Matplotlib Notes

Matplotlib is a Python **library** used for data visualization.
It is mainly used for creating static, animated plots. pyplot is the most commonly used module of Matplotlib – it provides simple functions to create plots.

Basic Workflow

Prepare Data → list, NumPy array, Pandas series. **Create Plot** → plt.plot(), plt.bar(), plt.scatter(), etc. **Customize** → title, labels, legend, colors, styles. **Show/Save** → plt.show() / plt.savefig("plot.png").

Example:

In [ ]:
```
import matplotlib as plt
print(plt.__version__)
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Sample Line Plot")
plt.show()
```

Common Chart Types

1. **Line Plot** - Shows trends over time.
2. **Bar Chart** - Compares quantities.
3. **Scatter Plot** - Shows relationships between variables.
4. **Histogram** - Shows frequency distribution.
5. **Pie Chart**
6. **Stacked Bar Chart**
7. **Area Plot**
8. **Box Plot (Whisker Plot)**
9. **Step Plot**

Bar Plot

What is a Bar Plot?

- A **graphical representation** of data using **rectangular bars**.
- The **height/length** of each bar represents the value of a category.
- Can be **vertical** (`plt.bar`) or **horizontal** (`plt.barh`).

Why Use a Bar Plot?

- Compare **different categories** easily.
- Simple and easy to read.
- Shows **ranking, growth, or decline**.
- Supports **grouped** and **stacked** data.

Examples

- Sales of different products
- Population of states
- Students per class

One-Liner (Interview Ready)

A bar plot uses rectangular bars to compare categorical data, making it easy to visualize differences between groups.

In [ ]:
```
# Data
categories = ['Apples', 'Bananas','Cherries','Dates']
values = [30, 25, 45, 10]

# Custom Bar Plot
plt.figure(figsize=(8,5))

plt.bar(categories, values)
plt.show()
```

In [ ]:
```
bars = plt.bar(categories,values,
        color=['red','yellow','pink','brown'],  # color
        edgecolor='b',     # border color
        linewidth=1.5,    # Border thickness
        alpha=0.8,       # Transparancy
       width=0.4)
# Add grid
plt.grid(axis='y', linestyle='--', alpha=0.7)
# Title and labels
plt.title("Fruit Sales in 2025", fontsize=16, fontweight="bold", color="navy")
plt.xlabel("Fruit", fontsize=12)
plt.ylabel("Quantity Sold", fontsize=12)
# Add values on top of bars
for bar in bars:
    plt.text(bar.get_x() + bar.get_width()/2,        # X position (middle of bar
             bar.get_height() + 1,                   # Y position (above bar)
             str(bar.get_height()),                  # Text = value
             ha='center', va='bottom', fontsize=10, fontweight="bold")
```

In [ ]:
```
#  Bar Plot with Legend & Title Position
import matplotlib.pyplot as plt
# Data
```

```
categories = ['Apples', 'Bananas', 'Cherries', 'Dates']
values1 = [30, 25, 45, 10]
values2 = [20, 15, 35, 5]

plt.figure(figsize=(8,5))

# Two sets of bars (grouped bar chart)
plt.bar(categories, values1, label="2024 Sales", color="skyblue", width=0.4, ali
plt.bar(categories, values2, label="2025 Sales", color="orange", width=-0.4, ali

# Title with position
plt.title("Fruit Sales Comparison", fontsize=16, fontweight="bold", loc="left")
# loc options: 'left', 'center', 'right'
# Axis labels
plt.xlabel("Fruit")
plt.ylabel("Quantity Sold")
# Legend position
plt.legend(title="Expense Type",loc="upper right")
# Common options: 'upper right', 'upper left', 'lower right', 'lower left', 'bes
plt.grid(axis='y', linestyle="--", alpha=0.7)
plt.show()
```

Scatter Plot

Example 1: Basic Scatter Plot

In [ ]:
```
male_height = [160, 170, 180, 175]
male_weight = [55, 70, 80, 75]
female_height = [150, 165, 172, 168]
female_weight = [48, 58, 65, 60]

plt.scatter(male_height, male_weight, color="blue", marker="o", s=90, label="Mal
plt.scatter(female_height, female_weight, color="magenta", marker="^", s=90, lab
plt.title("Height vs Weight Distribution")
plt.xlabel("Height (cm)")
plt.ylabel("Weight (kg)")
plt.legend(title="Gender")
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```

Key Customizations

- **color** : Set point color (`"red"` , `"blue"` , `"green"` , etc.)
- **marker** : Shape of points (`'o'` =circle, `'^'` =triangle, `'s'` =square, `'x'` =cross, `'*'` =star, etc.)
- **s** : Size of points (larger value = bigger points)
- **c** : Apply **color mapping** (based on another variable → gradient effect)
- **alpha** : Transparency (0 = fully transparent, 1 = opaque)
- **plt.legend(title="...")** : Adds legend with a **custom title**
- **plt.colorbar()** : Adds color reference scale (when using `c` with colormap)

Example Code

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

# Data
x = np.random.rand(50)
y = np.random.rand(50)
sizes = np.random.randint(50, 300, size=50)  # point sizes
colors = np.random.rand(50)  # color values

plt.scatter(x, y,
            s=sizes,         # size of points
            c=colors,        # color mapping
            cmap="viridis",  # colormap
            alpha=0.7,
            marker="o",
            edgecolors="black",
            label="Data Points")

plt.title("Customized Scatter Plot", fontsize=14, fontweight="bold")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

plt.legend(title="Legend Example")
plt.colorbar(label="Color Intensity")  # color scale

plt.show()
```

5. Multiple Plots

```
x = np.linspace(0,10,100)
fig, axs = plt.subplots(2, 2, figsize=(8,6))

axs[0,0].plot(x, np.sin(x))
axs[0,1].plot(x, np.cos(x))
axs[1,0].plot(x, np.tan(x))
axs[1,1].hist(np.random.randn(100))
plt.tight_layout()
```

Bar Plot Example with Legend & Title Position

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

# Data
months = ["Jan", "Feb", "Mar", "Apr", "May"]
food_exp = [2500, 2700, 2600, 2800, 3000]
travel_exp = [1200, 1100, 1000, 1300, 1400]

plt.figure(figsize=(9,6))

# Plotting grouped bars
plt.bar(months, food_exp, label="Food", color="teal", width=0.4, align="edge")
plt.bar(months, travel_exp, label="Travel", color="salmon", width=-0.4, align="e
```

```
# Title (custom position)
plt.title("Monthly Expenses - 2025", fontsize=16, fontweight="bold", loc="right"

# Labels
plt.xlabel("Month", fontsize=12)
plt.ylabel("Expenses (INR)", fontsize=12)

# Legend (custom position)
plt.legend(loc="upper left", fontsize=11)

# Grid
plt.grid(axis="y", linestyle="--", alpha=0.6)

plt.show()
```

Histogram Plot

What is a Histogram?

- A **graphical representation** of the **distribution of numerical data**.
- Shows how data is divided into **intervals (bins)** and the **frequency** of values in each bin.
- Useful for **data analysis, probability distribution, and pattern detection**.

Key Parameters

- **bins** : Number of intervals (e.g., `bins=10`)
- **color** : Fill color of bars
- **edgecolor** : Border color of bars
- **alpha** : Transparency (0 → invisible, 1 → opaque)
- **density=True** : Normalize histogram (probability distribution instead of counts)
- **histtype** : Style of histogram (`'bar'`, `'barstacked'`, `'step'`, `'stepfilled'`)
- **cumulative=True** : Cumulative frequency

Example 1: Basic Histogram

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

data = np.random.randn(1000)  # Normal distribution

plt.hist(data, bins=20, color="skyblue", edgecolor="black")
plt.title("Basic Histogram")
plt.xlabel("Value Ranges")
plt.ylabel("Frequency")
plt.show()
```

Why Use Histogram?

- To **understand distribution** of data (how values spread across ranges).
- Helps in detecting:
  - **Skewness** (left/right shift of data)
  - **Spread** (how wide values are)
  - **Outliers** (extreme values)
- Widely used in **EDA (Exploratory Data Analysis)**, statistics, and **Machine Learning preprocessing**.

Real-Life Example: Student Exam Scores

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

# Example: Scores of 100 students
np.random.seed(42)
scores = np.random.randint(35, 100, 100)  # random marks between 35 and 100

plt.hist(scores, bins=10, color="teal", edgecolor="black", alpha=0.7)

plt.title("Distribution of Student Exam Scores", fontsize=14, fontweight="bold")
plt.xlabel("Score Ranges")
plt.ylabel("Number of Students")
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.show()
```

Real-Life Example: Employee Salaries

In [ ]:
```
salaries = np.random.randint(20000, 100000, 200)  # salaries of 200 employees
plt.hist(salaries, bins=15, color="orange", edgecolor="black", alpha=0.8)
plt.title("Distribution of Employee Salaries", fontsize=14, fontweight="bold")
plt.xlabel("Salary Range")
plt.ylabel("Number of Employees")
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.show()
```

Shows the spread of salaries across ranges.

HR can identify:

Most common salary brackets

Presence of very high or very low earners

Useful for policy decisions, increments, and budgeting

Pie Chart

What is a Pie Chart?

- A circular chart divided into slices.
- Each slice represents a **proportion of the whole**.
- Best for **percentage or part-to-whole comparisons**.

Why Use a Pie Chart?

- To **visualize proportions** of categories.
- Easy to understand with **few categories (≤ 6)**.
- Not suitable for large categories (use bar chart instead).

Syntax

```
plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=90)
```

Example : Sales Distribution

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

# Data
products = ['Electronics', 'Clothing', 'Groceries', 'Books']
sales = [45000, 30000, 15000, 10000]

# Plot
plt.figure(figsize=(6,6))
plt.pie(sales,
        labels=products,
        autopct='%1.1f%%',
        startangle=90,
        colors=['skyblue','lightgreen','orange','pink'],
        explode=[0.1,0,0,0],  # highlight Electronics
        shadow=True)

plt.title("Sales Distribution by Category")
plt.legend(title="Product Categories")
plt.show()
```

Key Customizations in Pie Chart

- **labels** → category names
- **autopct** → show percentages on slices (e.g., `'%1.1f%%'`)
- **colors** → custom colors for slices
- **explode** → separate a slice for emphasis (e.g., `[0.1, 0, 0, 0]`)
- **shadow=True** → add shadow for 3D effect
- **startangle** → rotate chart (e.g., `90` starts from top)
- **plt.legend(title="...")** → add legend with a title

Subplots in Matplotlib

Subplots allow you to create **multiple plots in a single figure**, useful for comparing datasets side by side.

Methods to Create Subplots

1. `plt.subplot(rows, cols, index)`

- Creates a single subplot inside a grid.
- `rows` : total rows of subplots
- `cols` : total columns of subplots
- `index` : position of current plot (starts from 1)

---

2. `plt.subplots(rows, cols)`

- Returns a **figure (`fig`)** and **axes (`ax`)** object.
- More flexible (you can loop through axes).
- Allows easy customization.

Key Customizations

- `figsize=(w, h)` → control figure size
- `sharex` , `sharey` → share axis across subplots
- `tight_layout()` → auto adjust spacing
- `fig.suptitle("...")` → overall title

Example 1: Using `plt.subplot`

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

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

plt.figure(figsize=(8,4))

plt.subplot(1,2,1)   # 1 row, 2 columns, 1st plot
plt.plot(x, y1, color="blue")
plt.title("Square Numbers")

plt.subplot(1,2,2)   # 1 row, 2 columns, 2nd plot
plt.plot(x, y2, color="red")
plt.title("Reverse Numbers")
```

```
plt.suptitle("Comparison of Plots", fontsize=14, fontweight="bold")
plt.tight_layout()
plt.show()
```

6. Styling Options

- **Line Style:** `'-'`, `'--'`, `':'`, `'-.'`
- **Markers:** `'o'`, `'*'`, `'s'`, `'d'`, `'+'`
- **Colors:** `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, `'k'`
- **Grid:** `plt.grid(True)`
- **Transparency:** `alpha=0.5`

7. Annotations & Text

```
plt.plot([1,2,3,4],[10,20,25,30])
plt.annotate("Peak", xy=(3,25), xytext=(2,27),
             arrowprops=dict(facecolor='black', shrink=0.05))
```

8. Save Plots

```
plt.savefig("figure.png", dpi=300, bbox_inches="tight")
```

---

9. Useful Functions

- `plt.figure(figsize=(w,h))` → change figure size
- `plt.xlim(), plt.ylim()` → set axis limits
- `plt.xticks(), plt.yticks()` → set ticks
- `plt.style.use("ggplot")` → use predefined style

---

10. Common Interview Qs

1. **Difference between `figure` and `axes`?**
   - *Figure = entire canvas, Axes = one plot inside it.*
2. **How to plot multiple plots in one figure?**
   - Use `plt.subplot()` or `plt.subplots()`.
3. **What is difference between `plt.plot()` and `ax.plot()`?**
   - `plt.plot()` → state-based (quick plotting).
   - `ax.plot()` → object-oriented (preferred for multiple plots).

In [ ]: