Matplotlib

This powerful library of matplotlib creates almost every type of data visualization with simple line plots to complex 2D and 3D graphs. It is quite famous for data exploration and presentation purposes, providing wide customization scope on colors, labels, and layouts. Whether you are a beginner or pro, Matplotlib makes data visualization intuitive and efficient.

Installing Matplotlib

```
pip install matplotlib
```

Importing Matplotlib

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

Basic Plots

Command	Execution
plt.plot(x, y)	Line plot connects the data points with continuous Techniques.
plt.scatter(x, y)	Scatter plot display individual data points
plt.bar(x,height)	Bar plot shows the distribution of data for a categorical variable.
plt.barh(y, width)	A horizontal bar plot is a version of the bar plot with the bars going horizontally instead of vertically.
plt.hist(data, bins=10)	Histogram useful for understanding frequency distribution in numerical data and picking up patterns such as skewness, outliers, or gaps.
plt.boxplot(data)	A box plot is another way of visual summarization of the data distribution, showing median and quartiles, as well as outliers.
plt.pie(sizes, labels=labels, autopct='%1.1f%%')	A pie chart is a circular statistical graphic used to illustrate numerical data as slices of the whole.

Styles and Themes

Command	Execution
plt.style.use('style_name')	Changing the style of your plots in Matplotlib really helps make them much better looking and more readable.

Saving and Showing

Command	Execution
plt.show()	In order to display a plot, use `plt.show()` following your plotting call.
plt.savefig('filename.png', dpi=300)	You can save a plot to a file using the function plt.savefig().

Customization

Command	Execution
plt.title("Title Here")	Customizing titles in Matplotlib makes your visualizations clearer and aesthetically pleasing.
plt.xlabel("X-axis Label") plt.ylabel("Y-axis Label")	Customizing axis labels in Matplotlib helps in making your plots more readable and presentable.
plt.grid(True)	Matplotlib's use of grids can make plots easier to read.
plt.style.use('seaborn-darkgrid')	Matplotlib offers several styles to alter the look of your plots.
plt.xlim(min, max) plt.ylim(min, max)	You can limit the range of your x-axis and y-axis in Matplotlib to concentrate only on a portion of your data.
plt.legend(["Label1", "Label2"])	Legends help in plotting interpretation because they explain each element presented to view in the graph.

Creating Subplots

Command	Execution
plt.subplot(rows, columns, index)	The plt.subplot(nrows, ncols, index) function in matplotlib creates a grid of subplots.
ig, axs = plt.subplots(2, 2) axs[0, 0].plot(x, y) axs[0, 1].bar(categories, values)	We use the `plt.subplots()` function for creating multiple subplots. It returns a figure and an array of axes.

Customizing Subplots

Command	Execution
plt.suptitle('Main Title for All Subplots')	For writing a general title that applies to all subplots of a figure using Matplotlib, you may use the function plt.suptitle().
plt.tight_layout()	Use the plt.subplots_adjust() function to adjust the layout of subplots in Matplotlib.

Advanced Features

Command	Execution
plt.annotate('Text', xy=(x, y), xytext=(x+1, y+1), arrowprops=dict(facecolor='black', arrowstyle='->'))	Annotations in Matplotlib are used to add text or markers that explain something about a point in a plot.
plt.yscale('log')	Logarithmic scales are very important in Matplotlib for plotting data that ranges over several orders of magnitude.
rom mpl_toolkits.mplot3d import Axes3D ax = plt.figure().add_subplot(projection='3d') ax.scatter(x, y, z)	Matplotlib is used for performing 3D plotting and visualizes data in three dimensions.

Matplotlib Cheat sheet

A Matplotlib cheat sheet is a concise guide that summarizes the key functions, commands, and techniques for creating visualizations using Matplotlib. It's a handy reference for beginners and experienced users alike helping them quickly recall how to generate and customize plots, charts and graphs efficiently.

Load the Matplotlib Libraries

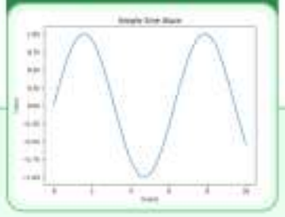
```
import matplotlib.pyplot as plt
import numpy as np
```

Basic plots

Line plots

```
import matplotlib.pyplot as plt
 # Sample data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Create a simple line plot
plt.plot(x, y, label="y = x^2")

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

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

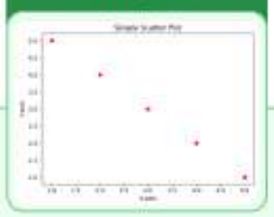
Output



Scatter plot

```
# Sample data
x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

# Create a scatter plot
plt.scatter(x, y, color='red')

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

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

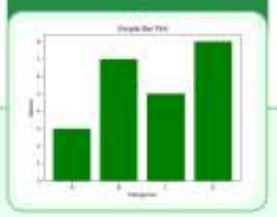
Output



Bar plot

```
# Categories and values
categories = ['A', 'B', 'C', 'D']
values = [3, 7, 5, 8]

# Create a bar plot
plt.bar(categories,values,
color='green')

# Add title and labels
plt.title("Simple Bar Plot")
plt.xlabel("Categories")
plt.ylabel("Values")

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

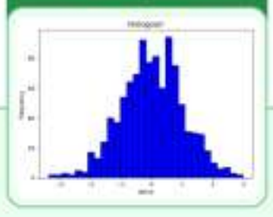
Output



Histogram

```
import numpy as np

# Generate random data
data=np.random.randn(1000)

# Create a histogram
plt.hist(data,bins=30,color='blue',
edgecolor='black')

# Add title and labels
plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")

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

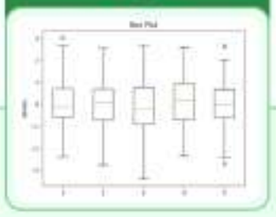
Output



Box Plot

```
 Sample data
data=  [np.random.normal(0,1,
100) for _ in range(5)]

# Create a box plot
plt.boxplot(data)

# Add title and labels
plt.title("Box Plot")
plt.ylabel("Values")

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

Output



Pie chart

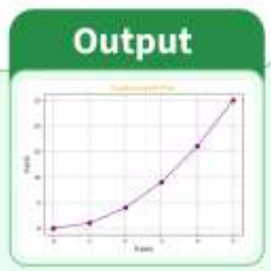
```
# Data for pie chart
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]

# Create a pie chart
plt.pie(sizes,
labels=labels,
autopct='%1.1f%%',
startangle=140)

# Add title
plt.title("Pie Chart")

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

Output

Customizing Plots

```
# Sample data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Create a customized plot
plt.plot(x,  y,  linestyle='-',  color='purple',
marker='o', markersize=8)

# Customize title and labels
plt.title("Customized  Plot",  fontsize=14,
color='orange')
plt.xlabel("X-axis", fontsize=12)
plt.ylabel("Y-axis", fontsize=12)

# Display grid
plt.grid(True)

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

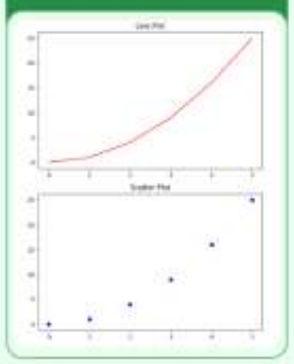
Output



Subplots

```
# Create subplots
fig, axs = plt.subplots(2, 1, figsize=(6, 8))

# Plot on the first subplot
axs[0].plot(x, y, 'r-')
axs[0].set_title("Line Plot")

# Plot on the second subplot
axs[1].scatter(x, y, color='blue')
axs[1].set_title("Scatter Plot")

# Display the plots
plt.tight_layout()
plt.show()
```

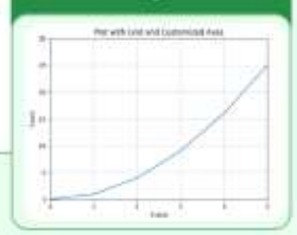
Output



Customized Grid and Axes

```
# Sample data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Create plot
plt.plot(x, y)

# Add grid
plt.grid(True,  which='both',  axis='both',
linestyle=':', color='gray')

# Customize axes limits
plt.xlim(0, 5)
plt.ylim(0, 30)

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

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

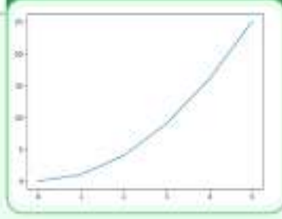
Output



Saving plot

```
# Sample data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Create plot
plt.plot(x, y)

# Save the plot
plt.savefig('plot.png')

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

Output



GeeksforGeeks