



# MATPLOTLIB

## 👉 Welcome to *Matplotlib Mastery*

This notebook is my **visualization workshop** for mastering **Matplotlib**, the foundational library behind **data visualization in Python**.

Here, I will:

- Build **strong plotting fundamentals**
  - Customize plots from **basic to publication-ready**
  - Understand data through **visual intuition**
  - Write **clear, flexible, and reusable plotting code**
- 

## 🎯 Mission

To **master Matplotlib deeply**, not just draw plots — but to **control every visual detail with confidence**.

---

## 🚀 What You'll Find Inside

### 📌 Matplotlib Fundamentals

- `pyplot` basics
- Figures & Axes anatomy
- Object-Oriented (OO) approach

### 📌 Core Plots

- Line plots
- Scatter plots
- Bar & horizontal bar charts
- Histograms

## 📌 Customization Essentials

- Titles, labels, legends
- Colors, markers, linestyles
- Grid, ticks & annotations

## 📌 Subplots & Layouts

- `subplot()` vs `subplots()`
- Figure sizing
- Tight & constrained layouts

## 📌 Statistical & Distribution Plots

- Histograms
- Boxplots
- Error bars

## 📌 Advanced Visualization

- Multiple axes
  - Styling best practices
  - Exporting high-quality figures
- 

## 🤝 Let's Learn Together

If this notebook:

- sharpens your plotting skills 
- improves your visual thinking 
- or helps you present data better 

feel free to **connect, share, and grow together.**

---

✍ Created & Maintained by

**Babar Ali Mahar**

 *Sharing my data visualization journey on LinkedIn — one notebook at a time.*

---

 “Matplotlib doesn’t limit creativity —  
it rewards those who master the details.”

# Introduction to Matplotlib

Matplotlib is one of the most widely used **data visualization libraries in Python**. It provides a flexible and powerful way to create plots, graphs, and charts, helping us **understand patterns, trends, and insights** in data.

---

## Key Points

- **Library type:** Low-level plotting library
  - **Core Idea:** Gives full control over every element of a figure (titles, axes, labels, colors, styles, etc.)
  - **Origin:** Inspired by MATLAB’s plotting system, hence its name **Matplotlib**
  - **Flexibility:** Can create anything from simple line plots to highly customized multi-plot figures
  - **Integration:** Works seamlessly with NumPy, Pandas, and SciPy
- 

## Why Use Matplotlib?

- To **visualize data** effectively before analysis and modeling
  - To create **publication-quality figures**
  - To **customize every detail** of plots (size, colors, fonts, legends, grids)
  - Acts as the **foundation** for other visualization libraries like **Seaborn**, **Pandas .plot()**, and **Plotly (partially inspired)**
- 

## Anatomy of a Matplotlib Figure

A Matplotlib figure consists of several key components:

1. **Figure** → The whole window or page that contains everything
2. **Axes** → The actual area where data is plotted (x-axis, y-axis)
3. **Axis** → The scales and ticks for x and y directions
4. **Plot elements** → Titles, labels, legends, lines, bars, markers, etc.

---

# Typical Workflow

## 1. Import the library

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

## Basic Example: Plotting Two Arrays

python Copy code

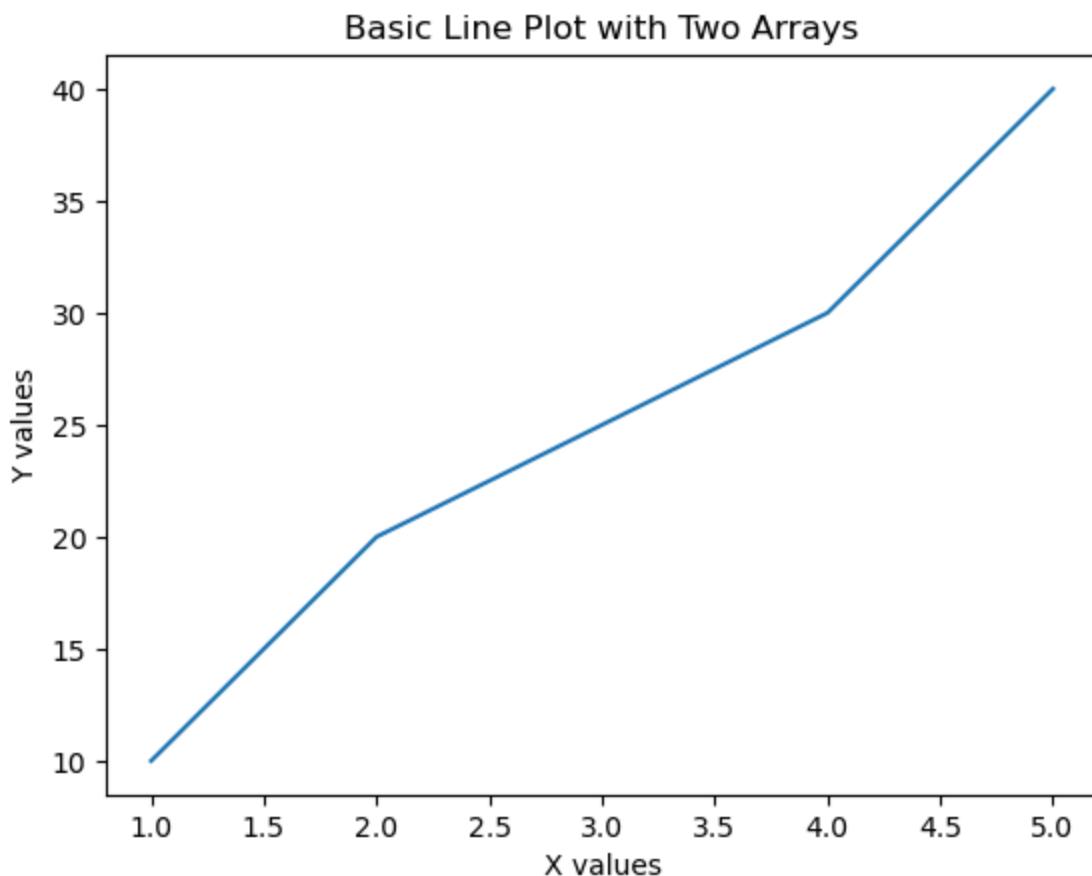
```
In [1]: import matplotlib.pyplot as plt

# Two simple arrays
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

# Create a basic line plot
plt.plot(x, y)

# Add labels and title
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Basic Line Plot with Two Arrays")

# Show the plot
plt.show()
```



## Canvas in Matplotlib

A **canvas** is the entire drawing area where all your plots live think of it as a blank sheet of paper on which you can add one or many plots.

You create a canvas with `plt.figure()` or `plt.subplots()`.

```
In [2]: import matplotlib.pyplot as plt

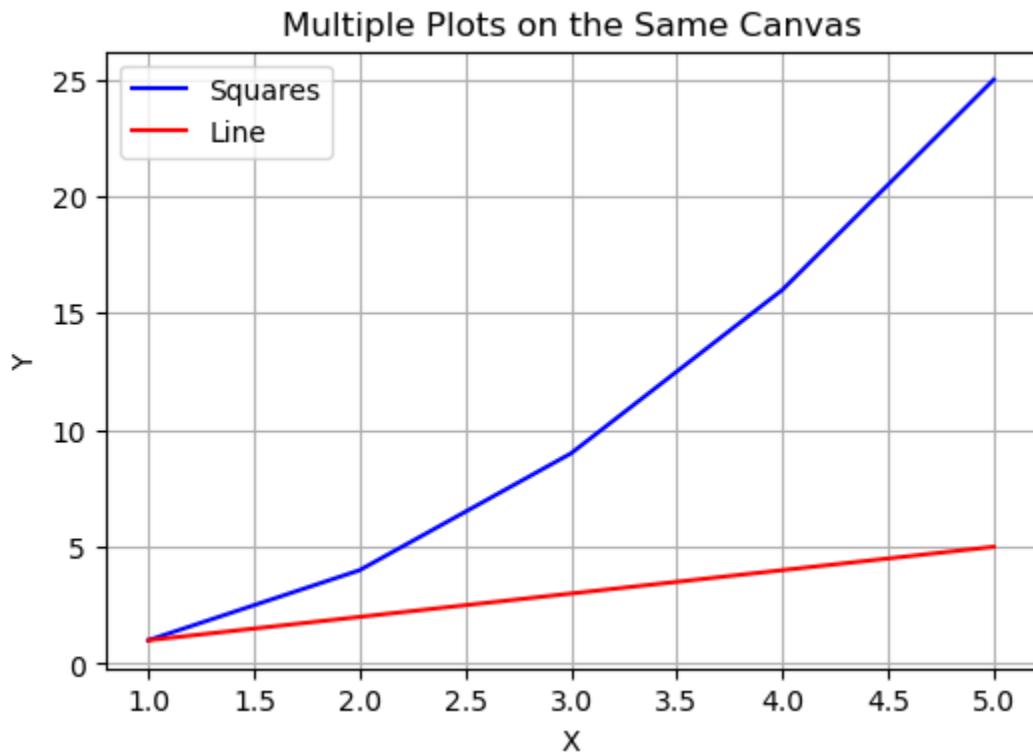
# Create one canvas and draw multiple plots on it
x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [1, 2, 3, 4, 5]

plt.figure(figsize=(6,4))    # <-- create the canvas

plt.plot(x, y1, label="Squares", color="blue")
plt.plot(x, y2, label="Line", color="red")

plt.title("Multiple Plots on the Same Canvas")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid(True)
```

```
plt.show()
```



## Subplots with plt.subplot()

`plt.subplot()` is a **simple way** to place multiple plots on one figure.

- Syntax: `plt.subplot(nrows, ncols, index)`
- It splits the figure into a grid (`nrows × ncols`)
- `index` tells which cell to draw in (counted left → right, top → bottom)

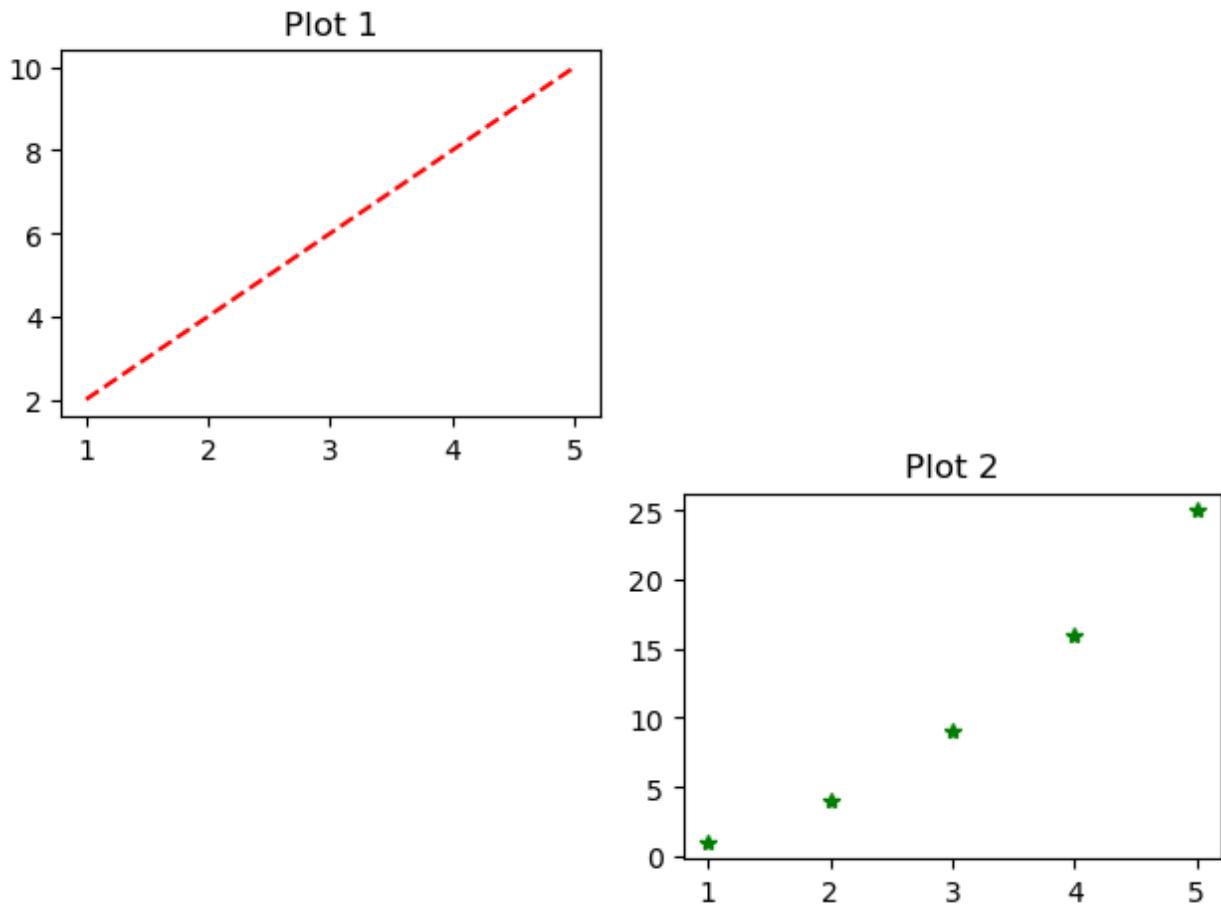
```
In [3]: import matplotlib.pyplot as plt
```

```
x = [1, 2, 3, 4, 5]
y1 = [2, 4, 6, 8, 10]
y2 = [1, 4, 9, 16, 25]

# First plot in position 1 of a 2x2 grid
plt.subplot(2, 2, 1)
plt.plot(x, y1, 'r--')
plt.title("Plot 1")

# Second plot in position 4 of a 2x2 grid
plt.subplot(2, 2, 4)
plt.plot(x, y2, 'g*')
plt.title("Plot 2")
```

```
plt.tight_layout()  
plt.show()
```



## Matplotlib Object-Oriented Interface

Instead of using only `plt` commands, the **OO style** gives more control:

1. Create a **Figure** → acts as the **canvas**
2. Add one or more **Axes** → areas where plots live

This is preferred for professional or complex visualizations.

```
In [4]: import matplotlib.pyplot as plt  
  
# Sample data  
x = [1, 2, 3, 4, 5]  
y = [2, 4, 6, 8, 10]  
  
# Step 1: Create a Figure (canvas)  
fig = plt.figure(figsize=(6,4))  
  
# Step 2: Add Axes (x0, y0, width, height) in 0-1 range  
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
```

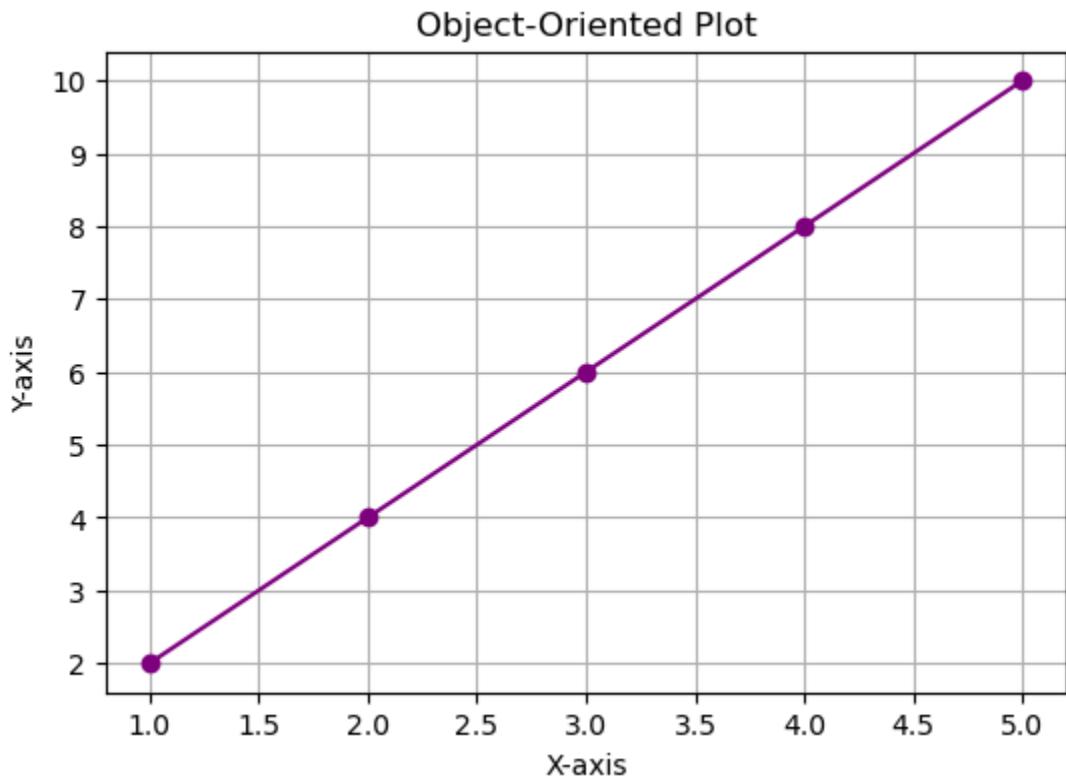
```

# Step 3: Plot data on the Axes
ax.plot(x, y, color="purple", marker="o", linestyle="-")

# Step 4: Customize
ax.set_title("Object-Oriented Plot")
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.grid(True)

plt.show()

```



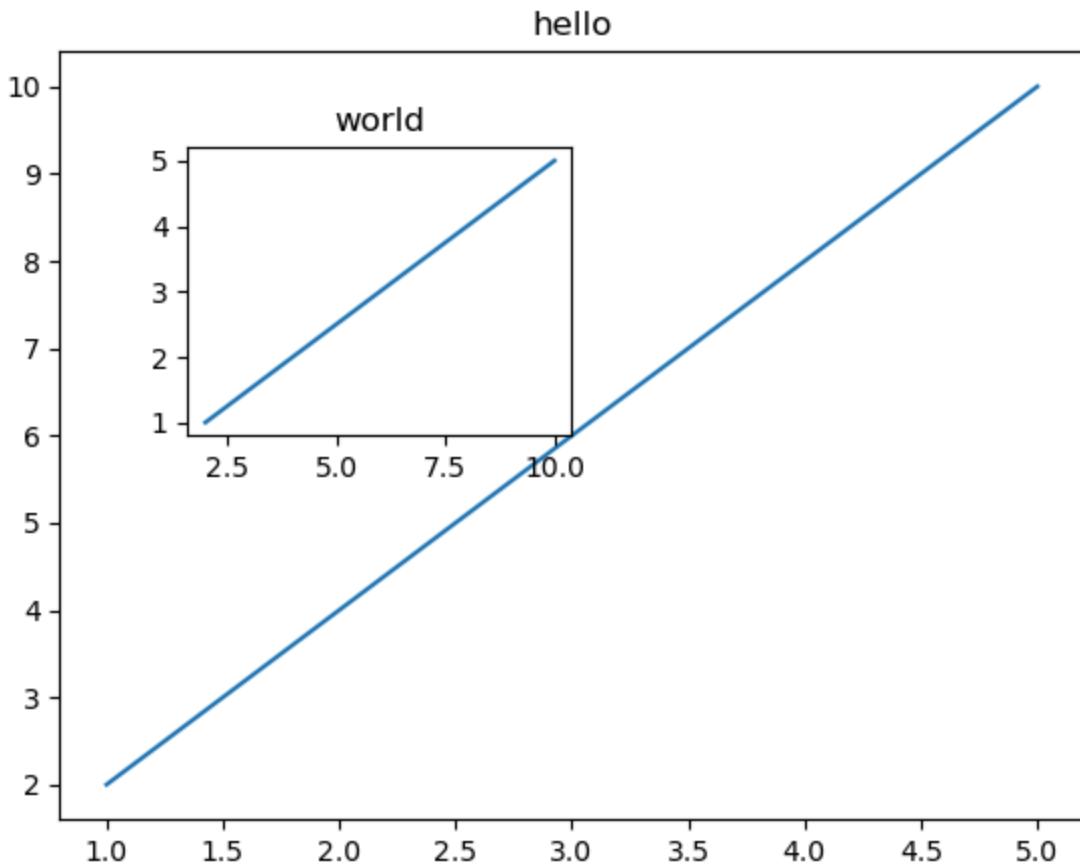
Code is a little more complicated, but the advantage is that we now have full control of where the plot axes are placed, and we can easily add more than one axis to the figure:

```

In [5]: fig = plt.figure()
axes1 = fig.add_axes([0.1,0.1,0.8,0.8])
plt.title("hello")
axes2 = fig.add_axes([0.2,0.5,0.3,0.3])
plt.title("world")
axes1.plot(x,y)
axes2.plot(y,x)

```

Out[5]: [<matplotlib.lines.Line2D at 0xlaale992e70>]



## Figure Size, Aspect Ratio & DPI in Matplotlib

- **Figure Size** → Width × Height of the canvas (in inches)
- **Aspect Ratio** → Relationship between width and height
- **DPI (Dots Per Inch)** → Resolution (number of pixels per inch)

These settings help you control how large and sharp your plots appear. [python](#)  
[Copy code](#)

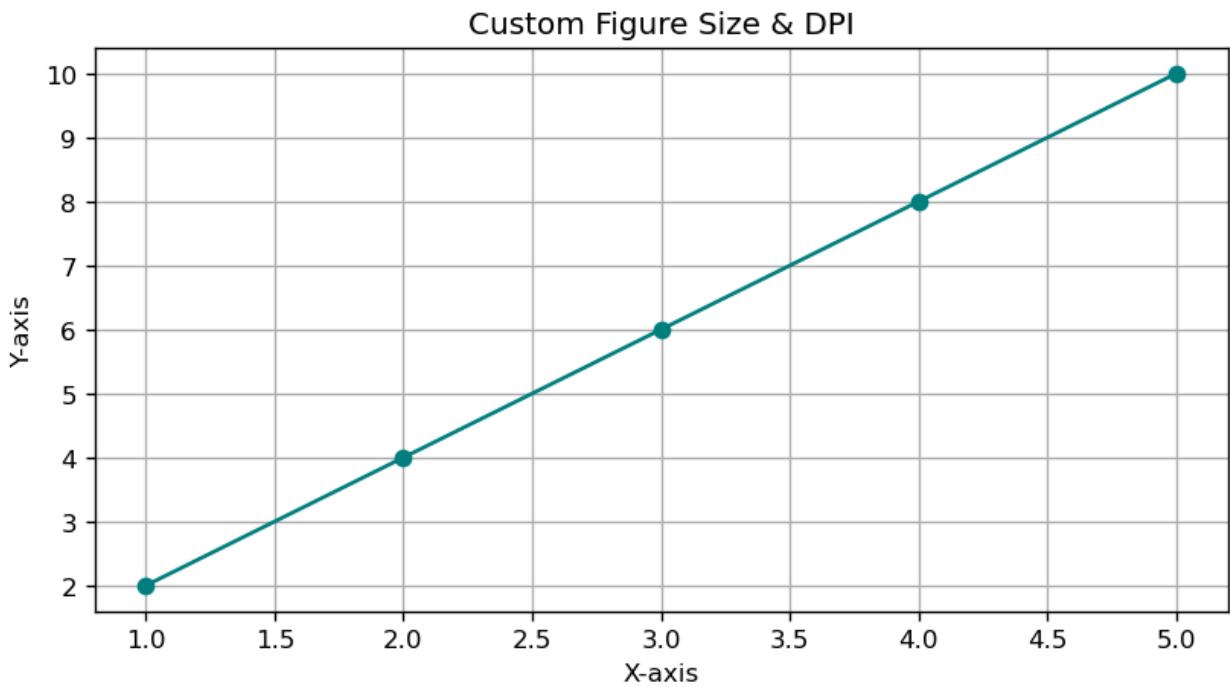
```
In [6]: import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a figure with custom size & DPI
plt.figure(figsize=(8, 4), dpi=120)    # width=8in, height=4in, 120 pixels/inch

plt.plot(x, y, color="teal", marker="o")
plt.title("Custom Figure Size & DPI")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)

plt.show()
```



## Legends, Labels & Title

Matplotlib lets you describe your plot with:

- **Title** → Explains what the plot shows
- **Axis Labels** → Describe x-axis & y-axis
- **Legend** → Identifies different lines or markers

```
In [7]: import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [1, 2, 3, 4, 5]

plt.figure(figsize=(7,4))

# Plot first line
plt.plot(x, y1, color="blue", marker="o", label="Squares")

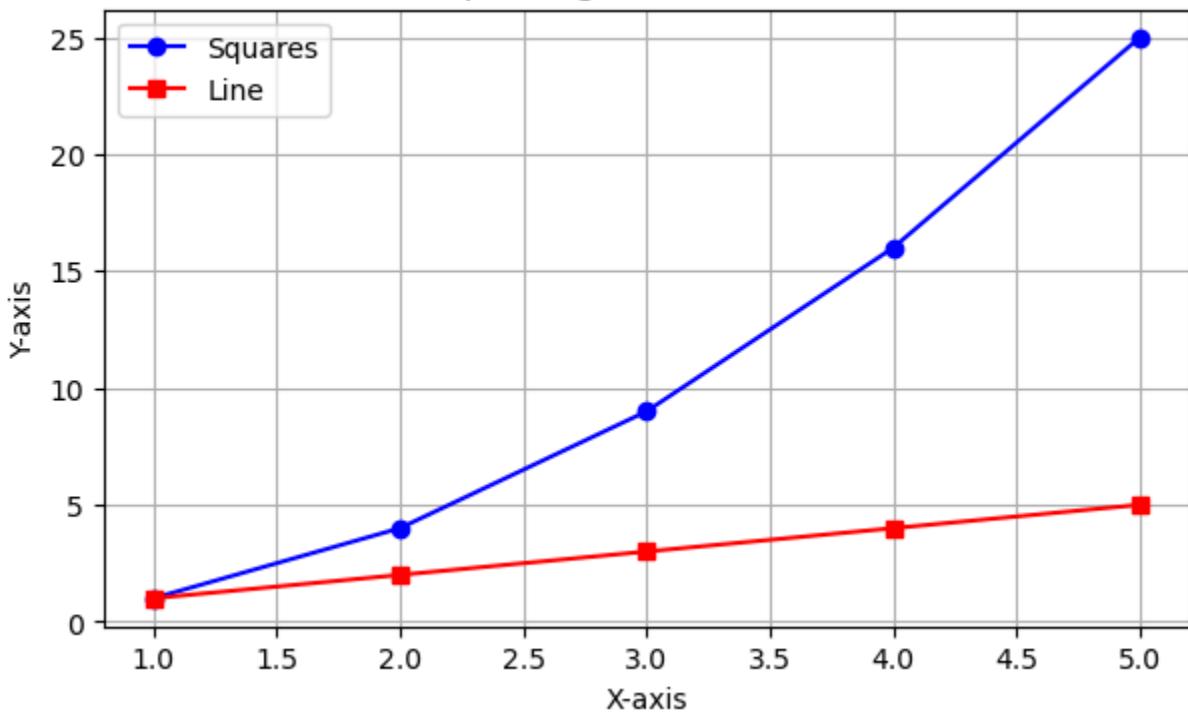
# Plot second line
plt.plot(x, y2, color="red", marker="s", label="Line")

# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Example: Legends, Labels & Title")

# Show legend
plt.legend(loc="upper left")
```

```
plt.grid(True)  
plt.show()
```

Example: Legends, Labels & Title



## Line Styles, Colors, Widths & Markers

Matplotlib lets you customize almost every visual property of a line:

- **Color** (`color=` or `c=`)
- **Line Width** (`linewidth=` or `lw=`)
- **Line Style** (`linestyle=` or `ls=`)
- **Markers** (`marker=`, `markersize=`, `markerfacecolor=`, etc.)

```
In [8]: import matplotlib.pyplot as plt  
  
# Sample data  
x = [0, 1, 2, 3, 4, 5]  
  
fig, ax = plt.subplots(figsize=(12, 6))  
  
# Different line widths (all in orange)  
ax.plot(x, [v + 1 for v in x], color="orange", linewidth=0.5, label="width=0.5")  
ax.plot(x, [v + 2 for v in x], color="orange", linewidth=1.5, label="width=1.5")  
ax.plot(x, [v + 3 for v in x], color="orange", linewidth=3, label="width=3")  
  
# Line styles with constant width  
ax.plot(x, [v + 4 for v in x], color="green", lw=2, linestyle='-' ) # solid
```

```

ax.plot(x, [v + 5 for v in x], color="green", lw=2, linestyle='--') # dashed
ax.plot(x, [v + 6 for v in x], color="green", lw=2, linestyle='-.') # dash-dot
ax.plot(x, [v + 7 for v in x], color="green", lw=2, linestyle=':') # dotted

# Custom dash pattern
line, = ax.plot(x, [v + 8 for v in x], color="black", lw=2)
line.set_dashes([10, 5, 2, 5]) # dash, space, dash, space...

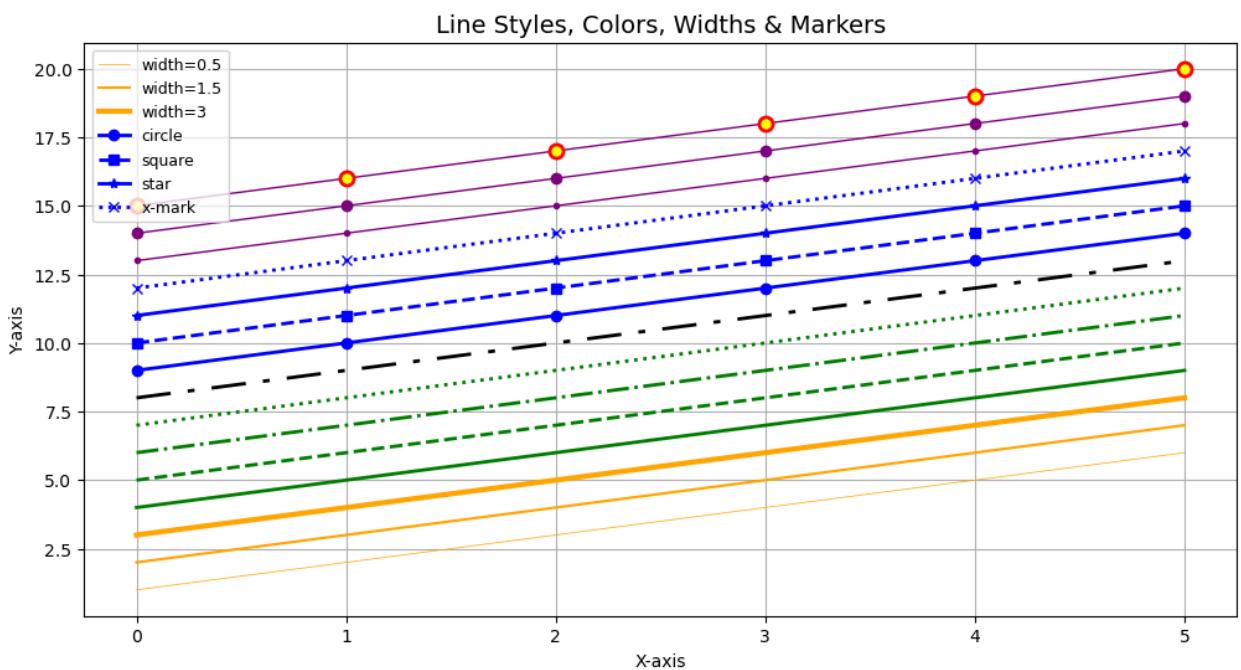
# Marker styles with a blue line
ax.plot(x, [v + 9 for v in x], color="blue", lw=2, ls='--', marker='o', label="o")
ax.plot(x, [v + 10 for v in x], color="blue", lw=2, ls='--', marker='s', label="s")
ax.plot(x, [v + 11 for v in x], color="blue", lw=2, ls='--', marker='*', label="*")
ax.plot(x, [v + 12 for v in x], color="blue", lw=2, ls='--', marker='x', label="x")

# Playing with marker size & colors
ax.plot(x, [v + 13 for v in x], color="purple", lw=1, ls='--', marker='o', markersize=6, markeredgecolor="red", markeredgewidth=1, markerfacecolor="yellow")
ax.plot(x, [v + 14 for v in x], color="purple", lw=1, ls='--', marker='o', markersize=8, markeredgecolor="red", markeredgewidth=2, markerfacecolor="yellow")
ax.plot(x, [v + 15 for v in x], color="purple", lw=1, ls='--', marker='o', markersize=10, markeredgecolor="red", markeredgewidth=3, markerfacecolor="yellow")

# Final touches
ax.set_title("Line Styles, Colors, Widths & Markers", fontsize=14)
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.grid(True)
ax.legend(loc="upper left", fontsize=9)

plt.show()

```



## Setting Plot Range (Axis Limits)

You can control how much of your data is visible by setting **x-axis** and **y-axis** limits.

Matplotlib provides:

- `plt.xlim()` / `plt.ylim()` → Quick way
- `ax.set_xlim()` / `ax.set_ylim()` → Object-Oriented way

```
In [9]: import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

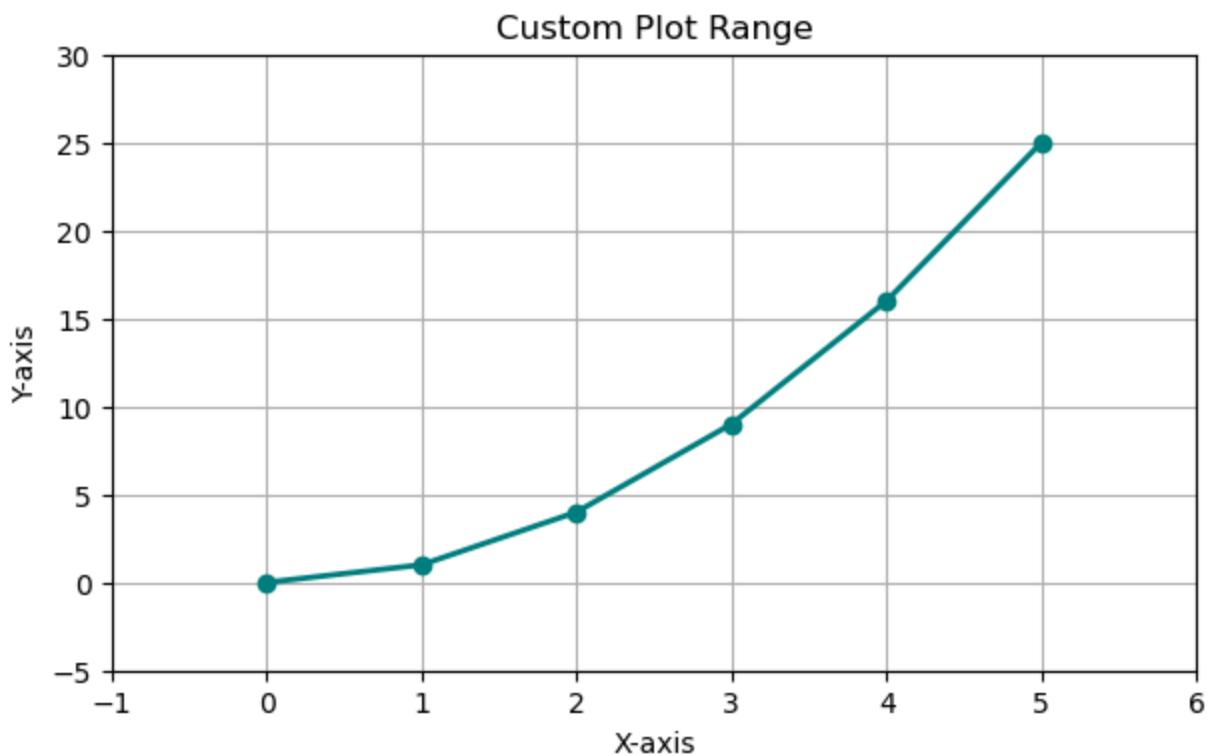
fig, ax = plt.subplots(figsize=(7, 4))

ax.plot(x, y, color="teal", marker="o", linewidth=2)

# Set custom plot range
ax.set_xlim(-1, 6)      # x-axis range
ax.set_ylim(-5, 30)      # y-axis range

ax.set_title("Custom Plot Range")
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.grid(True)

plt.show()
```



## ★ Special Plot Types: Scatter, Histogram & Boxplot

```
In [10]: import matplotlib.pyplot as plt
import numpy as np

# Data
x = [1, 2, 3, 4, 5]
y = [5, 7, 4, 6, 8]
nums = np.random.randn(100)

plt.figure(figsize=(12,4))

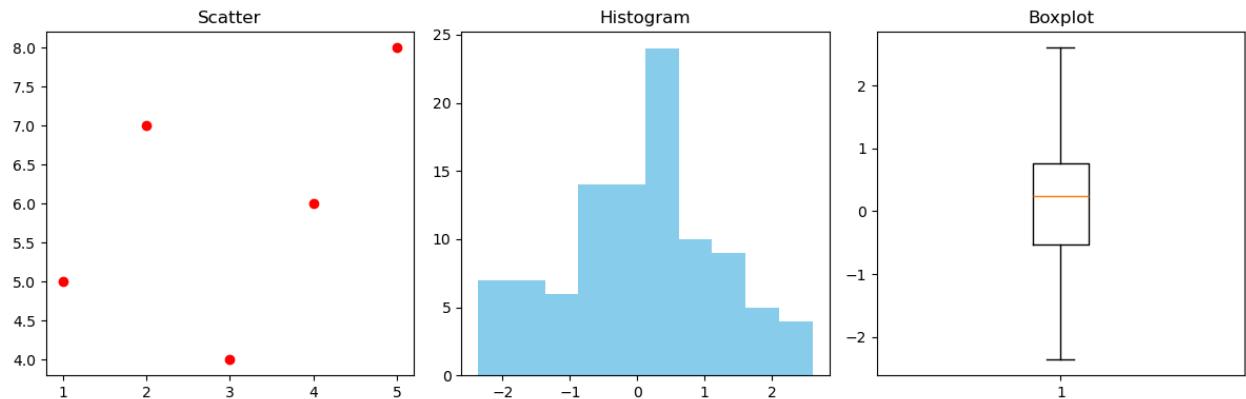
# 1 ◇ Scatter
plt.subplot(1,3,1)
plt.scatter(x, y, color="red")
plt.title("Scatter")

# 2 ◇ Histogram
plt.subplot(1,3,2)
plt.hist(nums, bins=10, color="skyblue")
plt.title("Histogram")

# 3 ◇ Boxplot
plt.subplot(1,3,3)
plt.boxplot(nums)
plt.title("Boxplot")

plt.tight_layout()
```

```
plt.show()
```



#### Note:

Scatter plots, histograms, and boxplots are available in Matplotlib — but libraries like **Seaborn** make these (and many more) much easier, prettier, and require less code.

For now, this is all you really need from Matplotlib regarding these plot types.

Let's move on and explore **Seaborn** to see how it simplifies and enhances our visualizations!

## The End!