UNIT = 5

Introduction to NumPy

NumPy (Numerical Python) is a powerful Python library used for **numerical and scientific computing**. It provides support for **large, multi-dimensional arrays** and **mathematical functions** to operate on these arrays efficiently.

Why NumPy is important

Normal Python lists are slow and limited for heavy calculations.
NumPy solves this by providing:

- Fast execution (written in C internally)
- Easy handling of large datasets
- Built-in mathematical functions

That's why NumPy is widely used in:

- Data Science
- Machine Learning
- AI
- Scientific Computing
- Image Processing

Core feature: NumPy Array

The main object in NumPy is the **ndarray (N-dimensional array)`** .

Example (Python list vs NumPy array)

Example (Python list vs NumPy array)

python Copy code

```python
# Python list
a = [1, 2, 3, 4]
b = [5, 6, 7, 8]

# This will NOT work as expected
print(a + b)   # [1,2,3,4,5,6,7,8]
```

python Copy code

```python
import numpy as np

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

print(a + b)   # [ 6  8 10 12 ]
```

✅ NumPy performs **element-wise operations** automatically.

Key features of NumPy

1. Fast performance 🚀

- Uses optimized C code
- Much faster than Python loops

2. Multidimensional arrays

You can create:

- 1D arrays (vectors)
- 2D arrays (matrices)
- 3D+ arrays (tensors)

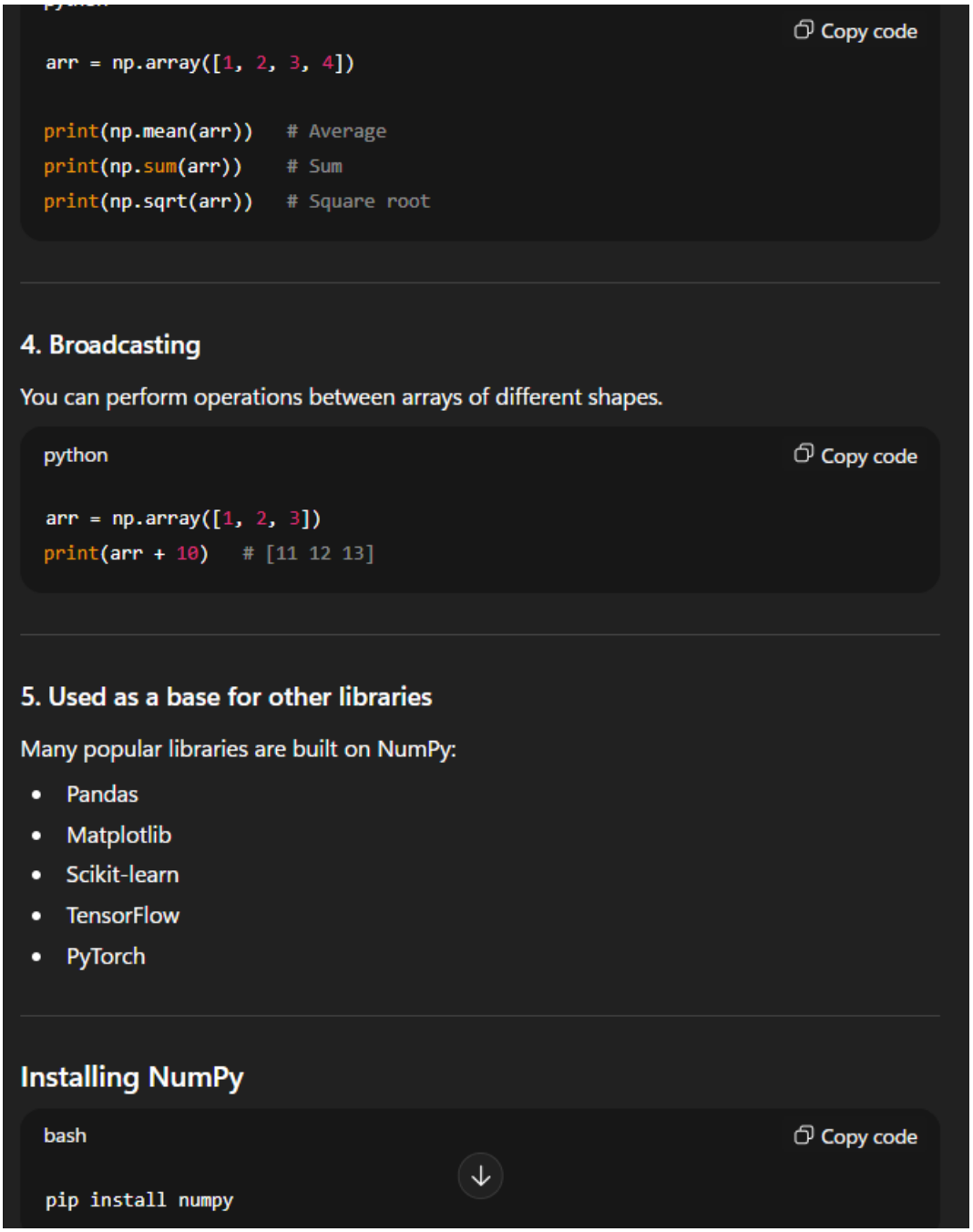
```
arr = np.array([[1, 2], [3, 4]])
```

3. Mathematical operations

NumPy supports:

- Addition, subtraction, multiplication, division
- Trigonometry

- Statistics



Simple real-life example

Imagine you have marks of students and want to calculate average:

```python
import numpy as np

marks = np.array([78, 85, 90, 88, 76])
average = np.mean(marks)

print("Average marks:", average)
```

Introduction to NumPy

NumPy (Numerical Python) is a powerful Python library used for **numerical and scientific computing**. It provides support for **large, multi-dimensional arrays** and **mathematical functions** to operate on these arrays efficiently.

Unlike Python lists, NumPy arrays:

- Are **faster**
- Consume **less memory**
- Support **vectorized operations**

NumPy is widely used in:

- Data Science
- Machine Learning
- Scientific calculations
- Image processing

The core object of NumPy is the **ndarray (N-dimensional array)**.

Creating Arrays in NumPy

NumPy arrays can be created in **two main ways**:

1. Using **existing arrays (lists/tuples)**
2. Using **scalars and built-in functions**

---

1. Creating Arrays Using Existing Arrays

We can create NumPy arrays from:

- Python lists
- Tuples

1D Array (Vector)

```python
import numpy as np

arr = np.array([10, 20, 30, 40])
print(arr)
```

Output:

```csharp
[10 20 30 40]
```

2D Array (Matrix)

```python
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print(arr)
```

Output:

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

3D Array

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

Key Points

- All elements must be of the **same data type**
- NumPy automatically converts data types if needed
- Arrays support **element-wise operation**

Key Points

- All elements must be of the **same data type**
- NumPy automatically converts data types if needed
- Arrays support **element-wise operations**

2. Creating Arrays Using Scalars

A **scalar** is a single value (number). NumPy provides functions to create arrays using scalars.

Constant Value Array

python Copy code

```python
arr = np.full(5, 7)
print(arr)
```

Output:

csharp Copy code

```
[7 7 7 7 7]
```

Range of Values

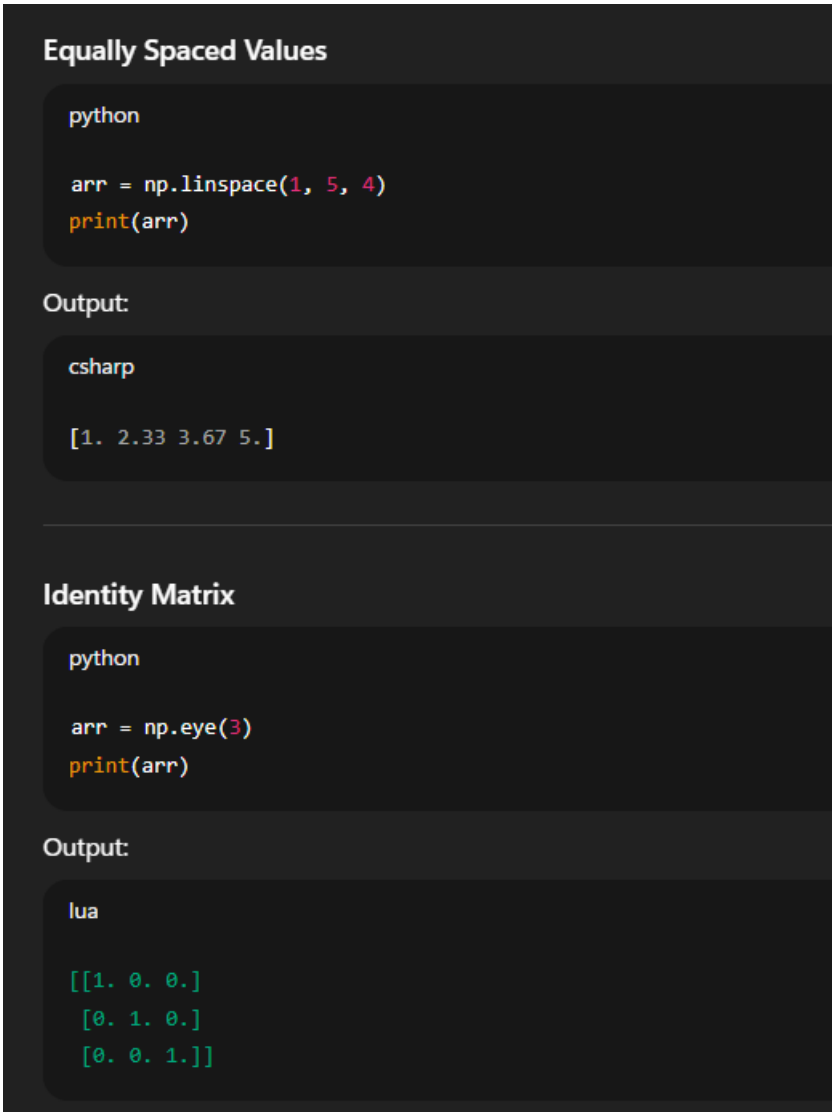
python Copy code

```python
arr = np.arange(1, 10, 2)
print(arr)
```

Output:

csharp Copy code

```
[1 3 5 7 9]
```



Introduction to Pandas

Pandas is a Python library built on **NumPy** that is used for **data manipulation and analysis**.

While NumPy handles **numerical arrays**, Pandas handles **structured data** like:

- Tables
- CSV files
- Excel sheets
- Databases

Main Data Structures in Pandas

1. Series

- One-dimensional labeled array
- Similar to a column in a table

python

```
import pandas as pd

s = pd.Series([10, 20, 30, 40])
print(s)
```

2. DataFrame

- Two-dimensional table (rows & columns)
- Similar to an Excel sheet

python

```
data = {
    "Name": ["Amit", "Rahul", "Neha"],
    "Marks": [85, 90, 88]
}

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

Why Pandas is Used

- Easy data cleaning
- Fast data filtering
- Handling missing values
- Works smoothly with NumPy

Relationship Between NumPy and Pandas

NumPy	Pandas
Works with arrays	Works with tables
Faster for math	Better for data analysis
No labels	Labeled data
Low-level operations	High-level operations

What is Pandas?

Pandas is an **open-source Python library** used for **data manipulation and data analysis**.
It provides **easy-to-use data structures** that help us store, clean, modify, and analyze **structured data** efficiently.

Pandas is built **on top of NumPy**, which means it uses NumPy internally for fast numerical operations, but adds **labels (rows and columns)** to make data easier to understand and work with.

👉 In simple words:
Pandas helps us work with data in table form (like Excel) using Python.

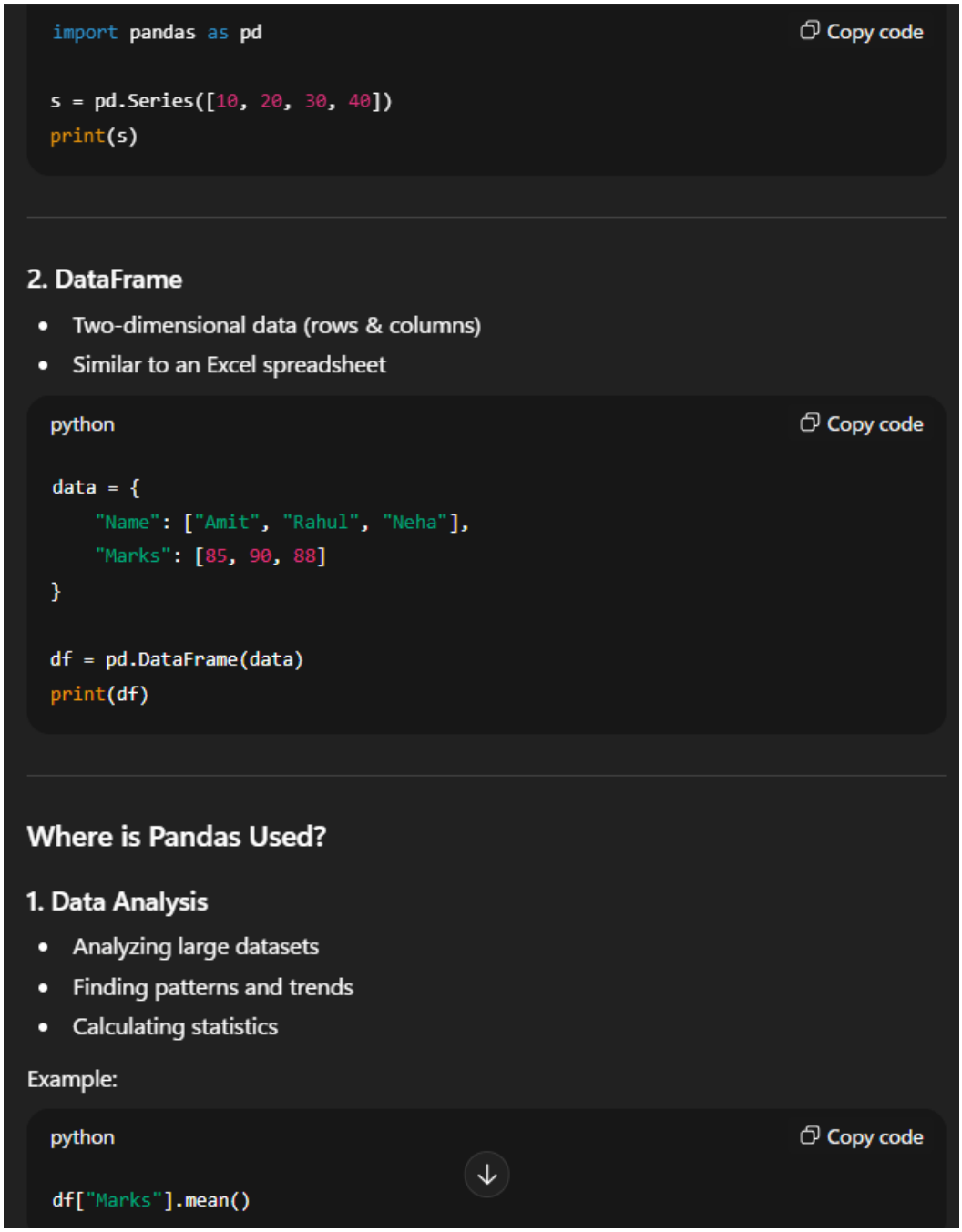
Key Features of Pandas

- Works with **tabular data**
- Handles **missing values**
- Easy **filtering, sorting, and grouping**
- Fast and efficient
- Supports multiple data formats (CSV, Excel, SQL, JSON)

Main Data Structures in Pandas

1. Series

- One-dimensional labeled data
- Similar to a single column in a table



2. Data Cleaning

- Removing duplicate data
- Filling missing values
- Correcting wrong data

Example:

df.fillna(0)

---

3. Data Science & Machine Learning

- Preparing data before training models
- Feature selection and transformation

Used with:

- Scikit-learn
- TensorFlow
- PyTorch

---

4. Business & Financial Analysis

- Sales reports
- Profit/loss analysis
- Market trend analysis

---

5. Working with Files

Pandas can read and write:

- CSV files
- Excel files
- JSON data
- SQL databases

```
df = pd.read_csv("data.csv")
```

---

6. Academic & Research Work

- Scientific data analysis
- Research statistics
- Survey data processing

Difference Between NumPy and Pandas

NumPy	Pandas
Works with arrays	Works with tables
Faster for math	Better for data analysis
No labels	Labeled rows & columns
Numerical data only	Mixed data types

Series in Pandas

What is a Series?

A **Series** in Pandas is a **one-dimensional labeled array** that can store:

- Integers
- Floats
- Strings
- Boolean values

Each value in a Series has an **index (label)**.

👉 In simple words:
A Pandas Series is like a single column of a table with labels.

Creating a Series

1. Series from a list

```python
import pandas as pd

s = pd.Series([10, 20, 30, 40])
print(s)
```

Output:

```go
0    10
1    20
2    30
3    40
dtype: int64
```

2. Series with custom index

```python
s = pd.Series([85, 90, 88], index=["Amit", "Rahul", "Neha"])
print(s)
```

Output:

```nginx
Amit     85
Rahul    90
Neha     88
```

3. Series from dictionary

python Copy code

```python
data = {"Math": 80, "Science": 85, "English": 78}
s = pd.Series(data)
print(s)
```

Accessing Series Elements

python Copy code

```python
print(s["Math"])
print(s[0])
```

Operations on Series

python Copy code

```python
s = pd.Series([10, 20, 30])

print(s + 5)
print(s * 2)
print(s.mean())
```

Features of Pandas Series

- One-dimensional
- Indexed data
- Supports mathematical operations
- Handles missing values (NaN)
- Built on NumPy

Matplotlib

What is Matplotlib?

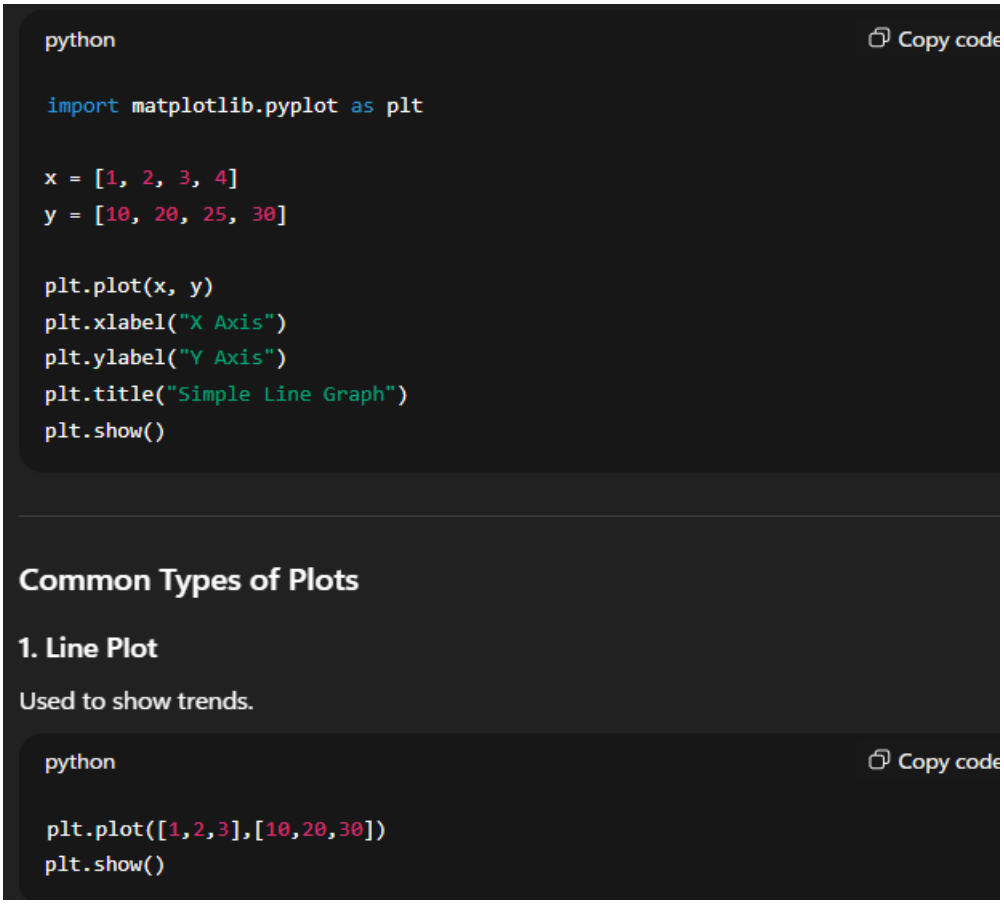
Matplotlib is a **Python plotting library** used for **data visualization**.
It helps us represent data using **graphs and charts**.

👉 In simple words:
Matplotlib is used to draw graphs in Python.

Where is Matplotlib Used?

- Data analysis
- Data science
- Machine learning
- Academic projects
- Business reports

Basic Plot Using Matplotlib



2. Bar Chart

Used for comparison.

python Copy code

```python
plt.bar(["A", "B", "C"], [20, 35, 30])
plt.show()
```

3. Histogram

Used for data distribution.

python Copy code

```python
plt.hist([1,2,2,3,3,3,4])
plt.show()
```

4. Scatter Plot

Used to show relationship between data.

python Copy code

```python
plt.scatter([1,2,3],[4,5,6])
plt.show()
```

Difference Between Pandas Series and Matplotlib

Pandas Series	**Matplotlib**
Data structure	Visualization tool
Stores data	Displays data
Built on NumPy	Built on NumPy
Used for analysis	Used for plotting

Python for Data Visualization

What is Data Visualization?

Data Visualization is the process of **representing data graphically** using charts, graphs, and plots so that patterns, trends, and insights are easy to understand.

👉 Instead of reading numbers, we **see the data visually**.

Why Python for Data Visualization?

Python is widely used for data visualization because:

- Easy to learn and use
- Powerful visualization libraries
- Works well with data analysis tools
- Used in industry and academics

Popular Python Libraries for Data Visualization

1. Matplotlib

- Basic and most popular library
- Used for simple to complex plots

2. Seaborn

- Built on Matplotlib
- Used for statistical visualization
- Better design and themes

3. Pandas (Built-in Plotting)

- Quick plotting from Series & DataFrames
- Uses Matplotlib internally

4. Plotly

- Interactive graphs
- Used in dashboards and web apps

---

1. Matplotlib for Data Visualization

Line Plot

```
import matplotlib.pyplot as plt


x = [1, 2, 3, 4]

y = [10, 20, 25, 30]


plt.plot(x, y)

plt.xlabel("X Axis")

plt.ylabel("Y Axis")

plt.title("Line Plot")

plt.show()
```

Used for: trends over time

Bar Chart

```
plt.bar(["A", "B", "C"], [20, 35, 30])

plt.title("Bar Chart")

plt.show()
```

Used for: comparison between categories

Histogram

```
plt.hist([10, 20, 20, 30, 30, 30])

plt.title("Histogram")

plt.show()
```

Used for: data distribution

Scatter Plot

```
plt.scatter([1, 2, 3], [4, 5, 6])
```

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

plt.show()
```

Used for: relationship between variables

---

2. Pandas for Visualization

Pandas makes visualization easier when data is in **Series or DataFrame** form.

Series Plot

```
import pandas as pd


s = pd.Series([10, 20, 30, 40])

s.plot()

plt.show()
```

DataFrame Plot

```
data = {
   "Math": [80, 85, 90],
   "Science": [75, 88, 92]
}


df = pd.DataFrame(data)

df.plot()

plt.show()
```

---

3. Seaborn for Visualization

Seaborn provides **attractive and statistical plots**.

```
import seaborn as sns

import matplotlib.pyplot as plt
```

```
sns.barplot(x=["A","B","C"], y=[10,20,30])

plt.show()
```

---

4. Plotly for Interactive Visualization

Plotly allows zooming, hovering, and interaction.

```
import plotly.express as px


df = px.data.iris()

fig = px.scatter(df, x="sepal_width", y="sepal_length")

fig.show()
```

Applications of Python Data Visualization

- Data Science
- Machine Learning
- Business analytics
- Academic research
- Dashboard creation

Advantages of Python Visualization

- Easy to learn
- High flexibility
- Supports large datasets
- Strong community support