 Python libraries

NumPy

NumPy (Numerical Python) is a powerful library in Python used for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these data structures efficiently.

**Key Features of NumPy:**

1. **Ndarray (N-dimensional Array)** – Provides fast and efficient array operations.
2. **Mathematical Functions** – Includes functions for linear algebra, statistics, and Fourier transforms.
3. **Broadcasting** – Enables operations on arrays of different shapes.
4. **Indexing and Slicing** – Similar to Python lists but more powerful.
5. **Integration with Other Libraries** – Used in Pandas, Matplotlib, Scikit-learn, and TensorFlow.

### ****Why Use NumPy Instead of Python Lists?****

| **Feature** | **NumPy Arrays** | **Python Lists** |
| --- | --- | --- |
| Speed | Faster | Slower |
| Memory Usage | Less | More |
| Functionality | Supports mathematical operations | Limited operations |
| Broadcasting | Yes | No |

### ****Installing NumPy****

If you don’t have NumPy installed, you can install it using:

//In bash

pip install numpy

### ****Basic Operations in NumPy****

#### **1. Importing NumPy**

import numpy as np

#### **2. Creating Arrays**

//In python

# Creating a 1D array

arr1 = np.array([1, 2, 3, 4, 5])

print(arr1) #np.array() is a NumPy func used to create array

# Creating a 2D array

arr2 = np.array([[1, 2, 3], [4, 5, 6]])

print(arr2)

#### **3. Checking Array Shape and Type**

//In python

print(arr1.shape) # Output: (5,)

print(arr2.shape) # Output: (2,3)

print(arr1.dtype) # Output: int32 or int64 (depends on system)

#### **4. Creating Special Arrays**

//In python

# Array of zeros

zeros = np.zeros((3, 3))

print(zeros)

# Array of ones

ones = np.ones((2, 2))

print(ones)

# Identity matrix

identity = np.eye(3)

print(identity)

# Random numbers

random\_array = np.random.rand(3, 3)

print(random\_array)

#### **5. Array Operations**

//In python

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

# Element-wise addition

print(a + b) # Output: [5 7 9]

# Element-wise multiplication

print(a \* b) # Output: [4 10 18]

# Dot Product

print(np.dot(a, b)) # Output: 32 (1\*4 + 2\*5 + 3\*6)

#### **6. Indexing & Slicing**

//In python

arr = np.array([10, 20, 30, 40, 50])

print(arr[2]) # Output: 30

print(arr[1:4]) # Output: [20 30 40]

print(arr[::-1]) # Reverse array

#### **7. Reshaping Arrays**

//In python

arr = np.array([1, 2, 3, 4, 5, 6])

reshaped\_arr = arr.reshape(2, 3)

print(reshaped\_arr)

# Output:

[[ 1 2 3]

[4 5 6]]

#### **8. Statistical Functions**

//In python

arr = np.array([10, 20, 30, 40, 50])

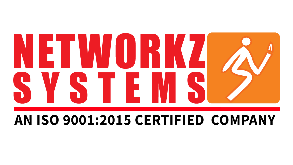
print(np.mean(arr)) # Mean: 30.0

print(np.median(arr)) # Median: 30.0

print(np.std(arr)) # Standard Deviation

print(np.sum(arr)) # Sum of elements: 150

print(np.min(arr)) # Minimum value: 10

print(np.max(arr)) # Maximum value: 50

### 9.Create a Random Array

//In python

rand\_arr = np.random.rand(4, 4) # 4x4 matrix with random values

print("Random Array:\n", rand\_arr)

### 10. Reshape an Array

//In python

arr = np.arange(1, 10) # Array from 1 to 9

reshaped = arr.reshape(3, 3) # Reshape into 3x3

print("Reshaped Array:\n", reshaped)

11. Matrix multiplication

import numpy as np

A = np.array([[1, 2], [3, 4]])

B = np.array([[5, 6], [7, 8]])

result = np.dot(A, B)

print("Matrix Multiplication Result:\n", result)

## 12. **Generating Random Data for Testing**

//In python

import numpy as np

# Generate random data (100 rows, 5 columns)

data = np.random.rand(100, 5)

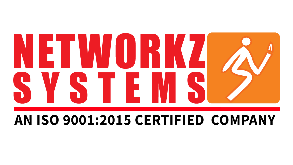
print(data[:5]) # Display first 5 rows

13. Convert a coloured image to numpy array

import numpy as np

from PIL import Image

# Load an image and convert to a numpy array

img = Image.open('sample.jpg')

img\_array = np.array(img)

# Convert to grayscale

gray = 0.2989 \* img\_array[:, :, 0] + 0.5870 \* img\_array[:, :, 1] + 0.1140 \* img\_array[:, :, 2]

// img\_array[:, :, 0] – Extracts the **Red** channel values

 img\_array[:, :, 1] – Extracts the **Green** channel values

 img\_array[:, :, 2] – Extracts the **Blue** channel values

# Convert back to an image and display

gray\_img = Image.fromarray(gray.astype('uint8'))

gray\_img.show()

**PANDAS**

The **Pandas** library in Python is used for data manipulation and analysis. It provides powerful data structures like **Series** and **DataFrame** to handle and analyze structured data efficiently.

**🔹 Installation**

Install Pandas using pip:

//in bash

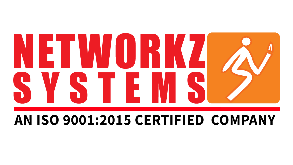
pip install pandas

**🔹 Main Data Structures**

1. **Series** – One-dimensional array-like object.
2. **DataFrame** – Two-dimensional table with rows and columns (like a spreadsheet).

**Creating a Series**

//In python

import pandas as pd

data = [10, 20, 30, 40]

series = pd.Series(data)

print(series)

**Output:**

0 10

1 20

2 30

3 40

dtype: int64

**Creating a DataFrame**

In **Pandas**, a **DataFrame** is a two-dimensional, tabular data structure with labeled rows and columns (like a spreadsheet or SQL table).

//In python

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

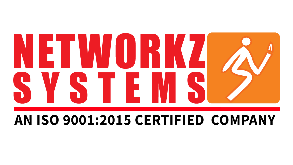
### Define Custom Index During DataFrame Creation

import pandas as pd

data = { 'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35] }

df = pd.DataFrame(data, index=['ID101', 'ID102', 'ID103'])

print(df)



**Reading/Writing Data**

* **Read from CSV:**

//In python

df = pd.read\_csv('data.csv')

Eg:

import pandas as pd

# Load CSV file (replace 'filename.csv' with your file path)

df = pd.read\_csv("C:\\Users\\user\\Downloads\\survey1.csv")

# Display first 5 rows

print(df.head())

* **Write to CSV:**

//In python

df.to\_csv('output.csv', index=False)

## **1. Write to CSV**

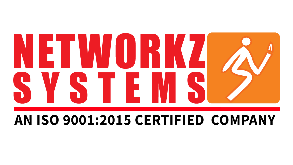
You can create a DataFrame and write it to a CSV file using to\_csv().

### ****Example: Write to CSV****

//In python

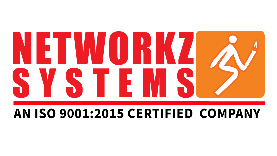
import pandas as pd

# Create a DataFrame

data = { 'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35],

'City': ['New York', 'Los Angeles', 'Chicago']}

df = pd.DataFrame(data)



# Write to CSV

df.to\_csv('data.csv', index=False)

print("Data written to data.csv")

**👉 Output:** A CSV file data.csv will be created with the following content:

Name,Age,City

Alice,25,New York

Bob,30,Los Angeles

Charlie,35,Chicago

**2. Read from CSV**

You can read a CSV file using read\_csv().

### ****Example: Read from CSV****

//In python

import pandas as pd

# Read CSV file

df = pd.read\_csv('data.csv')

# Display the data

print(df)

**👉 Output:**

Name Age City

0 Alice 25 New York

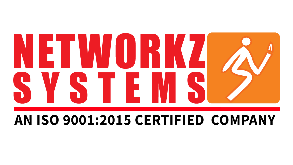
1 Bob 30 Los Angeles

2 Charlie 35 Chicago

## **3. Append New Data to CSV**

You can add new data to an existing CSV using mode='a' (append mode).

### ****Example: Append Data****

//In python

new\_data = {'Name': ['David'], 'Age': [40], 'City': ['Houston']}

new\_df = pd.DataFrame(new\_data)

# Append to existing CSV (without header)

new\_df.to\_csv('data.csv', mode='a', index=False, header=False)

print("Data appended to data.csv")

**👉 Output:** data.csv now contains

Name,Age,City

Alice,25,New York

Bob,30,Los Angeles

Charlie,35,Chicago

David,40,Houston

## **4. Write CSV Without Index**

You can remove the index while writing to CSV using index=False.

### ****Example: Write Without Index****

//In python

df.to\_csv('data.csv', index=False)

## **5. Read CSV with Specific Columns**

You can read only specific columns from a CSV file.

### ****Example: Read Specific Columns****

//In python

df = pd.read\_csv('data.csv', usecols=['Name', 'Age'])

print(df)

**👉 Output:**

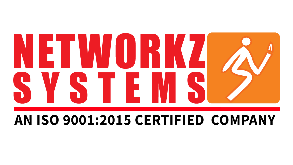
Name Age

0 Alice 25

1 Bob 30

2 Charlie 35

3 David 40

 **6. Read CSV Without Headers**

You can read CSV files without headers using header=None.

### ****Example: Read Without Headers****

//In python

df = pd.read\_csv('data.csv', header=None)

print(df)

**👉 Output:**

0 1 2

0 Name Age City

1 Alice 25 New York

2 Bob 30 Los Angeles

3 Charlie 35 Chicago

4 David 40 Houston

## **7. Save CSV with a Custom Separator**

You can change the separator (like ; or |) using the sep parameter.

### ****Example: Write with Custom Separator****

//In python

df.to\_csv('data.csv', sep='|', index=False)

**👉 Output:**

Name|Age|City

Alice|25|New York

Bob|30|Los Angeles

Charlie|35|Chicago

David|40|Houston

**Selecting Data**

* Select column:

//In python

print(df['Name'])

* Select row by index:

//In python

print(df.loc[0])

* Select row by index position:

// In python

print(df.iloc[0])

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| df.loc[0] |  | **Index label** | When index is 0 or you want to fetch row with label = 0 | |
| df.iloc[0] |  | **Row position** | Always fetches the first row, regardless of index |

**Filtering Data**

//In python

filtered = df[df['Age'] > 25]

print(filtered)

**Updating Data**

//In python

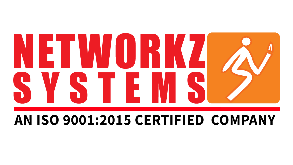
df.loc[0, 'Age'] = 28

**Adding New Column**

//In python

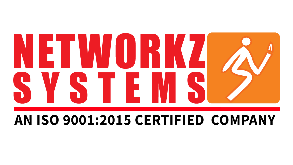
df['Country'] = 'USA'

**Deleting a Column**

//In python

df.drop('City', axis=1, inplace=True)

* axis=0 → rows
* axis=1 → columns

So here, axis=1 ensures 'City' is treated as a column name, not a row label.

**Descriptive Statistics**

//In python

print(df.describe())

**Handling Missing Data**

* Drop missing rows:

//In python

df.dropna(inplace=True)

Fill missing values:

//In python

df.fillna(0, inplace=True)

### ****Drop rows with missing values using**** dropna()

This program creates a DataFrame, introduces some missing values (NaN), and removes rows with missing values using dropna()

import pandas as pd

# Sample data with missing values

data = {'Name': ['John', 'Anna', 'Peter', 'Linda', None],

'Age': [28, 22, None, 32, 29],

'City': ['New York', 'Paris', 'London', None, 'Tokyo']}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

# Remove rows with missing values

df.dropna(inplace=True)

print("\nDataFrame after dropping rows with missing values:")

print(df)

**Output:**

Original DataFrame:

Name Age City

0 John 28.0 New York

1 Anna 22.0 Paris

2 Peter NaN London

3 Linda 32.0 None

4 None 29.0 Tokyo

DataFrame after dropping rows with missing values:

Name Age City

0 John 28.0 New York

1 Anna 22.0 Paris

### 2. ****Fill missing values using**** fillna()

This program creates a DataFrame and fills missing values using fillna().

import pandas as pd

# Sample data with missing values

data = {'Name': ['John', 'Anna', 'Peter', 'Linda', None],

'Age': [28, 22, None, 32, 29],

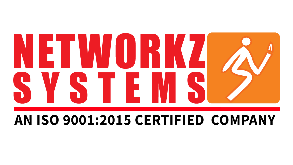
'City': ['New York', 'Paris', 'London', None, 'Tokyo']}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

# Fill missing values with default values

df.fillna({'Name': 'Unknown', 'Age': 0, 'City': 'Unknown'}, inplace=True)

print("\nDataFrame after filling missing values:")

print(df)

**Functions**

**1. Creating Data**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| pd.Series() | Creates a Pandas Series | s = pd.Series([1, 2, 3]) |
| pd.DataFrame() | Creates a DataFrame | df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]}) |
| pd.read\_csv() | Reads data from a CSV file | df = pd.read\_csv('data.csv') |
| pd.read\_excel() | Reads data from an Excel file | df = pd.read\_excel('data.xlsx') |
| pd.read\_json() | Reads data from a JSON file | df = pd.read\_json('data.json') |

🔹 **2. Viewing Data**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df.head() | Displays the first 5 rows | df.head() |
| df.tail() | Displays the last 5 rows | df.tail() |
| df.info() | Displays DataFrame info | df.info() |
| df.describe() | Displays statistical summary | df.describe() |
| df.shape | Returns dimensions of DataFrame | df.shape |
| df.columns | Returns column names | df.columns |

## 1. df.head() – Displays the first 5 rows

Returns the top 5 rows of the DataFrame. You can pass a number to display a specific number of rows.

### Example:

import pandas as pd

# Sample DataFrame

data = {

'Name': ['John', 'Anna', 'Mike', 'Tom', 'Sara', 'Alex'],

'Age': [25, 30, 22, 28, 24, 27],

'City': ['NY', 'LA', 'SF', 'TX', 'DC', 'CHI']

}

df = pd.DataFrame(data)

# Display first 5 rows

print(df.head())

### Output:

Name Age City

0 John 25 NY

1 Anna 30 LA

2 Mike 22 SF

3 Tom 28 TX

4 Sara 24 DC

👉 **You can display the first n rows using** df.head(n).

## 2. df.tail() – Displays the last 5 rows

Returns the bottom 5 rows of the DataFrame. You can pass a number to display a specific number of rows.

### Example:

# Display last 5 rows

print(df.tail())

### Output:

Name Age City

1 Anna 30 LA

2 Mike 22 SF

3 Tom 28 TX

4 Sara 24 DC

5 Alex 27 CHI

👉 **You can display the last n rows using** df.tail(n).

## 3. df.info() – Displays DataFrame information

Returns a summary of the DataFrame, including:  
✅ Number of rows and columns  
✅ Column names and data types  
✅ Non-null values in each column

### Example:

# Display DataFrame information

df.info()

### Output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 6 entries, 0 to 5

Data columns (total 3 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Name 6 non-null object

1 Age 6 non-null int64

2 City 6 non-null object

dtypes: int64(1), object(2)

memory usage: 272.0 bytes

👉 **Useful for checking data types, missing values, and data structure.**

🔹 **3. Selecting Data**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df['column'] | Select a column | df['Age'] |
| df[['col1', 'col2']] | Select multiple columns | df[['Age', 'Name']] |
| df.loc[] | Select rows by label | df.loc[0] |
| df.iloc[] | Select rows by index | df.iloc[0] |
| df.at[] | Fast access to a single value by label | df.at[0, 'Age'] |
| df.iat[] | Fast access to a single value by index | df.iat[0, 1] |

🔹 **4. Filtering Data**

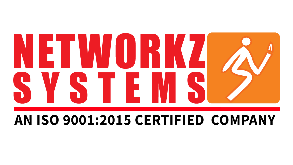
| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df[df['Age'] > 25] | Filter rows based on a condition | df[df['Age'] > 25] |
| df.query() | Query data | df.query('Age > 25') |
| df['column'].isin() | Filter by values in a list | df[df['City'].isin(['New York', 'Los Angeles'])] |

🔹 **5. Adding and Removing Data**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df['new\_col'] = value | Add a new column | df['Country'] = 'USA' |
| df.drop() | Remove columns or rows | df.drop('Age', axis=1) |
| df.insert() | Insert column at specific position | **Program using insert function**  import pandas as pd  # Create a sample DataFrame  data = {  'Name': ['John', 'Alice', 'Bob'],  'Age': [25, 30, 22]  }  df = pd.DataFrame(data)  print("Original DataFrame:")  print(df)  # Insert a new column at index 1 (after 'Name')  df.insert(1, 'City', ['New York', 'Los Angeles', 'Chicago'])  print("\nDataFrame after inserting 'City' column:")  print(df) |
| df.append() | Append rows | df.append(new\_row) |
| df.rename() | Rename columns | df.rename(columns={'A': 'Alpha'}) |

🔹 **6. Handling Missing Data**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df.isnull() | Check for missing values | df.isnull() |
| df.dropna() | Drop rows with missing values | df.dropna() |
| df.fillna() | Fill missing values | df.fillna(0) |
|  |  |  |



🔹 **7. Grouping and Aggregation**

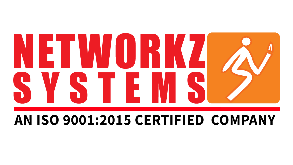
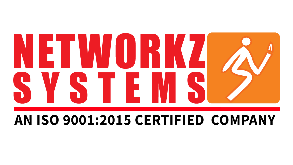
| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df.groupby() | Group data | df.groupby('City').mean() |
| df.agg() | Aggregate data | df.agg({'Age': 'mean'}) |
| df.pivot\_table() | Create a pivot table | df.pivot\_table(index='City', values='Age', aggfunc='mean') |
| df.cumsum() | Cumulative sum | df['Age'].cumsum() |

🔹 **8. Sorting and Ranking**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| df.sort\_values() | Sort by values | df.sort\_values(by='Age') |
| df.sort\_index() | Sort by index | df.sort\_index() |
| df.rank() | Rank values | df['Age'].rank() |

🔹 **9. Merging and Joining**

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| pd.concat() | Concatenate DataFrames | pd.concat([df1, df2]) |
| df.merge() | Merge DataFrames | df1.merge(df2, on='ID') |
| df.join() | Join DataFrames | df1.join(df2, on='ID') |
| 1. pd.concat() – Concatenate DataFrames Used to combine two or more DataFrames along rows (axis=0) or columns (axis=1). Example: import pandas as pd  # Create two sample DataFrames  data1 = {'ID': [1, 2, 3], 'Name': ['John', 'Anna', 'Mike']}  data2 = {'ID': [4, 5, 6], 'Name': ['Tom', 'Sara', 'Alex']}  df1 = pd.DataFrame(data1)  df2 = pd.DataFrame(data2)  # Concatenate along rows (default axis=0)  result = pd.concat([df1, df2])  print(result) Output: ID Name  0 1 John  1 2 Anna  2 3 Mike  0 4 Tom  1 5 Sara  2 6 Alex  **Note:** Indexes are not reset after concatenation. 2. merge() – Merge DataFrames based on a common columnUsed to combine DataFrames like an SQL JOIN based on common keys. Example:data1 = {'ID': [1, 2, 3], 'Name': ['John', 'Anna', 'Mike']}data2 = {'ID': [1, 2, 4], 'Age': [25, 30, 28]}df1 = pd.DataFrame(data1)df2 = pd.DataFrame(data2)# Merge on 'ID' column (INNER JOIN)result = df1.merge(df2, on='ID')print(result)Output:ID Name Age0 1 John 251 2 Anna 30\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3. join() – Join DataFrames based on the indexUsed to combine DataFrames on the index or a key column.Example:data1 = {'Name': ['John', 'Anna', 'Mike'],‘Age':[ 25, 30, 22]}data2 = {'City': ['NY', 'LA', 'SF']}df1 = pd.DataFrame(data1, index=[1, 2, 3])df2 = pd.DataFrame(data2, index=[1, 2, 3])# Join on indexresult = df1.join(df2)print(result)Output: Name Age City  1 John 25 NY  2 Anna 30 LA  3 Mike 22 SF  **Note:** join() is similar to merge() but works based on index by default.  Use on='column' to join on a specific column instead of the index. |  |  |

****  


**MATPLOTLIB**

**Matplotlib** is a popular Python library used for data visualization. It allows you to create a wide range of plots, including line plots, bar plots, scatter plots, histograms, and more.

Install matplotlib using pip:

pip install matplotlib

1.Example of a line plot using matplotlib

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [2, 4, 1, 3, 5]

# Create plot

plt.plot(x, y)

# Add title and labels

plt.title("Simple Line Plot")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.show()

## **Common Plot Types with Examples**

### 1. ****Line Plot****

Used to visualize trends over time or continuous data.

plt.plot([1, 2, 3, 4], [10, 20, 25, 30], marker='o', linestyle='--', color='b')

plt.title('Line Plot Example')

plt.show()

### 2. ****Bar Plot****

Used to compare categorical data.

categories = ['A', 'B', 'C', 'D']

values = [4, 7, 1, 8]

plt.bar(categories, values, color='skyblue')

plt.title('Bar Plot Example')

plt.show()

### 3. ****Scatter Plot****

Used to show the relationship between two variables

x = [5, 7, 8, 7, 2, 17, 2, 9]

y = [99, 86, 87, 88, 100, 86, 103, 87]

plt.scatter(x, y, color='red')

plt.title('Scatter Plot Example')

plt.show()

### 4. ****Histogram****

Used to show the distribution of data.

//In python

data = [22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27]

plt.hist(data, bins=5, color='green', edgecolor='black')

plt.title('Histogram Example')

plt.show()

### 5. ****Pie Chart****

Used to show proportions.

//In python

labels = ['Python', 'Java', 'C++', 'JavaScript']

sizes = [45, 25, 15, 15]

colors = ['blue', 'red', 'yellow', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)

plt.title('Pie Chart Example')

plt.show()

### 6. ****Multiple Plots in One Figure****

Use plt.subplot() to create multiple plots in one figure.

//In python

# First plot

plt.subplot(1, 2, 1)

plt.plot([1, 2, 3], [4, 5, 6])

plt.title('Plot 1')

# Second plot

plt.subplot(1, 2, 2)

plt.bar(['A', 'B', 'C'], [3, 5, 7])

plt.title('Plot 2')

plt.tight\_layout()

plt.show()

 plt.subplot(1, 2, 1): Creates a 1-row, 2-column grid and activates the first subplot.

 plt.subplot(1, 2, 2): Activates the second subplot in the same grid.

 plt.tight\_layout(): Prevents overlapping of titles or axes.

## **Customization Tips**

* **Color:** 'red', 'blue', '#ff5733'
* **Line style:** '-', '--', '-.', ':'
* **Markers:** 'o', '^', 's', 'd'
* **Labels:** plt.xlabel(), plt.ylabel(), plt.title()
* **Legend:** plt.legend()
* **Grid:** plt.grid()
* **Figure size:** plt.figure(figsize=(10, 5))

**Seaborn**

Seaborn is a Python data visualization library built on top of **Matplotlib** and closely integrated with **Pandas**. It provides a high-level interface for drawing attractive and informative statistical graphics.

 Simplifies complex visualizations.

 Beautiful default themes.

 Integrates well with Pandas data structures.

 Provides statistical plots like **histograms, KDE plots, pair plots, violin plots**, etc.

### ****Installation****

pip install seaborn

### ****Basic Usage****

//In python

import seaborn as sns

import matplotlib.pyplot as plt

# Load a sample dataset

tips = sns.load\_dataset("tips")

# Create a scatter plot

sns.scatterplot(x="total\_bill", y="tip", data=tip s)

# Show the plot

plt.show()

### 🔹 ****Common Seaborn Plots****

| **Plot Type** | **Function Name** |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scatter Plot | sns.scatterplot() |  |  |  |  |
| Line Plot | sns.lineplot() |  |  |  |  |
| Histogram | sns.histplot() |  |  |  |  |
| KDE Plot | sns.kdeplot() |  |  |  |  |
| Box Plot | sns.boxplot() |  |  |  |  |
| Violin Plot | sns.violinplot() |  |  |  |  |
| Heatmap | sns.heatmap() |  |  |  |  |
| Pair Plot | sns.pairplot() |  |  |  |  |

Histogram Plot

import seaborn as sns

import matplotlib.pyplot as plt

# Load dataset

tips = sns.load\_dataset("tips")

# Histogram plot without hue

sns.histplot(data=tips, x="total\_bill", kde=True, bins=20)

# Show the plot

plt.show()

#### **KDE Plot (**sns.kdeplot()**)**

KDE (**Kernel Density Estimation**) plot shows the **probability density function** of a continuous variable. It is like a smooth version of a histogram.

//In python

import seaborn as sns

import matplotlib.pyplot as plt

# Load sample dataset

tips = sns.load\_dataset("tips")

# KDE plot for total\_bill

sns.kdeplot(x=tips["total\_bill"], fill=True)

plt.show()

#### **Violin Plot (**sns.violinplot()**)**

A violin plot combines a **box plot and KDE plot** to show the distribution of data across different categories.

//In python

sns.violinplot(x="day", y="total\_bill", data=tips)

plt.show()

#### **Pair Plot (**sns.pairplot()**)**

A pair plot visualizes pairwise relationships between multiple numeric variables.

//In python

sns.pairplot(tips, hue="sex")

### ****SciPy (Scientific Python Library)****

**SciPy** is a powerful Python library built on top of **NumPy**, used for **scientific computing, mathematics, and engineering**. It provides efficient implementations of **linear algebra, optimization, signal processing, integration, and more**.

### 🔹 ****Installation****

pip install scipy

### 🔹 ****Key Features of SciPy****

SciPy contains different modules specialized for specific scientific computations:

| **Module** | **Purpose** |
| --- | --- |
| scipy.linalg | Linear algebra (matrix operations, determinants, etc.) |
| scipy.stats | Statistical functions (mean, median, normal distribution, etc.) |
| scipy.integrate | Numerical integration (solving integrals, ODEs) |
| scipy.optimize | Optimization algorithms (finding minima/maxima) |
| scipy.spatial | Spatial data structures (distance calculations, clustering) |
| scipy.signal | Signal processing (filters, FFT) |
| scipy.fft | Fast Fourier Transform (frequency analysis) |

## **SciPy for Image Processing**

SciPy provides basic image processing functions using scipy.ndimage.

### ****Blur Image Using Gaussian Filter****

from scipy.ndimage import gaussian\_filter

import imageio.v3 as imageio

import matplotlib.pyplot as plt

# Load image

image = imageio.imread("image.jpg")

# Apply Gaussian blur

blurred = gaussian\_filter(image, sigma=2)

# Show images

plt.subplot(1, 2, 1)

plt.imshow(image)

plt.title("Original")

plt.subplot(1, 2, 2)

plt.imshow(blurred)

plt.title("Blurred")

plt.show()

Program using different libraries together:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Generate Random Data

np.random.seed(42) # For reproducibility

data = {

'Age': np.random.randint(18, 60, 100), # Random ages between 18-60

'Salary': np.random.randint(30000, 100000, 100), # Random salaries between 30k-100k

'Experience': np.random.randint(0, 40, 100) # Random work experience (0-40 years)

}

# Step 2: Create Pandas DataFrame

df = pd.DataFrame(data)

# Step 3: Basic Statistical Analysis

print("Data Summary:\n", df.describe())

# Step 4: Data Visualization

# Histogram of Ages

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.hist(df['Age'], bins=10, color='blue', alpha=0.7)

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.title('Age Distribution')

# Scatter Plot of Salary vs Experience

plt.subplot(1, 2, 2)

plt.scatter(df['Experience'], df['Salary'], color='red', alpha=0.6)

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Salary vs Experience')

# Show the plots

plt.tight\_layout()

plt.show()

### ****Creating a Pandas DataFrame from a NumPy Array****

//In python

import numpy as np

import pandas as pd

# Create a NumPy array

data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Convert it into a Pandas DataFrame

df = pd.DataFrame(data, columns=['A', 'B', 'C'])

print(df)

### **#Adding a Column Using NumPy Operations**

df['D'] = np.sum(df, axis=1) # Sum of each row

print(df)

**#**Calculating column-wise sum and adding a new row

df.loc['Sum'] = np.sum(df, axis=0)

print("\nDataFrame after adding column-wise sum:")

print(df)

**Dataframe operations using numpy array**

import numpy as np

import pandas as pd

# Create a NumPy array

data = np.random.randint(1, 100, size=(5, 3))

# Create a Pandas DataFrame using the NumPy array

df = pd.DataFrame(data, columns=['A', 'B', 'C'])

# Display the DataFrame

print("Original DataFrame:")

print(df)

# Perform NumPy operations (e.g., adding 10 to each element in the DataFrame)

df = df + 10

# Display the modified DataFrame

print("\nModified DataFrame (after adding 10 to each element):")

print(df)

# Calculate the mean of each column using NumPy functions

column\_means = np.mean(df, axis=0)

print("\nMean of each column:")

print(column\_means)

**Example 2:Custom row selection**

import numpy as np

import pandas as pd

# Create a 2D NumPy array

data = np.array([[12, 15, 18],[22, 25, 28],[32, 35, 38]])

# Create a DataFrame with custom row indices

df = pd.DataFrame(data, columns=['A', 'B', 'C'], index=['row1', 'row2', 'row3'])

print("DataFrame with custom indices:")

print(df)

# Accessing a row using a custom index

row2 = df.loc['row2']

print("\nRow 2 data:")

print(row2)

**Example 4:Filter Method**

import numpy as np

import pandas as pd

# Create a DataFrame with random data

data = np.random.rand(10, 3)

df = pd.DataFrame(data, columns=['A', 'B', 'C'])

# Add a 'Category' column based on a condition

df['Category'] = np.where(df['A'] > 0.5, 'High', 'Low')

print("DataFrame with a new Category column:")

print(df)

# Group by the 'Category' column and calculate the mean of each group

grouped = df.groupby('Category').mean()

print("\nGrouped Data by Category (mean of each column):")

print(grouped)

## **Creating and Visualizing a DataFrame**

python

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Generate a NumPy array with random values

data = np.random.randint(1, 100, size=(10, 3))

# Convert NumPy array into a Pandas DataFrame

df = pd.DataFrame(data, columns=['A', 'B', 'C'])

print("DataFrame:\n", df)

# Plot the data

df.plot(kind='bar', figsize=(8, 5))

plt.title("Bar Plot of Random Data")

plt.xlabel("Index")

plt.ylabel("Values")

plt.legend()

plt.show()

## **2. Scatter Plot with Random Data**

Python

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Generate random data

x = np.random.randn(100)

y = np.random.randn(100)

# Create a Pandas DataFrame

df = pd.DataFrame({'X': x, 'Y': y})

# Scatter plot

plt.scatter(df['X'], df['Y'], color='blue', alpha=0.5)

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.title("Scatter Plot of Random Data")

plt.grid()

plt.show()

## **3. Histogram of Randomly Generated Data**

python

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Generate random normal distribution data

data = np.random.randn(1000)

# Create Pandas DataFrame

df = pd.DataFrame({'Values': data})

# Plot histogram

df['Values'].hist(bins=30, color='green', edgecolor='black', alpha=0.7)

plt.xlabel("Value Range")

plt.ylabel("Frequency")

plt.title("Histogram of Random Data")

plt.show()

**Scikit-LEARN(sklearn)**

Scikit-Learn (sklearn) is one of the most popular Python libraries for machine learning. It provides simple and efficient tools for data preprocessing, classification, regression, clustering, and more.

## **Step 1: Install Scikit-Learn**

Make sure you have scikit-learn installed:

pip install scikit-learn

## **Step 2: Basic Hands-on Exercises**

Here are some beginner-friendly exercises using Scikit-Learn.

### ****1. Load a Dataset (Iris Dataset)****

Scikit-Learn provides several built-in datasets. Let’s load the famous **Iris dataset**.

python

from sklearn.datasets import load\_iris

import pandas as pd

# Load dataset

iris = load\_iris()

# Convert to DataFrame

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['target'] = iris.target

print(df.head())

* load\_iris() returns a **dictionary-like object** known as a Bunch object.
* It contains the following main attributes:
  + iris.data: the numeric data of the features (e.g., petal length, sepal width)
  + iris.target: the class labels (0, 1, or 2)
  + iris.feature\_names: list of column names for the features
  + iris.target\_names: array of string names for each class ('setosa', 'versicolor', 'virginica')

Then adds a new column to the DataFrame named 'target'.

Each value in this column corresponds to the class label:

* 0: Setosa
* 1: Versicolor
* 2: Virginica

You're loading a famous dataset for classification (Iris).

Converting it into a pandas DataFrame for analysis.

Adding the target labels to the DataFrame.

Displaying the first few rows to inspect the data.