How to Change Font Sizes on a Matplotlib Plot?

Authored by

stats writer

December 22, 2025

When creating visualizations using Matplotlib, controlling the visual aesthetic is paramount for effective data communication. One of the most common requirements is adjusting font sizes to ensure clarity and readability across various plot elements, such as titles, axis labels, and tick values. While Matplotlib provides sensible defaults, these often need customization depending on the presentation medium--be it a high-resolution print or a small slide deck figure. This guide, written for the intermediate Python user, provides a comprehensive overview of how to manage and modify these textual properties effectively and globally.

The standard process for fine-tuning text attributes involves utilizing Matplotlib's powerful configuration system. Before displaying your final figure using the `matplotlib.pyplot.show()` function, you must initialize your plot and then apply specific configuration commands. Although you can individually set font parameters for elements like axis labels using methods such as `Axes.set_xlabel()` or `Axes.set_ylabel()`, the most efficient and clean way to manage font sizes across multiple components is by employing global configuration settings. This approach ensures consistency and simplifies code maintenance, especially when dealing with complex visualizations or multiple plots.

The Global Configuration Approach: Using plt.rc()

In Matplotlib, configuration settings are managed through the runtime configuration system, accessible primarily via the `matplotlib.pyplot.rc()` function. This function allows users to set properties for various graphical elements universally, overriding the library's defaults. This method is highly favored because it provides a centralized mechanism for managing stylistic choices, ensuring that changes propagate uniformly across all subsequent plot elements created in the session.

Using `plt.rc()` simplifies the process significantly compared to applying font parameters to each individual object. Instead of passing keyword arguments like `fontsize=20` repeatedly to functions such as `plt.title()` or `plt.xlabel()`, you can define a default size once. This level of abstraction is particularly useful when developing a consistent visual style for a series of plots in a report or publication. The parameters passed to `plt.rc()` correspond to different groups of elements, such as `'axes'` for titles and labels, `'xtick'` and `'ytick'` for axis ticks, and `'legend'` for the legend text.

The following example demonstrates how to set a consistent font size for various components of a plot using the `plt.rc()` function. Notice how specific groups (like `'axes'` or `'xtick'`) are targeted to control their respective text attributes. If you only specify `'font'`, it acts as a global baseline for text size, which is convenient for scaling all elements simultaneously. However, for fine-grained control, targeting specific parameters is essential.

import matplotlib.pyplot as plt

```
plt.rc('font', size=10) #controls default text size
plt.rc('axes', titlesize=10) #fontsize of the title
plt.rc('axes', labelsize=10) #fontsize of the x and y labels
plt.rc('xtick', labelsize=10) #fontsize of the x tick labels
plt.rc('ytick', labelsize=10) #fontsize of the y tick labels
plt.rc('legend', fontsize=10) #fontsize of the legend
```

Understanding Matplotlib Font Control Parameters

To effectively manage typography in your visualizations, you must understand the specific parameters Matplotlib uses to categorize text elements. Matplotlib separates text properties into logical groups. For instance, the `'axes'` group contains parameters governing the appearance of the plot title and the labels on the X and Y axes. Within this group, `titlesize` and `labelsize` are the keys used to control the font size for the title and the axes labels, respectively. These keys allow precise manipulation without affecting other text elements.

The Tick labels, which are the numeric or categorical values displayed along the axes, are controlled by their own dedicated groups: `'xtick'` and `'ytick'`. Within these groups, the key `labelsize` dictates the font size. This segregation is vital because tick labels often need to be smaller than axis labels to prevent cluttering the plot area, yet they must remain legible. Similarly, the `'legend'` group controls the text size of the legend entries using the `fontsize` key. Mastering these distinct configuration groups is the foundation of high-quality Matplotlib styling.

It is important to note the hierarchy of these settings. If you set a global font size using `plt.rc('font', size=12)`, this size will apply to all text elements unless a more specific configuration overrides it. For example, if you set the global font size to 12 but then specifically set `plt.rc('axes', titlesize=18)`, the title will display at size 18 while other elements default to size 12. This hierarchical structure allows for broad adjustments followed by meticulous detail work, ensuring that your plots maintain both aesthetic appeal and data integrity.

Setting up the Base Plot for Comparison

Before demonstrating the font size modifications, we need a foundational plot to serve as our control visual. We will use a simple scatter plot, which involves a title, X and Y axis labels, and axis Tick labels. This baseline visualization will, by default, utilize Matplotlib's standard font size, which is typically set to 10 points. Observing the initial output helps illustrate the significant visual impact of the subsequent configuration changes.

We begin by importing the necessary Matplotlib module and defining simple lists for our x and y coordinates. The plotting process involves calling `plt.scatter()` to generate the points, followed

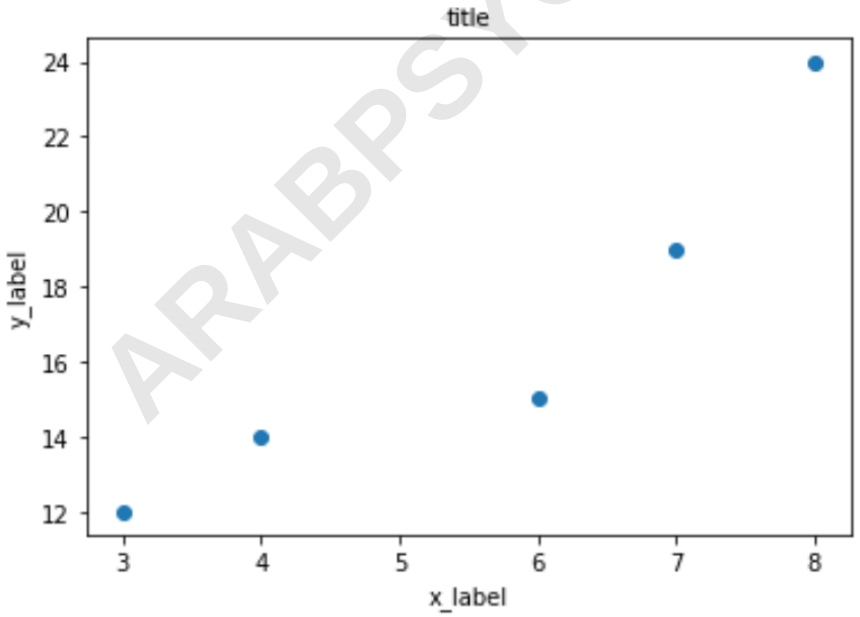
by `plt.title()`, `plt.xlabel()`, and `plt.ylabel()` to assign the necessary contextual text elements. The final step, `plt.show()`, renders the figure. The text elements in this initial plot will appear small, often necessitating an increase in font size for better presentation, especially in larger displays or publications.

Review the base code snippet below. This structure establishes the core plot components that we will manipulate throughout the subsequent examples. Pay close attention to the visual output of this default plot, as it provides the essential benchmark against which all font size adjustments will be measured. The subsequent examples will only prepend the configuration changes before this core plotting code.

import matplotlib.pyplot as plt

x =
y =

plt.scatter(x, y)
plt.title('title')
plt.xlabel('x_label')
plt.ylabel('y_label')
plt.show()



Note: *The default font size for all elements in this specific configuration is* ***10****.*

Example 1: Changing the Font Size of All Elements Simultaneously

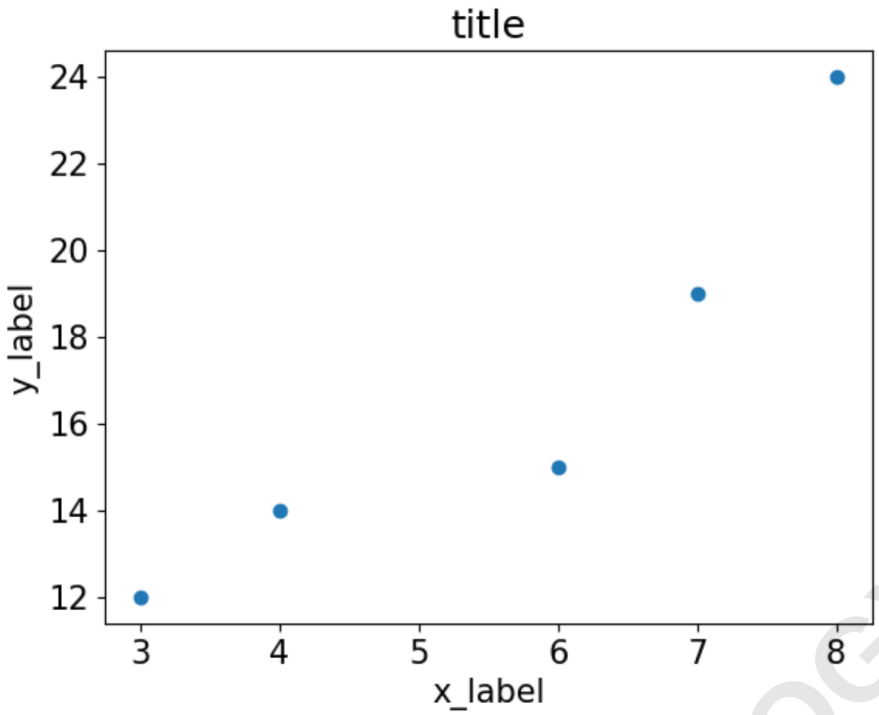
A common scenario is needing to globally scale the text size of an entire plot, perhaps when resizing the figure for a publication or a presentation slide. Instead of setting sizes individually for the title, axes, and ticks, we can use the main `'font'` parameter within `plt.rc()`. By setting the `size` attribute under the `'font'` group, we apply a single, uniform font size across all text components that haven't been explicitly overridden by more specific settings like `titlesize` or `labelsize`.

In this example, we explicitly set the default font size to 15. This single command significantly increases the legibility of the title, axis labels, and Tick labels. This method is the simplest approach for ensuring that all textual elements are easily readable at a glance. It's an excellent starting point for achieving stylistic uniformity across your visualization portfolio before applying more specific overrides where necessary. Remember, this global setting acts as the baseline for the entire session until it is reset or overridden.

Observe the implementation below. The line `plt.rc('font', size=15)` precedes the plot creation commands. This configuration ensures that every text object created afterward inherits the size 15 property, resulting in a noticeably larger and clearer visualization compared to the default size 10 plot shown previously. This technique is indispensable for rapid prototyping and quick style adjustments.

#set font of all elements to size 15
plt.rc('font', size=15)

#create plot
plt.scatter(x, y)
plt.title('title')
plt.xlabel('x_label')
plt.ylabel('y_label')
plt.show()



Example 2: Customizing the Plot Title Font Size

The plot title is often the most critical piece of text, demanding high prominence. Therefore, controlling its size independently from other text elements is crucial. Matplotlib allows us to target the title size using the `'axes'` group configuration parameter, specifically the `titlesize` key. By adjusting this parameter, we can make the title significantly larger, drawing immediate attention to the visualization's core subject without affecting the readability of the axis labels or Tick labels.

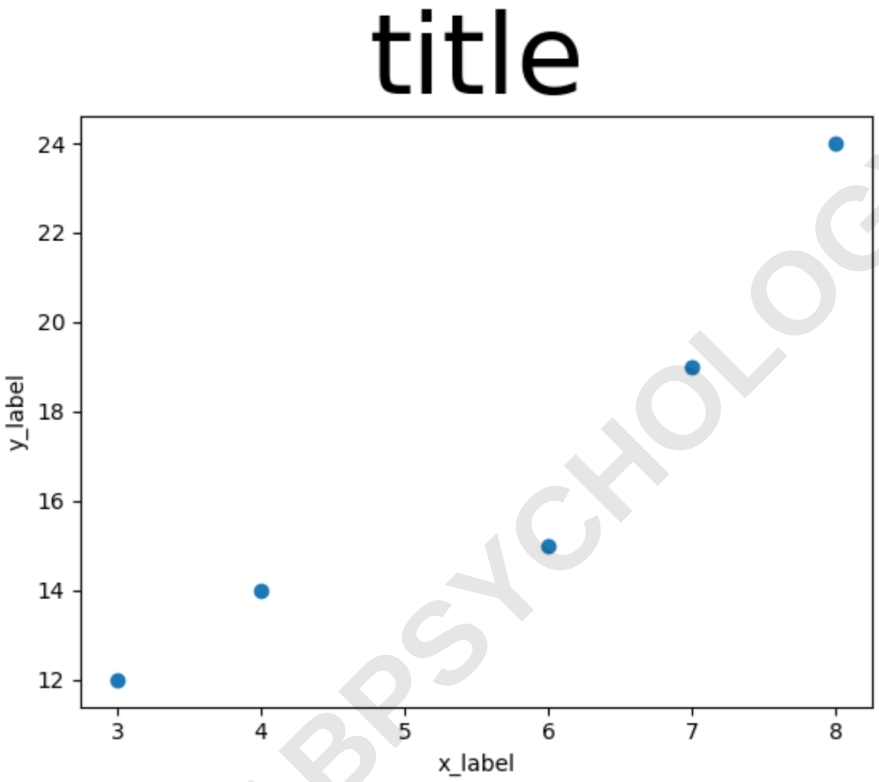
For this demonstration, we will set the title font size dramatically high, to 50. This exaggerated value serves to clearly illustrate the effect of the `titlesize` parameter. If we had previously set a global font size, this specific `titlesize` setting would override the global value for the title text only, maintaining the global size for other elements. This demonstrates the fine-tuning capabilities inherent in the Matplotlib configuration system, allowing for specific stylistic hierarchies.

When implementing this, we use `plt.rc('axes', titlesize=50)`. Note that the `'axes'` group controls several text properties related to the plot's central frame, not just the labels. This powerful command ensures that the title's scale meets specific visualization requirements. Ensuring the title is appropriately sized is crucial for posters and presentations where the audience needs to grasp the plot's context instantly, even from a distance. The resulting image clearly shows the title dominating the visual space.

#set title font to size 50

plt.rc('axes', titlesize=50)

#create plot
plt.scatter(x, y)
plt.title('title')
plt.xlabel('x_label')
plt.ylabel('y_label')
plt.show()



Example 3: Adjusting X and Y Axes Label Font Sizes

Axis labels--`x_label` and `y_label`--are essential for contextualizing the data being plotted, indicating units or variables. While the title provides the overall context, the axis labels define the dimensions. We control the size of these labels using the `labelsize` key, which is also part of the `'axes'` configuration group in `plt.rc()`. Unlike the title, which often stands out, axis labels should be clearly visible but subordinate to the data points and the title itself.

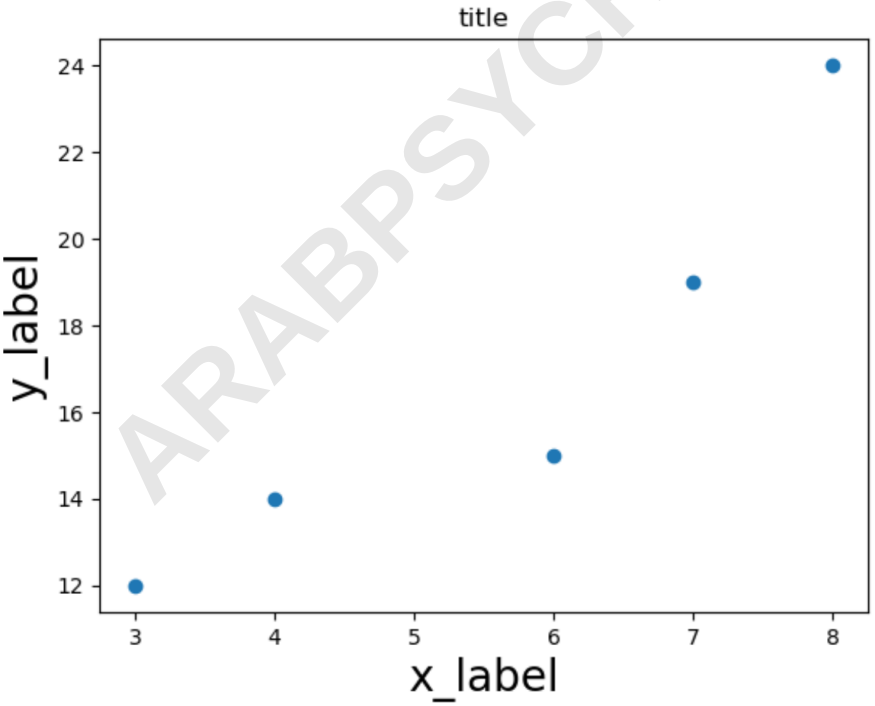
Setting the label size involves the command `plt.rc('axes', labelsize=20)`. This single configuration line impacts both the X and Y axis labels simultaneously. If you needed to set them to different sizes, you would have to use the individual `plt.xlabel(..., fontsize=...)` and

`plt.ylabel(..., fontsize=...)` functions, overriding the global setting. However, for most standard visualizations, maintaining a uniform size for both axes labels is the recommended practice for aesthetic balance.

In the code below, we increase the size to 20, making the labels much more prominent than the default size 10, yet not overwhelmingly large like the title in the previous example. This adjustment improves accessibility and clarity, especially when the labels contain lengthy descriptions or technical terms. Understanding the distinction between `titlesize` and `labelsize` within the `'axes'` group is key to professional plotting in Matplotlib.

```
#set axes labels font to size 20
plt.rc('axes', labelsize=20)

#create plot
plt.scatter(x, y)
plt.title('title')
plt.xlabel('x_label')
plt.ylabel('y_label')
plt.show()
```



Example 4: Modifying Tick Labels Font Sizes

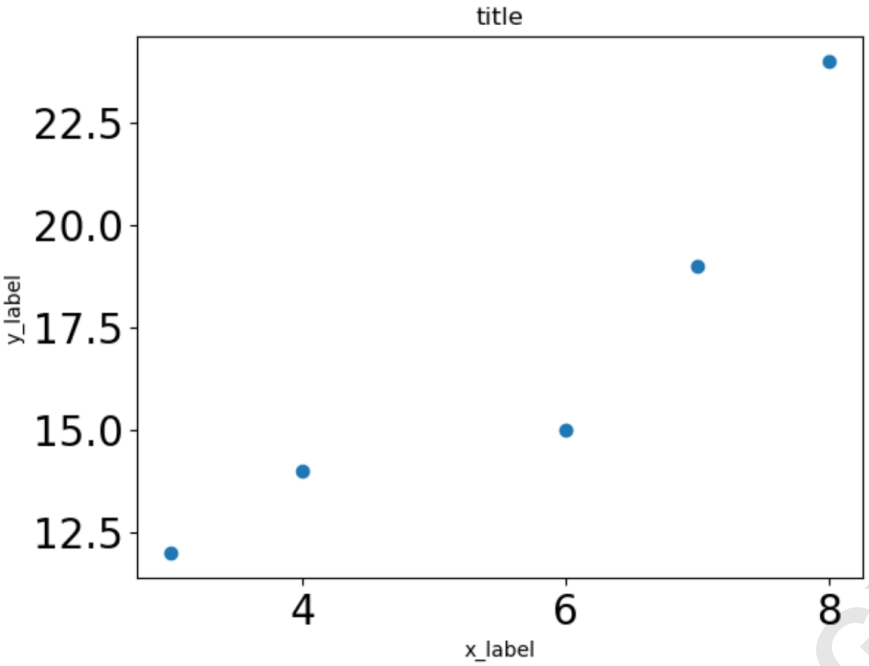
The Tick labels--the numerical markers along the X and Y axes--are crucial for reading precise data values. However, they must be managed carefully to avoid overlap and crowding near the plot margins. Matplotlib provides distinct configuration groups for these elements: `'xtick'` and `'ytick'`. Both groups use the `labelsize` key to control their respective font sizes, independent of the main axis labels or the plot title.

In complex plots with dense data or long numerical tick labels, reducing their size might be necessary. Conversely, in simple plots, increasing their size improves readability. This example demonstrates setting both X and Y tick labels to size 20. We must apply `plt.rc()` twice, once for `'xtick'` and once for `'ytick'`, though often they are set to the same size for uniformity.

The explicit separation of tick label control from axis label control (which uses the `'axes'` group) highlights Matplotlib's granular approach to customization. By managing tick size separately, you ensure that even if you choose a large font for the main axis labels, the underlying numerical information remains clear and spatially optimized. The resultant visualization shows prominently sized tick labels that are easy to read alongside the larger axis labels.

```
#set tick labels font to size 20
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)

#create plot
plt.scatter(x, y)
plt.title('title')
plt.xlabel('x_label')
plt.ylabel('y_label')
plt.show()
```



Advanced Control: Leveraging the rcParams Dictionary

While `plt.rc()` is the convenient function interface for runtime configuration changes, it is essentially a wrapper around the global `matplotlib.rcParams` dictionary. This dictionary holds all of Matplotlib's default settings. Understanding `rcParams` offers a deeper level of control and is particularly useful when you need to inspect or modify configuration settings in a more programmatic way, such as saving a custom style profile.

Each parameter you pass to `plt.rc()` corresponds directly to a key-value pair in `rcParams`. For example, setting `plt.rc('axes', titlesize=20)` is equivalent to setting `plt.rcParams = 20`. Directly manipulating `rcParams` is a powerful technique for batch updating multiple properties at once or for integrating Matplotlib configuration into larger application frameworks. This method is often employed by libraries or style packages that build on top of Matplotlib to enforce specific corporate or publication styling guidelines.

The relevant keys for font sizing within the `rcParams` dictionary are highly descriptive. For example:

`font.size`: Controls the default global font size.
`axes.titlesize`: Specific font size for the plot title.
`axes.labelsize`: Specific font size for the X and Y axis labels.
`xtick.labelsize`: Specific font size for the X Tick labels.
`ytick.labelsize`: Specific font size for the Y Tick labels.

Accessing and modifying these keys directly allows for script optimization and ensures that complex visualizations maintain consistent typography throughout.

Bonus: Restoring Default Font Settings

When working interactively, especially within environments like Jupyter notebooks, configuration changes made using `plt.rc()` persist across cells. If you make global modifications to the font size or other parameters, these changes will affect every subsequent plot generated in that session. To prevent contamination between experiments or to reset the environment to a clean state, it is often necessary to restore the original, default Matplotlib settings.

Matplotlib provides a straightforward mechanism for this restoration. By using the `plt.rcParams.update()` method combined with `plt.rcParamsDefault`, you can revert all runtime configuration parameters back to the state they were in when Matplotlib was initially imported. The `plt.rcParamsDefault` variable stores a copy of the default settings, making it easy to roll back any customizations applied via `plt.rc()` or direct manipulation of `rcParams`.

The code snippet below is crucial for maintaining a clean working environment and ensuring that each plot starts from a predictable baseline. It serves as a safety net when experimenting with aggressive styling changes or when transitioning between projects that require vastly different visual aesthetics.

plt.rcParams.update(plt.rcParamsDefault)

This comprehensive approach to font sizing--from global scaling with `plt.rc('font')` to element-specific adjustments using `titlesize` and `labelsize`, and finally, managing the configuration state via `rcParams`--empowers users to create visually compelling and contextually appropriate plots in Matplotlib.

You can find more Matplotlib tutorials here.