PYTHON LIBRARIES

Python Libraries: What & Why

- Python libraries **are collections of pre-written code (functions, modules, tools) that help you perform specific tasks more efficiently-think of them as ready-made building blocks for p rog ra mming .**
- **To use a library, you must first install it (e.g., via Anaconda Navigator) and then import it into your code using the** `import` **statement.**

Importing Libraries in Python

- Basic import: `import library_name` **(e.g.,** `import math`)
- With alias: `import library_name as alias`(e.g., `import numpy as np`)
- Import specific functions: `from library_name import function_name` **(e.g.,** `from math import sqrt`**)**
- Import all (discouraged): `from library_name import *`

Key Data Science Libraries

NumPy

- Purpose: **Efficiently handles large, multi-dimensional arrays and matrices, and provides fast mathematical operations.**
- Arrays: **Homogeneous (all elements same type), can be 1D, 2D (matrix), or 3D (cube-like).**
- Creating arrays: **Use** `np.array()`, `np.zeros()`, `np.ones()`.
- Operations: **Arithmetic (add, subtract, multiply, divide) are performed element-wise.**
- Indexing & Slicing: **Access specific elements, rows, columns, or subarrays using indices and slices (e.g.,** arr[^1_1] **for 2D arrays).**

Pandas

- Purpose: **Makes data analysis and manipulation easy, especially for tabular data.**
- Data Structures:
  - Series : **Like a single column (with labels called indices).**
  - Data Frame: **Like a table (rows and columns, similar to Excel).**

- Why Pandas over Excel: **Handles much larger datasets, automates tasks, integrates with other libraries, and is scriptable/reproducible.**
- Creating DataFrames: **With dictionaries, lists, or by loading files CSV, Excel, etc.).**
- Custom indices: **You can label rows and columns for easier access.**

Matplotlib

- Purpose: **Creates data visualizations (plots, charts, graphs).**
- Usage: **Import as** `import matplotlib.pyplot as plt.`
- Types of plots: **Line plots, bar charts, histograms, etc.**
- Customization: **Add labels, titles, colors, and more.**
- Integration: **Works well with Pandas and NumPy for visualizing their data.**

Seaborn

- Purpose: **Built on Matplotlib, makes attractive statistical plots with less code.**
- Features: **High-level interface, beautiful default styles, easy integration with Pandas Da t a Fra mes.**

Why Use These Libraries?

- Speed: **Operations are optimized for large datasets.**
- Convenience: **Simplify complex tasks.**
- Compatibility: **Work together seamlessly for the full data science workflow.**
- Community: **Lots of support and resources available.**

In short:

- **Install and import libraries to use their features.**
- **NumPy is for fast numerical operations on arrays.**
- **Pandas is for handling and analyzing tabular data.**
- **Matplotlib (and Seaborn) are for visualizing data.**
- **These tools are essential for data science and analysis in Python, making your work faster, easier, and more powerful.**

Pan d as

Pandas is a fundamental Python library for data analysis and manipulation. It allows you to work with large datasets efficiently, automate data tasks, and integrate seamlessly with other data science libraries. Pandas is especially useful when handling complex or large datasets, providing more flexibility and power than spreadsheet tools like Excel 1 .

Key Features of Pandas:

- **Handles large datasets without slowing down or crashing.**
- **Automates repetitive tasks with Python scripts.**
- **Cleans and preprocesses data (handling missing values, outliers, etc.).**
- **Reshapes, pivots, merges, and restructures data easily.**
- **Integrates with libraries like NumPy, Matplotlib, and Scikit-Learn 1 .**

Core Data Structures:

- Series: **Aone-dimensionallabeled array (like a single column in Excel).**
- DataFrame: **Atwo-dimensionallabeled data structure (like an Excel table).**

Example: Creating a Pandas Series

```
import pandas as pd
data = [1, 2, 3, 4, 5]
my_series = pd.Series(data)
print(my_series)
```

Output:

```
0 1 1 2 2 3
3 4
4 5
dtype: int64
```

You can also specify custom index labels:

```
my_series = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
print(my_series)
```

Output:

```
a 1 b 2 c 3 d
4 e 5
dtype: int64
```

Example: Creating a Pandas DataFrame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}           df           =
pd.DataFrame(data)
print(df)
```

Output:

```
      Name  Age          City
0    Alice   25      New York
1      Bob   30   Los Angeles
2  Charlie   35       Chicago
```

You can also set a custom index:

```
df = pd.DataFrame(data, index=['a', 'b', 'c'])
print(df)
```

Matplotlib

Matplotlib is a powerful Python library for creating visualizations such as plots, charts, and graphs. It is essential for data visualization, exploration, and presentation 1 .

Key Features of Matplotlib:

- **Visualizes data to reveal trends, patterns, and outliers.**
- **Supports a wide range of plot types: line, bar, histogram, pie, etc.**
- **Highly customizable (colors, labels, markers, fonts).**
- **Professional-quality figures for reports and presentations.**

Basic Usage Example: Line Plot

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
```

```
y = [10, 12, 5, 8, 6] plt.plot(x,
y) plt.xlabel("X-axis Label")
plt.ylabel("Y-axis     Label")
plt.show()
```

Bar Plot Example

```
import matplotlib.pyplot as plt
categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 12, 5, 8, 6]
plt.bar(categories, values, color='green')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Plot')
plt.show()
```

Histogram Example

```
import matplotlib.pyplot as plt
data = [10, 12, 5, 8, 6, 15, 18, 20, 22, 30, 35, 40]
plt.hist(data, bins=5, color='purple', edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```

Seab o r n

Seaborn is a high-level Python data visualization library built on top of Matplotlib. It is designed to make it easy to create attractive and informative statistical graphics ~~1~~.

Key Features of Seaborn:

- **Beginner-friendly, requiring less code for beautiful plots.**
- **Offers a variety of color palettes and themes.**
- **Integrates easily with Pandas DataFrames.**
- **Enhances Matplotlib by providing statistical plotting tools and better aesthetics.**

Importing Seaborn

```
import seaborn as sns
```

Seaborn is especially useful for exploring data and visualizing statistical relationships. It simplifies the process of creating complex visualizations and makes your plots look professional with minimal effort.

Summary Table: Pandas vs. Matplotlib vs. Seaborn

Arrays in NumPy: Detailed Explanation with Examples

What is an Array?

An array in NumPy is a powerful data structure that stores elements of the same type (homogeneous) and allows you to perform fast, efficient operations on large datasets. Arrays can be one-dimensional (like a list), two-dimensional (like a table or matrix), or even higher-dimensional (like a cube or more complex structures)

Creating NumPy Arrays

You typically create arrays using the `np.array()` function after importing NumPy:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr) # Output: [1 2 3 4 5]
```

You can also create arrays from tuples or lists of lists, and specify the data type if needed[1][4].

Array Dimensions and Examples

Dimension	Description	Example Code	Output
0 D Scal ar	Single value	np.array(42)	42
1 D Vect o r	List of values	np.array([^2_1][^2_2][^2_3][^2_4][^2_5])	1 2 3 4 5
2 D Mat ri x	List of lists	np.array([[^2_1][^2_2][^2_3],[^2_4][^2_5] [^2_6]])	1 2 3 4 5 6
3 D Tensor	List of lists of lists	np.array([[[^2_1][^2_2][^2_3],[^2_4][^2_5] [^2_6]], [[^2_7][^2_8][^2_9],[^2_10][^2_11] [^2_12]]])	1 2 3 4 5 6 7 8 9 10 11 12

Example: Creating a 2D Array

```
import numpy as np
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)
# Output:
# [[1 2 3]
# [4 5 6]]
```

Example: Creating a 3D Array

```
import numpy as np
arr3d = np.array([[[10, 20], [40, 50]], [[70, 80], [90, 50]]])
print(arr3d)
# Output:
# [[[10 20]
```

```
#    [40 50]]
#   [[70 80]
#    [90 50]]]
```

You can check the number of dimensions with `.ndim`:

```
print(arr3d.ndim)  #Output: 3
```

Special Arrays: Zeros and Ones

NumPy provides convenient functions to quickly create arrays filled with zeros or ones:

```
zeros_array = np.zeros((3,4))     #3x4 array of zeros
ones_array = np.ones(5)           #1D array of five ones
```

Array Operations

NumPy arrays support fast, element-wise operations:

```
arr = np.array([1, 2, 3, 4, 5])
print(arr + 10)      # Output: [11 12 13 14 15] #
print(arr * 3)       Output: [ 3 6 9 12 15]
```

You can also perform operations between arrays of the same shape:

```
a   =   np.array([1,   2,   3])   b   =
np.array([4,  5,  6])  print(a  +  b)  #
Output: [5 7 9]
```

Indexing and Slicing

Indexing allows you to access specific elements:

- 1D array: `arr[^2_2]` **gives the third element.**
- 2D array: `arr[^2_1][^2_2]` **gives the element in the second row, third column.**
- 3D array: `arr[^2_1][^2_1]` **navigates through layers, rows, and columns.**

Slicing **extracts parts of arrays:**

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[1:4]) # Output: [20 30 40]

arr2d = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
```

```
print(arr2d[0,    :])    #First row: [10 20 30]
print(arr2d[:, 2])       #Third column: [30 60 90]
```

Why Use NumPy Arrays?

- Much faster than Python lists for numerical operations.
- Support for multi-dimensional data.
- Enable element-wise operations and powerful mathematical functions.
- Essential for scientific computing, data analysis, and machine learning tasks

In summary:

NumPyarrays are the foundation of numerical computing in Python. They allow you to efficiently store and manipulate data in one or more dimensions, perform fast operations, and access elements precisely using indexing and slicing. Mastering arrays is key to working effectively in data science and scientific programming

Library	Purpose	Example Use Case	Integration
Pandas	Data manipulation/analysis	Cleaning, transforming, reshaping	Works with Matplotlib, Seaborn
Matplotlib	Data visualization (basic)	Line, bar, histogram plots	Foundation for Seaborn
Seaborn	Statistical data visualization	Attractive, statistical plots	Built on Matplotlib, uses Pandas Dat aF ram es