😊 Matplotlib Subplots

Matplotlib subplots refer to multiple plots or graphs arranged within a single **figure**. Each individual plot is created within its own **axes** (plotting area), and multiple axes can coexist in one figure—this layout is referred to as a subplot grid.

Why Use Subplots?

Subplots are useful when you want to:

- Compare multiple datasets side by side.
- Show different views of the same data.
- Present related data compactly in one visual frame.

Key Concepts

1. **Figure vs Axes**
    - A **figure** is the whole window or page.
    - An **axes** is a single plot inside the figure.
    - A figure can contain **one or many axes (subplots)**.
2. **Layout in Rows and Columns**
   Subplots are usually arranged using a grid system defined by rows and columns (e.g., 2 rows and 2 columns for four subplots).
3. **Independent Control**
   Each subplot (or axes) can have its own title, labels, limits, and style. This gives you fine-grained control over each visualization.
4. **Automatic Spacing**
   When using subplots, spacing between plots may need to be adjusted to avoid overlap of titles, labels, or legends.

Use Cases

- Visualizing time series data across multiple variables.
- Showing original data and transformations (like raw vs normalized).
- Displaying different chart types (bar, line, scatter) side by side for the same dataset.

In summary, **Matplotlib subplots** let you organize multiple plots in one figure, offering a powerful way to build comparative or multi-faceted visualizations efficiently.

😄 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. 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
```

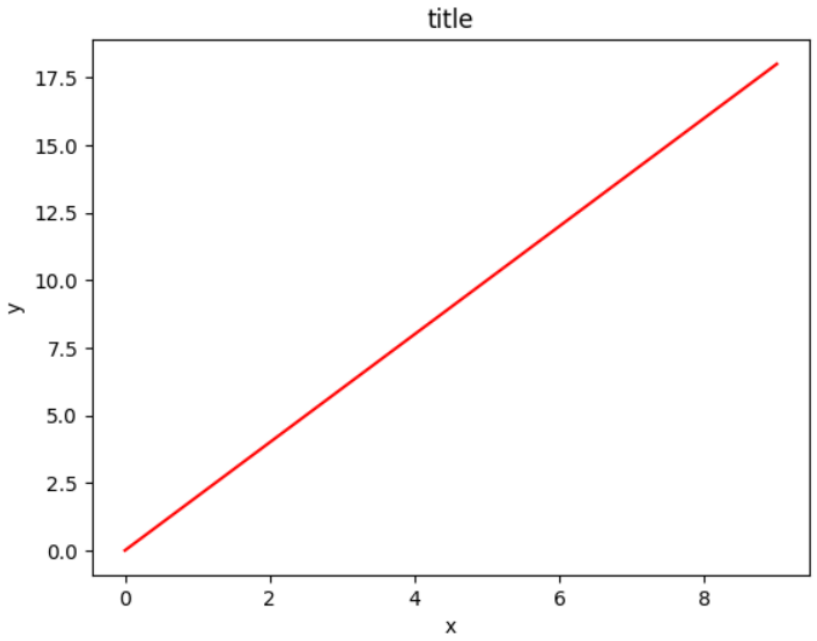
[14]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[16]:
```
y
```

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

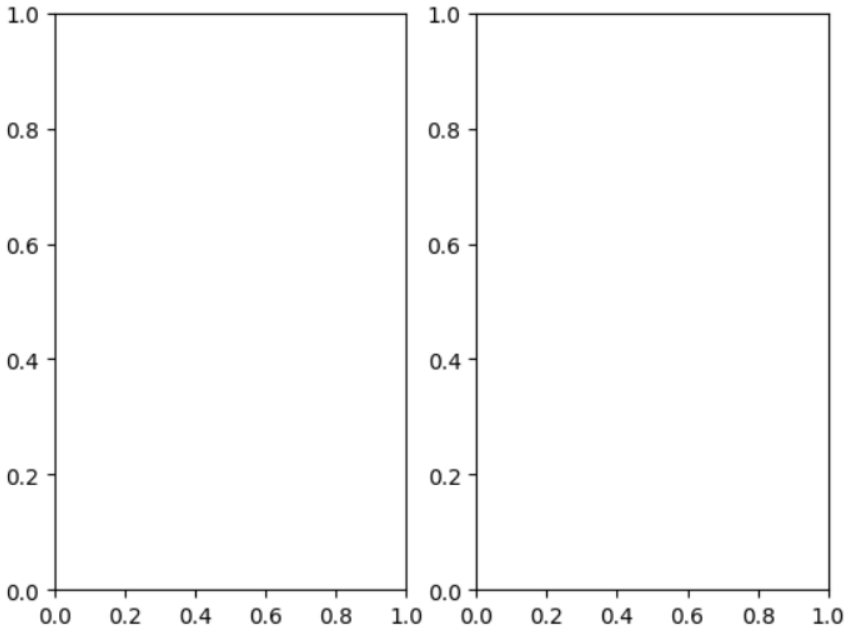
7. This code uses plt.subplots() to create a figure (fig) and a single subplot (axes) using tuple unpacking.
8. It plots the data x and y with a red line ('r') on the axes, then sets labels for the x-axis and y-axis using set_xlabel() and set_ylabel().
9. Finally, it adds a title to the plot using set_title('title'). The semicolon at the end prevents the plot object from displaying as output in environments like Jupyter Notebook.
10. This method is a cleaner and more scalable way to create plots, especially when dealing with multiple subplots.

[19]:
```
# Use similar to plt.figure() except use tuple unpacking to grab fig and axes
fig, axes = plt.subplots()

# Now use the axes object to add stuff to plot
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title'); #; hides Out[]
```



11. This code uses `plt.subplots()` to create multi-panel plots. The first call creates a figure with 1 row and 2 columns of subplots, resulting in an array of 2 axes.
12. The second call creates a 2x2 grid of subplots, giving a 2D NumPy array of 4 axes (shape `(2, 2)`).
13. The `axes` object holds references to each subplot, allowing individual plotting on each.
14. This layout is useful for comparing multiple plots side by side or in grid formats.
15. The `.shape` attribute helps confirm the structure of the subplot array.

[22]:
```
# Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)
```



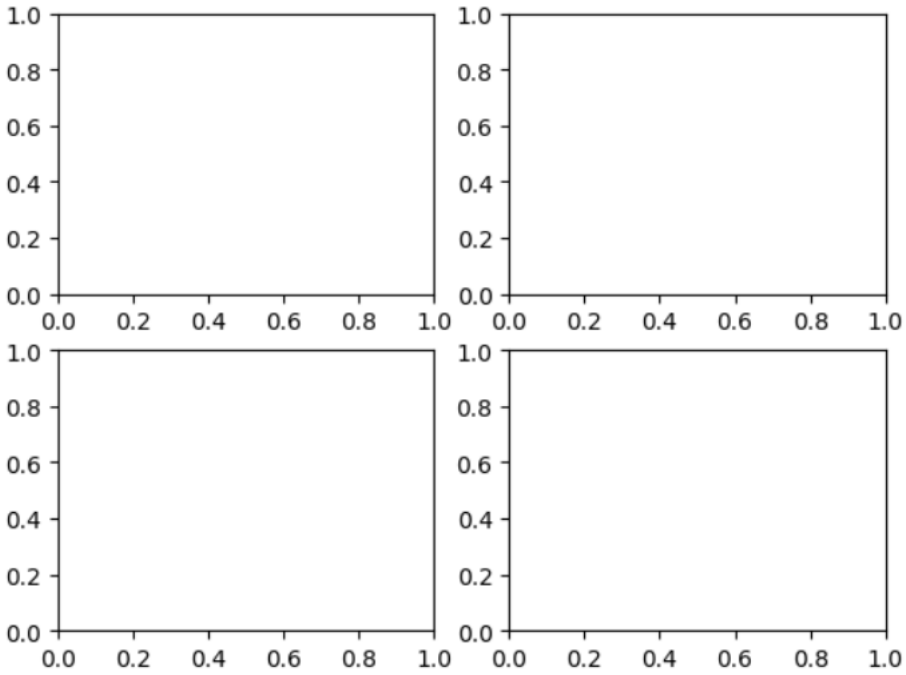
[24]:
```
# Axes is an array of axes to plot on
axes
```

[24]:
```
array([<Axes: >, <Axes: >], dtype=object)
```

[26]:
```
axes.shape
```

[26]:
```
(2,)
```

[28]:
```
# Empty canvas of 2 by 2 subplots
fig, axes = plt.subplots(nrows=2, ncols=2)
```



[30]:
```
axes
```

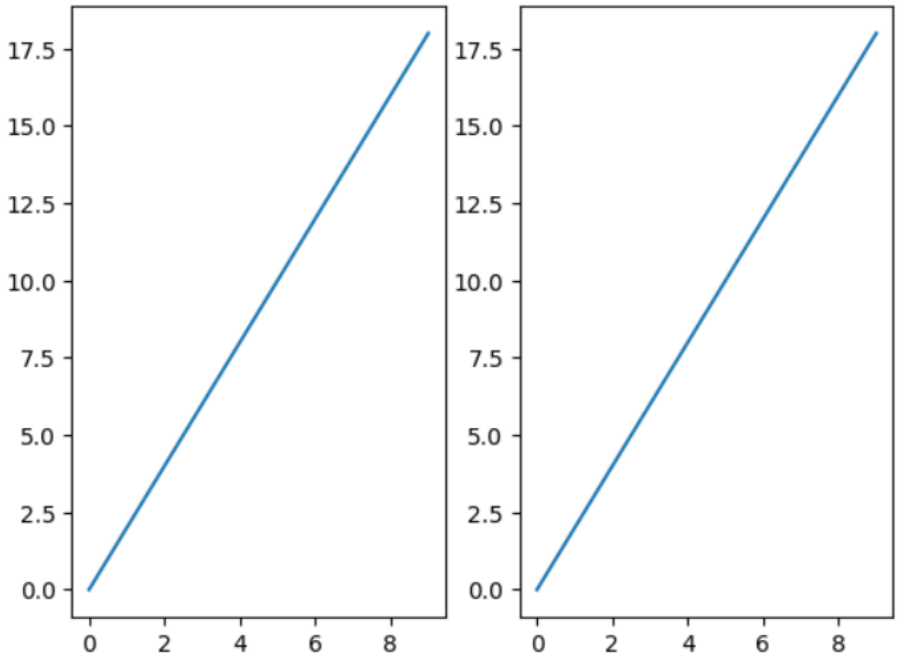
[30]:
```
array([[<Axes: >, <Axes: >],
       [<Axes: >, <Axes: >]], dtype=object)
```

[32]:
```
axes.shape
```

[32]:
```
(2, 2)
```

16. This code creates a single-row, two-column subplot layout using plt.subplots(nrows=1, ncols=2), producing two subplot axes.
17. A for loop then iterates over these axes and plots the same x and y data on each subplot using axe.plot(x, y).
18. This approach allows you to easily apply the same or similar plots to multiple subplots in one figure.

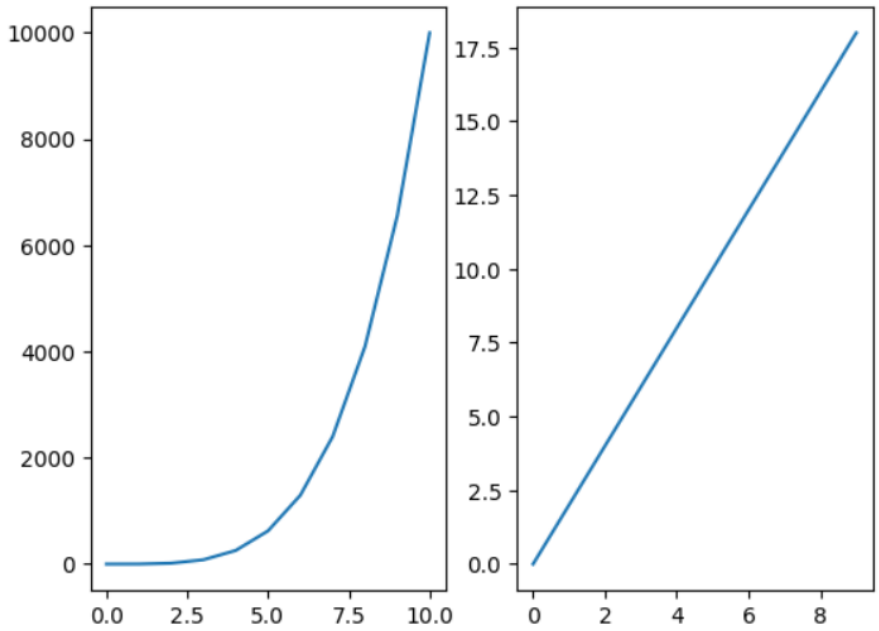
[35]:
```
fig,axes = plt.subplots(nrows=1,ncols=2)

for axe in axes:
    axe.plot(x,y)
```



19. This code creates a figure with two subplots arranged in one row using plt.subplots(nrows=1, ncols=2).
20. It then plots different data on each subplot: the first subplot (axes[0]) plots a vs. b, and the second subplot (axes[1]) plots x vs. y.
21. This is useful for displaying multiple related plots side by side for comparison.

```
[37]: fig,axes = plt.subplots(nrows=1,ncols=2)

      axes[0].plot(a,b)
      axes[1].plot(x,y)
```

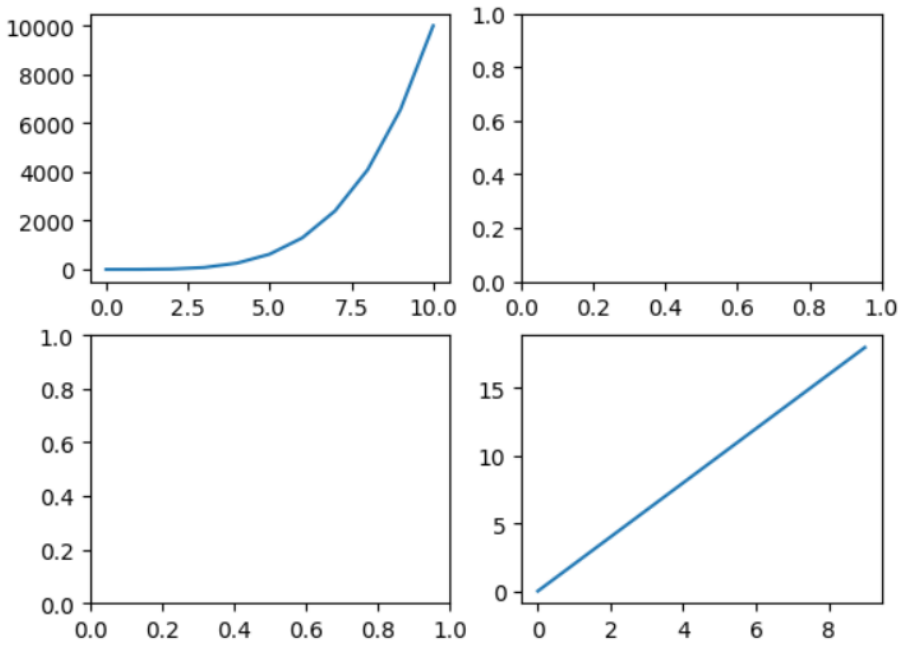
```
[37]: [<matplotlib.lines.Line2D at 0x20c44c8c290>]
```



22. This code creates a 2×2 grid of subplots using plt.subplots(nrows=2, ncols=2), which returns a 2D array of axes.
23. It then plots a vs. b in the top-left subplot (axes[0][0]) and x vs. y in the bottom-right subplot (axes[1][1]).
24. This structure helps organize multiple plots neatly within one figure.

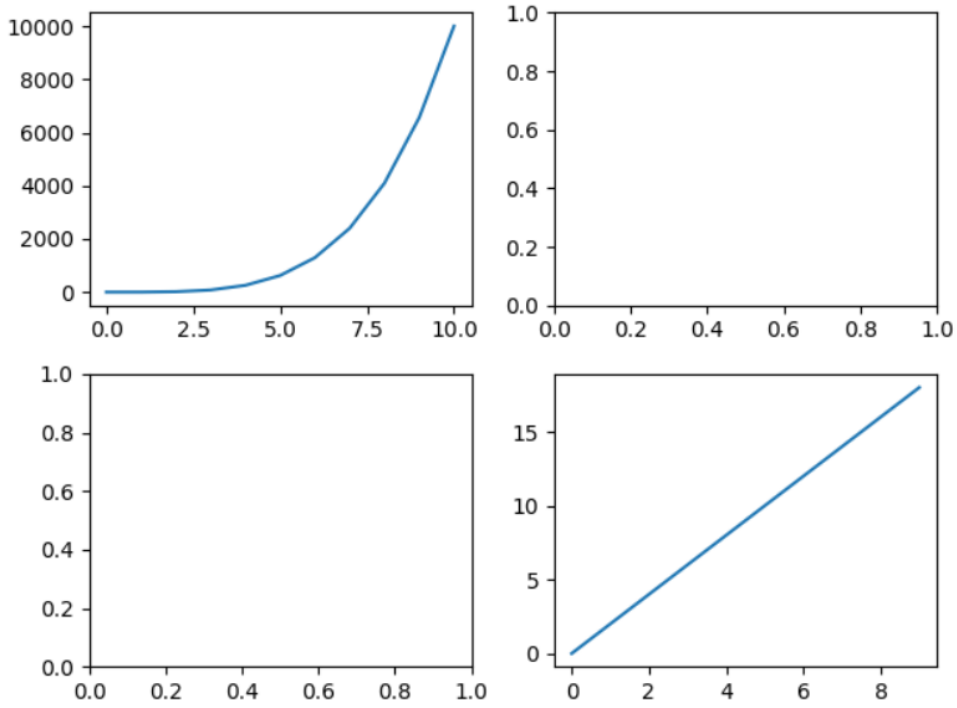
[39]:
```
# NOTE! This returns 2 dimensional array
fig,axes = plt.subplots(nrows=2,ncols=2)

axes[0][0].plot(a,b)
axes[1][1].plot(x,y)
```

[39]:
```
[<matplotlib.lines.Line2D at 0x20c44bf22d0>]
```



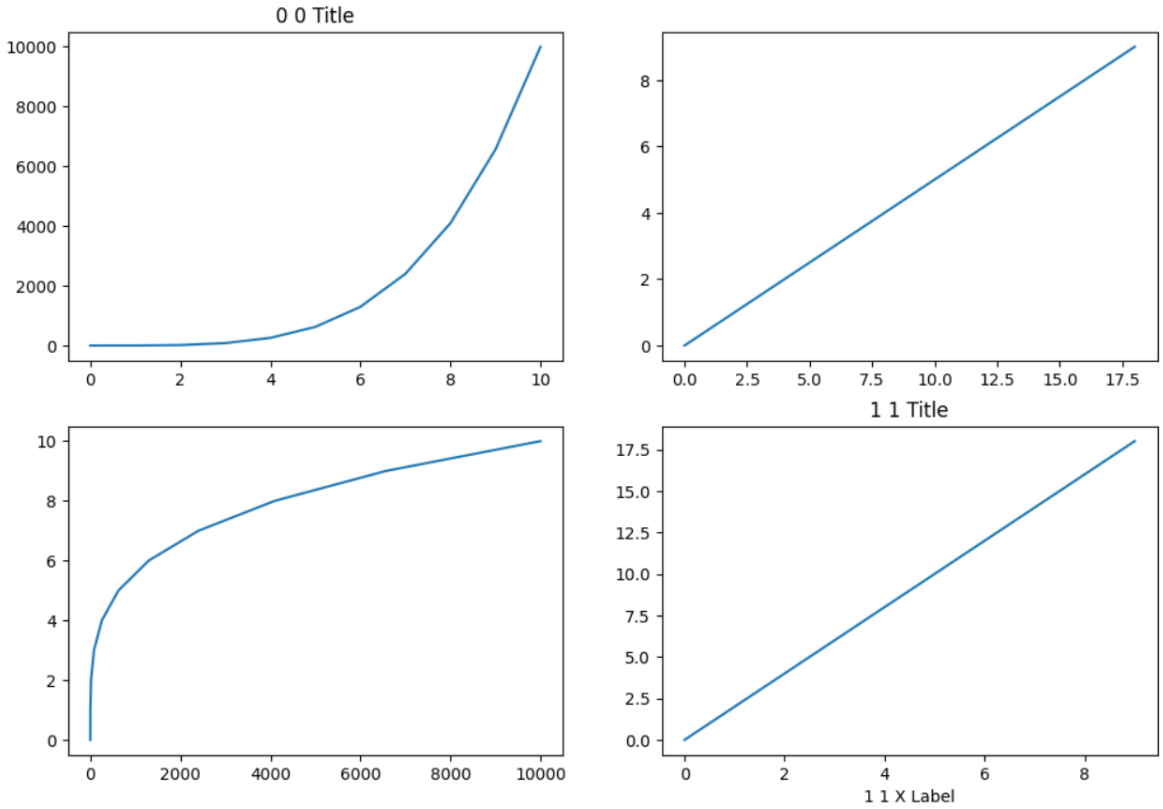
25. This code creates a 2×2 grid of subplots using `plt.subplots`, which returns a 2D NumPy array of axes.
26. It plots `a` vs. `b` in the top-left subplot (`axes[0][0]`) and `x` vs. `y` in the bottom-right subplot (`axes[1][1]`).
27. The call to `plt.tight_layout()` adjusts the spacing between subplots to prevent overlapping elements, improving the overall layout and readability of the figure.

[42]:
```
# NOTE! This returns 2 dimensional array
fig,axes = plt.subplots(nrows=2,ncols=2)

axes[0][0].plot(a,b)
axes[1][1].plot(x,y)

plt.tight_layout()
```



28. This code creates a 2×2 grid of subplots with a specified figure size of 12×8 inches.
29. Each subplot (axes[row][col]) is customized individually: data is plotted, and titles or axis labels are set where needed.
30. For example, the top-left plot has a title, and the bottom-right has both a title and X-axis label.
31. The entire figure is given a main title (suptitle). This distinction between subplot-level and figure-level customization helps organize complex visualizations clearly

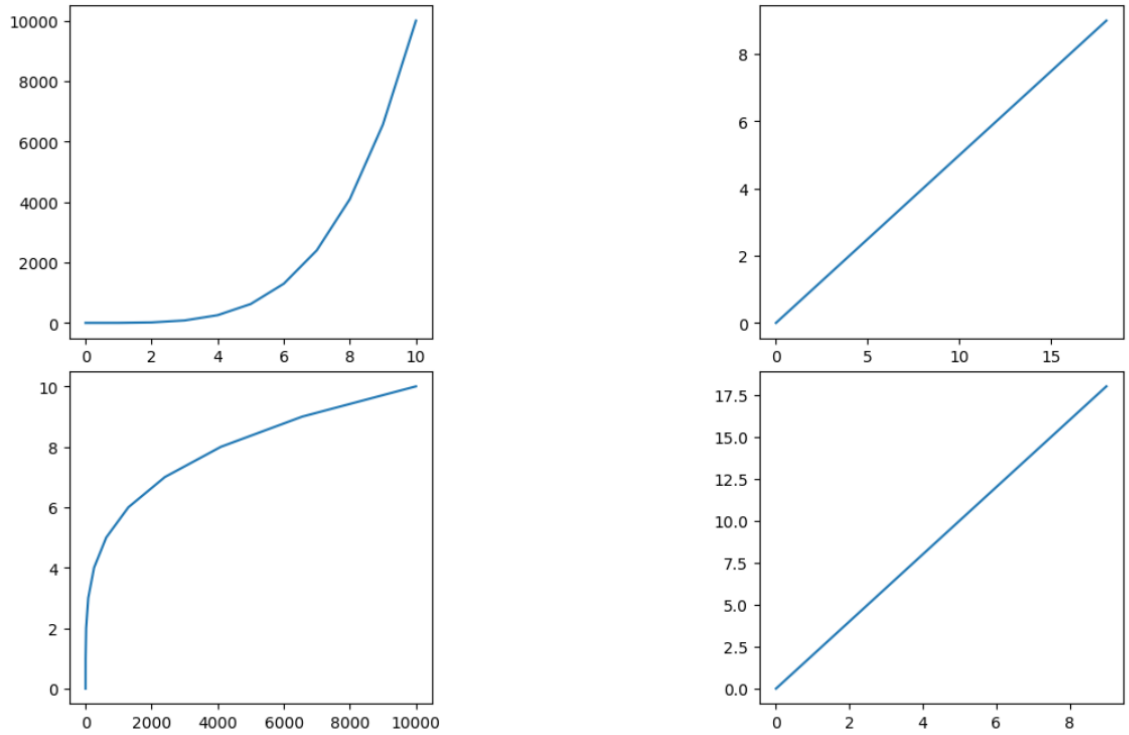
[45]:
```
fig,axes = plt.subplots(nrows=2,ncols=2,figsize=(12,8))

# SET YOUR AXES PARAMETERS FIRST

# Parameters at the axes level
axes[0][0].plot(a,b)
axes[0][0].set_title('0 0 Title')


axes[1][1].plot(x,y)
axes[1][1].set_title('1 1 Title')
axes[1][1].set_xlabel('1 1 X Label')

axes[0][1].plot(y,x)
axes[1][0].plot(b,a)

# THEN SET OVERALL FIGURE PARAMETERS

# Parameters at the Figure level
fig.suptitle("Figure Level",fontsize=16)


plt.show()
```



32. This code creates a 2×2 grid of subplots using plt.subplots() and customizes their layout using fig.subplots_adjust().
33. Each subplot is filled with a different plot of the variables a, b, x, and y. The wspace=0.9 argument increases the horizontal space between the subplots, while hspace=0.1 reduces the vertical space between them.
34. The other layout parameters (left, right, top, bottom) are left as default (None).

35. This allows fine-tuning of subplot spacing within the overall figure for better readability or aesthetic purposes.

```
[48]: fig,axes = plt.subplots(nrows=2,ncols=2,figsize=(12,8))

# Parameters at the axes level
axes[0][0].plot(a,b)
axes[1][1].plot(x,y)
axes[0][1].plot(y,x)
axes[1][0].plot(b,a)

# Use left,right,top, bottom to stretch subplots
# Use wspace,hspace to add spacing between subplots
fig.subplots_adjust(left=None,
    bottom=None,
    right=None,
    top=None,
    wspace=0.9,
    hspace=0.1,)

plt.show()
```



36. This code creates a 2×2 grid of subplots using plt.subplots() with a figure size of 12 by 8 inches.
37. Each subplot is populated with different plots using the plot() function on specific axes.
38. The resulting figure is saved as an image file named 'subplots.png', and bbox_inches='tight' ensures the saved image trims unnecessary whitespace.
39. Finally, plt.show() displays the figure. This approach is useful for organizing multiple plots in one figure and exporting it cleanly.

[51]:
```
# NOTE! This returns 2 dimensional array
fig,axes = plt.subplots(nrows=2,ncols=2,figsize=(12,8))

axes[0][0].plot(a,b)
axes[1][1].plot(x,y)
axes[0][1].plot(y,x)
axes[1][0].plot(b,a)

fig.savefig('subplots.png',bbox_inches='tight')

plt.show()
```