MATPLOTLIB

plt.plot()

- x, y → values to plot on x-axis and y-axis.
- color or c → line color, e.g. "red", "blue", "g", or hex codes like "#FF5733".
- linestyle or ls → style of line. Options:
  - "-" = solid (default)
  - "--" = dashed
  - "-." = dash-dot
  - ":" = dotted
  - "" (empty) = no line
- linewidth or lw → thickness of line (e.g. 2).
- marker → symbol at each data point. Examples: "o" circle, "s" square, "^" triangle, "x", "*" star, "D" diamond.
- markersize or ms → size of marker (e.g. 8).
- markerfacecolor or mfc → fill color inside marker.
- markeredgecolor or mec → outline color of marker.
- markeredgewidth or mew → thickness of marker outline.
- label → name for the line (used when calling plt.legend()).

Labels and title

- plt.xlabel("text") → label for x-axis.
- plt.ylabel("text") → label for y-axis.
- plt.title("text") → plot title.

Axis limits and scaling

- plt.xlim(xmin, xmax) → set range of x-axis.
- plt.ylim(ymin, ymax) → set range of y-axis.
- plt.axis([xmin, xmax, ymin, ymax]) → set both x and y in one go.
- plt.axis("equal") → equal aspect ratio (circles look circular).
- plt.axis("tight") → fit axes tightly around data.
- plt.axis("off") → hide the axes completely.

Grid, legends, and text

- plt.grid(True) → show gridlines. You can style them, e.g. plt.grid(True, ls="--", lw=0.5).
- plt.legend() → show legend (works if you gave lines a label). You can choose location: plt.legend(loc="upper left").

- plt.text(x, y, "text") → place text at coordinates.
- plt.annotate("label", xy=(x, y)) → add annotation with arrow.

Ticks and rotation

- plt.xticks([...], rotation=deg) → customize tick marks on x-axis.
- plt.yticks([...]) → customize tick marks on y-axis.

Figure management

- plt.figure(figsize=(w,h)) → create a new figure of given size (width, height in inches).
- plt.subplot(r,c,i) → create subplots (r = rows, c = columns, i = index).
- plt.tight_layout() → auto adjust spacing between subplots.

Saving and displaying

- plt.savefig("filename.png", dpi=300) → save figure to file with resolution.
- plt.show() → display the plot (must be used at the end in scripts).

SCATTER PLOT

```
plt.scatter(x, y,
        s=sizes,              # marker size
        c=colors,             # marker colors
        cmap="viridis",       # colormap for colors
        alpha=0.7,            # transparency
        marker="o",           # shape of marker
        edgecolors="black",   # outline color
        linewidths=1,         # outline thickness
        label="Data points")  # label for legend

# Plot styling
```

```
plt.xlabel("X values")              # x-axis label
plt.ylabel("Y values")              # y-axis label
plt.title("Scatter Plot Distribution Example")  # plot title
plt.xlim(-4, 4)                     # x-axis limits
plt.ylim(-4, 4)                     # y-axis limits
plt.grid(True, ls="--", lw=0.5)     # grid with style
plt.legend(loc="upper right")       # show legend
plt.colorbar(label="Color intensity")  # add color scale
plt.tight_layout()                  # adjust layout

# Save and show
plt.savefig("scatter_distribution.png", dpi=300)
plt.show()
```

plt.scatter()

- Only plots **individual points**, no connecting line.
- Allows **variable marker size (s)** and **color mapping (c + cmap)** → great for multivariate visualization.
- Best for showing **relationships** (correlation, clusters, distributions).

BAR PLOT

Core Parameters

- x (or y for horizontal) → categories or positions of bars.
- height (or width for horizontal) → values/lengths of bars.

Example:

```
categories = ['Pizza','Burger','Pasta','Sushi']
values = [120, 80, 90, 60]
```

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

◇ Style Parameters

- color → fill color of bars ('blue', '#FF5733', etc.).
- edgecolor → outline color of bars.
- linewidth → thickness of bar edges.
- alpha → transparency (0 = invisible, 1 = solid).
- width → thickness of bars (default 0.8).

Example:

```
plt.bar(categories, values, color='skyblue', edgecolor='black', linewidth=1, alpha=0.8, width=0.6)
```

◇ Labels & Title

- plt.xlabel("text") → x-axis label.
- plt.ylabel("text") → y-axis label.
- plt.title("text") → chart title.

◇ Bar Labels

- plt.text(x, y, "label") → add text above each bar.
- plt.bar_label(container) (newer Matplotlib) → auto add labels.

Example:

```
bars = plt.bar(categories, values)
plt.bar_label(bars)
```

◇ Grouped / Multiple Bars

- Offset the bars with np.arange + width.

Example:

```
import numpy as np
categories = ['Pizza','Burger','Pasta']
values1 = [120, 80, 90]
values2 = [100, 60, 70]

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

plt.bar(x, values1, width, label='2023')
plt.bar(x+width, values2, width, label='2024')
```

```
plt.xticks(x+width/2, categories)
plt.legend()
```

◇ Horizontal Bars

```
plt.barh(categories, values, color='orange')
```

◇ Error Bars

Add error margins:

```
errors = [5, 8, 6, 4]
plt.bar(categories, values, yerr=errors, capsize=5, color='lightgreen')
```

◇ Grid & Axis Control

- plt.grid(True, axis='y') → grid lines.
- plt.ylim(ymin, ymax) → control vertical axis range.
- plt.xlim(xmin, xmax) → control horizontal axis range.

◇ Saving & Displaying

- plt.savefig("barchart.png", dpi=300) → save figure.
- plt.show() → show chart.

Example Full Bar Chart

```
import matplotlib.pyplot as plt

categories = ['Pizza','Burger','Pasta','Sushi']
values = [120, 80, 90, 60]
errors = [5, 8, 6, 4]

plt.figure(figsize=(8,6))
bars = plt.bar(categories, values,
        color='skyblue', edgecolor='black', linewidth=1,
        alpha=0.9, width=0.6, yerr=errors, capsize=5, label="Orders")

plt.xlabel("Food Items")
plt.ylabel("Number of Orders")
plt.title("Bar Chart Example: Food Orders")
plt.grid(True, axis='y', linestyle='--', alpha=0.6)
plt.bar_label(bars, padding=3)
plt.legend()
plt.tight_layout()
plt.show()
```

When to Use a Bar Chart

- **Comparisons**: Comparing discrete categories (restaurants, cuisines, weekdays).
- **Counts**: Frequency of ratings, number of orders.
- **Trends across categories**: Average delivery time by cuisine type.

HISTOGRAM

When to Use a Histogram

- To visualize the **distribution** of a single numeric variable.
- To check for **skewness** (normal, right-skewed, left-skewed).
- To detect **outliers** (extremely high or low values).
- To compare **two groups' distributions** (e.g., weekday vs weekend delivery times).
- To prepare for statistical/ML modeling (knowing if data is normal or skewed).

◇ Key Parameters in plt.hist()

- **x** → the numeric data to plot.
- plt.hist(df['delivery_time'])
- **bins** → number of intervals (or exact bin edges).
  - bins=10 → 10 equal ranges.
  - bins=[0,10,20,30,40,50] → custom ranges.
- **color** → fill color of bars.
- plt.hist(df['delivery_time'], bins=20, color='skyblue')
- **edgecolor** → outline color for bars (useful for clarity).
- plt.hist(df['delivery_time'], bins=20, color='orange', edgecolor='black')
- **alpha** → transparency (0 = invisible, 1 = solid).
- plt.hist(df['delivery_time'], bins=20, alpha=0.6)
- **rwidth** → relative bar width (default = 1). Smaller = gaps between bars.
- plt.hist(df['delivery_time'], bins=20, rwidth=0.8)
- **density** → if True, scales heights to form a probability density (area under histogram = 1).
- plt.hist(df['delivery_time'], bins=20, density=True)
- **histtype** → style of bars. Options:
  - "bar" (default, filled rectangles)
  - "step" (outline only)
  - "stepfilled" (outlined + filled)
  - "barstacked" (for multiple sets stacked).
- **label** → add label for legend (useful if plotting multiple histograms).

- plt.hist(df['delivery_time'], bins=20, label='Delivery Time')
- plt.legend()

◇ Example with All Parameters

```
plt.figure(figsize=(8,6))
plt.hist(df['delivery_time'],
        bins=20,
        color='skyblue',
        edgecolor='black',
        alpha=0.8,
        rwidth=0.9,
        density=False,
        histtype='bar',
        label='Delivery Time')

plt.xlabel("Delivery Time (minutes)")
plt.ylabel("Number of Orders")
plt.title("Histogram of Delivery Times")
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```

Summary

- **Use histograms** when analyzing **frequency distributions** of numeric data.
- **Best parameters to tweak**: bins, color, edgecolor, alpha, rwidth, density.
- **Multiple groups** can be shown in the same histogram using different colors and alpha.

PIE CHART

Use Case:

- Show proportions/percentages of categories in a whole.
- Good for small category sets (e.g., cuisines, weekday vs weekend orders).

Key Parameters:

- x → values (counts).
- labels → category names.
- autopct → show % inside slices (e.g., "%.1f%%").

- colors → custom slice colors.
- explode → pull slices outward.
- startangle → rotate chart.
- shadow → add shadow.

Matplotlib Styles You Can Use with plt.style.use

- seaborn-v0_8
- seaborn-v0_8-bright
- seaborn-v0_8-colorblind
- seaborn-v0_8-dark
- seaborn-v0_8-dark-palette
- seaborn-v0_8-darkgrid
- seaborn-v0_8-deep
- seaborn-v0_8-muted
- seaborn-v0_8-notebook
- seaborn-v0_8-paper
- seaborn-v0_8-pastel
- seaborn-v0_8-poster
- seaborn-v0_8-talk
- seaborn-v0_8-ticks
- seaborn-v0_8-white
- seaborn-v0_8-whitegrid
- seaborn (alias for seaborn-v0_8)
- Solarize_Light2
- bmh
- classic
- dark_background
- fast
- fivethirtyeight
- ggplot
- grayscale
- tableau-colorblind10

Advanced Matplotlib Plots

---

1. Box Plot

```
plt.boxplot(df['delivery_time'])
```

Use for spread, median, quartiles, and outliers.

---

2. Violin Plot

```
plt.violinplot(df['cost_of_the_order'])
```

Use when you want **distribution shape + spread**.

---

3. Stacked Area Plot

```
plt.stackplot(x, y1, y2, labels=['A','B'])
```

Use for **cumulative trends** (e.g., orders by cuisine over time).

---

4. Heatmap (basic with Matplotlib)

```
plt.imshow(df.corr(), cmap='coolwarm', interpolation='nearest')
plt.colorbar()
```

Use to visualize **correlation or matrix data**.

---

5. Hexbin Plot

```
plt.hexbin(df['cost_of_the_order'], df['delivery_time'], gridsize=20, cmap='Blues')
plt.colorbar()
```

Use for **large scatter data** with overlapping points.

---

6. Stem Plot

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

Use for **discrete events** (e.g., orders per day).

---

7. Step Plot

```
plt.step(x, y, where='mid')
```

Use for **step-like changes** (e.g., order counts per time slot).

---

8. Polar Plot

```
theta = np.linspace(0, 2*np.pi, 100)
r = np.sin(theta)
plt.polar(theta, r)
```

Use for **cyclical patterns** (e.g., time-of-day).

---

9. Quiver Plot

```
plt.quiver(X, Y, U, V)
```

Use for **vector fields** (direction + magnitude).

---

10. 3D Plot (Scatter)

```
from mpl_toolkits.mplot3d import Axes3D
ax = plt.figure().add_subplot(projection='3d')
ax.scatter(x, y, z)
```

Use to show **3D relationships** (cost vs delivery vs rating).

---

11. Contour Plot

```
plt.contour(X, Y, Z, cmap='viridis')
```

Use for **regions of equal values** (like elevation maps).

---

12. Bar with Error Bars

```
plt.bar(categories, values, yerr=errors, capsize=5)
```

Use for **averages + uncertainty**.