How to Write AVERAGEIF and AVERAGEIFS Functions in VBA (Easy Guide)

Authored by

stats writer

November 20, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Write AVERAGEIF and AVERAGEIFS Functions in VBA (Easy Guide)*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=98357

How do I use the **AVERAGEIF** and **AVERAGEIFS** Functions in **VBA**?

Introduction to Conditional Averaging in VBA

The ability to perform conditional calculations is fundamental to advanced data analysis in **Excel**. When automating complex reports or large-scale data processing using **VBA** (Visual Basic for Applications), developers frequently need to calculate averages based on one or more specific criteria. The native Excel functions, **AVERAGEIF** and **AVERAGEIFS**, are the ideal tools for this task. These functions allow macros to dynamically assess data ranges and return a statistical average only for records that meet the specified conditions. Implementing these functions directly within a VBA procedure streamlines workflows, minimizes manual intervention, and significantly increases the speed and reliability of data manipulation tasks.

To effectively leverage these powerful spreadsheet tools within the VBA environment, one must utilize the **WorksheetFunction** object. This object acts as a necessary gateway, exposing nearly all of Excel's built-in formulas for use in programming scripts. While alternative methods exist, such as building custom loops or using the `Evaluate` method, integrating the standard Excel functions via **WorksheetFunction** is often the cleanest and most efficient approach, especially when dealing with familiar criteria-based calculations. Understanding the proper syntax for passing range arguments and criteria strings is crucial for successful implementation.

Accessing Excel Functions via WorksheetFunction Object

When writing VBA code, native Excel functions like **AVERAGEIF** and **AVERAGEIFS** are not inherently recognized commands within the Basic language structure. To bridge this gap, VBA provides the **WorksheetFunction** object. This object is a member of the Application object and serves as a wrapper, allowing VBA procedures to call and execute the same functions available directly on an Excel worksheet. This method ensures that the complex statistical calculations are performed by Excel's robust calculation engine, guaranteeing accuracy and performance, particularly with large datasets.

The general syntax involves calling `Application.WorksheetFunction.`, followed by the required arguments enclosed in parentheses. When working with range references, it is essential to pass them as proper `Range` objects within VBA, as the function expects these inputs to correctly identify the data areas. Crucially, any criteria used--whether numeric values, text strings, or logical operators (e.g., ">10", "Mavs")--must be formatted correctly as string literals within the VBA code, matching how they would be quoted in a standard Excel formula. The following methods illustrate how to use these conditional average functions effectively within a **VBA** subroutine.

Method 1: Implementing the AVERAGEIF Function in VBA

The **AVERAGEIF** function is used when you need to calculate the average of a range of cells based on a single condition. In a VBA environment, the structure requires three primary arguments: the range containing the criteria (the criterion range), the criterion itself, and the range containing the values to be averaged (the average range). Note that unlike the worksheet formula version where the average range is optional, in VBA it is best practice to always explicitly define all three ranges to avoid ambiguity and potential errors.

The following code snippet demonstrates the implementation of **AVERAGEIF** within a standard VBA sub-procedure. The function identifies all cells in the criteria range (A2:A12) that match the text "Mavs" and then averages the corresponding values in the average range (B2:B12). The result of this calculation is then immediately assigned to cell E2 on the active worksheet, showcasing a powerful way to automate data output based on dynamic calculation.

```
Sub Averageif_Function()
Range("E2") = WorksheetFunction.AverageIf(Range("A2:A12"), "Mavs", Range("B2:B12"))
End Sub
```

This specific code example calculates the average value found in the range **B2:B12**, but only for those rows where the corresponding entry in the criteria range **A2:A12** is precisely equal to the string "Mavs". The resulting numerical average is subsequently stored in the target cell, **E2**. This immediate assignment within the **VBA** macro provides a clean, single-line method for executing and displaying the results of complex conditional statistical analysis. This mechanism is far more efficient than iterating through cells manually using loops and conditional statements (like `If...Then`) to achieve the same result.

Method 2: Implementing the AVERAGEIFS Function in VBA

When data analysis requires applying two or more simultaneous conditions, the **AVERAGEIFS** function becomes indispensable. Unlike its single-criterion counterpart, **AVERAGEIFS** allows for an unlimited number of criterion range/criterion pairs to be supplied, enabling highly specific filtering of data. A crucial difference in the argument structure, however, is that the average range must be listed as the very first argument, followed by the multiple criteria pairs. This positional difference from **AVERAGEIF** must be strictly observed when writing the VBA code to prevent runtime errors.

In the context of **VBA** programming, utilizing the **WorksheetFunction.AverageIfs** method follows the precise syntax required by Excel. The first range specified (the average range) dictates which column's numerical values will be averaged. This is then followed by the first criteria range and its

associated condition, then the second criteria range and its condition, and so forth. All criteria, whether comparing text strings or numerical thresholds using operators like ">" or "<", must be encased in quotation marks when passed as arguments in the subroutine call.

```
Sub Averageifs_Function()
Range("E2") = WorksheetFunction.AverageIfs(Range("C2:C12"), Range("A2:A12"), "Mavs", Range("B2:B12"), ">20")
End Sub
```

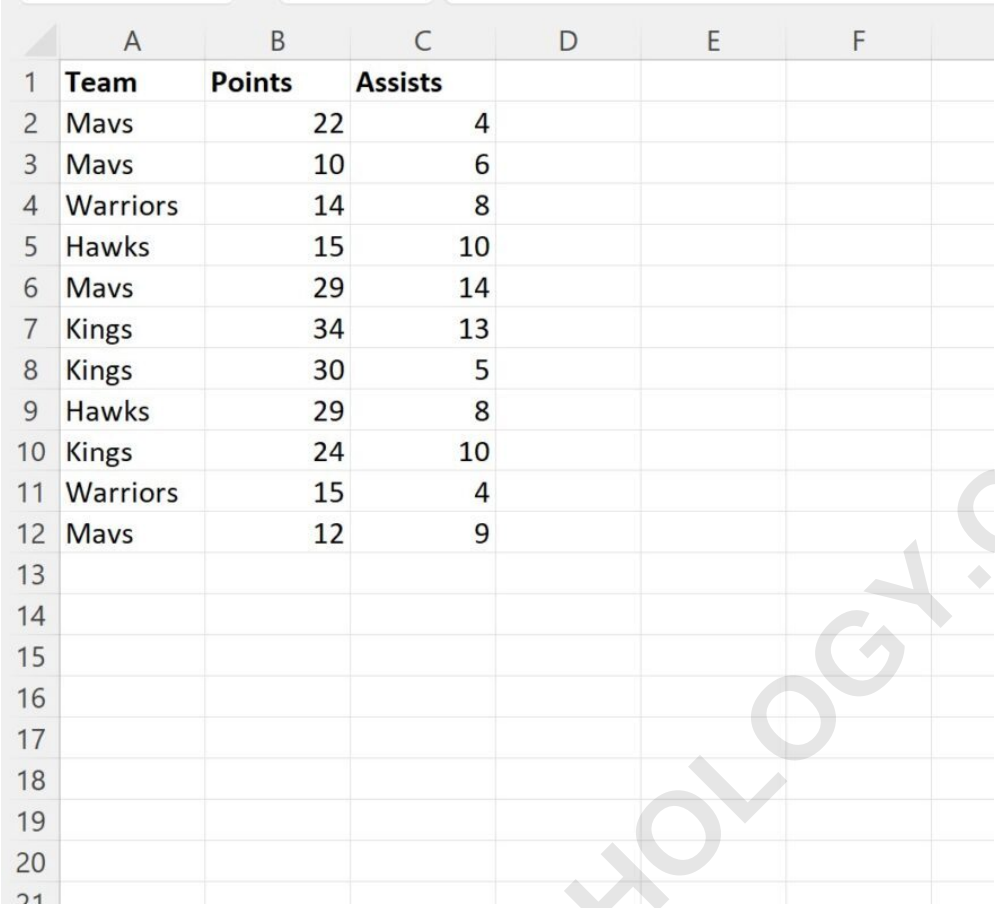
This sophisticated example calculates the average value located in the range **C2:C12**. However, the calculation is only performed on rows that satisfy a dual condition: the corresponding value in range **A2:A12** must equal "Mavs" *and* the corresponding value in range **B2:B12** must be greater than 20. This conditional logic provides granular control over data aggregation, making **AVERAGEIFS** an extremely powerful tool for filtering datasets based on complex business or statistical requirements. The final output is, once again, routed directly to cell **E2** upon execution of the macro.

Setting Up the Sample Dataset for Analysis

To demonstrate the practical application of both the single and multi-criteria averaging functions, we will utilize a sample dataset that tracks statistics for various basketball players. This dataset is structured logically across three columns, making it an excellent candidate for conditional analysis. Column A contains the player's team affiliation (the primary categorical criterion), Column B tracks the points scored (a key numerical criterion), and Column C records the assists generated (the range we might want to average).

The data spans rows 2 through 12, providing a sufficient number of entries to test the filtering capabilities of our **VBA** macros. By visually inspecting this data, we can manually confirm the results generated by the **AVERAGEIF** and **AVERAGEIFS** functions, ensuring the accuracy of our programmatic approach. The structure of the data setup is paramount, as the accuracy of the range references in the VBA code depends entirely on the layout presented below.

The following table snippet illustrates the dataset used throughout the subsequent examples, highlighting the relationship between the Team (Column A), Points (Column B), and Assists (Column C) columns:



	A	B	C	D	E	F
1	**Team**	**Points**	**Assists**			
2	Mavs	22	4			
3	Mavs	10	6			
4	Warriors	14	8			
5	Hawks	15	10			
6	Mavs	29	14			
7	Kings	34	13			
8	Kings	30	5			
9	Hawks	29	8			
10	Kings	24	10			
11	Warriors	15	4			
12	Mavs	12	9			
13						
14						
15						
16						
17						
18						
19						
20						

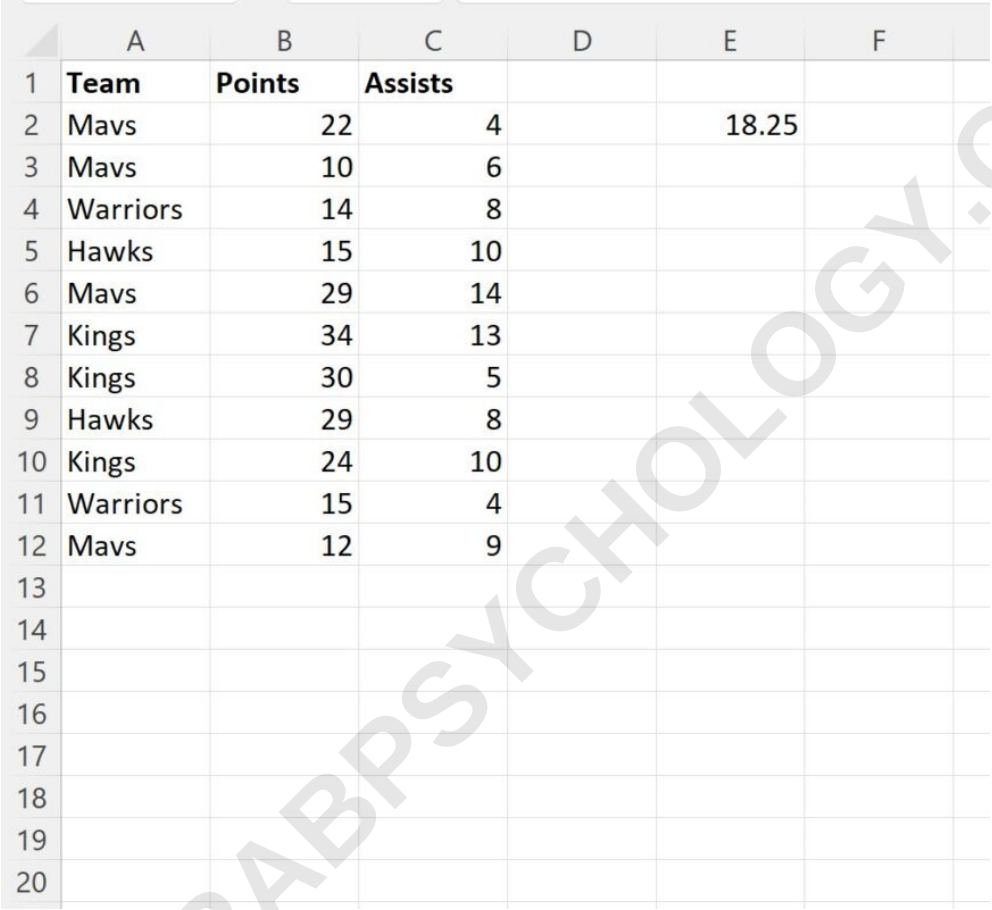
Practical Example 1: Calculating Average with Single Criterion (AVERAGEIF)

Our first goal is to determine the average points scored exclusively by players associated with the "Mavs" team. This task requires applying a single condition to the team column (A2:A12) while averaging the values in the points column (B2:B12). This is a classic application for the **AVERAGEIF** function, demonstrating how quickly **VBA** can extract specific statistical summaries from raw data. The criterion "Mavs" isolates the relevant player records, and the function then performs the necessary aggregation.

To execute this analysis, we define a simple macro named `Averageif_Function`. Within this subroutine, the **WorksheetFunction.AverageIf** method is called. We specify the criteria range as `A2:A12`, the criterion as the string `"Mavs"`, and the average range as `B2:B12`. This sequence precisely mirrors the required arguments for the standard Excel formula, ensuring that the calculation proceeds without error, and yielding the average of points for only the specified team.

```
Sub Averageif_Function()
Range("E2") = WorksheetFunction.AverageIf(Range("A2:A12"), "Mavs", Range("B2:B12"))
End Sub
```

Upon running this macro, the result is computed instantly and displayed in the designated output cell. The following image shows the result of the successful execution. Notice that cell **E2** now contains the calculated value of **18.25**. This figure represents the mean score when considering only those rows where Column A equals "Mavs." We can verify this result by manually calculating the average of the points for the Mavs players: (22 + 10 + 29 + 12) / 4 = **18.25**. This confirms the accuracy and efficiency of utilizing the **WorksheetFunction** method in **VBA** for conditional averaging tasks.



	A	B	C	D	E	F
1	**Team**	**Points**	**Assists**			
2	Mavs	22	4		18.25	
3	Mavs	10	6			
4	Warriors	14	8			
5	Hawks	15	10			
6	Mavs	29	14			
7	Kings	34	13			
8	Kings	30	5			
9	Hawks	29	8			
10	Kings	24	10			
11	Warriors	15	4			
12	Mavs	12	9			
13						
14						
15						
16						
17						
18						
19						
20						

Practical Example 2: Calculating Average with Multiple Criteria (AVERAGEIFS)

For scenarios requiring more precise data filtering, such as finding the average assist count only for top-scoring players on a specific team, we must employ the **AVERAGEIFS** function. This function enables the developer to chain multiple conditions together using an AND logic--meaning a record must satisfy all criteria simultaneously to be included in the average calculation. This capability is essential for generating highly segmented statistical summaries.

For this example, we seek to calculate the average value in the Assists column (C2:C12) for

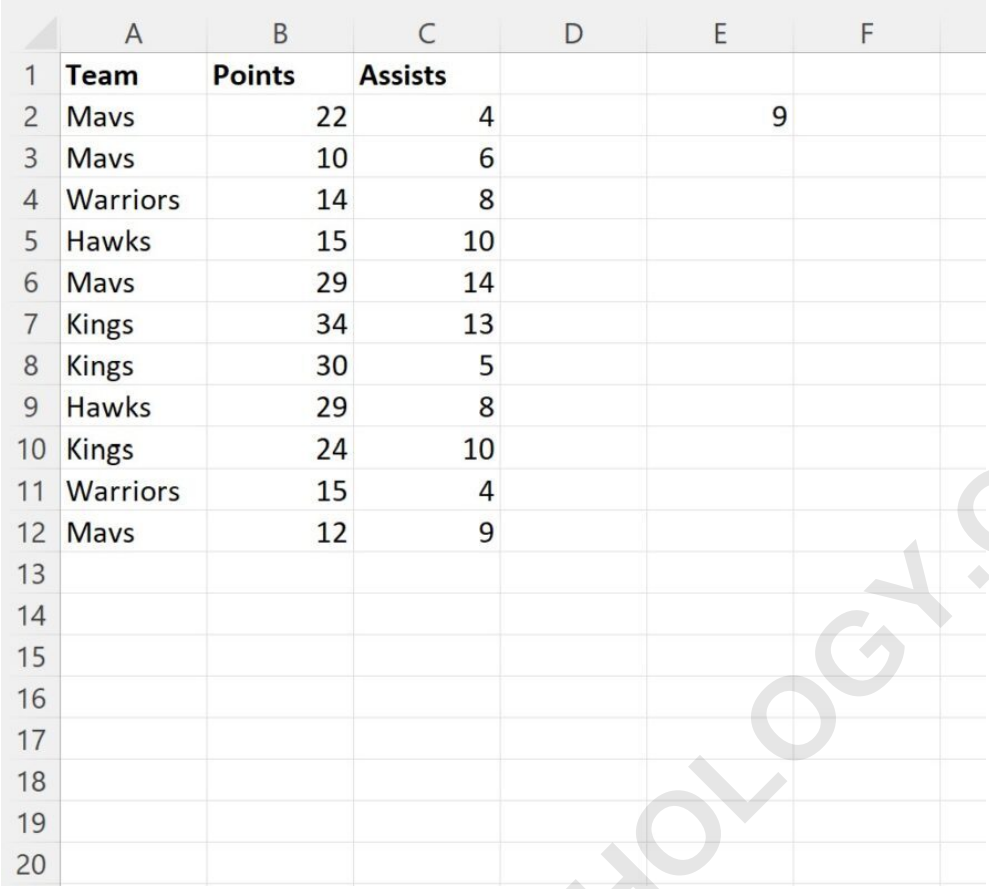
players who meet the following strict set of criteria:

Player must be on the **Mavs** team (Criteria 1).
Player must have scored **more than 20 points** (Criteria 2).

To perform this complex analysis, we write a macro utilizing `WorksheetFunction.AverageIfs`. Remember the critical syntax requirement: the range to be averaged (C2:C12) must be the first argument. It is then followed sequentially by the criteria pairs. In our case, the first pair is the Team column (A2:A12) and the condition "Mavs", and the second pair is the Points column (B2:B12) and the numerical threshold condition ">20".

```
Sub Averageifs_Function()
Range("E2") = WorksheetFunction.AverageIfs(Range("C2:C12"), Range("A2:A12"), "Mavs", Range("B2:B12"), ">20")
End Sub
```

Executing the `Averageifs_Function` macro yields the output displayed below. Cell **E2** now contains the calculated average of **9**. This result is derived only from the players who belong to the Mavs team AND exceeded the 20-point mark. When reviewing the raw data, only two players meet both conditions (Row 2: Mavs, 22 Pts, 12 Assists; and Row 4: Mavs, 29 Pts, 6 Assists). The average of their assists is (12 + 6) / 2 = 9. This demonstrates the precision and power that **AVERAGEIFS** brings to automated reporting through **VBA**. It is important to remember that while this example uses two criteria, the `WorksheetFunction.AverageIfs` method supports virtually any number of criteria ranges, limited only by the memory constraints of the system.



	A	B	C	D	E	F
1	**Team**	**Points**	**Assists**			
2	Mavs	22	4		9	
3	Mavs	10	6			
4	Warriors	14	8			
5	Hawks	15	10			
6	Mavs	29	14			
7	Kings	34	13			
8	Kings	30	5			
9	Hawks	29	8			
10	Kings	24	10			
11	Warriors	15	4			
12	Mavs	12	9			
13						
14						
15						
16						
17						
18						
19						
20						

Advanced Considerations and Best Practices

While using the **WorksheetFunction** method is generally robust, developers should be aware of a critical limitation: if the calculation results in an error (e.g., division by zero because no rows meet the criteria), the VBA code will halt and generate a runtime error (Run-time error '1004'). A best practice to mitigate this is to employ conditional checks or alternative methods. For instance, instead of `WorksheetFunction`, one could use the `Evaluate` method, which processes the formula as if it were entered directly into the sheet, often returning an error value (like `#DIV/0!`) that can be handled using standard VBA error trapping mechanisms (e.g., `On Error Resume Next`) without crashing the macro.

Furthermore, mastering the criteria syntax is vital. When the criterion involves comparing values using operators (e.g., "", "<="), the entire expression must be enclosed in double quotes, such as `">20"`. If the criterion is based on a dynamic cell value, the developer must concatenate the operator and the cell content. For example, if cell F1 contains the minimum threshold, the criterion should be written as `">" & Range("F1").Value`. This practice ensures flexibility, allowing the macro to adapt to changing criteria without requiring direct code modification. Always ensure that the criteria ranges and the average range are of the exact same size and orientation (all vertical or

all horizontal) to prevent incorrect alignment during the conditional assessment performed by the Excel engine.