Learning to Fill Areas Between Lines in Matplotlib for Data Visualization

Authored by

Mohammed looti

November 6, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learning to Fill Areas Between Lines in Matplotlib for Data Visualization*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=11827

When generating professional and insightful data visualization using the powerful Matplotlib library in Python, it is frequently essential to emphasize specific ranges or regions within a plot. This technique, universally known as area filling or area shading, serves a critical purpose in statistical and analytical contexts. It is crucial for visually representing key statistical concepts such as confidence intervals, illustrating predefined statistical thresholds, or segmenting distinct phases of a time series analysis. By shading the area between curves, we transform a simple line plot into a richer, more informative graphical representation that immediately draws the viewer's attention to the most relevant data space.

The ability to accurately and flexibly fill areas between plotted lines or curves is a cornerstone of effective data storytelling. Understanding the nuanced application of Matplotlib's dedicated functions ensures that the resulting visualizations are both aesthetically pleasing and scientifically rigorous. This comprehensive guide will explore the mechanics behind Matplotlib's area filling capabilities, focusing on the core functions that enable both horizontal and vertical shading, providing developers and analysts with the tools necessary to elevate their plotting expertise.

The Core Functions: fill_between() vs. fill_betweenx()

To execute the technique of area filling, Matplotlib provides two specialized and highly effective functions. Choosing the correct function depends entirely on the orientation of the region you intend to shade--whether the boundaries are defined along the vertical axis (Y-values) or the horizontal axis (X-values). Mastering the distinctions between these two commands is the first step toward precise graphical output.

These two functions allow for robust control over the shaded region, enabling complex visualizations where regions of interest are dynamically determined by data arrays rather than fixed coordinates. The parameters required for each function reflect its primary axis of operation, dictating the order in which the independent variable and the boundary variables must be supplied. We rely on these distinctions to ensure that the area filling accurately corresponds to the underlying data structure.

fill_between(): This function is the primary tool designed to fill the area between two horizontal curves. It operates by holding the X-values constant (the independent variable) and defining the shaded boundaries using Y-values (y1 and y2). This is the standard function utilized when shading regions like error bands around a central trend line or highlighting standard deviations in a time series plot. It is implicitly assumed that the X-array defines the shared domain for both boundary curves.
fill_betweenx(): Conversely, this function is used to fill the area between two vertical curves. In this configuration, the Y-values serve as the common independent variable, and the boundaries are defined by X-values (x1 and x2). This function is particularly suitable for visualizations where

the vertical dimension represents a continuous variable, and we need to demarcate fixed horizontal thresholds or categorical segments.

This tutorial provides detailed, practical, and replicable examples demonstrating how to properly initialize and utilize these powerful functions within a Python environment. We will utilize the NumPy library for efficient array generation, providing the foundational numerical data required for Matplotlib's plotting capabilities.

Example 1: Fundamental Horizontal Shading Using fill_between()

The `fill_between()` function is the most frequently used command for shading areas in standard Cartesian plots. It is specifically engineered for scenarios where the shaded region is bounded by two y-coordinates (y1 and y2) across a shared range of x-coordinates. At its minimum, the function requires the independent variable array (x-array), the lower y-boundary (y1), and optionally, the upper y-boundary (y2).

In the simplest and most common use case, if only the x-array and a single y-array (y2) are provided, the `fill_between()` function automatically defaults the lower boundary (y1) to the value zero (the x-axis baseline). This behavior makes it extremely straightforward to shade the entire region situated between a curve and the zero baseline. This is often the first step when visualizing cumulative distributions or absolute magnitudes.

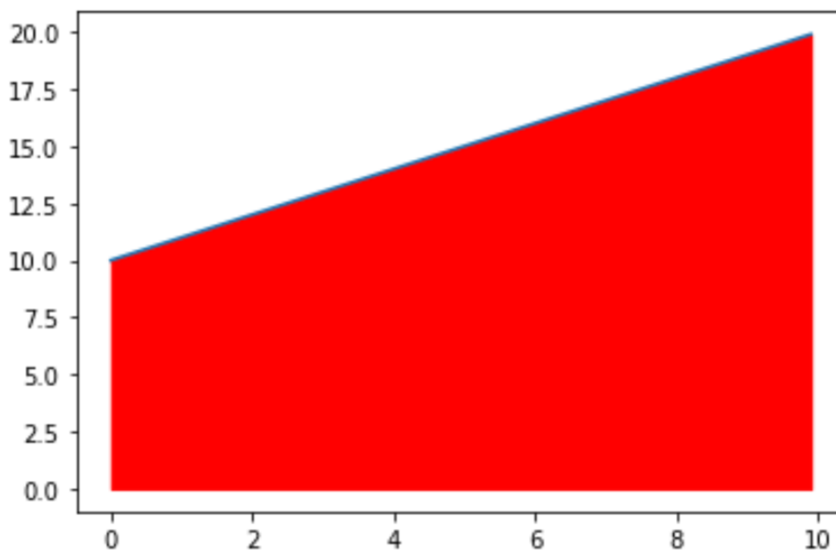
We leverage NumPy in the code below to efficiently generate linear array data for both X and Y axes. Notice how the function call only includes `x` and `y`, triggering the default shading behavior down to y=0, resulting in the red area filling the space under the plotted line.

```
import matplotlib.pyplot as plt
import numpy as np

#define x and y values
x = np.arange(0,10,0.1)
y = np.arange(10,20,0.1)

#create plot of values
plt.plot(x,y)

#fill in area between the two lines (defaults y1=0)
plt.fill_between(x, y, color='red')
```



To significantly enhance the visual clarity of the plot and provide better context for the filled area, several optional parameters can be introduced. Gridlines, added via the `plt.grid()` function, help viewers track coordinates and determine the precise extent of the shading. Furthermore, the `alpha` parameter is vital for controlling the opacity of the fill color. Setting `alpha` to a value between 0 (fully transparent) and 1 (fully opaque) is essential when the filled region overlaps with other plot elements, ensuring that underlying data or gridlines remain visible. Using a moderate alpha value, such as `0.5`, generally strikes a good balance between visibility and plot clarity.

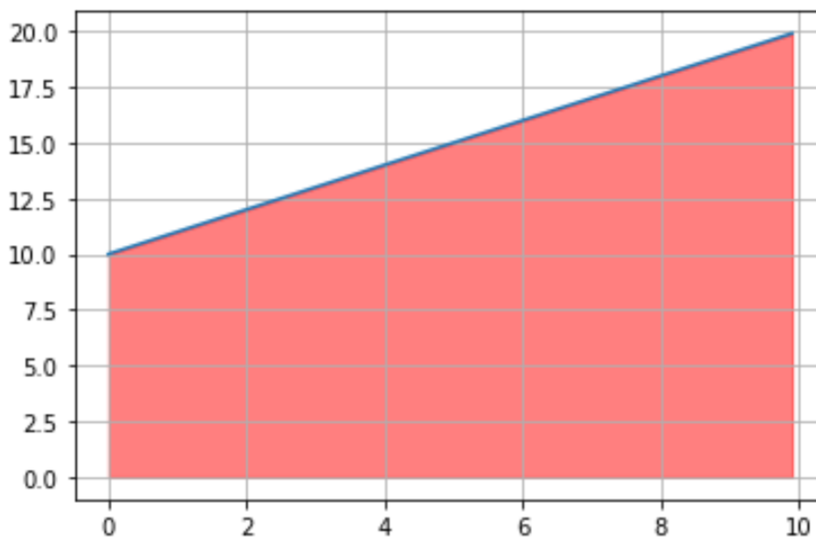
```
import matplotlib.pyplot as plt
import numpy as np

#define x and y values
x = np.arange(0,10,0.1)
y = np.arange(10,20,0.1)

#create plot of values
plt.plot(x,y)

#fill in area between the two lines with opacity control
plt.fill_between(x, y, color='red', alpha=.5)

#add gridlines
plt.grid()
```



Example 2: Visualizing Quantitative Measures (Area Under a Curve)

Visualizing the area under a curve is not merely a stylistic choice; it is a fundamental operation in disciplines ranging from calculus (representing definite integrals) to statistics (representing cumulative probabilities or total counts). When we shade this region, we are visually aggregating the contribution of the dependent variable across the entire domain of the independent variable. Since this operation inherently involves shading down to the zero baseline (the x-axis), it perfectly aligns with the default behavior of the `fill_between()` function when only the x-array and the curve's y-array are supplied.

To illustrate a more complex relationship than a simple linear progression, this example utilizes a non-linear function: `y = x**4` (y equals x to the power of four). This generates a curve that rapidly increases, emphasizing the growing magnitude of the area being shaded. Despite the complexity of the data relationship, the implementation of the area filling remains remarkably concise. We only need to provide the x-coordinates and the calculated y-values as the primary parameters for the filling function, allowing Matplotlib to handle the baseline definition.

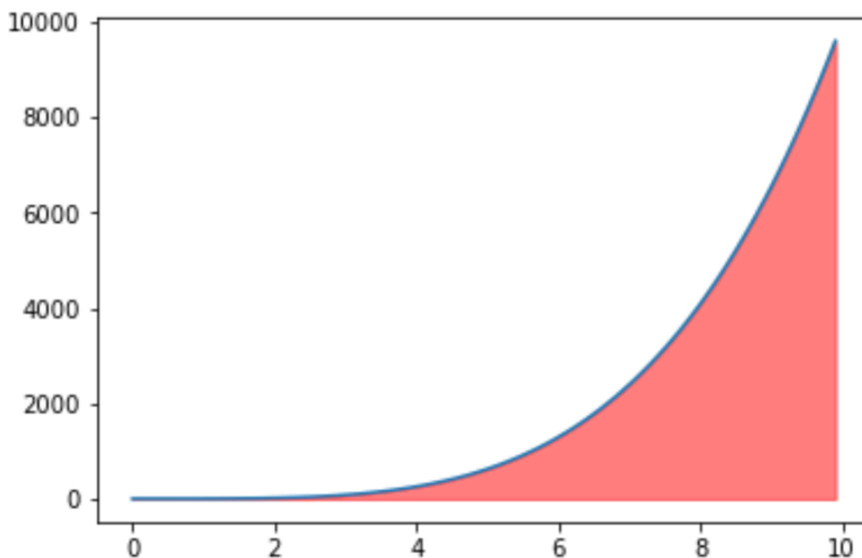
The code below defines the data using the power function, plots the line, and then uses `fill_between()` to shade the region between the curve and y=0. The consistent use of the `alpha` parameter ensures that the resulting visualization is subtle yet effective, clearly demarcating the area of accumulated magnitude without overpowering the plotted line itself. This technique is invaluable for scientific reporting where the total area under a function holds significant meaning.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
#define x and y values
x = np.arange(0,10,0.1)
y = x**4

#create plot of values
plt.plot(x,y)

#fill in area between the curve and the x-axis
plt.fill_between(x, y, color='red', alpha=.5)
```



Example 3: Defining Boundaries for Upper Limits (Shading Above the Curve)

While shading the area under a curve is common, situations often arise where the objective is to visualize the space remaining between a data curve and a predefined maximum threshold or theoretical limit. This technique is useful for illustrating potential capacity, quantifying inefficiency, or showing the difference between an achieved value and a maximum possible value. To successfully shade the area *above* a curve, we must explicitly define the curve itself as the lower boundary (y1) and specify a constant, higher maximum value as the upper boundary (y2).

In this example, we utilize the same non-linear data set (y = x?) as the curve to shade from. However, instead of defaulting to y=0, we define the upper limit dynamically. We use the NumPy function `np.max(y)` to programmatically determine the highest data point achieved in our dataset. This maximum value then serves as the constant "ceiling" for the shaded region across the entire X-domain.

By defining the parameters as `(x, y1, y2)` where `y1 = y` (the curve) and `y2 = np.max(y)` (the

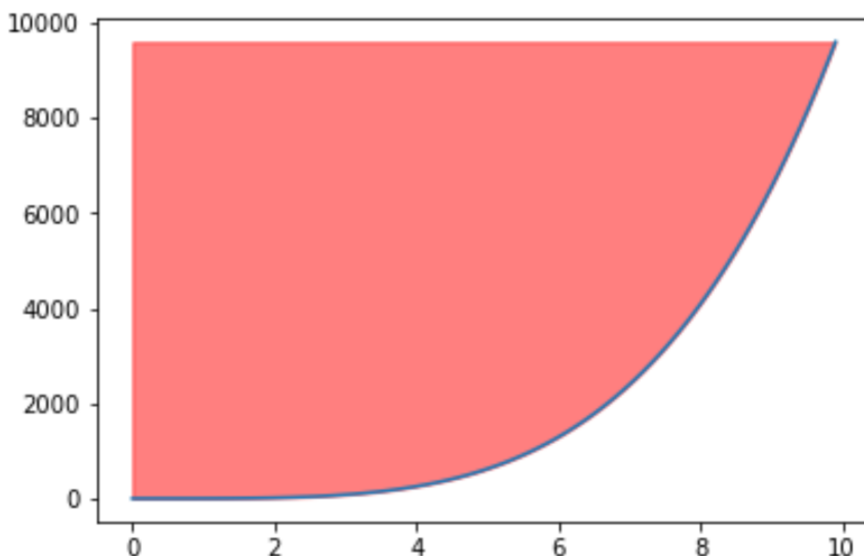
constant maximum), we instruct Matplotlib to shade only the space between the curve and that upper limit. This demonstrates the powerful flexibility of `fill_between()` in defining custom boundaries that are not fixed plot edges but rather constants derived directly from the data itself. This visualization technique is far more informative than simply plotting the line alone, as it quantifies the "gap" or residual space available up to the maximum observed value.

import matplotlib.pyplot as plt
import numpy as np

#define x and y values
x = np.arange(0,10,0.1)
y = x**4

#create plot of values
plt.plot(x,y)

#fill in area between the curve (y) and the max observed value (np.max(y))
plt.fill_between(x, y, np.max(y), color='red', alpha=.5)



Example 4: Vertical Area Delimitation with fill_betweenx()

While `fill_between()` handles most horizontal shading requirements, the `fill_betweenx()` function provides an essential orthogonal capability. This function is necessary when the visualization requirement demands a shaded area that spans a fixed horizontal range (defined by X-boundaries) across the entire vertical extent (Y-values) of the plot. This is typically employed to visually isolate specific time periods, mark critical experimental phases, or delineate categorical

divisions within a continuous data series plotted on the vertical axis.

The key differentiator is the parameter ordering. Unlike its horizontal counterpart, `fill_betweenx()` requires the Y-array (the independent variable) first, followed by the left x-boundary (x1) and the right x-boundary (x2). If only one x-boundary is supplied, it defaults the other boundary to x=0, similar to the horizontal function defaulting to y=0. By supplying two explicit X-coordinates, we create a vertical strip of shading that extends across the range of the Y data provided.

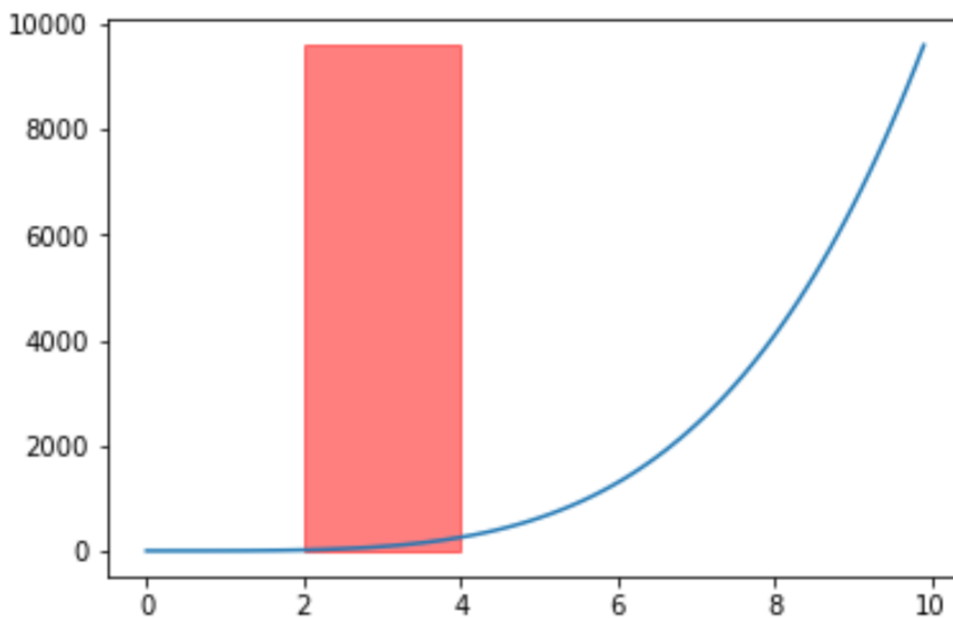
In the following code, we generate a simple diagonal line and then use `fill_betweenx()` to shade the vertical strip between fixed x-coordinates, specifically between x=2 and x=4. This clearly demonstrates the function's application in demarcating a region based on horizontal limits, providing a distinct contrast to the boundary definition techniques used in the previous examples. This precise control over vertical strips is crucial for specialized analytical plots.

```
import matplotlib.pyplot as plt
import numpy as np

#define x and y values
x = np.arange(0,10,0.1)
y = np.arange(10,20,0.1)

#create plot of values
plt.plot(x,y)

#fill in area between the two lines (x1=2, x2=4 across the y-range)
plt.fill_betweenx(y, 2, 4, color='red', alpha=.5)
```



Conclusion and Advanced Applications

By thoroughly understanding the precise application and distinct parameter requirements of `fill_between()` and `fill_betweenx()`, developers and data analysts gain significant control over the visual communication aspects of their data. These functions are essential tools for delineating regions of statistical significance, highlighting error margins, or visually segmenting complex data based on defined thresholds. The ability to dynamically set boundaries, as demonstrated with `np.max(y)`, allows for sophisticated visualizations that move beyond simple fixed-axis shading.

Furthermore, these functions support conditional filling, where the shading is applied only when a certain criterion is met (e.g., shading only where Curve A is greater than Curve B). This advanced technique allows users to plot two lines and shade the area between them only when they diverge significantly, providing immediate insight into periods of statistical difference. This level of customization ensures that Matplotlib remains the premier Matplotlib library for generating publication-quality scientific figures.

Mastering area filling is a fundamental skill that greatly improves the interpretability of complex plots, making the visual representation of data clearer, more precise, and ultimately, more persuasive in analytical reports.

Related: How to Plot a Smooth Curve in Matplotlib