Matplotlib is a Python library used for creating graphs and visualizations. Think of it like a digital drawing tool that helps you turn numbers into pictures. If you can understand charts in Excel, you can learn Matplotlib easily.

🔹 **Step 1: Install Matplotlib**

If you don't have it yet, install it using:

```
pip install matplotlib
```

🔹 **Step 2: Import Matplotlib**

You need to import it before using:

```
import matplotlib.pyplot as plt
```

We use plt as a shortcut to make coding easier.

📌 **Basic Plotting**

Let's plot a simple **line graph**.

```
import matplotlib.pyplot as plt
# Data
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
# Create plot
plt.plot(x, y)
# Show plot
plt.show()
```

✅ This will draw a line connecting the points.

📌 **Adding Labels and Title**

```
plt.plot(x, y)
# Add labels and title
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis Label")
plt.title("My First Graph")
plt.show()
```

✅ Now your graph has a title and axis labels.

📌 **Changing Line Style & Color**

```
plt.plot(x, y, color='red', linestyle='dashed', marker='o')
plt.show()
```

✅ This makes a **red dashed line** with **circle markers** at each point.

📌 **Bar Chart**

```
plt.bar(["A", "B", "C", "D"], [5, 7, 3, 8], color='green')
plt.show()
```

✅ This creates a **bar chart**.

📌 **Histogram**

```
import numpy as np
data = np.random.randn(1000)  # 1000 random numbers
plt.hist(data, bins=30, color='purple')
plt.show()
```

✅ This creates a **histogram** (useful for data distribution).

📌 **Scatter Plot**

```
plt.scatter([1, 2, 3, 4], [10, 20, 25, 30], color='blue')
plt.show()
```

✅ This makes a **scatter plot** (dots instead of a line).

📌 **Pie Chart**

```
labels = ["Apples", "Bananas", "Cherries"]
sizes = [40, 30, 30]
plt.pie(sizes, labels=labels, autopct="%1.1f%%")
plt.show()
```

✅ This creates a **pie chart** with percentage labels.

📌 **Saving Your Plot**

```
plt.plot(x, y)
plt.savefig("myplot.png")  # Saves as an image
plt.show()
```

✅ This saves the plot as a **PNG file**.

🔹 **1. Adding Grid, Legends, and Annotations**

✅ **Grid**

Grids make it easier to read values on the graph.

```
import matplotlib.pyplot as plt
```

```
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
plt.plot(x, y, label="Growth", color='blue')
# Add grid
plt.grid(True)
plt.legend()  # Shows the legend
plt.show()
```

- plt.grid(True) → Adds a grid
- plt.legend() → Displays the label on the graph

✅ **Annotations (Text on Graph)**

Use this to highlight key points!

```
plt.plot(x, y, label="Growth", color='blue')
# Add annotation
plt.annotate("Highest Point", xy=(5, 40), xytext=(3, 35),
             arrowprops=dict(facecolor='red', shrink=0.05))

plt.legend()
plt.show()
```

- plt.annotate(text, xy, xytext, arrowprops) → Adds arrows & labels

---

🔹 **2. Subplots (Multiple Charts in One Figure)**

Use plt.subplot(rows, cols, index) to create multiple plots together.

```
import numpy as np
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.figure(figsize=(10,5))  # Set figure size
# First plot
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.plot(x, y1, color='red')
plt.title("Sine Wave")
# Second plot
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
plt.plot(x, y2, color='blue')
```

```
plt.title("Cosine Wave")
plt.show()
```

✅ This creates **side-by-side** graphs!

🔹 3. 3D Plotting in Matplotlib

Matplotlib can also create 3D graphs!

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
ax.plot_surface(X, Y, Z, cmap="viridis")
plt.show()
```

✅ This creates a **3D surface plot** with a color map!

🔹 4. Interactive Graphs with Zoom & Hover

Matplotlib can be **interactive** when used with mplcursors!

```
import mplcursors
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
plt.plot(x, y, marker="o")
# Make interactive
mplcursors.cursor()
plt.show()
```

✅ Now, when you **hover** over a point, it shows its value!

🔹 1. Advanced Styling (Fonts, Background, Themes)

✅ **Changing Font Style & Size**

You can modify fonts for better readability.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
plt.plot(x, y, label="Growth", color='blue')
```

```
# Custom fonts
plt.xlabel("X-axis", fontsize=14, fontweight='bold', fontfamily='serif')
plt.ylabel("Y-axis", fontsize=14, fontstyle='italic')
plt.title("Styled Graph", fontsize=16, fontweight='bold')
plt.legend()
plt.show()
```

- fontsize, fontweight, fontstyle, fontfamily → Customize text style

---

✅ **Changing Background Color**

```
plt.figure(facecolor='lightgray')  # Set figure background
plt.plot(x, y, color="red")
plt.show()
```

- facecolor='color' → Changes background color

🔹 **2. Seaborn Integration (Beautiful Plots)**

Seaborn is a library that makes Matplotlib plots more **beautiful and professional**.

First, install Seaborn if you haven't:

```
pip install seaborn
```

✅ **Using Seaborn for Styling**

```
import seaborn as sns
sns.set_style("darkgrid")  # Choose theme
plt.plot(x, y, color='blue')
plt.show()
```

- Try different styles: "darkgrid", "whitegrid", "dark", "white", "ticks"

🔹 **3. Creating Animated Plots (Real-Time Data)**

✅ **Basic Animation**

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
fig, ax = plt.subplots()
x = np.arange(0, 10, 0.1)
y = np.sin(x)
line, = ax.plot(x, y)
def update(frame):
```

```
    line.set_ydata(np.sin(x + frame / 10.0))
    return line,
ani = animation.FuncAnimation(fig, update, frames=100, interval=50)
plt.show()
```

✅ This creates a **moving sine wave**!

🔹 4. Handling Large Datasets Efficiently

When dealing with **millions of data points**, Matplotlib can slow down. Here's how to **optimize performance**:

✅ Use Line Simplification

```
plt.plot(x, y, linestyle='-', linewidth=0.5)  # Thin lines reduce processing
plt.show()
```

✅ Use Scatter Instead of Line Plot

For large datasets, scatter plots render faster.

```
plt.scatter(x, y, s=1)  # s=1 makes dots smaller
plt.show()
```

✅ Downsampling Large Data

```
x = x[::10]  # Keep every 10th data point
y = y[::10]
plt.plot(x, y)
plt.show()
```

🎯 Summary

✓ **Advanced Styling** → Fonts, backgrounds, themes
✓ **Seaborn Integration** → Prettier graphs
✓ **Animated Plots** → Real-time data visualization
✓ **Handling Large Datasets** → Faster rendering

Now you've learned **100% of Matplotlib**! 🎯 🔥

Would you like to test your skills with a **practice challenge**? 🚀