# How do I copy visible rows to another sheet with VBA?

Authored by

**stats writer**

November 19, 2025

One of the most common tasks faced by users utilizing Microsoft Excel for data management is handling filtered datasets. When you apply an AutoFilter to a large table, Excel efficiently hides rows that do not meet the specified criteria, leaving only the relevant information visible. The challenge arises when attempting to transfer this subset of data to a new location, often another Worksheet, without manually selecting each row. A simple copy-and-paste operation on a filtered range will, by default, include the hidden rows, frustrating the user seeking a clean transfer.

To overcome this limitation and ensure that only the filtered, visible data is copied, we must turn to VBA (Visual Basic for Applications). VBA provides powerful object models and methods necessary to interact with Excel at a granular level. The key to this solution lies in identifying and selecting only the cells that remain visible after filtering. This process requires a sophisticated understanding of the Range.SpecialCells method, which is specifically designed to handle non-contiguous or unique cell types within a specified range.

This comprehensive guide will walk you through the precise VBA code required to execute this task flawlessly. We will dissect the necessary components, explain the function of critical constants like xlCellTypeVisible, and provide a practical, illustrative example. By implementing this robust automation script, you can dramatically improve the efficiency of your data extraction workflows and ensure the integrity of your transferred data.

## Understanding the Core VBA Mechanism: SpecialCells

The solution hinges entirely on the use of the Range.SpecialCells method. When working with Excel ranges, you often deal with contiguous blocks of cells. However, when a filter is applied, the visible cells become non-contiguous--they are interrupted by hidden rows. Standard selection methods fail in this scenario because they treat the entire range as a single block, inadvertently including the hidden data.

The Range.SpecialCells method allows developers to select cells based on specific characteristics or types. It accepts arguments that define which type of special cell should be returned. For our purpose of copying filtered data, we must use the constant xlCellTypeVisible. When this constant is passed as the argument to the method, Excel processes the specified range and returns a new Range object comprising only those cells that are currently displayed on the Worksheet, effectively bypassing all hidden rows and columns.

Once this specialized Range object--containing only the visible cells--has been identified, the subsequent step is straightforward: applying the standard `.Copy` method. This streamlined approach ensures that only the filtered data is placed onto the clipboard, ready for transfer. This method is far superior to attempting complex looping mechanisms to check the hidden status of each individual row, offering both performance advantages and code simplicity.

## Detailed Syntax Breakdown: The CopyVisibleRows Macro

To successfully copy visible rows, the VBA code must manage several critical steps: defining the source and destination sheets, setting the range, identifying the special cells, and executing the copy and paste operations. Below is the standard, efficient macro structure designed for this purpose. We utilize object variables to make the code highly readable and maintainable.

You can use the following syntax in VBA to copy only the visible rows from one sheet to another:

```
Sub CopyVisibleRows()

Dim sourceWS As Worksheet
Dim destinationWS As Worksheet

Set sourceWS = ThisWorkbook.Sheets("Sheet1")
Set destinationWS = ThisWorkbook.Sheets("Sheet2")

sourceWS.Range("A1:D999").SpecialCells(xlCellTypeVisible).Copy
destinationWS.Cells(1, 1).PasteSpecial
Application.CutCopyMode = False

End Sub
```

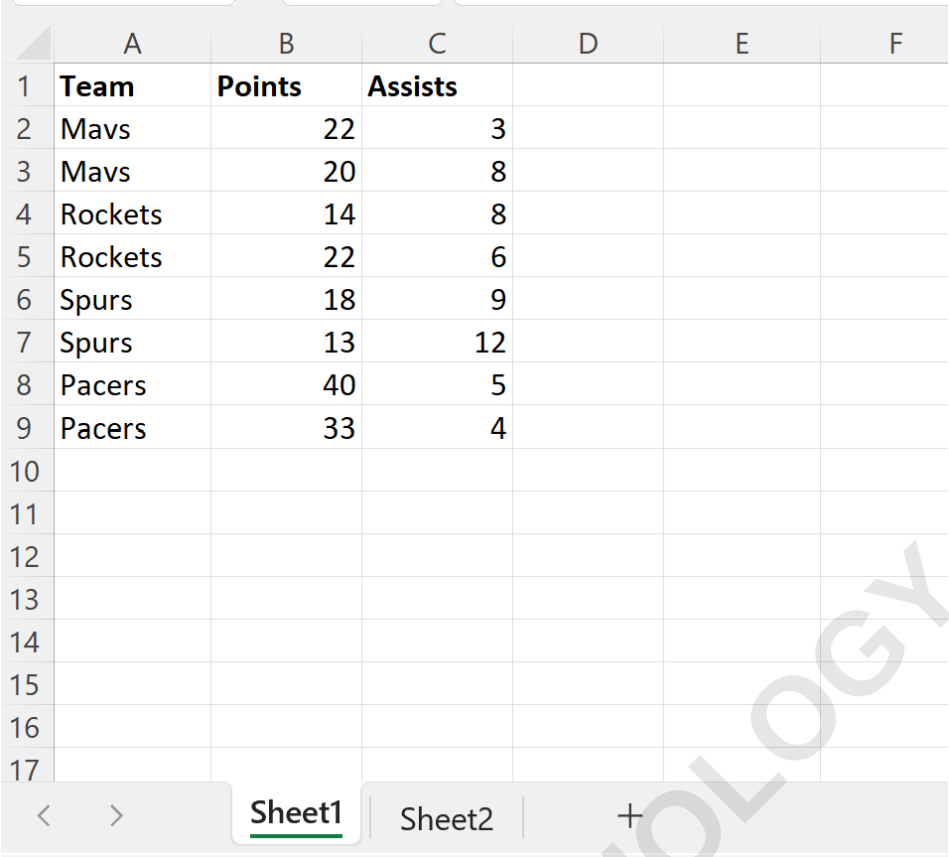
This particular macro is configured to copy every visible row within the range **A1:D999** located on the Worksheet named **Sheet1**. The copied data is then pasted, starting at cell **A1**, onto the destination Worksheet named **Sheet2**. Note that the hardcoded range `A1:D999` should be adapted to match the true scope of your dataset.

It is best practice to include the line `Application.CutCopyMode = False`. This command explicitly specifies that the cut and copy selection border (the "marching ants") should be deactivated after the macro finishes its execution. This step ensures a clean exit from the routine and prevents accidental further pasting or unintended operations based on the clipboard contents.

## Step-by-Step Implementation Example

Let us apply this powerful technique using a practical scenario involving a dataset of basketball players. This example demonstrates how to filter the data first and then use the VBA macro to isolate and copy only the results that meet the AutoFilter criteria. The following steps will guide you through the process, from initial setup to final output.

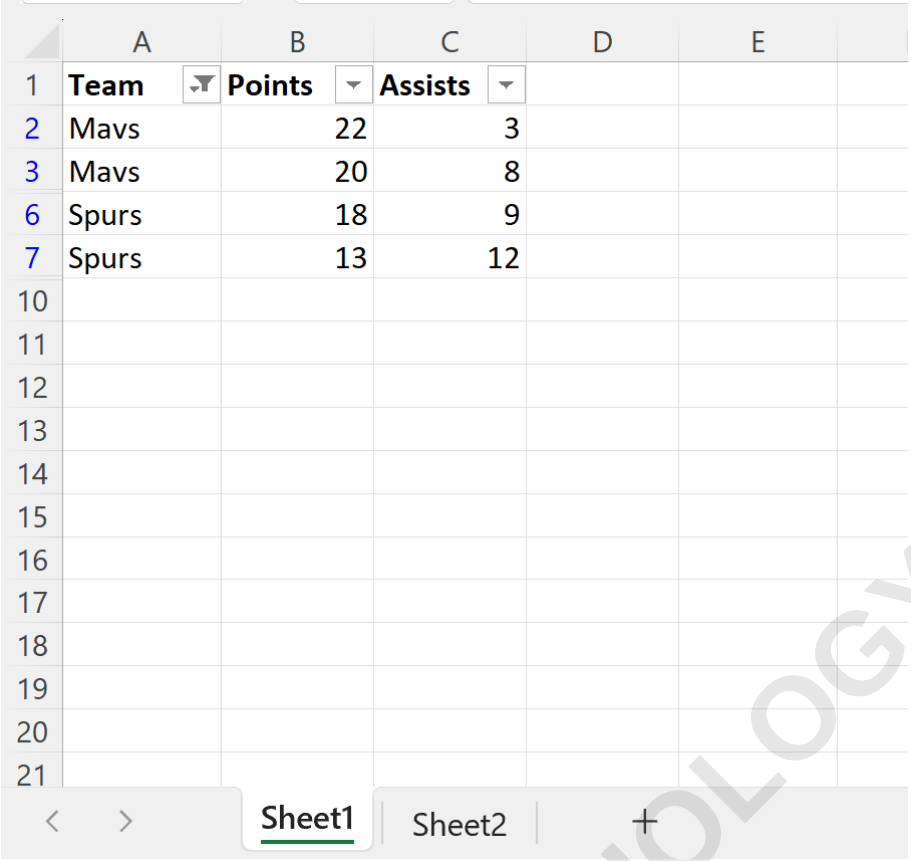
Suppose we have the following dataset in **Sheet1** that contains information about various basketball players:



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | **Assists** | | | |
| 2 | Mavs | 22 | 3 | | | |
| 3 | Mavs | 20 | 8 | | | |
| 4 | Rockets | 14 | 8 | | | |
| 5 | Rockets | 22 | 6 | | | |
| 6 | Spurs | 18 | 9 | | | |
| 7 | Spurs | 13 | 12 | | | |
| 8 | Pacers | 40 | 5 | | | |
| 9 | Pacers | 33 | 4 | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |

Sheet1 Sheet2

Now suppose we apply a filter to the dataset to only show the rows where the team name is equal to Mavs or Spurs:



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Team** | **Points** | **Assists** | | |
| 2 | Mavs | 22 | 3 | | |
| 3 | Mavs | 20 | 8 | | |
| 6 | Spurs | 18 | 9 | | |
| 7 | Spurs | 13 | 12 | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |

Sheet1 Sheet2 +

We now employ the VBA macro to transfer the filtered results to **Sheet2**. By targeting the entire data range and using `.SpecialCells(xlCellTypeVisible)`, we guarantee that only the rows visible in the image above are processed and copied. We create and run the following macro:

**Sub CopyVisibleRows()**

Dim sourceWS As Worksheet
Dim destinationWS As Worksheet

Set sourceWS = ThisWorkbook.Sheets("Sheet1")
Set destinationWS = ThisWorkbook.Sheets("Sheet2")

sourceWS.Range("A1:D999").SpecialCells(xlCellTypeVisible).Copy
destinationWS.Cells(1, 1).PasteSpecial
Application.CutCopyMode = False

End Sub

## Analyzing the Code and Key Properties

Understanding the internal workings of the code is vital for debugging and adapting it to future needs. The power of this solution is concentrated in a single line of code, but the setup ensures its reliability. Let's break down the key components used in the script.

**Worksheet Object Declaration:** The first few lines use `Dim` and `Set` statements to assign variables (`sourceWS` and `destinationWS`) to specific Worksheet objects. This practice is preferred over repeatedly calling `Sheets("Name")` as it improves both execution speed and code clarity.

**The Range.SpecialCells Method:** The core of the operation is `sourceWS.Range("A1:D999").SpecialCells(xlCellTypeVisible)`. This expression takes the specified range (A1:D999) and filters it down to only include the cells currently visible on the screen. Because the range is filtered using AutoFilter before the macro runs, the resulting special range contains only the desired data rows.

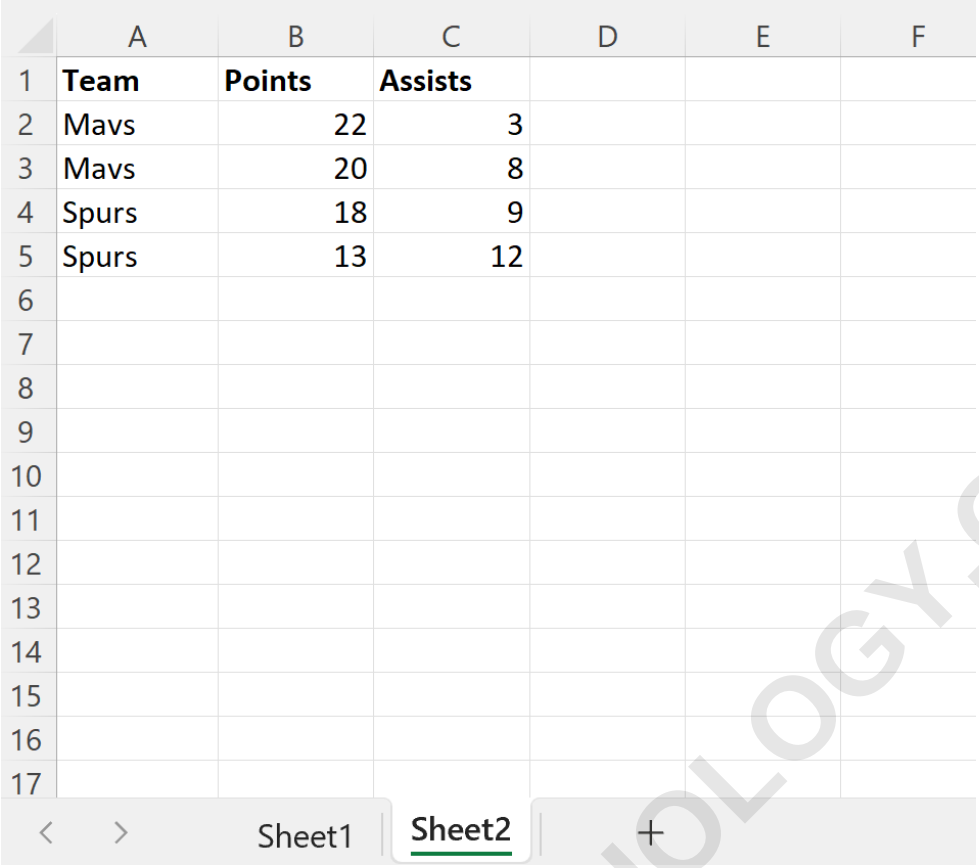
**The xlCellTypeVisible Constant:** This built-in VBA constant is the essential parameter for `SpecialCells`. It instructs Excel to identify and return only those cells that have not been hidden by manual hiding operations, filtering, or grouping/outlining features. Without this specific constant, the copy operation would default back to selecting the entire contiguous range, including all hidden rows.

**`.PasteSpecial`:** While a simple `.Paste` might work in many scenarios, using `.PasteSpecial` provides control over how the data is pasted. By omitting specific arguments (like `Paste:=xlPasteValues`), it performs a default paste, transferring both values and formatting while remaining resilient in different environments.

## Result Verification and Output

Upon successfully running the `CopyVisibleRows` macro, the system copies the filtered range from **Sheet1** and pastes the results starting in the top-left cell of **Sheet2**. This process yields a completely clean dataset, containing only the records that matched the filter criteria established beforehand. Hidden rows are successfully excluded from the transfer operation.

When we run this macro, we receive the following output in **Sheet2**:



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Team** | **Points** | **Assists** | | | |
| 2 | Mavs | 22 | 3 | | | |
| 3 | Mavs | 20 | 8 | | | |
| 4 | Spurs | 18 | 9 | | | |
| 5 | Spurs | 13 | 12 | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |

Sheet1 Sheet2 +

Notice that each row that was visible in **Sheet1** has been copy and pasted into **Sheet2**. The row numbers are reset on the new sheet, starting from row 1, confirming the successful isolation and transfer of the visible data subset.

## Handling Edge Cases and Best Practices

While the basic macro is robust, an expert implementation should account for potential edge cases and incorporate best practices to ensure longevity and error handling.

**Handling Headers:** If your filter includes header rows (which should always be the case for proper filtering), those headers will be considered visible cells and will be copied. If you need to copy only the body data (excluding the header), you must adjust the source range accordingly. For example, if A1 is the header, change the source range to `A2:D999`.

**Empty Filters:** A common failure point is when the AutoFilter returns zero visible rows. If the filtered range results in no visible cells other than possibly the header, attempting to apply `.SpecialCells(xlCellTypeVisible).Copy` will trigger a runtime error (Run-time error '1004': No cells were found). Robust code must include error trapping (using `On Error Resume Next` before the copy line, and `On Error GoTo 0` after it) or check if the filter returned data using the

`Range.Areas.Count` property.

**Dynamic Range Selection:** Hardcoding a large range like `A1:D999` is acceptable but often inefficient. For maximum flexibility, use dynamic range detection (e.g., `sourceWS.UsedRange` or determining the last row/column) to ensure the macro works regardless of the dataset size. For instance, finding the last row (`lRow = sourceWS.Cells(sourceWS.Rows.Count, "A").End(xlUp).Row`) allows you to define the range dynamically (`sourceWS.Range("A1:D" & lRow)`).

## Conclusion and Advanced Data Handling

The method of using Range.SpecialCells combined with the xlCellTypeVisible constant represents the definitive and most efficient approach for copying only filtered data in Excel using VBA. This technique avoids the pitfalls of manual selection or tedious row-by-row checks, delivering immediate, clean results.

Mastering this simple but powerful macro enhances your ability to automate complex data preparation tasks. Whether you are generating reports, extracting specific subsets for analysis, or preparing data for external applications, the ability to selectively copy visible rows is an invaluable tool in your automation repertoire. Always remember to test your macros thoroughly, especially when dealing with production data, and consider dynamic range specification for universal applicability across different datasets.