How to Easily Change Legend Font Size in Seaborn Plots

Authored by

stats writer

December 6, 2025

The key to mastering powerful Python visualization libraries like Seaborn lies not just in generating plots, but in refining their aesthetic details. One common requirement in professional data visualization is customizing the legend, particularly its font size, to ensure optimal readability and visual hierarchy. Since Seaborn is built atop Matplotlib, the most effective method for manipulating legend properties is through the `plt.legend()` function, inherited from Matplotlib's powerful API.

Specifically, you utilize the `fontsize` and `title_fontsize` optional arguments when calling this function. This approach provides precise control over both the descriptive labels and the primary title of the legend box, ensuring your visualizations are both informative and aesthetically polished.

The best way to change the font size of a legend in a seaborn plot is to use the `fontsize` optional argument when calling a legend function. For example, `sns.legend(fontsize=10)` would set the font size of the legend to 10.

Leveraging Matplotlib's plt.legend() for Fine Control

When working with Seaborn, which excels at high-level statistical plotting, customization often requires dropping down to the underlying Matplotlib layer. The legend object is inherently managed by Matplotlib, meaning the primary mechanism for adjusting its appearance is the `plt.legend()` function. This function allows developers and analysts to specify various parameters that dictate how the legend box, its labels, and its title are rendered on the canvas. Understanding the role of this function is fundamental to achieving high-quality visual outputs in Python.

The core syntax for controlling the textual elements within the legend involves two specific parameters: `fontsize` and `title_fontsize`. These parameters accept either numeric values (representing font sizes in points) or specific string aliases, offering great flexibility depending on the desired level of precision and scaling. Using `plt.legend()` after a Seaborn plotting function allows us to override the default aesthetic settings applied by the statistical library.

You can use the following syntax to change the font size within a legend of a seaborn plot:

plt.legend(title='Team', fontsize='10', title_fontsize='14')

The **fontsize** argument specifies the font size for the labels in the legend, which typically represent the categories or hues plotted. Conversely, the **title_fontsize** specifies the font size for the title of the legend, usually the name of the variable used to differentiate the plot elements (e.g., the `hue` variable).

Detailed Explanation of Font Size Parameters

The distinction between `fontsize` and `title_fontsize` is crucial for maintaining visual hierarchy within the legend box. The `title_fontsize` parameter controls the size of the overarching label, which serves as the primary identifier for the categories displayed. This title provides context for the entire legend block. The `fontsize` parameter, on the other hand, applies to the individual labels corresponding to each category or marker within the legend.

For optimal readability and professional presentation, the title font size is often set slightly larger than the label font size to emphasize its importance, mirroring standard typography practices in charts and graphs. Both parameters offer two main ways to define the size: by using integer or float values, which provide absolute size definitions in points (e.g., 12pt), or by utilizing predefined string aliases, which offer relative scaling based on the plot's overall context and style settings.

Using numerical values (e.g., 10, 12, 14) grants explicit control, which is useful when precise alignment with other text elements in the plot (like axis labels) is necessary or when creating publication-ready figures that mandate specific font dimensions. We will explore both methods in the following practical examples.

The following example shows how to use this function in practice.

Practical Example 1: Applying Numeric Font Sizes

To effectively demonstrate how these parameters modify the legend's appearance, we will generate a simple scatterplot using sample data structured with Pandas and visualized using Seaborn. This approach highlights the necessity of using plt.legend() to manipulate elements created by Seaborn's high-level functions.

In this demonstration, we explicitly set the label font size to 10 points and the legend title font size to 14 points. This deliberate choice ensures the legend title is clearly emphasized and visually separated from the category labels, improving the overall structure of the data visualization. This technique is standard practice when creating publication-quality graphics where exact font sizes are mandated by journals or corporate style guides.

The steps involve setting up the environment, creating a mock dataset, generating the statistical plot using the `hue` parameter to define the legend entries, and finally calling plt.legend() to override the default font settings.

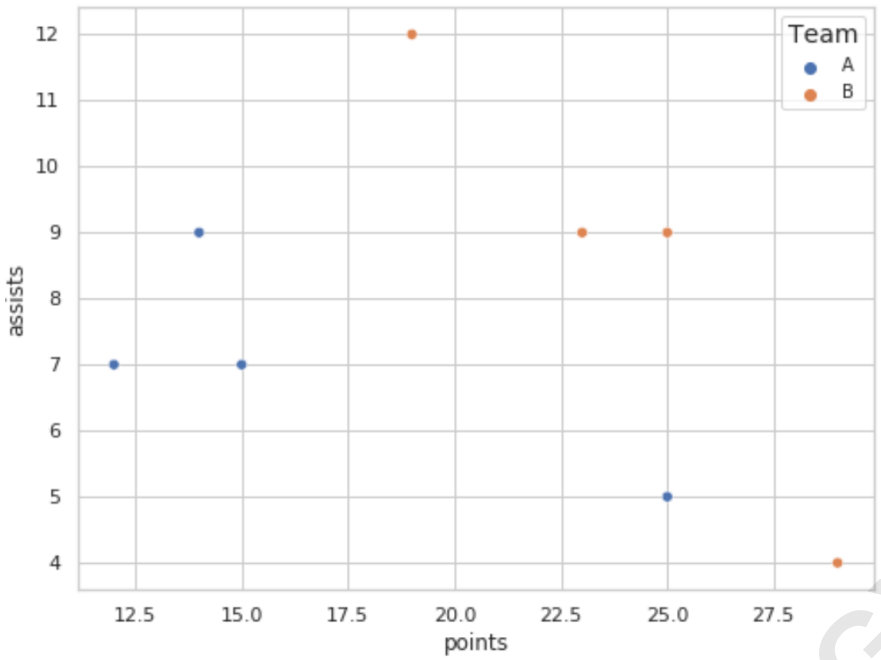
The following code shows how to create a scatterplot in Seaborn and specify the font size for both the labels and the title within the legend:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style('whitegrid')

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

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

#add legend
plt.legend(title='Team', fontsize='10', title_fontsize='14')
```

Visual Confirmation of Numeric Size Adjustments

Upon executing the code, the generated plot confirms the specified font adjustments. The legend title, "Team," is visually distinct and dominant due to its 14-point size, while the individual team labels ("A" and "B") are set slightly smaller at 10 points. This intentional contrast aids the viewer in quickly identifying the purpose of the legend and relating the categories back to the plotted data.

It is important to remember that when using Matplotlib commands like `plt.legend()` in a scripting environment, you must ensure the plot is explicitly displayed (e.g., using `plt.show()`) if not running in an interactive session like a Jupyter Notebook. The output image below demonstrates the successful result of applying these numeric size customizations.



Utilizing Named String Values for Relative Scaling

While numeric sizes offer precise control, string aliases offer flexibility by scaling the font size relative to the overall plot size or the default font configuration of the chosen Seaborn style (such as 'whitegrid'). These string arguments are inherited from CSS font size specifications and are highly valuable when you want your plot to maintain a consistent visual hierarchy even if the plot size changes, or if you are using a dynamic environment where absolute point sizes are less critical.

The ability to use these named sizes means that you can easily make the legend text significantly smaller or larger than the default without needing to manually test different point values. This greatly simplifies the process of achieving suitable contrast and scale across different output devices or presentation formats. The acceptable named values are highly descriptive and easy to implement, providing standard scaling increments.

The font size arguments can also take on the following values:

xx-small
x-small
small
medium
large
x-large
xx-large

These values correspond to relative steps, allowing the user to quickly adjust text size based on proportional increases or decreases relative to the default setting.

Practical Example 2: Implementing Named String Sizes

Using string aliases is particularly suitable for rapid prototyping or when you need a clear step-change in size without specifying an exact point value. In this second example, we will modify the same scatterplot but replace the numeric values with string aliases. We will set the label font size to `medium` (which is often close to the default size, or slightly adjusted by the style) and the title font size to `x-large`, creating an even more pronounced visual separation between the title and the labels.

This demonstrates the adaptability of the plt.legend() function. Regardless of whether you pass an integer, a float, or a predefined string, Matplotlib handles the necessary rendering calculations to ensure the font sizes are applied correctly according to the defined parameters. The structured nature of the aliases ensures proportional scaling is maintained across different visualization contexts.

The following example shows how to use these arguments in practice:

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style('whitegrid')

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

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

#add legend
plt.legend(title='Team', fontsize='medium', title_fontsize='x-large')

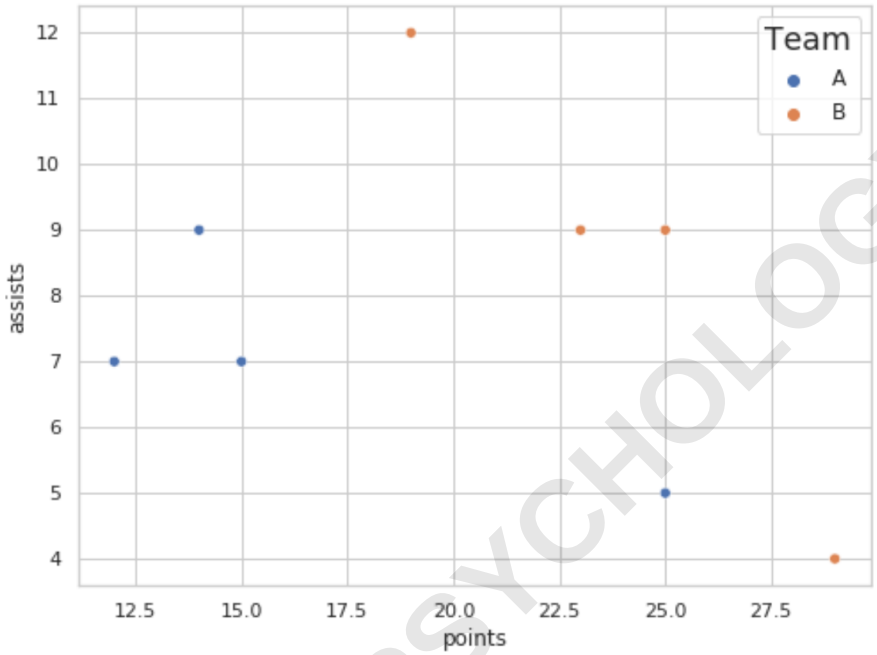
Comparing Named and Numeric Font Outputs

When comparing the visual output of the two examples, the immediate visual impact of using `x-large` for the title font is evident. It generates a title that is significantly larger than the surrounding text, providing strong visual emphasis. Choosing between numeric values and string aliases

depends heavily on the project's requirements for precision and adaptability:

If **absolute precision** (e.g., 10pt for print media) is required, use **numeric values**.
If **relative scaling** and consistency across different style presets are prioritized, use **string aliases**.

Both methods effectively achieve the goal of customizing the legend font size in a clean, maintainable manner within the Python visualization environment. Ensuring the legend is readable and visually balanced is paramount to effective data visualization.



Further Customization and Resources

While adjusting font size is a critical aspect of plot refinement, the plt.legend() function offers a wealth of other customization options that further enhance plot clarity. These include controlling the legend's precise placement on the canvas (via the `loc` parameter), setting the number of columns (`ncol`), and modifying the frame properties (e.g., controlling the background visibility with `frameon` or setting transparency with `framealpha`). Mastering these elements allows for complete control over the legend, ensuring it complements the main plot without obstructing critical data points.

For those requiring deeper dives into the capabilities of this core function, the official Matplotlib documentation provides an exhaustive overview of every parameter and their interaction with Matplotlib objects. Understanding the documentation is key to unlocking advanced features for high-end statistical plotting when integrating with libraries like Seaborn.

Reference the Matplotlib Documentation for an in-depth explanation of the **plt.legend()** function.

Related Seaborn and Matplotlib Tutorials

For users looking to expand their knowledge beyond font customization, several other common tasks related to legends and plot adjustments often arise when using Seaborn. These topics include manipulating the position of the legend relative to the plot area, particularly when the default location interferes with the data display, and adjusting overall plot aesthetics for publication.

The following tutorials explain how to perform other common tasks in seaborn:

How to Place Legend Outside a Seaborn Plot