Learn How to Change Histogram Colors in Matplotlib: A Step-by-Step Guide

Authored by

Mohammed looti

October 31, 2025

Understanding Histograms and Color Customization in Matplotlib

Effective data visualization is fundamental to modern data science, and the Matplotlib library stands as the cornerstone for generating plots in Python. Among its many capabilities, creating a histogram is essential for visualizing the distribution of a dataset. While Matplotlib provides sensible defaults, tailoring the aesthetic elements--specifically color--is often necessary to meet branding requirements, enhance clarity, or improve overall visual appeal. This comprehensive guide details the precise methods for modifying the fill and edge colors of a Matplotlib histogram using the core parameters available in the `plt.hist()` function.

A histogram plots the frequency of data points falling into defined bins, providing immediate insight into the underlying statistical distribution. By default, Matplotlib applies a standard, often dark blue, color scheme. Customization, however, allows analysts to differentiate between multiple distributions plotted simultaneously or simply align the plot with the theme of a report. The primary mechanism for achieving this customization lies within two key arguments: `color`, which controls the interior fill, and `ec` (edge color), which defines the borders of the bins.

To effectively control the visual characteristics of the bars in your frequency plot, you must utilize specific parameters within the `plt.hist()` function call. Understanding these parameters is crucial for achieving precise visual results. The basic syntax for applying custom colors involves passing string values representing desired colors directly to the function arguments.

The Core Syntax for Coloring Matplotlib Histograms

You can use the following basic syntax to modify the color of a histogram in Matplotlib, specifically addressing both the interior fill and the exterior boundary of the histogram bins:

plt.hist(data, color = "lightblue", ec="red")

This simple line of code encapsulates the essential modification parameters. The parameters accept various inputs, including standard HTML color names (like 'red' or 'lightblue'), hexadecimal codes (e.g., '#FF5733'), or single-letter abbreviations (e.g., 'k' for black). Mastering these inputs allows for high-precision control over the visual outcome of your data plots.

Let's break down the function arguments responsible for color modification:

data: This required argument represents the list, array, or Python iterable containing the numerical values that the histogram will analyze and display.
color: This parameter dictates the primary **fill color** for the interior of the rectangular bars representing the frequency counts in the histogram.
ec: Short for **edge color**, this parameter controls the color of the outline or border surrounding

each individual bar. Defining an edge color is often beneficial for plots where bars are closely packed or when a strong contrast is desired against the background.

The following comprehensive example demonstrates how to apply this foundational syntax in a typical plotting workflow, moving from a standard default plot to a fully customized visual output.

Practical Example: Customizing Fill and Edge Colors

To illustrate the power of the **`color`** and **`ec`** parameters, we will first define a sample dataset. This data represents a simple, non-normal distribution, which is ideal for testing the visual impact of different color schemes. We begin by importing the necessary Matplotlib library component, specifically pyplot, which is conventionally aliased as `plt`.

Suppose we have the following list of values that we wish to visualize:

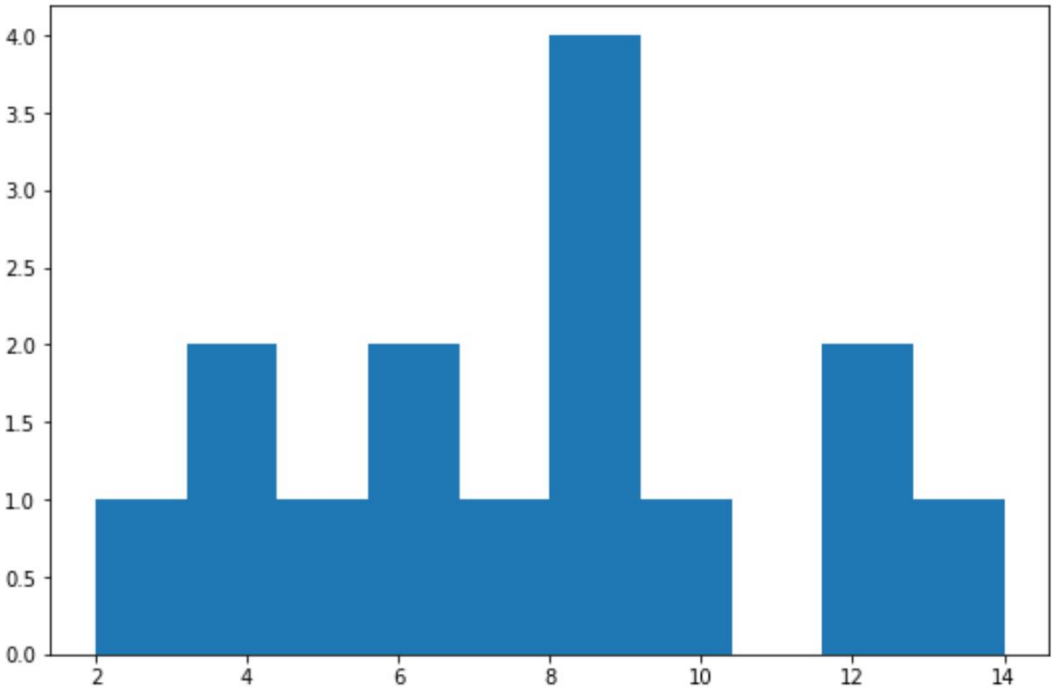
#define list of data
data =

Before applying any customization, it is useful to see the default output generated by Matplotlib. If we simply call **`plt.hist(data)`**, the library automatically selects a default color (typically a dark, solid blue) and, crucially, sets the edge color to be invisible or matching the fill color, meaning no distinct border is visible between bins.

We can use the following basic syntax to create a Matplotlib histogram to visualize the values in this dataset:

import matplotlib.pyplot as plt

#create histogram
plt.hist(data)



As demonstrated in the visualization above, Matplotlib's default behavior is to create a histogram with a dark blue fill color and no visible edge color. This default setting can sometimes make distinguishing individual bins difficult, especially when the bar heights are similar or when the contrast against the background is low. This necessitates the use of customization arguments to improve clarity and visual impact.

Applying Custom Fill Color and Edge Color

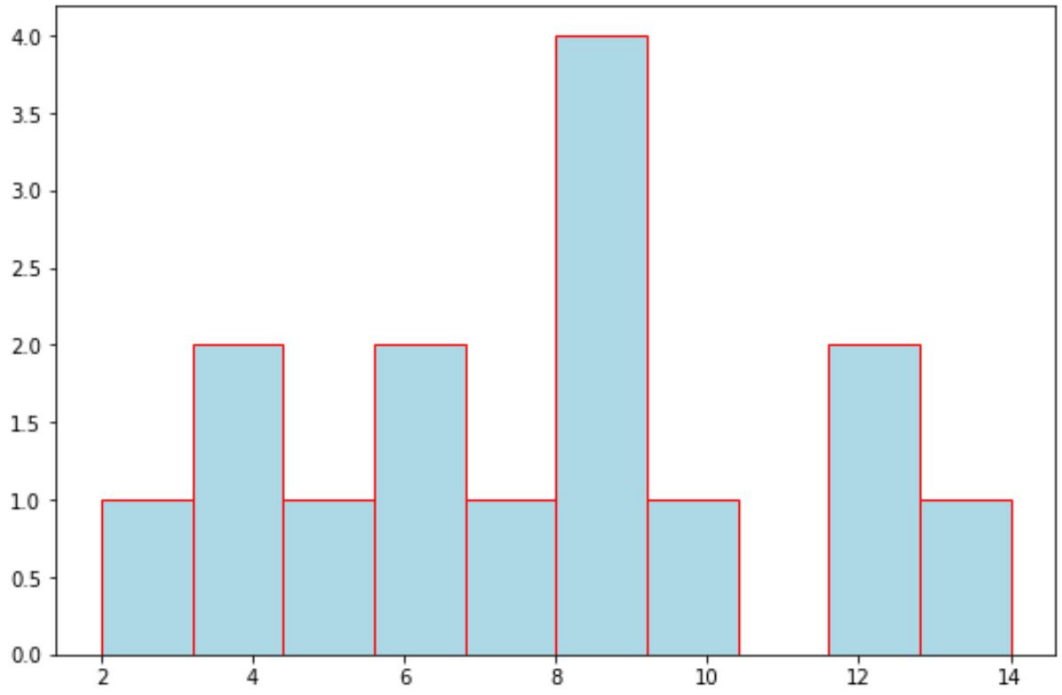
The goal of customization is to enhance readability and align the plot with specific design standards. We will now apply the `color` and `ec` parameters introduced earlier. For this demonstration, we select a `lightblue` fill color, which is softer than the default, and a contrasting `red` edge color to clearly delineate the boundaries of the bins. This combination immediately improves the visual separation between the frequency counts, making the plot easier to interpret quickly.

To achieve this specific aesthetic, we modify the `plt.hist()` call to include the chosen color arguments. Remember that color strings must be enclosed in quotation marks, and Matplotlib supports a vast library of named colors and precise hexadecimal codes for more specific needs.

However, we can use the following syntax to change the fill color to light blue and the edge color to red:

import matplotlib.pyplot as plt

```
#create histogram with light blue fill color and red edge color
plt.hist(data, color = "lightblue", ec="red")
```



Upon execution, the resulting visualization confirms that the new histogram has successfully adopted a light blue fill color paired with a distinct red edge color. This visual modification significantly aids in interpreting the distribution, particularly by emphasizing the boundaries between adjacent bins. This technique is universally applicable whether you are using predefined color names or precise hexadecimal values, providing essential flexibility in design for publication or presentation.

Controlling Edge Thickness: The 'lw' Parameter

Beyond simply setting the color of the edges, Matplotlib also offers fine control over the thickness or weight of those boundary lines. This is achieved using the `lw` argument, which stands for **line width**. Adjusting the line width is particularly useful when creating plots intended for high-resolution displays or print materials where a thin line might disappear, or conversely, when a highly subtle border is required.

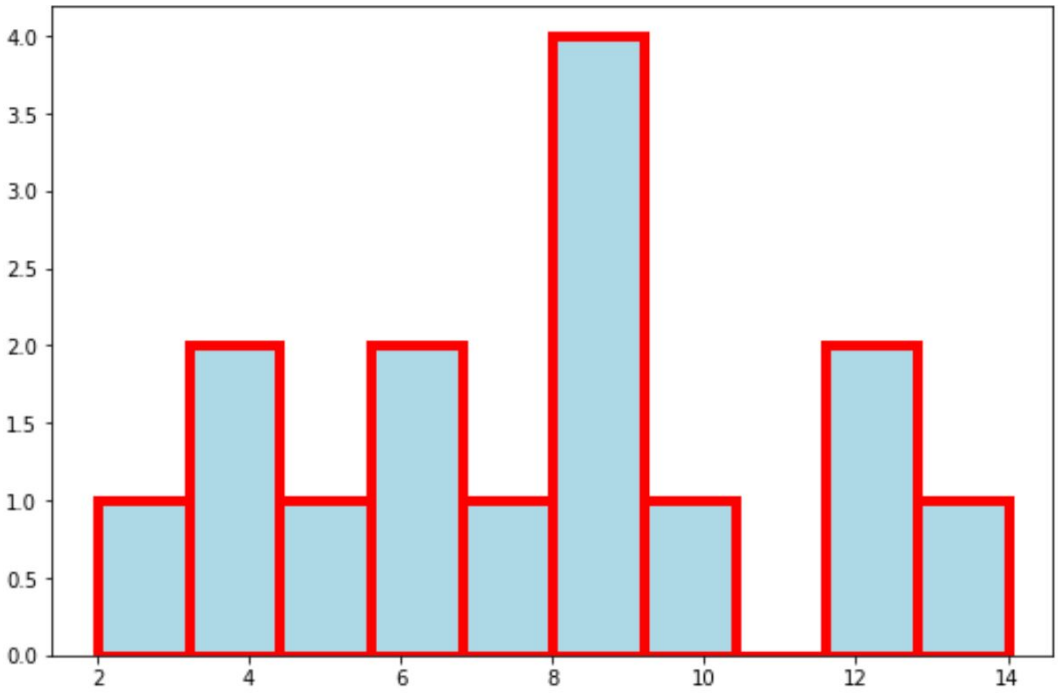
The `lw` parameter accepts a numerical value, typically an integer or float, which dictates the thickness of the border line in points. A larger value results in a thicker line, while a smaller value produces a more subtle border. Integrating `lw` complements the `ec` parameter, giving the user complete control over the visual presentation of the bin boundaries and enhancing the overall

visual hierarchy of the plot.

You can also use the `lw` argument to modify the line width for the edges of the histogram:

import matplotlib.pyplot as plt

#create histogram
plt.hist(data, color = "lightblue", ec="red", lw=5)



As is evident in this final visualization, by setting `lw=5`, the edges of the bars become significantly thicker and more pronounced, drawing greater attention to the individual bins. The larger the numerical value you use for `lw`, the bolder the edges on the bars will be. It is generally recommended to experiment with values between 1 and 3 for standard reports, reserving larger values for highlighting specific features or for highly stylized presentations.

Advanced Color Options and Official Documentation

While using simple color strings like 'red' or 'blue' is effective for quick modifications, pyplot offers extensive support for comprehensive color palettes, or colormaps. For instance, if you were plotting several histograms on the same axes to compare different populations, you might want to use Matplotlib's built-in colormaps (like 'viridis' or 'plasma') in conjunction with other parameters (such as the `alpha` parameter for transparency) to ensure all distributions are distinguishable yet aesthetically cohesive.

For users requiring finer control over visual output, it is highly recommended to consult the official documentation. The `plt.hist()` function supports dozens of parameters related to binning, normalization, stacking, and orientation, all of which interact with the color choices. For instance, if using the `rwidth` parameter to reduce the width of the bars, a thicker `lw` might be necessary to ensure the edges remain clearly visible.

Note: You can find the complete documentation for the Matplotlib **hist** function, which covers all accepted color formats and advanced aesthetic controls, including the integration of color cycle properties and named color palettes. Mastering these tools allows for the creation of truly professional and insightful data visualizations in Python.

Additional Resources for Matplotlib Mastery

To further enhance your skills in data visualization and statistical plotting using the Matplotlib ecosystem, consider exploring related tutorials. Understanding how to manage colors in histograms is just one step; comprehensive data analysis often requires proficiency in other plotting techniques, such as scatter plots, bar charts, and line graphs.

The following resources offer guidance on other common operations in Python, building upon the foundational knowledge of histogram customization: