NumPy Library

Getting Familiar with NumPy

NumPy is a powerful library for numerical computing in Python. It provides support for arrays, matrices, and many mathematical functions. Here are some core functionalities:

-Creating Arrays -Basic operations like dot product,element wise multiplication,adittion and more -finding properties of array like shape of the array,array dimension,total number of elements in an array.

Here is a sample code for the above core functionalities of NumPy:

In [7]:
```python
import numpy as np

# Creating a 1D array
arr1 = np.array([1, 2, 3, 4, 5])
print("1D array: ",arr1)

# Creating a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print("2D array: \n",arr2)

# Creating an array with a range of values
arr3 = np.arange(0, 10, 2)
print("3D array: ",arr3)

# Element-wise addition
arr_sum = arr1 + arr3
print("array addition: ",arr_sum)

# Element-wise multiplication
arr_mul = arr1 * 2
print("array multiplication: ",arr_mul)

# Dot product
dot_product = np.dot(arr1, arr1)
print("dot product: ",dot_product)

print("Shape of the array: ",arr1.shape)  # Shape of the array
print("Number of dimensions: ",arr2.ndim)   # Number of dimensions
print("Total number of elements: ",arr3.size)   # Total number of elements
```

```
1D array:  [1 2 3 4 5]
2D array:
 [[1 2 3]
 [4 5 6]]
3D array:  [0 2 4 6 8]
array addition:  [ 1  4  7 10 13]
array multiplication:  [ 2  4  6  8 10]
dot product:  55
Shape of the array:  (5,)
Number of dimensions:  2
Total number of elements:  5
```

Data Manipulation

In [8]:
```python
import numpy as np

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

# Indexing
print("indexing: ",data[0, 1])  # Accessing element at first row, second column

# Slicing
print("slicing: ",data[:, 1])  # Accessing all rows, second column

# Reshaping
reshaped_data = data.reshape(1, 9)
print("reshaping: ",reshaped_data)

# Applying mathematical operations
squared_data = np.square(data)
print("squared_data: \n",squared_data)
```

```
indexing:  2
slicing:  [2 5 8]
reshaping:  [[1 2 3 4 5 6 7 8 9]]
squared_data:
 [[ 1  4  9]
 [16 25 36]
 [49 64 81]]
```

Data Aggregation

In [11]:
```python
import numpy as np

# Creating an array
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print("Data: ",data)

# Summary statistics
mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)
total_sum = np.sum(data)

print("mean: ",mean)
print("median:",median,)
print("standard deiation:",std_dev)
print("total_sum: ",total_sum)
```

```
Data:  [ 1  2  3  4  5  6  7  8  9 10]
mean:  5.5
median: 5.5
standard deiation: 2.8722813232690143
total_sum:  55
```

Data Analysis

In [10]:
```python
import numpy as np

# Creating a dataset
data = np.random.randn(1000)

# Correlation (example with another dataset)
data2 = np.random.randn(1000)
correlation = np.corrcoef(data, data2)[0, 1]
print(f"Correlation: {correlation}")

# Identifying outliers
mean = np.mean(data)
std_dev = np.std(data)
outliers = data[np.abs(data - mean) > 2 * std_dev]
print(f"Outliers: {outliers}")

# Calculating percentiles
percentile_25 = np.percentile(data, 25)
percentile_75 = np.percentile(data, 75)
print(f"25th Percentile: {percentile_25}, 75th Percentile: {percentile_75}")
```

```
Correlation: -0.011924604269112397
Outliers: [ 2.1011038   2.00651626  2.13183854  2.03134624 -2.2219537   2.1497022
 -2.83416553 -1.93863412  2.50401255 -2.71673936  2.0971569   2.06834981
  2.03955236 -2.05975047 -2.3338825  -2.2549853  -2.42969081 -2.23656137
  2.27821125  2.24386833 -2.29032485 -1.99551978 -2.26909505  2.06568575
  2.20879503  2.60019534  2.94456217  2.1958932   2.33397566 -2.50755772
 -2.4474833   2.10295812  3.91697704  2.50707592  2.05248219 -2.01993888
  2.35061654 -2.18847129 -2.6828178  -3.05566965 -2.15268327 -2.47943932
 -2.21543717  2.20445886  2.04934451 -2.27809959  2.00109487]
25th Percentile: -0.6223453961150498, 75th Percentile: 0.6657244689104468
```

Application in Data Science:

NumPy is essential for data science professionals due to its efficiency and versatility:

Advantages: -Speed: NumPy operations are faster than traditional Python lists due to optimized C code. -Memory Efficiency: NumPy arrays consume less memory. -Functionality: Provides a wide range of mathematical and statistical functions.

Real-World Examples: -Machine Learning: Used for data preprocessing, feature extraction, and model evaluation. -Financial Analysis: Helps in performing quantitative analysis, risk management, and portfolio optimization. -Scientific Research: Facilitates complex simulations, data modeling, and statistical analysis.

Summary:

NumPy enhances the efficiency of numerical computations, making it a cornerstone for data science. Its ability to handle large datasets, perform complex operations, and integrate with other libraries like Pandas and Scikit-learn makes it indispensable for data analysis and machine learning tasks.