# Learning Matplotlib: How to Change Plot Background Color with set_facecolor()

Authored by

**Mohammed looti**

November 6, 2025

Visualizing data effectively often requires careful attention to aesthetics. In the realm of Python data visualization, Matplotlib serves as the cornerstone library for creating static, interactive, and animated plots. A fundamental customization task for improving plot readability is changing the background color of the plotting area, which is easily accomplished using the **set_facecolor()** method.

This powerful function operates directly on the Axes object, allowing precise control over the visual appearance of the plot canvas. When you initialize a typical Matplotlib structure, you define both the main container (the Figure) and the specific plotting area (the Axes). For instance, using the standard initialization pattern with `plt.subplots()`:

**fig, ax = plt.subplots()**

If this structure is established, altering the background color of the plot area--the 'face' of the graph--becomes straightforward. You simply invoke the **set_facecolor()** method on the Axes object (represented here as `ax`), passing the desired color specification as an argument. This method is the most efficient and recommended approach for customizing the visual environment surrounding your data points, as demonstrated below:

**ax.set_facecolor('pink')**

Throughout this tutorial, we will explore several practical applications of this function, demonstrating how to use different color inputs--from simple names to specific hexadecimal codes--and how to apply these customizations to complex layouts involving multiple subplots.

## Understanding Matplotlib's Figure and Axes Objects

Before diving into the code examples, it is crucial to understand the hierarchical structure that Matplotlib employs. Every plot consists of a **Figure** object and one or more **Axes** objects. The Figure acts as the overarching container, essentially the window or canvas that holds everything. The Axes object, however, is the actual plotting area--it is where the data is displayed, complete with ticks, labels, and the background that we intend to modify.

When we use the common initialization function `plt.subplots()`, we are simultaneously creating both of these key components. The first returned variable, typically named `fig`, references the Figure, and the second variable, usually `ax` (or an array of `ax` objects when creating multiple plots), references the Axes. The background color we are targeting with **set_facecolor()** is specifically the background of the Axes object, ensuring that only the region immediately surrounding the plotted data is altered, not the entire window canvas.

The choice to use **set_facecolor()** on the Axes object, rather than modifying the Figure's color, is usually dictated by visualization goals. Changing the Axes background is ideal for highlighting the data visualization itself, often providing a stronger contrast or a thematic color palette. If the goal were to change the background of the entire physical window holding the graph, a different method, such as `fig.patch.set_facecolor()`, would be required. However, for standard plot customization, focusing on the Axes is the primary approach for manipulating the visual field.

## Example 1: Utilizing Named Colors for Backgrounds

The simplest method for defining a background color is by using a standard, human-readable color name. Matplotlib supports a wide range of common color names, making the code highly intuitive and easy to read. This approach is perfect for quick adjustments or when a specific, precise shade is not critical to the visualization's purpose. We will demonstrate this using a simple scatter plot and setting the background to 'pink'.

In this example, we begin by importing `matplotlib.pyplot`, defining our figure and axes, and creating the dummy data arrays (A and B). The core action occurs when we call `ax.set_facecolor('pink')`. Notice that the color name is passed as a string argument directly to the function. This immediately overrides the default white or gray background typically applied by Matplotlib, resulting in a visually distinct plot area that draws attention to the displayed data points.

**import matplotlib.pyplot as plt**

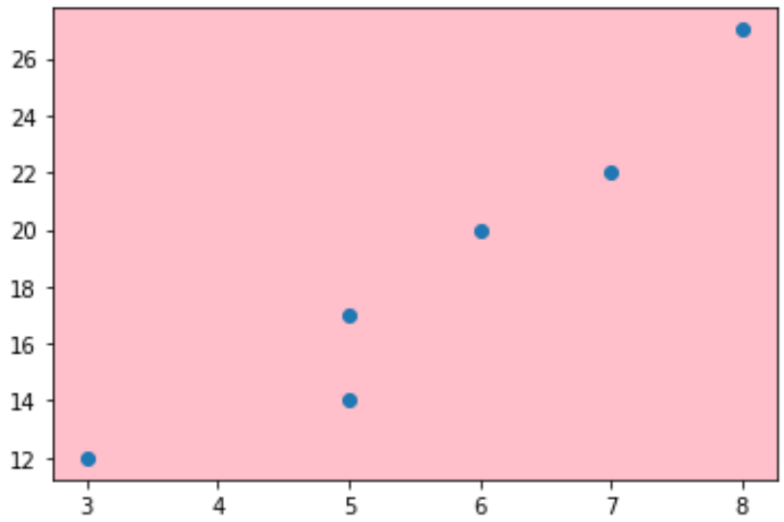
# Initialize the plot figure and the Axes object
fig, ax = plt.subplots()

# Define two arrays containing sample data points
A =
B =

# Create the scatterplot
ax.scatter(A, B)

# Specify background color using a standard color name ('pink')
ax.set_facecolor('pink')

# Display the final scatterplot
plt.show()




## Example 2: Achieving Precision with Hex Color Codes

While named colors are convenient, they lack the precision often required for corporate branding, scientific publication, or complex visual themes. For these scenarios, using a Hex color code is the superior method. A Hex color code is a six-digit (or three-digit shorthand) alphanumeric string preceded by a hash symbol (#), representing the exact mix of red, green, and blue (RGB) values, providing over 16 million possible color variations.

The **set_facecolor()** method accepts these codes seamlessly. By providing a Hex code, you ensure that the background color is precisely defined, regardless of the system or display settings, guaranteeing consistency in your data visualizations. This is critical when adherence to a specific color standard is mandatory, providing granular control over the output.

In the following code block, we reuse the basic scatter plot structure but replace the named color 'pink' with the Hex code `#33FFA2`, which corresponds to a bright spring green. Note that the usage pattern remains identical; only the format of the string argument passed to **set_facecolor()** changes. This flexibility underscores why Matplotlib remains the preferred tool for intricate graphical customization.

**import matplotlib.pyplot as plt**

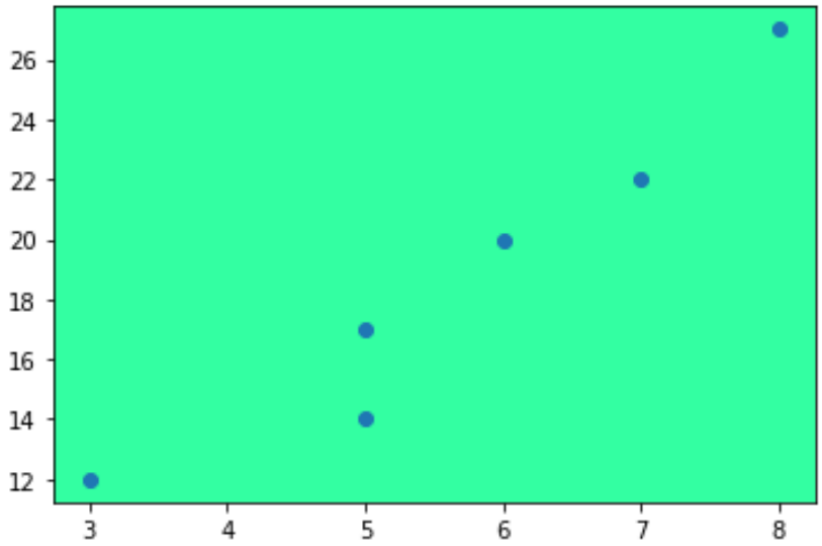
# Define plot figure and axis
fig, ax = plt.subplots()

# Define two arrays for plotting
A =
B =

```
# Create scatterplot
ax.scatter(A, B)

# Specify background color using a Hex color code
ax.set_facecolor('#33FFA2')

# Display scatterplot
plt.show()
```




## Example 3: Customizing Multiple Subplots Independently

In advanced data analysis, it is common practice to display several related visualizations side-by-side using subplots. When working with multiple Axes objects within a single Figure, the ability to customize each subplot individually becomes essential for differentiating visualizations or highlighting specific data trends. Since **set_facecolor()** operates on the individual Axes object, this customization is straightforward and highly flexible.

When creating multiple subplots using `plt.subplots(rows, columns)`, the `ax` variable returned is typically a NumPy array of Axes objects. We can access and manipulate each subplot using standard array indexing (e.g., `ax`). This enables us to apply a unique background color to every single plot within the layout, which is particularly useful when comparing different experimental conditions or visualizing distinct data subsets that require visual separation.

The following example generates a 2x2 grid of subplots. We iterate through the array structure implicitly, applying four distinct colors--blue, pink, green, and red--to each quadrant. We also include `fig.tight_layout()` to automatically adjust the spacing between the plots, preventing

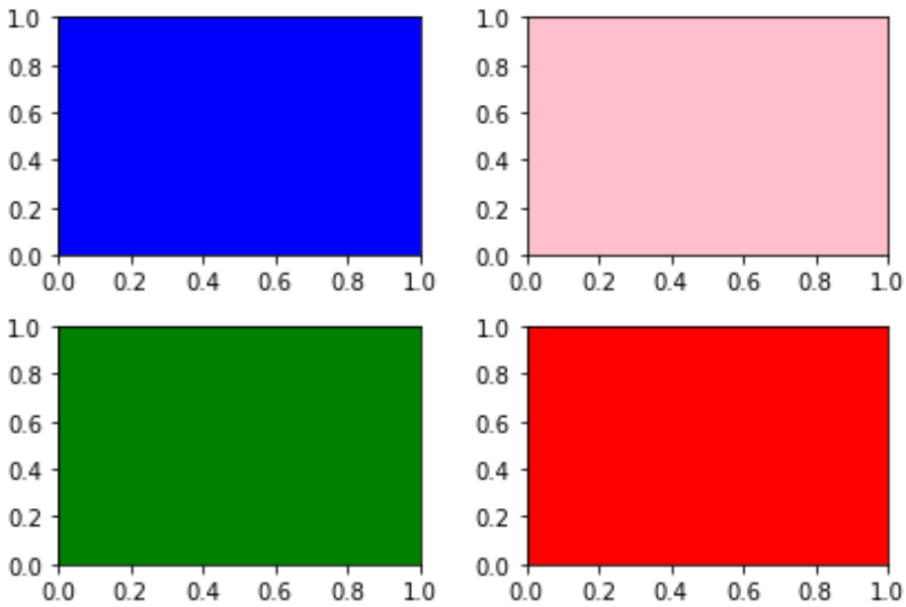
titles or labels from overlapping, which is a crucial step when managing complex subplot arrangements in Matplotlib and ensuring a clean final presentation.

**import matplotlib.pyplot as plt**

# Define subplots in a 2x2 grid. 'ax' is now an array of Axes objects.
fig, ax = plt.subplots(2, 2)
fig.tight_layout()

# Define background color for each subplot using array indexing
ax.set_facecolor('blue')
ax.set_facecolor('pink')
ax.set_facecolor('green')
ax.set_facecolor('red')

# Display the final subplots
plt.show()




## Advanced Considerations: Coloring the Figure vs. the Axes

While the primary focus of this guide is customizing the Axes background using **set_facecolor()**, a related customization often requested by users is changing the color of the Figure itself. It is important to distinguish between these two components. The Axes background is the area directly behind the data, bounded by the plot's axes lines. The Figure background is the padding or margin space surrounding the entire Axes object(s).

To change the color of the Figure object (`fig`), you must access the Figure's patch attribute, which represents the rectangular area defining the Figure. This is accomplished using the syntax `fig.patch.set_facecolor()`. Both the Figure and Axes objects support the same color inputs (named colors, Hex color codes, or RGB tuples), maintaining consistency across the library's styling methods.

Understanding this distinction is vital for achieving complex visual effects. For instance, you might set the Axes background to a dark color to make light data points pop, while setting the Figure background to a neutral gray to cleanly separate the plot from the surrounding interface. When working with Axes objects and Figures, utilizing both `ax.set_facecolor()` and `fig.patch.set_facecolor()` grants complete command over the visual composition of your final graphic. Mastering these methods ensures that every element of your subplots adheres to your desired aesthetic and readability standards.

**Related:** How to Adjust Spacing Between Matplotlib Subplots