

How to Easily Create a Bar Chart in VBA

Authored by
stats writer

November 20, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Create a Bar Chart in VBA*. PSYCHOLOGICAL SCALES.
Retrieved from <https://scales.arabpsychology.com/?p=97998>

The ability to generate visualizations automatically is a cornerstone of advanced data processing in Microsoft Excel. Creating a bar chart programmatically using Visual Basic for Applications (VBA) allows users to streamline repetitive reporting tasks, ensuring consistency and efficiency across various datasets. This process moves beyond manual chart creation by leveraging VBA's extensive object model to control every facet of the visualization. To successfully implement this automation, a developer must meticulously define the data source, specify the desired chart type (in this case, a column or bar chart), establish the chart's placement, and apply necessary formatting, such as titles and axis labels.

The fundamental sequence for generating a chart via VBA requires interaction with several key objects within the Excel hierarchy. Initially, the relevant data series must be identified, typically using a Range object to reference the input data on a worksheet. Subsequently, the code instantiates a new chart container, known as the ChartObject, before applying the data source and defining the chart's visual style. While advanced formatting, such as customizing colors or adding data labels, can be implemented later, the core function focuses on rapidly displaying meaningful graphical representations of underlying numerical information. The successful deployment of such a macro leads to immense time savings in data reporting cycles.

Establishing the Essential VBA Syntax

Automating chart creation relies on a precise sequence of commands within a VBA subroutine. The following basic syntax provides a robust foundation for generating a clustered column chart--which is often referred to interchangeably with a bar chart in Excel's object model--by dynamically prompting the user for the data range required for visualization. This structure ensures that the macro is flexible and reusable, minimizing the need for hardcoding specific cell references into the script.

This macro template utilizes the Application.InputBox method, specifying the Type parameter as 8, which instructs the function to return a **Range** object selected directly by the user on the worksheet. Defining the input range dynamically is crucial for creating macros that adapt to varying data sizes and locations. Once the range is selected, the script proceeds to instantiate and configure the new chart object, positioning it relative to the active selection on the spreadsheet.

Sub CreateBarChart()

```
Dim MyChart As ChartObject
```

```
'get input range from user
```

```
Set Rng = Application.InputBox(Prompt:="Select chart input range", Type:=8)
```

```
'create bar chart
```

```
Set MyChart = Worksheets("Sheet1").ChartObjects.Add(Left:=ActiveCell.Left, _  
Width:=400, Top:=ActiveCell.Top, Height:=300)  
MyChart.Chart.SetSourceData Source:=Rng  
MyChart.Chart.ChartType = xlColumnClustered  
  
End Sub
```

The preceding macro, named `CreateBarChart`, is engineered to handle the entire chart generation workflow automatically. Upon execution, it first utilizes the `Application.InputBox` dialog box to acquire the user-specified input range. Following the data selection, the script proceeds to initialize and position the chart. Specifically, it uses the `ChartObjects.Add` method to embed the new chart directly onto the worksheet named **Sheet1**. Crucially, the location of this new chart is anchored to the coordinates of the **currently active cell**, ensuring that the chart appears immediately adjacent to the user's cursor.

The two critical lines for defining the chart's content and appearance are detailed below:

`MyChart.Chart.SetSourceData Source:=Rng`: This command assigns the collected data range (Rng) to the newly created chart object, dictating which numerical data and category labels will be visualized.

`MyChart.Chart.ChartType = xlColumnClustered`: This line explicitly defines the chart's structure as a **Clustered Column** chart. This type is generally used in Excel when vertical bars are required for comparison between distinct categories.

The initial dimensions are set using **Width** (400 points) and **Height** (300 points), which developers can easily adjust based on their reporting requirements.

Practical Example: Visualizing Performance Data with VBA

To solidify the theoretical understanding of the macro structure, this section provides a practical, step-by-step example demonstrating how to implement and run this VBA code successfully within an Excel workbook. This walk-through utilizes a typical small dataset, illustrating the immediate graphical output generated by the automated script. We assume the user has data ready and the Developer tab is enabled.

Consider a scenario where we maintain a simple dataset tracking points scored by several basketball players over a particular period. This data is structured in two adjacent columns: one containing the player names (categories) and the other containing the corresponding point totals (values). This columnar arrangement is ideal for a column or bar chart visualization, allowing for easy comparison of performance metrics among individuals.

The sample data used for this demonstration is structured as follows, spanning cells A1 through B7 on a worksheet:

	A	B	C	D	E	F
1	Team	Points				
2	Mavs	22				
3	Nets	40				
4	Spurs	23				
5	Lakers	28				
6	Rockets	25				
7	Heat	18				
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						

Our objective is to employ the previously defined VBA macro to quickly transform this raw data into a dynamic graphical representation. This approach eliminates the need to navigate through the 'Insert Chart' menu manually, providing immediate visualization capabilities directly within the spreadsheet environment, particularly useful when generating many similar charts.

Inserting and Running the Macro

To execute this automation, the provided code must be placed within a standard module in the Visual Basic Editor (VBE). Accessing the VBE is typically done by pressing **Alt + F11** in Excel. Once in the editor, developers should navigate to the 'Insert' menu, select 'Module', and paste the full subroutine code into the newly opened module window. This setup ensures that the macro is correctly defined and available for execution across the entire workbook.

The full script block, which should reside in a VBA Module (e.g., `Module1`), is shown below:

Sub CreateBarChart()

```
Dim MyChart As ChartObject
```

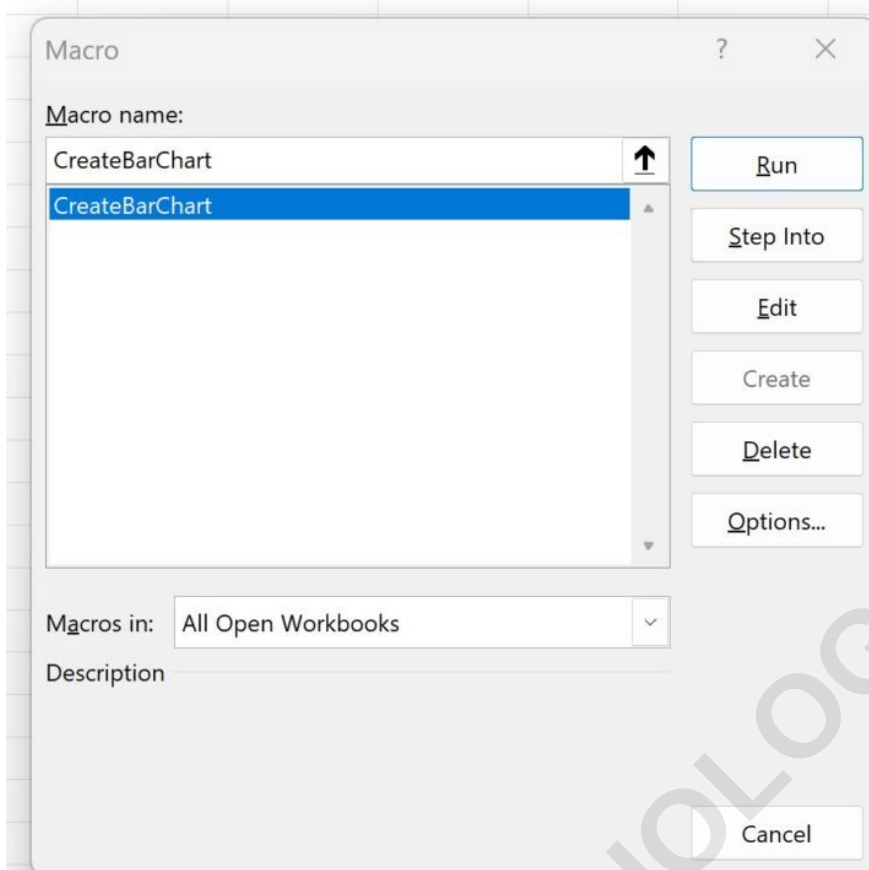
```
'get input range from user
Set Rng = Application.InputBox(Prompt:="Select chart input range", Type:=8)

'create bar chart
Set MyChart = Worksheets("Sheet1").ChartObjects.Add(Left:=ActiveCell.Left, _
Width:=400, Top:=ActiveCell.Top, Height:=300)
MyChart.Chart.SetSourceData Source:=Rng
MyChart.Chart.ChartType = xlColumnClustered

End Sub
```

Once the code is correctly stored in a module, executing the macro requires accessing the Excel user interface. This is most efficiently done through the **Developer** tab on the Excel ribbon. Within the **Developer** tab, locate and click the **Macros** button (or press Alt + F8). This action opens the Macro dialog box, listing all available subroutines in the workbook.

Within the Macro dialog box, select the macro named **CreateBarChart**. After selecting the name, clicking **Run** initiates the execution sequence defined in the VBA script. The interface for selecting and running the macro is visualized below, confirming the standard procedure for macro deployment in Excel:

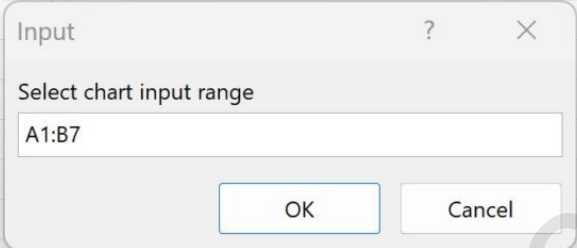


Defining the Data Range and Generating Output

Immediate subsequent to clicking **Run**, the VBA interpreter executes the line that calls `Application.InputBox`. This command pauses the script and displays a specialized dialog box titled 'Select chart input range,' which prompts the user to physically select the desired data range directly on the worksheet. This step is critical as it passes the necessary Range reference to the subsequent charting commands.

The prompt requires careful selection of the data, including both the category labels (Player names) and the corresponding numerical series (Points). The input box appearance confirming the selection request is shown here, allowing for direct cell selection via the mouse:

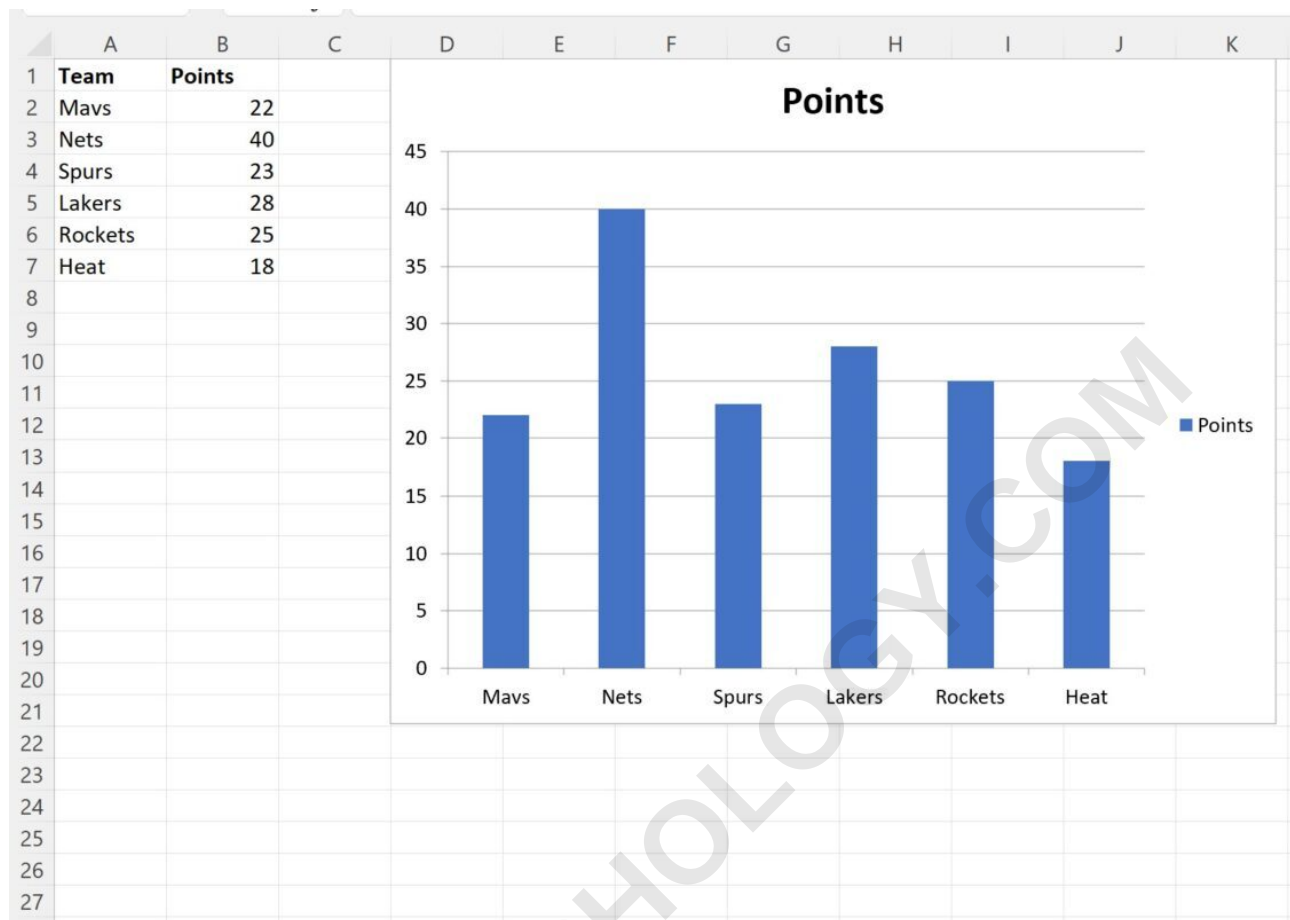
	A	B	C	D	E	F	G
1	Team	Points					
2	Mavs	22					
3	Nets	40					
4	Spurs	23					
5	Lakers	28					
6	Rockets	25					
7	Heat	18					
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							



For our specific dataset, which spans from the header row in cell A1 down to the last data entry in B7, the required input range is **A1:B7**. The user should either type this range string into the box or, more efficiently, click and drag the mouse over the cells A1 through B7. Clicking **OK** confirms the selection and allows the remainder of the macro to execute instantly, creating the chart object.

Final Chart Placement and Review

The final steps involve the script utilizing the `ChartObjects.Add` method, which defines the position and size, followed by `SetSourceData` and `ChartType` definitions. This instantaneous process embeds the newly configured visualization onto the specified sheet (**Sheet1**). As specified in the code, the chart's top-left corner is positioned relative to the `ActiveCell`. If cell **D1** was the active cell when the macro was initiated, the resulting visualization will be neatly inserted starting at that location, showcasing the scores in a clear clustered column format:



The resulting chart is a fully functional Excel chart object, meaning it can be manually resized, styled, or repositioned after creation, just like any chart created through the standard user interface. The primary benefit here is the automation of the creation process itself, which significantly reduces interaction time when numerous charts are required.

Customizing Chart Dimensions and Aesthetics

While the automated process provides a functional chart immediately, developers frequently need to customize the size and position for better integration into a dashboard or report layout. The dimensions of the embedded chart are controlled by the arguments passed to the `ChartObjects.Add()` function, specifically **Width** and **Height**. These values are measured in points, a standard unit for screen layout. For instance, increasing the **Width** from the default 400 points to 600 points would result in a significantly wider chart area, requiring adjustment of surrounding data or elements.

Furthermore, positioning the chart precisely without relying on the dynamic `ActiveCell` is possible by defining explicit numerical values for the **Left** and **Top** arguments within the `ChartObjects.Add` method. These arguments also accept values in points, measured from the top-left corner of the

worksheet (cell A1). Developers can calculate these offsets based on expected cell dimensions to ensure perfect alignment with specific parts of the spreadsheet, making the macro entirely independent of the user's cursor location at the time of execution.

Finally, the choice of visualization is determined by the `ChartType` property. Although we used `xlColumnClustered`, the `ChartObject` framework supports a vast library of constants. For instance, changing the constant to `xlBarClustered` would automatically generate a horizontal bar chart from the same data, providing immediate flexibility in presentation style. Understanding these core properties allows for high-level customization of automated charting routines.

ARABPSYCHOLOGY.COM