Guiding Material: matplotlib.pyplot, numpy, and pandas

Cosmic Charade

This guide covers the fundamental usage of three powerful libraries in Python: **`matplotlib.pyplot`** for plotting, **`numpy`** for numerical operations, and **`pandas`** for data manipulation and analysis. Together, they form the backbone of Python's data science and numerical computing ecosystem.

1. `import matplotlib.pyplot as plt`

- **`matplotlib`**: This is a plotting library used in Python for creating static, interactive, and animated visualizations.
- **`pyplot`**: A submodule of `matplotlib`, which provides a MATLAB-like interface for plotting.
- **`as plt`**: This creates an alias, so instead of typing `matplotlib.pyplot` every time you want to use it, you can just type `plt`. It is a common convention in Python programming to use `plt` as the alias for `pyplot`.

What It Does:

- `matplotlib.pyplot` provides functions to create various kinds of plots (line plots, bar charts, scatter plots, etc.).
- `plt` is used to refer to the functions from this submodule. For example, `plt.plot()` is used to create a line plot.
- **syntax**

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

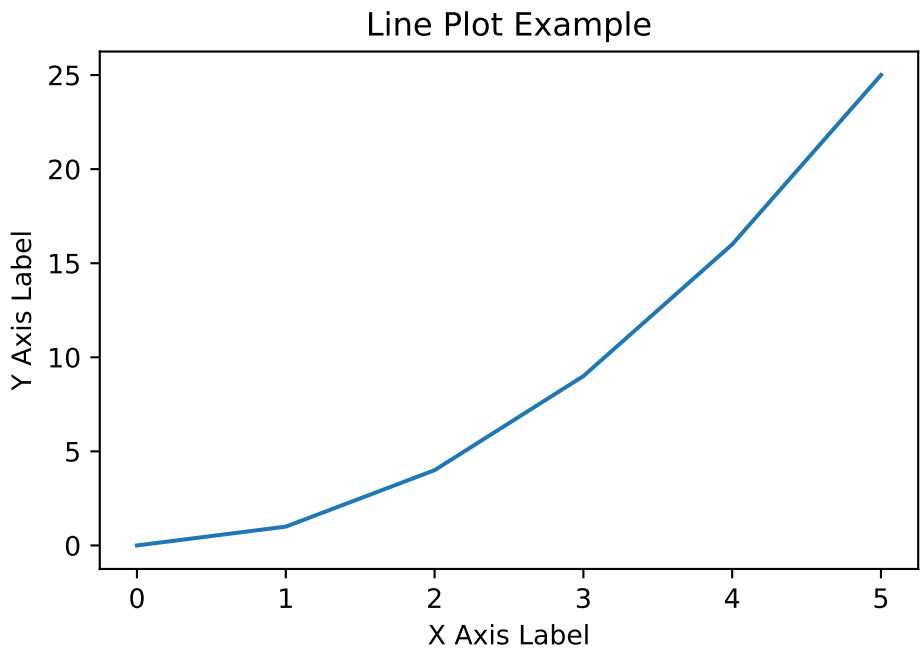
Key Functions and Examples:

1. **Line Plot**:

```
import matplotlib.pyplot as plt

# Data for plotting
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Create a line plot
plt.plot(x, y)
plt.xlabel("X Axis Label")
plt.ylabel("Y Axis Label")
plt.title("Line Plot Example")
plt.show()
```

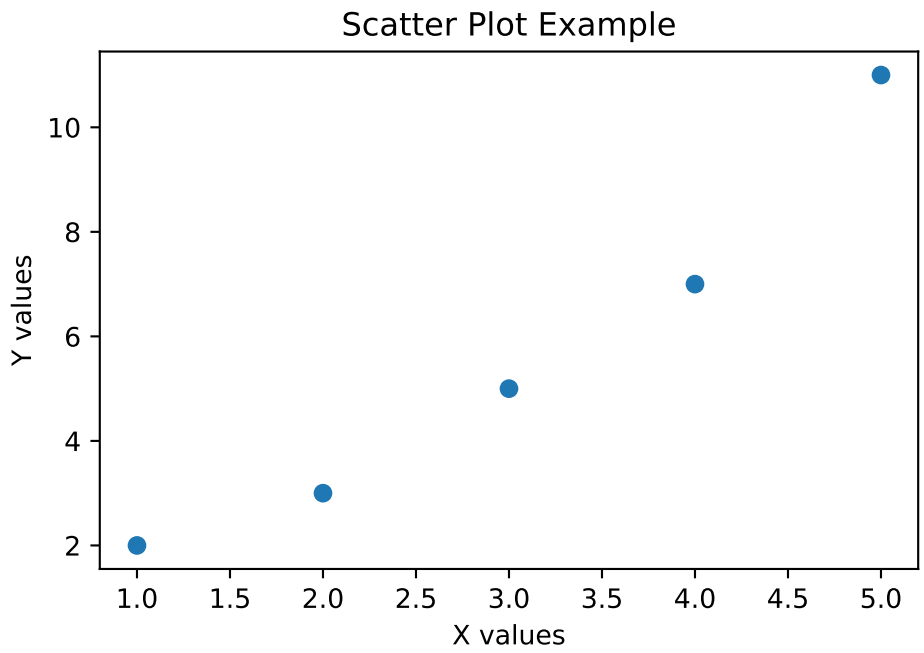


2. **Scatter Plot**:

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Create a scatter plot
plt.scatter(x, y)
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Scatter Plot Example")
plt.show()
```

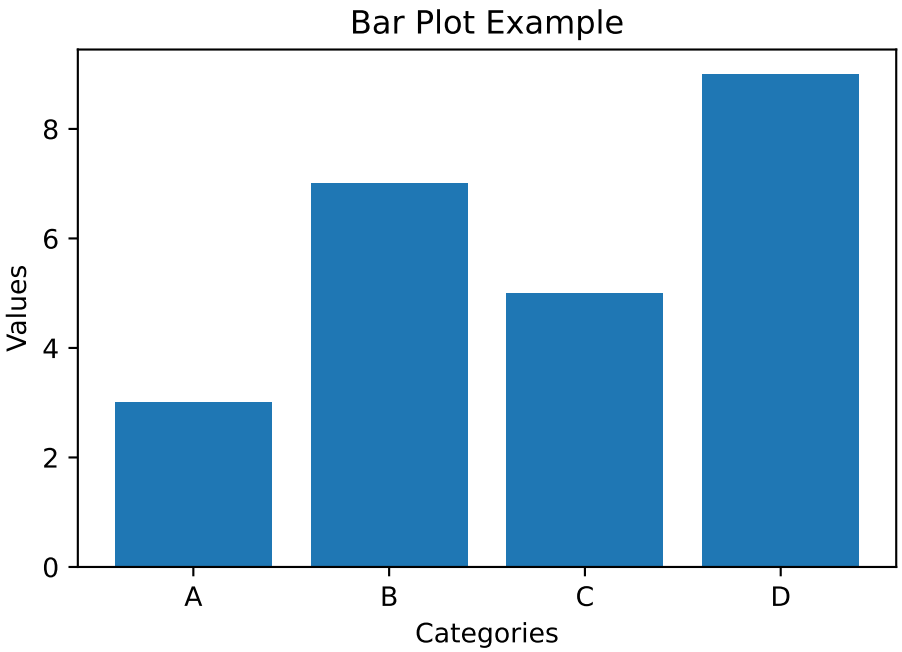


3. **Bar Plot**:

```
import matplotlib.pyplot as plt

# Categories and values
categories = ['A', 'B', 'C', 'D']
values = [3, 7, 5, 9]

# Create a bar plot
plt.bar(categories, values)
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Plot Example")
plt.show()
```

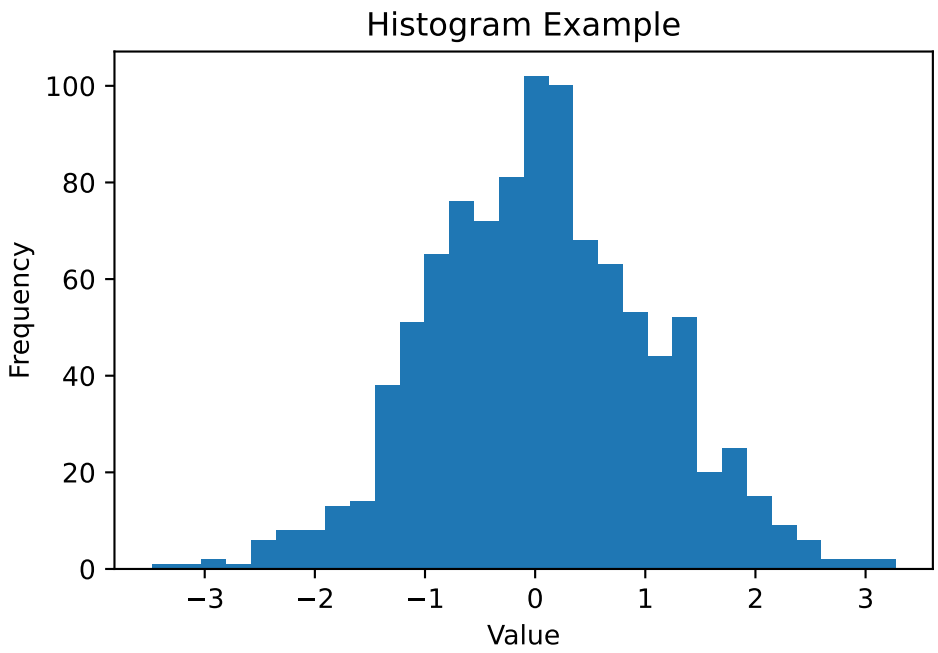


4. **Histogram**:

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

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

# Create a histogram
plt.hist(data, bins=30)
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Histogram Example")
plt.show()
```



2. `import numpy as np`

- **`numpy`**: This is a powerful library used for numerical computing in Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
- **`as np`**: Like `plt`, this is an alias for `numpy`, allowing you to refer to `numpy` functions by using `np`, which is a common convention in Python.

What It Does:

- `numpy` is widely used for array manipulations, mathematical operations, and working with large datasets in Python.
- `np` is used to call `numpy` functions, such as `np.array()`, `np.linspace()`, `np.sin()`, and many more.

Basic Syntax:

```
import numpy as np
```

Key Functions and Examples:

Creating Arrays:

```
import numpy as np

# 1D array
arr = np.array([1, 2, 3, 4, 5])
print(arr)

# 2D array
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
```

```
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
```

Generating Sequences:

```
# Evenly spaced numbers between 0 and 10
x = np.linspace(0, 10, 5)
print(x)

# Numbers between 0 and 10 with a step size of 2
y = np.arange(0, 10, 2)
print(y)
```

```
[ 0.   2.5  5.   7.5 10. ]
[0 2 4 6 8]
```

Basic Operations:

```
# Element-wise operations
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)  # [5 7 9]
print(a * b)  # [ 4 10 18]
print(a ** 2)  # [1 4 9]
```

```
[5 7 9]
[ 4 10 18]
[1 4 9]
```

Matrix Multiplication:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

# Matrix multiplication
result = np.dot(a, b)
print(result)
```

```
[[19 22]
 [43 50]]
```

Statistical Functions:

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

mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)

print(f"Mean: {mean}, Median: {median}, Std Dev: {std_dev}")
```

```
Mean: 0.03137260594605054, Median: 0.006293509315410843, Std Dev: 0.9889724992893345
```

Note

The function **np.linspace()** is a part of the **numpy** library in Python. It is used to create evenly spaced numbers over a specified range. This is very useful in plotting functions or when you need a specific number of points between two numbers.

Syntax:

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
```

Parameters:

1. **`start`**: The starting value of the sequence.
2. **`stop`**: The end value of the sequence.
3. **`num`** (optional): The number of evenly spaced samples to generate. Default is 50.
4. **`endpoint`** (optional): If `True` (default), the `stop` value is included in the sequence. If `False`, it is not.
5. **`retstep`** (optional): If `True`, the function will return both the array of values and the spacing between them. Default is `False`.
6. **`dtype`** (optional): The data type of the output array. If not provided, it will infer the type from the input.
7. **`axis`** (optional): The axis in the result to store the samples. Default is `0`.

Returns:

- An array of `num` evenly spaced values between `start` and `stop`.

Example 1: Basic Usage

```
import numpy as np

# Generate 5 numbers evenly spaced between 0 and 10
arr = np.linspace(0, 10, 5)
print(arr)
```

```
[ 0.   2.5  5.   7.5 10. ]
```

This generates 5 evenly spaced numbers between `0` and `10`.

Example 2: Excluding the Endpoint

```
import numpy as np

# Generate 5 numbers between 0 and 10, excluding the endpoint
arr = np.linspace(0, 10, 5, endpoint=False)
print(arr)
```

```
[0. 2. 4. 6. 8.]
```

In this case, the `stop` value (`10`) is not included in the sequence.

Example 3: Return Step Size

```
import numpy as np

# Generate numbers and return the step size
arr, step = np.linspace(0, 10, 5, retstep=True)
print(f"Array: {arr}")
print(f"Step size: {step}")
```

```
Array: [ 0.   2.5  5.   7.5 10. ]
Step size: 2.5
```

The `retstep=True` argument returns both the array and the step size (spacing) between each consecutive element.

Example 4: Using `np.linspace()` for Plotting

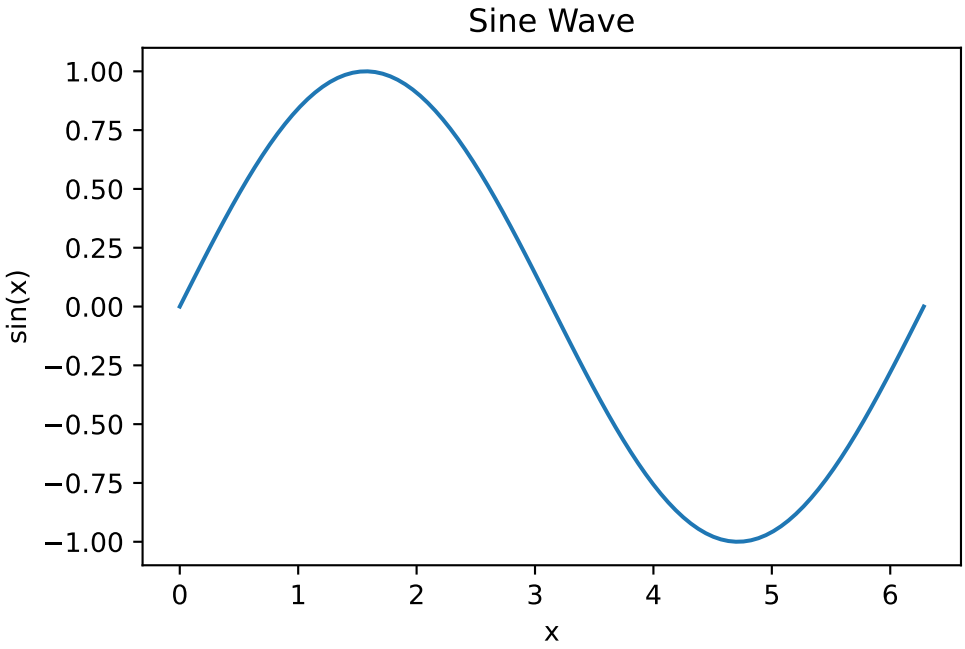
You can use `np.linspace()` to generate values for the x-axis when plotting functions, for example, sine waves.

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

# Generate 100 points between 0 and 2
x = np.linspace(0, 2 * np.pi, 100)
```

```
y = np.sin(x)

plt.plot(x, y)
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()
```



This code generates a smooth sine wave by creating 100 points evenly spaced between 0 and 2 .

Key Use Cases:

- Creating ranges for plotting graphs.
- Generating test data.
- Evaluating functions at evenly spaced points.

3. `pandas`

Overview:

- **pandas** is a powerful library for data manipulation and analysis in Python. It is widely used for working with structured data such as tables (similar to Excel or SQL).
- Provides data structures like **Series** and **DataFrame** to handle tabular data.

```
import pandas as pd
```

Key Functions and Examples:

1. **Creating a DataFrame**:

```
import pandas as pd

# Create a simple DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Paris', 'Berlin']}

df = pd.DataFrame(data)
print(df)
```

```
      Name  Age      City
0    Alice   25  New York
1      Bob   30     Paris
2  Charlie   35    Berlin
```

2. **Reading Data from a CSV File**:

```
# Assuming you have a CSV file "data.csv"
df = pd.read_csv('data.csv')
print(df.head())  # Print the first 5 rows of the DataFrame
```

3. **Basic DataFrame Operations**:

```
# Access a specific column
print(df['Name'])

# Filter rows based on a condition
older_than_30 = df[df['Age'] > 30]
print(older_than_30)

# Add a new column
df['Salary'] = [50000, 60000, 70000]
print(df)
```

4. **Statistical Summary**:

```
# Summary statistics
print(df.describe())
```

5. **Handling Missing Data**:

```
import pandas as pd
import numpy as np

# Create a DataFrame with missing values
data = {'Name': ['Alice', 'Bob', np.nan],
        'Age': [25, np.nan, 35],
        'City': ['New York', 'Paris', np.nan]}

df = pd.DataFrame(data)

# Fill missing values
df_filled = df.fillna('Unknown')
print(df_filled)

# Drop rows with missing values
df_dropped = df.dropna()
print(df_dropped)
```

```
      Name      Age      City
0    Alice     25.0  New York
1      Bob  Unknown     Paris
2  Unknown     35.0   Unknown
    Name   Age      City
0  Alice  25.0  New York
```

Summary of Usage:

- **`matplotlib.pyplot as plt`**: Used for creating a variety of plots like line plots, scatter plots, bar plots, histograms, etc.
- **`numpy as np`**: Provides powerful array manipulation and numerical operations like matrix multiplication, array slicing, mathematical functions, etc.
- **`pandas as pd`**: Facilitates data analysis by providing tools to work with tabular data (DataFrames), filtering, aggregation, and handling missing data.

These libraries are often used together in data science, machine learning, and scientific computing workflows to analyze data, create models, and visualize results.


thank you