**VISUALIZATION**

­Data visualization is a critical step in the data science workflow. It transforms raw data into visual representations, allowing analysts, scientists, and stakeholders to grasp patterns, trends, and outliers more intuitively and effectively.

There are multiple libraries that has been used to visualize the data in the data visualization such as pandas, matplotlib, seaborn, numpy, plotly, etc.

Let’s get into the concept of data visualization in matplotib and pandas.

**MATPLOTLIB**

Matplotlib is a powerful, low-level data visualization library in Python. It provides comprehensive control over graph and chart customization and is often used to create static, animated, and interactive plots. It's the backbone of many other visualization libraries in Python.

**Key Features:**

- Generates high-quality 2D plots (line, bar, scatter, pie, histogram, etc.).

- Highly customizable (fonts, colors, labels, ticks, etc.).

- Can output to various formats (PNG, PDF, SVG, etc.).

- Works well with NumPy and Pandas data.

**PYPLOT:**

pyplot is a module in the Matplotlib library that provides a simple and convenient interface for creating plots, similar to how plotting works in MATLAB.

It acts as a collection of functions that make it easy to create, customize, and display plots such as line graphs, bar charts, histograms, scatter plots, etc.

You typically import it like this:



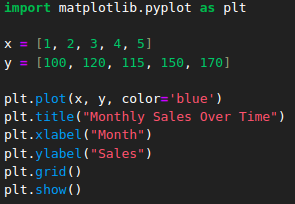
Some of the types of plots in Matplotlib:  
  
**1.Line Plot**

The plt.plot() function in Matplotlib draws lines connecting points (x, y). It is ideal for visualizing continuous data sequences and trends over time or indices. It’s flexible and supports markers, colors, and styles.

**Use Case:**

Track monthly sales, temperature over days, or any variable changing over time.

**Code Snippet:**

****

**Output:** **Description:**plot(): Plots a line through (x, y) values.

grid(): Adds grid lines for better visual tracking.

show(): Displays the output.

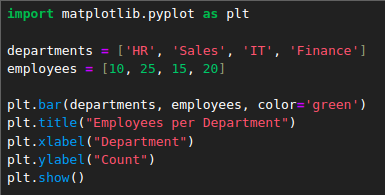
**2. Bar Chart**

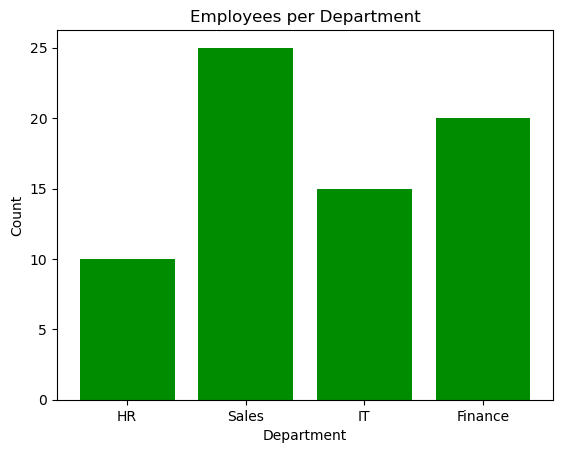
The plt.bar() function creates vertical bars to represent categorical data. The height of each bar represents the value of that category.

**Use Case:**

Visualize employee count per department, sales by region, or comparison across categories.

**Code Snippet:**

****

**Output:  
**

**Description:**

bar(x, y): Plots bars with height y for each category x.

color: Sets bar fill color.

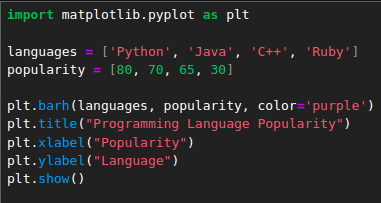
**3. Horizontal Bar Chart**

The plt.barh() function draws horizontal bars instead of vertical. It's useful when category names are long or to improve readability.

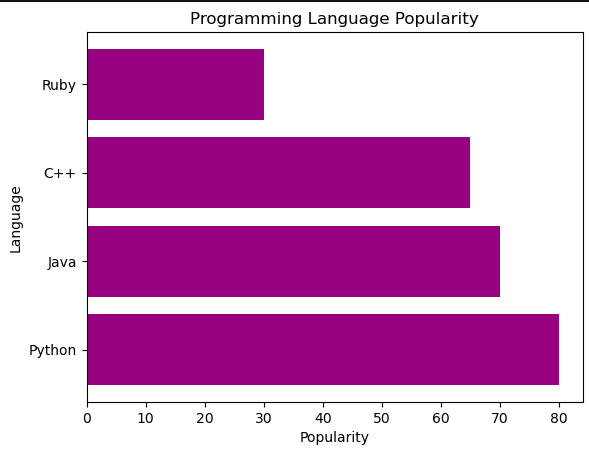
**Use Case:**

Compare programming language popularity where labels are long or many.

**Code Snippet:**

****

**Output:**

**  
  
  
  
  
  
  
  
  
Description:**barh(y, x): Plots horizontal bars.

Reverses x and y from vertical bar chart.

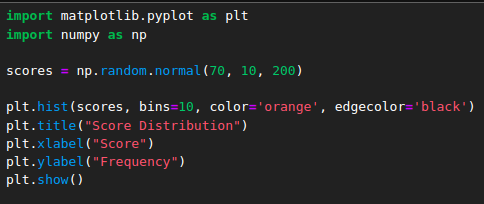
**4. Histogram**

The plt.hist() function groups numerical data into bins and displays the frequency of values in each range. It's great for understanding distribution.

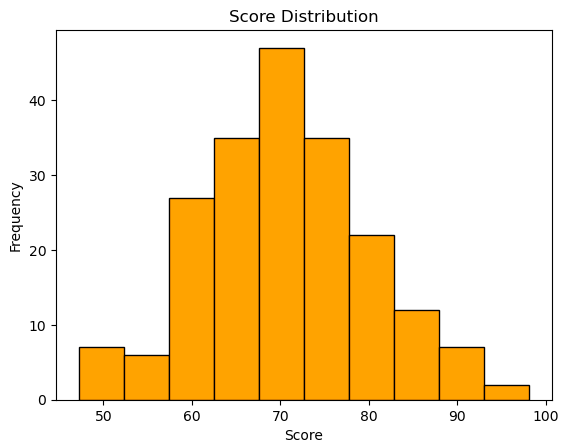
**Use Case:**

Analyze student score distribution, income levels, or ages.

**Code Snippet:**

****

**Output:**

****

**Description:**hist(data, bins): Plots histogram from data split into bins.

edgecolor: Outlines each bar.

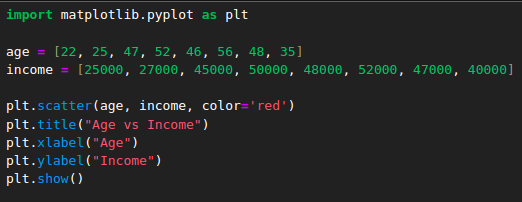
**5. Scatter Plot**

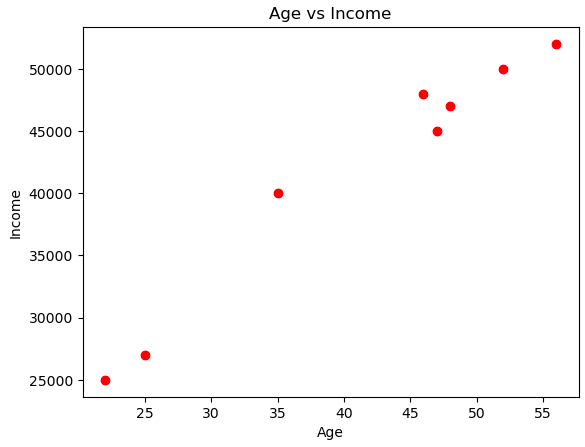
plt.scatter() creates a plot with unconnected points (dots). It's commonly used to show correlation or distribution of two variables.

**Use Case:**

Plot height vs weight, advertising spend vs revenue, or age vs income.

**Code Snippet:**

****

**Output:  
**

**Description:**

scatter(x, y): Draws dots for data pairs (x, y).

Best for observing relationships.

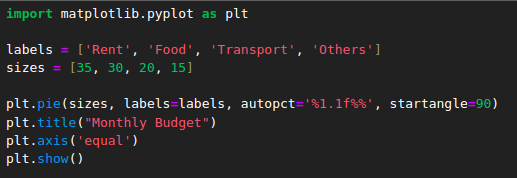
**6. Pie Chart**

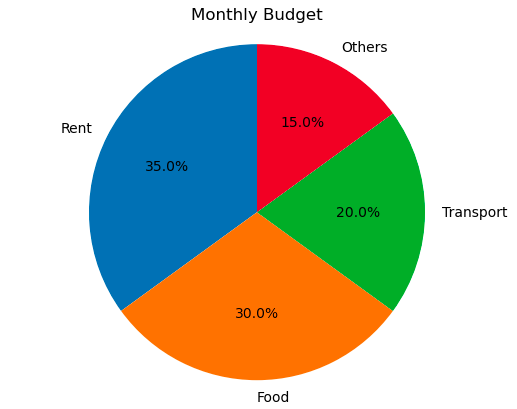
The plt.pie() function creates circular charts divided into wedges to illustrate proportions. It's used to show how a whole is split among parts.

**Use Case:**

Visualize market share, budget allocations, or survey response breakdown.

**Code Snippet:**



**Output:  
  
**

**Description:**

pie(sizes): Divides the circle by percentage of sizes.

autopct: Adds percentage labels.

axis('equal'): Ensures perfect circle.

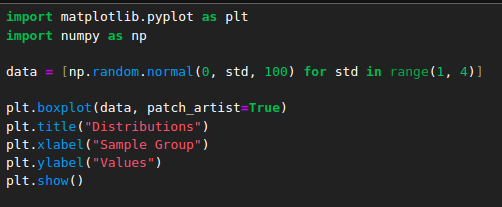
**7. Box Plot**

plt.boxplot() creates a summary of data based on median, quartiles, and outliers. This statistical visualization helps in comparing data spread and detecting anomalies.

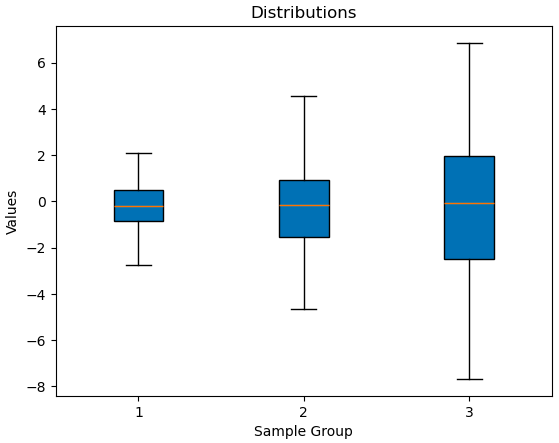
**Use Case:**

Compare test scores between classes, or analyze salary ranges in departments.

**Code Snippet:**

****

**Output:**

**  
  
  
  
  
  
  
  
  
  
  
  
Description:**

boxplot(data): Creates a box-and-whisker plot.

patch\_artist: Fills boxes with color

**Benefits of Matplotlib:**

1.Highly Customizable

- Control over every element of the plot (colors, fonts, ticks, layout).

2.Versatile

- Supports a wide range of plot types (line, bar, scatter, pie, 3D, etc.).

3.Publication-Quality Graphics

- Can produce high-resolution, professional-looking plots suitable for reports or papers.

4.Works with Multiple Libraries

- Integrates seamlessly with NumPy, Pandas, Seaborn, and more.

**Limitations:**

1.Verbose Syntax

- Requires more lines of code for simple plots compared to higher-level libraries.

2.Steeper Learning Curve

- Complex to master, especially for fine-tuning layouts and annotations.

3.Limited Built-in Aesthetics

- Default style is basic; often needs customization for visual appeal.

**Applications:**

- Exploratory data analysis (EDA)

- Reporting and dashboards

- Scientific visualization

- Custom publication-quality plots

**PANDAS**

Pandas is a fast, flexible, and expressive library designed for data manipulation and analysis. It introduces powerful data structures like DataFrame and Series, making it easy to clean, analyze, and manipulate data.

**Key Features:**

- Easy reading/writing of data (CSV, Excel, SQL, etc.)

- Rich functions for filtering, grouping, merging, pivoting, etc.

- Time series handling

- Built-in plotting (via integration with Matplotlib)  
  
**Core Data Structures in Pandas**

Pandas has two primary data types:

Series – One-dimensional labeled array

DataFrame – Two-dimensional labeled table

Let’s explore each in detail:

**1. Series**

A Series is a one-dimensional array-like object that contains:

Data (of any type — integer, string, float, object, etc.)

An associated index that labels each element

It’s similar to a single column in a spreadsheet or database table.

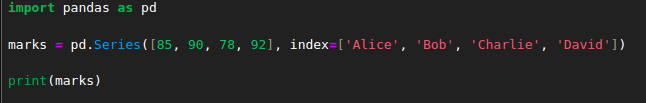
**Use Cases:**

Representing a single column of a dataset

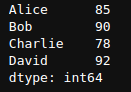
Storing and working with time-series data

Performing element-wise operations or statistical computations

**Example:**

****

**Output:**

****

Some of the functions and operations that can be performed in the series are,

**1.head() and tail()**

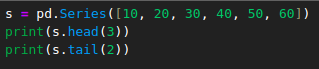
head(n) returns the first n elements.

tail(n) returns the last n elements.

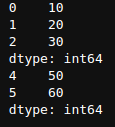
**Use Case:**

Used in data analysis to quickly view the start or end of a Series.

**Code Snippet:**

****

**Output:**

****

**Description:**

head(3) = [10, 20, 30]

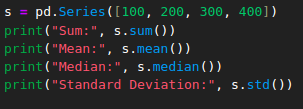
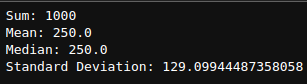
tail(2) = [50, 60]  
  
**2. sum(), mean(), median(), std()**

Statistical operations on numerical data.

**Use Case:**

Summarize financial data, sensor values, etc.

**Code Snippet:**

**  
  
  
  
  
Output:  
  
  
  
  
  
Description:**

sum() adds all values

mean() is the average

median() is the middle value

std() calculates variability  
  
**3. value\_counts()**

Counts the frequency of unique values.

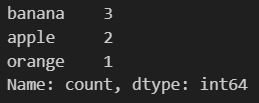
**Use Case:**

Used for categorical data analysis like user preferences, survey results.

**Code Snippet:**



**Output:**

****

**Description:**

value\_counts() counts the actual value of the elements.

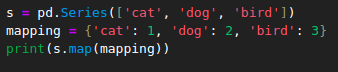
**4.map()**

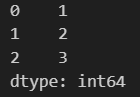
Transforms values using a mapping function or dictionary.

**Use Case:**

Label encoding or transforming categories.

**Code Snippet:**

**  
  
  
Output:**



**Description:**

map()-Maps string values to numbers using a dictionary.  
  
**Key Features:**

Automatically assigns an index if not provided

Supports NumPy-like operations

Can be sliced, filtered, and vectorized

Supports label-based indexing

**2. DataFrame**

A DataFrame is a two-dimensional tabular data structure. It contains:

Rows and columns

Each column is a Series

Columns can have different data types

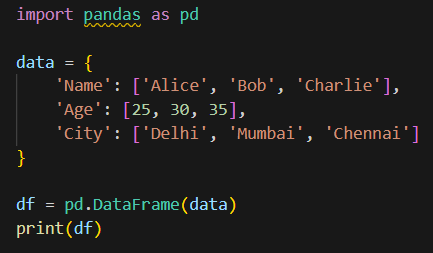
It’s similar to a table in a SQL database or a sheet in Excel.  
  
**Use Cases:**

Representing structured datasets (sales records, sensor data, customer info)

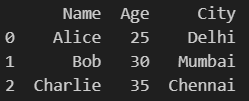
Cleaning and transforming raw datasets

Preparing data for visualization or machine learning

**Example:**

****

**Output:**

****

The important functions and operations that can be performed in the dataframe are,  
- Aggregation Functions

* Handling Missing Values

**1. Aggregation Functions in Pandas**

Aggregation is the process of combining or summarizing multiple values into one (like totals, averages, min, max). It’s especially powerful when used with groupby().

**Syntax:**



Commonly used aggregation functions are,

| **Function** | **Description** |  |
| --- | --- | --- |
| sum() | Total sum of values |  |
| mean() | Average value |  |
| median() | Median value |  |
| min() | Minimum value |  |
| max() | Maximum value |  |
| count() | Number of non-null values |  |

**Example:**

A computer screen shot of text

AI-generated content may be incorrect.

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Description:**

Sum : Total salary per department.

mean (Salary) : Average salary in each department.

max (Salary) : Highest salary in the department.

min : Lowest salary in the department.

std : Salary variation (standard deviation).

mean (Experience) : Average experience per department.

count : Number of employees.

max (Experience) : Most experienced employee.

**2. Handling Missing Values**

Real-world data often contains null/missing values. Pandas provides various methods to detect and handle them.

**A. isnull() and notnull()**

isnull() checks for missing values and returns True where data is missing.

notnull() returns True where data is not missing.

**Use Case:**

Check for null entries before data cleaning or processing.

**Code Snippet:**

A computer screen with text

AI-generated content may be incorrect.

**Output:**

A black background with white text

AI-generated content may be incorrect.

**Description:**

- Each NaN value is marked as True.

- Helps you identify where data is incomplete.

**B. fillna() – Fill Missing Values**

Fills missing data with a default value or computed statistic like the mean.

**Use Case:**

Replace missing names with 'Unknown' and missing ages with the average.

**Code Snippet:**



**Output:**

**A black background with white text

AI-generated content may be incorrect.**

**Description:**

- 'Name' column: missing entries are filled with 'Unknown'.

- 'Age' column: missing entries are filled with mean age ((25 + 30)/2 = 27.5).

**C. dropna() – Remove Missing Values**

Removes rows or columns that contain missing values.

**Use Case:**

Drop incomplete records to ensure data quality.

**Code Snippet:**

**A black background with white text

AI-generated content may be incorrect.**

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Description:**

- dropna() removes rows by default.

- axis=1 removes columns instead.

- Useful for eliminating data noise or errors.

**Library Comparison**

| **Feature** | **Matplotlib** | **Pandas** |
| --- | --- | --- |
| Ease of Use | Moderate (more code, more control) | Very easy (minimal code) |
| Customization | High (fine control over visuals) | Limited (inherits from Matplotlib) |
| Interactivity | Basic (static by default) | Limited (good for quick EDA) |
| Performance | Excellent for small/large datasets | Good for small to medium datasets |
| Best Use Case | Final production-quality plots | Fast plotting during data analysis |

**Comparison between matplotlib and pandas**

| **Feature** | **Matplotlib** | **Pandas** |
| --- | --- | --- |
| Main Purpose | Data visualization | Data manipulation & analysis |
| Type | Visualization library | Data handling library |
| Output | Charts and plots | Cleaned, structured data (plus charts) |
| Integrates with | Pandas, NumPy, Seaborn | Matplotlib, NumPy, scikit-learn |
| Common Use Case | Plotting data trends and summaries | Cleaning and analyzing datasets |