How to Count Cells by Color in Excel Using VBA: A Step-by-Step Tutorial

Authored by

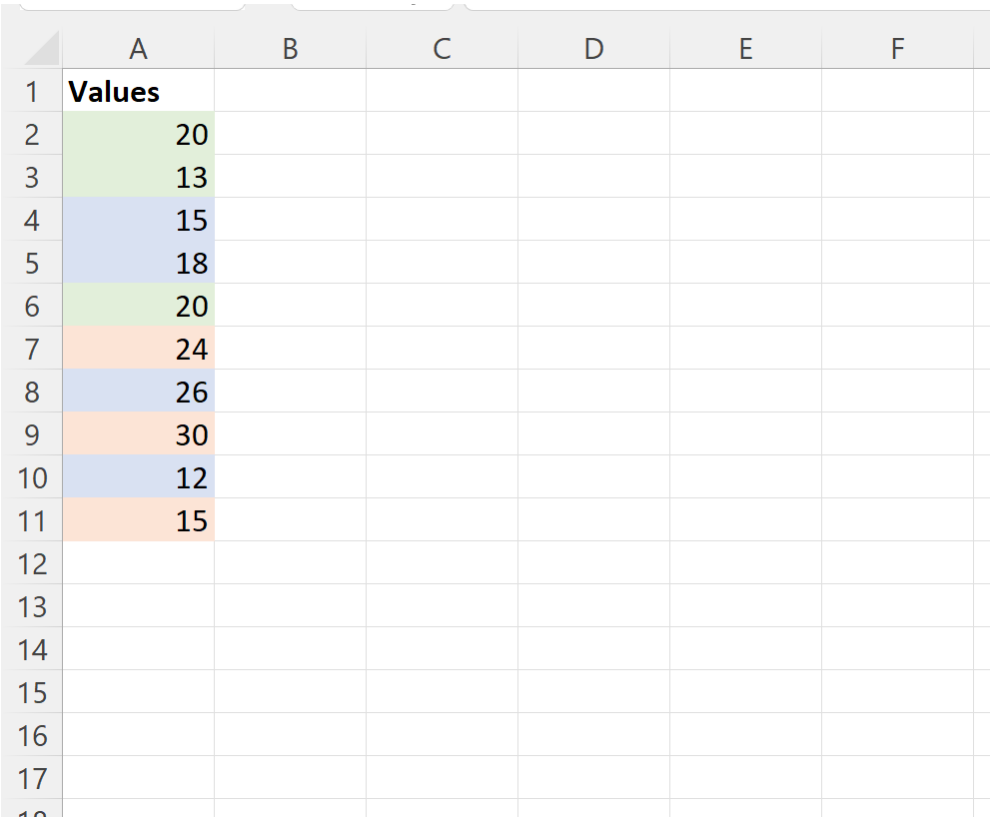
Mohammed looti

November 10, 2025

In advanced data analysis using Excel, a frequent requirement is the ability to count the number of cells based on specific formatting attributes, such as their background color. Standard Excel functions, like COUNTIF, are primarily designed to handle numerical or text-based criteria, making them ineffective for criteria based purely on visual formatting.

To successfully address this limitation and achieve dynamic color-based counting, we must leverage the power of VBA (Visual Basic for Applications). This tutorial provides a comprehensive, step-by-step guide on creating a custom user-defined function (UDF) that efficiently counts cells based on a reference color, ensuring accurate results even for large, color-coded datasets.

For example, consider the following sample dataset where we need to quickly tally the frequency of items categorized by their visual color coding:



	A	B	C	D	E	F
1	**Values**					
2	20					
3	13					
4	15					
5	18					
6	20					
7	24					
8	26					
9	30					
10	12					
11	15					
12						
13						
14						
15						
16						
17						

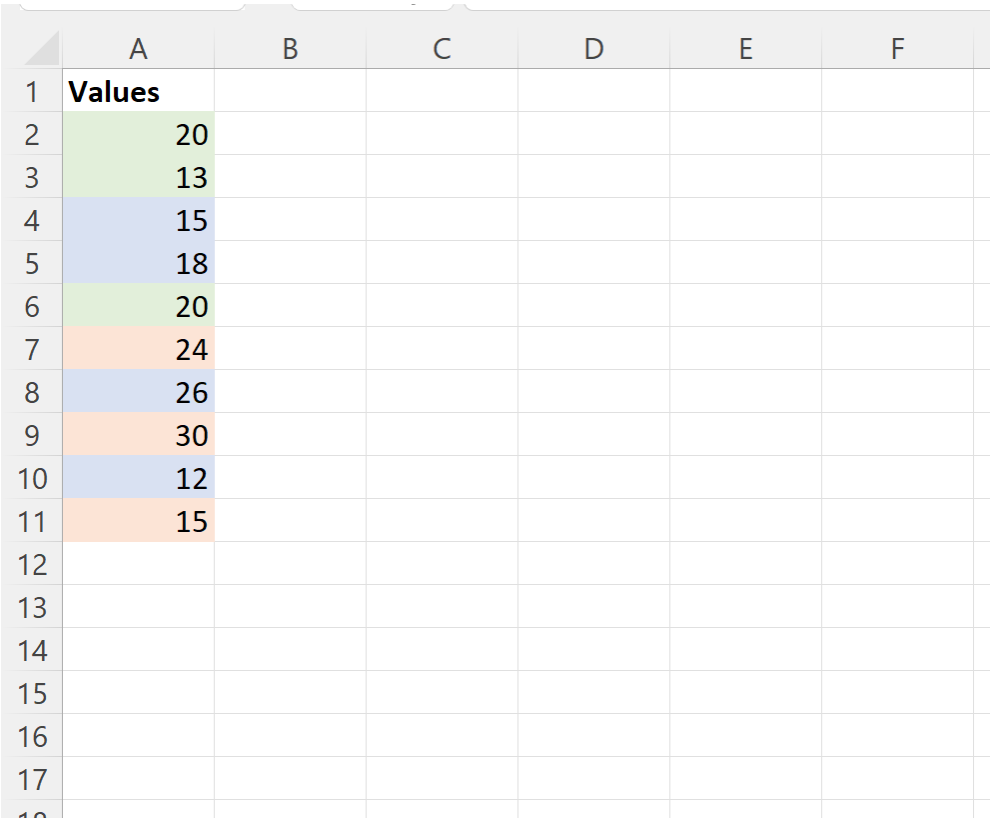
While implementing a custom Macro might initially seem challenging, especially for users new to VBA programming, the process outlined below is surprisingly direct and highly effective. We will walk through the exact steps required to transform this manual task into an automated, formula-driven solution.

Step 1: Preparing the Dataset

The foundational step involves preparing the necessary data values and ensuring proper

formatting within your Excel worksheet. It is critical that the cells you intend to count are already formatted with the desired background colors, as this visual coloring serves as the core criterion for our custom counting function.

As illustrated in the image below, our primary data range (Column A) contains various numerical values, each assigned a specific background color representing a category or status. This preparation allows us to move forward with the programming stage.



	A	B	C	D	E	F
1	**Values**					
2	20					
3	13					
4	15					
5	18					
6	20					
7	24					
8	26					
9	30					
10	12					
11	15					
12						
13						
14						
15						
16						
17						

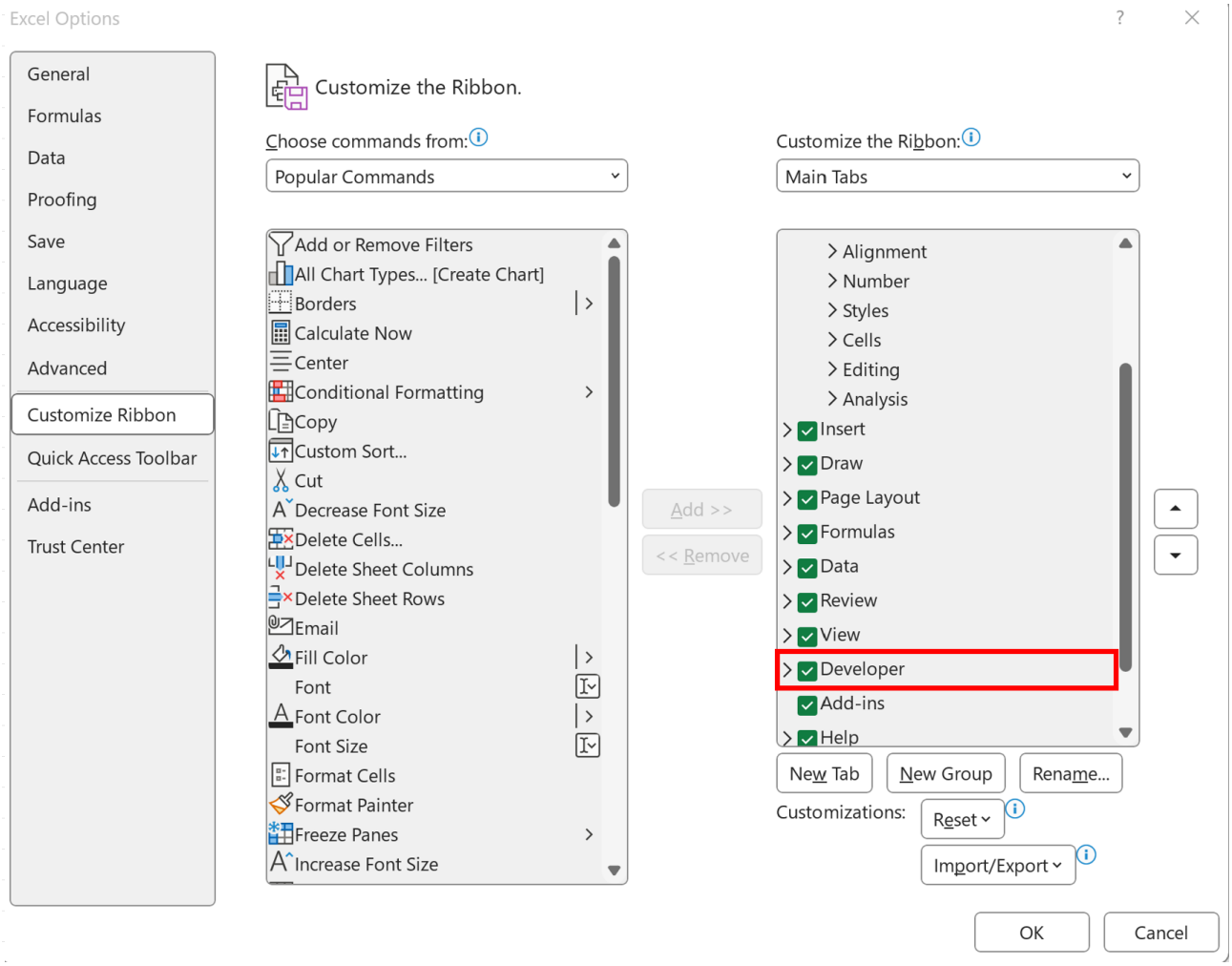
Once the data is correctly entered and formatted, we can proceed to enable the tools required to introduce custom functionality into the workbook.

Step 2: Enabling the Developer Tab

Before we can write any VBA code, we must ensure that the **Developer** tab is visible within the Excel top ribbon. This tab contains all the essential tools required for creating, editing, and managing Macros and other custom application components. If the **Developer** tab is already visible, you may skip this step and proceed directly to Step 3.

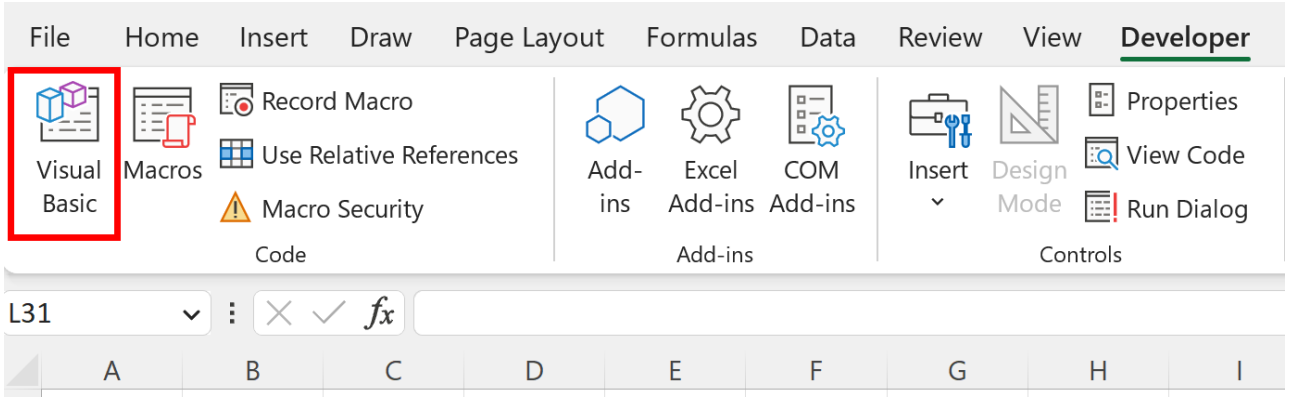
To activate the **Developer** tab, navigate through the Excel menu by clicking the **File** tab, selecting **Options**, and then choosing the **Customize Ribbon** category from the left-hand panel.

Within the **Customize Ribbon** settings, locate the list of tabs under the section labeled **Main Tabs**. Scroll down, ensure the checkbox next to **Developer** is checked, and finalize the changes by clicking **OK**. This action immediately adds the necessary development tools to your main worksheet interface.

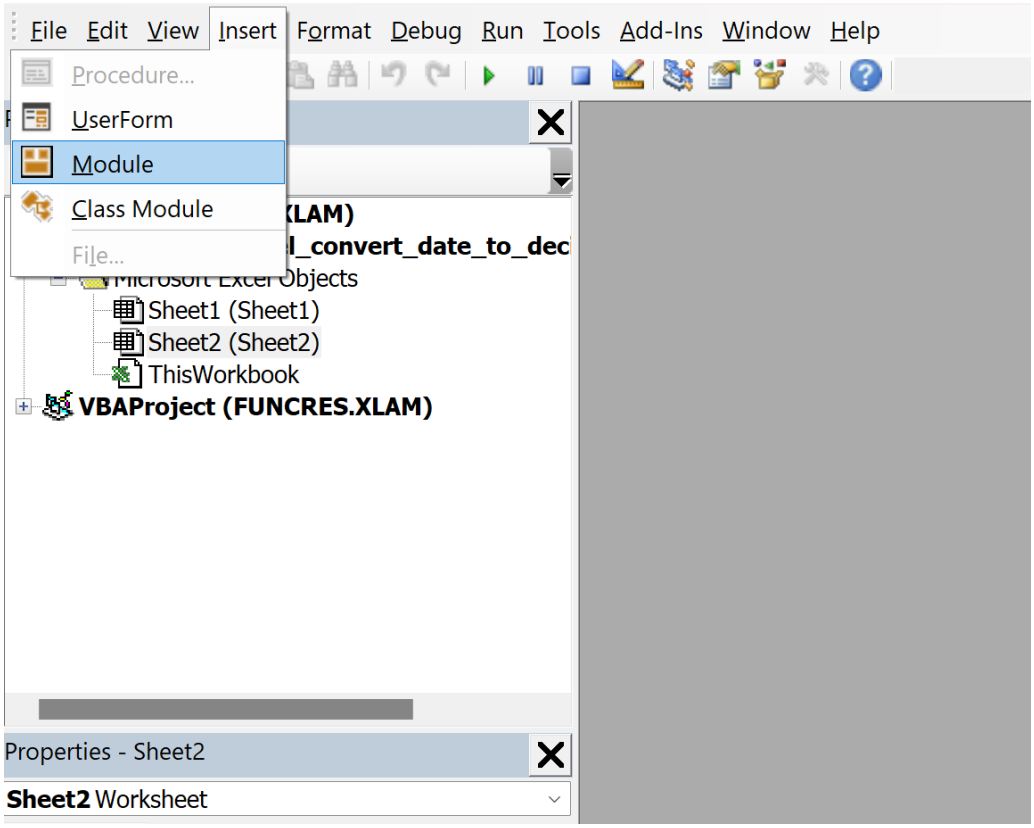


Step 3: Implementing the Custom VBA Function

With the **Developer** tab now active, the next crucial step is to access the Visual Basic Editor (VBE), which is the environment where we will input and manage the custom function's code. Click the **Developer** tab on the top ribbon, and then select the **Visual Basic** icon, typically found on the far left of the tab.



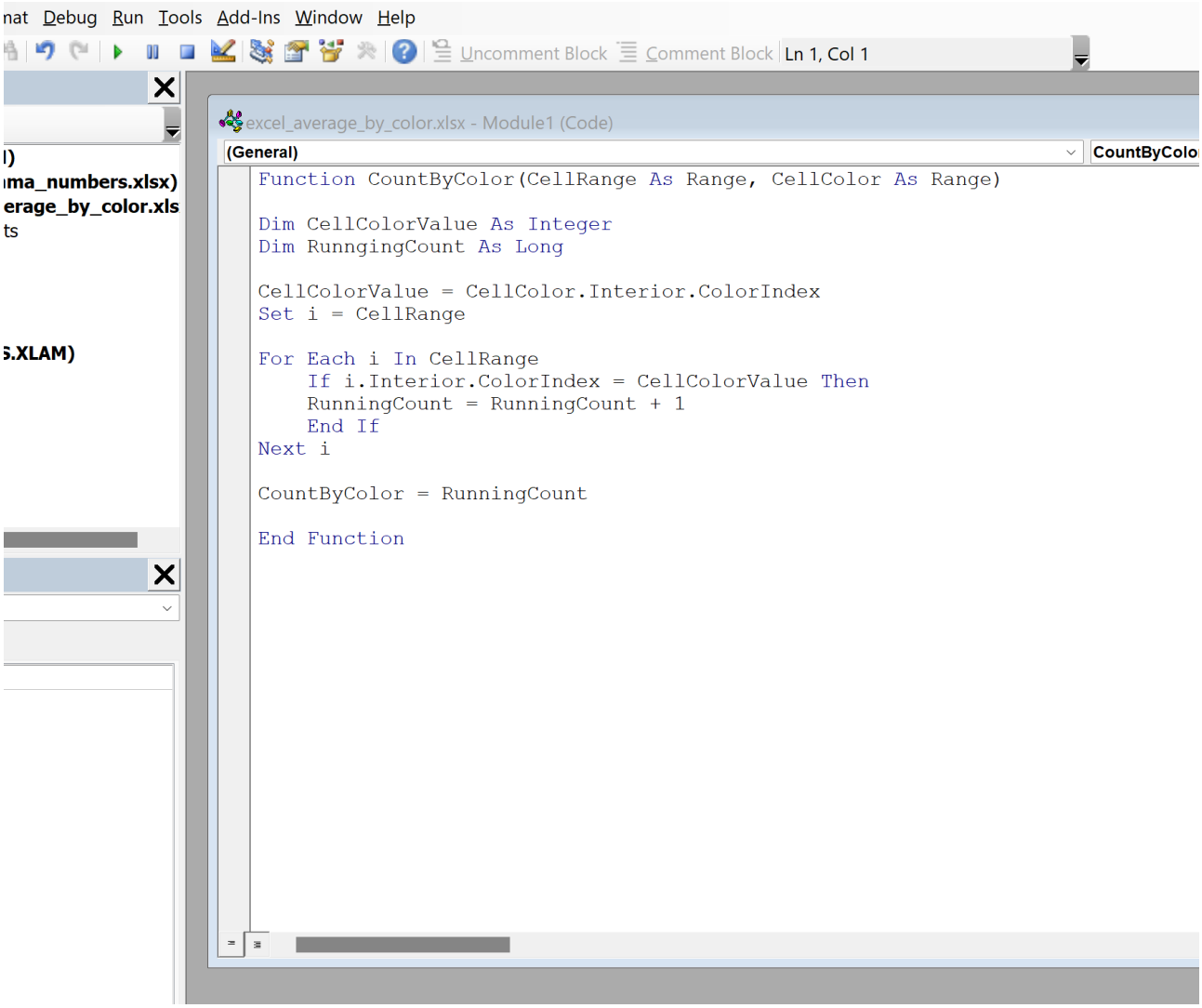
Inside the VBE, we must insert a new standard **Module** to house our code, ensuring it is globally accessible within the workbook. Go to the **Insert** tab in the VBE menu and click **Module** from the subsequent dropdown list. A new, blank code window will appear in the main VBE pane, ready for programming input.



Carefully paste the following VBA code into the newly created module. This code defines a custom function named **CountByColor**, which is designed to iterate through a specified range of cells. It uses the ColorIndex property to compare the background color of each cell in the input range against the color of a single reference cell, incrementing a counter upon every successful match.

```
Function CountByColor(CellRange As Range, CellColor As Range)

Dim CellColorValue As Integer
Dim RunngingCount As Long

CellColorValue = CellColor.Interior.ColorIndex
Set i = CellRange

For Each i In CellRange
If i.Interior.ColorIndex = CellColorValue Then
RunningCount = RunningCount + 1
End If
Next i

CountByColor = RunningCount

End Function
```

This screenshot illustrates the correct placement of the code within the VBE module window, ensuring that the function is properly compiled and recognized by Excel:



Once the code is successfully entered, close the Visual Basic Editor. The custom function is now saved and ready to be used just like any standard, native Excel formula directly within the worksheet cells.

Step 4: Executing the Custom Function in the Worksheet

The final step involves applying the newly created Macro function, **CountByColor**, to calculate the necessary counts based on the visual criteria. To facilitate this, we need to set up a column dedicated to defining the specific colors we want the function to target.

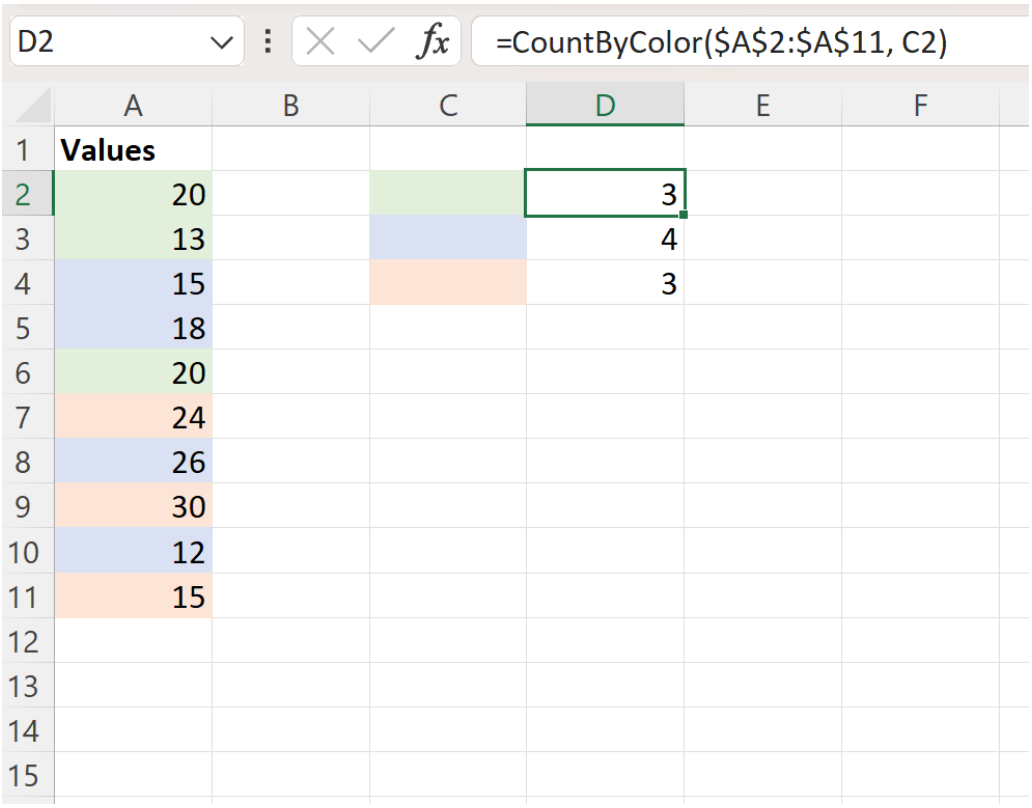
Start by formatting cells **C2:C4** with the exact background colors corresponding to the categories in your dataset (Column A). These cells will serve as the essential color references for the function, telling it precisely which color index to search for.

Next, in cell **D2**, input the following formula. This structure instructs Excel to look through the

absolute range **$A$2:$A$11** and count how many cells match the color formatting found in the reference cell **C2**:

=CountByColor($A$2:$A$11, C2)

By utilizing the relative cell reference **C2** for the color reference, you can efficiently drag this formula down to the remaining cells in Column D (D3 and D4). The function will automatically adjust to reference **C3** and **C4**, providing an immediate and accurate count for each distinct color category in the dataset.



Based on the calculated results, we can confirm the frequency of cells categorized by color:

The total count of cells with a light green background is **3**.
The total count of cells with a light blue background is **4**.
The total count of cells with a light orange background is **3**.

Important Consideration: If the reference cell provided in Column C contains a color that does not exist anywhere within the defined lookup range (A2:A11), the custom function will correctly execute the iteration process but will return a final numerical value of 0, accurately indicating that no cells matched the specified color criteria.

Further Excel and VBA Resources

If you are interested in exploring more advanced conditional formatting techniques, custom user-defined functions, or other automation features within Excel, the following related tutorials offer explanations on how to perform other common statistical and data manipulation operations: