Clear Filters in Excel Using VBA (With Example)

Authored by

Mohammed looti

November 15, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Clear Filters in Excel Using VBA (With Example)*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=1886

Automating Filter Removal Using VBA

For users who frequently work with large datasets in Excel, managing and manipulating filters is a common requirement. While manual clearing is straightforward, repetitive tasks benefit greatly from automation. This article provides an expert guide on how to utilize Visual Basic for Applications (VBA) to efficiently clear all applied filters from an active spreadsheet. This technique ensures consistency and saves valuable time, especially when integrating filter management into larger automated workflows. The core solution relies on a concise macro that checks the current filter status before executing the clear command.

The following syntax represents the most robust and standard way in **VBA** to ensure all filters are removed from the currently active sheet. This short subroutine incorporates essential error prevention logic, which we will detail in the subsequent sections, making it highly reliable for production environments.

Sub ClearFilters()

If ActiveSheet.AutoFilterMode Then ActiveSheet.ShowAllData

End Sub

Deconstructing the ClearFilters Macro

The macro presented above, named `ClearFilters`, is designed for targeted efficiency. It operates exclusively on the **ActiveSheet**--the worksheet currently visible to the user. Its fundamental purpose is to reset the visibility of all rows, ensuring that data previously hidden by filtering criteria are immediately displayed. This process is crucial because simple filtering merely hides rows; it does not delete or alter the underlying data.

The key to this functionality lies in two interconnected components: the conditional statement and the execution method. The line `If ActiveSheet.AutoFilterMode Then` serves as a guard clause. The **AutoFilterMode** property returns a Boolean value, indicating whether the auto filter feature is currently enabled on the sheet. By checking this property, the macro prevents a potential runtime error that could occur if the **ShowAllData** method is called when no filters are active or applied. This practice is standard in professional **VBA** development, promoting stability in the code.

The actual work of clearing the filters is performed by the **ShowAllData** method. This powerful method forces all rows that are currently hidden due to filtering criteria to become visible again. It achieves a complete filter reset, effectively displaying the entire dataset irrespective of any previously set criteria across any columns. If the macro is executed and there are no rows currently being filtered--meaning the **AutoFilterMode** is active but no criteria have been applied, or if the

AutoFilterMode is simply false--then the macro will execute without altering the sheet's appearance, which is the desired non-destructive behavior.

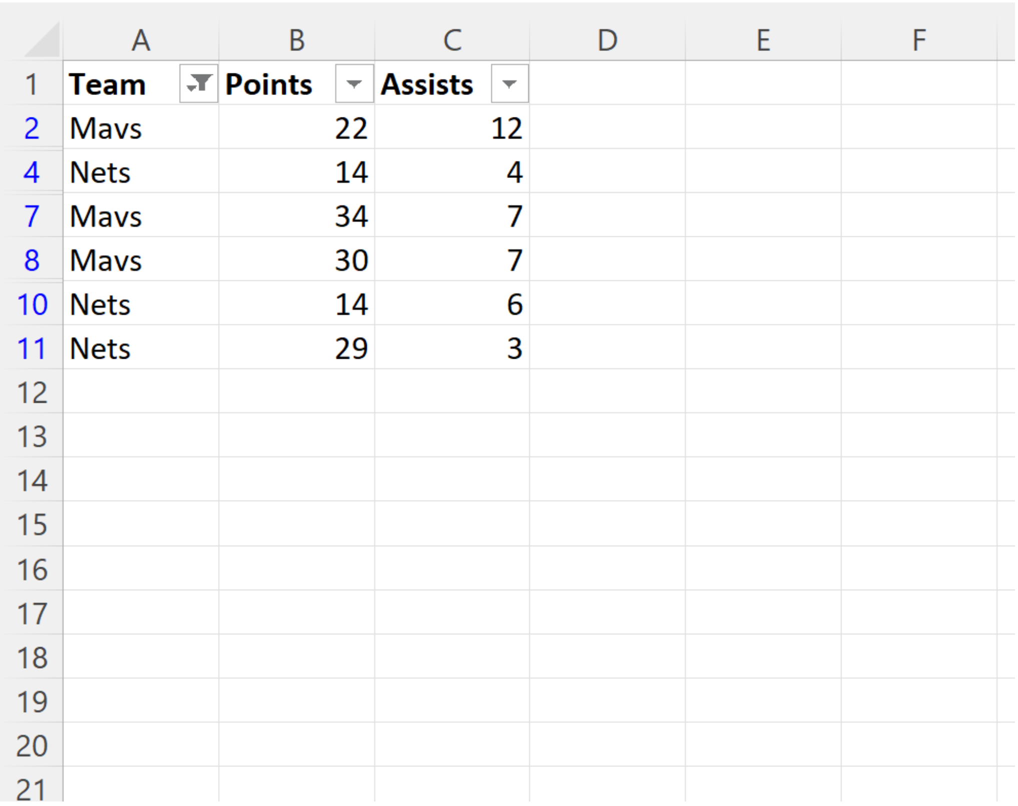
Practical Example: Clear All Filters in Excel Using VBA

To fully illustrate the utility of the `ClearFilters` macro, let us walk through a typical scenario. Suppose we are analyzing performance data for various basketball players. Our initial dataset, laid out in an Excel spreadsheet, contains columns such as Player Name, Team, Points Per Game (PPG), and Rebounds. This raw, unfiltered data looks like the following illustration:

	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						

In the process of data exploration, we often apply filters to focus on specific subsets of information. Now, suppose we apply a filter to the dataset specifically targeting the Team column, intending to display only the rows where the team value is either "Mavs" or "Nets." This action immediately hides all other rows, narrowing our focus to only those players meeting the criteria.

After applying this specific filter criterion, the spreadsheet view is dramatically altered, only showing the filtered rows. This state is visually represented below, highlighting the filtered subset of the data:



	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						

Implementing the VBA Solution

Having analyzed the filtered data, we now wish to revert to the complete, unfiltered dataset without manually clicking the filter options in the ribbon. This is the precise moment when automating the process using **VBA** becomes advantageous. We assume the code is placed within a standard module in the **VBA** editor (accessed via Alt + F11), ready to be executed.

To achieve this goal, we employ the same robust macro defined earlier. When we execute this subroutine, **VBA** first checks the **AutoFilterMode** of the **ActiveSheet**. Since a filter is clearly applied (as shown in the image above), the condition evaluates to true, allowing the **ShowAllData** method to execute.

We create the following macro within the **VBA** environment:

Sub ClearFilters()

If ActiveSheet.AutoFilterMode Then ActiveSheet.ShowAllData

End Sub

Upon running this macro, the action is instantaneous. The filter applied to the Team column is automatically cleared, and the underlying structure of the worksheet is restored to its default state. This immediate clearing demonstrates the efficiency of using **VBA** for filter management.

Verifying the Results and Understanding the Impact

Once the `ClearFilters` macro is executed, the visual representation of the data reverts entirely, confirming that the **ShowAllData** method successfully reset the filtering state. The spreadsheet now displays all original rows, irrespective of the values in the Team column.

The resulting spreadsheet, post-macro execution, will look identical to the initial state shown in the first example image:

	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						

It is important to notice the immediate and complete effect: all of the rows that were previously hidden due to the "Mavs" or "Nets" filtering criteria are now visible again. This successful outcome confirms that the filter was entirely cleared, not just deactivated. Furthermore, the filter dropdown icons at the top of the columns will no longer display the funnel icon, which signifies that specific criteria were applied. This simple **VBA** routine provides a powerful tool for spreadsheet control and data management, ensuring that data integrity and visibility are easily maintained during complex analysis. This specific routine is superior to simply toggling the filter off and on, as it guarantees the display of all data regardless of the previous filtering state.

Advanced Considerations for VBA Filtering

While the `ClearFilters` macro is highly effective for the active sheet, professional **VBA** users might need to manage filters across multiple sheets or workbooks. If the requirement was to clear filters on a sheet named "Summary" instead of the currently active sheet, the code would be modified slightly to reference the specific sheet object, for example: `Worksheets("Summary").ShowAllData`. This illustrates the flexibility of **VBA** objects.

Understanding the **AutoFilterMode** property is also essential for error handling. If we removed the initial `If...Then` statement and ran only `ActiveSheet.ShowAllData` on a sheet where the filter feature hadn't been enabled yet (meaning the filter buttons were not present in the column headers), **Excel** would trigger a runtime error. The conditional check ensures that the **ShowAllData** method is only executed when **Excel** recognizes that an auto filter context exists, thus preventing program interruptions and improving the user experience.

Using such concise and well-structured **VBA** allows developers to integrate filter clearing into larger macros--for instance, running a data refresh process that requires clearing old filters before applying new ones, or ensuring a report always starts from a completely unfiltered state. Mastering this simple code snippet is a foundational step in advanced **Excel** automation using **VBA**.

Additional Resources

To further enhance your proficiency in using VBA for data manipulation and automation in Excel, the following tutorials explain how to perform other common tasks:

How to automate data sorting using VBA.
Techniques for looping through cells and ranges efficiently.
Writing custom functions (UDFs) to extend Excel capabilities.
Managing multiple worksheets and workbooks programmatically.