Learning Matplotlib: How to Display Only Horizontal Gridlines in Your Plots

Authored by

Mohammed looti

October 29, 2025

In the realm of data visualization, the effective presentation of information is paramount. Tools like Matplotlib, the foundational plotting library for the Python programming language, offer unparalleled control over every element of a graph. While standard plots often display both horizontal and vertical gridlines, there are numerous scenarios--particularly when comparing discrete values or tracking changes over time on a categorical axis--where limiting these markers to only the **horizontal orientation** significantly enhances clarity and focus. This technique is crucial for reducing visual clutter and directing the viewer's attention to the magnitude of the data points rather than distracting from the underlying trends.

The strategic use of selective gridlines is a hallmark of professional data storytelling. By isolating the horizontal gridlines, we provide precise reference points along the y-axis (the value axis), enabling viewers to accurately gauge the height of bars in a bar chart or the position of points in a scatter plot. Conversely, removing the vertical gridlines prevents interference with the x-axis categories or continuous variable tracking. This targeted approach helps in drawing attention to the critical patterns that matter most, improving the overall interpretability of your plots. This guide serves as a detailed tutorial on achieving this precise control within Matplotlib, detailing the specific parameters and objects required to plot only **horizontal gridlines** effectively.

Achieving this level of customization requires interacting directly with the Axes object, which encapsulates the entire data space of the plot. Understanding the relationship between the Figure, the Axes, and the core plotting functions like `grid()` is fundamental for any advanced Matplotlib user. We will explore the concise syntax necessary to implement horizontal-only grids, followed by practical examples demonstrating how to control their appearance, visibility, and layering relative to the plotted data elements, ensuring your visualizations are both accurate and aesthetically pleasing.

Basic Syntax for Horizontal Gridlines

To control any specific visual element within a Matplotlib visualization, interaction with the Axes object is essential. The Axes object (often referenced as `ax`) is the canvas upon which the data is drawn, managing the scaling, ticks, limits, and gridlines. The method responsible for toggling grid visibility is the versatile `grid()` function, accessible directly from the Axes instance. This function is a core component of Matplotlib's utility, offering robust control over reference markers.

When invoking the `grid()` method without any parameters, Matplotlib defaults to displaying both horizontal (y-axis) and vertical (x-axis) gridlines, which can sometimes lead to visual confusion in complex plots. To restrict this display solely to the horizontal lines, we must utilize the `axis` parameter. By setting `axis='y'`, we explicitly instruct the function to draw gridlines only perpendicular to the y-axis, resulting in the desired horizontal reference markers across the plot area. This targeted approach is the most efficient way to achieve focused grid display.

The fundamental command is remarkably straightforward, making it an efficient addition to any plotting workflow in Python. The `ax` variable in the structure below represents the currently active Axes object, typically generated using the common `fig, ax = plt.subplots()` pattern, which is the recommended way to initialize figures in modern plotting scripts. Mastery of this concise syntax is the first step toward effective gridline management and reducing unnecessary visual noise on the chart.

ax.grid(axis='y')

Example 1: Implementing Basic Horizontal Gridlines

To demonstrate the practical application of the horizontal gridline command, we will construct a standard bar plot. Bar charts are an ideal visualization type for this technique, as vertical lines (x-axis gridlines) often add unnecessary complexity between discrete categories, while horizontal lines are essential for accurately comparing the magnitude of the bars. This example leverages the Pandas library for efficient data structure handling, using a simple DataFrame containing sample team scores.

The code sequence involves three critical steps: first, importing Pandas and Matplotlib; second, preparing the data into a DataFrame and initializing the plot using `plt.subplots()`; and finally, generating the bar chart using the Pandas plotting wrapper which utilizes the Matplotlib Axes object. The key instruction, `ax.grid(axis='y')`, is inserted immediately after the plot creation to activate the horizontal gridlines, providing clear visual benchmarks for bar height comparison.

It is important to note the default behavior in Matplotlib: gridlines are typically drawn above (on top of) the plotted data elements. While this is functional, as shown in the resulting image, it can sometimes lead to the gridlines visually interrupting the bars themselves, especially if the lines are thick or dark. The following code block illustrates this default behavior, setting the stage for the next example where we will refine the visual hierarchy.

import pandas as pd
import matplotlib.pyplot as plt

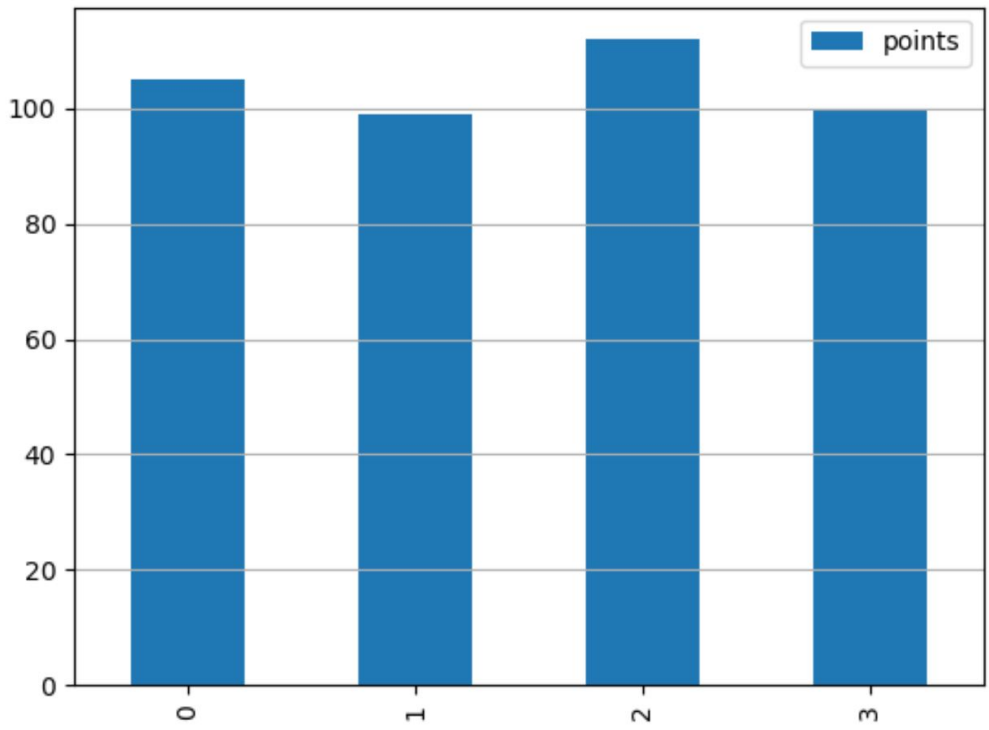
# Create DataFrame to hold sample scoring data
df = pd.DataFrame({'team':,
'points':})

Define the plot figure and axes
fig, ax = plt.subplots()

Create the bar plot using the DataFrame method

```
df.plot(kind='bar', ax=ax)

# Add horizontal gridlines exclusively
ax.grid(axis='y')

# Display the final plot
plt.show()
```

The output image clearly shows the bar plot with horizontal gridlines extending across the chart area. These gridlines are drawn on top of the bars, which is the default behavior. While functional, for some visualizations, placing gridlines behind the data elements can enhance clarity and aesthetics by ensuring the data remains the visual priority.



Example 2: Positioning Gridlines Behind Plot Elements

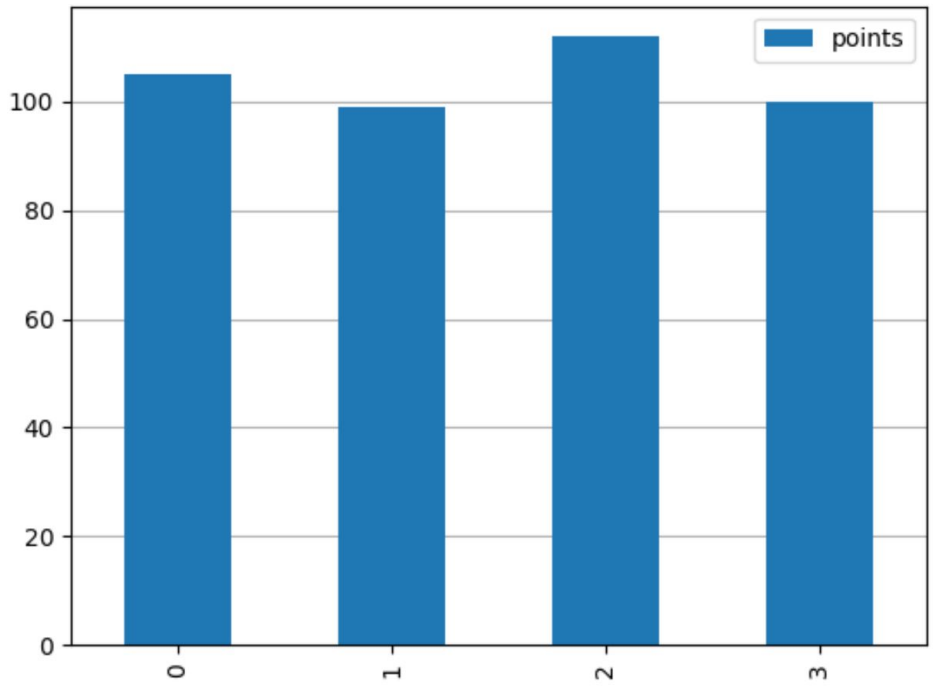
One of the key considerations in sophisticated data visualization is the visual hierarchy--ensuring that the data itself remains the primary focus. When gridlines, even subtle ones, are drawn directly over the data points or bars, they can inadvertently compete for attention or create a distracting visual noise that detracts from the underlying message. Matplotlib provides a simple yet powerful mechanism to control this layering behavior, allowing us to send the gridlines to the back layer, behind the main plot elements.

This control is managed by the `set_axisbelow()` method available on the Axes object. By passing the boolean value `True` to `ax.set_axisbelow(True)`, we instruct Matplotlib to adjust the drawing order. Specifically, this ensures that axis ticks, labels, and crucially, the gridlines, are drawn before the primary artists (such as bars, lines, or scatter points). This inversion of the drawing order results in a much cleaner visualization where the grid acts as a subtle background reference plane, highly effective for minimizing visual interference.

Implementing this change requires only a single line of code added prior to calling the `grid()` function. This modification is highly recommended for plots like bar charts, histograms, and area charts where the data elements occupy significant screen real estate. The following code demonstrates how to modify the previous example by integrating this vital command, thereby enhancing the overall aesthetic and interpretability of the plot by placing the horizontal gridlines behind the data bars:

```
import pandas as pd
import matplotlib.pyplot as plt

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

# define plot
fig, ax = plt.subplots()

# create bar plot
df.plot(kind='bar', ax=ax)

# IMPORTANT: Set axis elements (including gridlines) to be drawn below the data
ax.set_axisbelow(True)

# Add horizontal gridlines
ax.grid(axis='y')

# display plot
plt.show()
```

The visual difference achieved by positioning the gridlines behind the bars is significant. The bars now appear solid and uninterrupted, while the horizontal reference lines maintain their function without causing visual interference. This small adjustment is crucial for generating publication-quality figures where data clarity is paramount.



Example 3: Customizing Gridline Appearance

Beyond determining the orientation and layering of the gridlines, Matplotlib grants extensive control over their stylistic properties. Customization is vital for ensuring that the gridlines either blend subtly into the background or, conversely, stand out clearly enough to serve as effective reference markers without overpowering the plotted data. The core `grid()` function accepts several keyword arguments that allow for granular control over aesthetics, ensuring the grid complements the overall design.

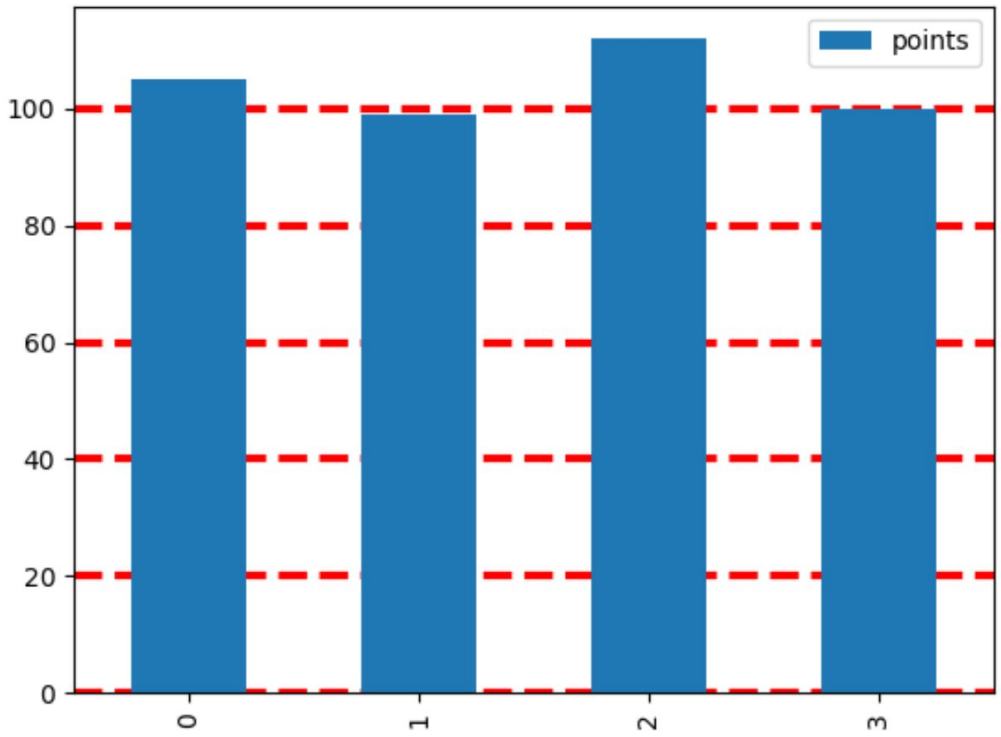
The most commonly adjusted parameters include `color`, which defines the hue of the lines (accepting standard color names, hex codes, or RGB tuples); `linestyle`, which controls the pattern of the line (e.g., `'solid'`, `'dashed'`, `'dotted'`, `'dashdot'`); and `linewidth`, which dictates the thickness of the line. Adjusting these three properties allows developers and analysts to align the grid appearance with corporate branding, publication standards, or simply personal aesthetic preference. For instance, using a light grey color with a thin linewidth often achieves the desired subtlety required for scientific or business reports.

In this final code example, we will combine the knowledge of positional control (`ax.set_axisbelow(True)`) with aggressive customization. We will set the horizontal gridlines to be bright red, dashed, and relatively thick. While this highly visible style may not be appropriate for all plots, it effectively demonstrates the power and flexibility available through the `grid()` function arguments. Remember that these parameters are applied only to the axis specified by `axis='y'`,

preserving the clean look of the x-axis.

```
import pandas as pd
import matplotlib.pyplot as plt

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

# define plot
fig, ax = plt.subplots()

# create bar plot
df.plot(kind='bar', ax=ax)

# Ensure gridlines are behind the bars
ax.set_axisbelow(True)

# Add horizontal gridlines with custom appearance
ax.grid(axis='y', color='red', linestyle='dashed', linewidth=3)

# display plot
plt.show()
```

The resulting plot offers a striking visual example of how stylistic choices impact interpretation. The gridlines are now bold red and dashed, providing a distinct visual style that can be tailored to match specific presentation requirements or branding guidelines. This level of granular control is what makes Matplotlib such a powerful tool for data visualization professionals.



Further Customization and Additional Resources

While the previous examples focused on the primary control mechanisms--orientation, layering, and line aesthetics--the Matplotlib `grid()` function is capable of much deeper customization. Data analysts often need to distinguish between major and minor gridlines, corresponding to the major and minor tick marks on the axis. By default, `grid()` applies to major gridlines, but you can target minor gridlines by setting the `which` parameter to `'minor'` or both by setting it to `'both'`. This allows for even finer reference points without the visual dominance of the major gridlines, which is useful when dealing with highly granular data.

Another powerful parameter for visual refinement is `alpha`, which controls the transparency of the gridlines. Setting `alpha` to a value between 0 (fully transparent) and 1 (fully opaque) is the easiest way to make gridlines subtle and ensure they support, rather than dominate, the visualization. Combining a light color (e.g., grey) with a low alpha value (e.g., 0.5) is a standard practice for creating professional, non-intrusive background grids that maintain readability while minimizing distraction. Furthermore, remember that the grid properties can be set globally using Matplotlib's configuration system, ensuring consistency across multiple plots.

Mastering these advanced techniques relies heavily on consulting the official library references. For a comprehensive overview of every available parameter, including settings for tick visibility, axis scaling, and more detailed styling options, users should refer directly to the official Matplotlib

documentation for `matplotlib.pyplot.grid`. This resource serves as the ultimate guide for unlocking the full potential of Matplotlib's grid functionality and ensuring your visualizations meet the highest standards of technical accuracy and aesthetic appeal.

Additional Matplotlib Resources

Effective plotting with Matplotlib often involves coordinating several different functions to achieve the perfect output. The ability to manage gridlines is fundamental, but proficiency also requires mastering other aspects of plot aesthetics and data presentation. The following resources offer focused guidance on related topics that complement the techniques discussed in this guide, helping you gain comprehensive control over your visualization environment:

How to Remove Ticks from Matplotlib Plots
How to Change Font Sizes on a Matplotlib Plot