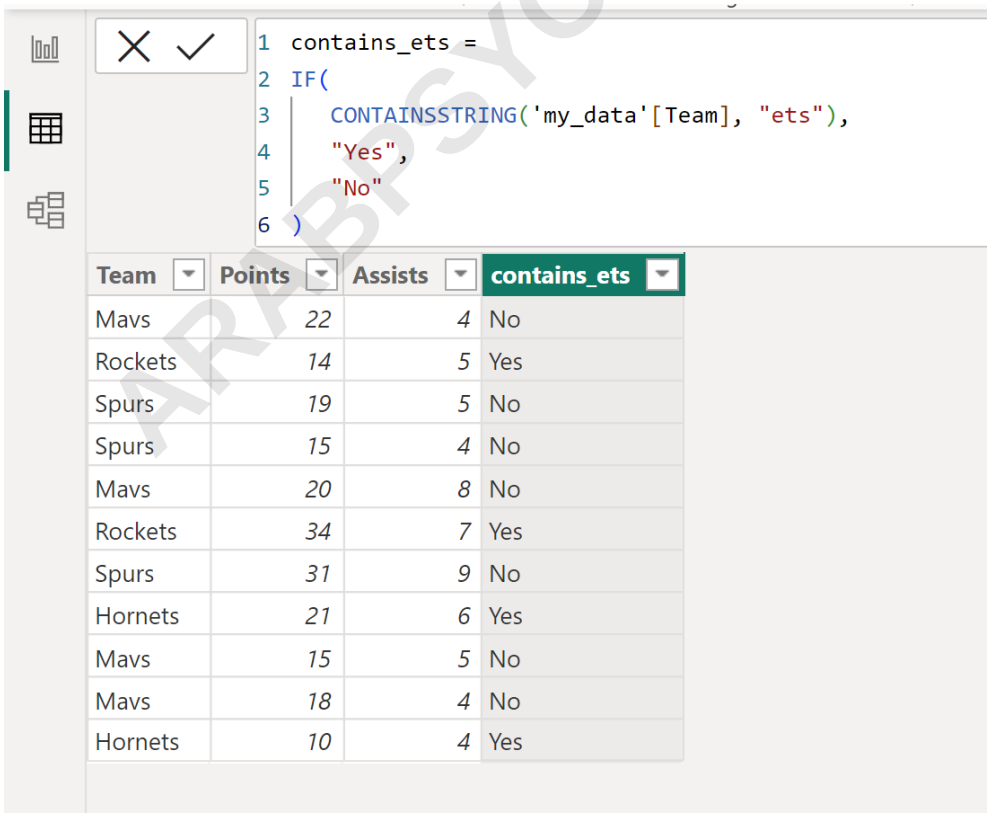
Applying the DAX Formula and Viewing Results

With the DAX formula bar active, you can now input the previously defined "If Contains" expression. Ensure that the table and column references match your dataset structure precisely. The syntax is:

contains_ets =
IF(
CONTAINSSTRING('my_data', "ets"),
"Yes",
"No"
)

Upon committing the formula, Power BI instantly calculates the results for every row. The new column, `contains_ets`, will appear in your data model, populated with either "Yes" or "No." This categorical output immediately makes it clear which teams satisfy the criteria--those containing the three letters "ets" anywhere in their name. For example, teams like "Nets" would return "Yes," while teams like "Bulls" would return "No."

The resulting table demonstrates the effectiveness of the formula. This new column serves as a powerful flag that can be dragged into visualizations or used as a slicer, drastically simplifying the process of analyzing subsets of the data based on text patterns.



Team	Points	Assists	contains_ets
Mavs	22	4	No
Rockets	14	5	Yes
Spurs	19	5	No
Spurs	15	4	No
Mavs	20	8	No
Rockets	34	7	Yes
Spurs	31	9	No
Hornets	21	6	Yes
Mavs	15	5	No
Mavs	18	4	No
Hornets	10	4	Yes

The calculated column `contains_ets` now provides a boolean representation of the string match, where "Yes" indicates the presence of the specified substring "ets" within the **Team** column, and "No" indicates its absence. This structure is highly beneficial for creating measures that count or aggregate data based on this specific classification.

Adapting the Output: Returning Numeric Values (1 or 0)

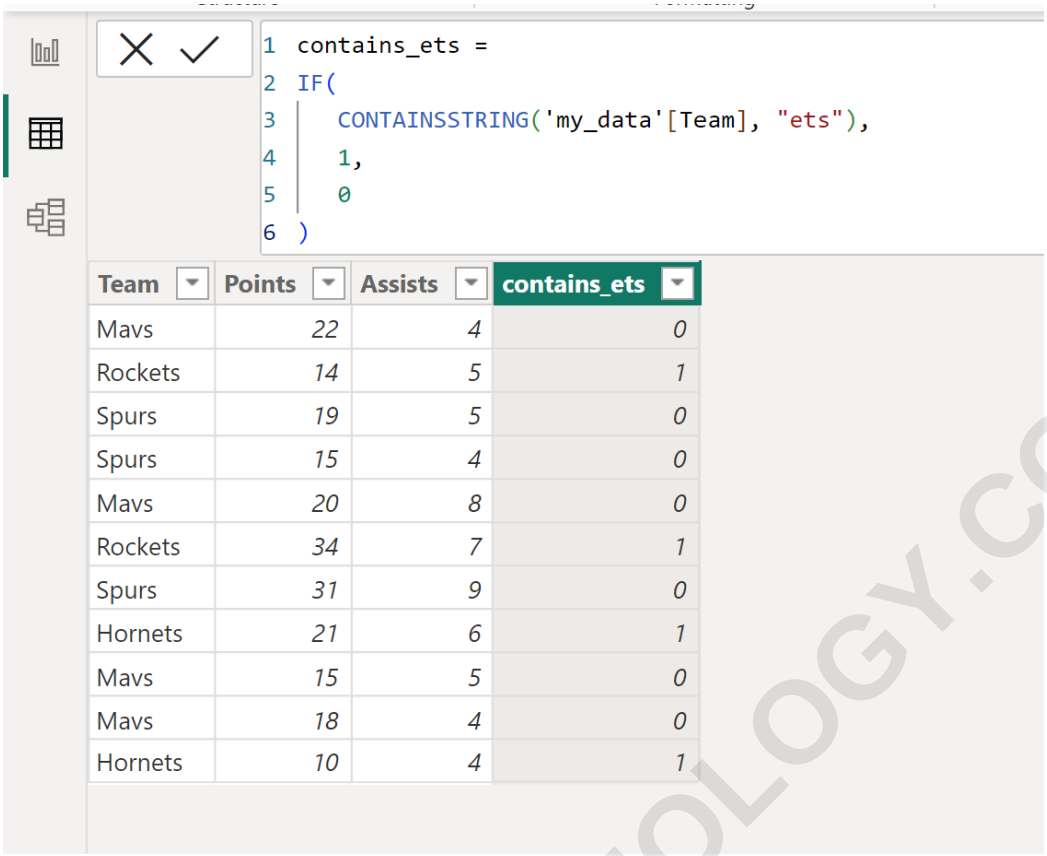
While returning "Yes" or "No" offers excellent clarity for textual reporting, data analysts often prefer numeric outputs (such as 1 or 0) for use in subsequent aggregation or mathematical calculations. Using numeric flags simplifies creating measures like counts, sums, or averages related to the condition, as they can be treated directly as numerical data types rather than requiring additional conversion functions.

The modification required to return 1 or 0 instead of "Yes" or "No" is minimal, residing solely in the output parameters of the outer `IF` function. The core logical test performed by CONTAINSSTRING remains unchanged, but the resulting values passed back to the column shift from strings to integers.

To implement this alternative approach, simply replace the string outputs ("Yes", "No") with the desired numeric outputs (1, 0) in the DAX formula:

```
contains_ets =
IF(
CONTAINSSTRING('my_data', "ets"),
1,
0
)
```

This modified formula provides an identical logical evaluation but yields a cleaner, numerical column suitable for immediate use in mathematical operations. A value of **1** represents a true condition (the substring was found), and **0** represents a false condition (the substring was not found). The resulting table structure, post-calculation, clearly shows these numerical flags:



```
contains_ets =
IF(
    CONTAINSSTRING('my_data'[Team], "ets"),
    1,
    0
)
```

Team	Points	Assists	contains_ets
Mavs	22	4	0
Rockets	14	5	1
Spurs	19	5	0
Spurs	15	4	0
Mavs	20	8	0
Rockets	34	7	1
Spurs	31	9	0
Hornets	21	6	1
Mavs	15	5	0
Mavs	18	4	0
Hornets	10	4	1

Using 1s and 0s is often preferred in complex data models because it aligns with standard boolean mathematics and simplifies interactions with other DAX functions that rely on numerical inputs for filtering and context transitions. For instance, creating a measure to count the number of teams containing "ets" simply requires summing this new column.

Deep Dive into CONTAINSSTRING and Case Sensitivity

The primary workhorse behind the "If Contains" formula is the CONTAINSSTRING function. It is essential for advanced users to understand the characteristics and nuances of this function to avoid common pitfalls in string analysis. The official Microsoft documentation clarifies that CONTAINSSTRING is designed specifically for searching text columns and is inherently case-insensitive, meaning it ignores the distinction between upper and lower letters during the search process.

The syntax for CONTAINSSTRING is straightforward: `CONTAINSSTRING(<Text>, <Find_Text>)`. The first argument, `<Text>`, is the column (or string expression) being searched, and the second argument, `<Find_Text>`, is the specific substring you are looking for. Because it is case-insensitive, if you search for "apple" within a column containing "Apple Pie," it returns TRUE. This flexibility is often beneficial, but if you require a strict, case-sensitive match, Power BI offers

alternative functions that must be used instead.

For scenarios demanding case sensitivity, DAX provides functions like `FIND` or `SEARCH`, often coupled with `ISERROR`, to achieve the desired result. The `FIND` function is always case-sensitive, making it the preferred choice when precise character matching is mandatory, such as searching for specific product codes or serialized identifiers. Using CONTAINSSTRING should be reserved for general pattern matching where variations in capitalization are expected and acceptable.

Advanced String Matching: CONTAINSSTRING vs. FIND and SEARCH

While CONTAINSSTRING simplifies basic "If Contains" logic due to its case-insensitivity and direct boolean output, analysts often encounter situations requiring more nuanced string handling. Understanding the differences between the core DAX string searching functions is crucial for optimal formula design and performance in Power BI.

The key differences are summarized below:

CONTAINSSTRING: Returns TRUE/FALSE. Case-insensitive. Best for general pattern matching.
SEARCH: Returns the starting position (number) of the substring, or an error if not found. Case-insensitive. Useful if you need to know where the substring starts or if you want to search a dynamic expression. Requires `ISERROR` or `ISBLANK` to convert the result into a boolean test within an `IF` statement.
FIND: Returns the starting position (number) of the substring, or an error if not found. **Strictly Case-Sensitive.** Essential for matching specific codes or identifiers. Like `SEARCH`, it requires wrapping with error handling functions like `IF(NOT(ISERROR(FIND(...))), "Yes", "No")` to achieve the "If Contains" logic.

For most simple "If Contains" scenarios where speed and ease of writing are paramount, CONTAINSSTRING is the superior choice. However, if the business requirement explicitly dictates a case-sensitive match, the `FIND` function, despite the more complex wrapping required for error handling, is the only authoritative tool for the job. Choosing the correct function based on sensitivity requirements ensures accurate data classification and avoids errors in downstream reporting.

Practical Use Cases and Applications

The "If Contains" formula is not merely a technical exercise; it serves as a foundation for numerous practical applications across various industries when analyzing a dataset. Leveraging this conditional check allows data analysts to automate classifications that would otherwise require manual effort or complex data preparation outside of Power BI.

Key applications include:

Categorization and Segmentation: Automatically tagging products, customers, or transactions based on keywords present in their descriptive fields. For example, flagging all customer feedback entries that contain keywords like "broken," "slow," or "unhappy" to segment data for immediate investigation by the quality assurance team.
Data Cleansing and Auditing: Identifying malformed data entries or records that contain specific flags (e.g., "TEST," "DRAFT," or placeholder text like "N/A") within a log file or text field, allowing for targeted exclusion or correction during ETL processes.
URL and Source Analysis: In web analytics datasets, using "If Contains" to check URLs or referral strings for specific campaign parameters (e.g., "utm_source=google") to quickly categorize traffic sources without requiring complex URL parsing tools.
Conditional Formatting Flags: Creating measures that rely on the 1/0 output of the "If Contains" column to drive visual alerts or conditional formatting rules in reports, highlighting records that meet a specific textual criterion.

By transforming messy, unstructured text data into clear, binary flags (Yes/No or 1/0), the "If Contains" logic significantly elevates the analytical capability of any Power BI report, allowing for faster insight generation and more targeted data exploration.

Conclusion: Mastering Conditional Text Analysis in DAX

The simple combination of the `IF` and CONTAINSSTRING functions provides an indispensable tool for performing conditional text analysis within Power BI. This formula addresses the fundamental requirement of determining whether a specific pattern or substring exists within a larger text field, resulting in a flexible, categorical column that dramatically enhances data filtering and categorization capabilities.

Whether you choose to return "Yes"/"No" for reporting clarity or 1/0 for measure aggregation, the structured approach detailed here ensures that your DAX code is clean, efficient, and easily maintained. Mastering these basic string functions opens the door to much more complex analytical tasks, allowing you to transform raw data into actionable intelligence within your models.

To further your skills in data manipulation using Power BI and DAX, consider exploring how to combine these string functions with other conditional logic, such as nesting multiple `IF` statements or utilizing functions like `SWITCH` for multi-criteria analysis. The documentation for the core string functions is always the best resource for detailed syntax and performance considerations.

Note: You can find the complete documentation for the **CONTAINSSTRING** function in DAX.

Related Power BI Tutorials and Resources

For users looking to expand their knowledge of common data manipulation tasks in Power BI, the

following tutorials provide valuable insights into related analytical challenges:

Exploring techniques for applying conditional logic across different columns.
Understanding how to use the `SEARCH` and `FIND` functions for case-sensitive requirements.
Best practices for optimizing calculated columns versus calculated measures in large datasets.

By continuously building proficiency in these core DAX patterns, you can unlock the full potential of your Power BI environment.