Learning Matplotlib: A Guide to Repositioning Colorbars for Effective Data Visualization

Authored by

Mohammed looti

November 7, 2025

A **colorbar** is an essential element in high-quality data visualization, serving as a critical legend that maps numerical data values to the corresponding colors displayed on a plot. In the realm of scientific computing and graphical representation using powerful libraries like Matplotlib, the precise and effective placement of this visual key is paramount. While Matplotlib is designed to be highly user-friendly, its default mechanism positions the colorbar automatically, typically to the right of the main chart area. This default behavior, while convenient for simple figures, often falls short when developing sophisticated layouts or integrating multiple subplots into a single composite figure.

Standard placement frequently leads to visual clutter or inefficient use of space, especially when the figure aspect ratio is critical or when the colorbar needs to be aligned horizontally. Achieving granular control over the colorbar's location, orientation, padding, and size requires moving beyond the basic `plt.colorbar()` call. This level of customization is vital for producing publication-ready graphics where every graphical element must be meticulously placed to aid interpretation and maintain aesthetic standards.

To unlock this advanced positional control, visualization practitioners must utilize specialized utilities provided within the Matplotlib AxesGrid toolkit. Specifically, the function `make_axes_locatable` allows us to programmatically define the geometry of new axes relative to an existing plot. This tutorial will explore various practical demonstrations showing how to accurately reposition the colorbar--whether to the right, below, or above the main plot--thereby optimizing the overall visual architecture of your scientific figures.

Implementing Geometric Control with the AxesGrid Toolkit

The conventional Matplotlib method for generating a colorbar, `fig.colorbar(im)`, handles the creation of a new set of axes dedicated solely to the color scale. However, this simplicity comes at the cost of flexibility; the placement is determined internally by Matplotlib's layout manager, which attempts to find a "sensible" default spot. In scenarios requiring non-standard layouts, such as placing a horizontal colorbar directly below a plot or tightly integrating the colorbar within a grid of figures, this automated approach becomes insufficient.

This is precisely why the Matplotlib AxesGrid toolkit was developed. It offers sophisticated tools designed specifically for managing the relative positions and sizes of axes within complex figures. The foundational tool for colorbar repositioning is the `make_axes_locatable` function. This utility accepts an existing axis object (our primary chart, `ax`) and returns a specialized "divider" object. This divider essentially acts as a geometric manager, understanding the precise boundaries and dimensions of the main axis.

By leveraging this divider, we can dynamically generate new axes--specifically the colorbar axis--that are guaranteed to be positioned adjacent to the original axis. Crucially, this new axis creation

is done with explicitly controlled padding and size parameters. This technique ensures that the resulting colorbar scales correctly and maintains a consistent, defined spatial relationship with the primary plot, a significant advantage over manually defining axis coordinates, especially when dealing with figure resizing or dynamically generated content. The programmatic control offered by methods like `new_vertical` and `new_horizontal` allows us to precisely dictate the location and orientation of the color scale, overriding Matplotlib's default placement logic entirely.

Example 1: Establishing the Default Vertical Position (Right Side)

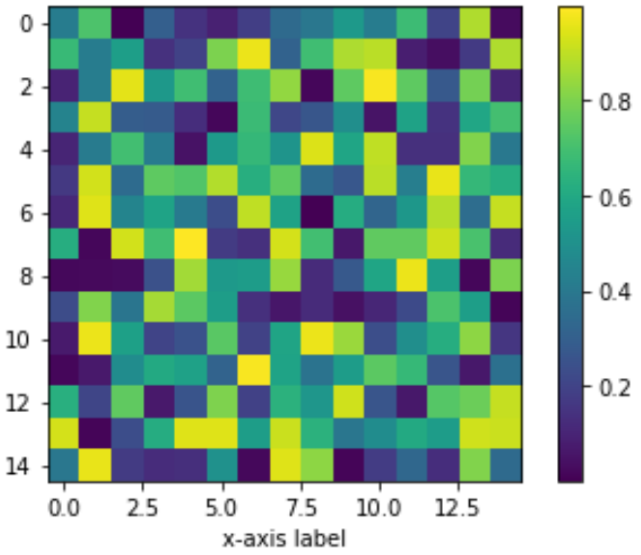
Before diving into complex repositioning, it is beneficial to establish the baseline visualization setup and observe Matplotlib's default behavior. This initial example demonstrates the minimum required code to generate a data visualization and attach a **colorbar** using the library's native functionality. This method is generally adequate for simple figures where the colorbar orientation is vertical and space is not a limiting factor on the right side of the plot.

The following script initiates the process by utilizing the Numpy library to generate a randomized 15x15 array of floating-point numbers. This array simulates typical scientific data that would be displayed using a continuous color mapping, such as in a heatmap or an image plot. The `plt.subplots()` function is used to initialize the figure and axes objects, while `ax.imshow()` handles the mapping of the data array to colors. The key function for adding the colorbar in its default position remains the simple call to `fig.colorbar(im)`, where `im` is the image object containing the color mapping details.

For the sake of preparation for the subsequent examples, we include the import of `make_axes_locatable`, even though it is not strictly necessary for this default placement. Furthermore, setting `np.random.seed(1)` is a fundamental practice in reproducible research, ensuring that the generated random data is consistent across every execution of the script. This baseline example serves as a contrast to the customized layouts that follow.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

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

#create chart
fig, ax = plt.subplots()
im = ax.imshow(np.random.rand(15,15))
ax.set_xlabel('x-axis label')
```

```
#add color bar using the default position (right side)
fig.colorbar(im)

plt.show()
```



Example 2: Achieving a Horizontal Colorbar Below the Chart

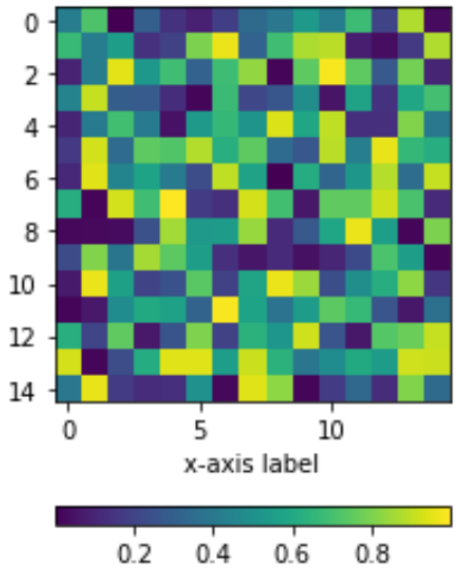
One of the most common requirements in customized visualization is placing the **colorbar** horizontally, usually below the main plot area. This configuration is often preferred when the plot is wider than it is tall, or when preserving the vertical aspect ratio of the main data presentation is critical. Relocating the colorbar to this specific position demands explicit creation and configuration of a new axis container using the geometric management capabilities of `make_axes_locatable`.

The repositioning technique involves three distinct steps after the data has been plotted using `ax.imshow()`: First, we instantiate the divider object using `divider = make_axes_locatable(ax)`. Second, we utilize the divider to define the geometry of the new colorbar axis, which we name `cax`. Crucially, to place the colorbar below the main axis, we call `divider.new_vertical()`. While this function name might seem misleading for a horizontal colorbar, it instructs the divider to create a new axis along the vertical dimension (either above or below the original axis).

Within the `new_vertical()` call, we use the parameter `pack_start=True`. This forces the new axis to be positioned at the "start" of the vertical packing order, which translates to the bottom of the plot area. We define the colorbar's height relative to the main plot's height using `size='5%'`, and we control the separation distance with `pad=0.6`. Finally, the new axis container `cax` is formally added to the figure using `fig.add_axes(cax)`. The last step involves calling `fig.colorbar()`, passing

both the image mapping (`im`) and the target axis container (`cax`), along with the mandatory `orientation='horizontal'` parameter to render the scale correctly.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

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

#create chart
fig, ax = plt.subplots()
im = ax.imshow(np.random.rand(15,15))
ax.set_xlabel('x-axis label')

#add color bar below chart
divider = make_axes_locatable(ax)
cax = divider.new_vertical(size='5%', pad=0.6, pack_start=True)
fig.add_axes(cax)
fig.colorbar(im, cax=cax, orientation='horizontal')

plt.show()
```



A deeper understanding of the `pad` and `size` arguments is essential for fine-tuning the layout. The **pad** argument dictates the buffer space, measured in inches or fractions of the plot size, between the edge of the main plot's x-axis and the top edge of the newly created colorbar axis. A

larger `pad` value increases this gap, which is crucial for preventing overlap with tick labels or axis titles. Conversely, the **size** argument controls the dimension of the new axis that is perpendicular to the boundary--in this horizontal placement, it determines the height of the colorbar itself. Specifying size as a percentage ensures that the colorbar scales proportionally with the main visualization.

Example 3: Positioning the Colorbar Above the Chart

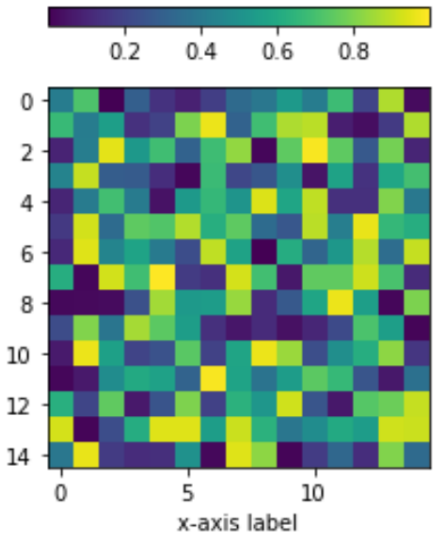
Relocating the **colorbar** to a horizontal position above the main chart requires a minor but crucial modification to the parameters used in the previous example. The underlying structure--creating the divider, defining a new axis, and specifying horizontal orientation--remains consistent, but the directional packing order must be adjusted. Placing a horizontal colorbar at the top is useful when the figure title or other metadata is positioned high, or when the bottom margin needs to be maximized for footnotes or detailed axis labels.

The key to positioning the colorbar above the plot lies in understanding the default behavior of the `new_vertical` method within the AxesGrid locator framework. When this function is called, it attempts to place the new axis at the "start" of the vertical packing, which, by convention, defaults to the top boundary of the existing axis unless otherwise specified. Therefore, unlike Example 2, we deliberately omit the `pack_start=True` parameter. By allowing `pack_start` to default to `False`, the `new_vertical` function creates the new axis (`cax`) immediately above the main plot axis (`ax`).

In this iteration, we use `pad=0.4`, a slightly smaller separation than the previous example. This demonstrates the iterative nature of layout design; developers can continually adjust the padding value to achieve optimal visual balance and ensure the color scale is neither too crowded nor too distant from the primary data display. The rest of the setup--creating the divider using `make_axes_locatable(ax)`, adding the new axis via `fig.add_axes(cax)`, and ensuring `orientation='horizontal'` in `fig.colorbar`--follows the robust methodology established for advanced positional control using the Numpy array data.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

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

#create chart
fig, ax = plt.subplots()
im = ax.imshow(np.random.rand(15,15))
```

```
ax.set_xlabel('x-axis label')

#add color bar above chart
divider = make_axes_locatable(ax)
cax = divider.new_vertical(size='5%', pad=0.4)
fig.add_axes(cax)
fig.colorbar(im, cax=cax, orientation='horizontal')

plt.show()
```



Summary of Key Parameters for Colorbar Positioning

Effective relocation of the Matplotlib **colorbar** is fundamentally dependent on mastering the precise interaction of parameters within the `make_axes_locatable` workflow, provided by the AxesGrid toolkit. The combined effects of size, padding, and the packing direction determine the final spatial relationship between the color scale and the main visualization axes. Understanding these arguments is critical for any developer seeking reliable, sophisticated layout control.

The most critical arguments governing spatial control are summarized below, categorized by the function they affect:

`make_axes_locatable(ax)`: This foundational step must be performed first. It initializes the geometric manager, returning a divider object that understands the boundaries of the primary axis (`ax`) and can calculate appropriate positions for adjacent elements.
`new_vertical()` vs. `new_horizontal()`: These methods define the orientation of the boundary along which the new axis will be placed. To position a colorbar above or below the plot, regardless

of whether the color scale is horizontal or vertical, `new_vertical()` is used. Conversely, `new_horizontal()` is used for placing the colorbar to the left or right of the plot.

`size`: This parameter governs the dimension of the new axis that runs perpendicular to the placement boundary. For a horizontal colorbar placed via `new_vertical()`, `size` controls its height (e.g., `'5%'`). For a vertical colorbar placed via `new_horizontal()`, `size` controls its width. It is best practice to specify this as a percentage of the main axis dimension for automatic scaling.

`pad`: This value explicitly sets the buffer space between the edge of the original axis and the closest edge of the new colorbar axis. Careful adjustment of `pad` is necessary to accommodate large tick labels or axis labels, preventing visual overlap and ensuring readability.

`pack_start`: This Boolean flag dictates the placement direction relative to the main axis. When using `new_vertical()`, setting `pack_start=True` forces the colorbar to the bottom (the start of the vertical packing). Setting it to `False` (the default) places it at the top. For placements using `new_horizontal()`, `pack_start=True` places the colorbar on the left.

`orientation`: This parameter, passed directly to the `fig.colorbar()` function, specifies whether the color scale itself is rendered horizontally or vertically. It must be logically aligned with the geometric placement defined by the divider methods (e.g., horizontal orientation is mandatory when the colorbar is placed above or below the plot).

By meticulously controlling these six parameters, visualization experts can ensure that auxiliary elements like the **colorbar** function optimally, enhancing the clarity and analytical impact of their scientific figures. The utilization of `make_axes_locatable` provides a robust, reusable, and highly reliable methodology for implementing complex, customized layouts in Matplotlib, moving far beyond its built-in default capabilities. This precision is invaluable when creating professional, high-fidelity graphics.

You can find more Matplotlib tutorials here.