Learning to Rank Values by Group in Power BI Using DAX

Authored by

Mohammed looti

November 12, 2025

The Importance of Grouped Ranking in Data Analysis

In advanced data analysis using Power BI, it is frequently necessary to determine the relative performance of an item within a specific subset rather than across the entire dataset. This technique, known as grouped ranking, allows analysts to quickly identify top performers within categorized fields, such as departments, regions, or, in our example, sports teams. Simply ranking all values together would obscure the internal competition and provide less meaningful insights to stakeholders.

To achieve this sophisticated calculation, we leverage DAX (Data Analysis Expressions). Specifically, we must combine the power of the RANKX function with the filtering capabilities of the FILTER function. This combination ensures that the ranking operation respects the boundaries established by the grouping column, effectively creating a "rank within group" calculation. Mastering this pattern is fundamental for anyone looking to move beyond simple aggregations in their Power BI models.

The following DAX syntax provides a robust and efficient method for creating a new calculated column that displays the rank of values in one column (e.g., Points), partitioned by another column (e.g., Team). This pattern uses variable definitions (**VAR** and **RETURN**) to enhance readability and performance by caching the current grouping context.

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

This specific formula generates a new column named **Points Rank**. It systematically assigns a rank to each value found in the **Points** column, but critically, it confines the ranking scope to only the rows that share the same value in the **Team** column. This ensures that the player with the highest points on Team A receives Rank 1 for Team A, and the player with the highest points on Team B receives Rank 1 for Team B, independent of their total points relative to the entire league.

Breaking Down the Grouped Ranking DAX Logic

Understanding how the formula works requires a deep dive into DAX evaluation context, particularly the transition from row context to filter context. When a calculated column is created, Power BI iterates through every row of the table (Row Context). The VAR statement captures the specific Team value of the current row being evaluated. This fixed value is crucial for defining the boundary of our group.

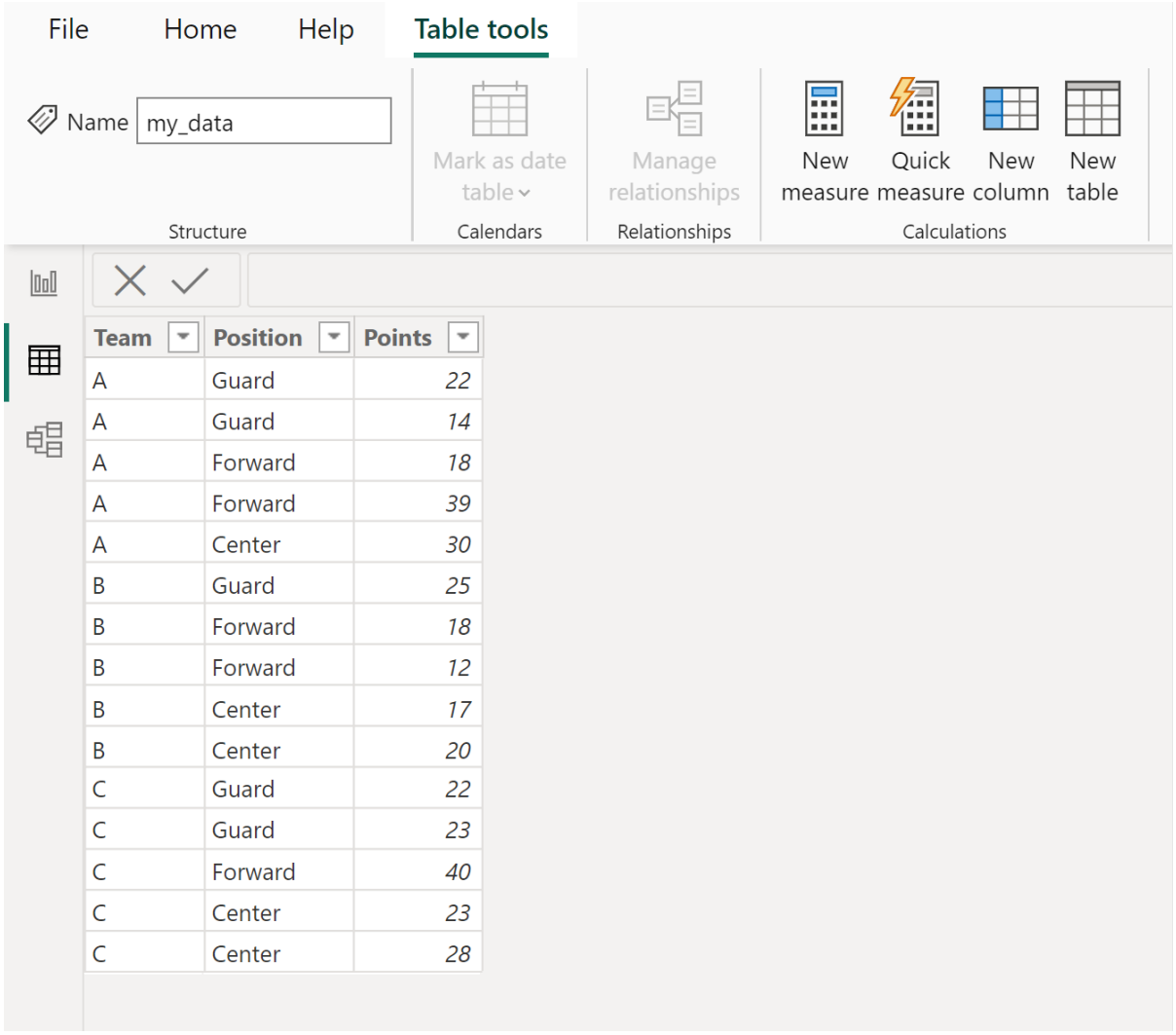
The **current_team** variable stores the value of the **Team** column for the row currently under calculation. This value is then used within the FILTER function. The FILTER function takes the entire **'my_data'** table and restricts it, keeping only the rows where the **Team** column matches the stored **current_team** value. This action is known as a Context Transition, where the row context is used to generate a filter context for the subsequent aggregation function.

Finally, the RANKX function performs the actual ranking. Its first argument is the filtered table returned by the FILTER function--meaning RANKX only ranks records belonging to the single team currently being processed. The second argument, **'my_data'**, specifies the column to be ranked. By leaving the third argument (the value to rank) blank, RANKX defaults to using the value of the current row being processed, ensuring the calculation works correctly within the iteration.

Step-by-Step Practical Example in Power BI

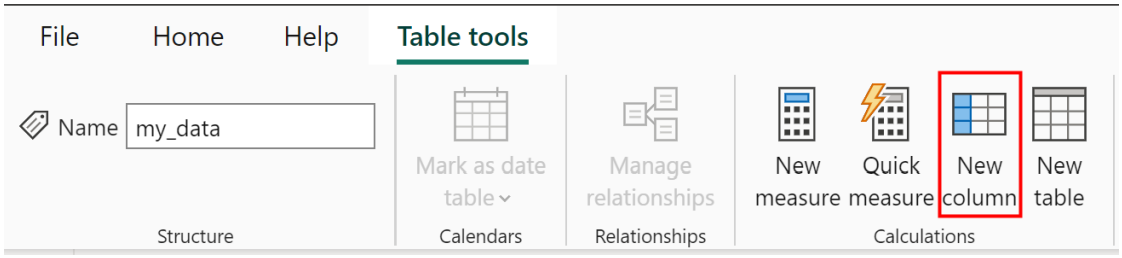
To demonstrate this powerful DAX technique, let us consider a sample dataset loaded into Power BI Desktop. Suppose we have a table named **my_data** that meticulously tracks the points scored by various basketball players, categorizing them by their respective teams. Our objective is to determine the internal ranking of players based on their points within each team.

The initial dataset, **my_data**, contains two crucial columns: **Team** (the grouping column) and **Points** (the value column to be ranked). Visualizing the data helps us understand the structure before applying the DAX calculation.



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

Our goal is to introduce a new column that reflects the rank of the values in the **Points** column, specifically grouped by the corresponding values in the **Team** column. This process begins within the Power BI interface. Navigate to the **Table tools** tab located on the top ribbon of the data view. From there, select the **New column** icon to initiate the column creation process and open the formula bar.

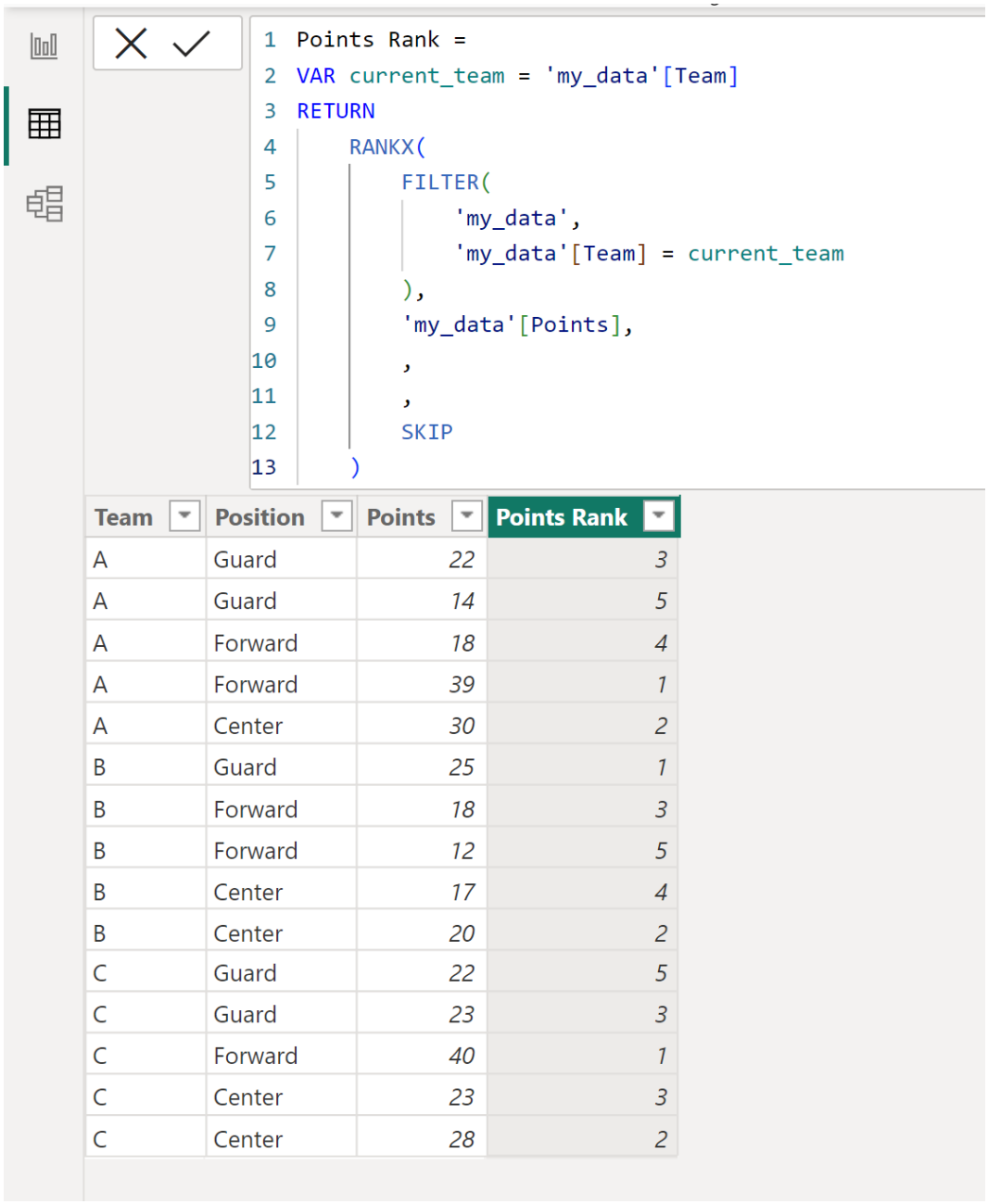


Once the formula bar is active, input the following comprehensive DAX formula. This syntax

ensures the correct isolation of context for accurate grouped ranking:

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

Upon execution, Power BI adds the new calculated column named **Points Rank**. Examining the resulting table confirms that the ranks are assigned correctly within each defined group. Notice how the ranking resets for each distinct team value, successfully achieving the goal of ranking by group.



```
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

Analyzing the Results and RANKX Tie-Handling

The generated **Points Rank** column clearly illustrates the effectiveness of the grouped ranking methodology. For instance, if we focus specifically on Team A, we can observe the following critical ranking assignments based solely on points scored within that team:

The player who achieved the highest point total on Team A (**39 points**) rightfully received a rank of **1**.
The player with the second-highest score on Team A (**30 points**) was assigned a rank of **2**.
The player securing the third-highest score on Team A (**22 points**) received a rank of **3**.

This pattern continues for all records, and the ranking process repeats independently for Team B, Team C, and so on. Furthermore, it is essential to consider how RANKX handles ties, which is controlled by the final argument in the function. In our initial formula, we used **SKIP**.

The **SKIP** parameter is one of two options for managing identical values (ties) during ranking. When **SKIP** is used, if two players are tied for the second-highest score (Rank 2), they both receive Rank 2, and the next player immediately receives Rank 4, skipping Rank 3. The other option, **DENSE**, assigns the same rank to all tied values, but the next rank assigned is consecutive. For example, if two players tie for Rank 2 using **DENSE**, the next rank assigned would be Rank 3, ensuring no ranks are skipped. Choosing between **SKIP** and **DENSE** depends entirely on the analytical requirement of the report.

Modifying the Sort Order: Ranking from Lowest to Highest

By default, when the sort order parameter of the RANKX function is left blank (as in the initial example), the function defaults to descending order (DESC), meaning the highest value receives Rank 1. However, analysts sometimes need to rank in ascending order, where the lowest value receives Rank 1 (e.g., ranking completion time in a race). This modification is straightforward, requiring only the specification of the sort order parameter.

To switch the ranking direction, we must explicitly provide the sort order argument in the RANKX formula. This parameter is the second-to-last argument. To achieve an ascending rank (lowest value ranks highest), we specify the value **1** (or ASC). If we wanted to explicitly specify descending order, we would use **0** (or DESC).

The revised DAX formula below incorporates the ascending sort order parameter. Note the explicit inclusion of the value **1** in the sort parameter position:

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

Executing this modified formula will now assign a rank of **1** to the lowest value within each group, a rank of **2** to the second lowest value in that group, and so forth. This flexibility ensures that the DAX solution can be adapted to various analytical requirements, whether ranking for achievement (highest is best) or cost/time (lowest is best).

Conclusion and Further Power BI DAX Resources

The ability to perform grouped ranking is a cornerstone of advanced data modeling in Power BI. By effectively combining the VAR, FILTER, and RANKX functions, we can overcome the inherent challenges of context transition and produce highly accurate, segment-specific analytical results. This approach ensures that performance metrics are always evaluated relative to their peers within the specified category.

For those seeking to deepen their expertise in DAX and complex calculations, a thorough understanding of the RANKX function is paramount. The official documentation provides comprehensive details on all parameters, including tie-handling and sorting options, which can be essential for handling edge cases in large datasets.

Note: You can find the complete documentation for the **RANKX** function in Power BI via the Microsoft DAX documentation.

Additional Resources

To continue expanding your knowledge of Power BI and sophisticated data modeling techniques, the following resources explain how to perform other common and complex tasks:

Tutorials on calculating running totals or cumulative sums.
Guides for implementing time intelligence functions, such as year-over-year comparisons.
Detailed explanations of how to manage and manipulate evaluation contexts using CALCULATE.