😊 Matplotlib Styling Plots

Matplotlib styling refers to the customization of plots to make them more visually appealing, easier to interpret, or aligned with specific aesthetic or presentation guidelines. It allows you to change various aspects of your plot such as colors, fonts, line styles, markers, grid styles, and background colors.

Key Components of Matplotlib Styling

1. **Line Styles and Colors**
    - You can change the color of lines, markers, and other elements using color names (e.g., 'red', 'blue'), hexadecimal color codes (e.g., #FF5733), or RGB values.
    - You can adjust line styles (solid, dashed, dotted) using the linestyle parameter and choose markers like dots, squares, or triangles with the marker parameter.
2. **Fonts and Text**
    - Titles, axis labels, legends, and tick labels can be customized using font size, font family, and font weight.
    - You can also adjust the color and position of the text for better alignment or emphasis.
3. **Gridlines**
    - Gridlines can be added to your plots to enhance readability, and their style (color, line type, width) can be customized.
    - You can also control which axes (x-axis, y-axis, or both) have gridlines.
4. **Background and Figure Customization**
    - The background color of the figure or axes can be changed to set the tone of the plot (e.g., white, gray, or black backgrounds).
    - You can also adjust the transparency of different elements in the plot for a more subtle look.
5. **Ticks**
    - Ticks represent the values shown on the axes. You can customize their size, color, direction, and positioning.
    - Custom tick labels can also be applied to display specific values or formats.
6. **Legends**
    - The legend can be customized for position, font size, frame color, and transparency. It is useful for labeling different elements on the plot when you have multiple data series.
7. **Themes**

- Matplotlib offers several predefined styles called **themes** (like "seaborn", "ggplot", "bmh", etc.) that apply a set of common aesthetic changes across your plots.
- You can set a style globally using plt.style.use().

Common Styling Options

- **Line Styles**: '-' (solid), '--' (dashed), ':' (dotted), '-.' (dash-dot)
- **Markers**: 'o' (circle), '^' (triangle up), 's' (square), '*' (star)
- **Colors**: 'blue', 'green', '#FF5733', (0.1, 0.2, 0.5) (RGB)
- **Font Settings**: fontsize, fontweight, fontname
- **Gridline Styles**: color, linestyle, linewidth
- **Figure/Background**: figfacecolor, axes.facecolor

How to Apply Styling

You can apply styling to specific elements of the plot:

- **Title, labels, and ticks**:
  Customize with plt.title(), plt.xlabel(), plt.ylabel(), and plt.xticks() or plt.yticks().
- **Gridlines**:
  Enable with plt.grid(True) and customize with grid_color, grid_linestyle, and grid_linewidth.
- **Legends**:
  Use plt.legend() and adjust position, font, and other settings.

Styling Example

- You can set a style globally by calling plt.style.use('seaborn').
- You can also customize specific parts of a plot like the line color, style, or grid lines directly when creating the plot.

Conclusion

Matplotlib styling provides a powerful set of tools to refine the appearance of your plots, making them more visually appealing or easier to understand. You can adjust nearly every element of a plot, from line styles and colors to fonts and gridlines. By using styling techniques, you can create highly professional and tailored visualizations for data analysis or presentation.

😄 To begin with the Lab

1. We start by importing matplotlib library in our Jupyter Notebook.

[2]:
```
# COMMON MISTAKE!
# DON'T FORGET THE .PYPLOT part

import matplotlib.pyplot as plt
```

2. The code imports the numpy library, which is used for numerical operations. Then, it creates an array x with values from 0 to 9 using np.arange(0, 10).
3. After that, it creates another array y where each value is double the corresponding value in x (i.e., y = 2 * x). This results in y being a scaled version of x.

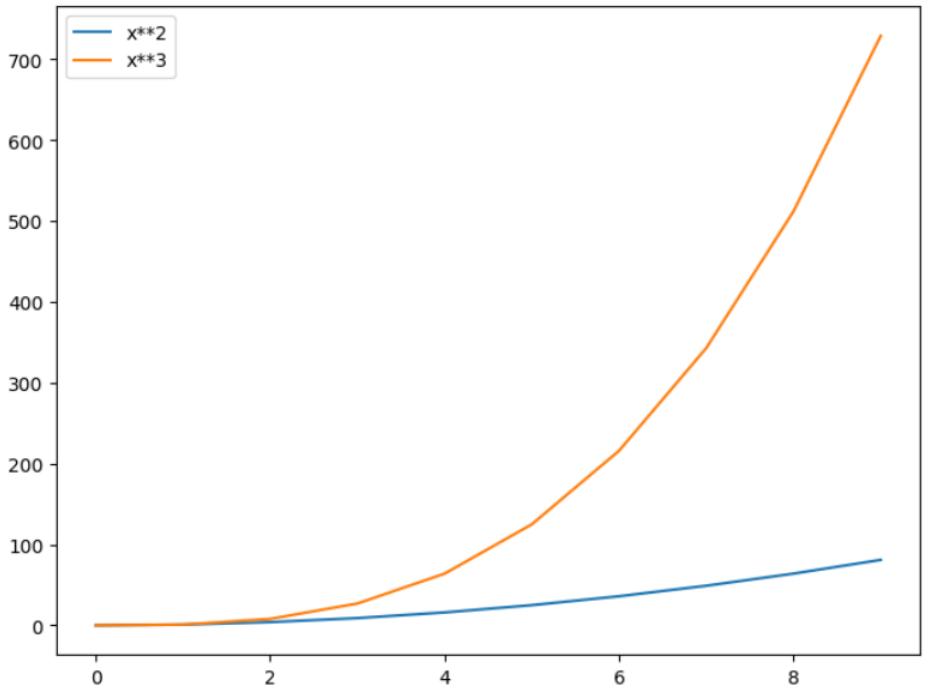
[4]:
```
import numpy as np
```

[6]:
```
x = np.arange(0,10)
y = 2 * x
```

4. The code creates a new figure using plt.figure() and then adds a set of axes to the figure with fig.add_axes([0,0,1,1]), where the numbers define the position and size of the axes within the figure.
5. It then plots two lines: x vs x**2 and x vs x**3 on the axes. The label argument assigns a label to each plot for the legend.
6. Finally, ax.legend() is used to display the legend, showing the labels for each plot (x**2 and x**3) on the graph.

[10]:
```
fig = plt.figure()

ax = fig.add_axes([0,0,1,1])

ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend()
```

[10]:
```
<matplotlib.legend.Legend at 0x1600d2f6790>
```

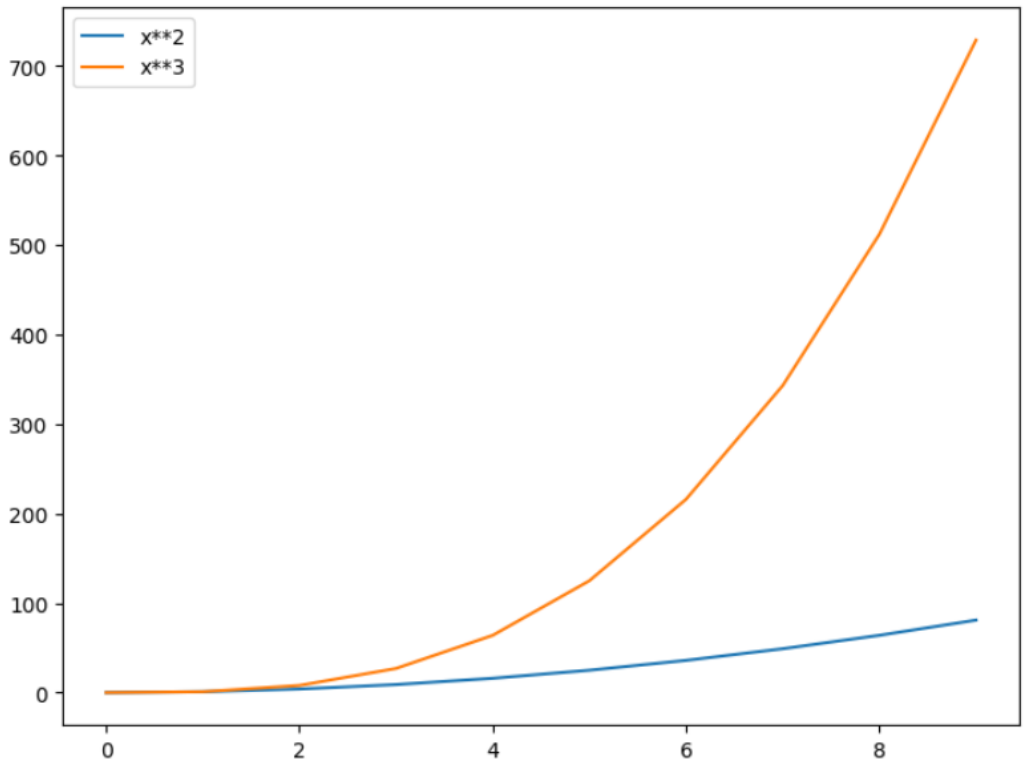


7. This code demonstrates how to control the placement of the legend on the plot using the `ax.legend()` method. The `loc` parameter specifies the location of the legend, where:

    - `loc=1` places the legend in the upper right corner.
    - `loc=2` places it in the upper left corner.
    - `loc=3` places it in the lower left corner.
    - `loc=4` places it in the lower right corner. By using `loc=0`, Matplotlib automatically decides the best location for the legend based on the plot layout. The `fig` at the end ensures that the figure is shown.

[13]:
```
# Lots of options....

ax.legend(loc=1) # upper right corner
ax.legend(loc=2) # upper left corner
ax.legend(loc=3) # lower left corner
ax.legend(loc=4) # lower right corner

# .. many more options are available

# Most common to choose
ax.legend(loc=0) # let matplotlib decide the optimal location
fig
```

[13]:

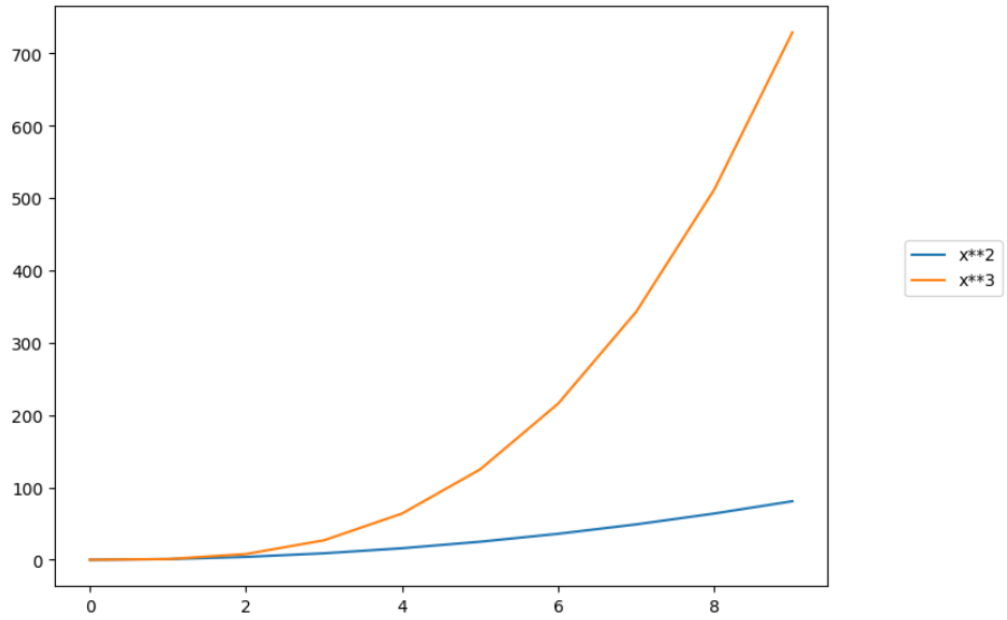


8. In this code, ax.legend(loc=(1.1, 0.5)) manually sets the legend's location on the plot using a tuple (x, y), where x and y are relative coordinates between 0 and 1.
9. The x value represents the horizontal position (in this case, 1.1 moves it slightly outside the plot area), and the y value represents the vertical position (0.5 places the legend in the middle vertically).

10. This allows for precise control over the legend's placement. The fig ensures that the figure is displayed.

```
[15]: ax.legend(loc=(1.1,0.5)) # manually set location
      fig
```

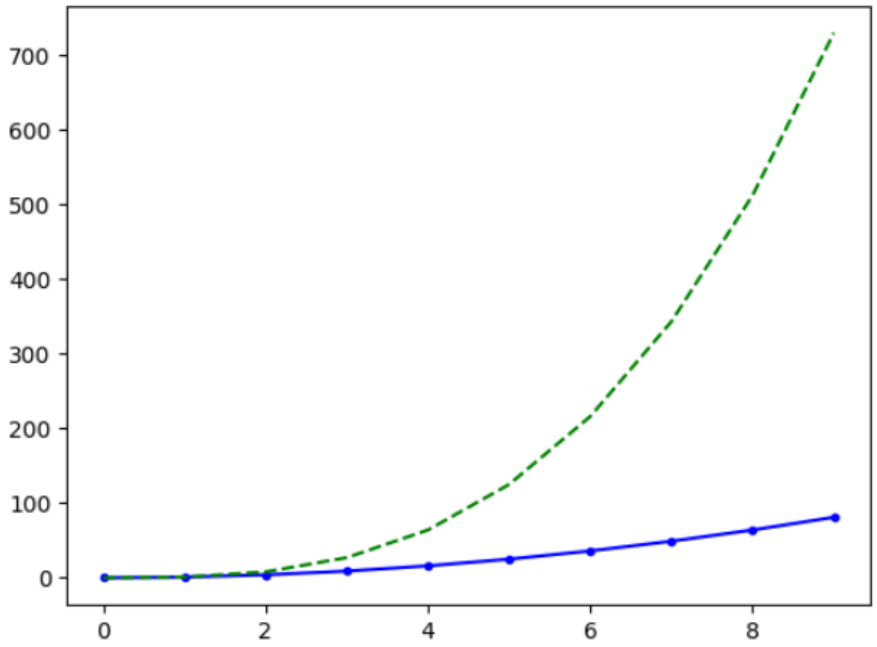
[15]:



11. In this code, ax.plot(x, x**2, 'b.-') plots the equation x2x^2x2 with a blue line and dots, indicated by 'b.-'.
12. Here, 'b' stands for blue, '.' represents dots along the line, and '-' represents a solid line.
13. Similarly, ax.plot(x, x**3, 'g--') plots x3x^3x3 with a green dashed line, represented by 'g--', where 'g' is for green and '--' indicates dashed lines.
14. This is a MATLAB-style shorthand to set line color and style in matplotlib.

[20]:
```
# MATLAB style line color and style
fig, ax = plt.subplots()
ax.plot(x, x**2, 'b.-') # blue line with dots
ax.plot(x, x**3, 'g--') # green dashed line
```

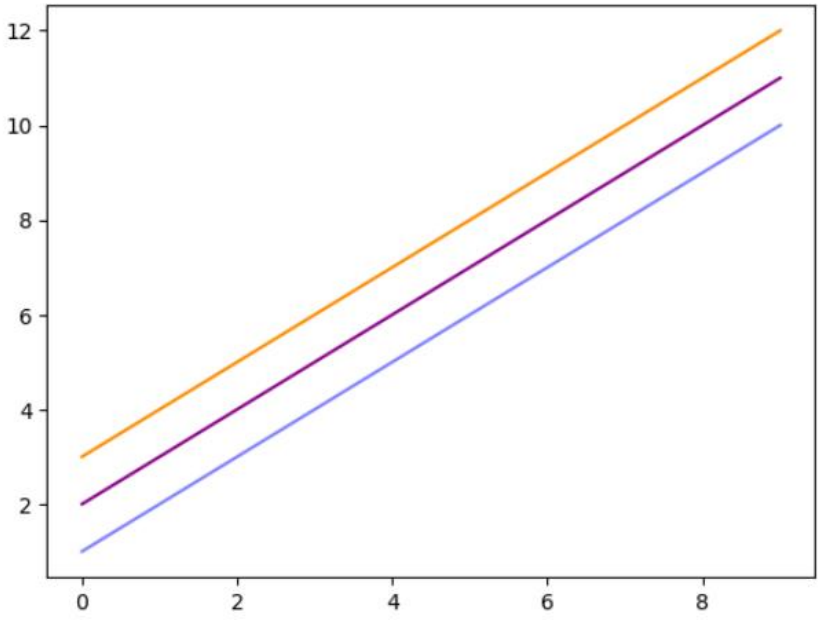
[20]:
```
[<matplotlib.lines.Line2D at 0x1600d49cb10>]
```



15. In this code, ax.plot(x, x+1, color="blue", alpha=0.5) plots a blue line with a transparency of 50% (alpha = 0.5).
16. The color="blue" sets the line color to blue, and alpha=0.5 makes it semi-transparent. ax.plot(x, x+2, color="#8B008B") uses an RGB hex code (#8B008B) to set a dark magenta color for the second line.
17. Similarly, ax.plot(x, x+3, color="#FF8C00") uses the hex code #FF8C00 to set the line color to dark orange. This approach allows precise control over colors using names or hex values.

[24]:
```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="blue", alpha=0.5) # half-transparant
ax.plot(x, x+2, color="#8B008B")        # RGB hex code
ax.plot(x, x+3, color="#FF8C00")        # RGB hex code
```

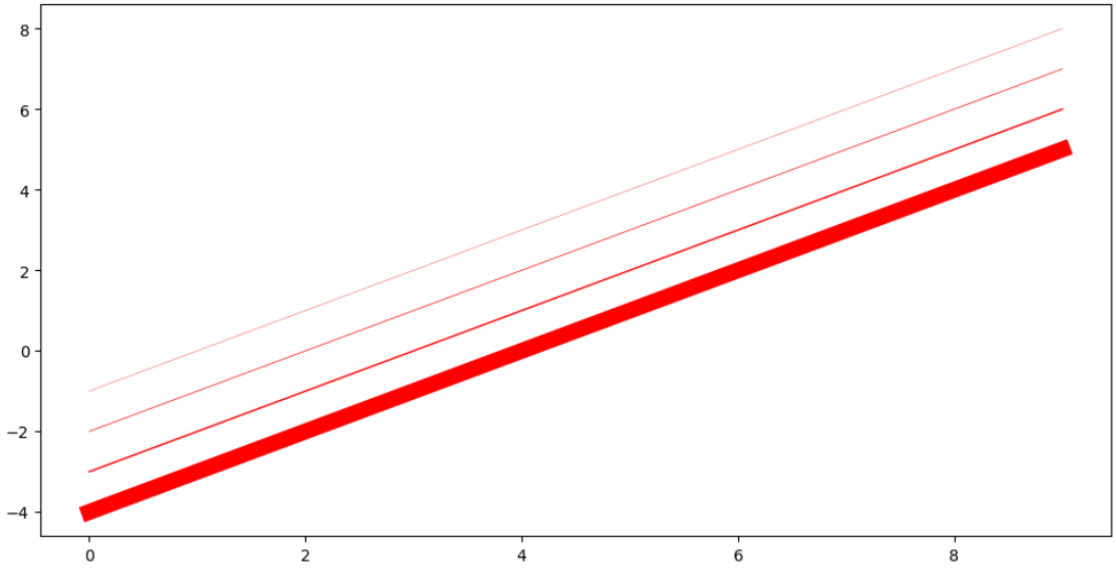
[24]:
```
[<matplotlib.lines.Line2D at 0x1600d5d5910>]
```



18. In this code, the ax.plot(x, x-1, color="red", linewidth=0.25) plots a red line with a line width of 0.25, making it very thin.
19. The next plot ax.plot(x, x-2, color="red", lw=0.50) uses lw as a shorthand for linewidth and sets the line width to 0.5, which is still thin but more visible.
20. The ax.plot(x, x-3, color="red", lw=1) sets a standard line width of 1, which is commonly used for clear visibility.
21. Finally, ax.plot(x, x-4, color="red", lw=10) sets a much thicker line with a width of 10, making it much bolder.
22. This demonstrates how linewidth or lw controls the thickness of the plot's lines.

[28]:
```
fig, ax = plt.subplots(figsize=(12,6))

# Use linewidth or lw
ax.plot(x, x-1, color="red", linewidth=0.25)
ax.plot(x, x-2, color="red", lw=0.50)
ax.plot(x, x-3, color="red", lw=1)
ax.plot(x, x-4, color="red", lw=10)
```

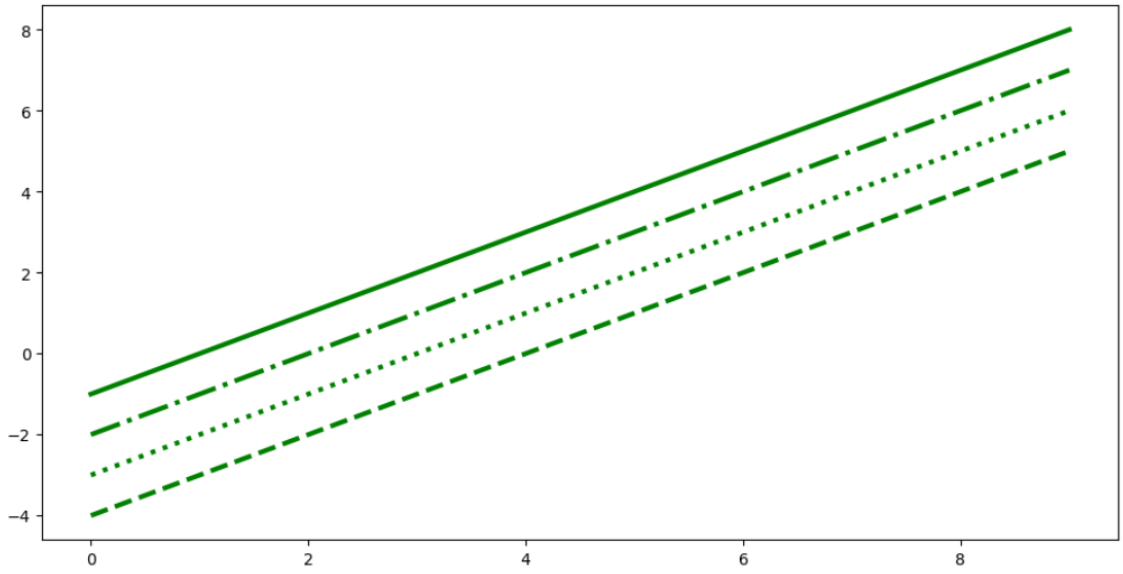
[28]:
```
[<matplotlib.lines.Line2D at 0x1600f7fab10>]
```



23. In this code, the ax.plot(x, x-1, color="green", lw=3, linestyle='-') plots a solid green line with a width of 3.
24. The ax.plot(x, x-2, color="green", lw=3, ls='-.') creates a green line with a dash-dot pattern (dashes and dots), and the ax.plot(x, x-3, color="green", lw=3, ls=':') draws a green dotted line.
25. The ax.plot(x, x-4, color="green", lw=3, ls='--') plots a green line made up of dashes.
26. These options for linestyle (ls) control the appearance of the line in terms of whether it is solid, dashed, dotted, or a combination.

[31]:
```
# possible linestype options '--', '-', '-.', ':', 'steps'
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(x, x-1, color="green", lw=3, linestyle='-') # solid
ax.plot(x, x-2, color="green", lw=3, ls='-.') # dash and dot
ax.plot(x, x-3, color="green", lw=3, ls=':') # dots
ax.plot(x, x-4, color="green", lw=3, ls='--') # dashes
```

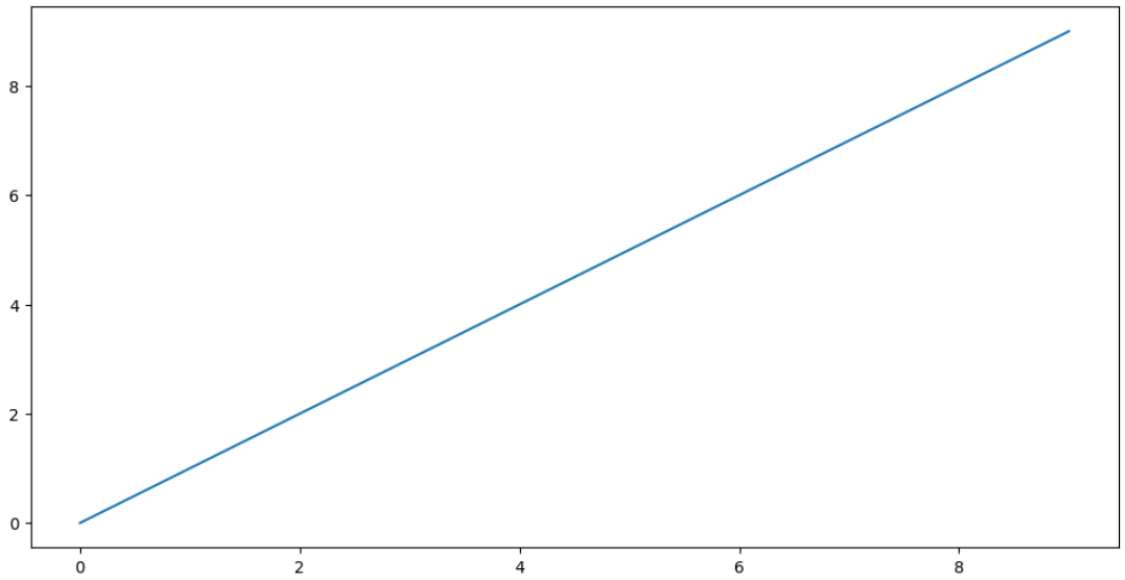
[31]:
```
[<matplotlib.lines.Line2D at 0x1600f86d190>]
```



27. In this code, fig, ax = plt.subplots(figsize=(12,6)) creates a figure and a single set of axes.
28. The line lines = ax.plot(x, x) plots a line where both the x and y values range from 0 to 9.
29. The plot() function returns a **list of Line2D objects**, even if you're only plotting one line.
30. When you print type(lines), it shows <class 'list'>, indicating that lines is a list (specifically, a list of matplotlib line objects).
31. This allows for future access or customization of the line(s).

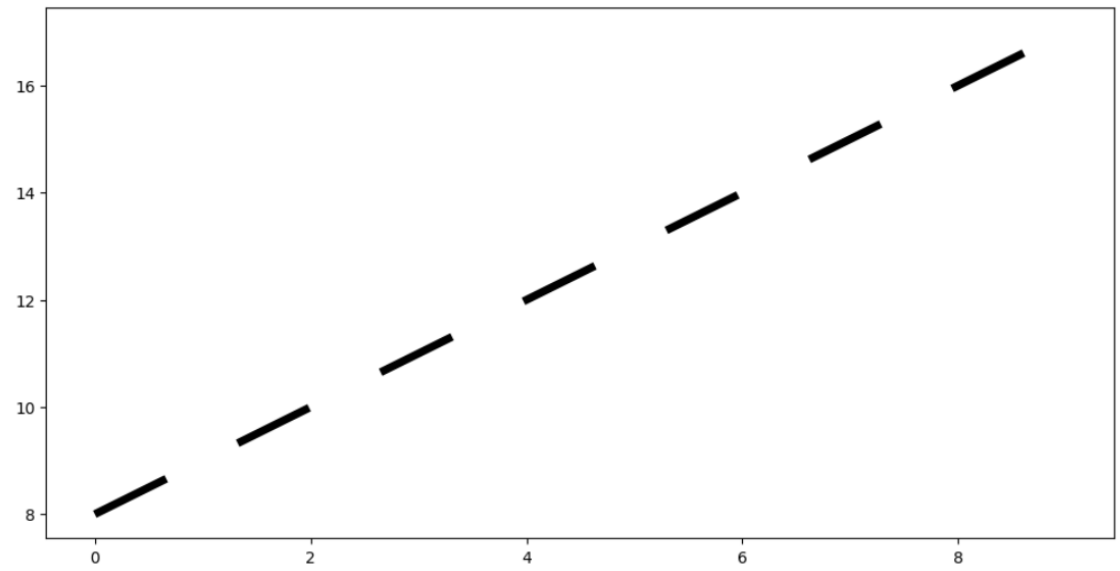
[34]:
```
fig, ax = plt.subplots(figsize=(12,6))

lines = ax.plot(x,x)

print(type(lines))
```

```
<class 'list'>
```



32. This code creates a custom dashed line using Matplotlib. The plot() function draws a black line with linewidth 5 for the data x and x + 8.
33. The set_dashes([10, 10]) method sets a custom dash pattern on the first (and only) line object in the lines list, where the pattern [10, 10] means 10 units of line followed by 10 units of space, repeatedly.
34. This gives you precise control over the dash appearance of the plotted line.

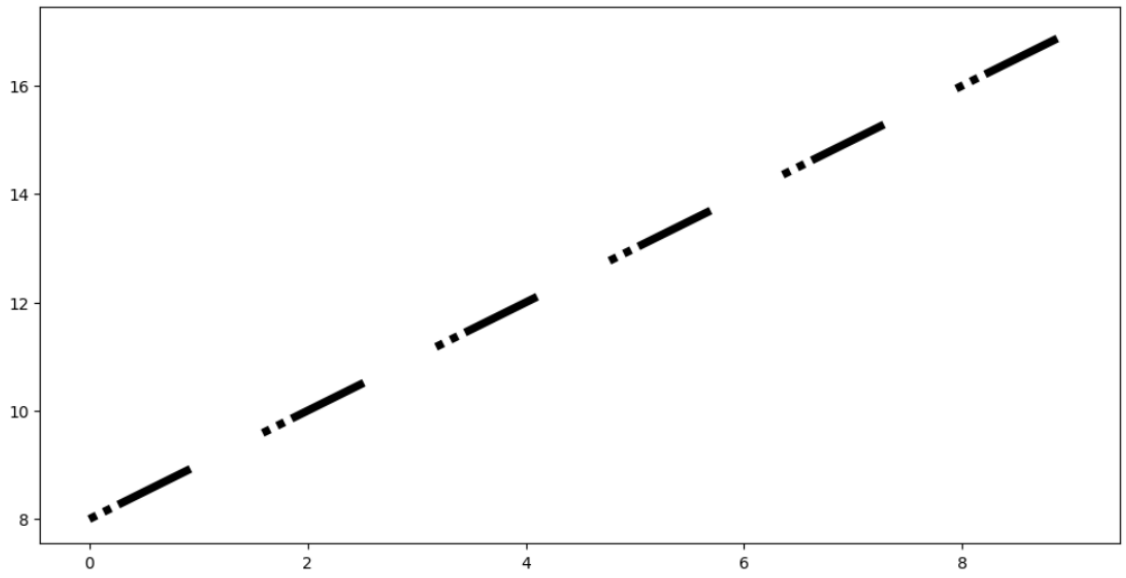
[36]:
```
fig, ax = plt.subplots(figsize=(12,6))
# custom dash
lines = ax.plot(x, x+8, color="black", lw=5)
lines[0].set_dashes([10, 10]) # format: line length, space length
```



35. This code creates a plot with a custom dash pattern applied to a black line of width 5.

36. The line is plotted for the data x and x + 8. The set_dashes([1, 1, 1, 1, 10, 10]) method defines a repeating dash pattern: 1 unit line, 1 unit space, 1 unit line, 1 unit space, 10 units line, 10 units space, and so on.
37. This allows for intricate line styling beyond standard dashed or dotted lines.

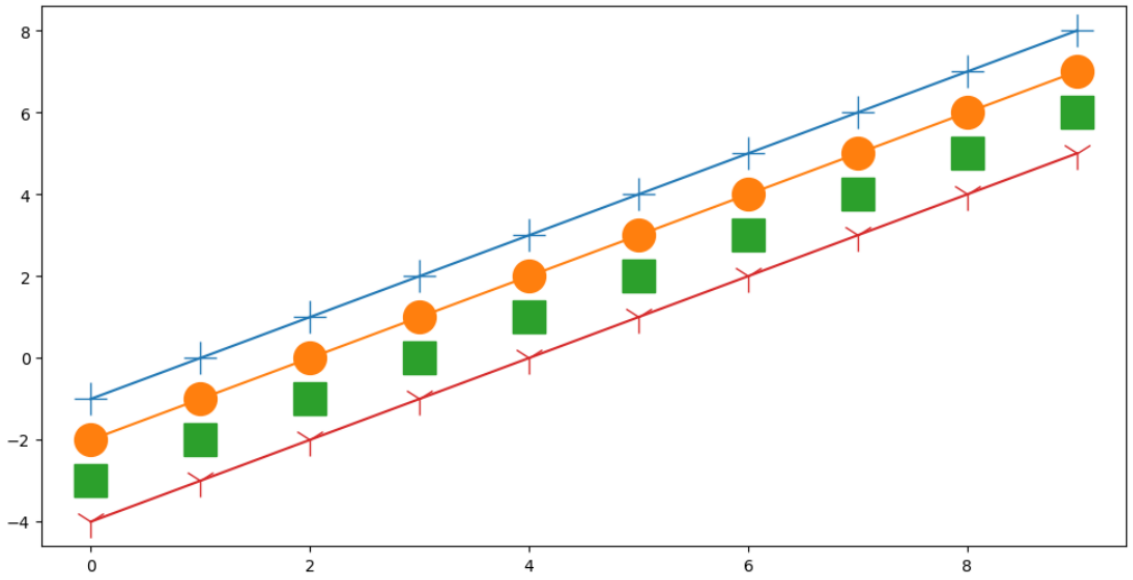
[38]:
```
fig, ax = plt.subplots(figsize=(12,6))
# custom dash
lines = ax.plot(x, x+8, color="black", lw=5)
lines[0].set_dashes([1, 1,1,1,10,10]) # format: line length, space length
```



38. This code generates a plot with four different lines, each using distinct marker styles to highlight individual data points.
39. The marker parameter specifies the shape (e.g., '+', 'o', 's', '1'), and markersize or ms controls the marker size.
40. Setting lw=0 (line width) makes the line invisible, showing only the markers. This helps emphasize data points without connecting lines.

[41]:
```
fig, ax = plt.subplots(figsize=(12,6))

# Use marker for string code
# Use markersize or ms for size

ax.plot(x, x-1,marker='+',markersize=20)
ax.plot(x, x-2,marker='o',ms=20) #ms can be used for markersize
ax.plot(x, x-3,marker='s',ms=20,lw=0) # make linewidth zero to see only markers
ax.plot(x, x-4,marker='1',ms=20)
```

[41]: [<matplotlib.lines.Line2D at 0x1601077e150>]



41. This code creates a plot with square markers along a black line. The marker='s' sets square-shaped markers, markersize=20 defines their size, markerfacecolor="red" fills the marker with red, markeredgewidth=8 sets a thick border, and markeredgecolor="blue" colors the border blue.
42. The line itself is black (color="black"), with a solid style (ls='-') and a width of 1 (lw=1).

[44]:
```
fig, ax = plt.subplots(figsize=(12,6))

# marker size and color
ax.plot(x, x, color="black", lw=1, ls='-', marker='s', markersize=20,
        markerfacecolor="red", markeredgewidth=8, markeredgecolor="blue");
```