How to Easily Count Cells with Specific Text in Excel Using VBA

Authored by

stats writer

November 20, 2025

VBA, or **Visual Basic for Applications**, provides powerful tools for automating tasks and performing complex data manipulations within Excel. While Excel offers native functions like COUNTIF, using VBA allows for greater flexibility, especially when integrating counting logic into larger, more intricate processes or when handling criteria that are too complex for standard spreadsheet formulas. This guide focuses on leveraging VBA to accurately count cells containing specific text strings within a defined range.

The fundamental approach involves iterating through a designated cell range and applying a conditional check to determine if the cell's content matches the required text pattern. If the condition is met--meaning the cell **contains** the specified substring--a counter variable is incremented. This methodical process continues until every cell in the target range has been examined. The final value of the counter variable represents the total number of cells that meet the criteria. However, for sheer efficiency in simple counting tasks, we can often bypass the explicit loop structure by utilizing built-in Excel functions directly accessible through VBA, specifically the **WorksheetFunction** object.

Understanding how to integrate these concepts is crucial for any advanced VBA user. Although a custom loop offers absolute control and is necessary for complex pattern matching (like using Regular Expressions), relying on the optimized **WorksheetFunction** methods, such as WorksheetFunction.CountIf, is significantly faster for pattern matching tasks involving standard text strings and **wildcard characters**. This approach minimizes execution time and keeps the code clean and highly readable, which is paramount when developing scalable solutions within the spreadsheet environment.

Understanding the Core Mechanism: Utilizing Worksheet Functions in VBA

When aiming to perform calculations or comparisons within Excel via a **macro**, the most efficient method often involves integrating native spreadsheet functions. VBA allows direct access to virtually all standard Excel functions through the **WorksheetFunction** object. This method is preferred over manual iteration (a `For Each` loop) because the built-in functions are highly optimized C++ components, executing much faster than interpreted VBA code. For counting cells based on criteria, the **CountIf function** is the ideal tool, provided it is called correctly within the VBA environment.

The basic syntax for accessing and executing the CountIf function through the **WorksheetFunction** object requires two primary arguments: the **range** to evaluate and the **criteria** for counting. Critically, when searching for text that exists *within* a cell's contents, rather than an exact match, we must employ specific symbols known as **wildcard characters** within the criteria string. Failure to use these **wildcard characters** will result in the function only counting cells where the content is an exact match for the search term, which usually leads to an incorrect

count for substring searches.

The following code snippet demonstrates the foundational structure for using WorksheetFunction.CountIf within a standard VBA procedure. This method is straightforward and immediately assigns the calculated count to a specified cell, bypassing the need for temporary variable declaration if the output location is known beforehand. This is the simplest and most direct approach for quick output display in the spreadsheet itself.

Method 1: Using the Basic CountIf Syntax for Direct Output

To perform a count and immediately place the result into a specific cell on the worksheet, we use a single line of code within our VBA subroutine. This method is exceptionally useful for dashboard creation or when refreshing calculation cells based on macro execution. We define the target cell on the left side of the assignment operator (equals sign), and the counting logic on the right side.

The structure begins with the target cell reference, such as `Range("D2")`, followed by the formula execution: `WorksheetFunction.CountIf`. Within the parentheses, the first argument defines the scope (e.g., `Range("A2:A13")`), and the second argument defines the text criteria (e.g., `"*avs*"`). The inclusion of the asterisk **wildcard characters** ensures that the function counts the cells that contain "avs" anywhere within the text string, whether at the beginning, middle, or end.

You can use the following basic syntax to count the number of cells in a range that contain a specific text using VBA, where the result is written directly back to a cell:

```
Sub CountCellsWithText()
Range("D2") = WorksheetFunction.CountIf(Range("A2:A13"), "*avs*")
End Sub
```

This particular example counts the number of cells in the range **A2:A13** that contain "avs" and then assigns the resulting count directly to cell **D2**. Note that this method is fast, clean, and ideal for outputting results directly into a report structure within Excel. It is a powerful way to integrate dynamic counting without manually entering a formula into the spreadsheet interface.

The Importance of Wildcard Characters in Text Search Criteria

The successful application of the CountIf function for substring matching hinges entirely on the correct use of **wildcard characters**. A **wildcard character** is a special symbol used to represent one or more unknown characters. In Excel formulas and the corresponding CountIf function, the asterisk (*) is the primary wildcard used for matching any sequence of characters, including zero characters.

When defining the criteria string in WorksheetFunction.CountIf, enclosing the target text within asterisks--as in `"*avs*"`--tells the function to look for the literal string "avs" preceded by any number of characters (represented by the first asterisk) and followed by any number of characters (represented by the second asterisk). This construction ensures that the function counts cells like "Cavaliers," "Avs_Team," or "The Avs," as all of them contain "avs" somewhere within their text.

If you were to omit the **wildcard characters** and simply use `"avs"` as the criteria, the CountIf function would only return a count if a cell contained the text "avs" and nothing else. This strict equality check is rarely what is intended when trying to find occurrences of a substring within a larger body of text. Therefore, remember: the asterisks (*****) are used as **wildcard characters** in the **Countif** function to enable partial text matching.

Method 2: Displaying the Count using a Message Box (MsgBox)

While outputting results directly to a cell is efficient for reporting, there are scenarios where displaying immediate, transient feedback to the user is preferred. This is where the MsgBox function comes into play. By using MsgBox, we can present the resulting cell count in a modal pop-up window, requiring the user to acknowledge the message before continuing with other spreadsheet tasks. This method is excellent for providing immediate status updates after a **macro** runs or for debugging purposes.

To implement this, we must first declare a variable to temporarily store the calculated count. This variable, typically defined as an `Integer` or `Long`, holds the output of the WorksheetFunction.CountIf calculation. After the calculation is complete and the count is stored in the variable (e.g., `cellCount`), we then use the **MsgBox** function to display a customized string that concatenates descriptive text with the numerical value stored in the variable.

If you would instead like to display the count of cells in a message box, you can use the following syntax. Notice the use of comments (lines starting with a single quote) to explain the different stages of the code, which is a best practice for maintaining readable and self-documenting **macros**.

Sub CountCellsWithText()

Dim cellCount As Integer

'Calculate number of cells that contain 'avs'
cellCount = WorksheetFunction.CountIf(Range("A2:A13"), "*avs*")

'Display the result
MsgBox "Cells that contain avs: " & cellCount

End Sub

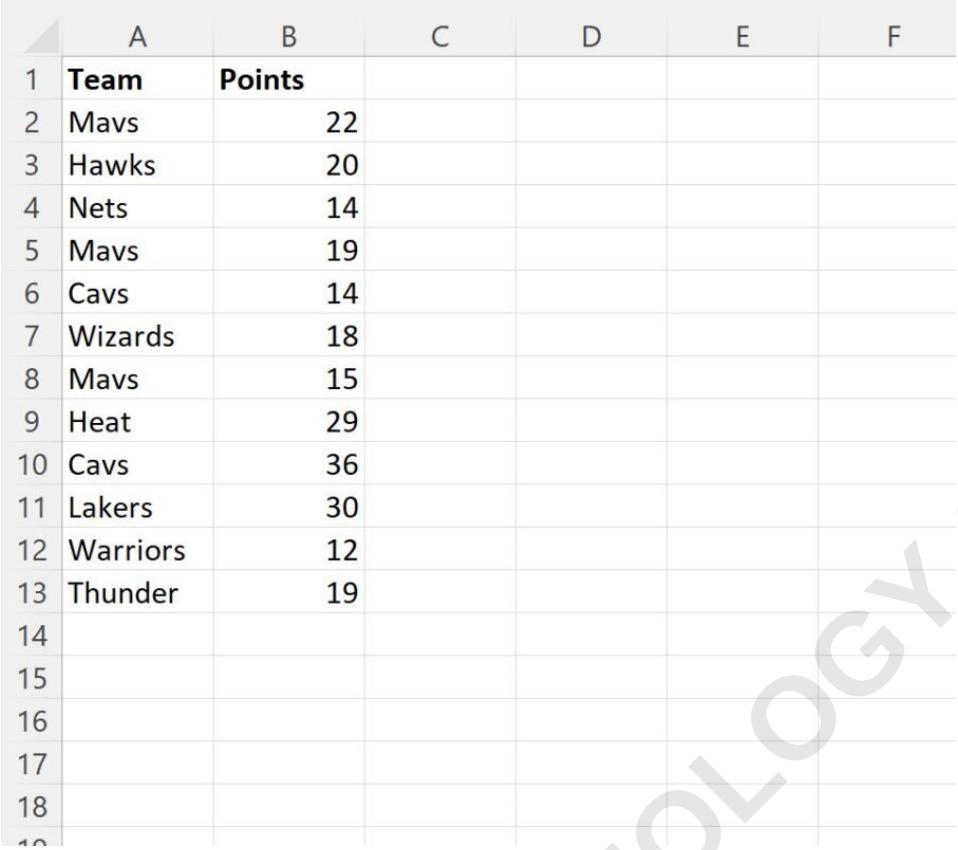
Practical Application: Analyzing Sample Dataset

To solidify our understanding of these two VBA counting methods, let us apply them to a real-world scenario using a sample dataset within Excel. Our dataset contains information about various basketball players, including their team affiliations. Our objective is to determine how many teams in the list contain the specific text string "avs".

The dataset is structured with team names located in the range A2:A13. This range will serve as the input for the range argument in our CountIf function calls. By visually examining the data, we can anticipate the expected count, which helps confirm the accuracy of our **macros**. Teams like the "Cavaliers," "Mavericks," and "Lakers" will be evaluated based on the criteria `"*avs*"`.

The following visual representation shows the starting point of our analysis, detailing the player and team columns before any **macro** execution. This context is important as we move into the specific examples demonstrating direct cell output versus interactive messaging.

The following examples shows how to use each of these methods in practice with the following dataset in Excel that contains information about various basketball players:



	A	B	C	D	E	F
1	**Team**	**Points**				
2	Mavs	22				
3	Hawks	20				
4	Nets	14				
5	Mavs	19				
6	Cavs	14				
7	Wizards	18				
8	Mavs	15				
9	Heat	29				
10	Cavs	36				
11	Lakers	30				
12	Warriors	12				
13	Thunder	19				
14						
15						
16						
17						
18						

Example 1: Count Cells with Specific Text Using VBA and Display Results in Cell

Our first task is to execute the counting operation and directly write the resulting number to a designated output cell, specifically **D2**. This is the implementation of Method 1, designed for silent background calculations where the result needs to be immediately available on the sheet for further analysis or reporting.

Suppose we would like to count the number of cells that contain "avs" in the team name within the range A2:A13 and output the results in a specific cell, **D2**. We initiate the **VBA** editor (ALT+F11), insert a new module, and define our subroutine. The **macro** is concise, making it highly efficient for this specific purpose.

We can create the following **macro** to do so, ensuring the range parameters and the criteria string are accurately defined to target the team names and the substring "avs":

```
Sub CountCellsWithText()
Range("D2") = WorksheetFunction.CountIf(Range("A2:A13"), "*avs*")
End Sub
```

When we run this **macro** (e.g., by pressing F5 in the VBA editor or assigning it to a button), the operation executes instantly. The formula calculates how many cells in the specified range satisfy the **wildcard characters** criteria and places the result into cell **D2**.

When we run this **macro**, we receive the following output, showing the result populated in cell D2:

	A	B	C	D	E	F
1	**Team**	**Points**				
2	Mavs	22		5		
3	Hawks	20				
4	Nets	14				
5	Mavs	19				
6	Cavs	14				
7	Wizards	18				
8	Mavs	15				
9	Heat	29				
10	Cavs	36				
11	Lakers	30				
12	Warriors	12				
13	Thunder	19				
14						
15						
16						
17						
18						
19						

Notice that cell **D2** now contains a value of **5**. This output confirms that five teams in the dataset contain the sequence of letters "avs" (e.g., Cavaliers, Mavericks). This demonstrates the speed and clarity of using WorksheetFunction.CountIf within VBA for direct reporting results.

Example 2: Count Cells with Specific Text Using VBA and Display Results in Message Box

In contrast to the direct cell output, this second example illustrates Method 2, where the primary goal is user interaction and immediate feedback. Instead of modifying the worksheet, the result is captured in a variable and presented using the **MsgBox** function. This is particularly useful when the count is part of a verification step or simply needs to be communicated to the user without altering the data layout.

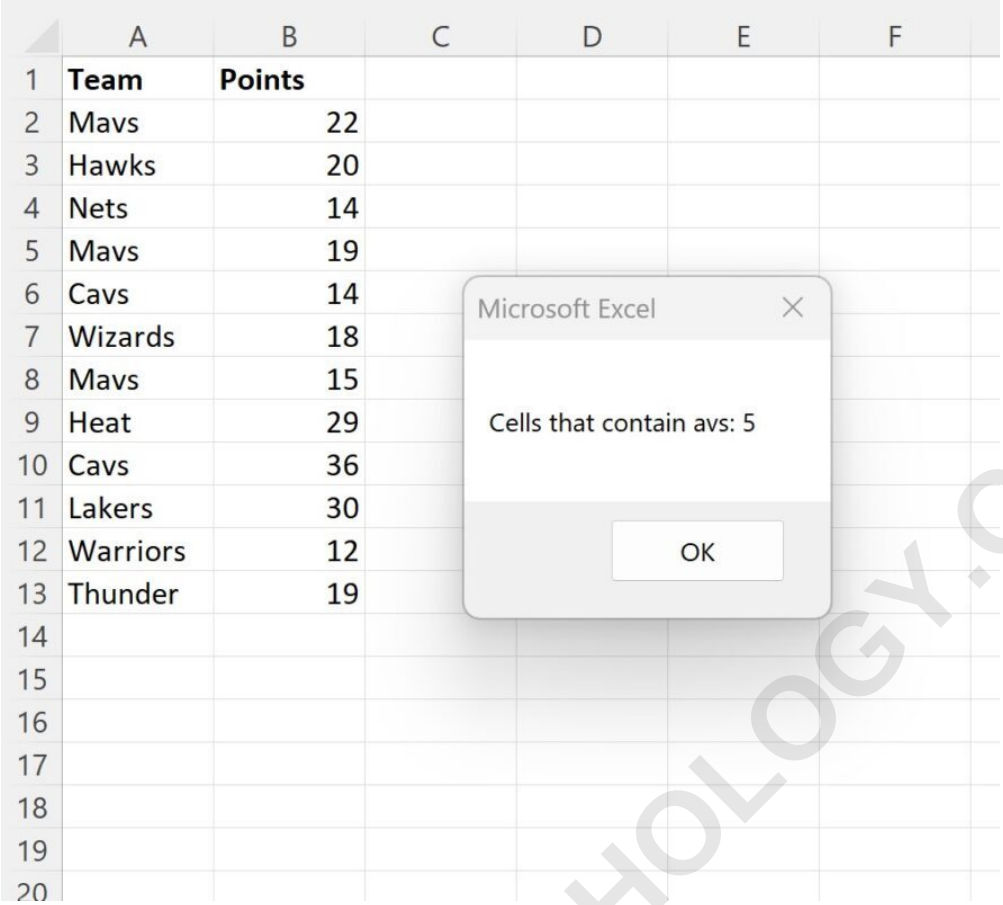
We will utilize the same calculation logic, `WorksheetFunction.CountIf(Range("A2:A13"), "*avs*")`, but the output will be routed to the `cellCount` variable. The subsequent use of the **MsgBox** command then formats and displays this variable's content. This two-step process (calculate, then display) is standard for user-facing outputs in VBA.

We can create the following **macro** to perform the calculation and display the result via a pop-up dialog box:

```
Sub CountCellsWithText()

Dim cellCount As Integer

'Calculate number of cells that contain 'avs'
cellCount = WorksheetFunction.CountIf(Range("A2:A13"), "*avs*")

'Display the result
MsgBox "Cells that contain avs: " & cellCount
End Sub
```

Upon running this **macro**, the counting logic is executed behind the scenes. The result is captured, and the **MsgBox** command then generates the pop-up window, clearly presenting the final tally to the user.

When we run this **macro**, we receive the following output:



The message box tells us that there are **5** cells that contain "avs" in the team name. This confirms that regardless of the output method--direct cell writing or user interface interaction--the core counting logic provided by **WorksheetFunction.CountIf** remains robust and effective for partial text matching in Excel.

Conclusion: Choosing the Right VBA Approach

Counting cells that contain specific text is a common necessity in data analysis, and VBA offers streamlined, powerful solutions through the **WorksheetFunction** object. By mastering the integration of CountIf and the crucial use of **wildcard characters**, developers can quickly implement reliable counting mechanisms.

Choosing between the two demonstrated methods depends on the purpose of the **macro**. If the goal is data reporting and integration into a dashboard, Method 1 (direct cell assignment) is preferable. If the goal is immediate user feedback, status updates, or interactive debugging, Method 2 (using a variable and MsgBox) is the better choice. Both methods leverage the highly optimized performance of Excel's native functions, ensuring fast execution even on large datasets.

For those seeking even more complex text analysis, VBA also supports manual looping combined

with string manipulation functions like `InStr` or `Like` operators, providing maximum flexibility when criteria go beyond what CountIf can handle. However, for standard substring counts, the **WorksheetFunction.CountIf** remains the benchmark for simplicity and speed.