How to Change Legend Font Size in Matplotlib

Authored by

stats writer

December 18, 2025

Effective data visualization relies heavily on clarity, and the legend is a critical component for interpreting graphical output. When using Matplotlib, the standard plotting library in Python, controlling visual elements like the font size of the legend text is essential for ensuring readability, especially when preparing figures for publication or presentation. A poorly sized legend can distract the viewer or, worse, make the plot unusable. Therefore, mastering the methods for adjusting font size is a fundamental skill for any data scientist or analyst utilizing this powerful library.

The primary mechanism for customizing the legend's appearance is through the built-in functions provided by the `pyplot` module. While advanced users might utilize the comprehensive prop parameter--which accepts a `matplotlib.font_manager.FontProperties` instance to set detailed attributes like font weight, family, and style--the quickest and most common approach for simple font size modification involves directly setting the `fontsize` argument within the `plt.legend()` call. This tutorial focuses on both the straightforward `fontsize` methods and briefly touches upon the flexibility offered by the **prop parameter** for comprehensive styling.

Before diving into specific size manipulation techniques, it is useful to understand how Matplotlib integrates a legend into a basic plot. A legend requires that individual plotted elements (lines, bars, etc.) are assigned a `label` attribute during their creation. Once labeled, calling the `plt.legend()` function aggregates these labels and displays them, defaulting to the current figure's font settings. We will now review the basic structure for adding this element.

Implementing a Basic Matplotlib Legend

To begin any legend customization process, we must first ensure a basic legend is present on the plot. This requires importing the necessary library and defining lines or objects with clear labels. Although the following code snippet simply demonstrates the inclusion of the legend function itself, it serves as the necessary foundation upon which all font adjustments are made. Note the standard convention of importing `matplotlib.pyplot` as `plt`, which simplifies subsequent function calls.

```
import matplotlib.pyplot as plt

#add legend to plot
plt.legend()
```

Once this function is successfully called, Matplotlib automatically generates the legend box, positioning it intelligently on the plot based on the default settings. The subsequent steps involve passing specific arguments to this function to override those defaults, specifically targeting the text size. We will explore two primary, highly efficient methods for achieving precise font control.

Method 1: Direct Font Sizing Using Numerical Values

The most direct and precise way to control the size of the legend text is by supplying an integer or float value to the `fontsize` parameter within the `plt.legend()` function. This numerical value corresponds directly to the desired font size measured in points (pt). Using a numerical input offers maximum control, allowing developers to ensure visual consistency across multiple plots or adhere to specific presentation standards that mandate exact point sizes.

When choosing a numerical size, standard practice suggests values typically range between 8 (for small, detailed plots) and 20 (for large, accessible presentations). Matplotlib interprets this number as an absolute size, meaning the legend text will maintain that size regardless of the overall figure size scaling, unless explicit scaling is applied elsewhere. To implement this technique, simply set the `fontsize` argument equal to the desired point size, as shown in the abbreviated example below, where the text is dramatically enlarged for demonstration purposes.

plt.legend(fontsize=18)

This method is highly recommended when you need guaranteed consistency or when integrating the plot into a larger document where exact typographic specifications are required. For instance, academic papers often require a consistent 10pt or 12pt font across all textual elements, including axis labels, titles, and legends. Employing this numerical approach ensures adherence to such stringent guidelines effortlessly.

Example 1: Specifying Legend Font Size Using a Number

The following complete Python script demonstrates the creation of two line plots, each assigned a label. We then call `plt.legend()`, specifying a font size of 18 points. This significant increase from the default size clearly illustrates the effectiveness of the numerical method in enhancing the visual presence of the legend text. Notice how all elements within the legend box--the text and the corresponding markers--are scaled appropriately.

import matplotlib.pyplot as plt

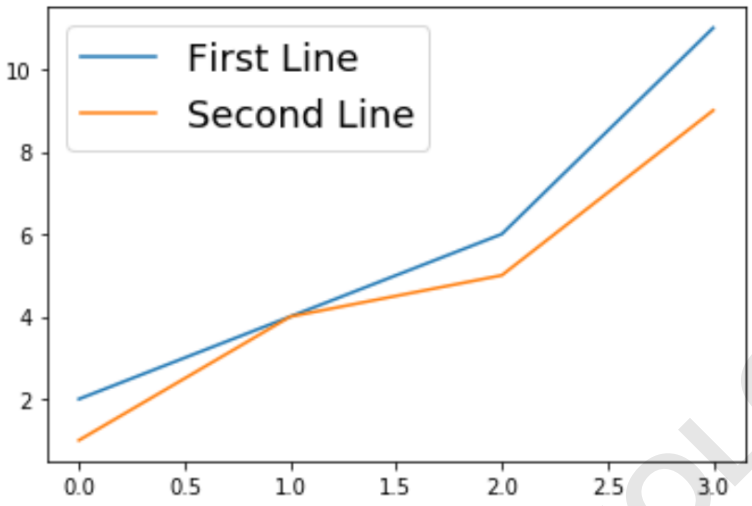
#create data
plt.plot(, label="First Line")
plt.plot(, label="Second Line")

#add legend
plt.legend(fontsize=18)

#show plot

plt.show()

Upon executing the code above, the output visualization clearly shows the impact of the numerical font size assignment. The legend text is significantly larger, improving visibility and accessibility for the viewer. This demonstrates the power of direct numerical control over typographic elements within Matplotlib figures.



Method 2: Leveraging Predefined String Sizes for Legends

While numerical sizing provides absolute control, Matplotlib also supports relative font sizing using predefined string keywords. This method is exceptionally useful when you want the legend font to be sized relative to the default font size of the surrounding plot environment, or relative to the standard font sizes used in common typesetting systems. Using strings ensures that the legend text scales somewhat dynamically, maintaining a visually pleasing relationship with other text elements on the graph.

To use this method, you pass a string corresponding to a standardized size descriptor directly to the `fontsize` parameter. These string values are based on conventional CSS font sizing standards, which are mapped internally by Matplotlib to specific point sizes relative to a base font size. This approach offers a simple, declarative way to adjust size without needing to know the exact point measurement required.

plt.legend(fontsize="small")

Choosing a string size like "small" or "large" is often preferred in exploratory data analysis where absolute precision is less critical than rapid, readable output. It allows for quick adjustments that

look proportional, enhancing the overall aesthetic balance of the visualization with minimal effort. This avoids the need for constant numerical experimentation to find the perfect point size.

Available String Options for Relative Font Sizing

Matplotlib provides a comprehensive range of string options, derived from standard typographical scales, allowing for fine-grained relative control over the font size. These options span from extremely small to extremely large, giving the user flexibility in how prominent the legend should be compared to the rest of the plot annotations. It is important to remember that these are relative adjustments; "medium" usually corresponds to the default system font size.

The available string arguments that can be passed to the `fontsize` parameter are listed below. Experimenting with these options allows you to quickly gauge the visual impact of each relative size setting on your specific plot dimensions and target output format:

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

Understanding these options is crucial for maximizing readability across different display environments. For instance, if a plot is intended for a large projector screen, using "x-large" might be appropriate, whereas "x-small" might be suitable for compact embedded visualizations.

Practical Application of String-Based Font Sizing

To see the string-based method in action, we utilize the same dataset and plotting configuration used in the numerical example. However, instead of passing an integer, we apply the string "small" to the `fontsize` parameter. This demonstrates how relative sizing adjusts the legend text size down from the default, making it less visually dominant compared to the plot lines themselves, which can be desirable in cases where the focus must remain strictly on the data.

The immediate benefit of using string sizes is the quick, descriptive adjustment they provide. If you determine the default legend size is slightly too large, simply applying "small" often achieves the desired subtle reduction without requiring precise measurements. This method streamlines the styling process, especially during the initial drafting phases of data visualization in Python.

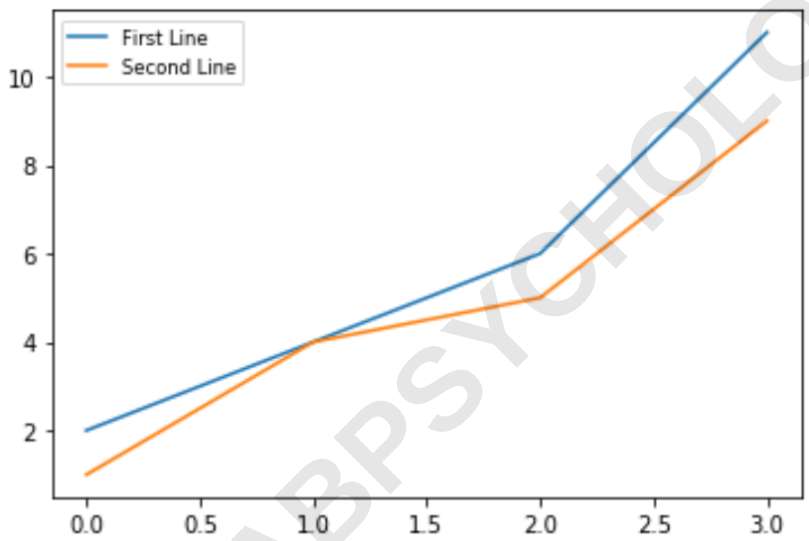
import matplotlib.pyplot as plt

```
#create data
plt.plot(, label="First Line")
plt.plot(, label="Second Line")

#add legend
plt.legend(fontsize="small")

#show plot
plt.show()
```

The resulting plot clearly shows the legend text reduced in size relative to the rest of the figure components, confirming that the string parameter successfully implemented the intended change. This exemplifies a versatile technique for balancing visual hierarchy within complex data graphics generated by Matplotlib.



Advanced Customization Using the 'prop' Parameter

While the `fontsize` parameter offers a rapid solution for size adjustment, highly tailored visualizations often require controlling other typographic details such as font family, weight, and style (e.g., italic or oblique). For this level of comprehensive control, Matplotlib provides the prop parameter within the `plt.legend()` function. This parameter expects an instance of `matplotlib.font_manager.FontProperties`, a powerful object that encapsulates all necessary font characteristics.

The advantage of using the `prop` object is its ability to define multiple characteristics simultaneously. For example, instead of just setting the size numerically, you can define the font to

be 14pt, bold, and using a specific typeface like 'serif' or 'sans-serif'. This ensures complete thematic integration of the legend text with the overall document or presentation style. Although more verbose than simply using `fontsize`, this method provides unparalleled flexibility for professional-grade output.

A common mistake when using `prop` is attempting to pass a dictionary of properties directly. Instead, one must explicitly create the `FontProperties` object first, passing the desired arguments (like `size`, `weight`, and `family`) during its instantiation, and then supply this object to the prop parameter of the legend function. However, if the only property being set is the size, Matplotlib allows a shorthand: passing a dictionary containing only the 'size' key directly to `prop`, or better yet, simply using the dedicated `fontsize` parameter we explored earlier, which is syntactically cleaner for size-only adjustments.

Summary and Best Practices for Legend Styling

Customizing the legend font size in Matplotlib is a straightforward task, primarily accomplished through the `fontsize` parameter within the `plt.legend()` function. Developers have the choice between specifying an exact numerical value for absolute control or using relative string keywords for dynamic scaling. Both methods are highly effective and should be chosen based on the specific demands of the visualization project.

Best Practice Tip 1: Consistency is Key. When creating a series of plots, ensure that the legend font size remains consistent across all figures. Inconsistent sizing can lead to visual fatigue and confusion. If using numerical sizes, define a constant variable for the preferred size (e.g., `LEGEND_FONT_SIZE = 10`) and reuse it across all plotting scripts to guarantee uniformity.

Best Practice Tip 2: Prioritize Readability. While it might be tempting to use extremely small text to save space, the primary goal of any data visualization is successful communication. Always test your plots at the intended display resolution (screen, print, projection) to ensure the legend text is clearly legible without requiring the viewer to strain. If space is tight, consider repositioning the legend (using the `loc` parameter) before drastically reducing the font size.

By mastering these two simple techniques--numerical sizing and relative string sizing--data professionals can significantly improve the clarity and professional appearance of their Matplotlib visualizations, transforming raw data into highly effective communication tools.

How to Change Font Sizes on a Matplotlib Plot
How to Add Text to Matplotlib Plots