How to Rank Values by Group in Power BI: A Step-by-Step Guide

Authored by

mohammed looti

January 11, 2026

Power BI is recognized globally as a robust platform for business intelligence and DAX is the expression language that unlocks its true potential. Effective data analysis often requires more than simple aggregation; it demands contextual comparison. One of the most frequently requested analytical operations is ranking values, not across the entire dataset, but specifically within defined subsets or groups. While basic sorting features in tables and charts provide superficial ranking, achieving a persistent, calculated rank that can be used in measures and visuals requires the deployment of a custom column using the powerful capabilities of DAX. This method allows analysts to easily compare and identify the highest or lowest values within each category, yielding crucial insights for operational and strategic decision-making.

Understanding the Need for Grouped Ranking in Data Analysis

Grouped ranking, often referred to as intra-group ranking, is a fundamental analytical technique. Unlike global ranking, which assigns a rank based on the entire column of values regardless of other attributes, grouped ranking isolates specific contexts. For example, if you are analyzing sales data, you might want to know the top-selling product within each specific region, rather than globally. This technique transforms raw data into immediately actionable intelligence by clearly defining performance boundaries. Without this specialized calculation, identifying the relative performance of an item within its peer group--such as a specific team, department, or product category--becomes cumbersome and prone to error when relying solely on visual sorting.

In the context of Power BI, leveraging calculated columns for ranking is essential because it embeds the result directly into the data modeling structure. This permanence ensures that the rank is stable and available for use across various reports, filters, and subsequent calculations. Furthermore, a calculated column allows the ranking logic to be dynamic yet fixed per row context, providing unparalleled flexibility compared to relying on visual-level filters or static aggregations. This approach supports complex scenarios where hierarchical ranking is required, allowing for deep dives into performance metrics across different dimensions simultaneously.

The core challenge in implementing grouped ranking lies in instructing DAX to evaluate the ranking function only against a filtered subset of data--specifically, the rows belonging to the current group defined by an attribute like 'Team' or 'Category'. This requirement necessitates the coordinated use of variable declaration and context transition functions, which we will explore in detail. Mastering this specific calculation opens the door to advanced analytical capabilities, allowing analysts to move beyond simple descriptive statistics toward prescriptive and diagnostic insights derived from relative performance metrics.

Introducing the DAX Framework for Contextual Calculation

To achieve contextual ranking within Power BI, we must utilize the power of Data Analysis

Expressions (DAX). The specific framework involves defining a variable that captures the current group's context and then using that variable within an iterative ranking function. This strategy ensures that for every row being evaluated, the ranking operation is scoped only to the rows sharing the same grouping identifier. The key components of this solution are the RANKX function, which performs the iteration and ranking, and the FILTER function, which dynamically controls the row context for the ranking process.

The use of VAR and RETURN statements is crucial for performance and readability. By defining a variable (e.g., current_team), we store the grouping value of the current row before the heavy lifting of the ranking calculation begins. This variable acts as a constant filter reference within the subsequent ranking iteration. Without this structured approach, the formula becomes less efficient and harder to debug. The RANKX function then iterates over the filtered table (defined by the FILTER function), evaluating the specified expression for each row and assigning a rank based on the results.

The general syntax pattern for creating a new column that displays the rank of values in one column, grouped by another column, looks like this. This formula ensures that the calculation respects the group boundaries defined by the specified column, offering highly accurate and segment-specific rankings required for high-level analytical reports:

You can use the following syntax in DAX to create a new column that displays the rank of values in one column, grouped by another column:

```
Points Rank =
VAR current_team = 'my_data'
RETURN
RANKX(
FILTER(
'my_data',
'my_data' = current_team
),
'my_data',
,
,
SKIP
)
```

This particular example creates a new column named **Points Rank** that assigns a ranking to each value in the **Points** column of the table, grouped specifically by the values in the **Team** column.

Dissecting the Core DAX Formula for Grouped Ranking

Understanding the components of the DAX formula is essential for successful implementation and modification. The formula starts by defining the VARiable, current_team, which captures the value of the column for the specific row context being processed. This value is critical because it ensures that the subsequent ranking calculation only interacts with other rows sharing that identical team identifier. This is the mechanism that achieves the grouping effect--it locks the context before evaluation begins.

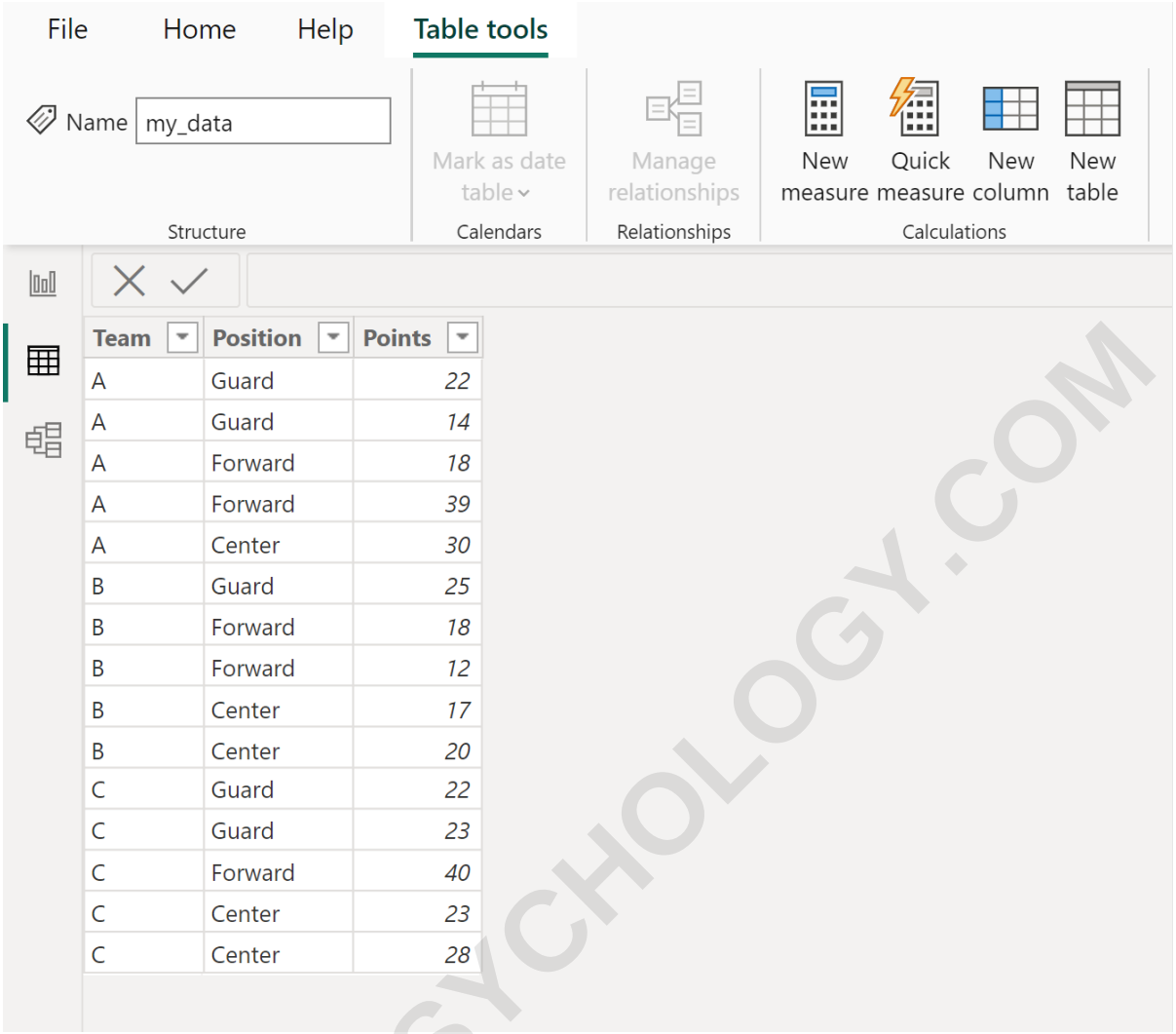
The heart of the calculation lies within the RANKX function. RANKX requires three main arguments: the table to iterate over, the expression to rank, and the value to rank against (which is optional here). The table provided to RANKX is not the entire 'my_data' table, but rather the result of the FILTER function. The FILTER function dynamically creates a virtual table containing only the rows where the 'my_data' equals the stored current_team variable. This context transition is what restricts the ranking scope to the current group.

The second argument within RANKX specifies the expression used for ranking, which is simply 'my_data'. This tells the function to use the numerical value in the Points column as the basis for ordering. The remaining arguments control the rank order and tie-handling. By default, omitting the order parameter (the fourth argument) results in descending rank (highest points get rank 1). The fifth and final argument, SKIP, defines how ties are handled. SKIP ensures that if two players have the same points, they receive the same rank, and the next rank skips a number (e.g., ranks 1, 2, 2, 4). This detailed configuration provides precision necessary for accurate sports or performance metric analysis.

Step-by-Step Implementation in Power BI Desktop

To demonstrate this powerful DAX calculation, we will walk through a practical scenario using sample sports data. Suppose we have a table named **my_data** within Power BI that tracks the points scored by various basketball players, categorized by their respective teams. Our goal is to calculate each player's rank solely based on the points they accumulated within their assigned team, ensuring a fair, intra-team comparison.

Suppose we have the following table in Power BI named **my_data** that contains information about points scored by basketball players on various teams:

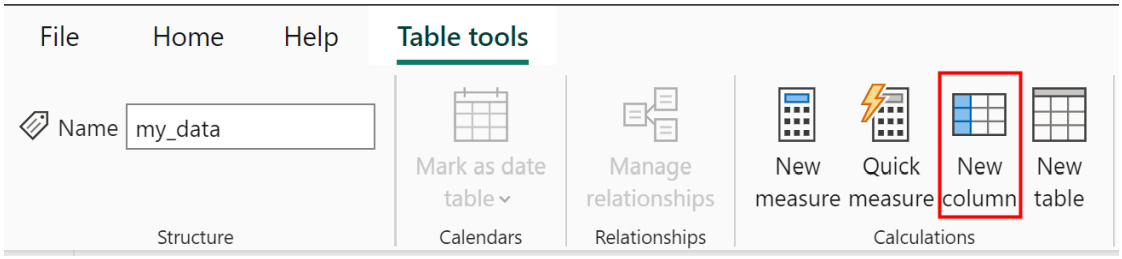


Team	Position	Points
A	Guard	22
A	Guard	14
A	Forward	18
A	Forward	39
A	Center	30
B	Guard	25
B	Forward	18
B	Forward	12
B	Center	17
B	Center	20
C	Guard	22
C	Guard	23
C	Forward	40
C	Center	23
C	Center	28

Suppose we would like to add a new column that displays the rank of the values in the **Points** column, grouped by the **Team** column. This calculation moves beyond simple visualization and embeds the analytical result directly into the underlying data modeling structure, making it available for subsequent measures and complex visual interactions.

To initiate the process, navigate to the **Table tools** tab located along the top ribbon interface of Power BI Desktop. Within this menu, select the **New column** option. This action opens the formula bar, allowing you to input the necessary DAX expression that defines the ranking logic. Ensure that you have the correct data table selected before clicking 'New column' so that the formula context is correctly applied to **my_data**.

To do so, click the **Table tools** tab along the top ribbon, then click the **New column** icon:



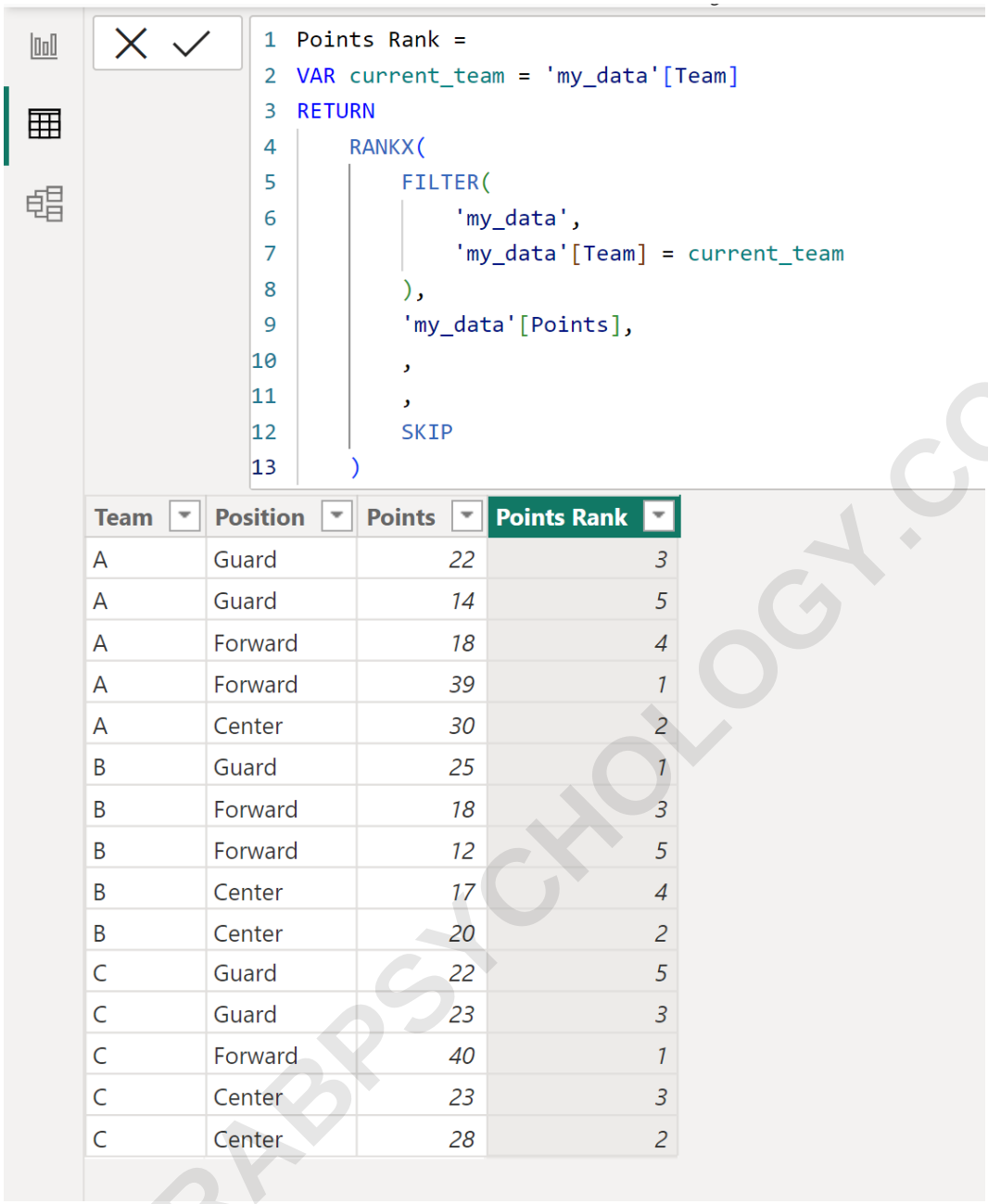
Then type in the following formula into the formula bar:

```
Points Rank =
VAR current_team = 'my_data'
RETURN
RANKX(
FILTER(
'my_data',
'my_data' = current_team
),
'my_data',
,
,
SKIP
)
```

Visualizing the Grouped Rank Results

Once the DAX formula is successfully entered and committed, Power BI processes the calculation row by row, applying the grouping logic defined by the current_team variable and the FILTER function. The result is a new column named **Points Rank** appended to the **my_data** table. This column now holds the relative ranking of each player's points total exclusively within their respective team context, providing an immediate and clear metric for performance comparison.

This will create a new column named **Points Rank** that displays the rank of the values in the **Points** column, grouped by the **Team** column:



```
Points Rank =
VAR current_team = 'my_data'[Team]
RETURN
    RANKX(
        FILTER(
            'my_data',
            'my_data'[Team] = current_team
        ),
        'my_data'[Points],
        ,
        ,
        SKIP
    )
```

Team	Position	Points	Points Rank
A	Guard	22	3
A	Guard	14	5
A	Forward	18	4
A	Forward	39	1
A	Center	30	2
B	Guard	25	1
B	Forward	18	3
B	Forward	12	5
B	Center	17	4
B	Center	20	2
C	Guard	22	5
C	Guard	23	3
C	Forward	40	1
C	Center	23	3
C	Center	28	2

By examining the resulting table, the effectiveness of the grouped ranking methodology becomes clear. Notice that the rank resets for each unique team. This is the intended behavior, demonstrating that the formula successfully applied context transition based on the team identifier. For example, the highest scorer on Team A receives Rank 1, and simultaneously, the highest scorer on Team B also receives Rank 1. This facilitates focused competitive analysis.

For example, we can observe the ranking behavior within Team A:

The player with the most points on team A (39 points) received a rank of **1**.
The player with the second most points on team A (30 points) received a rank of **2**.

The player with the third most points on team A (22 points) received a rank of **3**.

This hierarchical ranking structure is invaluable for reports focusing on internal group performance and identifying group leaders or outliers.

Adjusting Rank Order: Ascending vs. Descending

By default, when the ranking order argument in the RANKX function is omitted, the function assumes a **descending** order (highest value gets rank 1). This is the standard practice for ranking performance metrics like points, sales, or revenue, where a higher number signifies a better result. However, there are scenarios, particularly when dealing with costs, penalties, or time durations, where ranking needs to be performed in **ascending** order (lowest value gets rank 1). DAX provides simple flexibility to manage this requirement.

To switch the ranking logic to ascending order, you must explicitly specify the order parameter within the RANKX function. This is the fourth parameter in the function signature. Instead of leaving it blank or using 0 (for descending), you must input the value 1 to instruct RANKX to perform an ascending sort. Maintaining the structured approach using VAR and RETURN ensures that the context remains correctly scoped to the current team.

The adjusted formula below illustrates how to implement ascending ranking. By changing that single parameter to 1, the rank of **1** is now assigned to the lowest point total within each group, and the ranking increases as the points increase. This versatility allows analysts to tailor the ranking calculation precisely to the nature of the data being evaluated, whether it represents maximized gain or minimized cost.

Note that if you would instead like to rank from high to low, then you can specify the value **1** in the second to last parameter of the RANKX function:

```
Points Rank Ascending =
VAR current_team = 'my_data'
RETURN
RANKX(
FILTER(
'my_data',
'my_data' = current_team
),
'my_data',
,
1,
SKIP
```

)

This will now assign a rank of **1** to the lowest value in each group, a value of **2** to the second lowest value in each group, and so on.

Further Considerations and Advanced DAX Usage

While the calculated column approach detailed above is highly effective for static grouping, analysts should be aware of limitations and alternatives. Calculated columns consume memory because the results are stored row-by-row in the data modeling structure. For extremely large datasets or highly dynamic reports where ranking needs to respond instantly to slicer selections, using a DAX Measure combined with functions like ALLSELECTED might be a more performant alternative, although the complexity increases significantly due to context manipulation.

Another important consideration is tie-handling. We utilized the SKIP parameter in our examples. SKIP handles ties by assigning the same rank to tied values and skipping the next numerical rank. Alternatively, using the DENSE parameter ensures that the rankings are sequential even when ties occur (e.g., ranks 1, 2, 2, 3). Choosing between SKIP and DENSE depends entirely on the analytical requirement and how you wish to penalize or reward tied performers.

Finally, remember that the official documentation for the RANKX function contains exhaustive details regarding all parameters, including handling blanks and other nuanced behaviors. For advanced scenarios involving multiple grouping columns or non-numeric ranking criteria, consulting this documentation is essential for refining your DAX logic.

Note: You can find the complete documentation for the RANKX function in Power BI .

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