Learning DAX: How to Implement an “If Contains” Logic in Power BI

Authored by

Mohammed looti

November 12, 2025

The Essential Role of Conditional Logic in Data Modeling

In the landscape of modern business intelligence, the efficacy of data analysis frequently depends on the capacity to manipulate and precisely categorize textual information. When analysts interact with descriptive columns--such as detailed product descriptions, raw customer feedback, or organizational affiliations--a recurring requirement is the need to ascertain whether a specific textual fragment, often referred to as a string, is present within a larger field of text. This fundamental capability is paramount for the creation of robust flags, dynamic filters, and highly customized calculations that drive actionable insights.

For individuals leveraging Power BI for their analytical needs, the foundational language for implementing this conditional logic is DAX (Data Analysis Expressions). DAX offers a comprehensive suite of functions meticulously engineered to manage complex data transformation, stringent filtering, and accurate aggregation tasks. Among these essential operations, one of the most common requirements in text manipulation is checking for the existence of a substring--an operation functionally equivalent to the ubiquitous "If Contains" statement found across numerous programming and scripting environments.

By skillfully employing specialized DAX functions, we gain the ability to effortlessly generate new calculated columns or specialized measures that automatically classify data based on predefined textual patterns. This powerful feature not only contributes significantly to cleaning and structuring raw datasets but also radically simplifies the subsequent processes of reporting and sophisticated data visualization. Acquiring a mastery of the correct syntax and application for these text-based operations is absolutely critical for any professional seeking to harness the full potential and analytical power of their data model.

Introducing CONTAINSSTRING: The Core DAX Solution

The standard, and arguably most efficient, methodology for performing a case-insensitive "contains" verification in DAX involves the utilization of the `CONTAINSSTRING` function. This function is specifically designed to return a boolean value of `TRUE` if the specified target substring is located anywhere within the larger source text, and `FALSE` if it is not found. While `CONTAINSSTRING` provides this necessary boolean result directly, in practical application, data modelers invariably embed it within an `IF` function. This integration allows the formula to return user-friendly, custom outputs such as the text labels "Yes" or "No," or more functionally useful numeric flags like 1 or 0.

The exemplary syntax for constructing a new calculated column that checks for substring presence and returns a categorical result is illustrated below. This common pattern is exceptionally valuable for segmenting and labeling data based on their inherent textual characteristics:

contains_ets =

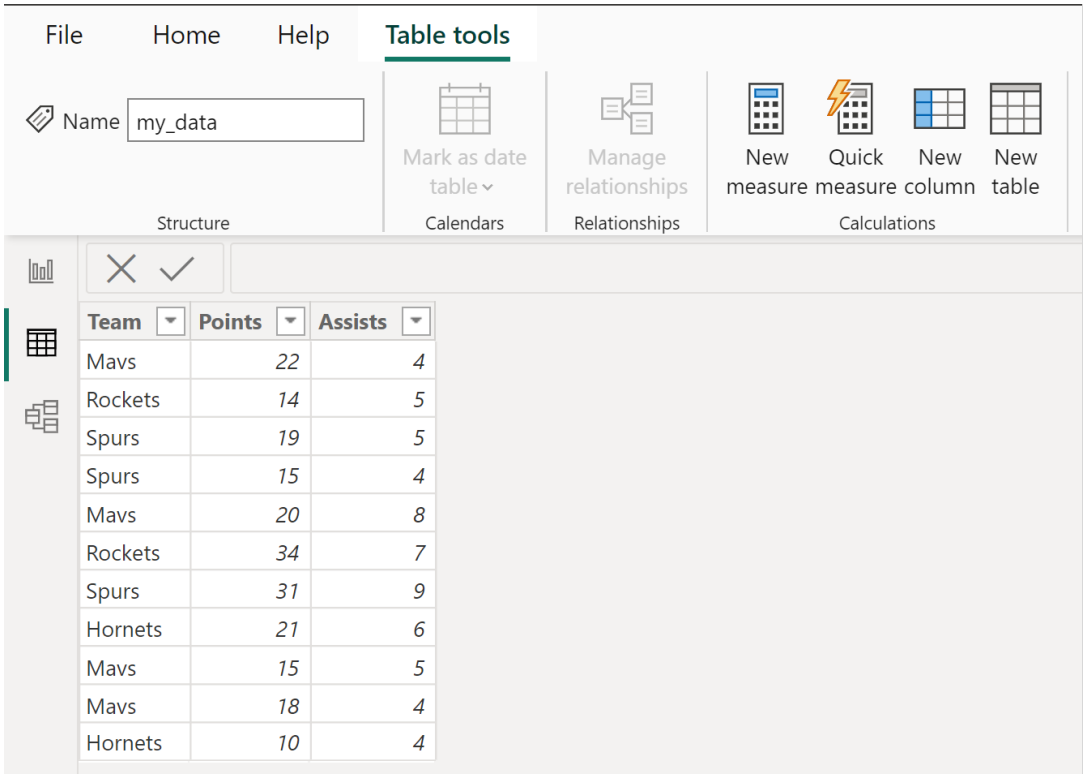
```
IF(
CONTAINSSTRING('my_data', "ets"),
"Yes",
"No"
)
```

This formula snippet creates a new calculated column, explicitly named `contains_ets`. Its operation involves examining the data housed within the column of the `'my_data'` table. If the sequence of characters "ets" is identified anywhere within the team name--crucially, without regard to capitalization (e.g., "Nets," "ETS," or "nets" all match)--the new column renders the result "Yes." Conversely, if the specified substring is absent, the column returns "No." This method is highly favored because the CONTAINSSTRING function is inherently optimized for rapid and straightforward text comparisons within the DAX engine.

Step-by-Step Implementation in Power BI Desktop

To fully appreciate the practical utility of the `CONTAINSSTRING` function, let us examine a typical business scenario involving a tabular dataset of sports teams. Imagine we are tasked with managing a table in Power BI, designated as **my_data**, which compiles various details concerning players and their corresponding team affiliations. Our immediate requirement is to quickly flag every team name that incorporates a specific three-letter sequence, such as "ets," regardless of how it is capitalized.

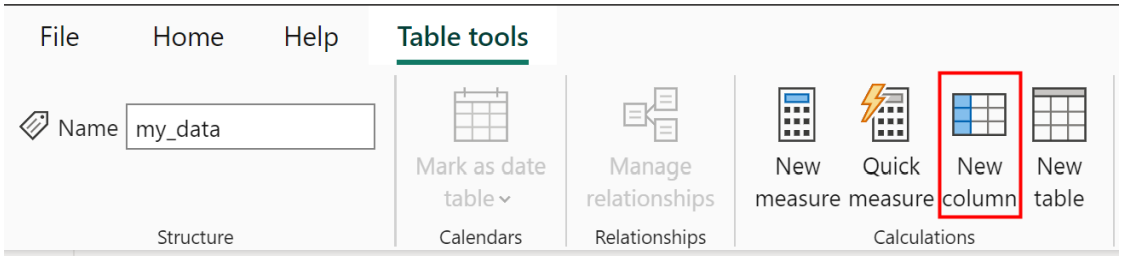
The initial state of the dataset, which forms the starting point for our detailed analysis, is structured as shown below. It is important to note the variation in capitalization and team descriptors, which strongly validates the need for a robust, case-insensitive search function provided by DAX:



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

Our goal is clear: we must append a new column to this existing table that clearly indicates whether the corresponding text string located in the **Team** column contains the sequence "ets." This newly generated column will then serve as an immediate mechanism for advanced filtering and streamlined conditional analysis in reports.

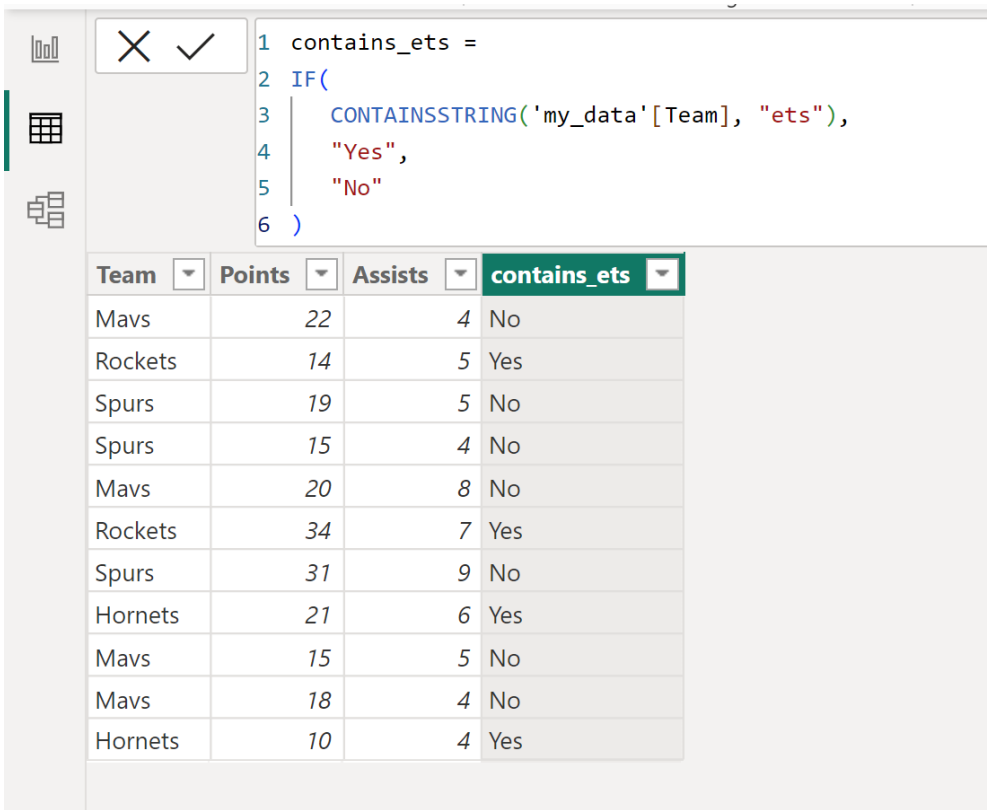
To initiate this calculation, the first action in Power BI Desktop is to create a new calculated column. This is typically executed by navigating to the "Table tools" or "Modeling" tab and selecting the **New column** icon:



Once the framework for the new column is established, the DAX formula bar becomes active. We then meticulously input the previously defined formula, ensuring absolute accuracy in referencing the table name (`'my_data'`) and the appropriate column name (). The required formula to be entered is:

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

Upon successful execution, this calculation immediately populates the new column, `contains_ets`, with the designated categorical results based on the presence of the target substring within the **Team** column. The resultant structured table clearly segments the data, making the identification of relevant records intuitive and rapid. This structured approach successfully yields the following output, showcasing the precise application of the conditional text search across the entire dataset:



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 resulting **contains_ets** column functions as an indispensable flag, definitively returning either "Yes" or "No" to indicate if the corresponding team name contains the specified substring "ets." This simple yet powerful column can now be seamlessly integrated as a primary filter within reports, or utilized as an efficient grouping mechanism for subsequent aggregate calculations.

Optimizing Output: Why Numeric Flags (1s and 0s) are Superior

While the output of "Yes" or "No" offers undeniable human readability, this approach often introduces functional limitations when the ultimate objective is to perform mathematical aggregations, such as counting, or complex filtering within Power BI reports. For instance, calculating the total count of teams containing the sequence "ets" necessitates additional steps when the column result is stored as a text data type.

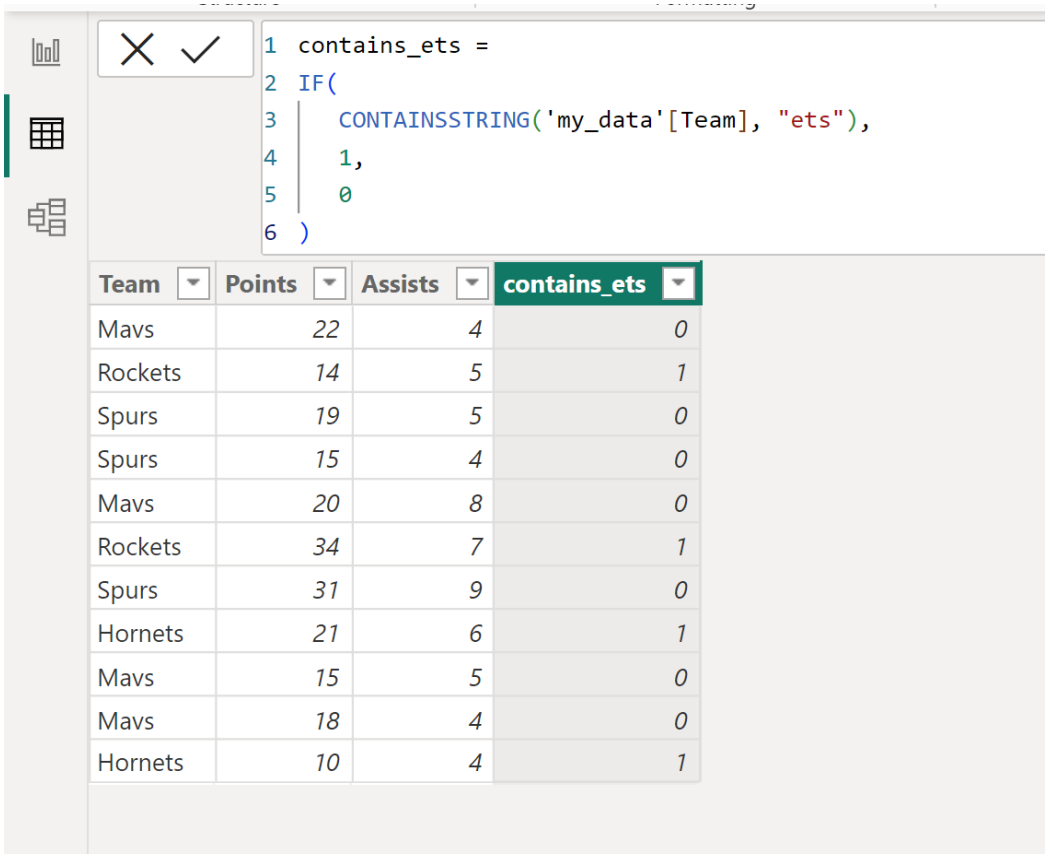
A highly recommended best practice standard in professional data modeling is the use of binary numeric values (1 or 0) for categorical flags. By assigning 1 to represent "True" (Yes) and 0 to represent "False" (No), we guarantee that the resulting column is correctly classified as a numeric data type. This distinction allows for immediate and automatic summation, averaging, and count operations without the need for supplementary measures or calculations designed solely to convert text fields into usable numerical values.

To implement this superior alternative, the only required modification is an adjustment to the output parameters of the `IF` statement. Instead of returning literal text strings ("Yes" and "No"), we instruct the function to return the integers 1 and 0:

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

This revised formula successfully preserves the rigorous conditional check provided by the CONTAINSSTRING function while simultaneously ensuring the output is perfectly optimized for numerical analysis and computation. If this formula were applied to our sample dataset, the resulting column would display 1s and 0s. For example, creating a simple measure to sum this column would instantaneously yield the total count of rows that satisfy the specified textual condition.

The visual outcome of employing numeric flags maintains the same clear distinction within the table structure, but with the critical added advantage of being primed for immediate aggregation and computation across the entire DAX model, drastically improving performance for summary statistics:



```
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

Ultimately, the decision between text output and numeric output should be driven entirely by the downstream reporting and analytical requirements. For straightforward visual clarity in a simple table or matrix, text labels are acceptable; however, for the development of complex measures, robust filtering, and high-performance aggregations, numeric flags (1s and 0s) are overwhelmingly the superior choice.

Beyond Simple Containment: Advanced DAX Text Analysis

While `CONTAINSSTRING` serves as the primary and most accessible tool for simple, case-insensitive substring checks, proficient data modelers must also be familiar with related DAX functions that cater to more complex and nuanced text analysis requirements. The fundamental difference among these functions often revolves around case sensitivity and the necessary handling of wildcards or specific positional information.

For specialized scenarios that necessitate strict adherence to case sensitivity, the function `CONTAINSSTRINGEXACT` must be employed. In contrast to the non-exact version, `CONTAINSSTRINGEXACT` will only return `TRUE` if the exact case structure of the substring matches the case in the source string precisely. Furthermore, in situations where the analyst requires knowledge of the exact starting position of the substring, or when sophisticated wildcard pattern

matching (similar to the LIKE operator in SQL) is necessary, the `SEARCH` and `FIND` functions become the relevant tools, though their integration within the standard `IF` statement structure is generally more complex.

When implementing any conditional logic using DAX, especially when dealing with very large fact tables, it is crucial to use calculated columns judiciously. Although the preceding examples utilize calculated columns for simplicity, if the resulting flag is only required for temporary filtering or within the context of a specific visualization, model performance may be significantly optimized by using a calculated measure instead. Calculated measures execute their computation dynamically only when they are referenced in the visualization layer, thereby minimizing the memory footprint compared to calculated columns, which consume memory for every single row in the table, irrespective of whether they are used in the report.

Finally, always consult the official documentation for the specific function being utilized. The operational behavior of text functions, particularly in relation to international character sets, varying locales, and null values, can sometimes lead to unexpected results if the function parameters and limitations are not fully comprehended. For instance, reviewing the complete documentation for the CONTAINSSTRING function in DAX is an essential practice for addressing advanced or edge-case scenarios.

Conclusion and Best Practice Summary

The capacity to accurately and efficiently determine if a text field contains a specific substring is an indispensable skill within Power BI data modeling. By effectively utilizing the `CONTAINSSTRING` function, meticulously integrated within an `IF` statement, data analysts can rapidly categorize, segment, and flag data based on complex textual patterns.

Key strategic takeaways from this comprehensive tutorial include:

The `CONTAINSSTRING` function provides a fast, case-insensitive boolean check for the existence of a substring.
Integrating this function with `IF` allows for the generation of customized output, such as descriptive "Yes"/"No" text or functional 1/0 numeric flags.
Employing numeric flags (1 and 0) is generally considered a best practice for optimal aggregation and streamlined measure creation within DAX models.
For situations demanding exact matches based on capitalization, modelers must switch to the `CONTAINSSTRINGEXACT` function.

To further advance your expertise in text manipulation and sophisticated conditional logic within the DAX environment, we strongly recommend exploring additional tutorials that cover other essential functions and common data transformation tasks in Power BI.

Additional Resources

The following resources detail how to perform other common tasks in Power BI and DAX, building directly upon the foundational knowledge of conditional string checking:

How to use wildcards in DAX for advanced pattern matching.
Effective techniques for handling blank values and errors in critical text columns.
Strategies for advanced filtering based on the results of calculated column outputs.