😊 Matplotlib Basics

Matplotlib is a comprehensive and widely used data visualization library in Python. It is primarily used for creating static, interactive, and animated plots and graphs in two dimensions. It serves as the foundational plotting library for many other Python visualization tools and is particularly popular in data analysis, scientific computing, and machine learning workflows.

Key Features of Matplotlib

1. **Versatile Plotting**
   It supports a wide range of plot types including:
   - Line plots
   - Bar charts
   - Histograms
   - Scatter plots
   - Pie charts
   - Box plots
   - Heatmaps
   - Error bars and more
2. **Fine-grained Control**
   You can customize almost every element of a plot including axes, labels, titles, ticks, fonts, colors, line styles, and annotations.
3. **Integration**
   - Works well with NumPy and Pandas.
   - Plots can be embedded in applications using GUI toolkits like Tkinter, Qt, or web applications using frameworks like Flask.
4. **Interactive Capabilities**
   Interactive backends allow zooming, panning, and updating plots in real time.
5. **Export Options**
   You can export plots to various formats such as PNG, PDF, SVG, and EPS.

Main Components

- **Pyplot module (matplotlib.pyplot)**
  This is the most commonly used interface of Matplotlib. It provides a MATLAB-like procedural API for plotting.
- **Figure**
  The entire figure or canvas where all plots and visual elements are drawn.

- **Axes**
  The area on which individual plots are drawn within a figure. A figure can have multiple axes (subplots).
- **Plot Elements**
  Includes titles, axis labels, legends, grid lines, ticks, and annotations.

Use Cases of Matplotlib

- Exploratory Data Analysis (EDA)
- Data presentation and reporting
- Visualizing trends and patterns
- Scientific and academic research
- Dashboard and GUI integration for visual reporting

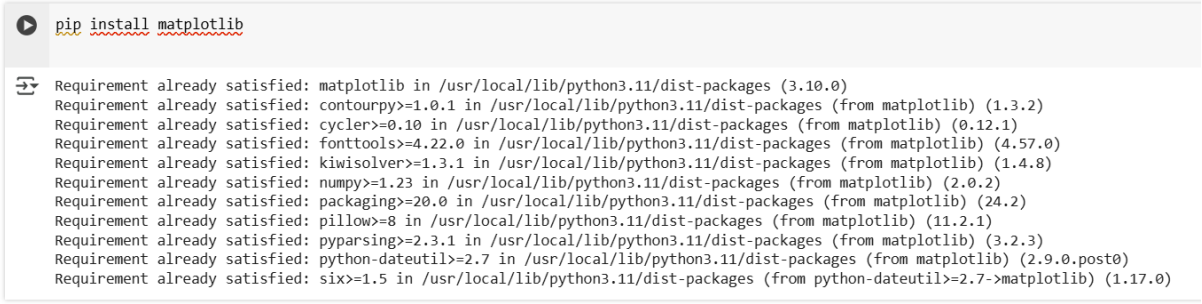
Summary

Matplotlib is a fundamental library for plotting in Python, offering the flexibility to create publication-quality figures and detailed control over every visual element. It's often used as a foundation for more advanced libraries like Seaborn and Plotly.

😄 To begin with the Lab

1. In this lab, we are going to see the fundamentals of the Matplotlib Library. First, we will install Matplotlib in our Jupyter notebook using the command below.

   !pip install matplotlib



```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

2. This code imports matplotlib.pyplot as plt for plotting and %matplotlib inline (used in Jupyter notebooks) to display plots directly below the code.
3. It also imports numpy as np and uses np.arange(0,10) to create an array of integers from 0 to 9.
4. Then it creates y = 2*x, which means each value in x is multiplied by 2.
5. This sets up two arrays, x and y, to be used for plotting. The .pyplot module of matplotlib is essential for creating visualizations like line graphs, scatter plots, etc

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

     import matplotlib.pyplot as plt
```

```
[3]  %matplotlib inline
```

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

```
[5]  x = np.arange(0,10)
```

```
[6]  y = 2*x
```

```
[7]  x
```

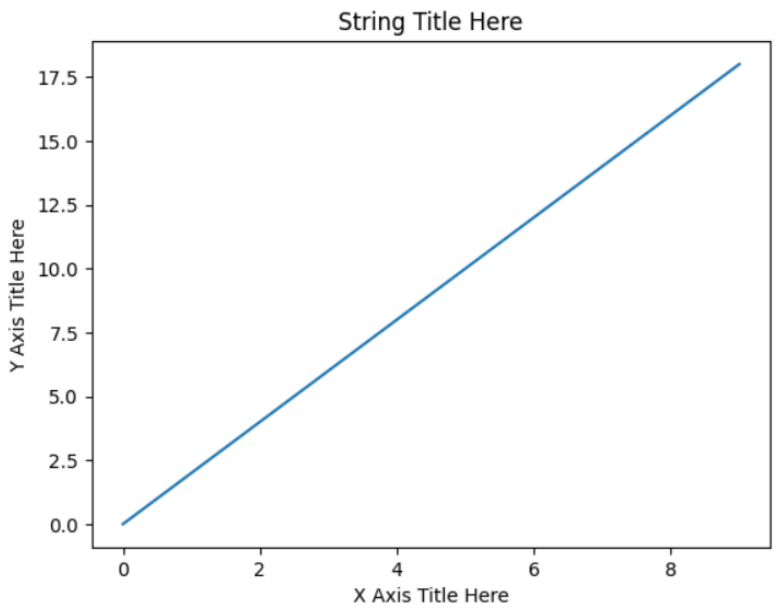
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[8]  y
```

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

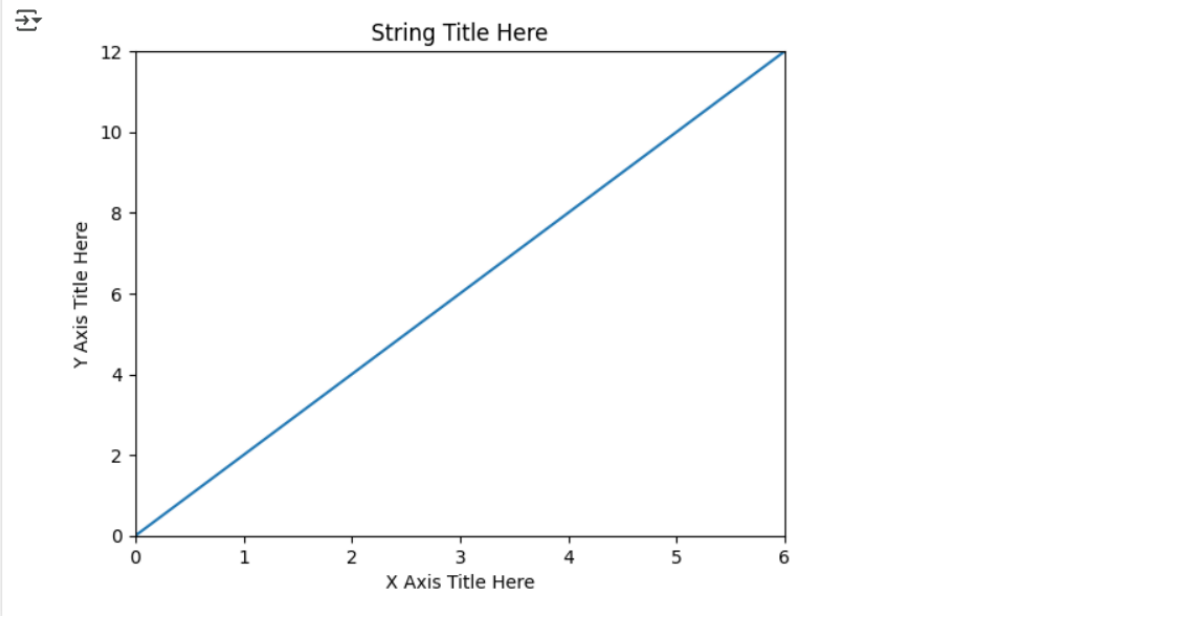
6. This code uses matplotlib.pyplot to create a simple line plot. plt.plot(x, y) draws a line graph using the x and y values defined earlier. plt.xlabel() and plt.ylabel() add labels to the x-axis and y-axis, respectively. plt.title() sets the title of the graph.
7. plt.show() displays the plot; it's especially necessary outside Jupyter notebooks to render the graph, and it also prevents extra output like Out[].
8. These functions help in building clear and informative visualizations.

```
[9]  plt.plot(x, y)
     plt.xlabel('X Axis Title Here')
     plt.ylabel('Y Axis Title Here')
     plt.title('String Title Here')
     plt.show() # Required for non-jupyter users , but also removes Out[] info
```



9. This code creates a line plot with custom axis limits using matplotlib.pyplot. After plotting x vs y with plt.plot(x, y), it adds axis labels and a title.
10. The plt.xlim(0,6) and plt.ylim(0,12) functions set the visible range of the x-axis (0 to 6) and y-axis (0 to 12), effectively zooming into a specific portion of the plot.
11. Finally, plt.show() displays the plot, ensuring it's rendered correctly, especially in non-Jupyter environments.

```
plt.plot(x, y)
plt.xlabel('X Axis Title Here')
plt.ylabel('Y Axis Title Here')
plt.title('String Title Here')
plt.xlim(0,6) # Lower Limit, Upper Limit
plt.ylim(0,12) # Lower Limit, Upper Limit
plt.show() # Required for non-jupyter users , but also removes Out[] info
```



12. This code plots a line graph of x versus y using plt.plot(x, y) and then saves the resulting figure as an image file named 'example.png' using plt.savefig('example.png').
13. The savefig() function captures the current figure and stores it in the specified file format (PNG in this case), allowing you to save visualizations for reports or sharing.

```
[12] plt.plot(x,y)
     plt.savefig('example.png')
```