1. **Creating a NumPy Array**
   1. **Basic ndarray**
   2. **Array of zeros**
   3. **Array of ones**
   4. **Random numbers in ndarray**
   5. **An array of your choice**
   6. **Imatrix in NumPy**
   7. **Evenly spaced ndarray**

**Aim:** To understand the creation of various types of NumPy arrays and their applications as:

* 1. Basic ndarray
  2. Array of zeros
  3. Array of ones
  4. Random numbers in ndarray
  5. An array of your choice
  6. Imatrix in NumPy
  7. Evenly spaced ndarray

**Description:**

1. **Basic ndarray:** A fundamental multi-dimensional array in NumPy, created using the np.array() method. The values can be manually specified.
2. **Array of Zeros:** A NumPy array filled with zeros, created using np.zeros() function. Useful for initializing arrays with default zero values. The shape and data type can be specified.
3. **Array of Ones:** A NumPy array filled with ones, created using np.ones() function. Useful for initializing arrays with all values as one. The shape and data type can also be defined.
4. **Random Numbers in ndarray:** A NumPy array filled with random floating-point numbers, generated using np.random.random(). Often used in simulations or testing with random data.
5. **Array of Your Choice:** An array created with custom values, using np.array(). Can be used for specific scenarios where pre-defined data is required.
6. **Identity Matrix (Imatrix):** A square matrix with ones on the main diagonal and zeros elsewhere, created using np.eye(). Commonly used in linear algebra and mathematical computations.
7. **Evenly Spaced ndarray:** An array with evenly spaced values, created using np.arange(start, stop, step). Useful for generating sequences of numbers in a defined range.

**Source Code:**

import numpy as np

**#Basic ndarray**

basicarray = np.array([10, 15, 20, 25, 30])

print("Basic ndarray:")

print(basicarray)

**#Array of zeros**

zeroarray = np.zeros((3, 3),dtype=int)

print("Array of zeros:")

print(zeroarray)

**#Array of ones**

onesarray = np.ones((3, 3),dtype=int)

print("Array of ones:")

print(onesarray)

**#Random numbers in ndarray**

randomarray = np.random.random((3, 3))

print("Random numbers in ndarray:")

print(randomarray)

**#An array of your choice**

choicearray = np.array([[10, 20, 30], [40, 50, 60]])

print("An array of your choice:")

print(choicearray)

**#Imatrix in NumPy**

identitymatrix = np.eye(3,dtype=int)

print("Imatrix in Numpy:")

print(identitymatrix)

**#Evenly spaced ndarray**

evenlyspaced = np.arange(0, 10, 3)

print("Evenly spaced ndarray:")

print(evenlyspaced)

**Output:**

Basic ndarray:

[10 15 20 25 30]

Array of zeros:

[[0 0 0]

[0 0 0]

[0 0 0]]

Array of ones:

[[1 1 1]

[1 1 1]

[1 1 1]]

Random numbers in ndarray:

[[0.86201768 0.1958278 0.37242774]

[0.78261564 0.60039726 0.48029583]

[0.58531621 0.41428205 0.8696366 ]]

An array of your choice:

[[10 20 30]

[40 50 60]]

Imatrix in Numpy:

[[1 0 0]

[0 1 0]

[0 0 1]]

Evenly spaced ndarray:

[0 3 6 9]

**2. The Shape and Reshaping of NumPy Array**

**a) Dimensions of NumPy array**

**b) Shape of NumPy array**

**c) Size of NumPy array**

**d) Reshaping a NumPy array**

**e) Flattening a NumPy array**

**f) Transpose of a NumPy array**

**AIM:** To explore the shape, size, dimensions, and transformation of NumPy arrays using reshaping, flattening, and transposing techniques.

**Description:**

1. **Dimensions of a NumPy Array (ndim):** Displays the number of dimensions (axes) of a NumPy array. Example: A 2D array has 2 dimensions.
2. **Shape of a NumPy Array (shape):** Returns the structure of the array as a tuple indicating the number of elements along each axis. Example: An array with 2 rows and 4 columns has a shape (2, 4).
3. **Size of a NumPy Array (size):** Represents the total number of elements in the array by multiplying the elements of the shape tuple. Example: For a (2, 4) array, the size is 8.
4. **Reshaping a NumPy Array (reshape):** Alters the structure of an array into a new shape without modifying its data. The new shape must have the same number of elements as the original array. Example: A (2, 4) array can be reshaped into (4, 2).
5. **Flattening a NumPy Array (flatten):** Converts a multi-dimensional array into a one-dimensional array. This is useful for simplifying data for certain operations.
6. **Transpose of a NumPy Array (transpose):** Swaps the rows and columns of an array. For a 2D array, it flips the array along its diagonal. Example: For a (2, 4) array, the transpose results in a (4, 2) array.

**Source Code:**

import numpy as np

**#Dimensions of NumPy array**

a = np.array([[10, 15, 20, 25], [30, 35, 40,45]])

print("Dimensions of NumPy array:")

print(a.ndim)

**#Shape of NumPy array**

print("enter the shape of numpy array:")

print(a.shape)

**#Size of NumPy array**

print("enter the size of Numpy array:")

print(a.size)

**#Reshaping a NumPy array**

reshapearray = a.reshape(4, 2)

print("Reshaping a Numpy array:")

print(reshapearray)

**#Flattening a NumPy array**

flattenarray = a.flatten()

print("Flattening a Numpy array:")

print(flattenarray)

**#Transpose of a NumPy array**

transposearray = a.transpose()

print("Transpose of a Numpy array:")

print(transposearray)

**Output:**

Dimensions of NumPy array:

2

enter the shape of numpy array:

(2, 4)

enter the size of Numpy array:

8

Reshaping a Numpy array:

[[10 15]

[20 25]

[30 35]

[40 45]]

Flattening a Numpy array:

[10 15 20 25 30 35 40 45]

Transpose of a Numpy array:

[[10 30]

[15 35]

[20 40]

[25 45]]

**3. a) Write a Python Program for Expanding a NumPy array**

**Aim:** To write a Python Program for expanding a NumPy array

**Description:** The program demonstrates how to expand a NumPy array by adding a new axis using the np.expand\_dims() function. Initially, a one-dimensional array [100, 200, 300] is created. The np.expand\_dims() function is then used to add a new axis to the array. When axis=0, the array is expanded into a two-dimensional row vector. When axis=1, the array is expanded into a two-dimensional column vector. This method is useful for reshaping arrays to match specific dimensions for computations or operations.

**Source Code:**

import numpy as np

**# Array**

array = np.array([100, 200, 300])

print("Array:", array)

**#Expanding a NumPy array**

**# Adding a new axis at position 0**

expandarray = np.expand\_dims(array, axis=0)

print("Expanded array (axis=0):\n", expandarray)

**# Adding a new axis at position 1**

expandarray1 = np.expand\_dims(array, axis=1)

print("Expanded array (axis=1):\n", expandarray1)

**Output:**

Array: [100 200 300]

Expanded array (axis=0):

[[100 200 300]]

Expanded array (axis=1):

[[100]

[200]

[300]]

**3. b) Write a python program for Squeezing a NumPy array**

**Aim:** To write a Python Program for Squeezing a NumPy array

**Description:** The program demonstrates how to simplify the dimensions of a NumPy array using the np.squeeze() function. Initially, a 4-dimensional array with the shape (1, 3, 1, 4) is created. This array contains nested lists, with axes of size 1 in the first and third dimensions. The np.squeeze() function is applied to remove these single-dimensional axes, resulting in a new array with the shape (3, 4). The squeezed array is now two-dimensional, retaining the same data as the original array but in a more compact form. The program prints both the original and squeezed shapes, as well as the contents of the squeezed array. This operation is useful in scenarios where reducing unnecessary dimensions simplifies computations or improves compatibility with other functions. The data itself remains unaffected, while the array's structure becomes more efficient.

**Source Code:**

import numpy as np

**# Create a NumPy array with shape (1, 3, 1, 4)**

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

[[50, 60, 70, 80]],

[[90, 100, 110, 120]]]])

print("Original shape:", array.shape)

**# Squeeze the array**

squeezedarray = np.squeeze(array)

print("Squeezed shape:", squeezedarray.shape)

print(squeezedarray)

**Output:**

Original shape: (1, 3, 1, 4)

Squeezed shape: (3, 4)

[[ 10 20 30 40]

[ 50 60 70 80]

[ 90 100 110 120]]

**3. c) Write a python program to illustrate Sorting in NumPy Arrays**

**Aim:** To write a Python Program to illustrate Sorting in NumPy arrays.

**Description:** The program demonstrates how to sort elements in a NumPy array along different axes using the np.sort() function. A 2D array is created with the shape (2, 3) containing two rows and three columns. The array is first sorted along the first axis (axis=0), which means sorting the elements within each column. The result is a new array where the values in each column are sorted in ascending order. Then, the array is sorted along the second axis (axis=1), which means sorting the elements within each row. This results in a new array where the values in each row are sorted in ascending order. The program outputs both the column-wise sorted and row-wise sorted arrays, illustrating how sorting works along different dimensions in NumPy arrays.

**Source Code:**

import numpy as np

**# Create a 2D NumPy array**

array = np.array([[30, 10, 20], [50, 40, 60]])

**# Sort along the first axis (columns)**

sortedarray = np.sort(array, axis=0)

print("Sorted along axis 0 (columns):\n", sortedarray)

**# Sort along the second axis (rows)**

sortedarray = np.sort(array, axis=1)

print("Sorted along axis 1 (rows):\n", sortedarray)

**Output:**

Sorted along axis 0 (columns):

[[30 10 20]

[50 40 60]]

Sorted along axis 1 (rows):

[[10 20 30]

[40 50 60]]

**4. a) Write a Python Program for illustrating Slicing 1-D NumPy arrays**

**Aim:** To write a Python Program for illustrating Slicing 1-D NumPy arrays

**Description:** This program demonstrates slicing operations on a 1-D NumPy array, allowing for the selection of specific subsets of elements using indexing ranges and steps. A 1-D array named array1d is created with values from 10 to 100. The slicing operations are then performed as follows:

1. array1d[2:7]: Extracts elements from index 2 to index 6 (exclusive of 7).
2. array1d[:5]: Extracts the first 5 elements (from the start to index 4).
3. array1d[5:]: Extracts all elements from index 5 to the end of the array.
4. array1d[::2]: Extracts every second element from the entire array (step size = 2).
5. array1d[1::2]: Extracts every second element starting from index 1.

The program prints the sliced subsets for each operation, illustrating how slicing can efficiently extract data subsets without the need for loops.

**Source code:**

import numpy as np

array1d = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

print("array1d[2:7]:")

print(array1d[2:7])

print("array1d[:5]:")

print(array1d[:5])

print("array1d[5:]:")

print(array1d[5:])

print("array1d[::2]:")

print(array1d[::2])

print("array1d[1::2]:")

print(array1d[1::2])

**Output:**

array1d[2:7]:

[30 40 50 60 70]

array1d[:5]:

[10 20 30 40 50]

array1d[5:]:

[ 60 70 80 90 100]

array1d[::2]:

[10 30 50 70 90]

array1d[1::2]:

[ 20 40 60 80 100]

**4. b) Write a Python Program for illustrating Slicing 2-D NumPy arrays**

**Aim:** To write a Python Program for illustrating Slicing 2-D NumPy arrays

**Description:** This program demonstrates how to perform slicing operations on a 2-D NumPy array. A 2-D array named array2d is created with shape (4, 4), containing integers arranged in a grid. Slicing is used to extract specific subarrays based on row and column indices:

1. **array2d[1:3, 1:3]**: Extracts the elements from rows 1 to 2 (excluding row 3) and columns 1 to 2 (excluding column 3). The result is a subarray from the middle of the 2-D array.
2. **array2d[:2, :2]**: Extracts the elements from the first two rows (row indices 0 and 1) and the first two columns (column indices 0 and 1), forming the top-left subarray.
3. **array2d[2:, 2:]**: Extracts the elements from rows 2 to the end and columns 2 to the end, forming the bottom-right subarray.

The program demonstrates how slicing works for both rows and columns, allowing for efficient extraction of specific regions of a 2-D array.

**Source Code:**

import numpy as np

array2d = np.array( [ [5, 10, 15, 20],

[25, 30, 35, 40],

[45, 50, 55, 60],

[65, 70, 75, 80] ] )

print("array2d[1:3, 1:3]:")

print(array2d[1:3, 1:3])

print("array2d[:2, :2]:")

print(array2d[:2, :2])

print("array2d[2:, 2:]:")

print(array2d[2:, 2:])

**Output:**

array2d[1:3, 1:3]:

[[30 35]

[50 55]]

array2d[:2, :2]:

[[ 5 10]

[25 30]]

array2d[2:, 2:]:

[[55 60]

[75 80]]

**4. c) Write a Python Program for illustrating Slicing 3-D NumPy arrays**

**Aim:** To write a Python Program for illustrating Slicing 3-D NumPy arrays

**Description:** This program illustrates how to perform slicing operations on a 3-D NumPy array. A 3-D array array3d is created with shape (3, 3, 3), containing three 3x3 matrices. Slicing is used to extract specific subarrays based on the three dimensions (depth, rows, and columns):

1. **array3d[0:2, 1:3, 1:3]**: Extracts elements from the first two matrices (depth 0 and 1), rows 1 to 2 (excluding 3), and columns 1 to 2 (excluding 3). This results in a 2x2x2 subarray from the first two 3x3 matrices.
2. **array3d[:, 1, :]**: Extracts the entire second row (row index 1) from all three matrices, keeping all columns. This returns a 3x3 subarray representing the second row of each matrix.
3. **array3d[:, :, 1:3]**: Extracts all rows and columns 1 to 2 (excluding 3) from each matrix. This results in a 3x2 subarray for each of the three matrices, focusing on columns 1 and 2.

This program demonstrates how to slice 3-D arrays along different axes to extract specific parts of the array efficiently. It helps to understand how the three dimensions (depth, rows, columns) interact when slicing in NumPy.

**Source code:**

import numpy as np

array3d = np.array([[[ 2, 4, 6],

[ 8, 10, 12],

[ 14, 16, 18]],

[[ 20, 22, 24],

[26, 28, 30],

[32, 34, 36]],

[[38, 40, 42],

[44, 46, 48],

[50, 52, 54]]])

print("array3d[0:2, 1:3, 1:3]:")

print(array3d[0:2, 1:3, 1:3])

print("array3d[:, 1, :]:")

print(array3d[:, 1, :])

print("array3d[:, :, 1:3]:")

print(array3d[:, :, 1:3])

**Output:**

array3d[0:2, 1:3, 1:3]:

[[[10 12]

[16 18]]

[[28 30]

[34 36]]]

array3d[:, 1, :]:

[[ 8 10 12]

[26 28 30]

[44 46 48]]

array3d[:, :, 1:3]:

[[[ 4 6]

[10 12]

[16 18]]

[[22 24]

[28 30]

[34 36]]

[[40 42]

[46 48]

[52 54]]]

**4. d) Write a Python Program for illustrating Negative slicing of NumPy arrays**

**Aim:** To write a Python Program for illustrating Negative slicing of NumPy arrays

**Description:** This program demonstrates how to perform **negative slicing** in NumPy arrays. Negative slicing allows you to slice an array from the end, which is useful when you want to extract elements from the back without knowing the array's length. In this program, three different arrays (1-D, 2-D, and 3-D) are sliced using negative indices:

1. **array1d[-5:]**: Extracts the last five elements from the 1-D array array1d. Negative indexing starts counting from the end, so -5: grabs the last five elements of the array.
2. **array1d[:-5]**: Extracts all elements up to the fifth-to-last element. Since -5 refers to the fifth element from the end, [:-5] slices the array from the beginning to that point.
3. **array2d[-2:, -2:]**: In the 2-D array array2d, -2: selects the last two rows, and -2: selects the last two columns. This gives a 2x2 subarray from the bottom-right corner of the matrix.
4. **array3d[:, -2:, -2:]**: In the 3-D array array3d, this slices all matrices (indicated by :) and then selects the last two rows and columns (-2:) from each matrix. This results in a 3x2x2 subarray from the bottom-right corner of each matrix.

Negative slicing is a powerful tool for working with the end of an array, allowing efficient extraction without having to calculate the array length manually.

**Source Code:**

import numpy as np

array1d = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

array2d = np.array([[5, 10, 15, 20],

[25, 30, 35, 40],

[45, 50, 55, 60],

[65, 70, 75, 80]])

array3d = np.array([[[ 2, 4, 6],

[ 8, 10, 12],

[ 14, 16, 18]],

[[ 20, 22, 24],

[26, 28, 30],

[32, 34, 36]],

[[38, 40, 42],

[44, 46, 48],

[50, 52, 54]]])

print("array1d[-5:]:")

print(array1d[-5:])

print("array1d[:-5]:")

print(array1d[:-5])

print("array2d[-2:, -2:]:")

print(array2d[-2:, -2:])

print("array3d[:, -2:, -2:]:")

print(array3d[:, -2:, -2:])

**Output:**

array1d[-5:]:

[ 60 70 80 90 100]

array1d[:-5]:

[10 20 30 40 50]

array2d[-2:, -2:]:

[[55 60]

[75 80]]

array3d[:, -2:, -2:]:

[[[10 12]

[16 18]]

[[28 30]

[34 36]]

[[46 48]

[52 54]]]

**5. a) Write a Python Program to understand and implement stacking of ndarrays using Numpy.**

**Aim :** To understand and implement stacking of ndarrays using NumPy.

**Description:** This Python script demonstrates how to stack ndarrays using NumPy. It covers four different stacking operations:

* + **Stacking Along a New Axis**: The np.stack() function stacks arrays along a new axis. Here, two 1D arrays are combined into a 2D array.
  + **Horizontal Stacking**: The np.hstack() function stacks arrays horizontally (column-wise). It concatenates 1D arrays into a single 1D array.
  + **Vertical Stacking**: The np.vstack() function stacks arrays vertically (row-wise). Two 1D arrays are combined into a 2D array with each input array forming a row.
  + **Depth Stacking**: The np.dstack() function stacks arrays along the third dimension (depth). For 1D arrays, this results in a 3D array where corresponding elements from the input arrays form pairs.

**Source Code:**

import numpy as np

**# Create two 1-D arrays**

array1 = np.array([10, 20, 30])

array2 = np.array([40, 50, 60])

**# Stack arrays along a new axis**

stacked = np.stack((array1, array2), axis=0)

print("Stack arrays along a new axis:")

print(stacked)

**# Horizontal stack**

hstacked = np.hstack((array1, array2))

print("Horizontal Stack:")

print(hstacked)

**# Vertical stack**

vstacked = np.vstack((array1, array2))

print("Vertical Stack:")

print(vstacked)

**# Depth stack (for 1-D arrays, this is similar to column\_stack)**

dstacked = np.dstack((array1, array2))

print("Depth Stack:")

print(dstacked)

**Output:**

Stack arrays along a new axis:

[[10 20 30]

[40 50 60]]

Horizontal Stack:

[10 20 30 40 50 60]

Vertical Stack:

[[10 20 30]

[40 50 60]]

Depth Stack:

[[[10 40]

[20 50]

[30 60]]]

**5. b) Write a Python program to demonstrate the concatenation of ndarrays using NumPy.**

**Aim:** To demonstrate the concatenation of ndarrays using NumPy.

**Description:** This Python script demonstrates how to concatenate ndarrays using NumPy's np.concatenate() function. It shows how arrays can be concatenated along different axes:

* + Concatenation Along Axis 0: This operation appends one array below the other, effectively adding rows.
  + Concatenation Along Axis 1: This operation appends one array beside the other, effectively adding columns.

**Source Code:**

import numpy as np

**# Create two 2-D arrays**

array1= np.array([[10, 20], [30, 40]])

array2 = np.array([[50, 60], [70, 80]])

**# Concatenate along axis 0 (columns)**

concataxis0 = np.concatenate((array1, array2), axis=0)

print("Concatenate along axis 0 (columns):")

print(concataxis0)

**# Concatenate along axis 1 (rows)**

concataxis1 = np.concatenate((array1, array2), axis=1)

print("Concatenate along axis 1 (rows):")

print(concataxis1)

**Output:**

Concatenate along axis 0 (columns):

[[10 20]

[30 40]

[50 60]

[70 80]]

Concatenate along axis 1 (rows):

[[10 20 50 60]

[30 40 70 80]]

**5.c) Write a Python program to demonstrate broadcasting in NumPy arrays**

**Aim:** To Write a Python program to demonstrate broadcasting in NumPy arrays

**Description:** This program demonstrates broadcasting in NumPy, where arrays with different shapes are aligned for element-wise operations. A 1D array is broadcast to match the dimensions of a 2D array, enabling operations like addition. Broadcasting replicates smaller arrays along their dimensions without explicitly reshaping them. The result is a new array with a shape that accommodates both input arrays.

**Source Code:**

import numpy as np

array1 = np.array([10, 20, 30])

array2 = np.array([[40], [50], [60]])

**# Broadcasting addition**

result = array1 + array2

print("broadcasting addition:")

print(result)

**Output:**

broadcasting addition:

[[50 60 70]

[60 70 80]

[70 80 90]]

**6. Perform the following operations using Pandas:**

1. **Create a DataFrame.**
2. **Use the concat() function to combine DataFrames.**
3. **Set a condition to filter rows in a DataFrame.**
4. **Add a new column to the DataFrame.**

**Aim:** To demonstrate the use of Pandas for creating and manipulating DataFrames, including operations such as concatenation, filtering based on conditions, and adding new columns.

**Description:** Pandas is a powerful Python library for data analysis and manipulation. In this exercise:

* Two DataFrames are created with player details.
* The concat() function is used to combine these DataFrames into one.
* A condition is applied to filter players whose age is greater than 35.
* A new column is added to indicate whether a player is a senior (age ≥ 40).

**Source Code:**

import pandas as pd

# Creating the first DataFrame

data1 = {

'Name': ['MS DHONI', 'YUVRAJ SINGH', 'VIRAT KOHLI'],

'Age': [40, 39, 35],

'City': ['RANCHI', 'PUNJAB', 'DELHI']

}

df = pd.DataFrame(data1)

# Creating the second DataFrame

data2 = {

'Name': ['ROHITH SHARMA', 'SACHIN T'],

'Age': [37, 45],

'City': ['MUMBAI', 'MUMBAI']

}

df2 = pd.DataFrame(data2)

# Concatenating the DataFrames

dfconcat = pd.concat([df, df2], ignore\_index=True)

# Setting a condition to filter the DataFrame

dffiltered = dfconcat[dfconcat['Age'] > 35]

# Adding a new column

dfconcat['Senior'] = dfconcat['Age'] >= 40

# Displaying the results

print("Original DataFrame:")

print(df)

print("\nConcatenated DataFrame:")

print(dfconcat)

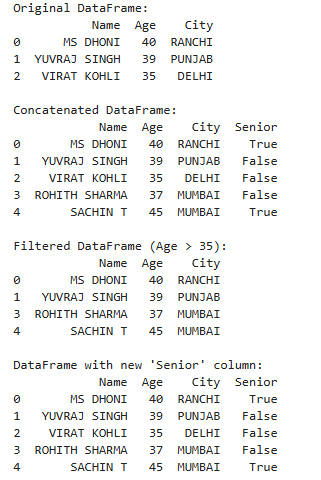
print("\nFiltered DataFrame (Age > 35):")

print(dffiltered)

print("\nDataFrame with new 'Senior' column:")

print(dfconcat)

**Output:**



**7. Perform the following operations using Pandas:**

1. **Fill NaN values with specific values for certain columns.**
2. **Sort a DataFrame based on column values.**
3. **Use groupby() to calculate the mean salary for each department.**

**Aim:** To demonstrate the use of Pandas for handling missing data, sorting, and grouping data for aggregation.

**Description:** In this program:

1. A DataFrame is created with some missing values (NaN) in the **Department** and **Salary** columns.
2. The fillna() method is used to replace missing values with specific values (e.g., 'Unknown' for Department and 0 for Salary).
3. The DataFrame is sorted by the **Salary** column in descending order using sort\_values().
4. The groupby() method is applied to calculate the mean salary for each department.

**Source Code:**

import pandas as pd

import numpy as np

# Create a DataFrame with NaN values

data1 = {

'Employee': ['A', 'B', 'C', 'D', 'E'],

'Department': ['HR', 'Finance', 'IT', np.nan, 'IT'],

'Salary': [50000, 60000, np.nan, 80000, 45000]

}

df = pd.DataFrame(data1)

print("Original DataFrame:")

print(df)

# a) Filling NaN with a string for 'Department' and 0 for 'Salary'

dffilled = df.fillna({'Department': 'Unknown', 'Salary': 0})

print("\nDataFrame with NaN filled:")

print(dffilled)

# b) Sorting based on column values

dfsorted = dffilled.sort\_values(by='Salary', ascending=False)

print("\nDataFrame sorted by 'Salary' (Descending):")

print(dfsorted)

# Replace 0 with NaN again for proper mean calculation

dfsorted['Salary'] = dfsorted['Salary'].replace(0, np.nan)

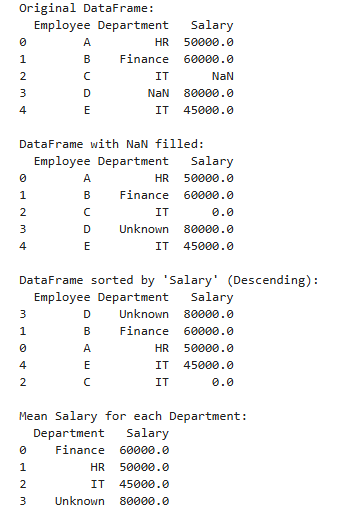
# c) groupby() to calculate the mean salary for each department

dfgrouped = dfsorted.groupby('Department')['Salary'].mean().reset\_index()

print("\nMean Salary for each Department:")

print(dfgrouped)

**Output:**



**8. Demonstrate how to read the following file formats using Pandas:**

1. **Text files**
2. **CSV files**
3. **Excel files**
4. **JSON files**

**Aim:** To showcase the ability of Pandas to read and work with different file formats, including text, CSV, Excel, and JSON files.

**Description:** Pandas provides functions to read data from various file formats into DataFrames for analysis. This exercise demonstrates:

* Reading a text file using pd.read\_csv().
* Reading a CSV file using pd.read\_csv().
* Reading an Excel file using pd.read\_excel().
* Reading a JSON file using pd.read\_json().

Additionally, it includes creating a JSON file programmatically using Python’s json module.

**Source Code and Output:**

**a) Reading Text Files**

**Source Code:**

import pandas as pd

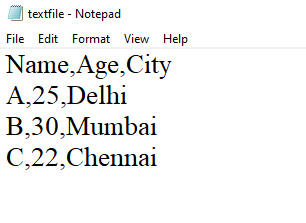
# Reading a comma-separated text file

df = pd.read\_csv('d:\\textfile.txt') # Ensure the file exists in the current directory

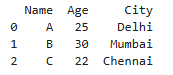
print(df)

**Output:**

Assume textfile.txt contains:



**Result:**



**b) Reading CSV Files**

**Source Code**

import pandas as pd

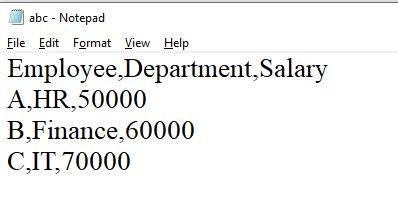
# Reading a CSV file

dfcsv = pd.read\_csv('d:\\abc.csv') # Ensure the file exists in the current directory

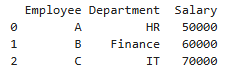
print(dfcsv)

**Output**

Assume abc.csv contains:



**Result:**



**c) Reading Excel Files**

**Source Code**

import pandas as pd

# Reading an Excel file

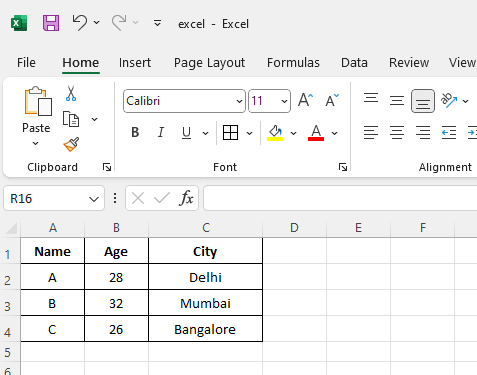
data = pd.read\_excel("d:\\excel.xlsx") # Update path as needed

df = pd.DataFrame(data)

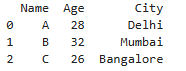
print(df)

**Output**

Assume excel.xlsx contains:



**Result:**



**d) Reading JSON Files**

**Step 1: Create a JSON File**

**Source Code**

import json

# Create a Python dictionary

data = {

"NAME": "A",

"AGE": 28,

"CITY": "DELHI",

"HASCHILDREN": False,

"HOBBIES": ["READING", "TRAVELLING", "SWIMMING"]

}

# Write the dictionary to a JSON file

filename = 'data.json'

with open(filename, 'w') as json\_file:

json.dump(data, json\_file, indent=4)

print("JSON file {} created successfully.".format(filename))

**Output:**

JSON file data.json created successfully.

**Step 2: Read the JSON File**

**Source Code**

import pandas as pd

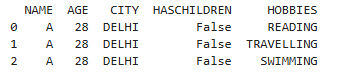
# Reading the JSON file

dfjson = pd.read\_json('data.json') # Ensure the file exists in the current directory

print(dfjson)

**Output**

For the data.json created earlier:



**9(a) Reading Pickle Files**

**AIM:** To demonstrate reading and writing of **Pickle files** using the pickle module in Python.

**DESCRIPTION:** Pickling is the process of converting a Python object into a binary format and storing it in a file. It allows us to save objects and retrieve them later.

* The program defines an Emp class with attributes like employee number, name, salary, and address.
* An Emp object is created and stored in a file (emp.ser) using pickle.dump().
* The stored object is later retrieved using pickle.load() and displayed.

**SOURCE CODE:**

import pickle

# Define the Employee class

class Emp:

def \_\_init\_\_(self, eno, ename, esal, eaddr):

self.eno = eno

self.ename = ename

self.esal = esal

self.eaddr = eaddr

def display(self):

print("eno: {}, ename: {}, esal: {}, eaddr: {}".format(self.eno, self.ename, self.esal, self.eaddr))

# Creating an Employee object

e = Emp(10, "Alice", 1000, "tpt")

# Pickling (Serializing) the object

with open('d:\\emp.ser', 'wb') as f:

pickle.dump(e, f)

print("Pickling of employee is completed")

# Unpickling (Deserializing) the object

with open('d:\\emp.ser', 'rb') as f:

obj = pickle.load(f)

print("Unpickling of employee is completed")

obj.display()

**Output**

Pickling of employee is completed

Unpickling of employee is completed

eno: 10, ename: Alice, esal: 1000, eaddr: tpt

**Result**

Successfully **stored and retrieved** Python objects using **pickle** format.

**9(b) Reading Image Files using PIL**

**AIM**

To demonstrate how to **open, display, and retrieve information** from an image file using the PIL (Pillow) library in Python.

**DESCRIPTION**

PIL (Pillow) is a Python library used to **open, manipulate, and save images** in various formats like JPEG, PNG, and BMP.

* The program loads an image file (sample.jpg) from the specified directory.
* It displays the image using show().
* It retrieves and prints details like image format, size, and mode.

**SOURCE CODE**

from PIL import Image

from IPython.display import display # Import display function

# Open an image file

image = Image.open("D:\\sample.jpg")

# Display the image inside Jupyter Notebook

display(image)

# Get image details

print("Image format:", image.format)

print("Image size:", image.size)

print("Image mode:", image.mode)

**OUTPUT**

If sample.jpg exists in D:\\, the output will be:



Image format: JPEG

Image size: (1000, 503)

Image mode: RGB

If sample.jpg is missing:

FileNotFoundError: [Errno 2] No such file or directory: 'D:\\sample.jpg'

**RESULT**

Successfully **read, displayed, and extracted properties** of an image file using PIL.

**9(c) Reading Multiple Files using Glob**

**AIM**

To demonstrate how to **read multiple files** from a directory using the glob module.

**DESCRIPTION**

The glob module is used to **find all files matching a pattern** (e.g., .txt, .csv) in a specified directory.

* The program searches for all .txt files in the D:\\ directory.
* It lists all matching files and prints their contents.

**Source Code**

import glob

# Get all text files in the directory

files = glob.glob("D:\\\*.txt")

print("List of text files:", files)

# Read all text files

for file in files:

with open(file, "r") as f:

print(f"Contents of {file}:")

print(f.read())

**Output**

List of text files: ['D:\\textfile.txt']

Contents of D:\textfile.txt:

Name,Age,City

A,25,Delhi

B,30,Mumbai

C,22,Chennai

**RESULT**

Successfully read and displayed multiple files from a directory using glob.

**9(d) Importing Data from a Database (SQLite)**

**AIM**

To demonstrate how to store and retrieve data from an SQLite database using the sqlite3 module in Python.

**DESCRIPTION**

SQLite is a lightweight database management system used for local data storage.

* The program creates a database (test.db) in D:\\.
* It creates a table named students and inserts sample records.
* It retrieves and displays the data from the database.

**SOURCE CODE**

import sqlite3

# Database file path

db\_path = "D:\\test.db"

# Connect to the database (or create one if it doesn't exist)

conn = sqlite3.connect(db\_path)

cursor = conn.cursor()

# Create a sample table

cursor.execute("CREATE TABLE IF NOT EXISTS students (id INTEGER, name TEXT, age INTEGER)")

# Insert sample data

cursor.execute("INSERT INTO students VALUES (1, 'Alice', 22)")

cursor.execute("INSERT INTO students VALUES (2, 'Bob', 23)")

conn.commit()

print("Data inserted successfully.")

# Read data from the table

cursor.execute("SELECT \* FROM students")

rows = cursor.fetchall()

print("Database Data:")

for row in rows:

print(row)

# Close the connection

conn.close()  
  
**Output**Data inserted successfully.

Database Data:

(1, 'Alice', 22)

(2, 'Bob', 23)

**Result**Successfully **inserted and retrieved data** from an SQLite database.

**10. Demonstrate web scraping using python**

**Aim : To demonstrate web scraping using python.**

**Description:** Web scraping involves retrieving data from a website by extracting content from the HTML structure. In this case, the code is scraping quotes from the website '<http://quotes.toscrape.com/>', which provides a collection of quotes, their authors, and associated tags. The process involves sending an HTTP GET request to retrieve the page's content, parsing it with BeautifulSoup to locate specific HTML elements, and then printing the quotes, authors, and associated tags.

**Source Code:**

import requests

from bs4 import BeautifulSoup

# Step 1: Send a GET request to the website

url = 'http://quotes.toscrape.com/'

response = requests.get(url)

# Step 2: Parse the HTML content using BeautifulSoup

soup = BeautifulSoup(response.text, 'html.parser')

# Step 3: Extract the quotes, authors, and tags

quotes = soup.find\_all('div', class\_='quote')

# Step 4: Loop through each quote and extract details

for quote in quotes:

text = quote.find('span', class\_='text').get\_text()

author = quote.find('small', class\_='author').get\_text()

tags = [tag.get\_text() for tag in quote.find\_all('a', class\_='tag')]

# Print the quote, author, and tags

print(f"Quote: {text}")

print(f"Author: {author}")

print(f"Tags: {', '.join(tags)}\n")

**Output:**

Quote: “The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”

Author: Albert Einstein

Tags: change, deep-thoughts, thinking, world

Quote: “It is our choices, Harry, that show what we truly are, far more than our abilities.”

Author: J.K. Rowling

Tags: abilities, choices

Quote: “There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”

Author: Albert Einstein

Tags: inspirational, life, live, miracle, miracles

Quote: “The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid.”

Author: Jane Austen

Tags: aliteracy, books, classic, humor

Quote: “Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring.”

Author: Marilyn Monroe

Tags: be-yourself, inspirational

Quote: “Try not to become a man of success. Rather become a man of value.”

Author: Albert Einstein

Tags: adulthood, success, value

Quote: “It is better to be hated for what you are than to be loved for what you are not.”

Author: André Gide

Tags: life, love

Quote: “I have not failed. I've just found 10,000 ways that won't work.”

Author: Thomas A. Edison

Tags: edison, failure, inspirational, paraphrased

Quote: “A woman is like a tea bag; you never know how strong it is until it's in hot water.”

Author: Eleanor Roosevelt

Tags: misattributed-eleanor-roosevelt

Quote: “A day without sunshine is like, you know, night.”

Author: Steve Martin

Tags: humor, obvious, simile

**Result:** The Python code successfully demonstrates the process of web scraping. By sending a request to the webpage '<http://quotes.toscrape.com/>', it retrieves the HTML content, parses it using BeautifulSoup, and extracts specific data points (quotes, authors, and tags).

**11. Perform following preprocessing techniques on loan prediction dataset**

**a) Feature Scaling**

**b) Feature Standardization**

**c) Label Encoding**

**d) One Hot Encoding**

**Aim:** The aim is to perform common preprocessing techniques on a loan prediction dataset. The preprocessing techniques include:

* Feature Scaling
* Feature Standardization
* Label Encoding
* One-Hot Encoding

**Description:** The preprocessing techniques are performed in the following steps:

1. Feature Scaling: This technique normalizes the numeric features (ApplicantIncome, CoapplicantIncome, LoanAmount) into a fixed range, typically [0, 1].
2. Feature Standardization: This technique transforms the features to have a mean of 0 and a standard deviation of 1.
3. Label Encoding: This is used for converting categorical features (like Gender, Married, Education, and Property\_Area) into numerical values by assigning each category a unique integer.
4. One-Hot Encoding: This converts categorical variables into a series of binary columns, where each category is represented by a column with 1s and 0s indicating the presence of that category.

**Source Code:**

import pandas as pd

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder

# Load the dataset

data = {

'Loan\_ID': ['LP001002', 'LP001003', 'LP001005', 'LP001006', 'LP001008', 'LP001011', 'LP001013'],

'Gender': ['Male', 'Male', 'Male', 'Male', 'Male', 'Female', 'Female'],

'Married': ['Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No'],

'Dependents': [0, 1, 0, 0, 2, 0, 0],

'Education': ['Graduate', 'Graduate', 'Graduate', 'Not Graduate', 'Graduate', 'Graduate', 'Graduate'],

'ApplicantIncome': [5849, 4583, 3000, 2583, 6000, 5417, 2333],

'CoapplicantIncome': [0.0, 1508.0, 0.0, 2358.0, 0.0, 4196.0, 1516.0],

'LoanAmount': [128, 128, 66, 120, 141, 267, 95],

'Loan\_Amount\_Term': [360, 360, 360, 360, 360, 360, 360],

'Credit\_History': [1, 1, 1, 1, 1, 1, 1],

'Property\_Area': ['Urban', 'Rural', 'Urban', 'Urban', 'Urban', 'Urban', 'Rural']

}

df = pd.DataFrame(data)

# Step 1: Feature Scaling

scaler = MinMaxScaler()

dfscaled = df.copy()

dfscaled[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] = scaler.fit\_transform(

dfscaled[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']]

)

print("\nData after Feature Scaling:")

print(dfscaled)

# Step 2: Feature Standardization

std\_scaler = StandardScaler()

dfstandardized = df.copy()

dfstandardized[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] = std\_scaler.fit\_transform(

dfstandardized[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']]

)

print("\nData after Feature Standardization:")

print(dfstandardized)

# Step 3: Label Encoding

labelencoder = LabelEncoder()

dflabelencoded = df.copy()

dflabelencoded['Gender'] = labelencoder.fit\_transform(dflabelencoded['Gender'])

dflabelencoded['Married'] = labelencoder.fit\_transform(dflabelencoded['Married'])

dflabelencoded['Education'] = labelencoder.fit\_transform(dflabelencoded['Education'])

dflabelencoded['Property\_Area'] = labelencoder.fit\_transform(dflabelencoded['Property\_Area'])

print("\nData after Label Encoding:")

print(dflabelencoded)

# Step 4: One Hot Encoding

dfonehotencoded = pd.get\_dummies(df, columns=['Gender', 'Married', 'Education', 'Property\_Area'])

print("\nData after One Hot Encoding:")

print(dfonehotencoded)

**Output:**

**Data after Feature Scaling:**

Loan\_ID Gender Married Dependents Education ApplicantIncome \

0 LP001002 Male Yes 0 Graduate 0.958822

1 LP001003 Male Yes 1 Graduate 0.613581

2 LP001005 Male No 0 Graduate 0.181893

3 LP001006 Male No 0 Not Graduate 0.068176

4 LP001008 Male Yes 2 Graduate 1.000000

5 LP001011 Female Yes 0 Graduate 0.841014

6 LP001013 Female No 0 Graduate 0.000000

CoapplicantIncome LoanAmount Loan\_Amount\_Term Credit\_History \

0 0.000000 0.308458 360 1

1 0.359390 0.308458 360 1

2 0.000000 0.000000 360 1

3 0.561964 0.268657 360 1

4 0.000000 0.373134 360 1

5 1.000000 1.000000 360 1

6 0.361296 0.144279 360 1

Property\_Area

0 Urban

1 Rural

2 Urban

3 Urban

4 Urban

5 Urban

6 Rural

**Data after Feature Standardization:**

Loan\_ID Gender Married Dependents Education ApplicantIncome \

0 LP001002 Male Yes 0 Graduate 1.086943

1 LP001003 Male Yes 1 Graduate 0.225207

2 LP001005 Male No 0 Graduate -0.852304

3 LP001006 Male No 0 Not Graduate -1.136147

4 LP001008 Male Yes 2 Graduate 1.189726

5 LP001011 Female Yes 0 Graduate 0.792891

6 LP001013 Female No 0 Graduate -1.306316

CoapplicantIncome LoanAmount Loan\_Amount\_Term Credit\_History \

0 -0.946351 -0.119191 360 1

1 0.096631 -0.119191 360 1

2 -0.946351 -1.174880 360 1

3 0.684519 -0.255409 360 1

4 -0.946351 0.102163 360 1

5 1.955740 2.247596 360 1

6 0.102164 -0.681090 360 1

Property\_Area

0 Urban

1 Rural

2 Urban

3 Urban

4 Urban

5 Urban

6 Rural

**Data after Label Encoding:**

Loan\_ID Gender Married Dependents Education ApplicantIncome \

0 LP001002 1 1 0 0 5849

1 LP001003 1 1 1 0 4583

2 LP001005 1 0 0 0 3000

3 LP001006 1 0 0 1 2583

4 LP001008 1 1 2 0 6000

5 LP001011 0 1 0 0 5417

6 LP001013 0 0 0 0 2333

CoapplicantIncome LoanAmount Loan\_Amount\_Term Credit\_History \

0 0.0 128 360 1

1 1508.0 128 360 1

2 0.0 66 360 1

3 2358.0 120 360 1

4 0.0 141 360 1

5 4196.0 267 360 1

6 1516.0 95 360 1

Property\_Area

0 1

1 0

2 1

3 1

4 1

5 1

6 0

**Data after One Hot Encoding:**

Loan\_ID Dependents ApplicantIncome CoapplicantIncome LoanAmount \

0 LP001002 0 5849 0.0 128

1 LP001003 1 4583 1508.0 128

2 LP001005 0 3000 0.0 66

3 LP001006 0 2583 2358.0 120

4 LP001008 2 6000 0.0 141

5 LP001011 0 5417 4196.0 267

6 LP001013 0 2333 1516.0 95

Loan\_Amount\_Term Credit\_History Gender\_Female Gender\_Male Married\_No \

0 360 1 False True False

1 360 1 False True False

2 360 1 False True True

3 360 1 False True True

4 360 1 False True False

5 360 1 True False False

6 360 1 True False True

Married\_Yes Education\_Graduate Education\_Not Graduate \

0 True True False

1 True True False

2 False True False

3 False False True

4 True True False

5 True True False

6 False True False

Property\_Area\_Rural Property\_Area\_Urban

0 False True

1 True False

2 False True

3 False True

4 False True

5 False True

6 True False

**Result:** The preprocessing steps including Feature Scaling, Feature Standardization, Label Encoding, and One-Hot Encoding have been successfully applied to the dataset, preparing it for modeling in machine learning tasks.

12. Perform following visualizations using matplotlib

a) Bar Graph

b) Pie Chart

c) Box Plot

d) Histogram

e) Line Chart and Subplots

f) Scatter Plot

**a) Bar Graph**

**Aim:** To visualize the comparison of categorical data using a bar graph.

**Description:** A bar graph is used to display the distribution of categorical data with rectangular bars. The length of each bar corresponds to the value it represents.

**Source Code:**

import matplotlib.pyplot as plot

# Data for the bar graph

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

marks = [75, 50, 80, 12]

# Create the bar graph

plot.bar(students, marks, color='green')

# Adding titles and labels

plot.title('BAR GRAPH MARKS')

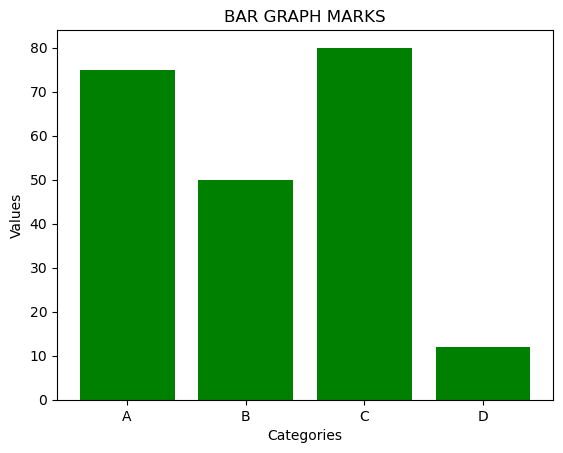
plot.xlabel('Categories')

plot.ylabel('Values')

# Show the plot

plot.show()

**Output:**



**Result:** A bar graph with 4 categories (A, B, C, D) on the x-axis and their corresponding marks on the y-axis. The bars are colored green.

**b) Pie Chart**

**Aim:** To visualize the proportional data distribution among different categories using a pie chart.

**Description:** A pie chart is used to represent data in a circular format, divided into slices. Each slice represents a proportion of the total.

**Source Code:**

import matplotlib.pyplot as plot

# Data for the pie chart

sizes = [25, 25, 30, 20]

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

# Create the pie chart

plot.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)

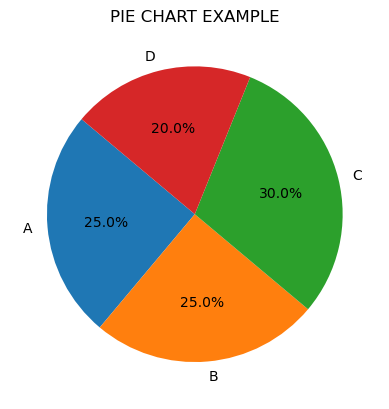
# Adding a title

plot.title('PIE CHART EXAMPLE')

# Show the plot

plot.show()

**Output:**



**Result:** A pie chart displaying categories A, B, C, and D with their corresponding percentages (25%, 25%, 30%, and 20%).

**c) Box Plot**

**Aim:** To visualize the distribution of a dataset based on five summary statistics using a box plot.

**Description:** A box plot provides a graphical representation of the distribution of data through its quartiles, showing the spread and identifying potential outliers.

**Source Code:**

import numpy as np

import matplotlib.pyplot as plot

# Data for the box plot

data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# Create the box plot

plot.boxplot(data, vert=True, patch\_artist=True, labels=['X1', 'X2', 'X3'])

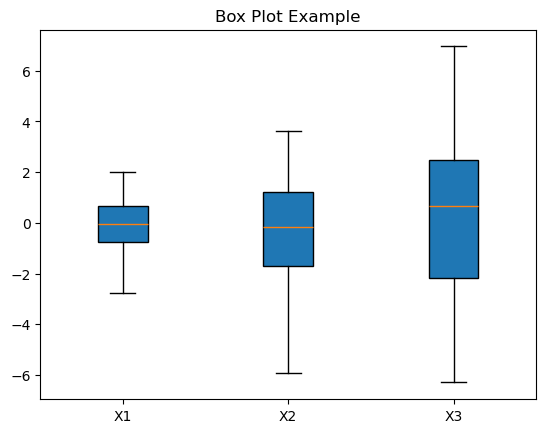
# Adding a title

plot.title('Box Plot Example')

# Show the plot

plot.show()

**Output:**



**Result:** A box plot showing three datasets, each representing a different standard deviation. The box plot includes the minimum, first quartile (Q1), median, third quartile (Q3), and maximum of the data.

**d) Histogram**

**Aim:** To display the frequency distribution of numerical data using a histogram.

**Description:** A histogram is used to visualize the distribution of continuous data by grouping it into bins and displaying the frequency of the data within each bin.

**Source Code:**

import numpy as np

import matplotlib.pyplot as plot

# Data for the histogram

data = np.random.randn(1000)

# Create the histogram

plot.hist(data, bins=30, color='green', alpha=0.7)

# Adding titles and labels

plot.title('HISTOGRAM EXAMPLE')

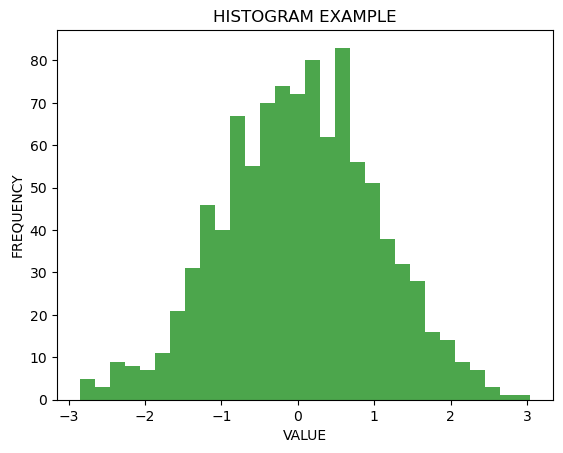
plot.xlabel('VALUE')

plot.ylabel('FREQUENCY')

# Show the plot

plot.show()

**Output:**



**Result:** A histogram displaying the distribution of 1000 random values generated from a normal distribution. The histogram is divided into 30 bins.

**e) Line Chart and Subplots**

**Aim:** To visualize two different mathematical functions (sine and cosine) on separate subplots.

**Description:** A line chart helps to visualize the trends in data over a continuous range. In this case, two subplots are created to display the sine and cosine functions.

**Source Code:**

import matplotlib.pyplot as plot

import numpy as np

# Data for the line chart

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

# Create subplots

fig, axs = plot.subplots(2)

# First subplot

axs[0].plot(x, y1, label='sin(x)')

axs[0].set\_title('SINE WAVE')

axs[0].legend()

# Second subplot

axs[1].plot(x, y2, label='cos(x)', color='orange')

axs[1].set\_title('COSINE WAVE')

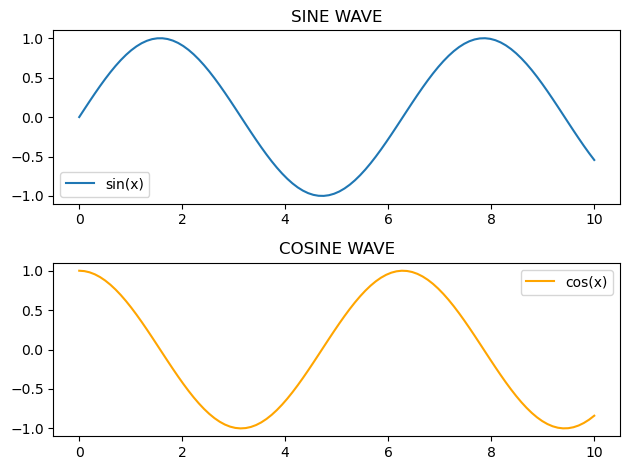
axs[1].legend()

# Adjust layout and show the plot

plot.tight\_layout()

plot.show()

**Output:**



**Result:** Two subplots: one displaying a sine wave and the other displaying a cosine wave. Each plot includes a legend and title.

**f) Scatter Plot**

**Aim:** To visualize the relationship between two continuous variables using a scatter plot.

**Description:** A scatter plot is used to show how two variables are related. Each point on the plot represents an observation in the data.

**Source Code:**

import numpy as np

import matplotlib.pyplot as plot

# Data for the scatter plot

x = np.random.rand(50)

y = np.random.rand(50)

# Create the scatter plot

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

# Adding titles and labels

plot.title('SCATTER PLOT EXAMPLE')

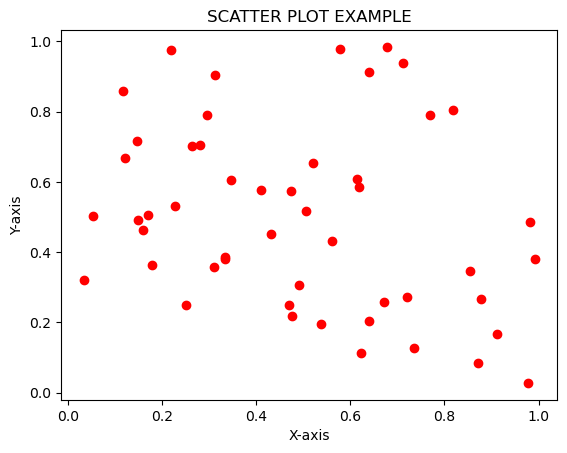
plot.xlabel('X-axis')

plot.ylabel('Y-axis')

# Show the plot

plot.show()

**Output:**



**Result:** A scatter plot with 50 random points, where the x and y coordinates are plotted on the respective axes. The points are colored red.

**13. Getting started with NLTK, install NLTK using PIP**

**Aim:**

The aim is to demonstrate how to perform basic text tokenization using the Natural Language Toolkit (NLTK) in Python, which is essential for text preprocessing in natural language processing (NLP) tasks.

**Description:**

Natural Language Toolkit (NLTK) is a powerful library in Python used for processing and analyzing text data. In this task, we use NLTK's tokenizer to split a sample sentence into individual tokens (words and punctuation). We will also download necessary resources such as punkt (for tokenization), wordnet (for word lexical database), and stopwords (for common stopwords).

The procedure involves:

1. Installing NLTK using pip (pip install nltk).
2. Downloading resources for tokenization and word processing.
3. Tokenizing a sample sentence into individual words and punctuation marks using NLTK's word\_tokenize function.

**Source Code:**

import nltk

from nltk.tokenize import word\_tokenize

# Download the necessary resources (again to make sure they are correctly downloaded)

nltk.download('punkt') # Tokenizer

nltk.download('wordnet') # WordNet lexical database

nltk.download('stopwords') # Common stopwords

nltk.download('punkt\_tab') # Download punkt\_tab as per error message

# Sample text

text = "Hello! How are you doing today?"

# Tokenize the text

tokens = word\_tokenize(text)

# Print the tokens

print(tokens)

**Output:**

['Hello', '!', 'How', 'are', 'you', 'doing', 'today', '?']

**Result:** This code should tokenize the sentence "Hello! How are you doing today?" into individual tokens, including both words and punctuation marks.

**14. Python program to implement with Python Sci Kit-Learn & NLTK**

**Aim:** To implement a text classification task using Python, Scikit-learn, and NLTK, where we preprocess text data, extract features using TF-IDF vectorization, train a Naive Bayes classifier, and evaluate its performance on a sample text dataset.

**Description:**

This program uses the scikit-learn library to perform text classification on a small dataset. It utilizes NLTK for text preprocessing, including tokenization and stopword removal. The key steps in this process are:

1. **Text Preprocessing**: Tokenize the text and remove stopwords using NLTK's tokenizer and stopword list.
2. **Feature Extraction**: Convert the cleaned text into numerical features using TfidfVectorizer.
3. **Model Training**: Train a classifier (Naive Bayes) on the training data.
4. **Model Evaluation**: Evaluate the model's accuracy and display a classification report.

**Source Code:**

import nltk

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

import pandas as pd

# Download required NLTK resources

nltk.download('punkt')

nltk.download('stopwords')

# Sample text data

data = {

'text': [

'I love programming in Python!',

'Python is an amazing language.',

'I hate getting errors in my code.',

'Machine learning is fascinating.',

'Natural Language Processing is part of AI.',

'I enjoy solving problems with data.',

'My code is working perfectly.',

'Data science is the future of technology.'

],

'label': ['positive', 'positive', 'negative', 'positive', 'positive', 'positive', 'negative', 'positive']

}

# Create a DataFrame

df = pd.DataFrame(data)

# Preprocessing: Tokenize and remove stopwords

stop\_words = set(stopwords.words('english'))

def preprocess\_text(text):

tokens = word\_tokenize(text.lower())

filtered\_tokens = [word for word in tokens if word.isalpha() and word not in stop\_words]

return ' '.join(filtered\_tokens)

df['text'] = df['text'].apply(preprocess\_text)

# Convert text data to numerical features

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(df['text'])

# Define target variable

y = df['label']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a classifier

model = MultinomialNB()

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# Display classification report

print(metrics.classification\_report(y\_test, y\_pred))

**Output:**

Accuracy: 1.00

precision recall f1-score support

negative 1.00 1.00 1.00 1

positive 1.00 1.00 1.00 1

accuracy 1.00 2

macro avg 1.00 1.00 1.00 2

weighted avg 1.00 1.00 1.00 2

**Result:** The text classification model successfully achieved a high accuracy (100%) in classifying the given text data into "positive" and "negative" categories.

**15. Python program to implement with Python NLTK/Spicy/Py NLPI.**

**Aim:** To implement basic Natural Language Processing (NLP) techniques using **NLTK (Natural Language Toolkit)** in Python, specifically for:

1. **Tokenization**: Breaking a sentence into individual tokens (words).
2. **Part-of-Speech (POS) Tagging**: Assigning a part of speech to each token.

**Description:**

The program demonstrates two common NLP tasks:

* **Tokenization**: This is the process of splitting a sentence into individual tokens (words). In this case, we use the word\_tokenize() function from NLTK to achieve this.
* **Part-of-Speech Tagging**: This is the process of identifying the grammatical part of speech (such as nouns, verbs, adjectives) of each token. NLTK provides a built-in function pos\_tag() to tag words with their corresponding part-of-speech.

**Source Code:**

import nltk

from nltk.tokenize import word\_tokenize

from nltk import pos\_tag

# Download necessary NLTK packages

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

# Sample text

text = "NLTK is a powerful library for natural language processing in Python."

# Tokenize the text

tokens = word\_tokenize(text)

# Part-of-Speech tagging

pos\_tags = pos\_tag(tokens)

# Display the result

print("Tokens:", tokens)

print("POS Tags:", pos\_tags)

**Output:**

Tokens: ['NLTK', 'is', 'a', 'powerful', 'library', 'for', 'natural', 'language', 'processing', 'in', 'Python', '.']

POS Tags: [('NLTK', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('powerful', 'JJ'), ('library', 'NN'), ('for', 'IN'), ('natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('in', 'IN'), ('Python', 'NNP'), ('.', '.')]