

# How to Easily Change the Color of Your Seaborn Histogram

Authored by  
**stats writer**

November 20, 2025

## RECOMMENDED CITATION

stats writer (2025). *How to Easily Change the Color of Your Seaborn Histogram*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=98518>

One of the most powerful features of the [Seaborn](#) library is its capacity to generate aesthetically pleasing statistical visualizations quickly. When creating a [histogram](#), controlling the visual aesthetics, such as the bar color, is crucial for effective [data visualization](#). You can easily change the color of a Seaborn histogram by utilizing specific parameters within the plotting function itself. While some general styling can be managed globally using functions like `sns.set_style()`, direct color manipulation for the bars in a histogram is most effectively achieved using the `color` and `edgecolor` arguments during the function call.

These arguments allow for precise control over the appearance of the graphical elements. The `color` parameter determines the primary fill color of the bars, while the `edgecolor` parameter sets the color for the outlines surrounding each bin. Mastering these two parameters ensures your visualizations are not only accurate but also visually aligned with your presentation or branding requirements.

## The Core Coloring Parameters: color and edgecolor

To customize the appearance of the bars in a Seaborn histogram, you must leverage the built-in arguments available in the `sns.histplot` function. Specifically, you can use the **color** and **edgecolor** arguments in Seaborn to change the fill color and outline color, respectively, of bars in the generated visualization. This offers immediate and precise control over the aesthetic elements of your plot.

The **color** argument accepts standard color names (e.g., 'blue', 'orange', 'red'), RGB tuples, or [hex color codes](#). Similarly, **edgecolor** takes the same input types. By setting these two independently, you can create high-contrast visuals that separate the bars distinctly from their background and from each other, which is especially useful when dealing with detailed distributions or multiple overlaid plots.

The general syntax for implementing these arguments within the function is straightforward, requiring only the addition of these parameters alongside your data and variable specifications:

```
sns.histplot(data=df, x='some_variable', color='orange', edgecolor='red')
```

The following detailed example demonstrates how to apply these arguments in practice, moving from a standard, default visualization to a highly customized plot using both named colors and hex codes.

## Setting Up the Demonstration Environment and Dataset

Before plotting, we need a representative dataset. For this demonstration, we will use the [Pandas](#) library to create a DataFrame simulating numerical data, which is typical for histogram analysis.

Our hypothetical dataset tracks the performance of 200 basketball players, recording their total points scored. This simulation relies on NumPy for generating normally distributed data, ensuring a realistic distribution suitable for histogram plotting.

Creating reproducible examples is vital in statistical programming. We set a random seed using `numpy.random.seed(1)` so that anyone running this code will generate the exact same dataset, guaranteeing consistency in the plotted results across different executions. This simulated dataset allows us to focus entirely on the visualization customization techniques without worrying about external data dependencies.

The DataFrame generation includes a 'team' column (A or B) and the primary variable of interest, 'points', which follows a normal distribution centered around 15 with a standard deviation of 4. This setup provides a clear variable ('points') to plot and analyze its frequency distribution using the histogram:

```
import pandas as pd
import numpy as np

#make this example reproducible
np.random.seed(1)

#create DataFrame
df = pd.DataFrame({'team': np.repeat('A', 200),
                  'points': np.random.normal(size=200, loc=15, scale=4)})

#view head of DataFrame
print(df.head())

team points
0 A 21.497381
1 A 12.552974
2 A 12.887313
3 A 10.708126
4 A 18.461631
```

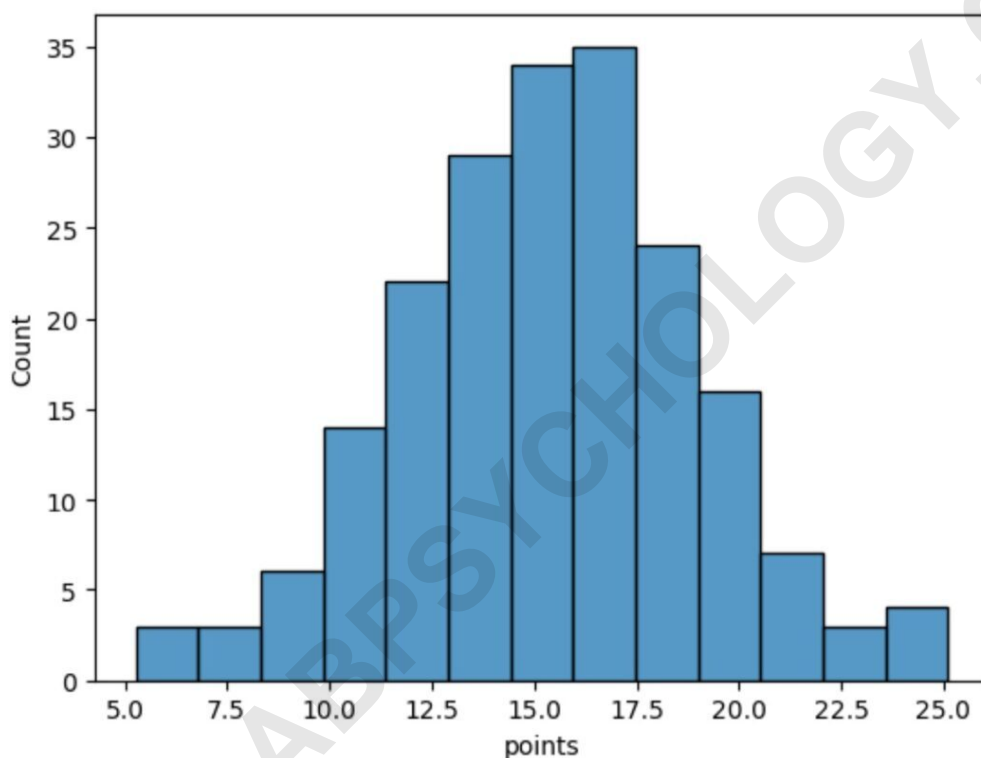
## Visualizing Data with the Default Seaborn Histogram

Once the data is prepared within the DataFrame `df`, we can generate the initial histogram. By calling `sns.histplot` without specifying the `color` or `edgecolor` arguments, Seaborn automatically applies its default styling. This default styling is dictated by the currently active global style (e.g., 'darkgrid' or 'whitegrid') and the default color palette.

The visualization below shows the distribution of values in the **points** column using the standard parameters. Typically, the default output utilizes a medium blue as the fill color and a dark color (often black or a deep gray, depending on the chosen style) for the bar outlines. This default configuration provides high readability and is generally sufficient for initial data exploration, but it lacks specific aesthetic branding.

```
import seaborn as sns
```

```
#create histogram to visualize distribution of points  
sns.histplot(data=df, x='points')
```



As observed, by default, Seaborn uses blue as the fill color and black as the outline color for the bars in the histogram. While clean, this presentation might not satisfy specific reporting requirements or aesthetic preferences.

## Applying Custom Colors Using Standard Names

To move beyond the default styling, we introduce the **color** and **edgecolor** arguments directly into the `sns.histplot` call. This method allows us to specify common color names, which are immediately recognizable and easy to implement. For instance, selecting 'orange' for the fill color and 'red' for the edge color creates a vibrant contrast that immediately draws attention to the

distribution.

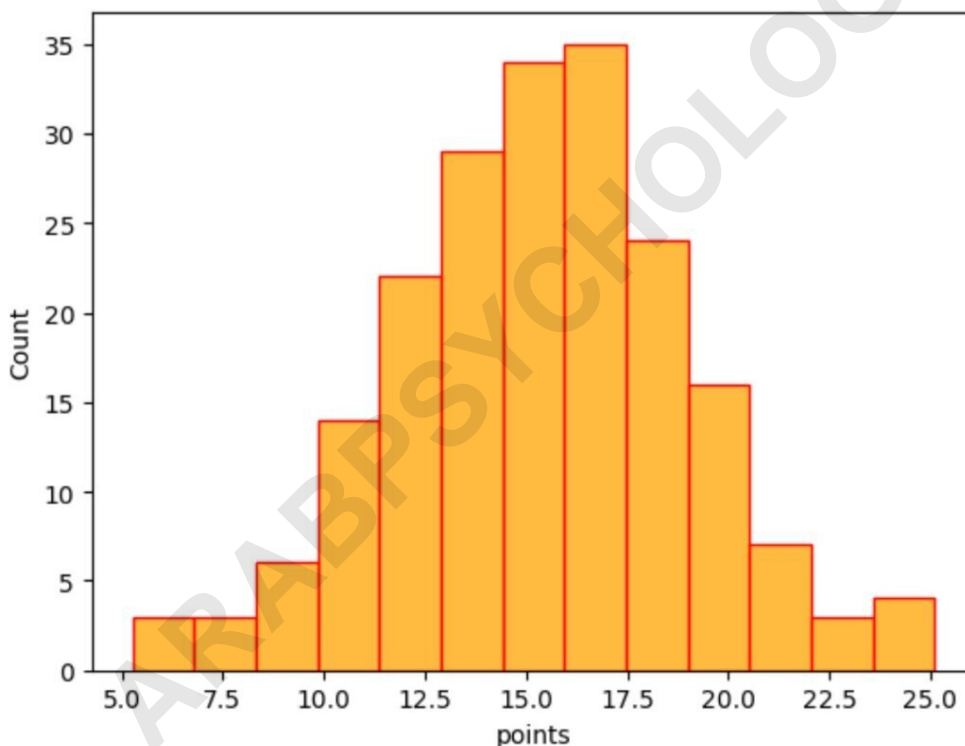
Choosing distinct colors for the fill and the edge is a best practice, as it helps define the boundaries of each bin clearly. This is particularly helpful when the bins are wide or when analyzing a plot where multiple datasets are overlaid. Using standard color names is the fastest way to customize the appearance without needing to consult specific color codes.

The code snippet below demonstrates this customization, showcasing how simple it is to override the default aesthetics provided by the library:

```
import seaborn as sns
```

```
#create histogram to visualize distribution of points
```

```
sns.histplot(data=df, x='points', color='orange', edgecolor='red')
```



Notice that the histogram now has a primary fill color of **orange** and a distinct outline color of **red**. This simple modification significantly alters the visual impact of the plot and enhances its utility in presentations where specific color palettes are required.

## Advanced Customization with Hex Color Codes

For situations requiring extreme precision in color selection, such as adherence to corporate

branding guidelines or the use of carefully curated, colorblind-safe palettes, using hex color codes is the superior method. Hex color codes provide a specific 6-digit (or 3-digit shorthand) alphanumeric sequence preceded by a hash symbol (#), corresponding to exact levels of red, green, and blue (RGB).

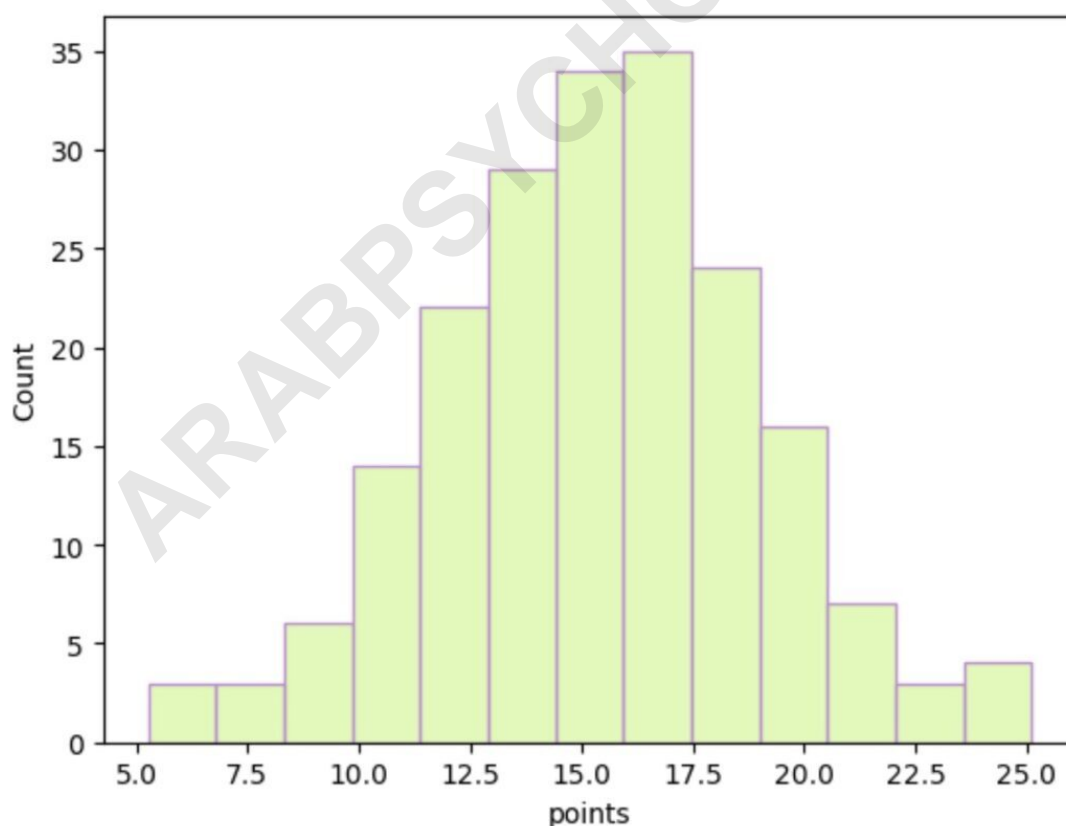
The use of hex codes allows for virtually limitless customization options. In the example below, we choose a soft, light green fill color (`#DAF7A6`) combined with a muted purple border (`#BB8FCE`). This demonstrates the flexibility available when standard named colors are insufficient for achieving the desired aesthetic or accessibility goals. This is particularly valuable in advanced data visualization applications.

To implement this, simply replace the standard color names in the `color` and `edgecolor` arguments with the desired hex codes:

```
import seaborn as sns
```

```
#create histogram to visualize distribution of points
```

```
sns.histplot(data=df, x='points', color='#DAF7A6', edgecolor='#BB8FCE')
```



The resulting plot shows the exact colors specified by the hex codes, illustrating the high degree of

customization achievable within the Seaborn ecosystem when utilizing these explicit color definitions.

## Alternative Methods for Global Color Management

While specifying `color` and `edgecolor` in the `sns.histplot` function provides localized control, there are scenarios where you might want to manage the color scheme globally for all plots generated within a session. This is often achieved through Seaborn's style and palette functions, which interact with Matplotlib's settings.

One primary tool for global color control is `sns.set_palette()`. By calling this function and passing a list of colors, a Matplotlib colormap name, or a specific Seaborn palette name (e.g., 'deep', 'muted', 'pastel'), you can set the default cyclical color scheme. Any subsequent plotting function that relies on the default palette (for example, plotting multiple lines or categories) will utilize this new scheme.

However, it is critical to note that for single-variable histograms using `sns.histplot`, the function typically defaults to using only the first color in the active palette unless the `color` argument is explicitly overridden. Therefore, while global palette settings are useful for complex plots, **color** and **edgecolor** remain the most direct way to style a singular histogram.

## Summary of Best Practices for Styling Seaborn Histograms

Effective color customization is a cornerstone of professional data visualization. When working with Seaborn histograms, adhering to a few best practices will ensure clarity and aesthetic quality:

**Use Direct Arguments for Specificity:** Always use the `color` and `edgecolor` parameters within `sns.histplot` when you need the histogram to display a specific, fixed color, overriding any global palette settings.

**Ensure Contrast:** Choose a fill color and an edge color that contrast well. This separation defines the bars clearly and improves readability, particularly important for printed or low-resolution visualizations.

**Leverage Hex Codes for Branding:** Utilize hex color codes when strict adherence to a corporate or academic color scheme is necessary, offering the highest level of color precision.

**Consider Accessibility:** When selecting colors (whether named or hex), prioritize schemes that are colorblind-friendly. Tools and resources are readily available online to check color combinations for accessibility compliance.

By applying these techniques, developers and analysts can transform standard statistical plots into customized, high-impact visuals suitable for any reporting environment.

How to Create a Grouped Barplot in Seaborn

ARABPSYCHOLOGY.COM