How to Easily Reposition Your Seaborn Plot Legend

Authored by

stats writer

December 6, 2025

Data visualization is a critical aspect of data analysis, and Seaborn, a powerful Python library built upon Matplotlib, provides excellent tools for creating informative and aesthetically pleasing statistical graphics. Central to the clarity of any complex plot--especially those using the `hue` parameter--is the legend() function. The legend acts as a key, explaining the meaning of different colors, markers, or line styles used in the visualization.

However, the default placement of the legend, often determined automatically by the underlying Matplotlib functionality, can sometimes obstruct crucial data points, significantly reducing the plot's effectiveness. Mastering the positioning of the legend is therefore essential for generating presentation-ready visualizations. In Seaborn and Matplotlib, legend position is primarily controlled using two parameters within the `plt.legend()` call: the positional keyword **`loc`** and the bounding box anchor **`bbox_to_anchor`**.

This guide provides an expert overview of how to precisely control the legend's location, whether placing it optimally inside the plotting area or moving it completely outside the boundaries for maximum data visibility. We will explore both the simple keyword placement using `loc` and the advanced, fine-grained control offered by `bbox_to_anchor`.

Understanding Legend Control through plt.legend()

Because Seaborn is tightly integrated with Matplotlib, we primarily use the `plt.legend()` function (where `plt` is typically imported as `matplotlib.pyplot`) to manage the appearance and location of the plot legend. When a Seaborn function like `scatterplot()` or `lineplot()` includes a `hue` variable, it automatically generates the legend entries, which can then be manipulated using `plt.legend()`.

The simplest method to reposition the legend is by utilizing the **`loc`** parameter. This parameter accepts various strings or numeric codes that map to predefined locations within the axes bounding box. By default, if the `loc` parameter is omitted, Matplotlib attempts to use `loc='best'`. The `'best'` setting is an intelligent heuristic that evaluates several standard positions and selects the one that results in the least overlap with existing data points, ensuring maximum data visibility.

For example, to explicitly place the legend in the upper right corner of the plotting area, the following concise syntax is employed:

plt.legend(loc='upper right')

While `'best'` is convenient, manual specification ensures the legend always appears exactly where intended, which is essential for standardization across multiple visualizations or for complex plots where `'best'` might make a suboptimal choice. The ability to precisely dictate legend

placement improves the overall professional quality and readability of the final graphic.

Utilizing the 'loc' Parameter for Internal Positioning

The `loc` parameter offers eleven standard locations based on the corners, edges, and center of the axes. These keyword arguments are highly intuitive and serve as the most common method for adjusting internal legend placement. It is crucial to remember that these locations are defined relative to the axes of the plot itself, meaning the legend will always reside within the data visualization area.

Here is the comprehensive list of available string values that can be passed to the `loc` parameter to control internal legend placement:

upper right
upper left
lower left
lower right
right
center left
center right
lower center
upper center
center
best (the default, algorithmically determined position)

Choosing the correct `loc` value often depends on the density of the data and where open space is naturally available within the plot. For instance, if the data is concentrated in the lower-left quadrant, setting `loc='upper right'` is usually the ideal choice to avoid obscuring any data markers. This simple keyword-based approach provides substantial control without requiring complex numerical coordinate system mapping.

Implementing Internal Legend Placement: Example 1

To illustrate the use of the `loc` parameter, we will generate a sample Seaborn scatterplot using a small DataFrame and demonstrate how different `loc` values instantly reposition the legend within the plot boundaries. This example uses a fictional dataset tracking points and assists for two different teams, labeled 'A' and 'B', which forms the basis for the hue grouping.

The following code snippet sets up the data and generates the scatterplot, placing the legend specifically in the `'center right'` of the plot. Note that we also use the `title` argument within `plt.legend()` to provide a descriptive label for the hue variable.

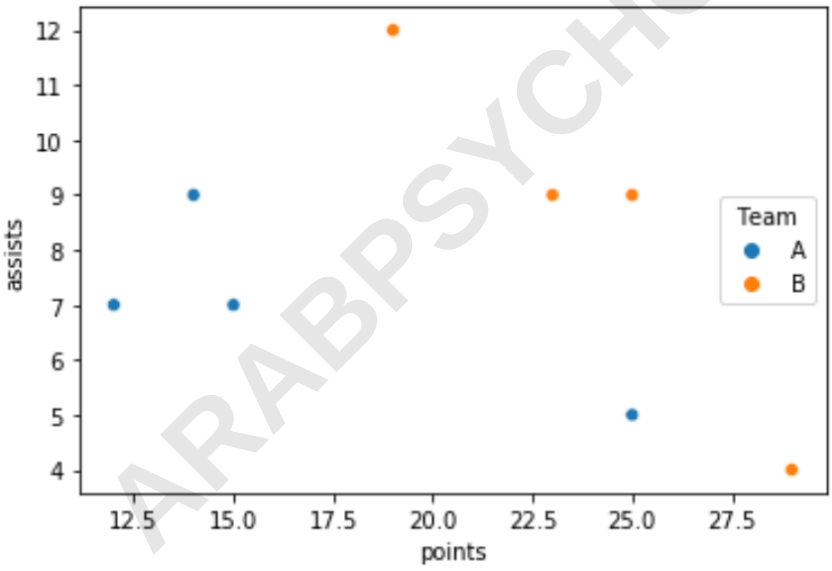
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#create fake data
df = pd.DataFrame({'points': ,
'assists': ,
'team': })

#create scatterplot
sns.scatterplot(data=df, x='points', y='assists', hue='team')

#place legend in center right of plot
plt.legend(loc='center right', title='Team')

The resulting plot clearly shows the legend aligned vertically along the right edge, positioned centrally. This placement is optimal if the data points are primarily concentrated on the left side of the visual field.

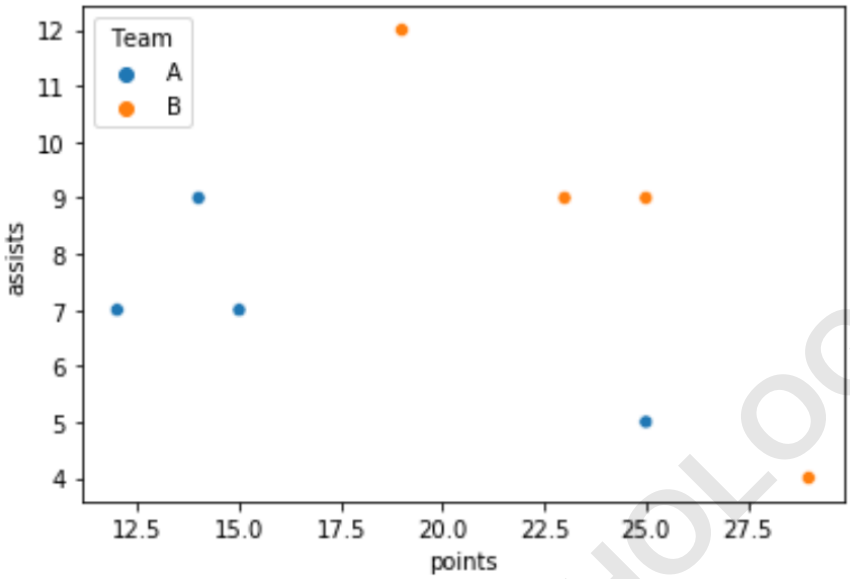


Alternatively, if we wish to place the legend in the upper left section--perhaps because the data is denser in the lower-right area--we simply adjust the `loc` parameter accordingly. This flexibility allows for rapid experimentation to find the best possible visual layout for any given dataset.

#create scatterplot
sns.scatterplot(data=df, x='points', y='assists', hue='team')

```
#place legend in upper left of plot
plt.legend(loc='upper left', title='Team')
```

This modification shifts the legend to the top-left corner, demonstrating the direct and immediate control provided by the `loc` parameter for internal legend positioning.



Advanced Placement: Introducing bbox_to_anchor

While the `loc` parameter is excellent for internal placement, there are many scenarios where the legend must be placed outside the main plot area to ensure zero data occlusion, especially in small plots or complex visualizations. For these cases, we must use the **`bbox_to_anchor`** argument alongside the `loc` parameter.

The `bbox_to_anchor` argument accepts a tuple of coordinates, typically `(x, y)`, which defines a precise point where the legend's bounding box should be anchored. This coordinate system is based on the axes, where `(0, 0)` represents the lower-left corner of the axes and `(1, 1)` represents the upper-right corner. Importantly, coordinates outside this range (e.g., `x > 1` or `y > 1`) will place the anchor point outside the plot boundaries.

When using `bbox_to_anchor`, the **`loc`** parameter changes its role. Instead of defining the overall position of the legend within the plot, `loc` now specifies which part of the *legend itself* should be placed at the `(x, y)` coordinates provided by `bbox_to_anchor`. For instance, if `bbox_to_anchor=(1.05, 1)` is used, and `loc='upper left'` is specified, it means the upper-left corner of the legend box will be positioned at `(1.05, 1)`, effectively pushing the legend outside the top-right corner of the plot.

A typical setup to place the legend outside the top right corner uses coordinates slightly greater than `(1, 1)`, such as `(1.02, 1)`, combined with `loc='upper left'`. This combination ensures that the legend is anchored just outside the plot border, extending neatly into the margin space. The structure often looks like this:

plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

The optional `borderaxespad=0` parameter, often included in these calls, specifies the padding between the axes and the border of the legend. Setting it to zero removes any additional automatic padding, allowing for extremely precise positioning relative to the specified coordinates.

Implementing External Legend Placement: Example 2

Using the same dataset established in Example 1, we now demonstrate how to place the legend entirely outside the plotting area. This maximizes the space available for the data visualization and is often preferred for reports and presentations where data clarity is paramount. We will first position the legend outside the top right corner.

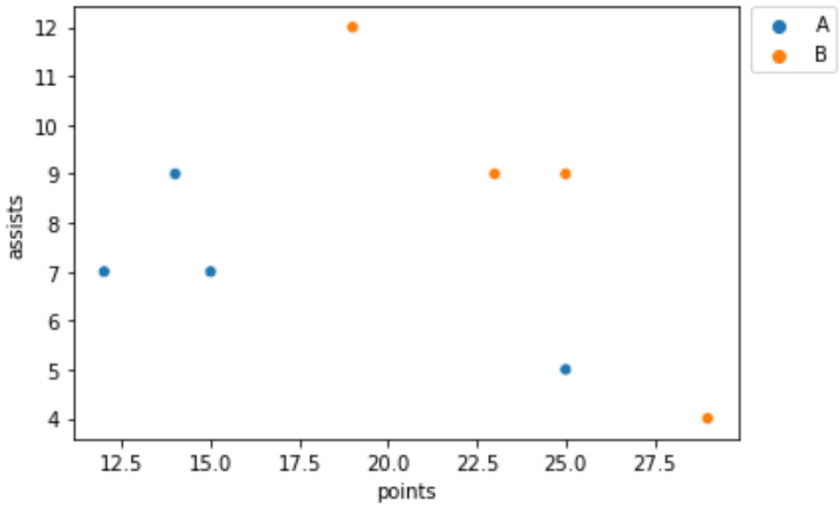
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#create fake data
df = pd.DataFrame({'points': ,
'assists': ,
'team': })

#create scatterplot
sns.scatterplot(data=df, x='points', y='assists', hue='team')

#place legend outside top right corner of plot
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)

In this code, `bbox_to_anchor=(1.02, 1)` places the anchor point just outside the right border (1.02) and exactly at the top border (1). Setting `loc='upper left'` instructs the upper-left corner of the legend itself to align with this anchor point, ensuring the entire legend box extends outward to the right.



Adjusting External Legend Positioning

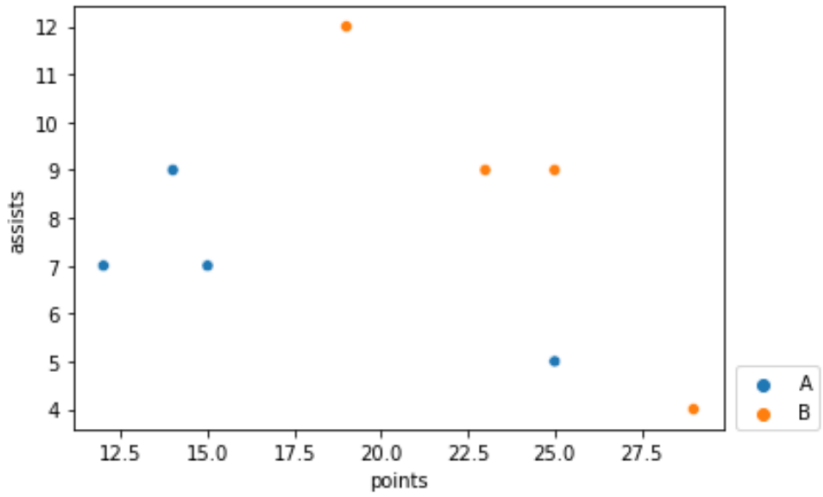
The true power of `bbox_to_anchor` lies in its flexibility to place the legend anywhere around the plot boundary. For instance, if the space is clearer below the plot, we might want to anchor the legend near the bottom right. To achieve this, we need to adjust the coordinates provided to `bbox_to_anchor`.

To place the legend outside the bottom right corner, we still need an X-coordinate greater than 1 (e.g., 1.02) to move it rightward. However, we must choose a Y-coordinate close to the bottom of the axis, such as 0.15. Combined with `loc='upper left'`, this pulls the legend down and right, positioning it neatly in the lower margin.

#create scatterplot
sns.scatterplot(data=df, x='points', y='assists', hue='team')

#place legend outside bottom right corner of plot
plt.legend(bbox_to_anchor=(1.02, 0.15), loc='upper left', borderaxespad=0)

By shifting the Y-coordinate to 0.15, the anchor point is now near the bottom of the vertical axis, providing a clean placement in the bottom-right margin space.



Summary of Positioning Techniques

Choosing between `loc` and `bbox_to_anchor` depends entirely on whether the visual design permits the legend to overlap data (`loc` only) or requires absolute separation (`bbox_to_anchor`). For quick, internal adjustments, the `loc` keywords are sufficient and highly recommended.

When external placement is necessary, a thoughtful combination of `bbox_to_anchor=(x, y)` and the `loc` parameter is required. Remember that `bbox_to_anchor` defines the anchor point in the axes coordinate system (0 to 1), and `loc` defines which part of the legend is attached to that anchor point. Experimentation with these coordinate values is key to achieving perfect alignment in the desired margin space.

Refer to the Matplotlib documentation for a detailed explanation of the `bbox_to_anchor()` argument and the various coordinate transformations possible, especially when working with multiple subplots or complex figure layouts.

Conclusion

Effective legend placement is non-negotiable for producing high-quality data visualizations. By leveraging the flexibility offered by Matplotlib's `plt.legend()` function, Seaborn users can precisely control where their legends appear. Whether using descriptive strings with the `loc` parameter for internal positioning or mastering the axes coordinate system via `bbox_to_anchor` for external placement, these techniques ensure that the legend enhances, rather than detracts from, the underlying data story. Employing these methods guarantees clear, compelling, and professional-grade statistical graphics.

For more detailed customization, such as altering the size or style of the legend text, additional

parameters within the legend() function can be explored.

How to Change Legend Font Size in a Seaborn Plot