# Matplotlib: Introduction to Visualization in Python

**What is Matplotlib?**

- **Definition**: Matplotlib is a low-level graph plotting library in Python designed for creating static, interactive, and animated visualizations. It serves as a powerful utility for data visualization in Python programming.
- **Purpose**: To create professional-grade visualizations for data analysis and reporting.

---

**History of Matplotlib**

- **Created by**: John D. Hunter
- **Developed**: 2002
- **Initial Release**: 2003

Matplotlib was inspired by MATLAB's plotting capabilities and has since become the backbone of visualization in Python, with various high-level libraries like Seaborn built on top of it.

---

## Installing Matplotlib on Windows

1. **Open Command Prompt**: Ensure you have administrative privileges to install Python packages.
2. **Set up User Account**: Confirm that Python and pip are installed and added to the system PATH.
3. **Run the Installation Command**:

```
py -m pip install matplotlib
```

4. **Verify Installation**: After installation, verify by importing the library in Python:

```
import matplotlib
print(matplotlib.__version__)
```

Ensure that your system is connected to the internet while running the pip command.

---

## Using Matplotlib

1. **Importing Matplotlib**:
    - Import the core library:

      ```
      import matplotlib
      ```

    - Import the commonly used `pyplot` submodule, typically aliased as `plt`:

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

2. **Check the Version**: To confirm which version of Matplotlib is installed, use the following:

   ```
   import matplotlib as mt
   print(mt.__version__)
   ```

---

## Key Features of Matplotlib

- **Pyplot Submodule**: Most of the utility functions for creating visualizations lie in `matplotlib.pyplot`, which simplifies plotting through concise function calls.
- **Customization**: Offers extensive control over every element of a figure, including colors, markers, line styles, and labels.
- **Integration**: Seamlessly integrates with NumPy, pandas, and Jupyter Notebooks for streamlined workflows.
- **Types of Visualizations**:
    - Line charts
    - Bar charts
    - Histograms
    - Scatter plots
    - Pie charts
    - Box plots
    - Heatmaps, and more.

---

## Basic Example: Plotting a Simple Graph

Here's a quick demonstration of creating a basic line plot:

```
import matplotlib.pyplot as plt

# Define data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create plot
plt.plot(x, y, label='Linear Growth', color='blue', marker='o')
```

```
# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Basic Line Plot')

# Add legend
plt.legend()

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

---

## Why Use Matplotlib?

1. **Flexibility**: Provides low-level control over visual elements for advanced customizations.
2. **Wide Range of Plots**: Supports nearly every type of data visualization.
3. **Extensibility**: Works well with high-level libraries like Seaborn for enhanced aesthetics.
4. **Community Support**: Extensive documentation and a vibrant community for troubleshooting.

---

## Hands-on Practice Ideas

- Plot data from a CSV file using pandas and Matplotlib.
- Experiment with different plot types (e.g., scatter, bar, and pie charts).
- Customize plot elements, including grid lines, ticks, and legends.
- Use Matplotlib in Jupyter Notebooks for interactive visualizations.

## Plotting x and y Points in Matplotlib

The `plot()` function in Matplotlib is a versatile tool for drawing points (markers) and lines on a 2D diagram. By default, it connects points with a line, making it suitable for creating line charts, trends, or simple graphs.

---

## Key Concepts:

1. **Function Overview**:
    - **`plot(x, y)`**: Plots a line or points connecting the x and y coordinates.
    - **Parameters**:
        - **x**: An array-like structure (list, tuple, or NumPy array) that contains the values for the x-axis (horizontal).
        - **y**: An array-like structure for the y-axis (vertical).

- If `y` is omitted, the function assumes values starting from 0 for the x-axis and takes `x` as the y-values.

---

## Plotting Points:

The `plot()` function can be used to draw specific points on a diagram. By default:

- It draws a **line** connecting consecutive points.
- **Markers** can be added to highlight the individual points explicitly.

**Example: Plot Points**

```
import matplotlib.pyplot as plt

# Define x and y coordinates
x = [0, 2, 4, 6]
y = [0, 50, 150, 250]

# Plot the points
plt.plot(x, y)

# Add title and labels
plt.title("Plotting x and y Points")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Show the graph
plt.show()
```

---

## Drawing a Line Between Points:

When a continuous line is needed to connect the points, the `plot()` function handles it by default.

**Example: Draw a Line** Draw a line connecting points (0,0) to (6,250):

```
import matplotlib.pyplot as plt

# Data points
x = [0, 6]
y = [0, 250]

# Plot the line
plt.plot(x, y)

# Label the graph
plt.title("Drawing a Line")
```

```
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

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

---

## Customizing the Line and Markers

Matplotlib allows you to customize the appearance of the line and markers:

- **Line Style**: `'-'` (solid), `'--'` (dashed), `':'` (dotted), etc.
- **Marker Style**: `'o'`, `'^'`, `'s'`, etc., to represent points.
- **Color**: Specify the line color using predefined color codes (e.g., `'b'` for blue, `'r'` for red).

### Example: Customize Line and Markers

```
import matplotlib.pyplot as plt

# Define points
x = [0, 2, 4, 6]
y = [0, 50, 150, 250]

# Customize the plot
plt.plot(x, y, linestyle='--', marker='o', color='r')

# Add labels and title
plt.title("Customized Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

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

---

## Adding Grid Lines for Better Readability

Grid lines enhance readability by making it easier to interpret data points.

### Example: Enable Grid

```
import matplotlib.pyplot as plt

# Data points
x = [0, 2, 4, 6]
y = [0, 50, 150, 250]

# Plot with grid
plt.plot(x, y, marker='o')
plt.grid()
```

```
# Add title and labels
plt.title("Line Plot with Grid")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Show plot
plt.show()
```

---

## Combining Multiple Lines

You can plot multiple lines on the same graph by calling `plot()` multiple times or passing additional x, y pairs.

### Example: Plot Multiple Lines

```
import matplotlib.pyplot as plt

# Line 1 data
x1 = [0, 2, 4, 6]
y1 = [0, 50, 150, 250]

# Line 2 data
x2 = [0, 2, 4, 6]
y2 = [0, 30, 90, 180]

# Plot the first line
plt.plot(x1, y1, label='Line 1', color='b', marker='o')

# Plot the second line
plt.plot(x2, y2, label='Line 2', color='g', linestyle='--')

# Add title, labels, legend, and grid
plt.title("Multiple Lines on One Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid()

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

---

## Tips for Beginners:

1. Always label your axes and give a title for clarity.
2. Use markers to differentiate between closely packed points.
3. Add a legend when plotting multiple lines to distinguish datasets.
4. Experiment with colors, line styles, and marker styles to improve aesthetics.

---

## Hands-on Exercises:

1. Plot a line connecting (0,0), (3,100), and (6,200) with square markers and a dashed line.
2. Add a grid to the graph and customize its color and style.
3. Create a graph with two intersecting lines and add a legend to identify them.

## Plotting Without Lines

To plot only markers without connecting them with lines, use the format string parameter `'o'`, which represents circular markers.

## Example:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 6])
y = np.array([0, 250])

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

**Explanation:**

- `plt.plot(x, y, 'o')`: Plots the points `(0, 0)` and `(6, 250)` with circular markers.

---

## Plotting Multiple Points

To plot multiple points, pass arrays of `x` and `y` values.

## Example:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([1, 2, 6, 8])
y = np.array([3, 8, 1, 10])

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

**Explanation:**

- This will connect the points `(1, 3)`, `(2, 8)`, `(6, 1)`, and `(8, 10)` with a line.

---

## Default X-Axis

If no `x` values are specified, Matplotlib automatically uses the indices of the `y` array as the `x` values.

## Example:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10, 5, 7])

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

### Explanation:

- The indices `[0, 1, 2, 3, 4, 5]` are used as the `x` values.

---

## Matplotlib Markers

Markers are used to emphasize points in a plot. You can specify markers using the `marker` keyword argument.

## Example:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, marker='o')
plt.show()
```

### Common Marker Symbols:

| Marker | Description |
|---|---|
| `'o'` | Circle |
| `'*'` | Star |
| `'.'` | Point |
| `'x'` | X (thin) |

| | |
|---|---|
| `'X'` | X (filled) |
| `'s'` | Square |
| `'D'` | Diamond |

---

## Format Strings (fmt)

The `fmt` parameter allows you to define the marker, line style, and color in a single string.

## Syntax:

```
marker | line | color
```

## Example:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, 'o:r')
plt.show()
```

### Explanation:

- `'o'`: Circular markers.
- `':'`: Dotted line.
- `'r'`: Red color.

---

## Line Styles

You can customize the appearance of lines using the following styles:

| Line Syntax | Description |
|---|---|
| `'-'` | Solid Line |
| `':'` | Dotted Line |
| `'--'` | Dashed Line |
| `'-. '` | Dash-Dot Line |

## Color Options

Specify the line color using the following single-character codes:

| Color Syntax | Description |
|---|---|
| `'r'` | Red |
| `'g'` | Green |
| `'b'` | Blue |
| `'c'` | Cyan |
| `'m'` | Magenta |
| `'y'` | Yellow |
| `'k'` | Black |
| `'w'` | White |

---

## Marker Size

To adjust the size of markers, use the `markersize` or `ms` argument.

## Example:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, marker='o', ms=20)
plt.show()
```

### Explanation:

- `ms=20`: Sets the marker size to 20.

## Adding Labels and Titles

## Labels for Axes:

Use `xlabel()` and `ylabel()` to label axes.

```
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([0, 8, 12, 20, 26, 38])

plt.plot(x, y)
plt.xlabel("Overs")
plt.ylabel("Runs")
plt.show()
```

## Title:

Add a title to the plot with `title()`.

```
plt.title("Sport Data")
```

## Font Customization:

Customize fonts with `fontdict`.

```
font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}

plt.xlabel("Overs", fontdict=font2)
plt.ylabel("Runs", fontdict=font2)
plt.title("Sport Data", fontdict=font1)
plt.show()
```

---

## Combining Multiple Lines

To plot multiple datasets, provide additional arrays.

```
x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])

x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```

---

## Marker Customization

## Markers Face Size

- **Keyword Argument:** `markerfacecolor` (short form `mfc`) controls the color of the marker face.
- **Example:**

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])
plt.plot(y, marker='o', ms=20, mfc='r')
plt.show()
```

---

## Linestyle Argument

- Use the `linestyle` (or `ls`) argument to change the line style.
- **Available Styles:**
  - `'-'` (solid)
  - `'--'` (dashed)
  - `'-.'` (dashdot)
  - `':'` (dotted)
- **Example:**

```
plt.plot(y, linestyle='dotted')
```

---

## Line Customization

## Line Color

- Use the `color` (or `c`) argument to set the line's color.

```
plt.plot(y, color='pink')
```

## Line Width

- Use the `linewidth` (or `lw`) argument to adjust line thickness.

```
plt.plot(y, linewidth=5)
```

---

## Labels and Title

## Adding Axis Labels

- Use `xlabel()` and `ylabel()` to name axes.

```
plt.xlabel("X-axis label")
plt.ylabel("Y-axis label")
```

## Adding a Title

- Use `title()` to add a title to the plot.

```
plt.title("Plot Title")
```

## Positioning the Title

- The `loc` argument in `title()` positions the title (`left`, `right`, `center`).

---

## Font Customization

- Use `fontdict` in `xlabel()`, `ylabel()`, and `title()` to set font properties like size, color, and family.

```
font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}
plt.xlabel("X-axis", fontdict=font1)
plt.ylabel("Y-axis", fontdict=font2)
plt.title("Title", fontdict=font1)
```

---

## Grid Lines

- Add grid lines using the `grid()` function.

```
plt.grid()
```

---

## Multiple Plots

## Subplots

- Use `subplot()` to create multiple plots in one figure.
- **Syntax:** `subplot(nrows, ncols, index)`

```
plt.subplot(1, 2, 1)
plt.plot(x, y)
plt.subplot(1, 2, 2)
plt.plot(x, z)
```

## Super Title

- Add a global title using `suptitle()`.

```
plt.suptitle("Overall Title")
```

---

## Scatter Plots

- Use `scatter()` for scatter plots.

- **Basic Example:**

```
plt.scatter(x, y)
```

## Customizing Scatter Plots

- **Set Color:** Use the `color` argument.

```
plt.scatter(x, y, color='red')
```

- **Color Each Dot:** Pass a list of colors for each point.

```
colors = ['red', 'blue', 'green']
plt.scatter(x, y, color=colors)
```

## Setting Font Properties for Title and Labels

You can customize font properties for plot titles and axis labels using the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()`.

### Example Code:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([0, 8, 12, 20, 26, 38])

font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}

plt.plot(x, y)
plt.xlabel('Overs', fontdict=font2)
plt.ylabel('Runs', fontdict=font2)
plt.title('Sport Data', fontdict=font1)
plt.show()
```

### Output:

The title and labels will be styled as specified by `font1` and `font2`.

---

## Positioning the Title

The title's position can be adjusted using the `loc` parameter in the `title()` method. Legal values are `'left'`, `'right'`, and `'center'` (default).

### Example Code:

```
plt.title('Sport Data', loc='left')
```

---

## Adding Grid Lines to a Plot

Grid lines enhance readability and can be added using the `grid()` function.

### Example Code:

```
plt.grid()
```

---

## Displaying Multiple Plots

### Using `subplot()`

The `subplot()` function allows you to display multiple plots in a single figure.

- **Syntax:** `plt.subplot(rows, columns, index)`

### Example Code:

```
# Plot 1
x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
plt.subplot(2, 1, 1)
plt.plot(x1, y1)

# Plot 2
x2 = np.array([0, 1, 2, 3])
y2 = np.array([10, 20, 30, 40])
plt.subplot(2, 1, 2)
plt.plot(x2, y2)

plt.show()
```

### Adding a Super Title:

Use `suptitle()` to add a title to the entire figure.

```
plt.suptitle('My Data')
```

---

## Creating Scatter Plots

### Basic Scatter Plot:

```
plt.scatter(x, y)
```

**Customizing Colors and Sizes:**

- Use the `color` parameter to set point colors.
- Use the `s` parameter to adjust point sizes.

**Example Code:**

```
mycolor = ['red', 'green', 'purple', 'lime', 'aqua', 'yellow']
size = [10, 60, 120, 80, 20, 190]
plt.scatter(x, y, color=mycolor, s=size)
```

**Adjusting Transparency:**

Use the `alpha` parameter to set transparency.

```
plt.scatter(x, y, alpha=0.5)
```

---

## Bar Plots

**Vertical Bar Graph:**

```
x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])
plt.bar(x, y)
```

**Horizontal Bar Graph:**

```
plt.barh(x, y)
```

**Adjusting Bar Width:**

```
plt.bar(x, y, width=0.5)
```

---

## Histograms

Histograms visualize frequency distributions. Use the `hist()` function.

**Example Code:**

```
x = np.random.normal(170, 10, 250)
plt.hist(x)
```

---

## Pie Charts

**Basic Pie Chart:**

```
x = np.array([35, 25, 25, 15])
plt.pie(x)
```

**Adding Labels:**

```
labels = ['Apples', 'Bananas', 'Cherries', 'Dates']
plt.pie(x, labels=labels)
```

**Customizing Start Angle:**

```
plt.pie(x, startangle=90)
```

---

## Setting Font Properties for Title and Labels

You can use the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to customize font properties for the title and labels.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([0, 8, 12, 20, 26, 38])

font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}

plt.plot(x, y)
plt.xlabel("Overs", fontdict=font2)
plt.ylabel("Runs", fontdict=font2)
plt.title("Sport Data", fontdict=font1)
plt.show()
```

---

## Positioning the Title

The `loc` parameter in `title()` can position the title. Legal values: `'left'`, `'right'`, `'center'`.

```
plt.title("Sport Data", loc='left')
```

---

## Adding Grid Lines

The `grid()` function adds grid lines to the plot.

```
plt.grid()
```

---

## Displaying Multiple Plots

The `subplot()` function allows multiple plots in one figure. It takes three arguments: rows, columns, and index of the plot.

### Example 1: Side-by-Side Plots

```
plt.subplot(1, 2, 1)
plt.plot(x, y)
plt.subplot(1, 2, 2)
plt.plot(x, y)
plt.show()
```

### Example 2: Stacked Plots

```
plt.subplot(2, 1, 1)
plt.plot(x, y)
plt.subplot(2, 1, 2)
plt.plot(x, y)
plt.show()
```

---

## Scatter Plots

The `scatter()` function creates scatter plots.

### Basic Scatter Plot

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

### Compare Two Scatter Plots

```
plt.scatter(x1, y1, color='red')
plt.scatter(x2, y2, color='green')
plt.show()
```

### Customizing Scatter Plots

- **Colors:** Use the `color` parameter with a list.
- **Sizes:** Use the `s` parameter with a list of sizes.
- **Transparency:** Use the `alpha` parameter for transparency.

```
colors = ['red', 'green', 'blue']
sizes = [20, 50, 80]
plt.scatter(x, y, color=colors, s=sizes, alpha=0.5)
plt.show()
```

---

## Bar Graphs

### Vertical Bars

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

### Horizontal Bars

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

### Adjusting Bar Width

```
plt.barh(x, y, width=0.5)
plt.show()
```

---

## Histograms

Histograms show frequency distributions.

```
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

---

## Pie Charts

### Basic Pie Chart

```
x = np.array([35, 25, 25, 15])
plt.pie(x)
plt.show()
```

### Adding Labels

```
labels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(x, labels=labels)
plt.show()
```

### Customizing Start Angle

```
plt.pie(x, labels=labels, startangle=90)
plt.show()
```

### Exploding a Wedge

```
explode = [0.2, 0, 0, 0]
plt.pie(x, labels=labels, explode=explode)
plt.show()
```

### Adding Shadows

```
plt.pie(x, labels=labels, explode=explode, shadow=True)
plt.show()
```

### Custom Wedge Colors

```
colors = ["black", "hotpink", "b", "#4CAF50"]
plt.pie(x, labels=labels, colors=colors)
plt.show()
```

### Adding a Legend

```
plt.legend(title="Four Fruits:")
plt.show()
```