What is the VBA equation for a “not equal to” auto filter?

Authored by

stats writer

November 18, 2025

Understanding the "Not Equal To" Operator in VBA Filtering

When working with large datasets in Excel, filtering is an essential technique for isolating specific records. If you are automating this process using VBA (Visual Basic for Applications), you will frequently encounter scenarios where you need to exclude certain values rather than include them. The logical operator used to represent "not equal to" within a VBA AutoFilter operation is the combined symbol: **<>**.

This symbol is standard across various programming contexts and serves as the primary method for expressing negative criteria when applying automated filters. Understanding how to correctly embed this operator within the criteria string is crucial for writing efficient and accurate filtering macros. We will explore how to integrate **<>** into the **Criteria1** argument of the **AutoFilter** method to exclude specific data points from your visible set.

While the **AutoFilter** method is powerful, it requires precise syntax, especially when dealing with operators. When defining a filter criterion that must exclude a specific string or numeric value, the criteria string must begin with **<>** followed immediately by the value you wish to negate. This ensures that the filter recognizes the intent is to show everything *except* the specified term.

The Core VBA Syntax for Negation

The standard **AutoFilter** method applies filters directly to a specified range, requiring arguments for the field index and the filter criteria. To implement the "not equal to" logic, the criteria argument is passed as a string prefixed by the negation operator. This technique is highly reliable for quick, exclusion-based filtering tasks in automated procedures.

Here is a fundamental example demonstrating how the negation operator is integrated into a typical VBA script. This structure forms the foundation for more complex filtering operations, ensuring that the second column (Field:=2) of the defined range excludes the value "Center":

```
Sub FilterNotEqualTo()

Range("A1:C11").AutoFilter Field:=2, Criteria1:="<>Center"

End Sub
```

In this specific snippet, the AutoFilter method is applied to the range **A1:C11**. The **Field:=2** parameter instructs the filter to evaluate the second column within that range (Column B, assuming A is the first). Most importantly, the argument **Criteria1:="<>Center"** ensures that only rows where the value in the second column is **not equal to Center** are displayed. All other matching rows are temporarily hidden.

Deconstructing the AutoFilter Method Arguments

A deeper understanding of the **AutoFilter** method is necessary to leverage its full potential, particularly when developing robust VBA solutions. The method operates on a Range object and accepts several optional parameters, but **Field** and **Criteria1** are mandatory for single-criterion filtering.

The **Field** argument is defined as a Long integer specifying the column number in the range to be filtered. This number is relative to the *start* of the range defined (e.g., if the range is C1:F10, Field:=1 refers to Column C, Field:=2 refers to Column D, and so on). This relative indexing is vital for preventing errors when copying code across different spreadsheets.

The Criteria1 argument, where we implement our negation, is typically a String. This string can contain operators like **<>** (not equal to), **>** (greater than), **<** (less than), or wildcard characters (like *****). By prefixing the desired value with **<>**, we are effectively telling the VBA engine to invert the match operation, ensuring an exclusion filter is applied effectively and immediately upon execution of the macro.

Practical Application: Filtering Single Criteria (The Basketball Dataset)

To illustrate this concept clearly, let us apply this knowledge to a practical scenario. Imagine a dataset containing information about various basketball players, including their names, positions, and scores. Our goal is to filter this list to exclude all players occupying the "Center" position, allowing us to focus only on Guards and Forwards.

Suppose our data structure in Excel looks like this:

	A	B	C	D	E
1	**Team**	**Position**	**Points**		
2	A	Guard	20		
3	A	Guard	25		
4	A	Forward	31		
5	A	Forward	14		
6	A	Center	19		
7	B	Guard	25		
8	B	Guard	28		
9	C	Forward	13		
10	C	Center	19		
11	C	Center	22		
12					
13					
14					
15					
16					

The dataset spans the range **A1:C11**. We intend to filter the second column, which corresponds to the **Position**. Since we only want to see rows where the value in the **Position** column is not equal to "Center", we must construct a VBA macro specifically tailored to this exclusion requirement. This approach provides a quick, automated way to analyze subsets of data without manual interaction with the filter menus.

Step-by-Step Implementation of the Single Criteria Macro

To execute the filtering operation, we create a simple subroutine that targets the defined range and applies the **AutoFilter** method. This macro encapsulates the entire process, making the filtering operation repeatable and efficient.

The following macro explicitly targets the Position column (Field 2) and specifies the negation criteria using **<>Center**:

Sub FilterNotEqualTo()

Range("A1:C11").AutoFilter Field:=2, Criteria1:="<>Center"

End Sub

When this routine is run, the VBA engine executes the **AutoFilter** on the specified range headers, applying the criterion defined in **Criteria1**. It evaluates each cell in the second column against the condition "is not equal to Center" and subsequently hides all rows that fail this test. This instantaneous visual feedback confirms the macro's effectiveness.

Analyzing the Results of Single Criteria Filtering

Upon successful execution of the macro detailed above, the spreadsheet will immediately update to reflect the filtered results. Only those rows that satisfy the exclusion rule will remain visible.

	A	B	C	D	E	F
1	**Team**	**Position**	**Points**			
2	A	Guard	20			
3	A	Guard	25			
4	A	Forward	31			
5	A	Forward	14			
7	B	Guard	25			
8	B	Guard	28			
9	C	Forward	13			
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						

Observing the output, it is clear that all records where the position was "Center" have been hidden. The visible dataset now consists solely of players designated as "Guard" or "Forward". This outcome demonstrates the successful application of the **<>** operator within the VBA **AutoFilter** method, confirming that the syntax correctly interpreted the requirement for exclusion.

A crucial point to remember is that the **AutoFilter** method does not delete data; it simply alters the visibility of rows based on the applied criteria. To revert to the original view, you can either manually clear the filter or, more efficiently, add a line of code at the beginning of your macro (or a separate macro) to check if a filter is active and clear it before applying a new one.

Advanced Filtering: Excluding Multiple Values

While filtering for a single exclusion is straightforward, real-world data manipulation often requires excluding records based on two or more separate criteria simultaneously. The AutoFilter method supports two distinct criteria arguments, **Criteria1** and **Criteria2**, which can be combined using the logical **And** or **Or** operators (though the logical operator defaults to **And** when **Criteria2** is provided).

When using two criteria, the filter typically uses an **And** logic, meaning that a row must satisfy both **Criteria1** AND **Criteria2** to be hidden or displayed, depending on the operator used. For exclusion filtering (using **<>**), we want rows to be displayed only if the value is *not* equal to the first value AND *not* equal to the second value.

For example, suppose we wish to refine our basketball player list further. We want to exclude players who are "Center" AND also exclude players who are "Guard." We must use both the **Criteria1** and **Criteria2** arguments, each employing the **<>** operator to define the respective negations. This expansion significantly enhances the flexibility of our automated filtering processes.

Implementing Multiple Criteria Using Criteria1 and Criteria2

To exclude both "Center" and "Guard" positions, we utilize the **Criteria1** and **Criteria2** parameters within a single **AutoFilter** call. Since we are using two exclusion criteria on the same field (Field:=2), the implicit relationship between the two criteria is an **And** operation--the row must satisfy that the position is not "Center" AND the position is not "Guard."

The following macro demonstrates the required syntax for dual exclusion filtering:

Sub FilterNotEqualTo()

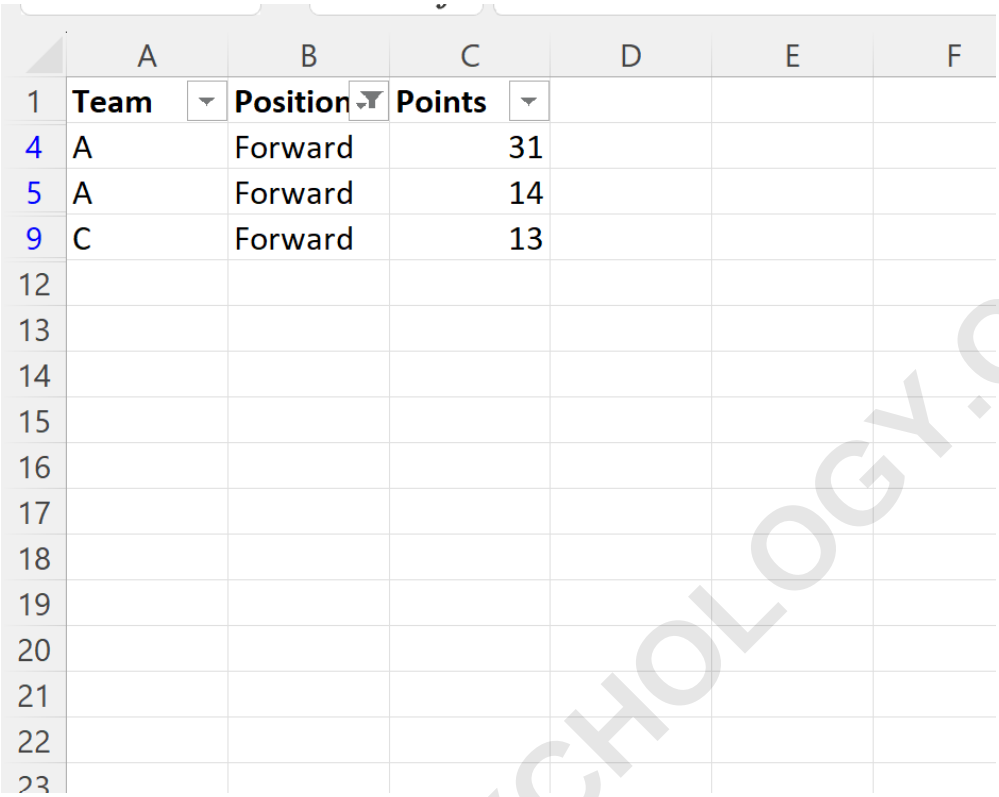
Range("A1:C11").AutoFilter Field:=2, Criteria1:="<>Center", Criteria2:="<>Guard"

End Sub

In this updated script, **Criteria1** excludes "Center" and **Criteria2** excludes "Guard." Only rows that successfully pass both negation tests will remain visible. This powerful feature allows developers to quickly isolate highly specific subsets of data in Excel, streamlining reporting and analysis tasks efficiently using VBA.

Visualizing the Multi-Criteria Results

Running the multi-criteria macro provides an even more refined view of the data. Since we have filtered out both "Center" and "Guard" positions, only those records corresponding to "Forward" players should now be displayed in the visible range.



	A	B	C	D	E	F
1	**Team**	**Position**	**Points**			
4	A	Forward	31			
5	A	Forward	14			
9	C	Forward	13			
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						

As demonstrated by the output, the dataset has been successfully filtered to only show rows where the value in the **Position** column is **not equal to Center** *and* **not equal to Guard**. This sophisticated application of the **AutoFilter** method using dual criteria allows for extremely precise data manipulation within macro execution.

Further Resources and Best Practices

Mastering the use of operators like **<>** is fundamental to writing effective and dynamic VBA code. When automating data filtering, always ensure that your range reference is accurate and that your field index correctly corresponds to the column you intend to filter.

For comprehensive details regarding all available parameters, including the optional **Operator** parameter (which defaults to **xlAnd** when using **Criteria2**), it is highly recommended to consult the official documentation. The **AutoFilter** method is robust, supporting date filtering, numerical comparisons, and complex string patterns using wildcards in conjunction with the negation operator.

For detailed official documentation on the **AutoFilter** method and its arguments, especially concerning **Criteria1** and **Criteria2**:

VBA Range.AutoFilter Method Documentation

For related advanced filtering techniques, consider exploring methods for applying filters using arrays or multiple criteria across different fields:

VBA: How to Use AutoFilter with Multiple Criteria