UNIT-2

1 What is NumPy, and why is it important in Python for numerical computations?.

NumPy (Numerical Python) is a Python library used for fast numerical computations. It provides an efficient way to store and manipulate large multi-dimensional arrays and offers a wide range of mathematical functions to operate on them.

Why NumPy is Important

- 1. **Speed** Operations are implemented in optimized C code, making them much faster than Python loops.
- 2. **Memory Efficiency** Stores data in contiguous memory blocks, unlike Python lists.
- 3. **Vectorization** Allows operations on entire arrays without writing explicit loops.
- 4. **Mathematical Power** Supports linear algebra, Fourier transforms, statistics, and more.
- 5. **Foundation for Data Science** Libraries like Pandas, SciPy, scikit-learn, and TensorFlow are built on top of NumPy.

Here, a + b is computed without loops, making it faster and cleaner.

Key Features of NumPy

1. N-Dimensional Array Object (ndarray)

- Core data structure for storing and manipulating large datasets efficiently.
- Much faster than Python's built-in lists for numerical operations.

2. Vectorized Operations

 Eliminates the need for explicit loops; operations are applied element-wise to entire arrays at once.

3. Mathematical & Statistical Functions

o Functions like sum(), mean(), std(), dot(), sin(), exp() are built in and optimized.

4. Linear Algebra Support

o Matrix multiplication, determinants, eigenvalues, etc.

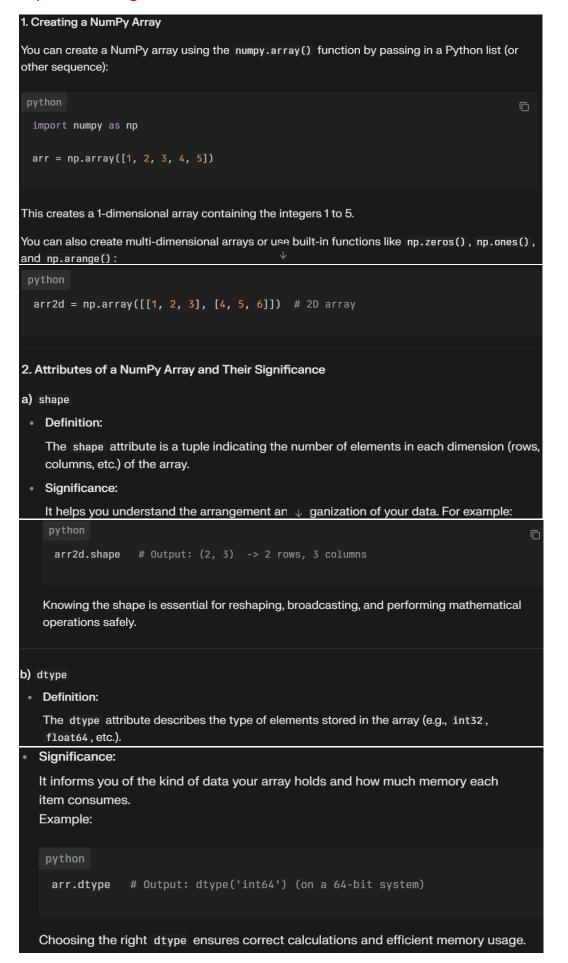
5. Random Number Generation

Useful for simulations, machine learning, and testing.

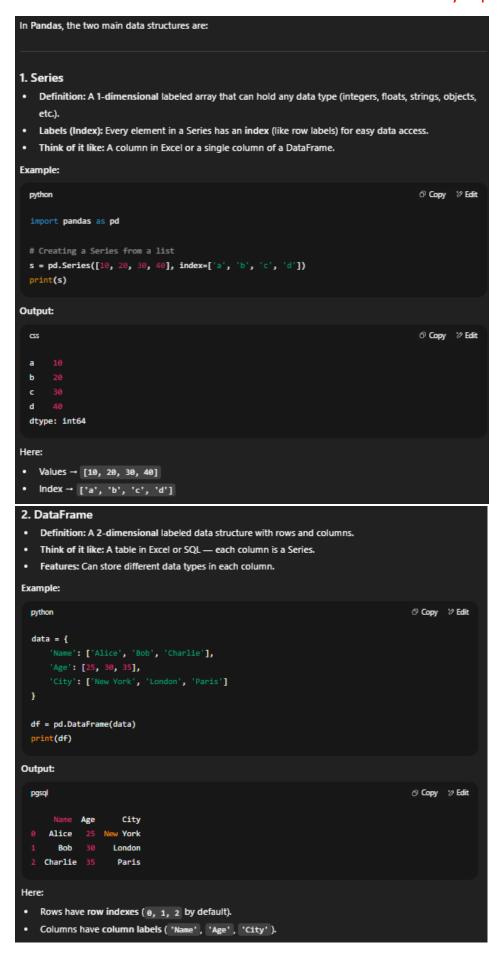
6. Interoperability

• Works seamlessly with libraries like Pandas, Matplotlib, SciPy, and TensorFlow.

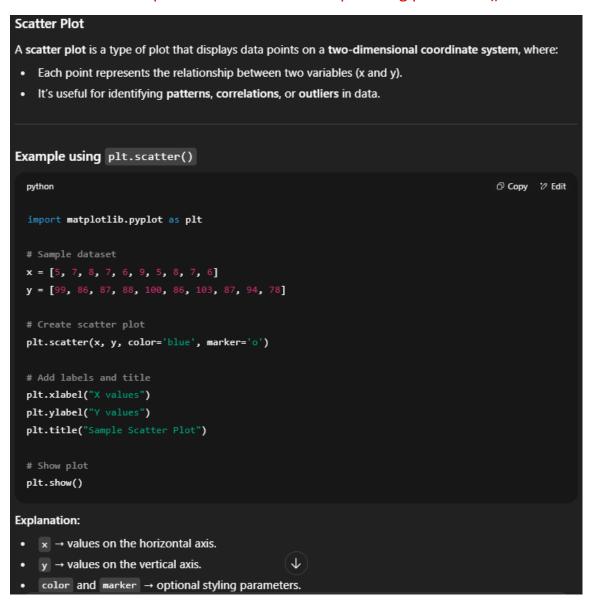
2. How do you create a NumPy array? Mention any two attributes of a NumPy array and explain their significance.



3. What are the two main data structures in Pandas? Briefly explain each with an example.



4. What is a scatter plot? Write a short example using plt.scatter() to visualize a dataset.



5. How does NumPy sort arrays? Differentiate between np.sort() and ndarray.sort() with one example.

NumPy Sorting Overview

NumPy sorts arrays using an efficient sorting algorithm (by default quicksort, but you can choose 'mergesort', 'heapsort', or 'stable').

Sorting can be along a specific axis or the entire flattened array.

Difference Between np.sort() and ndarray.sort()

Feature np.sort() ndarray.sort()

Type Function in the NumPy module Method of a NumPy array object

Returns New sorted array (original array unchanged) Sorts in place (modifies the original array)

 $\begin{tabular}{ll} U sage & np.sort(array, axis=-1) & array.sort(axis=-1) \end{tabular}$

Mutability Does not alter the original array

Alters the original array



6.Explain how aggregation functions like min(), max(), mean() and Aggregating Along an Axis are used in NumPy. Provide one code example.

Aggregation Functions in NumPy

Aggregation functions in NumPy perform calculations that **summarize data** from an array, such as finding the minimum, maximum, or average.

Common ones include:

- min() → smallest element
- max() → largest element
- mean() → average value

Aggregating Along an Axis

- axis=None (default) → operation is applied to the **entire array** (all elements).
- axis=0 → operation is applied column-wise (down each column).
- axis=1 → operation is applied row-wise (across each row).

```
Example
 python
                                                                                    ○ Copy ७ Edit
  import numpy as np
 # 2D array
 arr = np.array([[10, 20, 30],
                 [40, 50, 60]])
 # Aggregating without axis (entire array)
 print("Min:", np.min(arr)) # 10
 print("Max:", np.max(arr))
                                # 60
 print("Mean:", np.mean(arr)) # 35.0
 # Aggregating along axis 0 (column-wise)
 print("Column-wise min:", np.min(arr, axis=0)) # [10 20 30]
 print("Column-wise max:", np.max(arr, axis=0)) # [40 50 60]
 print("Column-wise mean:", np.mean(arr, axis=0))# [25. 35. 45.]
 # Aggregating along axis 1 (row-wise)
 print("Row-wise min:", np.min(arr, axis=1))
                                                 # [10 40]
 print("Row-wise max:", np.max(arr, axis=1))
                                                # [30 60]
 print("Row-wise mean:", np.mean(arr, axis=1)) # [20. 50.]
```

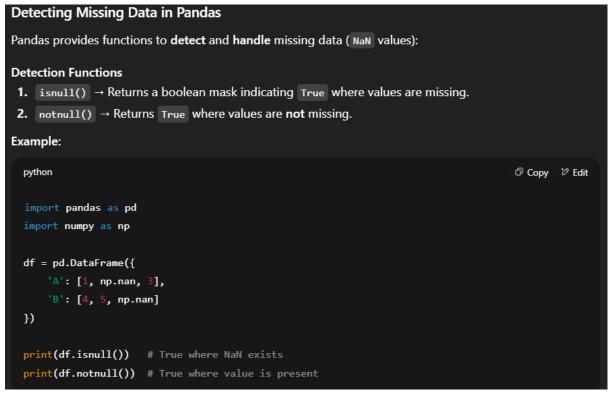
7. Write a Python program using Matplotlib to create a line plot for a dataset. Add labels, a title, and a legend. Explain each component used in the plot.

```
Python Program
  python
                                                                                  import matplotlib.pyplot as plt
  # Sample dataset
  x = [1, 2, 3, 4, 5]
  y2 = [1, 3, 5, 7, 9]
  # Create line plots
  plt.plot(x, y1, label='Line 1', color='blue', marker='o')
  plt.plot(x, y2, label='Line 2', color='red', linestyle='--', marker='x')
  plt.xlabel("X-axis Label")
  plt.ylabel("Y-axis Label")
  # Add title
  plt.title("Sample Line Plot")
  # Add legend
  plt.legend()
  # Display plot
  plt.show()
```

Explanation of Components

- 1. import matplotlib.pyplot as plt
 - o Imports Matplotlib's plotting module and gives it the alias plt.
- 2. **x**, **y**1, **y**2
 - o Lists containing the dataset for the x-axis and y-axis values.
- 3. plt.plot()
 - o Plots the data as a line graph.
 - o label \rightarrow name shown in the legend.
 - o color \rightarrow sets the line color.
 - o marker \rightarrow marks data points ('o', 'x', '^', etc.).
 - o linestyle \rightarrow style of the line ('-' solid, '--' dashed).
- 4. plt.xlabel() and plt.ylabel()
 - o Adds descriptive labels to the X-axis and Y-axis.
- 5. plt.title()
 - o Adds a title to the entire plot.
- 6. plt.legend()
 - o Displays a legend to differentiate multiple plotted lines using their label.
- 7. plt.show()
 - o Renders and displays the plot.

8. What functions does Pandas provide to detect and handle missing data? Mention two methods for dealing with missing values.



```
Two Methods to Handle Missing Values

1. dropna() - Remove missing values (rows or columns).

python

python

df_dropped = df.dropna()  # Drop rows with NaN

df_dropped_cols = df.dropna(axis=1)  # Drop columns with NaN

2. fillna() - Replace missing values with a specified value or method.

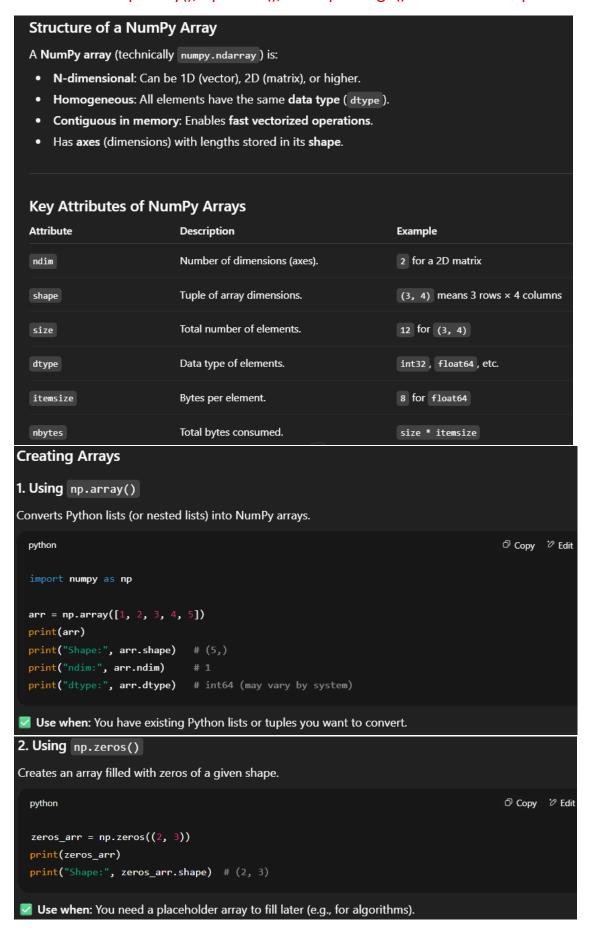
python

df_filled = df.fillna(0)  # Replace NaN with 0

df_forward = df.fillna(method='ffill')  # Forward fill (use previous value)
```

☑ Summary Table:		
Purpose	Function	
Detect missing	<pre>isnull(), notnull()</pre>	
Remove missing	dropna()	
Replace missing	fillna()	

9.Explain the structure and key attributes of NumPy arrays. How can arrays be created using functions like np.array(), np.zeros(), and np.arange()? Provide examples and explain each.



```
3. Using np.arange()

Creates evenly spaced values within a given range (like Python's range(), but returns an array).

python

© Copy ** Edit

range_arr = np.arange(0, 10, 2) # Start=0, Stop=10, Step=2

print(range_arr)

Use when: You need sequences for indexing, plotting, or numeric ranges.

Extra Examples

python

© Copy ** Edit

# Array of ones
ones_arr = np.ones((3, 2))

# Array with evenly spaced values
linspace_arr = np.linspace(0, 1, 5) # 5 numbers between 0 and 1

Summary Table
```

Summary Table		
Function	Purpose	Example Output
np.array()	Convert list/tuple to array	[1 2 3]
np.zeros()	Create zero-filled array	[[0. 0.], [0. 0.]]
np.arange()	Create range of values	[0 2 4 6 8]
np.ones()	Create one-filled array	[[1. 1.], [1. 1.]]
np.linspace()	Evenly spaced numbers	[0. 0.25 0.5 0.75 1.]

10.Describe Series and DataFrame objects in Pandas. What are their key characteristics? How are they created? Compare and contrast Series and DataFrame with relevant code examples

1. Series in Pandas

Definition

A Series is a 1-dimensional labeled array that can hold any data type — integers, floats, strings, Python objects, etc.

Think of it like a single column in an Excel sheet.

Key Characteristics

- 1D data with labels (index).
- Can hold different data types (but usually consistent).
- Index labels allow fast lookups.
- Built on top of NumPy arrays.

Creating a Series

```
python

import pandas as pd

# Creating from a list
s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(s)
```

Output:

```
css

a 10
b 20
c 30
d 40
dtype: int64
```

2. DataFrame in Pandas

Definition

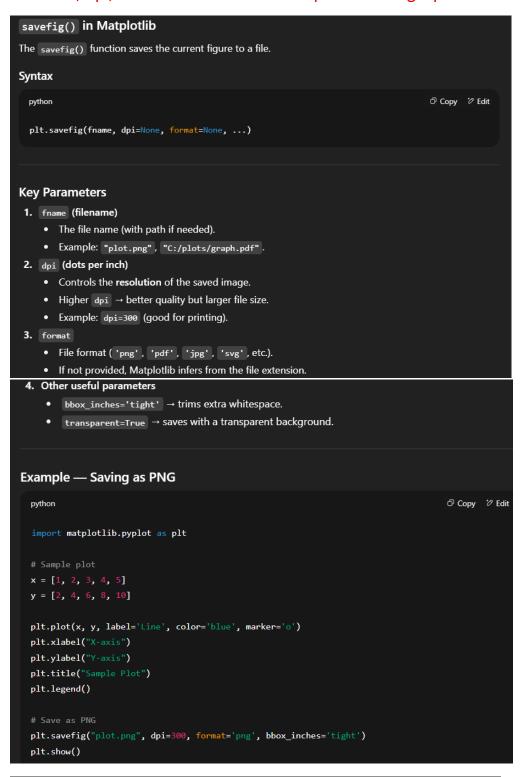
A **DataFrame** is a **2-dimensional labeled data structure** with rows and columns — essentially a **table**. Think of it like an **Excel spreadsheet** or a **SQL table**.

Key Characteristics

- 2D data with row and column labels.
- Columns can hold different data types.
- Each column is essentially a Pandas Series.
- Flexible for data manipulation and analysis.

```
Creating a DataFrame
                                                                                         ☐ Copy 🎖 Edit
  python
  # Creating from a dictionary
  data = {
      'Name': ['Alice', 'Bob', 'Charlie'],
  df = pd.DataFrame(data)
  print(df)
Output:
                                                                                         pgsql
       Name Age
                       City
      Alice
              25 New York
        Bob
                     London
    Charlie
                      Paris
3. Series vs DataFrame — Comparison
                                                                                                     ර
Feature
                        Series
                                                           DataFrame
Dimensions
                        1D
                                                           2D
Structure
                        Single column
                                                           Multiple columns
Index
                        Single index (row labels)
                                                           Row index + column labels
                        Homogeneous (one type per Series)
                                                           Heterogeneous (each column can be a different type)
Data Types
                        Column in Excel
                                                           Table in Excel
Analogy
Created from
                        List, array, dict, scalar
                                                           Dict of lists, 2D array, list of dicts
4. Code Example — Side-by-Side
                                                                                          ☐ Copy 🍪 Edit
  python
  # Series example
  s = pd.Series([100, 200, 300], index=['X', 'Y', 'Z'])
  print("Series:\n", s)
  df = pd.DataFrame({
      'Product': ['A', 'B', 'C'],
      'Price': [100, 200, 300]
  })
  print("\nDataFrame:\n", df)
Output:
 makefile
                                                                             ☐ Copy 🎖 Edit
 Series:
      200
      300
 dtype: int64
 DataFrame:
   Product Price
             100
        В
             200
             300
```

11. How do you save a plot using savefig() in Matplotlib? Discuss the parameters such as filename, dpi, and format. Provide examples of saving a plot as PNG and PDF.



```
python

plt.plot(x, y, color='red', marker='x')
plt.title("PDF Example")

# Save as PDF
plt.savefig("plot.pdf", dpi=300, format='pdf', bbox_inches='tight')
plt.show()
```

12. What are the key components of a plot in Matplotlib? Discuss general tips for making effective visualizations such as using titles, labels, legends, and styles. Provide examples.

1. Key Components of a Matplotlib Plot When you create a plot in Matplotlib, it usually consists of these core parts: Component Description The entire plotting area (can contain multiple subplots). **Figure** Axes The actual area where data is plotted (inside the figure). Title Describes what the plot is about. Axis Labels Names for the X and Y axes to clarify meaning. **Ticks & Tick Labels** Marks along the axes showing measurement intervals. Legend Explains what each line, color, or marker represents. Grid Helps visually align data points. Style The overall look (colors, line styles, background themes).

2. General Tips for Effective Visualizations

- Use a descriptive title → helps the viewer instantly understand the purpose of the plot.
- Label axes clearly → tell exactly what the X and Y values represent.
- Include a legend if there's more than one data series.
- Choose colors and markers wisely → avoid too many, keep contrast high.
- Use grid lines for easier value reading.
- Maintain aspect ratio and figure size for clarity.
- Use consistent style if multiple plots are shown together.

```
3. Example — A Well-Formatted Plot
  python
                                                                                    ☐ Copy % Edit
  import matplotlib.pyplot as plt
  # Sample data
  y1 = [2, 4, 6, 8, 10]
  y2 = [1, 4, 9, 16, 25]
  plt.figure(figsize=(8, 5))
  plt.plot(x, y1, label='Linear Growth', color='blue', marker='o', linestyle='-')
  plt.plot(x, y2, label='Quadratic Growth', color='red', marker='s', linestyle='--')
  # Add title and labels
  plt.title("Linear vs Quadratic Growth", fontsize=16, fontweight='bold')
  plt.xlabel("X Values", fontsize=12)
  plt.ylabel("Y Values", fontsize=12)
  plt.grid(True, linestyle=':', alpha=0.7)
  plt.legend()
  # Apply a style
  plt.style.use('seaborn-v0_8')
  plt.show()
```

Output Explanation Title: "Linear vs Quadratic Growth" — tells us what we're looking at. **X-axis label**: "X Values" — explains what the horizontal values mean. Y-axis label: "Y Values" — explains what the vertical values mean. Legend: Helps distinguish between the blue linear line and the red quadratic curve. Grid: Dotted lines make it easier to read values. Style (seaborn): Gives a cleaner look with a light background. Quick Style Tip: Matplotlib supports many built-in styles: ☐ Copy 10 Edit python plt.style.available Try: python ☐ Copy 🎖 Edit plt.style.use('ggplot') # Clean red-grid style or python ☐ Copy 🎖 Edit plt.style.use('fivethirtyeight') # Popular blog-style charts