Data Visualization is the graphical representation of data, which is essential for **Exploratory Data Analysis (EDA)**, communication, and identifying patterns. The go-to tool in the Python ecosystem is **Matplotlib**.

## Introduction to Matplotlib and Basic Plots 📊

**Matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python. Its module, pyplot (usually imported as plt), provides a MATLAB-like interface for simple plotting.

### 1. Line Plot (Trends Over Time)

Line plots are used to visualize the **relationship between two variables**, especially when one is a continuous sequence like time.

| Plot Type | Code Function | Purpose |
| --- | --- | --- |
| **Line Plot** | plt.plot(x, y) | Shows **trends** and **changes** over an interval or time series. |

### 2. Bar Plot (Categorical Comparison)

Bar plots display the count or aggregate value for different categories.

| Plot Type | Code Function | Purpose |
| --- | --- | --- |
| **Bar Plot** | plt.bar(categories, values) | Compares **discrete categories** (e.g., sales per region, count of product types). |

### 3. Histogram (Distribution of Numerical Data)

Histograms illustrate the **distribution** of a single numerical variable by dividing the data into "bins" and showing the frequency of observations in each bin.

| Plot Type | Code Function | Purpose |
| --- | --- | --- |
| **Histogram** | plt.hist(data, bins=n) | Visualizes the **frequency distribution** of a variable (e.g., age distribution). |

### 4. Scatter Plot (Relationship/Correlation)

Scatter plots show the individual data points and are used to examine the **relationship (correlation)** between two continuous numerical variables.

| Plot Type | Code Function | Purpose |
| --- | --- | --- |
| **Scatter Plot** | plt.scatter(x, y) | Identifies **correlation** and helps detect **outliers** (points far from the main cluster). |

### Python Example for Basic Plots

Python

import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
  
# Sample Data  
data = pd.DataFrame({  
 'Date': pd.to\_datetime(['2025-01-01', '2025-01-02', '2025-01-03']),  
 'Sales': [100, 150, 120],  
 'Category': ['A', 'B', 'A'],  
 'Price': [10.5, 20.2, 12.0]  
})  
  
# Line Plot Example (Time Series)  
plt.figure(figsize=(6, 4))  
plt.plot(data['Date'], data['Sales'], marker='o')  
plt.title('Daily Sales Trend')  
plt.xlabel('Date')  
plt.ylabel('Sales Revenue')  
plt.show()

## Customization and Enhancements ✨

Customizing plots is crucial for effective storytelling and clear communication of insights.

| Customization Element | Matplotlib Function | Purpose |
| --- | --- | --- |
| **Title** | plt.title('Your Plot Title') | Gives the overall context of the visualization. |
| **Labels** | plt.xlabel('X-Axis Label') plt.ylabel('Y-Axis Label') | Clearly identifies what the axes represent, including units. |
| **Legend** | plt.legend() | Explains what each color or line style represents, especially when multiple datasets are plotted. Requires the label parameter to be set in the plot function (e.g., plt.plot(..., label='Series 1')). |
| **Annotations** | plt.annotate('Text', xy=(x, y)) | Adds text or arrows to point out a specific data point, such as a max value or an outlier. |

## Subplots for Comparison 🖼️

Subplots allow you to create multiple charts within a single figure, which is essential for comparing distributions, correlations, or trends of different variables side-by-side.

### Python Example for Subplots

The plt.subplot() or, more commonly, plt.subplots() functions are used for this purpose.

Python

# Create a figure and a set of subplots (1 row, 2 columns)  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))  
  
# 1. Plot on the first axis (axes[0])  
axes[0].bar(data['Category'], data['Sales'])  
axes[0].set\_title('Sales by Category')  
axes[0].set\_ylabel('Total Sales')  
  
# 2. Plot on the second axis (axes[1])  
# Create a dummy numerical feature for the histogram  
hist\_data = np.random.normal(loc=50, scale=10, size=100)  
axes[1].hist(hist\_data, bins=10, color='skyblue')  
axes[1].set\_title('Distribution of Feature X')  
axes[1].set\_xlabel('Feature X Value')  
  
# Adjust the spacing between subplots  
plt.tight\_layout()   
plt.show()

* **plt.subplots(nrows, ncols):** Returns a tuple containing the main **Figure** (fig) and an array of **Axes** objects (axes).
* **axes[i]:** Each element in the axes array is an independent plot area where you apply plotting and customization functions (e.g., axes[0].hist(), axes[1].set\_title()).