

# How to Easily Adjust Bar Width in Matplotlib

Authored by  
**stats writer**

November 21, 2025

## RECOMMENDED CITATION

stats writer (2025). *How to Easily Adjust Bar Width in Matplotlib*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=98700>

When creating effective visualizations, the aesthetic control over chart elements is paramount. In the realm of Python data visualization, Matplotlib stands as the foundational library, offering granular control over every aspect of a plot. For bar charts, one of the most fundamental design decisions is determining the visual spacing and thickness of the bars themselves. This seemingly minor detail significantly impacts readability and the perceived distribution of the data.

The process of adjusting the width of bars in a Matplotlib plot is straightforward yet crucial for achieving professional-grade graphics. Unlike some other plotting libraries where bar width might be automatically calculated based purely on canvas size, Matplotlib provides a dedicated, explicit parameter within its core plotting function to manage this dimension. Ensuring that the bars are neither too thin--making them difficult to distinguish--nor too wide--causing visual clutter or overlap--is key to effective visual communication.

This tutorial will delve deep into the mechanics of controlling bar dimensions using the `plt.bar()` function. We will demonstrate how manipulation of the bar width allows developers and data scientists to tailor visualizations precisely to their audience and the specific characteristics of the underlying dataset. While altering the overall plot size using arguments like `figsize` can proportionally affect bar width, the most direct and precise method involves utilizing the function's dedicated width argument, providing absolute control regardless of the overall figure dimensions.

## The Central Role of the `width` Parameter in `plt.bar()`

The primary mechanism for controlling bar thickness in Matplotlib is the width parameter, which is an integral argument of the `plt.bar()` function. This parameter accepts a floating-point number representing the fraction of the spacing allocated to each bar along the categorical axis. By default, Matplotlib sets the value of the `width` parameter to **0.8**. This default value is often chosen to provide adequate separation between adjacent bars, ensuring clarity without wasting horizontal space.

To implement an adjustment, you simply specify the desired width value directly in the function call. For instance, executing `plt.bar(x, y, width=0.5)` explicitly sets the bar thickness to 0.5 units relative to the spacing between x-axis ticks. Increasing this value above 0.8 will result in thicker bars that occupy more of the available space, potentially reducing the gap between them. Conversely, decreasing the value below 0.8 yields narrower, thinner bars, increasing the visual whitespace between categories.

Understanding the effective range of the width parameter is vital. A value of **1.0** is the maximum practical setting; at this width, the bars will abut one another, eliminating all space between categories. Values exceeding 1.0 are possible but typically lead to bars overlapping, which can severely hinder visual interpretation and is generally discouraged unless specific layering effects are intended. Therefore, most customization efforts will involve values ranging between 0.1 (very

narrow) and 1.0 (maximum width without overlap).

The following foundational code snippet illustrates how to explicitly set the width parameter when generating a bar plot using the `plt.bar()` function:

You can use the **width** argument to adjust the width of bars in a bar plot created by Matplotlib:

```
import matplotlib.pyplot as plt
```

```
plt.bar(x=df.category, height=df.amount, width=0.8)
```

The default value for **width** is **0.8** but you can increase this value to make the bars wider or decrease this value to make the bars more narrow.

The following example shows how to use this syntax in practice.

## Setting Up the Data: Preparing a pandas DataFrame for Visualization

Before any visualization can occur, the data must be organized into a suitable format. In the Python ecosystem, the pandas library is the industry standard for data manipulation, and its core structure, the DataFrame, is typically used as the input source for Matplotlib plots. For a standard bar chart, we require at least two columns: one categorical column for the x-axis (the items being compared) and one numerical column for the height (the values).

For demonstration purposes, let us consider a scenario where we are tracking the total sales figures for a selection of grocery products. This requires initializing a DataFrame containing product names and their corresponding sales totals. This structure allows us to easily map the `'item'` column to the x-axis of the bar plot and the `'sales'` column to the height of the bars.

The structured data preparation using pandas ensures that the visualization process is smooth and that Matplotlib correctly interprets the data types for plotting. The following code snippet demonstrates the creation and inspection of our sample sales DataFrame:

### Example: Adjust Width of Bars in Matplotlib

Suppose we have the following pandas DataFrame that contains information about the total sales of various products at some grocery store:

```
import pandas as pd
```

```
#create DataFrame
```

```
df = pd.DataFrame({'item': ,
```

```
'sales': })  
  
#view DataFrame  
print(df)  
  
item sales  
0 Apples 18  
1 Oranges 22  
2 Kiwis 19  
3 Bananas 14  
4 Limes 24
```

### Default Bar Width Visualization (width = 0.8)

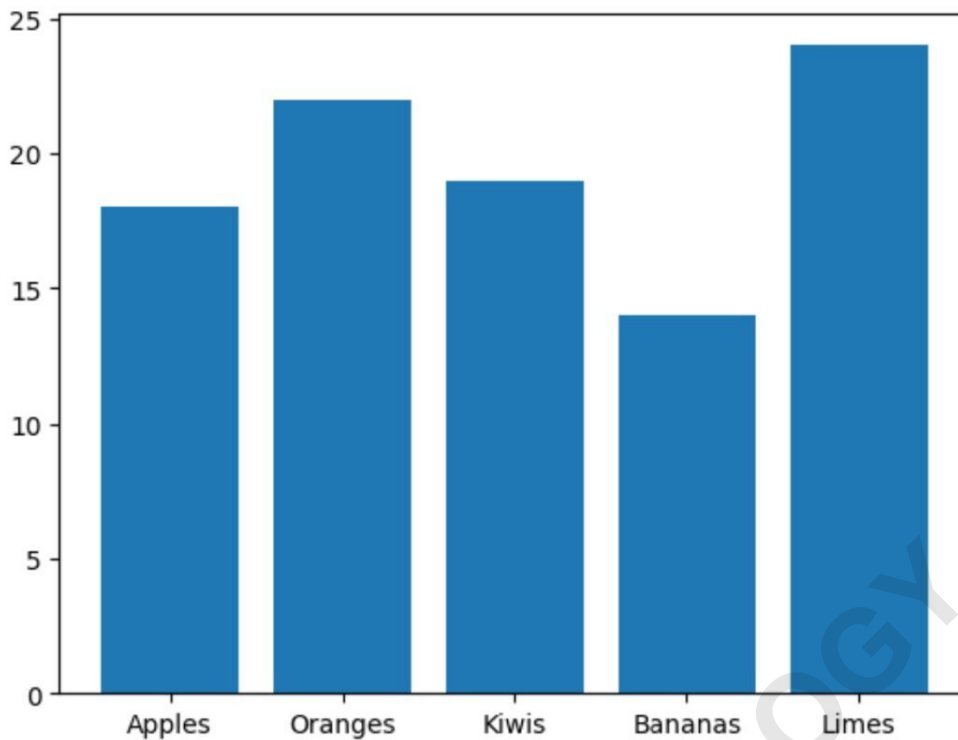
When generating a bar chart without supplying the optional `width` argument, [Matplotlib](#) automatically applies its established default value of **0.8**. This configuration provides a balanced look where the bars are thick enough to be visually substantial, yet sufficient white space (20% of the allocated category slot) remains between them to clearly delineate the boundaries of each category.

This default setting is usually adequate for exploratory data analysis (EDA) and simple presentations, as it requires minimal code customization while delivering a visually sound output. It ensures that the categorical labels are easily associated with their respective bars, minimizing visual ambiguity. Understanding this baseline is essential because all subsequent width adjustments will be judged relative to this 0.8 standard.

The following code utilizes our prepared [pandas](#) DataFrame to generate the default bar chart visualization. Notice that no `width` parameter is passed, relying entirely on the internal Matplotlib defaults:

We can use the following code to create a bar chart to visualize the number of sales of each item:

```
import matplotlib.pyplot as plt  
  
#create bar chart  
plt.bar(x=df.item, height=df.sales)
```



By default, Matplotlib uses a width of **0.8**.

### Adjusting Bar Width for Narrower Representation

There are several scenarios where a narrower bar width is preferable. If the plot contains a very large number of categories, using the default 0.8 width might cause the x-axis labels to overlap or result in a visually heavy chart. Decreasing the `width` parameter effectively thins the bars, increasing the empty space between them. This increased separation can dramatically improve clarity, especially when comparing many distinct, closely grouped values.

For instance, reducing the width to **0.4** means the bars occupy less than half of the space allocated to their respective category, resulting in significant whitespace. This technique is often employed in complex visualizations or dashboards where a cleaner, less visually dominant appearance is required. When using narrow bars, it is essential to ensure that the bars are still thick enough to represent the height values accurately without appearing as mere lines.

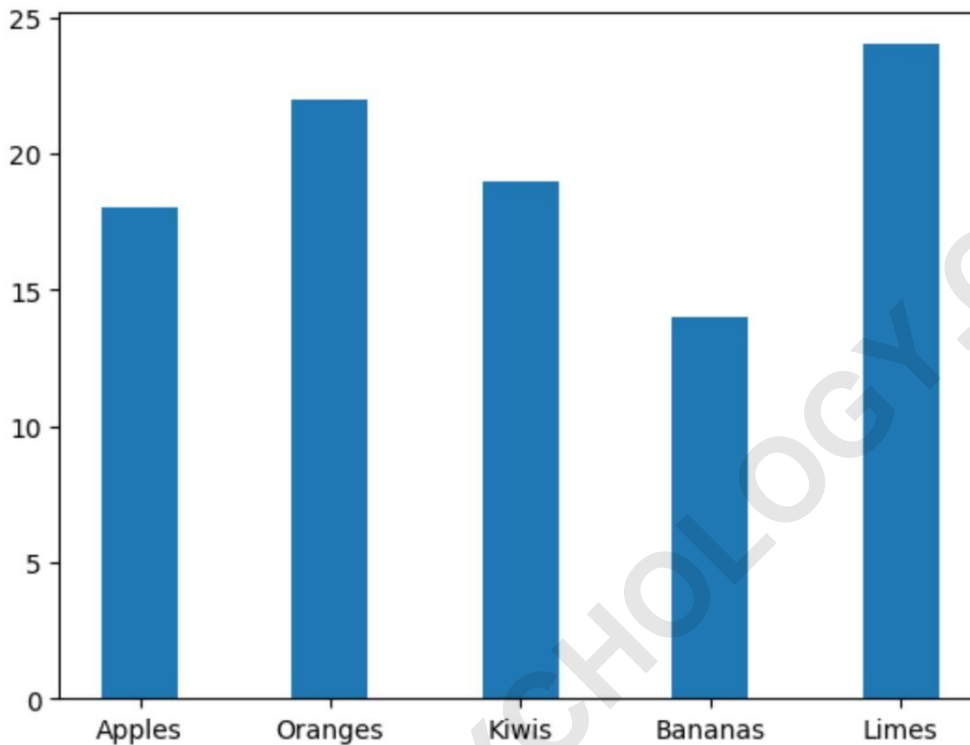
We apply this technique to our sales data, setting the width to 0.4. This customization is achieved by simply adding the `width=0.4` argument to the `plt.bar()` function call. Observe the visual transformation compared to the default plot:

However, we can use the **width** argument to specify a different value:

```
import matplotlib.pyplot as plt
```

```
#create bar chart with narrow bars
```

```
plt.bar(x=df.item, height=df.sales, width=0.4)
```



Notice that the bars are much more narrow.

### Maximizing Bar Width: Achieving Continuous Bars (width = 1.0)

Conversely, maximizing the bar width is sometimes necessary, particularly when the visualization is intended to mimic a histogram where continuity between bins is required. By setting the width parameter to **1.0**, we ensure that each bar occupies 100% of the horizontal space allocated to its category. This action eliminates the visual gap between adjacent bars, creating a continuous block effect.

When using `width=1.0`, it is often beneficial to define a distinguishing boundary, such as an `edgecolor`, to prevent the bars from blending into a monolithic block of color, which could confuse the viewer regarding where one category ends and the next begins. The addition of a thin black border, for example, maintains the continuity while still providing visual demarcation.

This technique is powerful for visualizing distributions where the categorical axis represents sequential or continuous bins, emphasizing the flow or progression of values across the spectrum.

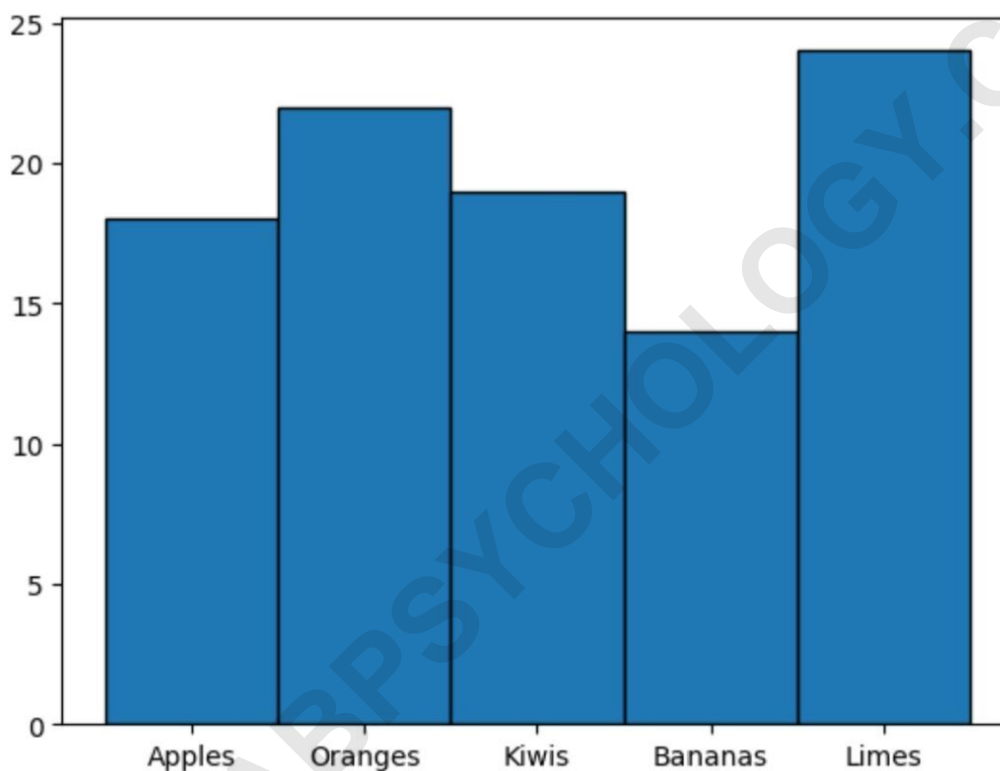
The code below demonstrates setting the maximum width and employing an `edgecolor` for aesthetic clarity:

Also note that if you use a value of **1** for width, the bars will touch each other:

```
import matplotlib.pyplot as plt
```

```
#create bar chart with width of 1
```

```
plt.bar(x=df.item, height=df.sales, width=1, edgecolor='black')
```



## Best Practices for Choosing Optimal Bar Widths

Selecting the optimal bar width is more of a design decision than a purely technical one, rooted in principles of visual perception and data clarity. Graphic design best practices often recommend that the space between bars should be approximately half the width of the bars themselves. Following this principle suggests that a width setting of around **0.66** (where 0.66 is the bar and 0.34 is the space, approximately a 2:1 ratio) or the default **0.8** often provides a good balance for standard categorical bar charts.

When dealing with time series data or ordered categories, narrower bars (e.g., 0.5 or 0.4) might be used if the sequence is more important than the magnitude comparison between immediate

neighbors. Conversely, if the plot is sparse, meaning there are few categories, a wider bar (closer to 0.9 or 0.95) can make the visualization feel more substantial and less dominated by white space.

Ultimately, the flexibility of the `width` argument in `plt.bar()` allows for precise customization. It is advisable to experiment with different values to find the aesthetic that best suits the data density and the overall presentation layout. Developers are encouraged to adjust the value for the **width** argument dynamically to ensure the bars in the plot are as wide or narrow as needed to maximize data interpretation and visual appeal.

Feel free to adjust the value for the **width** argument to make the bars in the plot as wide or narrow as you'd like.

The following tutorials explain how to perform other common tasks in [Matplotlib](#):