

# Basic Plot Graph (Line Graph)

A **line plot** shows the relationship between two variables using a continuous line. It is mainly used to show **trends**, **changes over time**, or **comparisons**.

## Simple Line Plot Example

```
import matplotlib.pyplot as plt

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

plt.plot(x, y)
plt.show()
```

## Most Used Parameters of plt.plot()

Parameter	Meaning	Example
x, y	Data points	plt.plot(x, y)
color	Line color	color='red'
linestyle	Line style	'--', '-.', ':'
linewidth	Thickness of line	linewidth=2
marker	Point marker	'o', '*', 's'
markersize	Marker size	markersize=8
label	Line name	label='Sales'

## Line Plot with Parameters

```
plt.plot(x, y,
         color='blue',
         linestyle='--',
         linewidth=2,
         marker='o',
         markersize=7,
         label='Growth')

plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("Simple Line Plot")
plt.legend()
plt.show()
```

# Bar Graph in Matplotlib

A **bar graph** represents **categorical data** using rectangular bars. It is used to **compare values across categories**.

## Simple Bar Graph Example

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

```

subjects = ['Math', 'Science', 'English']
marks = [80, 70, 85]

plt.bar(subjects, marks)
plt.show()

```

## Most Used Parameters of `plt.bar()`

Parameter	Meaning	Example
x	Categories	subjects
height	Values	marks
color	Bar color	color='green'
width	Bar width	width=0.5
label	Bar label	label='Marks'
align	Alignment	'center'
edgecolor	Border color	edgecolor='black'

## Bar Graph with Parameters

```

plt.bar(subjects, marks,
        color='orange',
        width=0.6,
        edgecolor='black',
        label='Student Marks')

plt.xlabel("Subjects")
plt.ylabel("Marks")
plt.title("Marks Bar Graph")
plt.legend()
plt.show()

```

## Two Bar Graphs in Same Plot (Different Colors)

### Example: Comparison of Two Students

```

import matplotlib.pyplot as plt
import numpy as np

subjects = ['Math', 'Science', 'English']
student1 = [80, 70, 85]
student2 = [75, 65, 90]

x = np.arange(len(subjects))
width = 0.35

plt.bar(x - width/2, student1,
        width=width,
        color='blue',
        label='Student 1')

plt.bar(x + width/2, student2,
        width=width,

```

```

        color='red',
        label='Student 2')

plt.xlabel("Subjects")
plt.ylabel("Marks")
plt.title("Marks Comparison")
plt.xticks(x, subjects)
plt.legend()
plt.show()

```

## Key Difference: Line Plot vs Bar Graph

Feature	Line Plot	Bar Graph
Data Type	Continuous	Categorical
Best For	Trends over time	Comparison
Shape	Line	Rectangular bars

## SCATTER PLOT

A **scatter plot** is used to show the **relationship between two numerical variables**. Each point represents one observation with an **X value** and a **Y value**.

### Used to:

- Find correlation (positive / negative / none)
- Observe trends
- Detect outliers

### When to Use

- Both variables must be **numeric**
- You want to see **how one variable affects another**

### Example

Study Hours (X) vs Marks (Y)

### Scatter Plot Code (Matplotlib)

```

import matplotlib.pyplot as plt

# Data
hours = [1, 2, 3, 4, 5, 6]
marks = [30, 40, 50, 60, 70, 85]

# Scatter plot
plt.scatter(hours, marks)

plt.xlabel("Study Hours")
plt.ylabel("Marks")
plt.title("Scatter Plot: Study Hours vs Marks")
plt.grid(True)

```

```
plt.show()
```

## Key Parameters (Remember)

- `x, y` → Data values
- `marker` → Shape of point
- `s` → Size of points
- `alpha` → Transparency
- `grid()` → Readability

## One-Line Answer Note:

Scatter plot shows the relationship between two numerical variables.

# HISTOGRAM

A **histogram** shows the **frequency distribution of one numerical variable** by grouping data into **bins**.

## Used to:

- Understand data distribution
- Check spread and skewness
- Identify outliers

## When to Use

- Only **one numeric variable**
- You want to see **how values are distributed**

## Example

Marks distribution of students

## Histogram Code (Matplotlib)

```
import matplotlib.pyplot as plt

# Data
marks = [30, 35, 40, 45, 50, 55, 60, 70, 80, 90]

# Histogram
plt.hist(marks, bins=5)

plt.xlabel("Marks Range")
plt.ylabel("Frequency")
plt.title("Histogram: Marks Distribution")
plt.grid(True)

plt.show()
```

---

## Key Parameters (Remember)

- `data` → Numeric values
- `bins` → Number of groups
- `frequency` → Count per bin
- `edgecolor` → Border
- `density` → Probability distribution

## What is a Pie Chart

A **Pie Chart** is a **circular chart** that represents **parts of a whole**.  
The full circle equals **100%**, and each slice shows a category's share.

### When to Use

- To show **percentages / proportions**
- When the number of categories is **small**
- To compare contribution of each category

### Basic Example

```
import matplotlib.pyplot as plt

sizes = [40, 30, 20, 10]
labels = ['A', 'B', 'C', 'D']

plt.pie(sizes, labels=labels)
plt.show()
```

### Most Used Pie Chart Parameters

Parameter	Use
<code>labels</code>	Names of slices
<code>autopct</code>	Show percentage values
<code>startangle</code>	Rotate the chart
<code>explode</code>	Separate a slice
<code>colors</code>	Set slice colors
<code>shadow</code>	Add shadow
<code>radius</code>	Size of the pie
<code>counterclock</code>	Direction of drawing
<code>wedgeprops</code>	Border style

### Example with Parameters

```
explode = (0.1, 0, 0, 0)

plt.pie(
    sizes,
```

```
    labels=labels,
    autopct='%1.1f%%',
    startangle=90,
    explode=explode,
    shadow=True
)
plt.show()
```

## SINGLE-POINT PIE CHART

A **Single-Point Pie Chart** shows **one main value compared with the remaining part** of the whole.

It usually has **two slices**:

- Selected value
- Remaining value (Others)

### When to Use

- To highlight **one key metric**
- To show **completion, progress, or contribution**
- Dashboard indicators

### Example

```
import matplotlib.pyplot as plt

plt.pie([1])
plt.show()
```

## DONUT CHART

A **Donut Chart** is a **Pie Chart with a hole in the center**.  
It represents proportions but looks **more modern** and clean.

Data meaning is same as a Pie Chart.

### When to Use

- Dashboard visualization
- Modern UI design
- Showing percentages with focus on center text

### Basic Donut Chart Example

```
plt.pie(
    sizes,
    labels=labels,
    autopct='%1.1f%%',
    wedgeprops={'width': 0.4}
)
```

```
plt.show()
```

Most Used Donut Chart Parameters

Parameter	Use
wedgeprops={'width':0.6}	Controls hole size
labels	Slice names
autopct	Show percentages
colors	Slice colors
startangle	Rotation
explode	Highlight slice

Comparison Table

Chart Type	Purpose
Pie Chart	Show parts of a whole
Single-Point Pie	Highlight one value
Donut Chart	Modern percentage display

Advantages

- ✓ Easy to understand
- ✓ Good for percentage data
- ✓ Visually attractive

Limitations

- ✓ Not suitable for large datasets
- ✓ Hard to compare similar values

What is a Stem Plot

A **Stem Plot** is a type of plot that shows **discrete data points** using:

- a **vertical (or horizontal) line** called a *stem*
- a **marker (dot)** at the end of each stem

Each stem starts from a **baseline (usually zero)** and ends at the data value.

It looks like **lollipops**

When to Use a Stem Plot?

- For **discrete data**
- When values are **independent**

- To clearly show **individual data points**
- As an alternative to **bar charts**

## Real-Life Example

- Number of students present each day
- Signal processing (amplitude vs time)
- Frequency of events

## Simple Example

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 30, 25]

plt.stem(x, y)
plt.show()
```

## Most Used Stem Plot Parameters

### linefmt

Controls **stem line style**.

```
plt.stem(x, y, linefmt='--')
```

Examples:

- '-' solid
- '--' dashed
- ':' dotted

### markerfmt

Controls **marker style**.

```
plt.stem(x, y, markerfmt='o')
```

Examples:

- 'o' circle
- '^' triangle
- 's' square

### basefmt

Controls **baseline style**.

```
plt.stem(x, y, basefmt='r-')
```

### **use\_line\_collection**

Improves performance (recommended).

```
plt.stem(x, y, use_line_collection=True)
```

### **orientation**

Changes plot direction.

```
plt.stem(x, y, orientation='horizontal')
```

### **label**

Adds legend label.

```
plt.stem(x, y, label='Data')  
plt.legend()
```

### **Complete Example with Parameters**

```
import matplotlib.pyplot as plt  
  
x = [1, 2, 3, 4, 5]  
y = [10, 20, 15, 30, 25]  
  
plt.stem(  
    x, y,  
    linefmt='--',  
    markerfmt='o',  
    basefmt='k-',  
)  
  
plt.xlabel("X Values")  
plt.ylabel("Y Values")  
plt.title("Stem Plot Example")  
plt.show()
```

### **Advantages**

- ✓ Best for **discrete data**
- ✓ Shows exact data points
- ✓ Cleaner than bar chart

### **Limitations**

- ✓ Not suitable for continuous data
- ✓ Cluttered with large datasets

## **Box-and-Whisker Plot**

A **Box Plot** is a statistical chart used to **summarize the distribution** of numerical data using **five key values**:

1. Minimum
2. First Quartile (Q1)
3. Median (Q2)
4. Third Quartile (Q3)
5. Maximum

It also helps in identifying **outliers**.

## Five-Number Summary

Minimum | Q1 | Median | Q3 | Maximum

- **Q1** → 25% data below
- **Median** → 50% data
- **Q3** → 75% data below

## Components of a Box Plot

- **Box** → From Q1 to Q3 (Interquartile Range – IQR)
- **Median Line** → Inside the box
- **Whiskers** → Extend to min & max (excluding outliers)
- **Outliers** → Points beyond whiskers

## Why Use a Box Plot?

- ✓ Shows data spread
- ✓ Detects outliers
- ✓ Compares multiple datasets
- ✓ Compact statistical summary

## When to Use?

- Distribution analysis
- Comparing groups
- Identifying skewness
- Outlier detection

## Simple Example

```
import matplotlib.pyplot as plt

data = [10, 20, 25, 30, 35, 40, 100]

plt.boxplot(data)
plt.show()
```

## Most Used Box Plot Parameters

**vert**

Controls orientation.

```
plt.boxplot(data, vert=False)
```

True → Vertical (default)

False → Horizontal

### **patch\_artist**

Fills box with color.

```
plt.boxplot(data, patch_artist=True)
```

### **showmeans**

Displays mean value.

```
plt.boxplot(data, showmeans=True)
```

### **meanline**

Draws mean as a line.

```
plt.boxplot(data, showmeans=True, meanline=True)
```

### **showfliers**

Controls outlier display.

```
plt.boxplot(data, showfliers=False)
```

### **labels**

Adds category labels.

```
plt.boxplot([data1, data2], labels=['Group A', 'Group B'])
```

### **whis**

Controls whisker length.

```
plt.boxplot(data, whis=1.5)
```

Default =  $1.5 \times \text{IQR}$

### **widths**

Controls box width.

```
plt.boxplot(data, widths=0.5)
```

## **Styling Parameters (Very Common)**

```
plt.boxplot(  
    data,  
    patch_artist=True,  
    boxprops=dict(facecolor='lightblue'),  
    medianprops=dict(color='red'),  
    whiskerprops=dict(color='black'),  
    capprops=dict(color='black')  
)
```

## Complete Example

```
import matplotlib.pyplot as plt  
  
data = [10, 20, 25, 30, 35, 40, 100]  
  
plt.boxplot(  
    data,  
    patch_artist=True,  
    showmeans=True,  
    boxprops=dict(facecolor='lightgreen'),  
    medianprops=dict(color='red')  
)  
  
plt.title("Box Plot Example")  
plt.ylabel("Values")  
plt.show()
```

## Advantages

- Shows spread & central tendency
- Identifies outliers
- Easy comparison

## Limitations

- No detailed distribution shape
- Exact values not visible