😊 Matplotlib Figures

In Matplotlib, a **figure** is the top-level container for all plot elements. Think of it as a blank canvas on which all visualizations—like plots, labels, titles, and legends—are drawn. When you create a plot, you're working within a figure, even if it's not explicitly defined.

Characteristics of a Matplotlib Figure

1. **Container for Visual Output**
   A figure acts as the outermost container that holds all elements of your visualization, including one or more **axes** (the actual plots), and other components like **titles**, **legends**, and **colorbars**.
2. **Multiple Subplots**
   A single figure can contain multiple subplots or axes, allowing you to display several plots in one visual output (for example, a 2x2 grid of graphs).
3. **Customizable Size and Layout**
   You can specify the figure size in inches and adjust the layout to control spacing between subplots.
4. **Exportable and Renderable**
   Figures can be rendered to a window, saved to a file (such as PNG, PDF, or SVG), or embedded in a GUI or web application.

Main Components Inside a Figure

- **Axes (Subplots)**: The individual plotting areas within a figure. Each axes is a region where data is plotted.
- **Title**: The overall title of the figure.
- **Legend**: The label guide, often shown for plots with multiple datasets.
- **Colorbar**: A guide to color-coded data in a plot (used in heatmaps and contour plots).
- **Grid and Ticks**: Optional elements that can be added to each subplot for readability.

Purpose and Importance

Figures are essential for:

- Managing multiple plots in a single visual.
- Customizing layout and aesthetics.
- Controlling resolution and file output when saving.

In summary, a **figure** in Matplotlib is the overarching structure that holds all plotting elements. It enables structured, multi-part visualizations and acts as the final product that can be displayed or saved.

😄 To begin with the Lab

1. We start by importing Matplotlib in our Jupyter notebook. You can also try and use Google Colab to work with these labs.

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

import matplotlib.pyplot as plt
```

2. This code uses NumPy to create numerical arrays and perform element-wise operations.
3. First, a = np.linspace(0,10,11) creates an array of 11 evenly spaced numbers between 0 and 10.
4. Then b = a ** 4 raises each element of a to the power of 4.
5. Separately, x = np.arange(0,10) creates an array from 0 to 9, and y = 2 * x multiplies each element in x by 2.
6. These arrays (a, b, x, and y) can be used for further analysis or visualization like plotting.

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

[6]:
```
a = np.linspace(0,10,11)
b = a ** 4
```

[8]:
```
a
```

[8]:
```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

10]:
```
b
```

10]:
```
array([0.000e+00, 1.000e+00, 1.600e+01, 8.100e+01, 2.560e+02, 6.250e+02,
       1.296e+03, 2.401e+03, 4.096e+03, 6.561e+03, 1.000e+04])
```

12]:
```
x = np.arange(0,10)
y = 2 * x
```

14]:
```
x
```

14]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

16]:
```
y
```

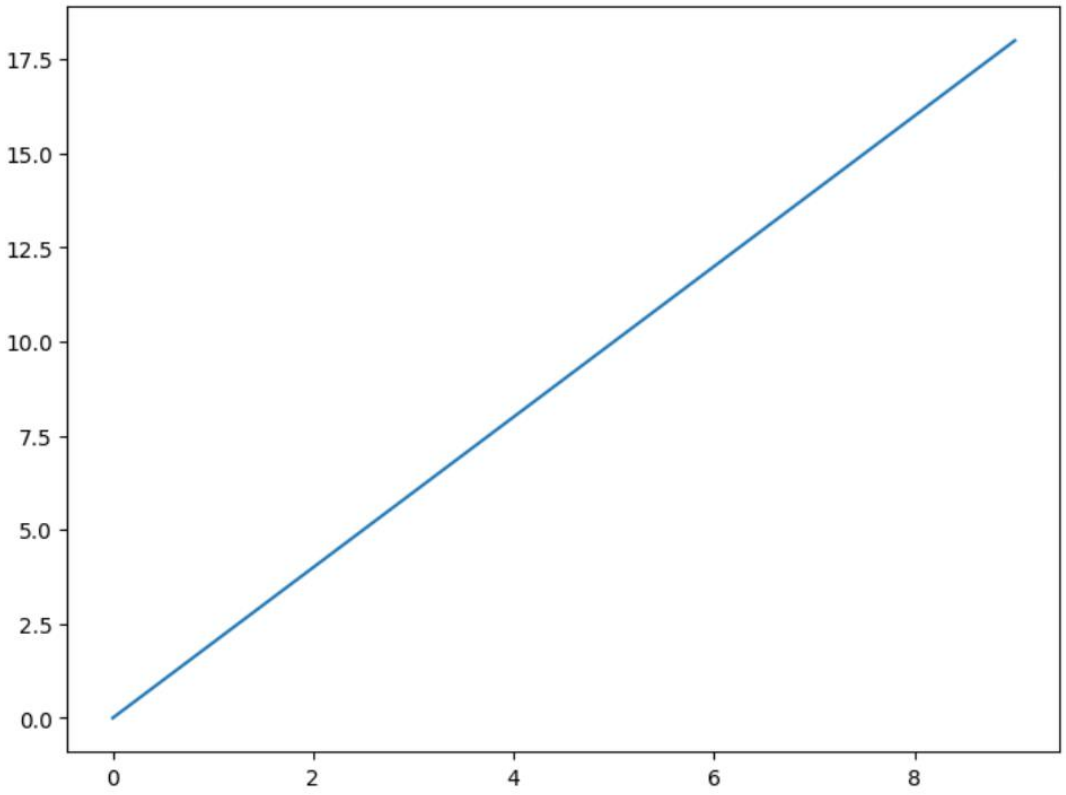
16]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

7. This code uses Matplotlib to manually create a figure and plot data. fig = plt.figure() initializes a blank canvas.
8. Then, fig.add_axes([0, 0, 1, 1]) adds a set of axes to the figure, where the list [0, 0, 1, 1] specifies the position and size of the axes on the canvas (from 0 to 1, in relative units).
9. axes.plot(x, y) plots the data on these axes. Finally, plt.show() displays the figure.
10. This method gives precise control over plot layout, especially useful for advanced or customized visualizations.

[18]:
```
# Creates blank canvas
fig = plt.figure()
```

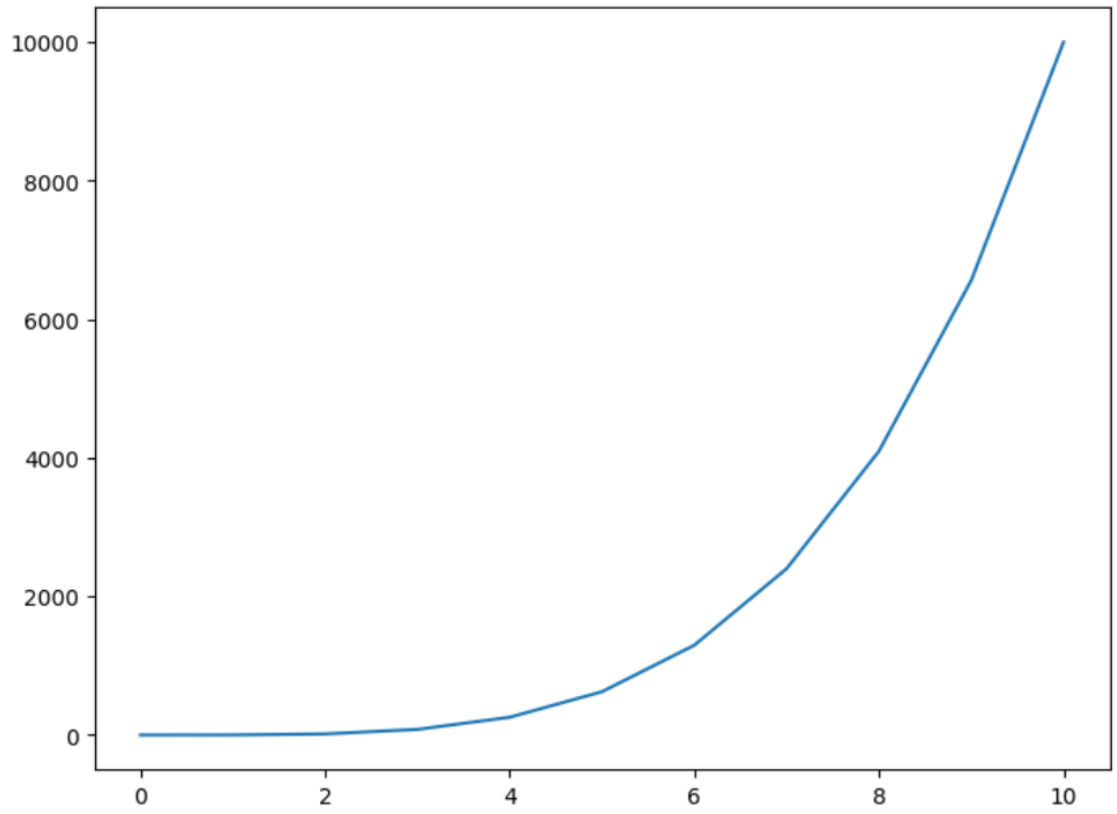
```
<Figure size 640x480 with 0 Axes>
```

[21]:
```
# Create Figure (empty canvas)
fig = plt.figure()

# Add set of axes to figure
axes = fig.add_axes([0, 0, 1, 1]) # left, bottom, width, height (range 0 to 1)

# Plot on that set of axes
axes.plot(x, y)

plt.show()
```



11. This code creates a customized plot using Matplotlib. First, fig = plt.figure() creates a blank figure.
12. Then, fig.add_axes([0, 0, 1, 1]) adds a set of axes that fills the entire figure (from left=0, bottom=0 to width=1, height=1).
13. The axes.plot(a, b) command plots the data points from arrays a and b on the axes.
14. Finally, plt.show() displays the plot. This method allows precise manual control over figure and axes placement, useful for creating detailed or layered visualizations.

[23]:
```
# Create Figure (empty canvas)
fig = plt.figure()

# Add set of axes to figure
axes = fig.add_axes([0, 0, 1, 1]) # left, bottom, width, height (range 0 to 1)

# Plot on that set of axes
axes.plot(a, b)

plt.show()
```



15. This code creates a figure with two sets of axes using Matplotlib. fig = plt.figure() initializes an empty canvas.
16. axes1 = fig.add_axes([0, 0, 1, 1]) adds a large set of axes filling the figure, while axes2 = fig.add_axes([0.2, 0.2, 0.5, 0.5]) adds a smaller inset figure.
17. The axes1.plot(a, b) and axes2.plot(a, b) commands plot the data on each set of axes.
18. The set_ methods add labels and titles to both sets of axes, with axes1 being the larger and axes2 the inset plot.

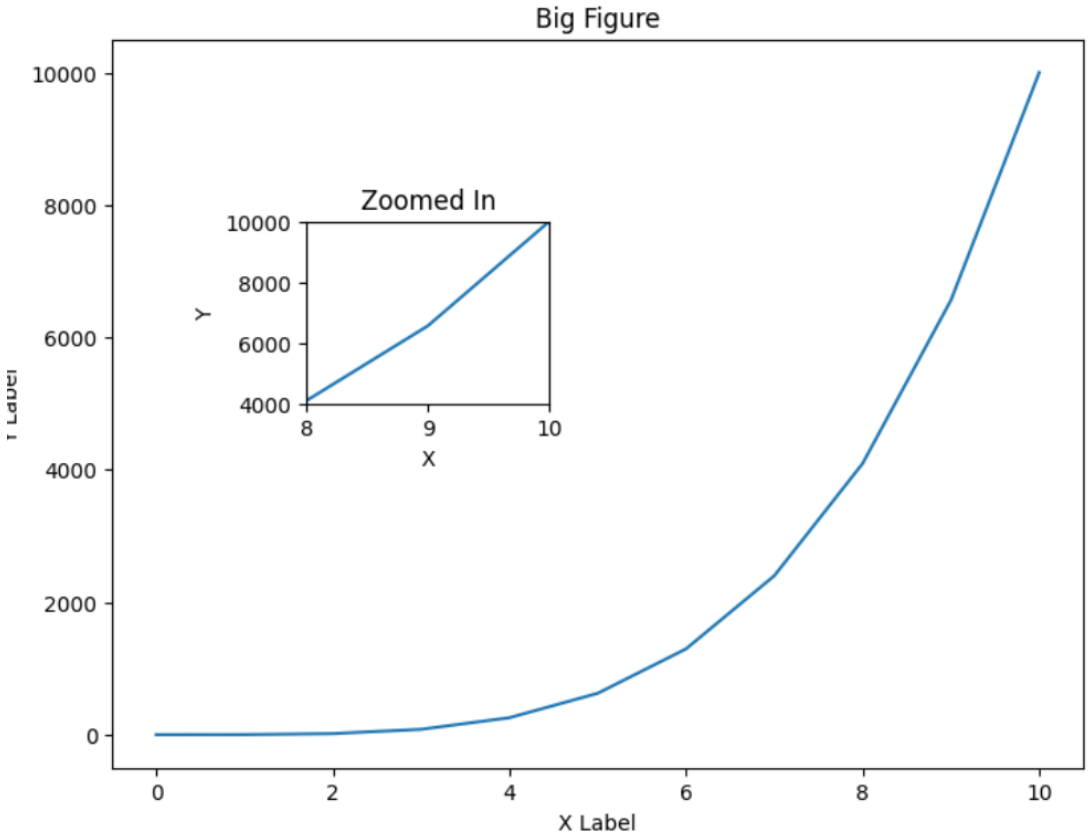
[25]:
```
type(fig)
```

[25]:
```
matplotlib.figure.Figure
```

[27]:
```
# Creates blank canvas
fig = plt.figure()

axes1 = fig.add_axes([0, 0, 1, 1]) # Large figure
axes2 = fig.add_axes([0.2, 0.2, 0.5, 0.5]) # Smaller figure

# Larger Figure Axes 1
axes1.plot(a, b)

# Use set_ to add to the axes figure
axes1.set_xlabel('X Label')
axes1.set_ylabel('Y Label')
axes1.set_title('Big Figure')

# Insert Figure Axes 2
axes2.plot(a,b)
axes2.set_title('Small Figure');
```

19. This code creates a figure with two sets of axes. The first axes, axes1, occupies the entire figure space, while axes2 is a smaller inset positioned at [0.2, 0.5, 0.25, 0.25].
20. The larger plot (axes1) displays the data with labels and a title. The smaller plot (axes2) is zoomed in with specific x and y limits using set_xlim and set_ylim.
21. Both plots have their own labels and titles set using the set_ methods. The result is a main plot with a zoomed-in inset showing a closer view of the data.

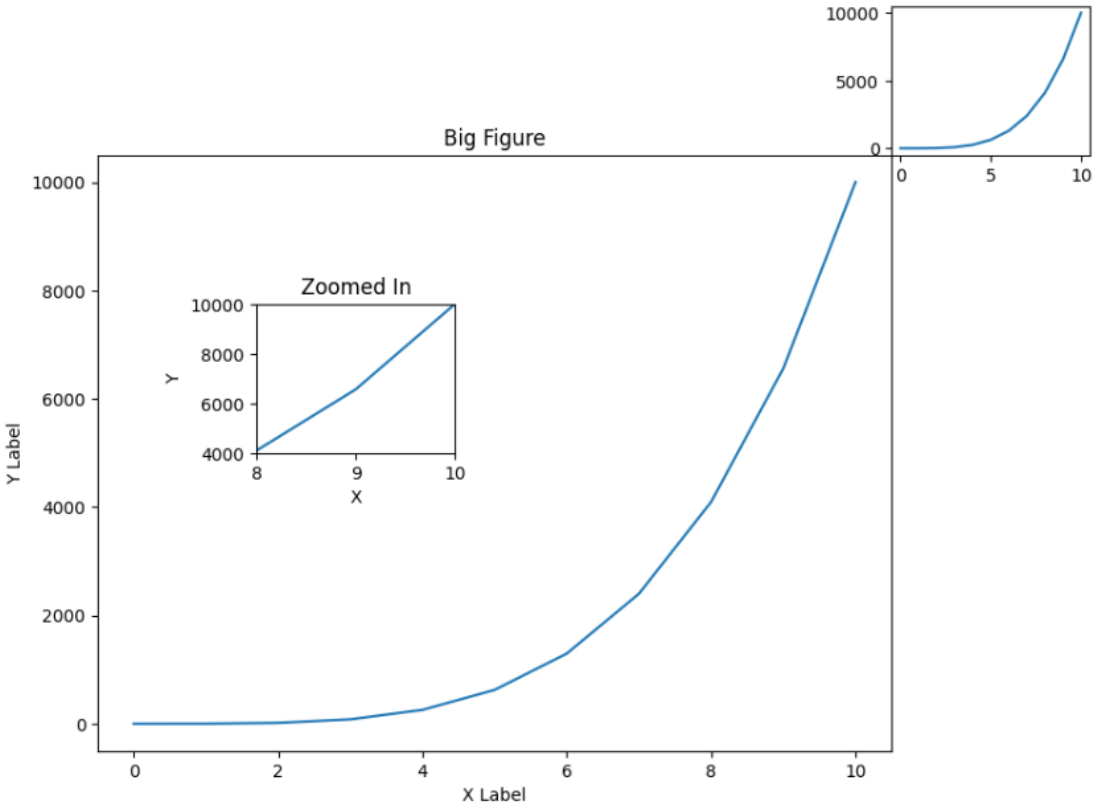
[29]:
```
# Creates blank canvas
fig = plt.figure()

axes1 = fig.add_axes([0, 0, 1, 1]) # Large figure
axes2 = fig.add_axes([0.2, 0.5, 0.25, 0.25]) # Smaller figure

# Larger Figure Axes 1
axes1.plot(a, b)

# Use set_ to add to the axes figure
axes1.set_xlabel('X Label')
axes1.set_ylabel('Y Label')
axes1.set_title('Big Figure')

# Insert Figure Axes 2
axes2.plot(a,b)
axes2.set_xlim(8,10)
axes2.set_ylim(4000,10000)
axes2.set_xlabel('X')
axes2.set_ylabel('Y')
axes2.set_title('Zoomed In');
```



22. This code creates a figure with three axes. `axes1` occupies the entire figure, displaying a plot with labels and a title.
23. `axes2` is a smaller inset with zoomed-in `x` and `y` limits, along with its own labels and title.
24. `axes3` is another inset placed at the top-right corner of the figure, where a plot is displayed without modifications.
25. The `set_` methods are used to set labels and titles for each plot, while `set_xlim` and `set_ylim` zoom in on the smaller inset (`axes2`).
26. Each axes is positioned within the figure using specific coordinates.

[31]:
```
# Creates blank canvas
fig = plt.figure()

axes1 = fig.add_axes([0, 0, 1, 1]) # Full figure
axes2 = fig.add_axes([0.2, 0.5, 0.25, 0.25]) # Smaller figure
axes3 = fig.add_axes([1, 1, 0.25, 0.25]) # Starts at top right corner!

# Larger Figure Axes 1
axes1.plot(a, b)

# Use set_ to add to the axes figure
axes1.set_xlabel('X Label')
axes1.set_ylabel('Y Label')
axes1.set_title('Big Figure')

# Insert Figure Axes 2
axes2.plot(a,b)
axes2.set_xlim(8,10)
axes2.set_ylim(4000,10000)
axes2.set_xlabel('X')
axes2.set_ylabel('Y')
axes2.set_title('Zoomed In');

# Insert Figure Axes 3
axes3.plot(a,b)
```

[31]:
```
[<matplotlib.lines.Line2D at 0x28469a542d0>]
```

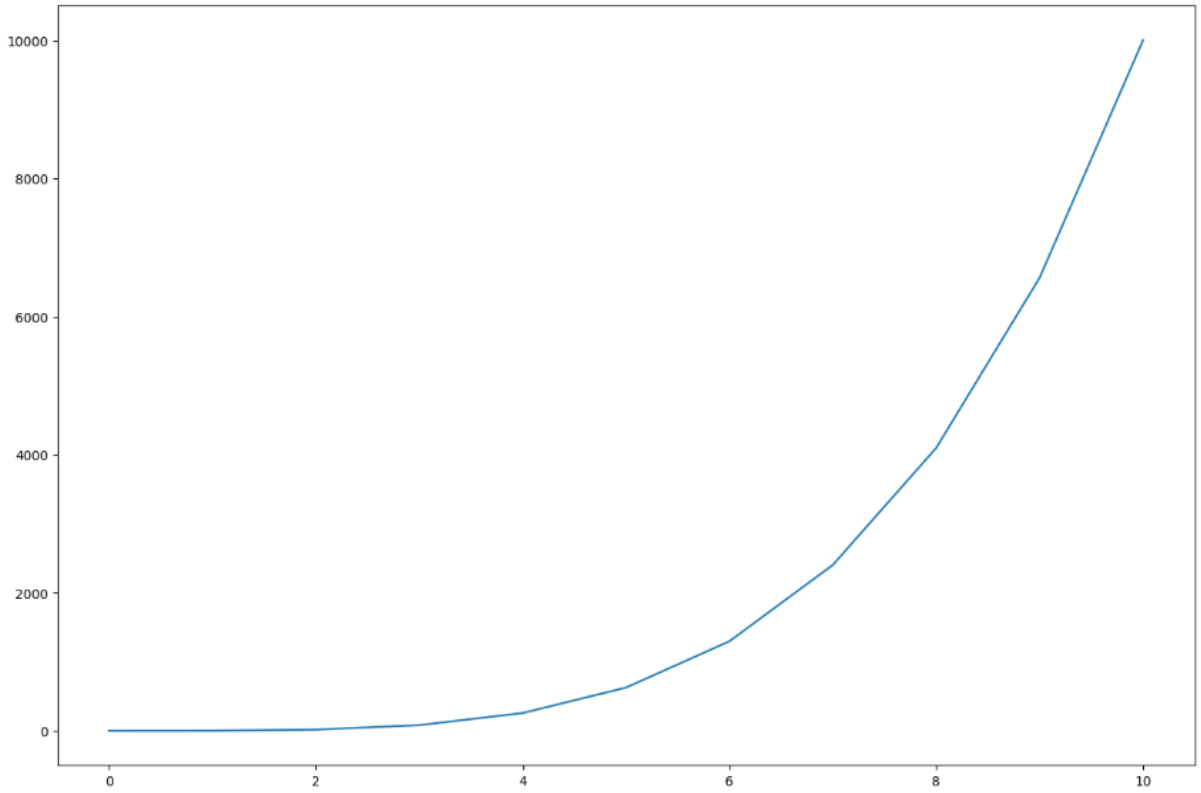


27. This code creates a figure with a specified size of 12 inches by 8 inches and a resolution (DPI) of 100.
28. It then adds a single axes (`axes1`) that occupies the entire figure area, and plots the data (`a` vs. `b`) onto this axes.

29. The `figsize` parameter controls the size of the figure, while `dpi` controls the resolution of the output figure.
30. The `add_axes` method places the axes within the figure, and `plot` draws the data on it.

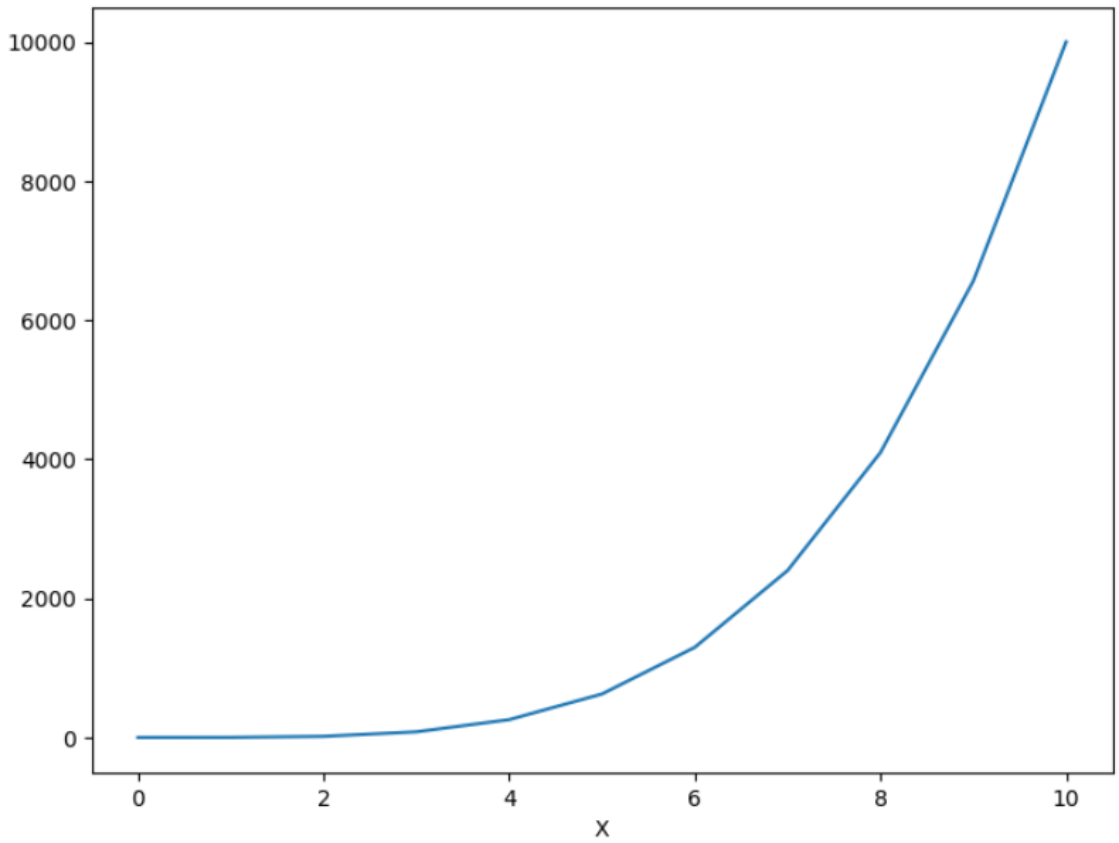
[33]:
```
# Creates blank canvas
fig = plt.figure(figsize=(12,8),dpi=100)

axes1 = fig.add_axes([0, 0, 1, 1])

axes1.plot(a,b)
```

[33]:
```
[<matplotlib.lines.Line2D at 0x28469db07d0>]
```



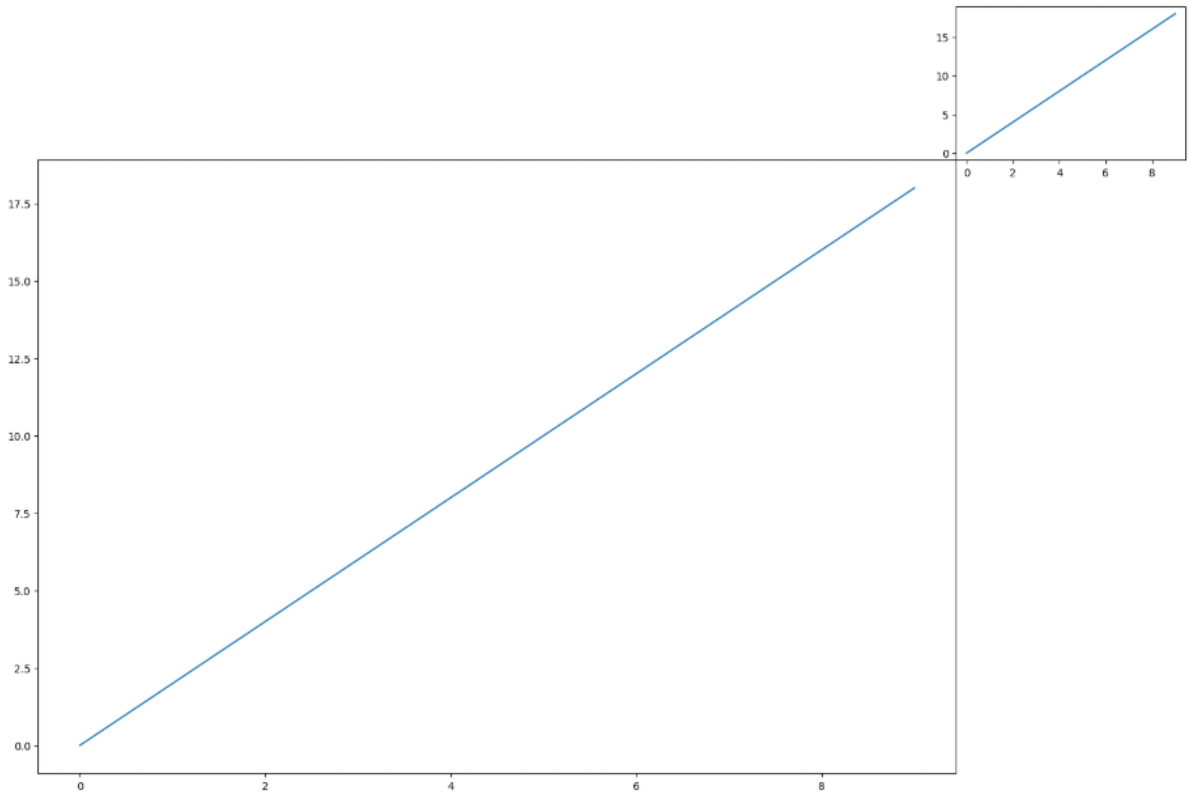
31. This code creates a figure and adds axes to it. The data (a vs. b) is plotted on the axes, with the x-axis labeled as 'X'.
32. The figure is then saved as a PNG file with the name 'figure.png'.
33. The bbox_inches='tight' option ensures that the bounding box of the saved figure tightly encloses the content, removing any unnecessary space around the plot.
34. This helps in saving the figure without extra margins.

[35]:
```
fig = plt.figure()

axes1 = fig.add_axes([0, 0, 1, 1])

axes1.plot(a,b)
axes1.set_xlabel('X')

# bbox_inches ='tight' automatically makes sure the bounding box is correct
fig.savefig('figure.png',bbox_inches='tight')
```



35. This code creates a figure with a size of 12x8 inches. It then adds two sets of axes to the figure: the first set (axes1) covers the full figure, and the second set (axes2) is a smaller figure that starts at the top-right corner.
36. Both sets of axes plot the data x vs. y. Finally, the figure is saved as 'test.png', with the bbox_inches='tight' option ensuring that the saved figure does not include unnecessary space around the plot, making it compact.

[37]:
```
# Creates blank canvas
fig = plt.figure(figsize=(12,8))

axes1 = fig.add_axes([0, 0, 1, 1]) # Full figure
axes2 = fig.add_axes([1, 1, 0.25, 0.25]) # Starts at top right corner!

# Larger Figure Axes 1
axes1.plot(x,y)

# Insert Figure Axes 2
axes2.plot(x,y)

fig.savefig('test.png',bbox_inches='tight')
```


15
10
5
0
0
2
4
6
8
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8