Python Matplotlib

Introduction to Matplotlib

- **Matplotlib** is a powerful Python library used for creating static, animated, and interactive visualizations.
- It is often used with **NumPy** to create plots of mathematical functions, data analysis, etc.
- The most commonly used part of Matplotlib is pyplot, which provides an easy interface for plotting.

Components of a Matplotlib Figure

1. **Figure**:
    - The entire canvas that holds everything in the plot (like a page in a book).
2. **Axes**:
    - The area within the figure where the data is plotted (like a graph paper).
    - A figure can have **one or more axes** (for example, multiple subplots).
3. **Axis**:
    - The **x-axis** (horizontal) and **y-axis** (vertical).
    - These axes show the **limits** and **tick marks** to help interpret data (e.g., time, quantity).
4. **Lines and Markers**:
    - **Lines**: Used to connect data points, showing trends or relationships (e.g., in line plots).
    - **Markers**: Symbols (e.g., circles, squares) that highlight individual data points (e.g., in scatter plots).
5. **Title and Labels**:
    - **Title**: Describes what the plot is about (e.g., "Sales Over Time").
    - **Labels**: Explain what the axes represent (e.g., "Time" for x-axis and "Sales" for y-axis).
6. **Legend**:
    - A small box that explains what the lines, markers, or colors in the plot represent (e.g., which line is for "Product A" and which one is for "Product B").
7. **Grid**:
    - The background gridlines make it easier to read the plot and compare values.

To begin with, we can install Matplotlib (if it's not already installed):
pip install matplotlib
Then, we can import matplotlib.pyplot for plotting:
import matplotlib.pyplot as plt

Basic Plotting in Matplotlib

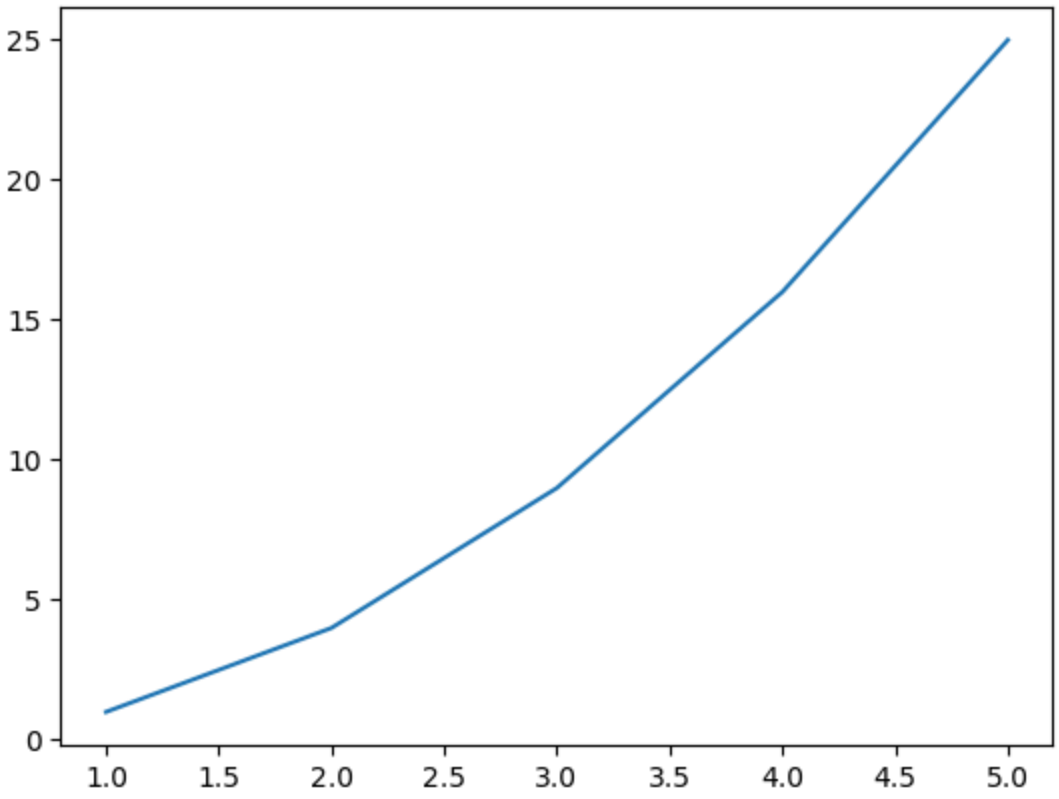
Matplotlib's plt.plot() function is the simplest way to create basic line plots. Let's start by creating a basic line plot:

Code Example:

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

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

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

Output:

This will display a simple line plot where x values are plotted on the x-axis and y values on the y-axis.



Markers and Line Styles

Markers:

Markers are symbols placed on the data points to emphasize their position on the graph. We can customize the markers by using the marker parameter in the plt.plot() function.

Common Marker Styles:

- **'o' : Circle**
- **'^' : Triangle up**
- **'s' : Square**
- **'*' : Star**
- **'x' : Cross**

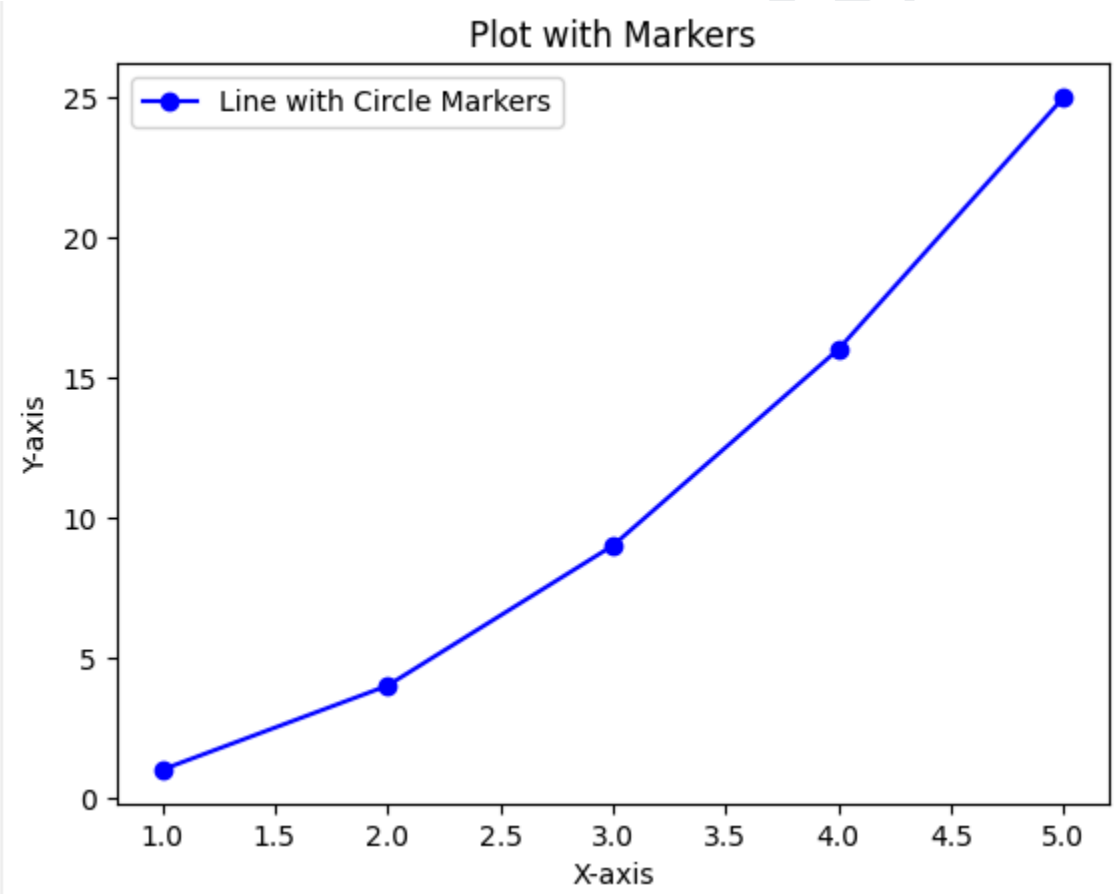
Code Example:

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

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

# Line plot with markers
plt.plot(x, y, marker='o', color='b', linestyle='-', label="Line with Circle Markers")

# Display the plot
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Plot with Markers")
plt.legend()
plt.show()
```

Output:

This plot will show a blue line with circle markers placed on each data point.



Line Styles:

You can change the line style by specifying the linestyle argument. The available options are:

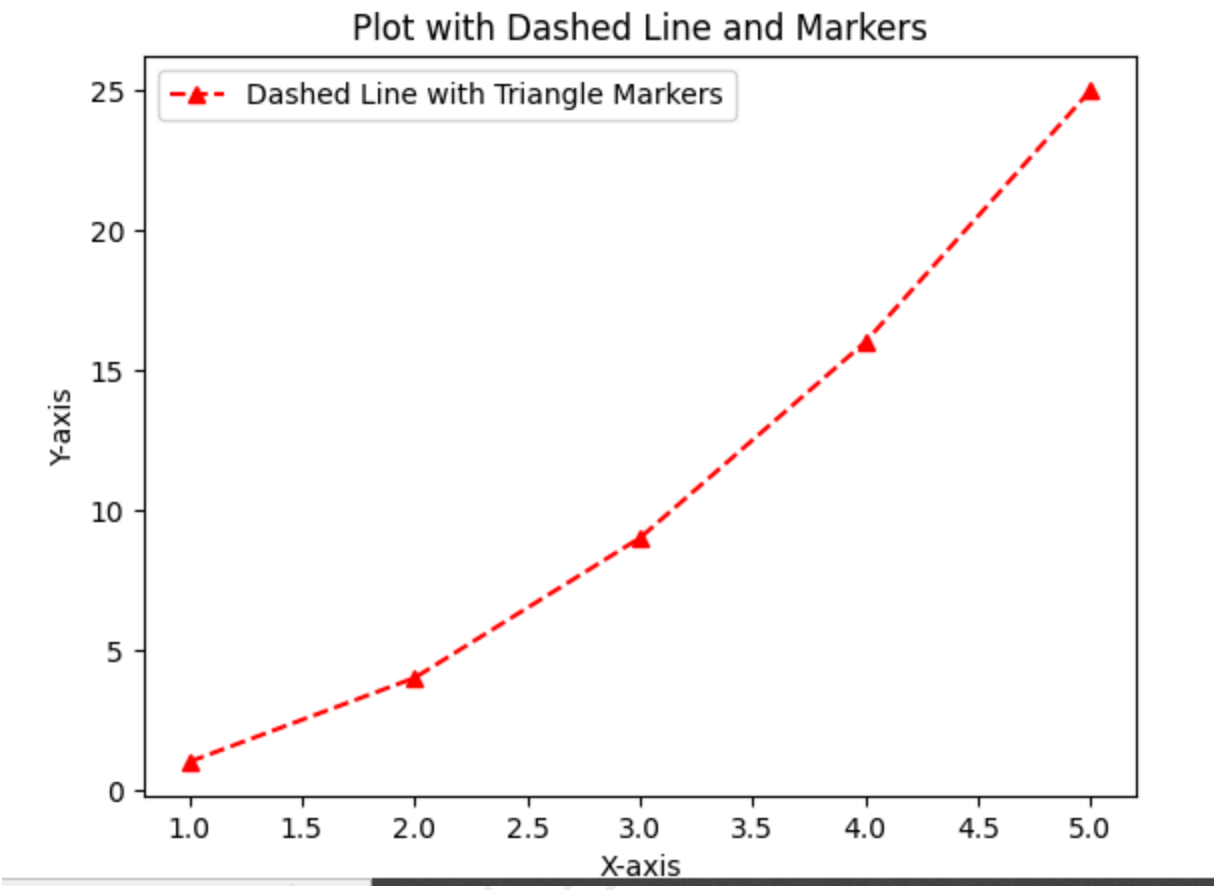
- '-' : Solid line (default)
- ':' : Dotted line
- '--' : Dashed line
- '-.' : Dash-dot line

Code Example:

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Line plot with dashed lines and markers
plt.plot(x, y, linestyle='--', marker='^', color='r', label="Dashed Line with Triangle Markers")

# Display the plot
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Plot with Dashed Line and Markers")
plt.legend()
plt.show()
```

Output:

This plot will show a red dashed line with triangle markers at each data point.



4. Adding Labels to Axes

Adding labels to the axes makes your plot more readable and provides context. The functions used for labeling are:

- plt.xlabel() for the x-axis
- plt.ylabel() for the y-axis

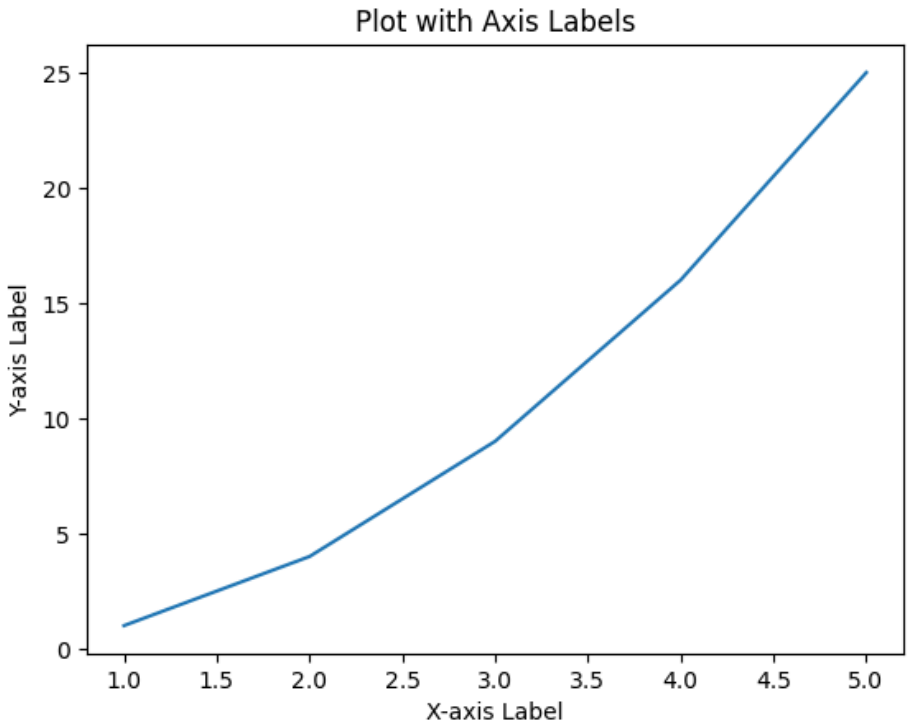
Code Example:

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Basic line plot with labels
plt.plot(x, y)

# Adding labels to axes
```

```
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')

# Adding a title
plt.title('Plot with Axis Labels')

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

Output:

This plot will show a basic line graph with the labels on the x-axis and y-axis as specified in the code.



5. Adding a Title

To add a title to your plot, use plt.title(). A title helps to explain what the plot is showing, which is especially useful for presentations or reports.

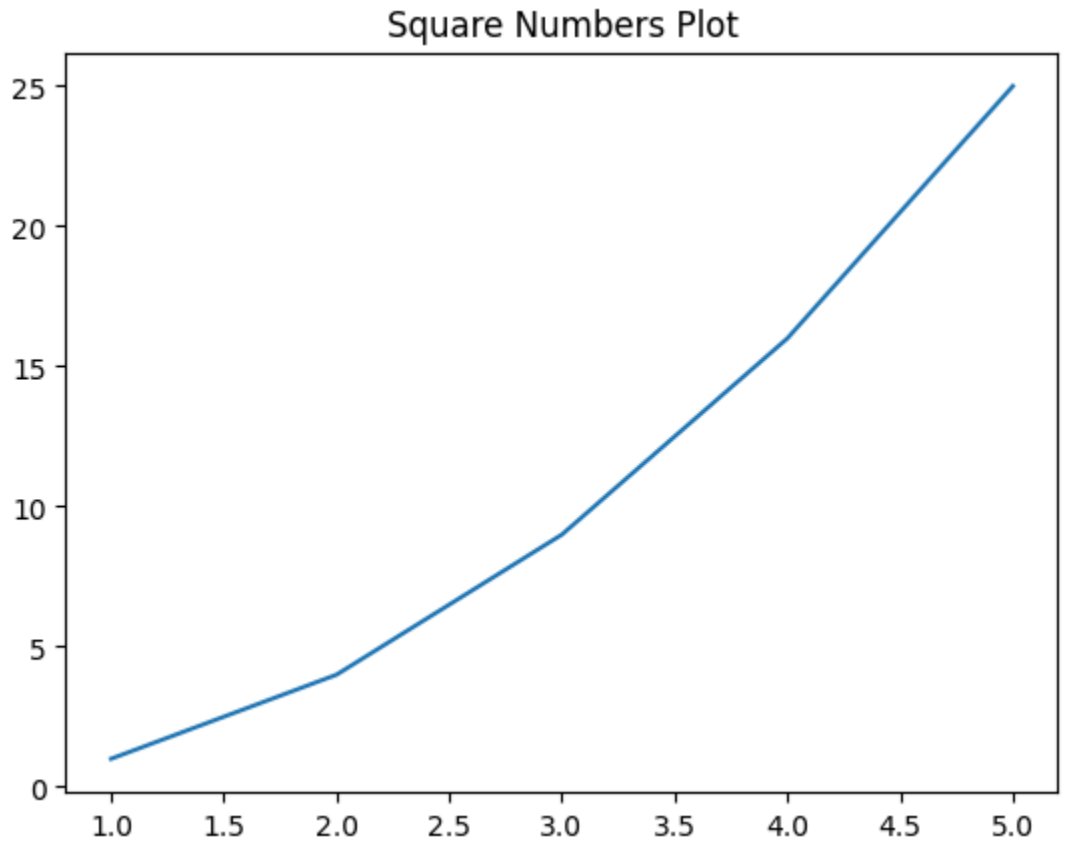
Code Example:

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

```
# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

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

# Add a title
plt.title('Square Numbers Plot')

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

Output:

The plot will display a title, "Square Numbers Plot", above the graph, describing the data being visualized.



Grid Lines

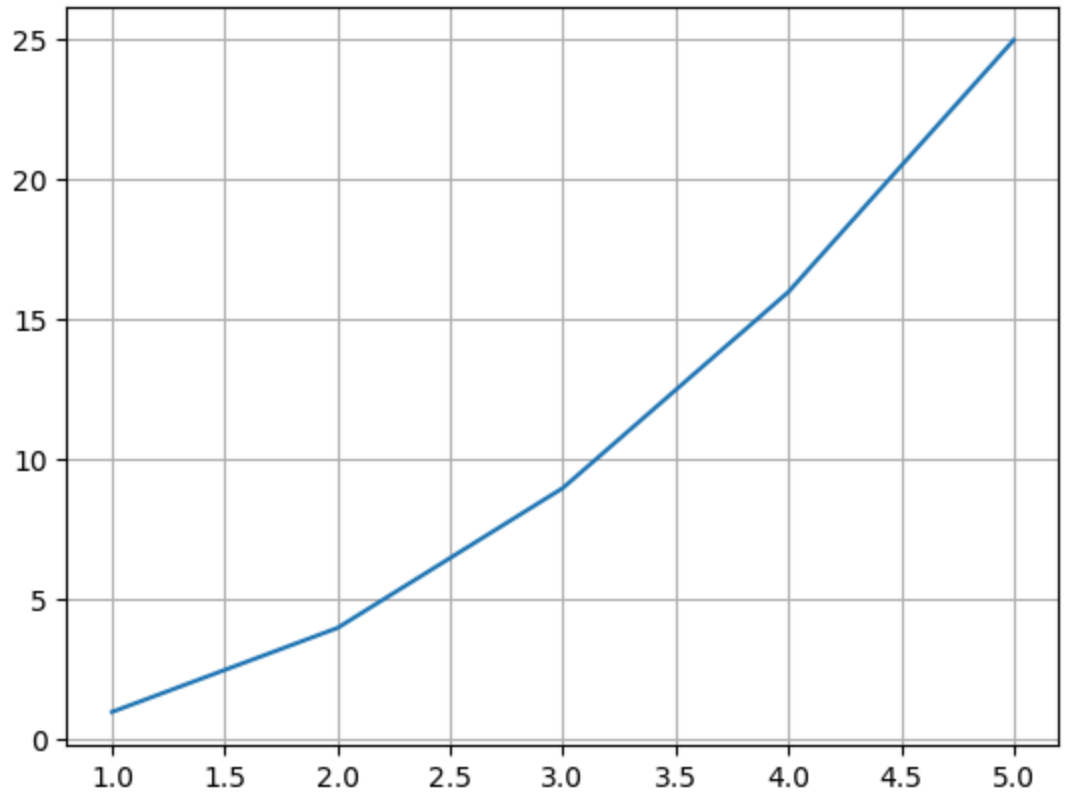
Grid lines are helpful for reading the values off a plot. They can be toggled on or off using plt.grid(). You can also customize the appearance of the grid lines (e.g., change the line style or color).

Code Example:

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Line plot with grid
plt.plot(x, y)

# Adding grid lines
plt.grid(True)  # Show grid

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

Output:

This plot will show a grid in the background, making it easier to interpret the values of the data points on the plot.



Customizing the Grid:

You can modify the grid lines, such as making them dotted or changing the color, using the linestyle and color parameters in the plt.grid() function.

You can customize the grid lines in several ways:

1. **Line Style**: You can change the style of the grid lines, such as making them solid, dashed, or dotted.
2. **Line Color**: You can adjust the color of the grid lines.
3. **Line Width**: You can change the thickness of the grid lines.

Grid Customization Parameters:

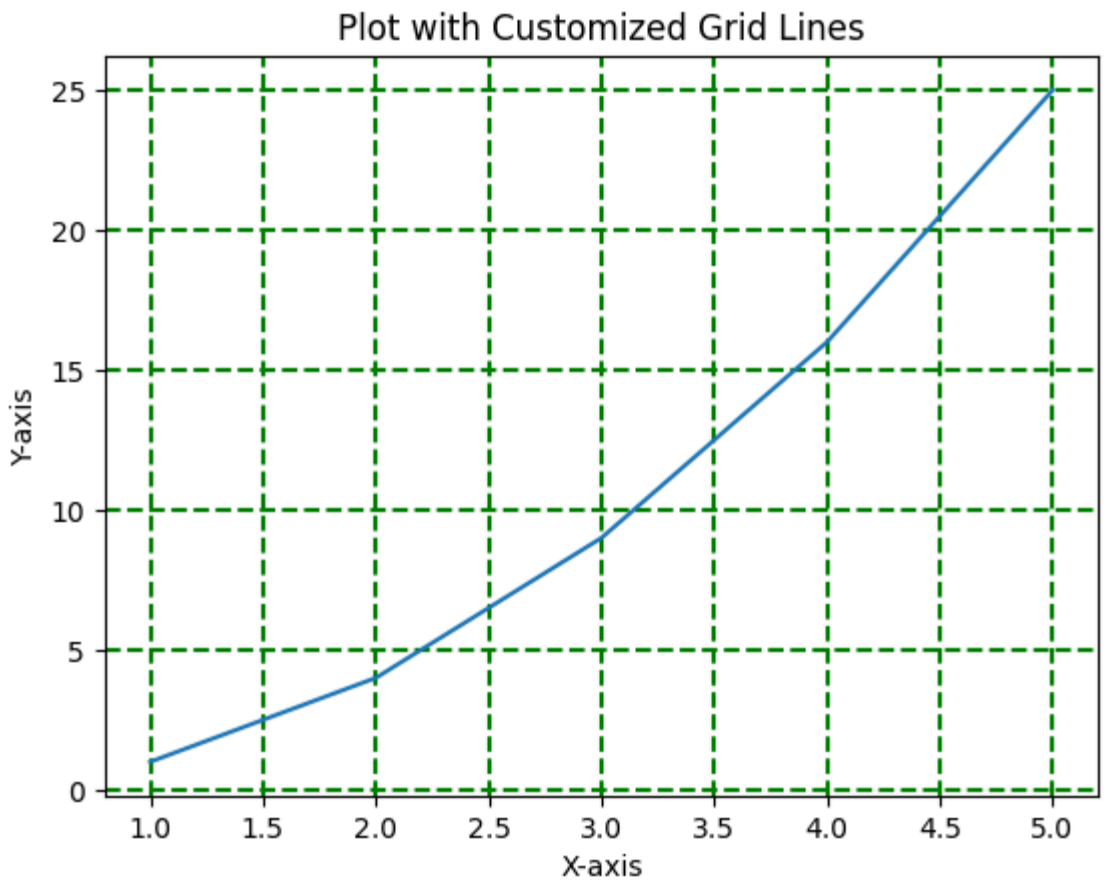
- linestyle: Defines the style of the grid lines (e.g., solid '-', dashed '--', dotted ':').
- color: Defines the color of the grid lines (e.g., 'r' for red, 'g' for green, 'blue', etc.).

- linewidth: Defines the width of the grid lines (e.g., 1, 2, 3).
- alpha: Adjusts the transparency of the grid lines (e.g., 0.5 for 50% transparency).

Code Example (Customized Grid):

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Line plot
plt.plot(x, y)

# Customizing grid lines
plt.grid(True, linestyle='--', color='g', linewidth=1.5)  # Green dashed grid lines with 1.5 width

# Adding labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Plot with Customized Grid Lines")

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

Output:-



In addition to the basic grid properties, you can also enable the grid for only the **x-axis** or **y-axis**. This can be useful when you only need to show horizontal or vertical reference lines.

Grid Lines on Only the X-axis or Y-axis:

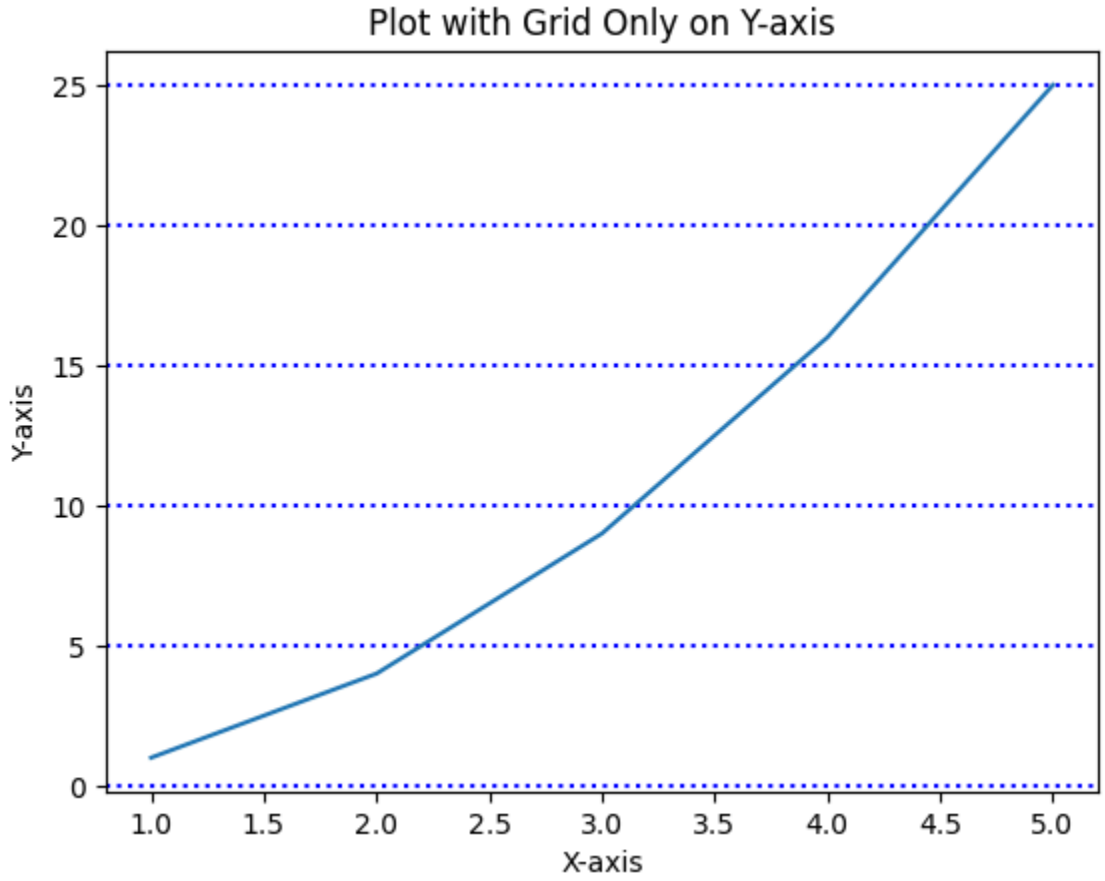
- axis='x': Display grid lines only for the x-axis.
- axis='y': Display grid lines only for the y-axis.

Code Example (Grid on Y-axis Only):

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Line plot
plt.plot(x, y)

# Enabling grid only for the y-axis
```

```
plt.grid(True, axis='y', linestyle=':', color='b', linewidth=1.5)  # Blue dotted lines for y-axis

# Adding labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Plot with Grid Only on Y-axis")

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

Output:-



Grid with Transparency (Alpha Value):

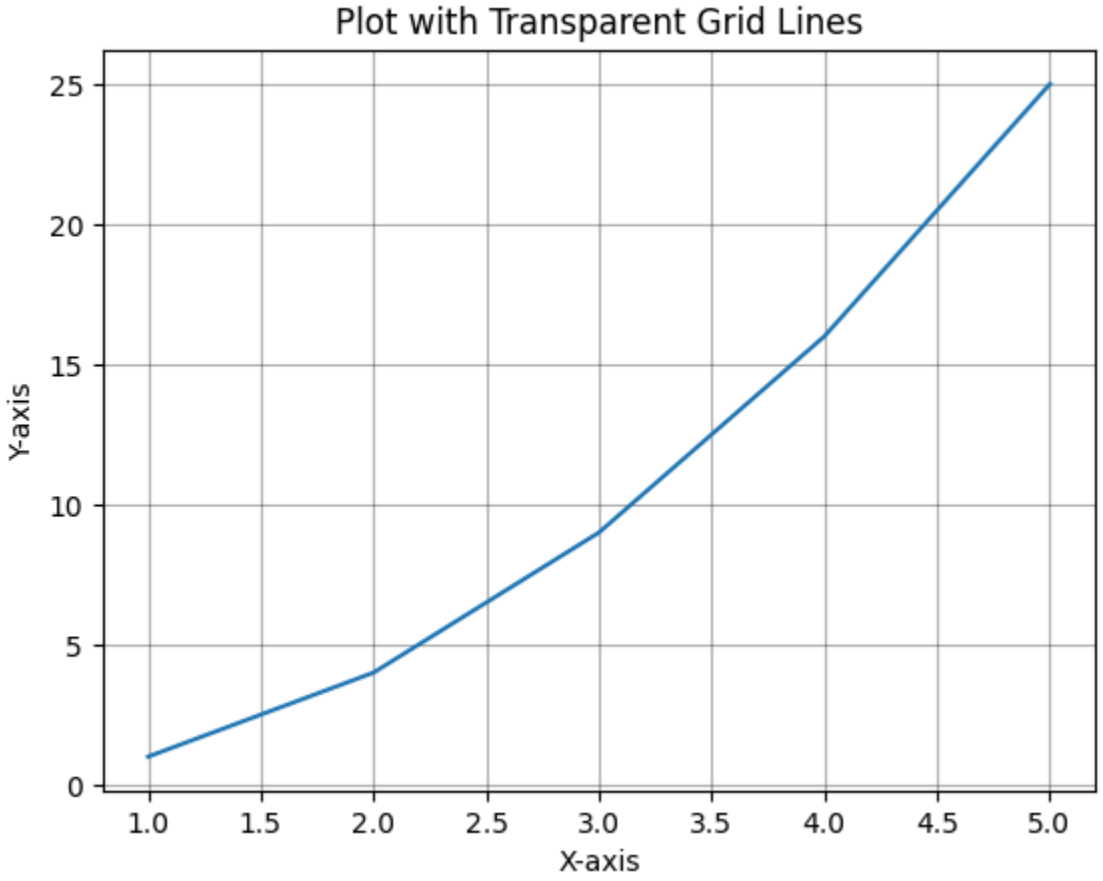
If you want to make the grid lines slightly transparent, you can set the alpha parameter. This can make the grid less intrusive while still keeping it visible.

Code Example (Grid with Transparency):

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
```

```
# Line plot
plt.plot(x, y)

# Adding grid with transparency (alpha)
plt.grid(True, linestyle='-', color='black', alpha=0.3)  # Semi-transparent black grid lines

# Adding labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Plot with Transparent Grid Lines")

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

Output:-



Subplots in Matplotlib

Subplots allow you to display multiple plots in a single figure. This is useful when you want to compare multiple visualizations side by side or in a grid layout.

Creating Subplots

You use plt.subplot(nrows, ncols, index) to create subplots.

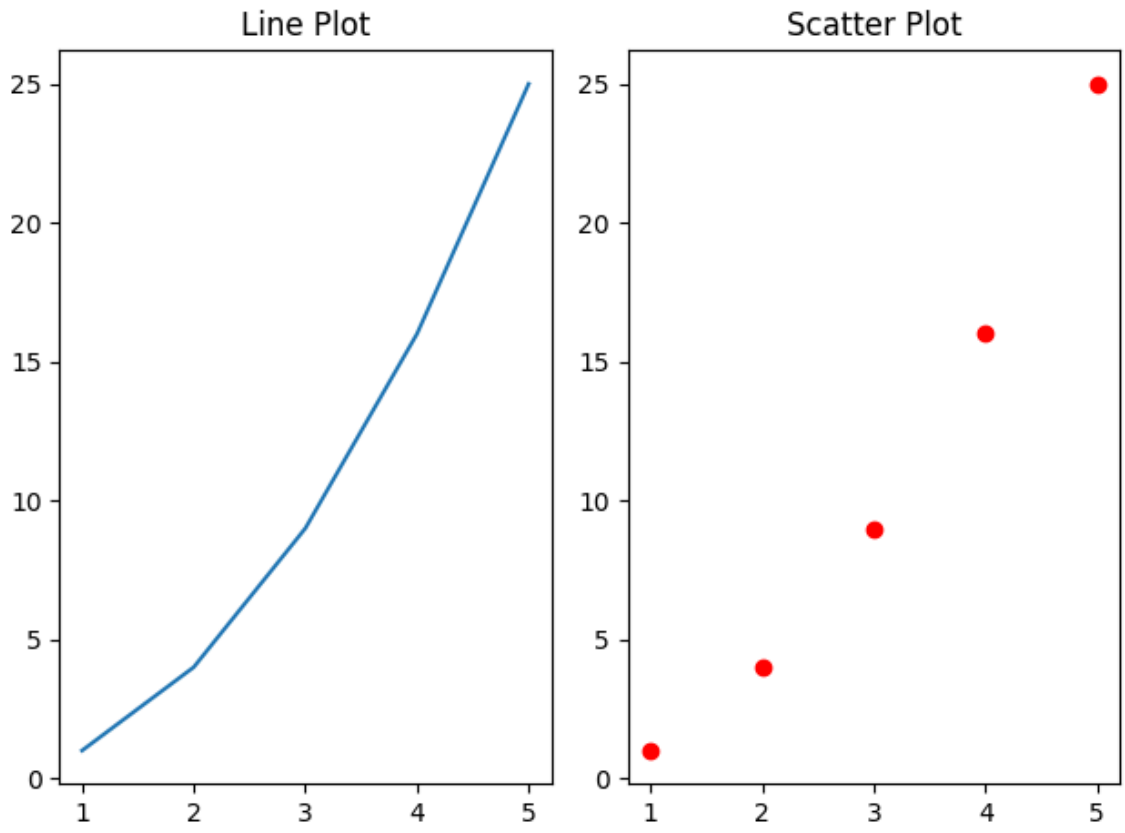
- nrows: number of rows of subplots.
- ncols: number of columns of subplots.
- index: the position of the current subplot in a 1-based index (starting from 1).

Code Example (Multiple Subplots in 1 Row and 2 Columns):

```
import matplotlib.pyplot as plt

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

# First subplot (row 1, column 2, position 1)
plt.subplot(1, 2, 1)  # (1 row, 2 columns, position 1)
plt.plot(x, y)
plt.title("Line Plot")

# Second subplot (row 1, column 2, position 2)
plt.subplot(1, 2, 2)  # (1 row, 2 columns, position 2)
plt.scatter(x, y, color='r')
plt.title("Scatter Plot")

# Display the plot
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

Output:

Two plots displayed side by side—one is a line plot and the other is a scatter plot.



Subplot Layouts:

You can adjust the layout as required, for example:

- **(2 rows, 2 columns)** → plt.subplot(2, 2, index)
- **(3 rows, 1 column)** → plt.subplot(3, 1, index)

Scatter Plots

A scatter plot displays individual data points as dots on a 2D plane, typically used to show relationships between two continuous variables.

Creating a Scatter Plot

You use plt.scatter(x, y) to create a scatter plot, where x and y are your data points.

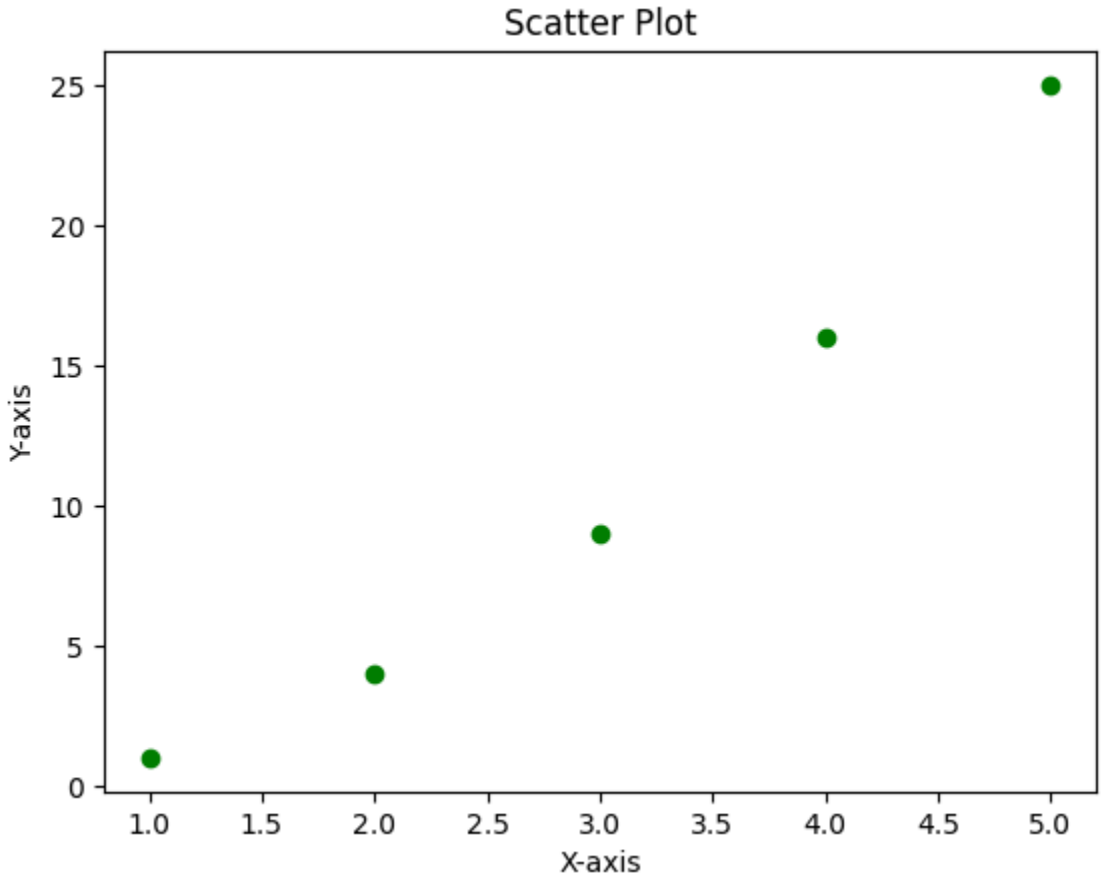
Code Example:

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

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

# Scatter plot
plt.scatter(x, y, color='g', marker='o')  # Green circles
plt.title("Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output:

A scatter plot with green circle markers, where each point represents a data value on the x and y axes.



Customization:

You can also customize the markers in a scatter plot using parameters like:

- color: Change marker color (e.g., 'r', 'g', 'b')

- marker: Change marker type ('o' for circles, '^' for triangles, etc.)
- size: Control the size of the markers using s=size
- alpha: Control transparency with alpha=0.5

3. Bar Plots

Bar plots are useful to compare discrete categories. In a bar plot, the length of each bar represents the value of the corresponding category.

Creating a Bar Plot

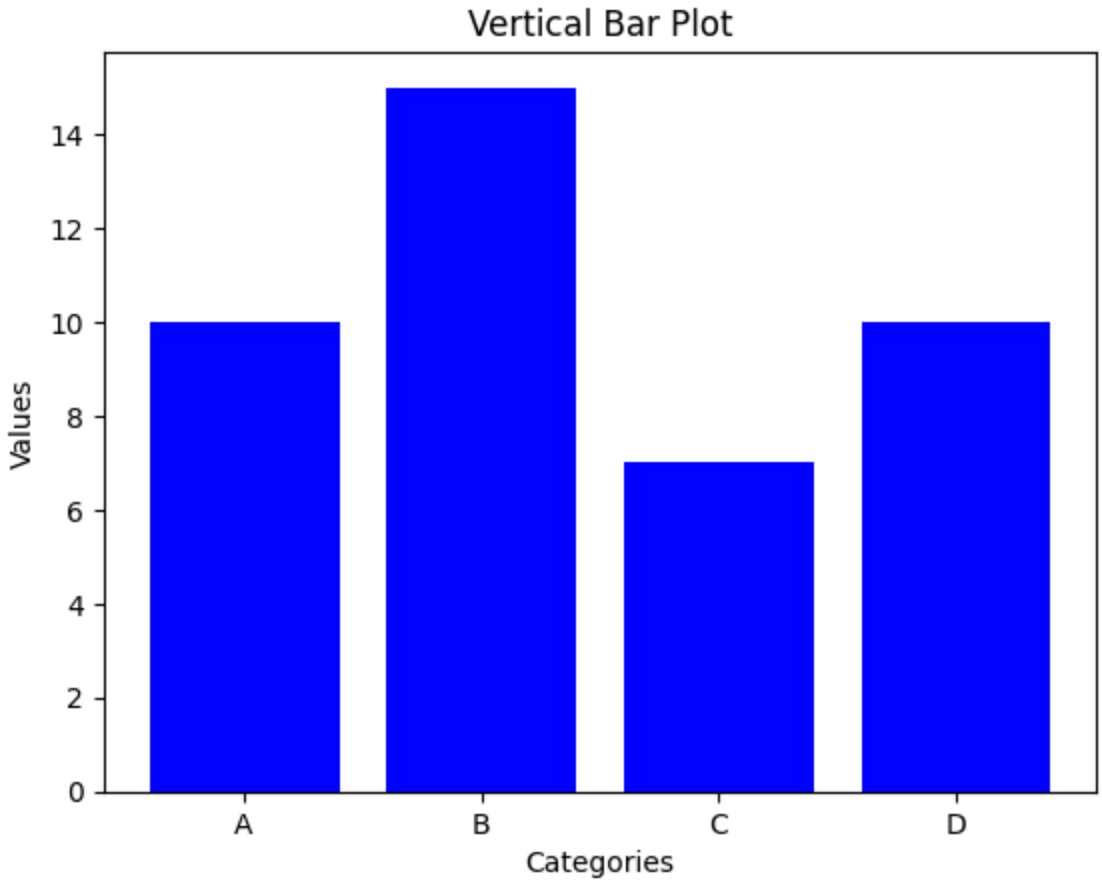
You use plt.bar() to create vertical bar plots and plt.barh() for horizontal bar plots.

Code Example (Vertical Bar Plot):

```
import matplotlib.pyplot as plt

categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 10]

# Vertical bar plot
plt.bar(categories, values, color='b')  # Blue bars
plt.title("Vertical Bar Plot")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```

Output:

A bar chart showing values for each category (A, B, C, D).



Code Example (Horizontal Bar Plot):
plt.barh(categories, values, color='orange')

Customization:

You can adjust the bar width, color, edge, and alignment:

- width: Controls the width of the bars.
- color: Changes the color of the bars.
- align: Aligns bars left or center.

Histograms

Histograms are used to display the distribution of a dataset, grouping data into bins. They are helpful for visualizing the frequency of data within certain ranges.

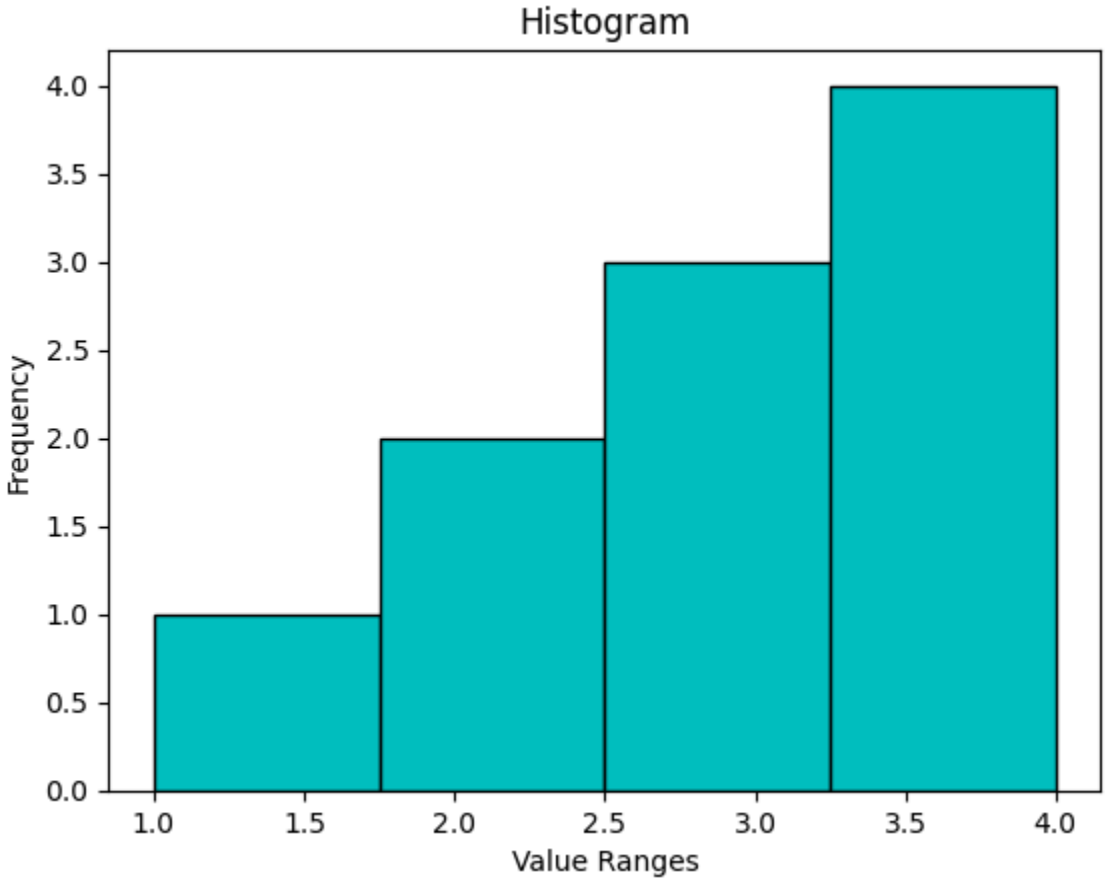
Creating a Histogram

You use plt.hist() to create histograms. You can define the number of bins to divide your data.

```
import matplotlib.pyplot as plt

data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

# Histogram with 4 bins
plt.hist(data, bins=4, color='c', edgecolor='black')
plt.title("Histogram")
plt.xlabel("Value Ranges")
plt.ylabel("Frequency")
plt.show()
```

Output:

A histogram with 4 bins showing the distribution of the data.



Customzation:

- bins: Defines the number of bins (intervals) to group the data.

- color: Sets the color of the bars in the histogram.
- edgecolor: Sets the color of the edges of the bars.

---

5. Pie Charts

Pie charts are used to show the proportion of different categories relative to the whole. Each slice of the pie represents a category, and the angle is proportional to the quantity.

Creating a Pie Chart

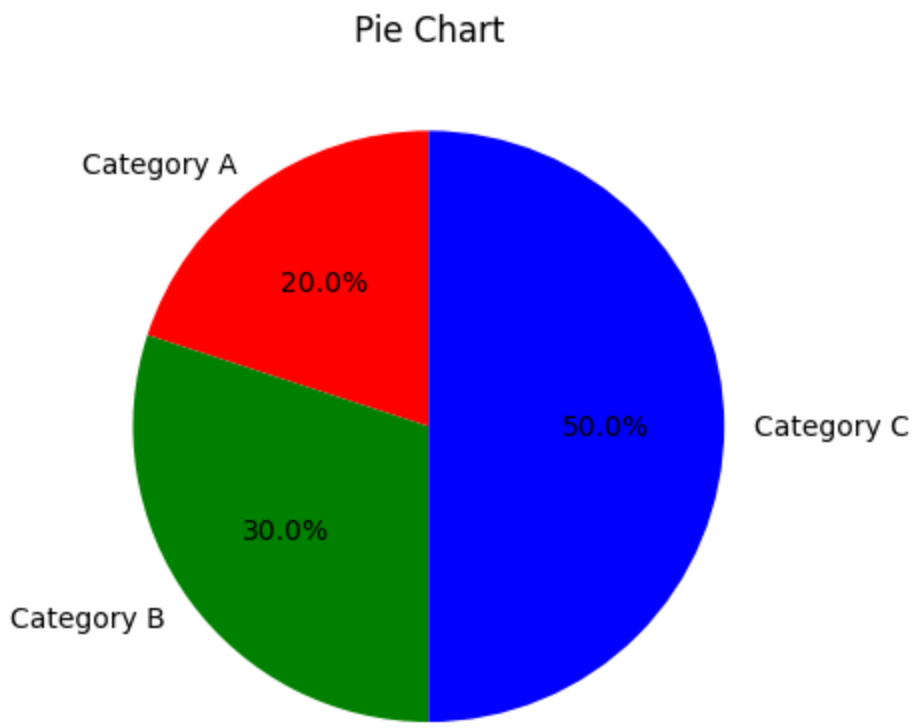
You use plt.pie() to create a pie chart, passing in the data to represent as proportions.

Code Example:

```
import matplotlib.pyplot as plt

sizes = [20, 30, 50]
labels = ['Category A', 'Category B', 'Category C']

# Pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=['red', 'green', 'blue'])
plt.title("Pie Chart")
plt.show()
```

Output:

A pie chart showing the proportion of each category, with the percentage displayed on each slice.



Customization:

- autopct: Displays percentage on each slice (e.g., '%1.1f%%' for one decimal place).
- startangle: Rotates the pie chart to the specified angle (e.g., startangle=90).
- colors: Custom colors for each slice.