Exploratory Data Analysis

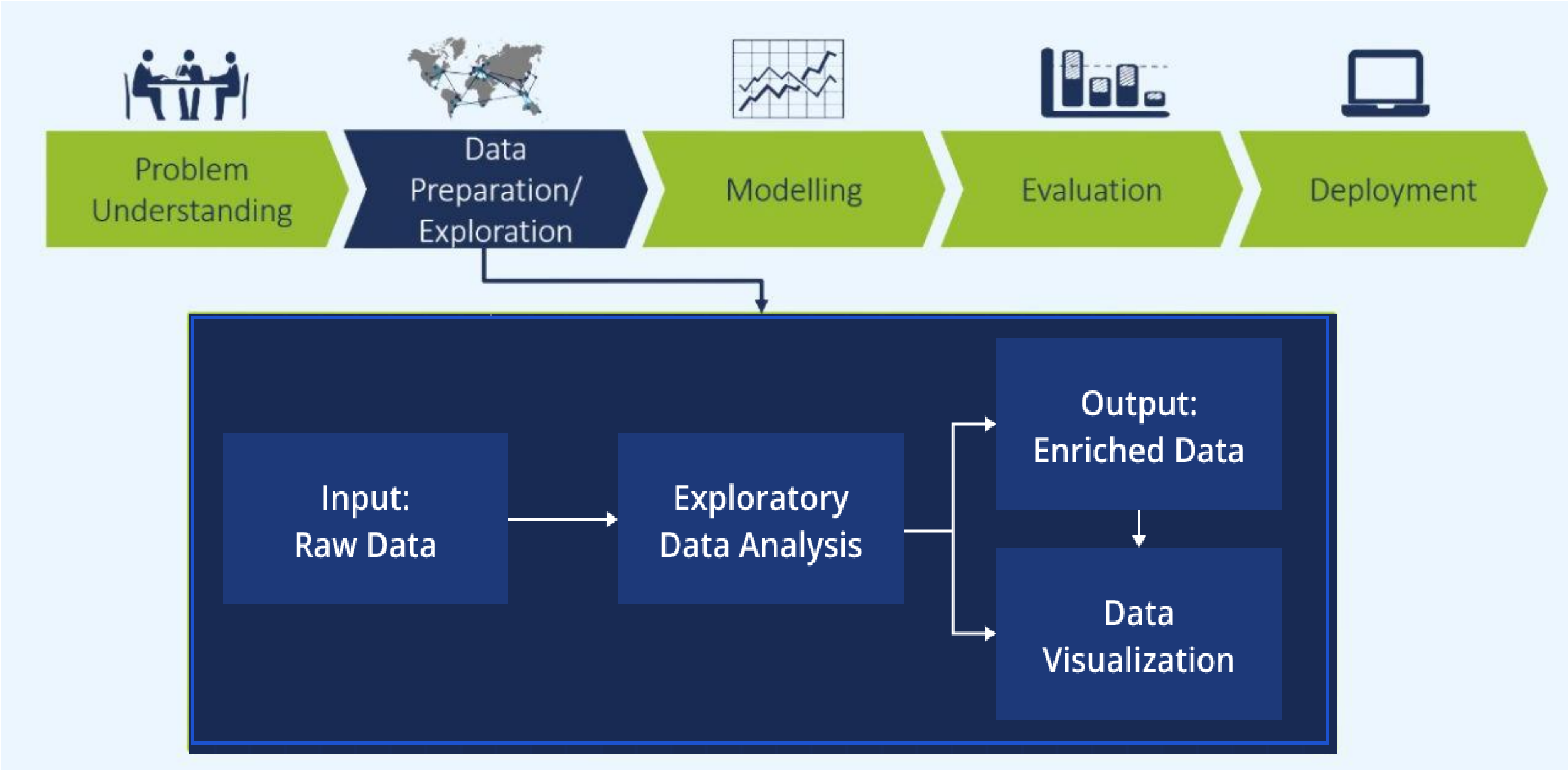
NumPy

Lecture 4 – HCCDA-AI

Imran Nawar

1 June 2025

Exploratory Data Analysis



Exploratory Data Analysis

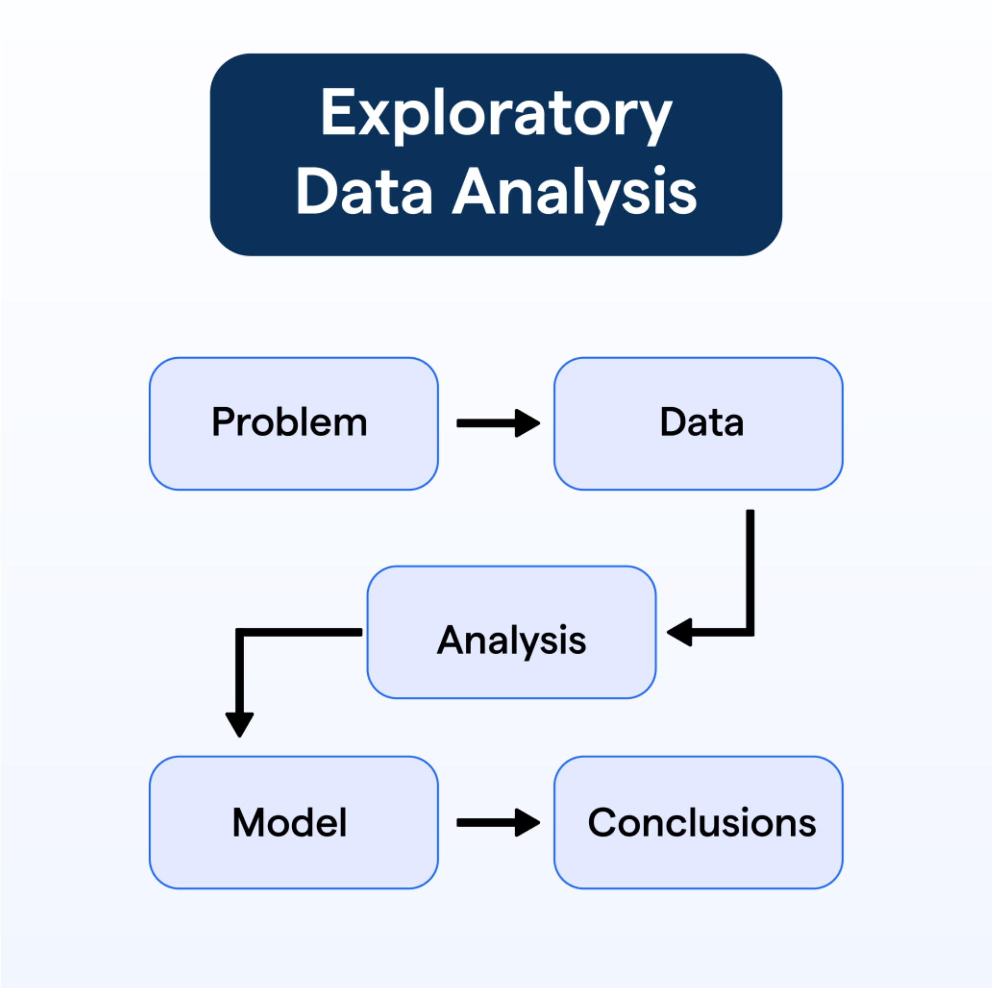
- The process of examining datasets to summarize their main characteristics, often with visual methods.
- **Purpose:** A crucial step in the data analysis workflow to gain a deep understanding of the dataset before modeling.

Objectives:

- Understand data structure and underlying patterns.
- Identify anomalies, missing values, and outliers.
- Detect trends and relationships between variables
- Form hypothesis to inform further analysis or modeling

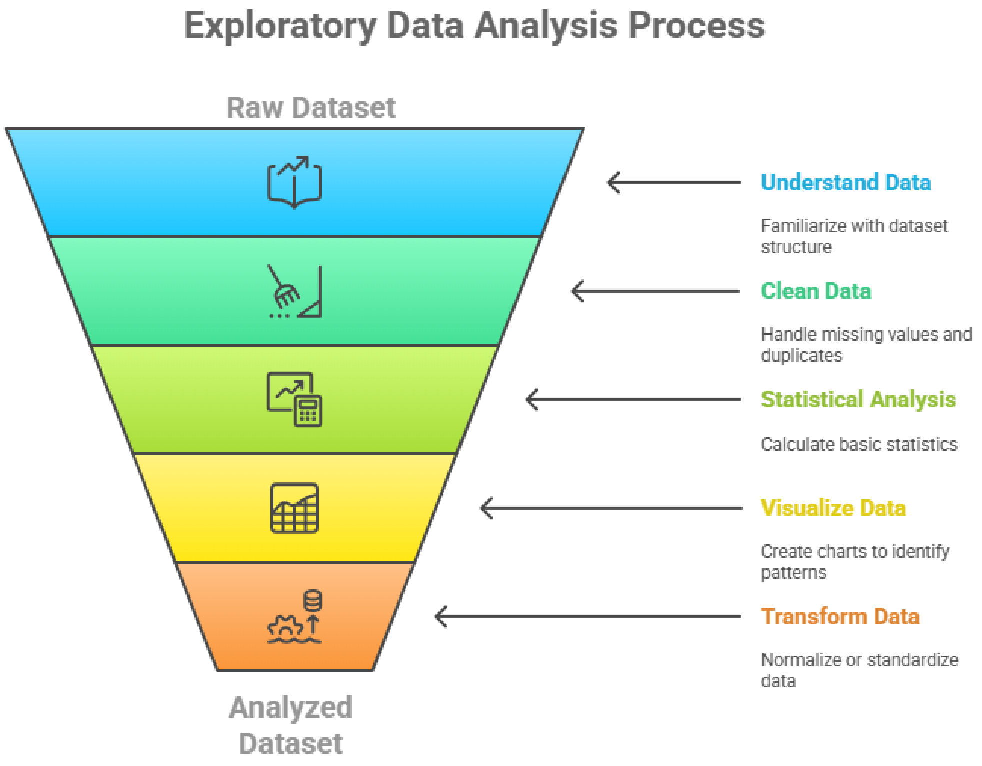
Importance:

- Provides insights for data-driven decision making.
- Improves predictive model quality by identifying issues early
- Ensures data integrity and readiness for analysis.



Key Steps in Exploratory Data Analysis (EDA)

- **Understanding the Data:** Get familiar with the dataset, look at the number of rows, columns, and types of data.
- **Data Cleaning:** Handle missing values, duplicates, and inconsistencies.
- **Statistical Analysis:** Use basic statistics like mean, median, and standard deviation to understand each variable.
- **Data Visualization:** Use charts to uncover patterns, trends and outliers.
- **Data Transformation** (if needed)**:** Normalize or standardize values, or convert data into a better format for analysis.



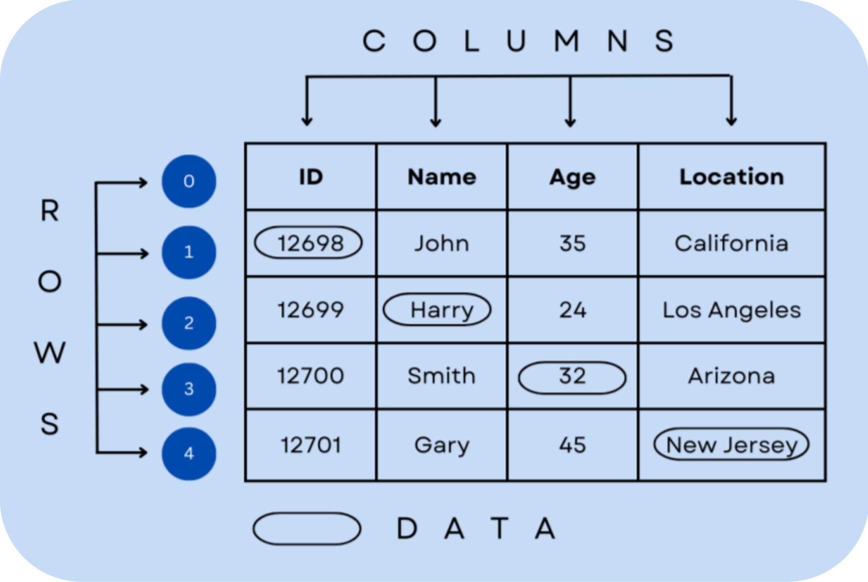
Python Libraries for EDA

Pandas

- **Initial release:** 2009
- **Purpose:** Data manipulation and analysis

Key Features:

- **DataFrame and Series Objects:** Provide a flexible structure for handling tabular and time-series data.
- **Data Cleaning:** Handle missing values, duplicates, and data type conversions
- **Filtering and Indexing:** Retrieve specific data easily based on conditions or indices.
- **Grouping and Aggregation:** Perform operations like sum, mean, and count on grouped data.
- **Data Merging and Joining:** Combine datasets using functions like `merge()` and `concat()`.
- **Time Series Support:** Includes built-in tools for resampling, shifting, and rolling computations.



Python Libraries for EDA

Matplotlib:

- **Initial release:** 2003
- First Python data visualization library.
  - Most popular and widely used data visualization library.
- Matplotlib is the grandfather of python visualization packages.
  - Foundation for many other libraries.
  - Popular libraries like Seaborn and Plotly are built on Matplotlib, using it as the core for rendering plots.
- **Highly Customizable:**
  - Low-level, highly customizable plotting library.
  - It offering fine-grained control over plot elements (axes, labels, ticks, colors), enabling a wide range of visualizations from basic bar charts to complex interactive 2D graphs.
- **Key Strengths:**
  - **Powerful but Complex:**
    - While flexible, it can be challenging for beginners compared to higher-level libraries like Seaborn.
  - **Multiple Visualization Modes:**
    - Supports static, animated, and interactive plots for diverse use cases.

Matplotlib's original logo (2003 -- 2008).

Matplotlib's logo (2008 -- 2015).

Python Libraries for EDA

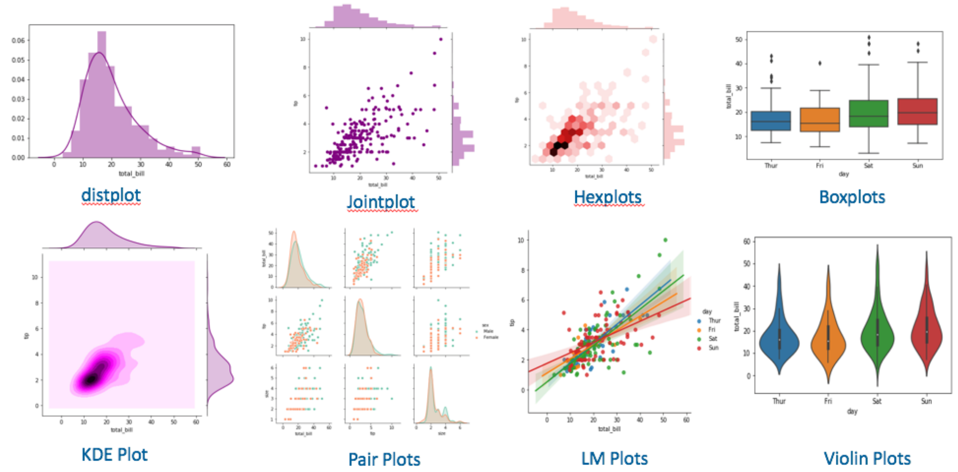
Seaborn:

- **Initial Release:** 2014
- It is an advanced data visualization library built on top of Matplotlib, designed to make complex statistical plots easier to create.

Key Features

- **High-Level Interface**
  - Intuitive API for statistical graphics
  - Minimal effort required
- **Advanced Visualizations**
  - Built-in functions for complex plots
  - **Examples:** heatmaps, pair plots, violin plots, regression p
- **Data Integration**
  - Seamless integration with Pandas DataFrames
  - Efficient handling of large datasets

Seaborn Plots



Key Strengths

- User-friendly and less complex compared to Matplotlib.
- Designed to make statistical visualizations accessible and attractive.

Python Libraries for EDA
Plotly:

- A powerful open-source graphing library for Python, R, and JavaScript.
- Creates interactive, publication-quality visualizations.
- Supports a wide range of chart types and interactive features.
- Enables Python users to create stunning web-based visualizations.
- Integrates seamlessly with web applications and Jupyter notebooks.
- Visualizations can be saved as standalone HTML files or displayed directly in Jupyter notebooks.

Installation and Setup

- Install Plotly using pip.

```
pip install plotly
```

- For notebook environments, install additional package *notebook* if required:

```
pip install "notebook>=5.3" "ipwidgets>=7.5"
```

- Basic usage in Python:

```
import plotly.graph_objects as go
import plotly.express as px
```

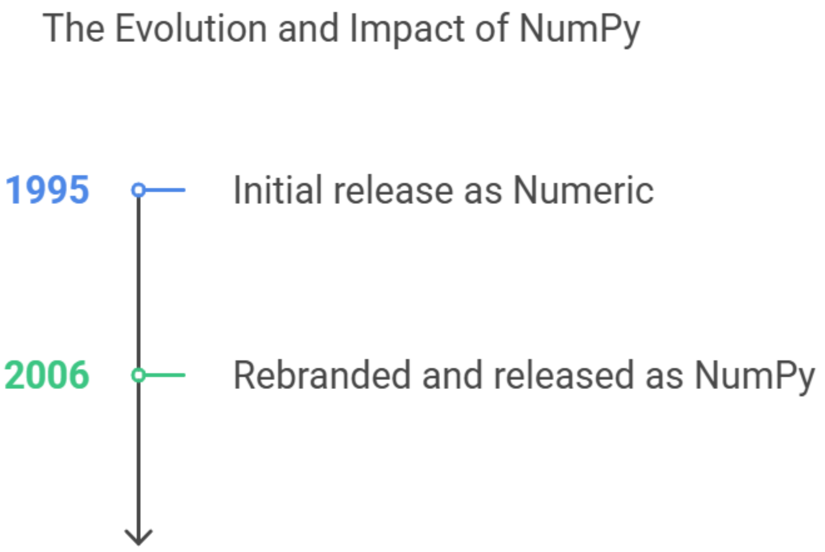
```
(DIP_7th) C:\Users\PC>pip install plotly
Collecting plotly
  Downloading plotly-5.24.1-py3-none-any.whl.metadata (7.
3 kB)
Collecting tenacity>=6.2.0 (from plotly)
  Downloading tenacity-9.0.0-py3-none-any.whl.metadata (1
.2 kB)
Requirement already satisfied: packaging in c:\users\pc\a
naconda3\envs\dip_7th\lib\site-packages (from plotly) (24
.1)
Downloading plotly-5.24.1-py3-none-any.whl (19.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 19.1/19.1 MB
241.9 kB/s eta 0:00:00
Downloading tenacity-9.0.0-py3-none-any.whl (28 kB)
Installing collected packages: tenacity, plotly
Successfully installed plotly-5.24.1 tenacity-9.0.0
```

NumPy

Introduction to NumPy

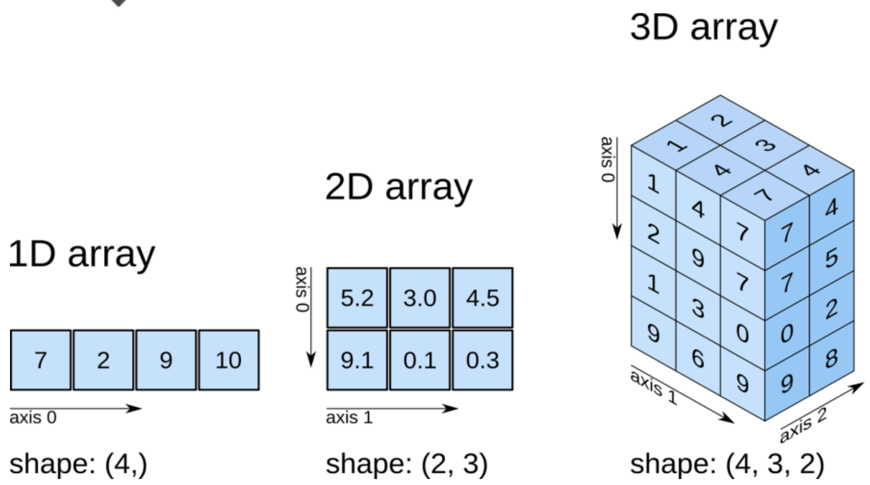
NumPy: "Numerical Python" – The foundation of scientific computing in Python

- **Initial release:** As **Numeric** (1995); as NumPy, (2006)
- Python library for efficient array operations and mathematical computations
- **Core Data Structure**: N-dimensional array (ndarray)



Why NumPy?

- **Speed:** Up to 50x faster than Python lists.
- **Memory Efficient:** Uses less memory than traditional Python data structures.
- **Foundation:** Backend for Pandas, SciPy, Matplotlib, and machine learning libraries
- **Versatile:** Supports 1D arrays, matrices, tensors, and higher dimensions.

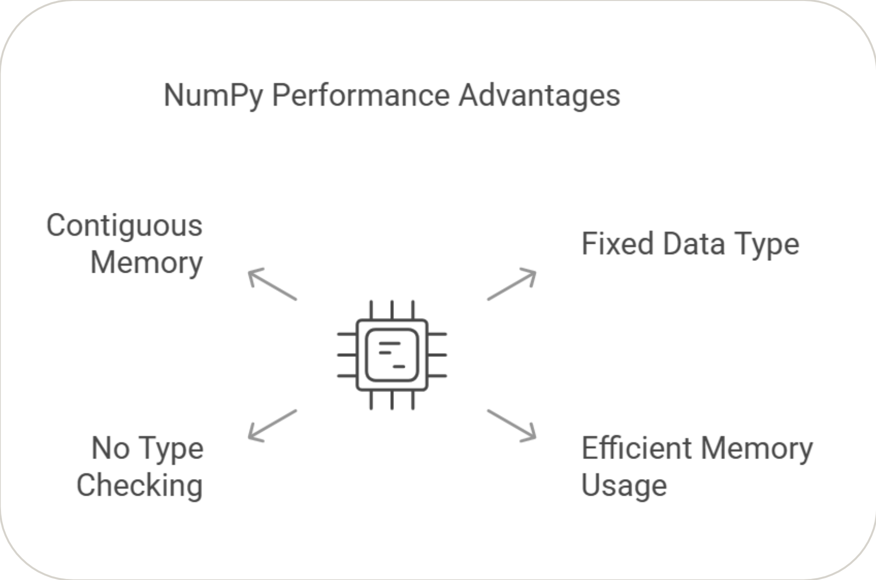


Key Features

1. **Efficient Data Structures**:
   - **Memory-efficient:** Contiguous storage reduces memory footprint
   - **Homogeneous data:** All elements have the same data type for optimal performance
2. **Mathematical Operations:**
   - **Vectorized operations:** Apply functions to entire arrays at once
   - **Built-in functions:** Comprehensive library for complex mathematical calculations
3. **Random Number Generation**:
   - Functions for generating random numbers and datasets.
   - Useful in simulations, statistical modeling, and testing.
4. **Performance Optimization**:
   - **Low-level implementation:** Core written in C and Fortran for speed
   - **Significant speedup:** Up to 50x faster than native Python operations
5. **Ecosystem Integration**:
   - **Foundation library:** Backend for major Python data science libraries
   - **Seamless compatibility:** Works with Pandas, SciPy, Matplotlib, Scikit-learn

Why NumPy Arrays are Faster Than Lists

1. **Fixed Data Type**:
   - NumPy arrays have a uniform data type, which eliminates the overhead of managing different data types like in Python lists.
2. **Contiguous Memory Storage**:
   - Data stored in continuous memory blocks enhances cache efficiency and reduces memory consumption.
3. **No Runtime Type Checking**:
   - Operations on NumPy arrays skip runtime type checks, making computations faster compared to Python lists.
4. **Optimized Implementation**:
   - Core operations written in C and Fortran for maximum performance.



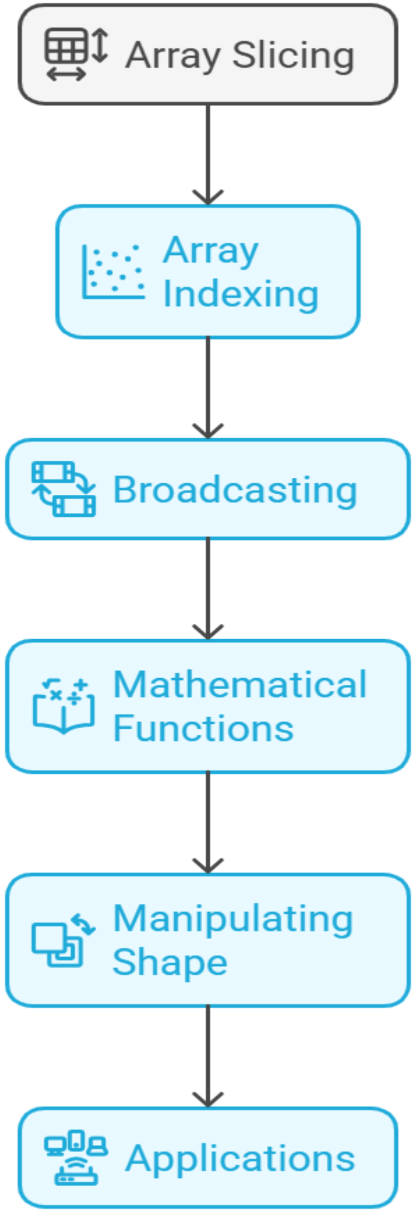
Differences Between Python Lists and NumPy Arrays:

Feature	Python Lists	NumPy Arrays
Data Type	Mixed Types	Homogeneous types
Memory Efficiency	Low	High
Computation Speed	Slow	Fast

Advanced Operations in NumPy

Slicing, Indexing, and Advanced Operations

1) **Array Slicing:**
   - Access subsets of data.
   - **Example:** array[1:5] retrieves elements from index 1 to 4.
2) **Array Indexing:**
   - Access or modify specific elements.
   - **Example**: array[0, 2] retrieves the element at row 0, column 2 in a 2D array.
3) **Broadcasting:**
   - Perform operations on arrays of different shapes.
   - **Example:** Add a scalar to an array.
4) **Mathematical Functions:**
   - Built-in functions for trigonometry, statistics, linear algebra, etc.
   - **Example:** `np.mean(array)` computes the mean.
5) **Shape Manipulation:**
   - Reshape arrays to desired dimensions.
   - **Example**: `array.reshape(3, 4)` reshapes flat array to 3x4.



Thank You