How to Easily Clear All Filters in Excel with a Simple VBA Macro

Authored by

stats writer

November 19, 2025

Introduction to VBA and Filter Management

Managing large datasets in Excel often requires the heavy use of filters to isolate specific records. While manually clicking the "Clear Filter" button works for occasional use, automating this process becomes essential when dealing with frequently updated reports or complex analytical workflows. VBA (Visual Basic for Applications) provides the powerful scripting tools necessary to execute these tasks instantly and reliably. Utilizing a concise **macro** allows users to bypass the interface steps and clear all applied criteria across the active sheet with a single command.

The primary goal of a filter-clearing **macro** is to reset the visibility of all rows. When filters are active, some rows are hidden based on criteria specified by the user. The VBA approach targets the fundamental state of the worksheet, ensuring that every record is restored to visibility, effectively removing all current filter criteria. This method is not just about convenience; it significantly reduces the potential for manual error, ensuring data integrity across automated processes.

Although one might initially think a complex loop structure is required to iterate through every column's filter status, Excel provides a streamlined method specifically designed for this purpose. The code we will explore leverages native VBA objects and methods that handle the internal complexity, offering a much cleaner and more robust solution than manually resetting individual filter objects. This efficiency is critical for scripts running in high-volume environments.

Understanding Excel's AutoFilter Mechanism

Before diving into the code, it is important to understand how Excel manages filtering. The filtering mechanism is contained within the **AutoFilter** object, which exists on the **Worksheet** level. When filters are applied to a data range, the sheet enters **AutoFilterMode**. This mode indicates that the header row contains filter drop-down arrows and that row visibility may be affected by set criteria.

The key to successful filter management using VBA lies in correctly identifying the state of the sheet. If a sheet is not currently in **AutoFilterMode** (meaning no filters have been turned on), attempting to clear non-existent filters would typically result in a runtime error if not properly handled. Therefore, professional macro writing always includes a conditional check to verify the presence of an active filter setup before proceeding with the clearing action.

While the original text mentioned iterating through individual filters--an approach sometimes used for complex scenarios involving specific slicers or pivot tables--the most common requirement is simply to clear **all** applied criteria across the standard data range. This comprehensive clearing action is encapsulated by a single, powerful method that instantly reverts the sheet to its unfiltered state, making all previously hidden rows visible again. This unified command avoids the overhead of managing individual column filter states.

The Core VBA Syntax for Clearing Filters

The fundamental function for clearing all applied filters in Excel via VBA relies on the **ShowAllData** method. This method forces all rows hidden by the **AutoFilter** mechanism to become visible, simultaneously removing all criteria set across the filtered range. This is the cleanest and most reliable way to achieve a complete filter reset.

The following macro structure provides the robust solution, incorporating a crucial check to see if filters are active before attempting to clear them:

Sub ClearFilters()

If ActiveSheet.AutoFilterMode Then ActiveSheet.ShowAllData

End Sub

This particular **macro** will clear all filters on the sheet that is currently active. The use of **ActiveSheet** ensures that the code targets the sheet the user is currently interacting with, minimizing ambiguity and simplifying the user experience. The conditional check using the **If...Then** structure prevents potential errors if the sheet is not currently filtered, ensuring smooth execution regardless of the sheet's current state.

The entire operation hinges on the execution of **ActiveSheet.ShowAllData**. This command is executed only if **ActiveSheet.AutoFilterMode** evaluates to **True**. If no filtering is active, the condition is skipped, and the macro ends harmlessly, which is a key trait of well-designed VBA code.

Deconstructing the `ShowAllData` Method

The **ShowAllData** method is central to filter clearing in VBA. It is a method of the **Worksheet** object designed specifically to remove the filtering effect. When executed, it achieves two results simultaneously: it resets the filter criteria back to "All" for every column, and it restores the visibility of all rows that were previously hidden due to the active criteria.

Understanding the internal workings of this method helps in debugging and advanced scenarios. The ShowAllData command effectively tells Excel to ignore all current filter settings and revert the display state. Critically, running this command does **not** turn off the filter arrows themselves; it merely clears the criteria. The sheet remains in AutoFilterMode until the user manually toggles the filter off, or until the macro explicitly sets **ActiveSheet.AutoFilterMode = False**, though clearing the data is usually sufficient for most user requirements.

This ShowAllData method is superior to attempting to loop through filters and manually setting **filt.On = False**, especially in cases where multiple criteria (like filtering by color, icon, or complex custom formulas) are involved. The single method ensures comprehensive clearing regardless of the complexity of the existing filter rules. Furthermore, if there are not any rows currently being filtered on the active sheet, then nothing will happen when you run this **macro**, providing a failsafe mechanism against unnecessary processing.

Step-by-Step Implementation of the Clearing Macro

Implementing this filter-clearing **macro** requires following standard VBA development practices. This involves accessing the Visual Basic Editor (VBE), inserting a new module, and pasting the code. Consistent naming conventions, such as **ClearFilters**, help organize the code library.

Access the VBE: Open the target Excel workbook and press **Alt + F11** to launch the Visual Basic Editor.

Insert a New Module: In the VBE project explorer window, right-click on the workbook name, select **Insert**, and then choose **Module**. This provides a blank slate for your new code.

Paste the Code: Carefully copy and paste the **Sub ClearFilters()** code block into the newly created module. Ensure the syntax, especially the conditional check, is preserved to maintain robustness.

Test Execution: Return to the Excel sheet, apply a simple filter, and then run the macro by pressing **Alt + F8**, selecting **ClearFilters**, and clicking **Run**. Verify that all hidden rows are restored.

For enhanced user experience, this macro can be assigned to a button, a custom keyboard shortcut, or integrated into a ribbon tab. Assigning it to a readily accessible control makes the clearing operation immediate, transforming a multi-click manual process into a single, automated step, significantly improving productivity when analyzing large, filter-dependent spreadsheets.

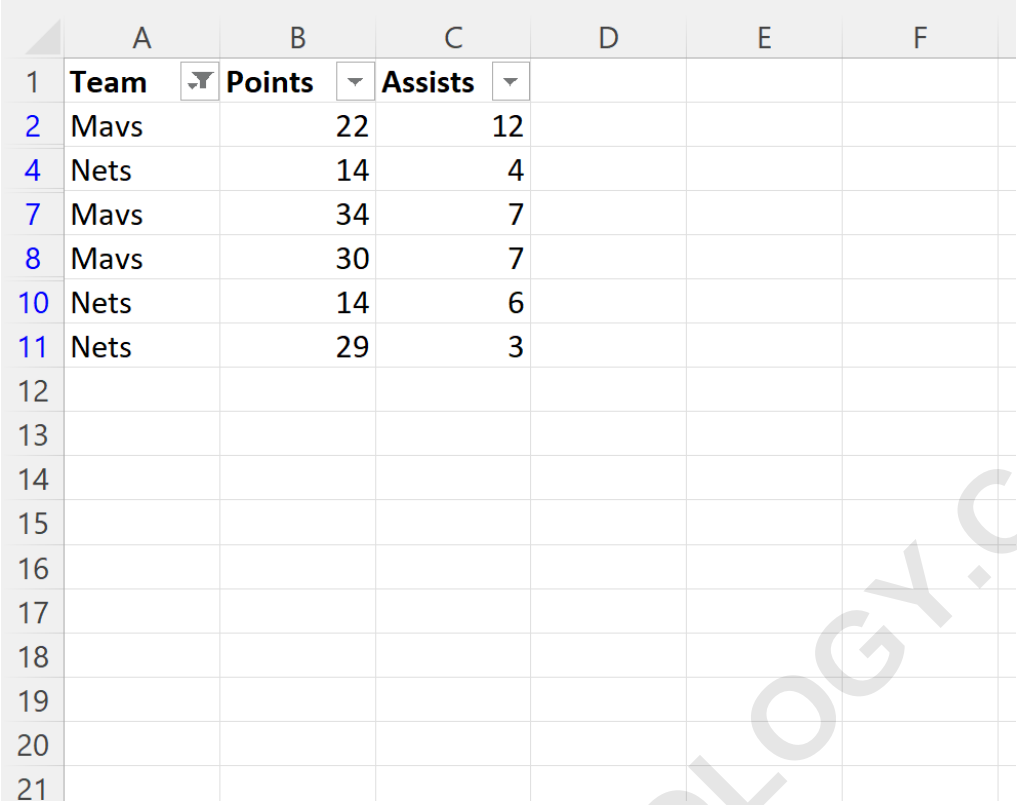
Practical Example: Clearing Filters on a Sample Dataset

The following example shows how to use this **macro** in practice. We will first establish a dataset, apply a specific filter, and then execute the VBA code to restore the dataset to its original, complete state. This demonstration highlights the efficiency and immediate effect of the **ShowAllData** method.

Suppose we have the following dataset in Excel that contains information about various basketball players. This dataset is the initial state before any filtering takes place:

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

Now suppose we apply a filter to isolate specific records. For instance, we might configure the filter in the **Team** column to only display the rows where the value is equal to "Mavs" or "Nets." This action hides several rows that do not meet the specified criteria, leaving a subset of the original data visible:



	A	B	C	D	E	F
1	**Team**	**Points**	**Assists**			
2	Mavs	22	12			
4	Nets	14	4			
7	Mavs	34	7			
8	Mavs	30	7			
10	Nets	14	6			
11	Nets	29	3			
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						

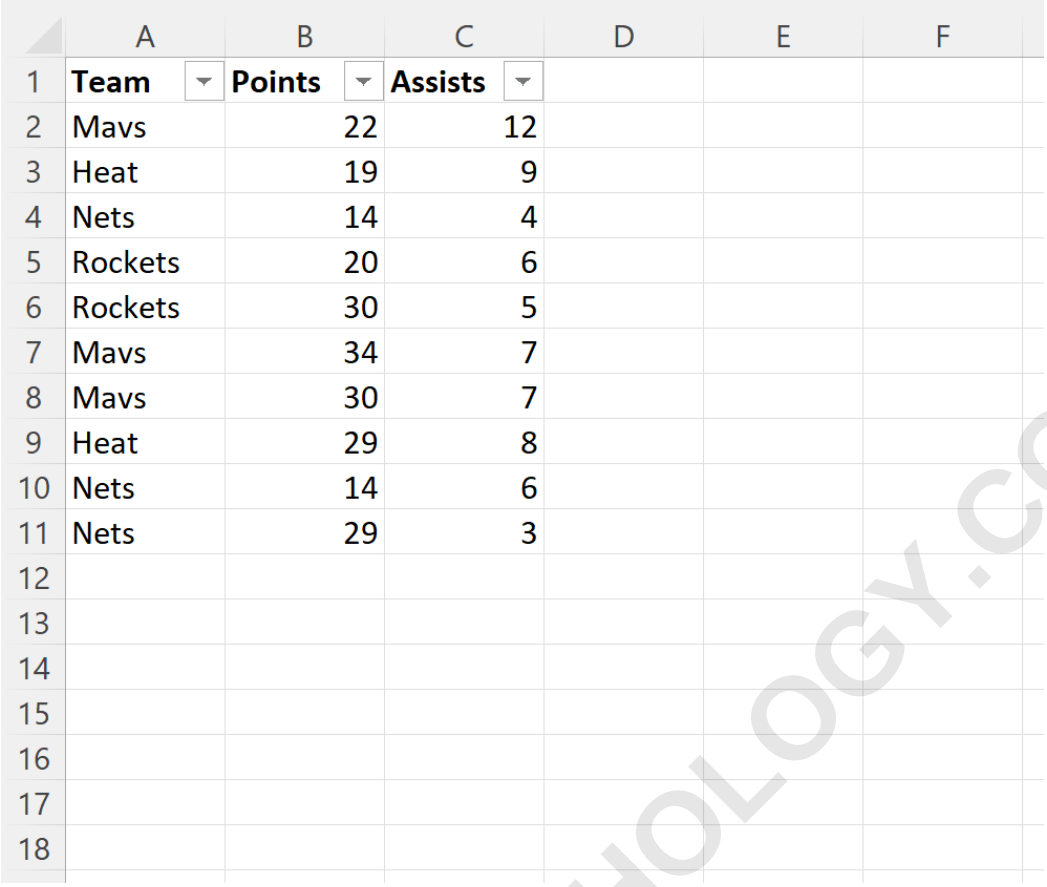
We need to clear this filter using VBA. At this point, the sheet is actively filtered, and the code below is executed to reset the view. We can create the following **macro** to do so:

Sub ClearFilters()

If ActiveSheet.AutoFilterMode Then ActiveSheet.ShowAllData

End Sub

When we run this **macro**, the filter will automatically be cleared from the sheet, and the full dataset is instantly displayed:



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

Notice that all of the rows that were previously hidden are now visible again because we cleared the filter using the **ShowAllData** method. This single command efficiently restored the visibility of the complete dataset, demonstrating the effectiveness of the ShowAllData method.

Advanced Considerations and Best Practices

While the basic **ClearFilters macro** is highly effective for standard data cleaning, advanced applications sometimes require more specific control or error trapping. It is crucial to distinguish between clearing filter criteria and removing the **AutoFilter** functionality entirely.

If the requirement is not only to clear the criteria but also to remove the filter arrows from the column headers completely, an additional line of code must be executed. This is achieved by setting the **AutoFilterMode** property to **False**. If this line is included, the filter arrows disappear, and the sheet is no longer considered filtered by Excel:

Clearing Criteria Only: ShowAllData (Keeps filter arrows active, removes row restrictions).

Removing AutoFilter: ActiveSheet.AutoFilterMode = False (Removes filter arrows and clears all criteria).

Another best practice involves ensuring the **macro** runs even if the active sheet is not the intended target. If your workbook contains multiple sheets and the filter needs to be cleared on a specific sheet (e.g., "DataSheet"), replace **ActiveSheet** with **Worksheets("DataSheet")** throughout the code. This explicit referencing prevents the user from accidentally running the macro on the wrong tab, adding another layer of application robustness and reliability to your VBA projects.