Learning Data Binning Techniques in Power BI with DAX: A Step-by-Step Guide

Authored by

Mohammed looti

November 12, 2025

One of the most powerful techniques in Power BI for enhancing the utility of continuous data is data binning (also known as data discretization or data segmentation). This process involves grouping similar numerical values into discrete, manageable ranges or categories. By transforming raw numerical scores into predefined bins, analysts can simplify complex distributions, improve visualization clarity, and facilitate easier comparative data analysis. In this tutorial, we will explore how to efficiently perform data binning directly within your data model using Data Analysis Expressions (DAX).

While Power BI offers built-in tools to create static bins for visualizations, using a calculated column powered by **DAX** provides superior flexibility, allowing the bins to be used across multiple visuals, filters, and subsequent calculations. This method is crucial when the bin definition requires complex, non-uniform logic or specific categorical labels. The structure we employ relies heavily on **nested IF functions**, which enable us to evaluate a series of sequential conditions to assign the appropriate bin category for each row of data.

The following syntax demonstrates the standard use of **DAX** to create a new calculated column that segments the values found in the `Points` column of the `my_data` table. This formula establishes four distinct, mutually exclusive bins based on specified numerical thresholds, providing a foundational approach to data segmentation within your Power BI model.

```
Bin =
IF(
'my_data' < 16,
"<16",
IF(
'my_data' <= 20,
"16-20",
IF( 'my_data' <= 25, "21-25", ">25")
)
)
```

Understanding DAX and the Nested IF Logic

The formula presented above creates a new calculated column named **Bin** within your data table. This column is populated row-by-row, evaluating the `Points` value for each player against a sequence of conditional tests. It is essential to understand that DAX processes these **nested IF functions** hierarchically. Once a condition is met, the evaluation stops, and the corresponding result is returned, meaning the order of conditions is critical for accurate data binning.

In this specific implementation, we are using three nested **IF** statements to define four discrete

categories. The resulting values returned by the **Bin** column are descriptive strings representing the numerical range for the score. Understanding how each condition interacts ensures data integrity and meaningful results in your subsequent data analysis visualizations and reports.

The logic proceeds sequentially, ensuring that every data point falls into one and only one bin. The results generated by the calculated column are defined as follows:

"<16": This bin is assigned if the value in the **Points** column is strictly less than 16. This is the first test evaluated.

Else, "16-20": If the score is not less than 16 (i.e., it is 16 or greater), the second **IF** function checks if the score is less than or equal to 20. This effectively captures the range .

Else, "21-25": If the score is greater than 20, the third **IF** function checks if the score is less than or equal to 25. This captures the range .

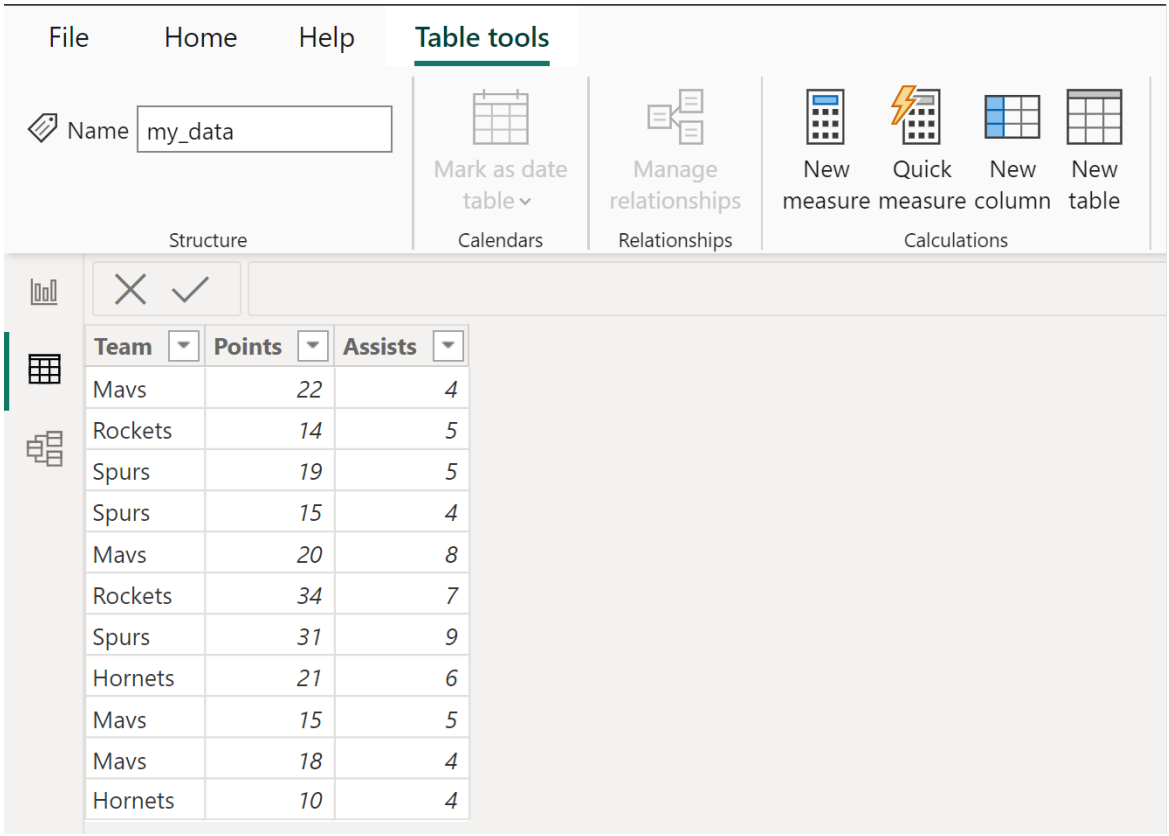
Else, ">25": If none of the preceding conditions are met, the final default value is returned, capturing scores greater than 25.

The following example demonstrates how to use this powerful formula in practice, starting from raw data input through to the final binned output.

Practical Implementation: Setting Up the Data Model

To illustrate the power of this technique, let us apply this **DAX** formula to a realistic scenario. Suppose we are working with a table in Power BI named **my_data**. This table contains performance metrics for various basketball players, specifically focusing on the number of points each player scored during a game. Our objective is to classify these players into performance categories based on their `Points` score, which requires robust data segmentation.

The source data, which must first be loaded into your Power BI environment, appears as follows. Notice the continuous nature of the `Points` column, which we aim to transform into discrete bins for easier statistical interpretation and visualization.



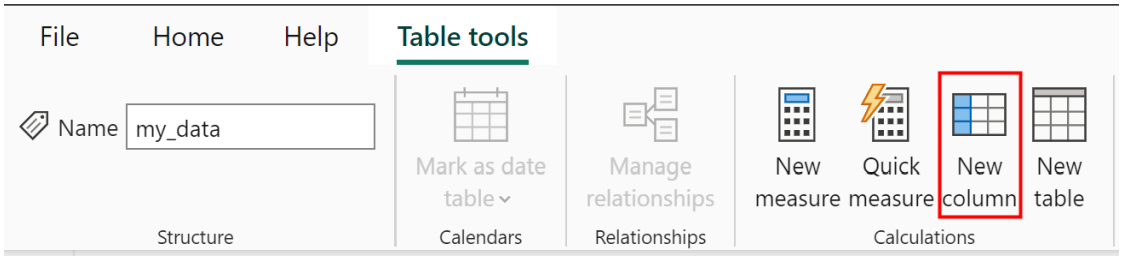
Team	Points	Assists
Mavs	22	4
Rockets	14	5
Spurs	19	5
Spurs	15	4
Mavs	20	8
Rockets	34	7
Spurs	31	9
Hornets	21	6
Mavs	15	5
Mavs	18	4
Hornets	10	4

The core requirement is to group these players into meaningful segments--for instance, determining which scores qualify as 'low,' 'medium,' or 'high' performance. This classification step is essential for creating aggregate reports or conditional formatting based on these new categories. The use of a calculated column ensures that this classification logic is inherent to the data model itself, rather than being confined to a single visual element.

Step-by-Step Guide to Creating Numerical Bins

To begin the data binning process, navigate to the **Table tools** ribbon in Power BI Desktop while viewing the data table. This is where we initiate the creation of a new, calculated column that will house our **DAX** logic. This calculated column is different from a measure because it performs a row-context calculation, meaning it evaluates the formula for every row in the table independently.

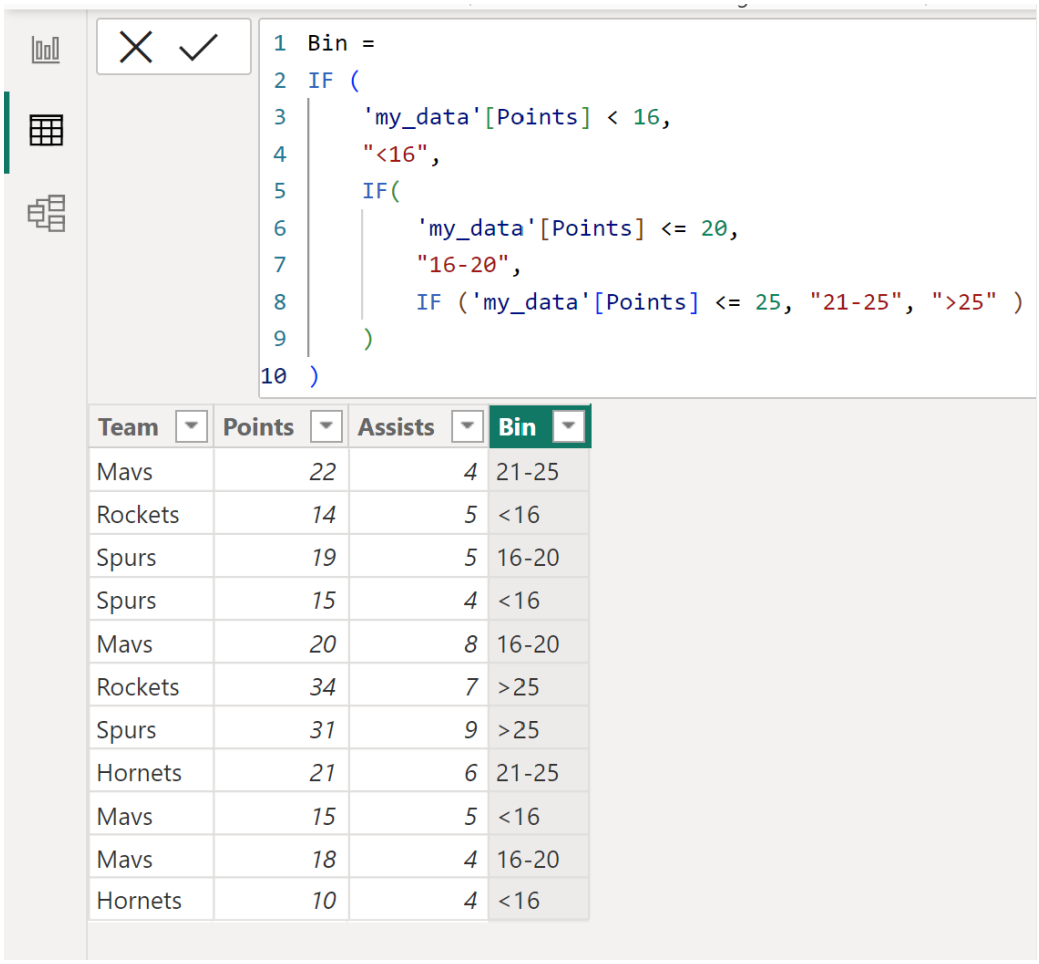
The first procedural step involves clicking the **New column** icon, which opens the formula bar ready for input. This action immediately prompts Power BI to expect a valid DAX expression that will define the values of the new column across all existing rows.



Then, carefully type in the formula into the formula bar. Accuracy in syntax, including correct column referencing (`'my_data'`) and quotation marks for string outputs, is vital for the **DAX** engine to execute the logic successfully. This complex formula defines the specific numerical boundaries for our bins:

```
Bin =
IF(
'my_data' < 16,
"<16",
IF(
'my_data' <= 20,
"16-20",
IF( 'my_data' <= 25, "21-25", ">25")
)
)
```

This will create a new column named **Bin** that places each of the players into one of four different bins based on their number of points scored. The resulting table clearly shows the segmentation of the continuous data into discrete categories, ready for visualization and reporting.



```
Bin =
IF (
    'my_data'[Points] < 16,
    "<16",
    IF(
        'my_data'[Points] <= 20,
        "16-20",
        IF ('my_data'[Points] <= 25, "21-25", ">25" )
    )
)
```

Team	Points	Assists	Bin
Mavs	22	4	21-25
Rockets	14	5	<16
Spurs	19	5	16-20
Spurs	15	4	<16
Mavs	20	8	16-20
Rockets	34	7	>25
Spurs	31	9	>25
Hornets	21	6	21-25
Mavs	15	5	<16
Mavs	18	4	16-20
Hornets	10	4	<16

By reviewing the generated output, we can confirm the successful application of the binning logic:

The first player who scored **22** points is correctly placed into Bin **21-25**.

The second player who scored **14** points is assigned to Bin **<16**.

The third player who scored **19** points is placed into Bin **16-20**.

Enhancing Readability with Categorical Bins

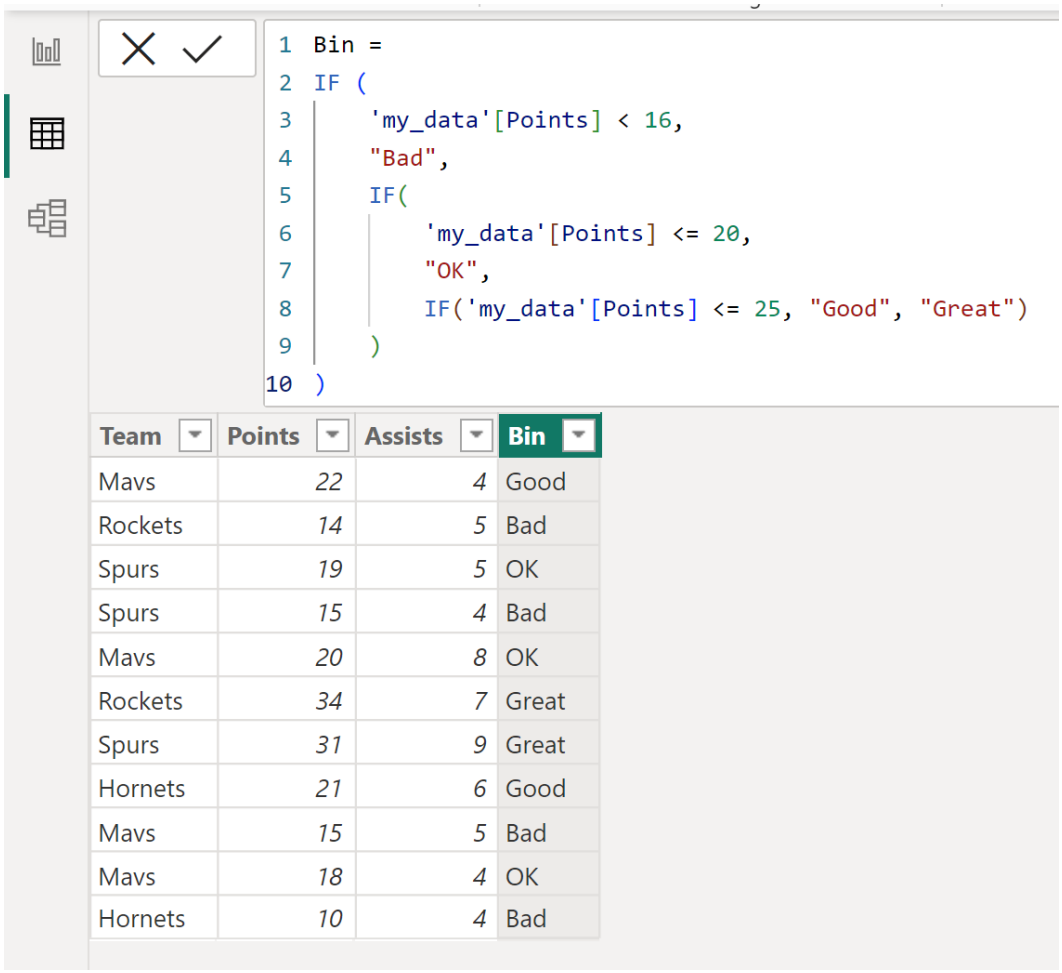
While numerical range labels are precise, for reporting purposes, it is often beneficial to replace these segments with descriptive, categorical strings that are immediately intuitive to the end-user. This improves the narrative quality of the data and makes visualizations easier to interpret, particularly for non-technical stakeholders. We can easily modify our existing DAX formula to return these categorical labels instead of numerical ranges.

For example, we can use the following revised formula to place each individual player into the bins of "Bad", "OK", "Good" or "Great." The underlying logic (the numerical thresholds) remains

identical; only the descriptive string output is altered to enhance readability for data analysis consumption.

```
Bin =
IF(
'my_data' < 16,
"Bad",
IF(
'my_data' <= 20,
"OK",
IF( 'my_data' <= 25, "Good", "Great")
)
)
```

This revised **DAX** expression will create a new column named **Bin** that places each of the players into one of the four performance bins based on their number of points. This output is generally preferred in executive summaries and dashboards where quick, qualitative assessments of performance are more valuable than precise numerical segmentation.



```
Bin =
IF (
    'my_data'[Points] < 16,
    "Bad",
    IF(
        'my_data'[Points] <= 20,
        "OK",
        IF('my_data'[Points] <= 25, "Good", "Great")
    )
)
```

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

Conclusion and Best Practices for Data Segmentation

The utilization of **nested IF functions** within **DAX** provides a highly versatile and powerful method for implementing custom data binning in Power BI. While the **IF** function is intuitive, be mindful of its limitations. For scenarios requiring dozens of bins, the nesting complexity can become unwieldy and difficult to debug. In such advanced cases, consider using the `SWITCH(TRUE(), ...)` function, which offers a cleaner, more scalable syntax for evaluating multiple sequential conditions against a single column.

Note: In these examples, we chose to use three nested **IF** functions to define four distinct bins. However, you can use even more nested **IF** functions if the analytical requirement is to place players into a greater number of bins. Always prioritize clear, logical ordering of conditions, starting with the most restrictive or lowest range, to prevent misclassification due to the sequential evaluation of the **DAX** engine. Proper binning significantly enhances data visualization and statistical reporting efficiency.

Additional Resources

To further your expertise in data manipulation and modeling within Power BI, we recommend exploring tutorials focused on alternative data transformation methods. Mastering these techniques will empower you to handle diverse and complex datasets effectively.

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