# Exploratory Data Analysis

# **Pandas**

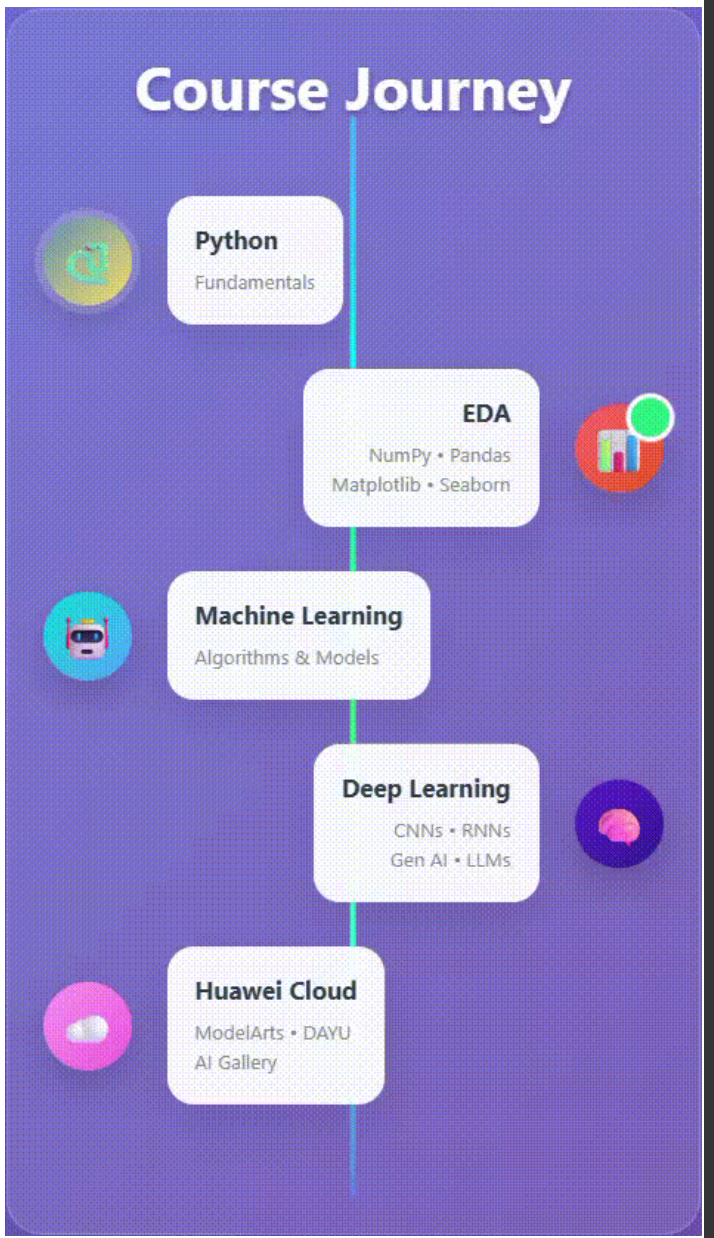
Lecture 5 – HCCDA-AI

Imran Nawar

14 June 2025

# **Course Progress:** Where We Are and What's Ahead

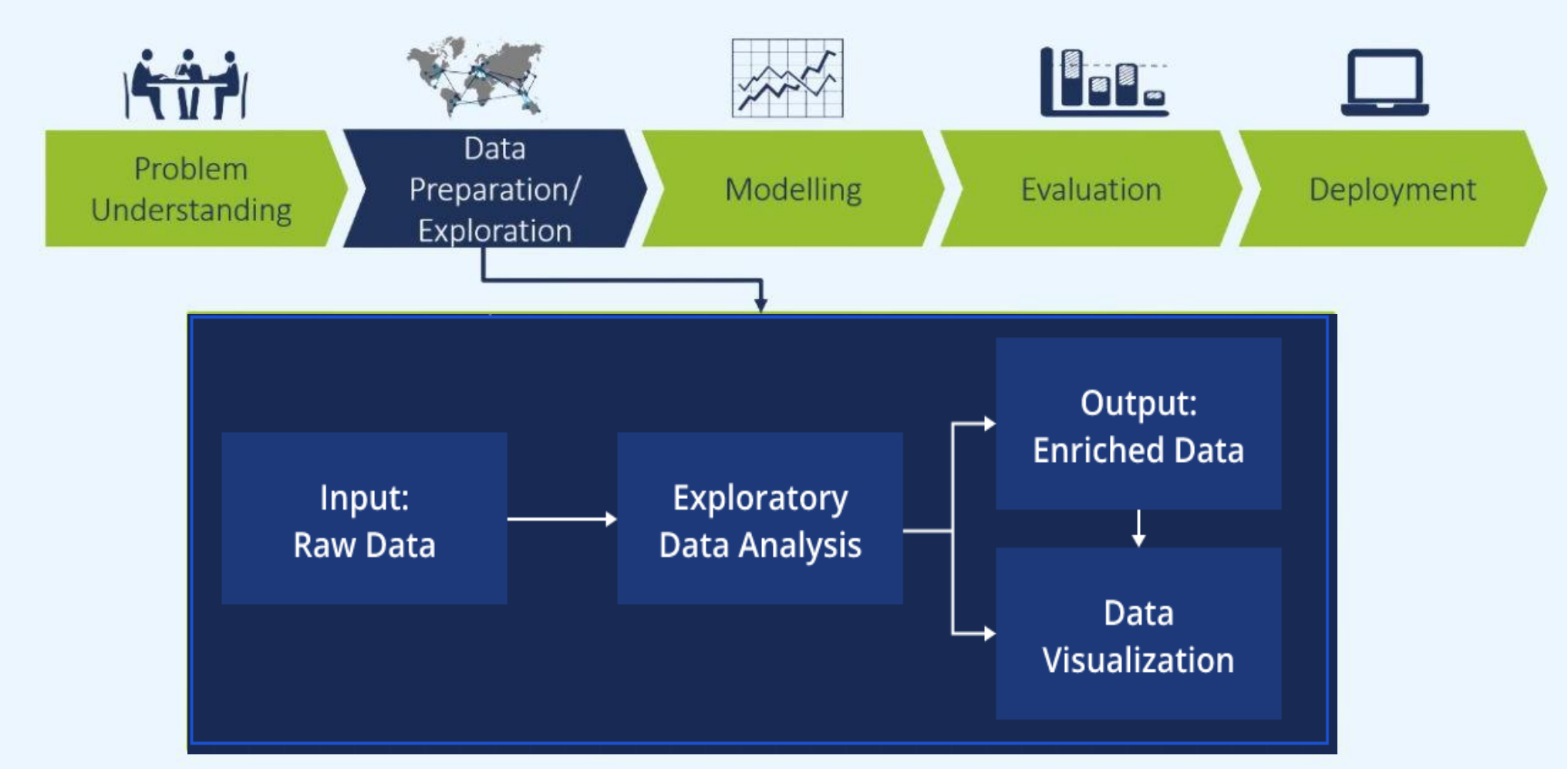
- **Python**
  - Python Fundamentals
- **Exploratory Data Analysis**
  - NumPy
  - Pandas
  - Data Visualization
    - Matplotlib
    - Seaborn
- **Machine Learning**
- **Deep Learning**
  - Convolutional Neural Networks (Deep Computer Vision)
  - Sequence Learning
  - Deep Generative AI
  - Large Language Models
- **Huawei Cloud AI Services**
  - ModelArts (AI Development Platform)
  - DAYU (Data Processing)
  - AI Gallery & Pre-trained Models
  - …



# Course Progress Overview

- **Lecture 1**:
  - Introduction to Programming, Installation and Setup
  - Variables, Data Types → *int, float, str, bool, list, tuple, dict*
  - Conditional Statements → *if, if-else, if-elif-else*
  - Loops → *while, for (range(), zip(), enumerate(), break, continue, pass)*
- **Lecture 2**:
  - Functions, Types of arguments → *positional , keyword, *args*
  - Programming Paradigms
  - Object Oriented Programming: Classes, Objects, Attributes, Methods
  - Constructor → __init__ Method
- **Lecture 3**:
  - Advanced OOP → (*Inheritance, Polymorphism, Abstraction, Encapsulation*)
- **Lecture 4**:
  - Exception Handling, File Handling
  - Exploratory Data Analysis, Python libraries overview for EDA
  - NumPy Library → *numpy array, slicing, indexing* etc.
- **Coming Up Next**: **Pandas Library**

# Exploratory Data Analysis



# Exploratory Data Analysis

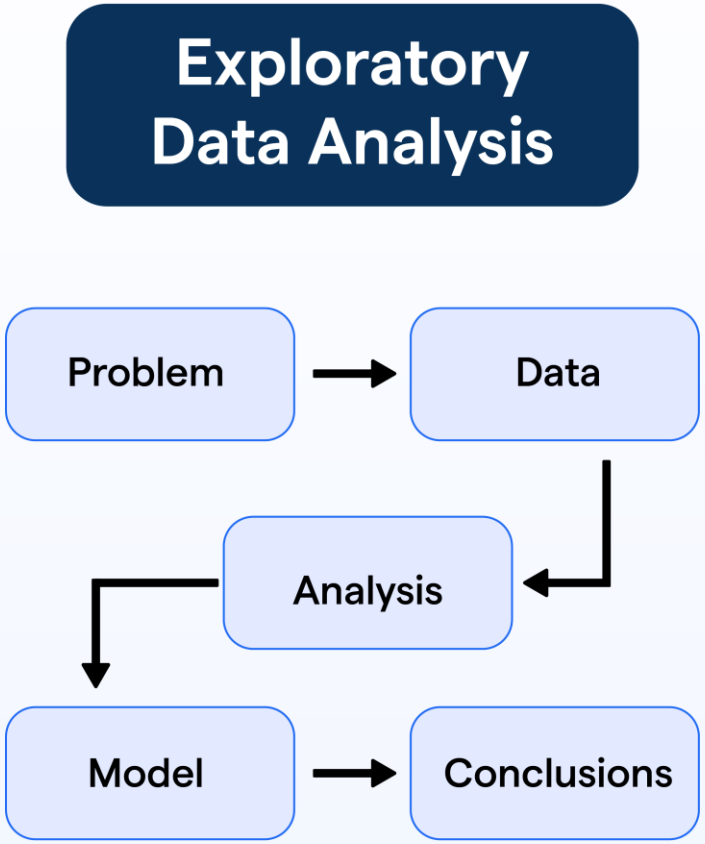
- The process of examining datasets – often with visual methods – to summarize their main characteristics.
- It is a crucial step in the data analysis workflow to gain a deep understanding of the dataset before modeling.

## 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.

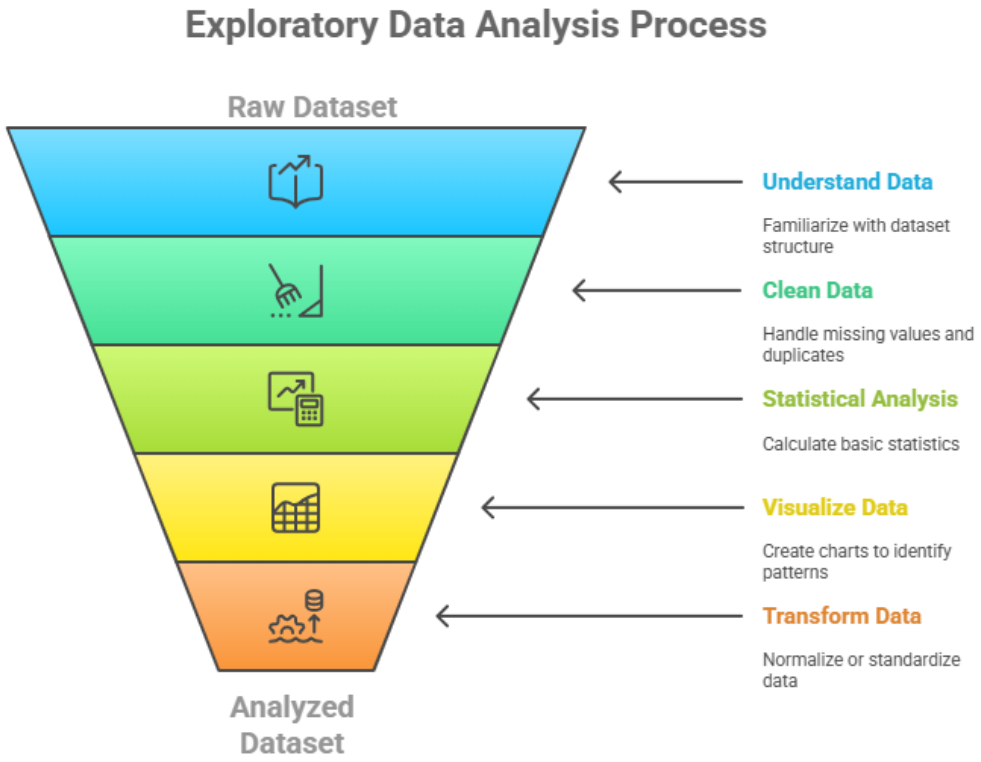
## Importance:

- 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 EDA

- **Understanding the Data:** Get familiar with the dataset, check number of rows, columns, and data types.
- **Data Cleaning:** Handle missing values, duplicates, and inconsistencies.
- **Statistical Analysis:** Use basic statistics (mean, median, standard deviation) to summarize 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 further analysis or modeling.



# Python Libraries for EDA

1. **NumPy** "Numerical Python"**:**
   - Foundation of scientific computing in Python.
   - Provides support for large, multi-dimensional arrays and matrices.
   - Offers mathematical functions for fast numerical computations.


2. **Pandas:**
   - Powerful library for data manipulation and analysis
   - Provides easy-to-use data structures: *Series* and *DataFrame*
   - Ideal for cleaning, transforming, and summarizing tabular data


3. **Matplotlib:**
   - First Python data visualization library
   - Highly customizable and widely used for 2D plotting
   - Useful for creating basic plots: *line, bar, histogram, scatter*, etc.

# Python Libraries for EDA

## 4. Seaborn:

- Built on top of Matplotlib (**initial Release:** 2014)
- Simplifies the creation of complex statistical visualizations.
- Offers beautiful default styles and functions for visualizing distributions, regression, and categorical data.


## 5. Plotly:

- Open-source interactive graphing library for Python, R, and JavaScript.
- Enables creation of interactive, publication-quality charts and dashboards.
- Supports 3D plots, animations, and web-based visualizations.


# Pandas

Python Data Analysis Library

# Introduction to Pandas



## Initial release: 2009

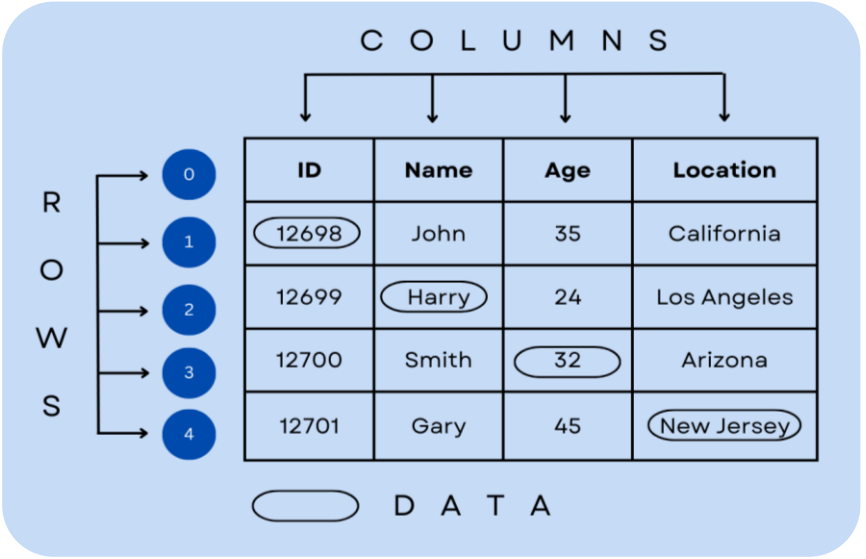
- Pandas is a powerful Python library for data manipulation and analysis.
- The name **"Pandas"** comes from **"Panel Data"** (a term for multidimensional structured datasets) and **"Python Data Analysis,"** reflecting its purpose as a library for data manipulation and analysis.
- Built on **NumPy**: Enables efficient handling of numerical data.
- Integrates seamlessly with **Matplotlib** and **Seaborn** for visualization.

## Key Strengths:

- Handles structured data efficiently.
- Simplifies tasks like data cleaning, transformation, and visualization.
- Offers powerful tools for working with tabular and time-series data.

## Code Example:

```
import pandas as pd
print(pd.__version__)
```



# Pandas Data Structures

## Series:

- A 1-dimensional, array-like structure with labeled indices.
- Used for storing and manipulating a single column or list of data.

**Code Example:**

```
import pandas as pd
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(s)
```

```
a 10
b 20
c 30
```

Series + Series = DataFrame

| | apples |
|---|---|
| 0 | 3 |
| 1 | 2 |
| 2 | 0 |
| 3 | 1 |

+

| | oranges |
|---|---|
| 0 | 0 |
| 1 | 3 |
| 2 | 7 |
| 3 | 2 |

=

| | apples | oranges |
|---|---|---|
| 0 | 3 | 0 |
| 1 | 2 | 3 |
| 2 | 0 | 7 |
| 3 | 1 | 2 |

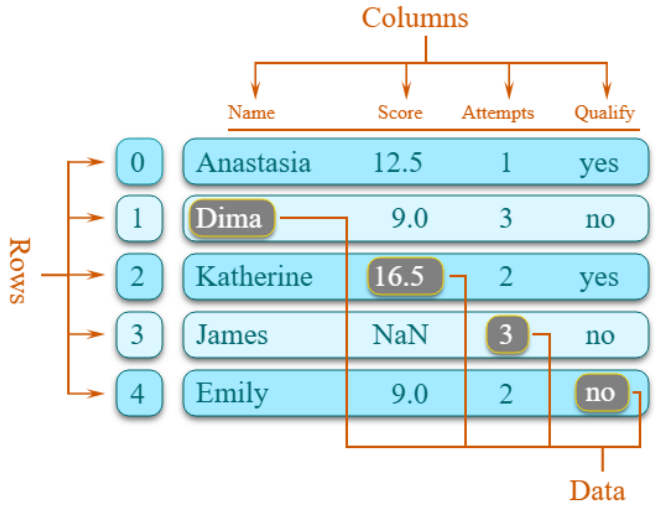
## DataFrame:

- A 2-dimensional tabular structure with rows and columns.
- Can be created from dictionaries, lists, or NumPy arrays.

- **Key Features**:
  - Supports heterogeneous data types.
  - Easy data manipulation and aggregation.
  - Offers methods to filter, group, and transform data efficiently.

**Code Example:**

```
data = 'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df = pd.DataFrame(data)
```




Pandas DataFrame

| | Name | Age |
|---|---|---|
| 0 | Alice | 25 |
| 1 | Bob | 30 |

# Basic Data Manipulation with Pandas

## Indexing and Selection:

Pandas provides flexible tools to retrieve specific subsets of data.

- **Key Methods:**
  - `loc[]` – Access rows/columns by **labels** (e.g., row/column names).
  - `iloc[]` – Access rows/columns by **integer positions** (e.g., index numbers).

```
df.loc[2, 'Price']     # Access value in row with label 2 and column 'Price'
df.iloc[2, 1]          # Access value at 3rd row, 2nd column
```

## Handling Missing Data:

Clean and prepare your data by identifying and dealing with missing values.

- **Key Functions:**
  - `isnull()` – Detect missing values (return True/False).
  - `fillna()` – Fill missing values using a strategy (e.g., mean, median, constant)
  - `dropna()` – Remove rows or columns with missing data.

```
df['Price'].fillna(df['Price'].mean(), inplace=True)
df.dropna(inplace=True)
```

# Filling in missing data

```
print(data)
```

```
     0   1   2   3
0  1.0  2.0  3.0 NaN
1  1.0  NaN  NaN NaN
2  NaN  NaN  NaN NaN
3  NaN  4.0  5.0 NaN
```

```
print(data.fillna(0))
```

```
     0   1   2   4
0  1.0  2.0  3.0  0.0
1  1.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  0.0  4.0  5.0  0.0
```

```
print(data.fillna(0, inplace=True))
print(data)
```

```
     0   1   2   4
0  1.0  2.0  3.0  0.0
1  1.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  0.0  4.0  5.0  0.0
```

Modify the dataframe instead of returning a new object (default)

```
print(data)
```

```
     0   1   2
0  1.0  2.0  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  4.0  5.0
```

```
print(data.fillna(data.mean(skipna=True)))
```

```
     0   1   2
0  1.0  2.0  3.0
1  1.0  3.0  4.0
2  1.0  3.0  4.0
3  1.0  4.0  5.0
```

replace NaN with column mean

# Data Aggregation and Transformation

- **Grouping & Aggregating Data:**

  Use `groupby()` to split data into groups based on column values and perform aggregate operations.

  **Common Aggregation Methods:**

  - mean(), sum(), count(), min(), max(), etc.

```python
import pandas as pd

df = pd.DataFrame({
    'Category': ['A', 'A', 'B', 'B'],
    'Sales': [100, 150, 80, 120]
})

grouped = df.groupby('Category').sum()
grouped
```

[12] ✓ 0.0s Python

| Category | Sales |
|---|---|
| A | 250 |
| B | 200 |

- **Applying Custom Functions:**

  - Use `apply()` or `transform()` to apply custom or built-in functions to rows or columns.

  **When to Use:**

  - Normalize data
  - Apply conditional logic
  - Add calculated features

**Example** – Add 5 to each value in the 'Age' column:

```python
def add_five(x):
    return x + 5

df = pd.DataFrame({'Age': [25, 30, 35]})
df['New_Age'] = df['Age'].apply(add_five)
df
```

[14] ✓ 0.0s Python

| | Age | New_Age |
|---|---|---|
| 0 | 25 | 30 |
| 1 | 30 | 35 |
| 2 | 35 | 40 |

# Working with Dates and Times

- Pandas offers powerful tools for handling **date** and **time** data, essential for time-series analysis.

## Key Features and Functions:

1. **Converting to Datetime:**
   - Use `pd.to_datetime()` to convert strings or other formats to datetime objects.
   - Support flexible parsing of various date/time formats.

```
df['Date'] = pd.to_datetime(df['Date'])
```

2. **DateTime Indexing:**
   - Set a datetime column as the index to enable time-based operations.
   - Enables easy filtering, slicing and resampling.

```
df.set_index('Date', inplace=True)
```

3. **Resmapling Time Series Data:**
   - Use `resample()` to change the frequency of observations (e.g., from daily to monthly).
   - Combine with aggregation functions like `mean(), sum()` or `count()`.

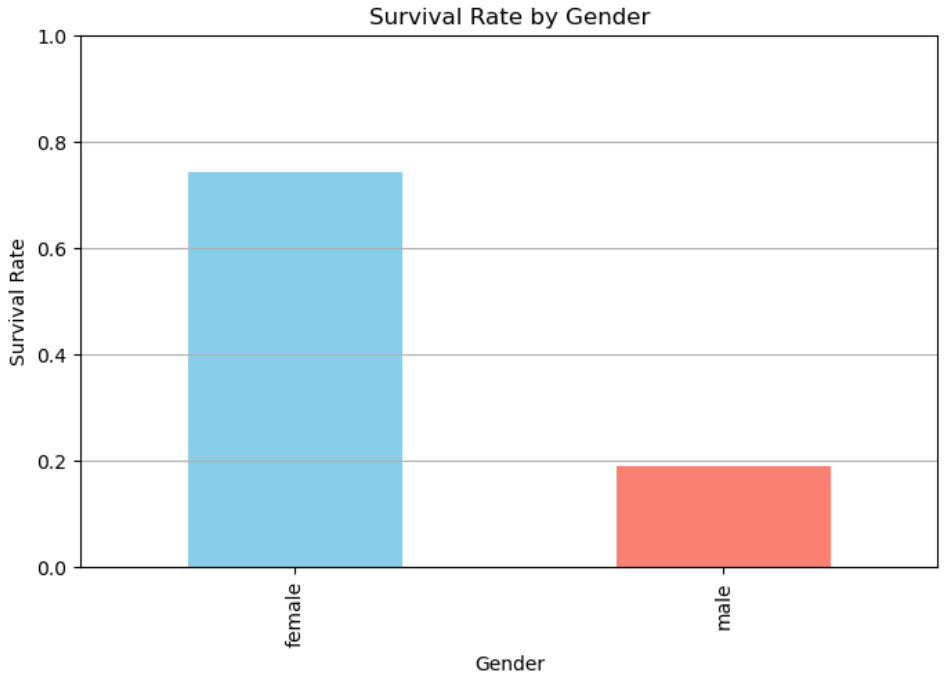
```
monthly_avg = df.resample('M').mean()
```

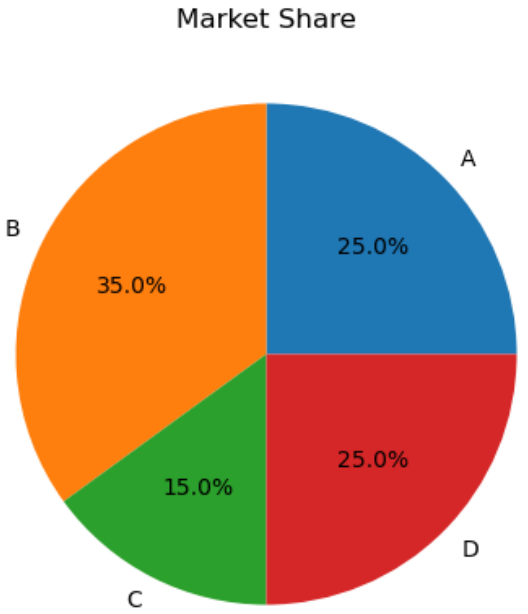
### Why it Matters

- Essential for trend analysis, forecasting, and seasonality detection.
- Helps in transforming irregular time-stamped data into consistent intervals.

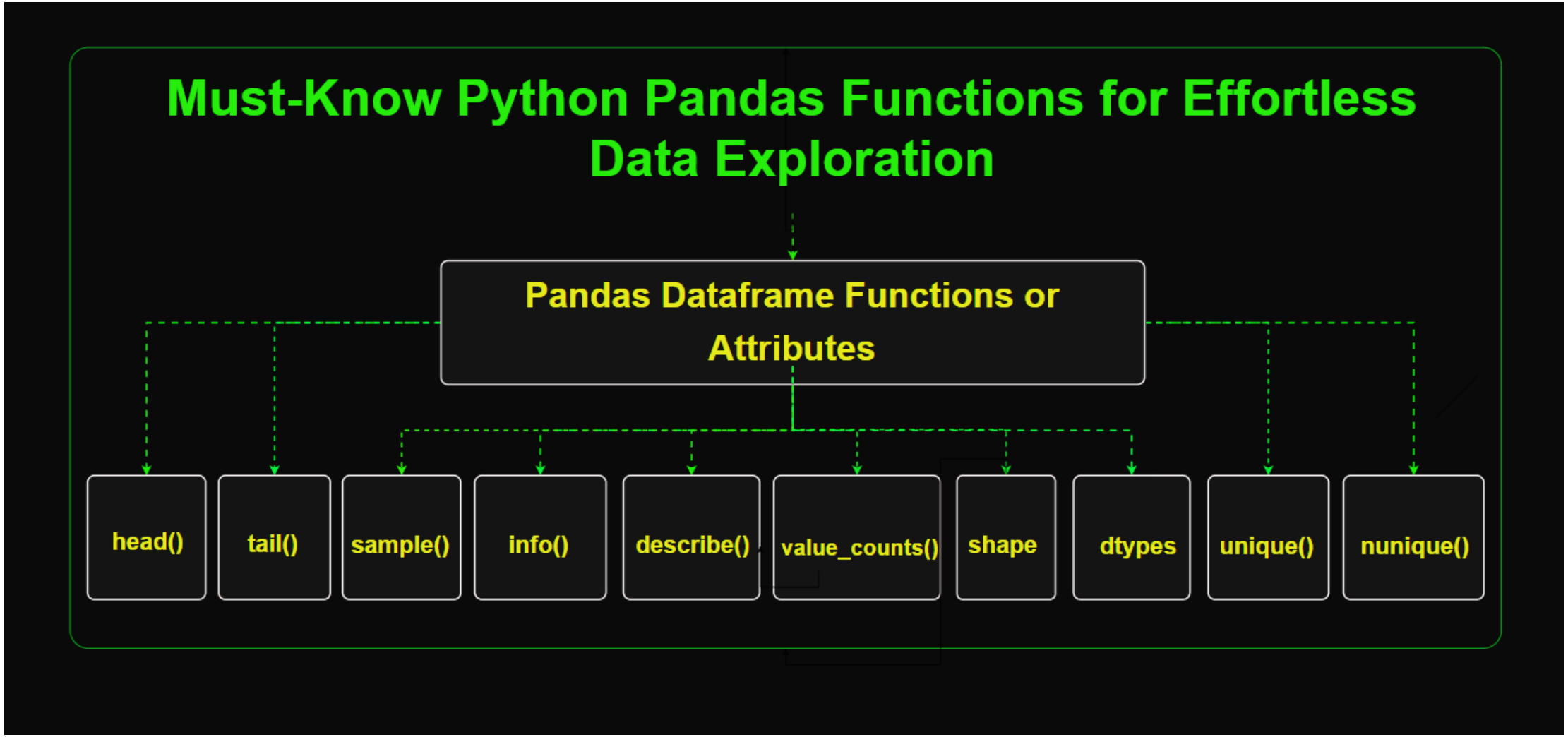
# Data Visualization with Pandas

- Pandas includes built-in plotting functions based on Matplotlib.
- **Features:**
  - Convenient for quick visualizations directly from DataFrames.
  - Ideal for simple, exploratory plots.
  - Compatible with Matplotlib for more customization.
- **Common Plots:**
  - **Line Plot:** Visualize trends over time.
  - **Bar Plot:** Compare categorical data.
  - **Histogram:** Analyze the distribution of data.
- **Use Case**: Ideal for rapid, straightforward visual analysis during EDA.



# Python Pandas Functions



# Thank You