```
In [1]: import numpy as np
```

```
In [ ]: #1. Getting Familiar with NumPy
```

```
In [2]: # Creating arrays
arr_1d = np.array([1, 2, 3, 4, 5])
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
```

```
In [3]: # Basic operations
arr_sum = arr_1d + 5
arr_product = arr_2d * 2
```

```
In [4]: # Array properties
print("1D Array:", arr_1d)
print("2D Array:\n", arr_2d)
print("Shape of arr_1d:", arr_1d.shape)
print("Shape of arr_2d:", arr_2d.shape)
print("Data type of arr_1d:", arr_1d.dtype)
print("Number of dimensions (ndim) in arr_2d:", arr_2d.ndim)
```

```
1D Array: [1 2 3 4 5]
2D Array:
 [[1 2 3]
 [4 5 6]]
Shape of arr_1d: (5,)
Shape of arr_2d: (2, 3)
Data type of arr_1d: int64
Number of dimensions (ndim) in arr_2d: 2
```

```
In [5]: #2. Data Manipulation with NumPy
```

```
In [6]: # Array creation
arr = np.arange(10)
```

```
In [7]: # Indexing and Slicing
element = arr[5]  # Accessing the 6th element
slice_arr = arr[2:7]  # Slicing from index 2 to 6
```

```
In [8]: # Reshaping
reshaped_arr = arr.reshape((2, 5))  # Reshaping the array to 2x5
```

```
In [9]: # Mathematical operations
arr_square = np.square(arr)  # Square of each element
arr_exp = np.exp(arr)  # Exponential of each element
```

```
In [10]: print("Original Array:", arr)
print("Element at index 5:", element)
print("Sliced Array:", slice_arr)
print("Reshaped Array (2x5):\n", reshaped_arr)
print("Squared Array:", arr_square)
print("Exponential Array:", arr_exp)
```

```
Original Array: [0 1 2 3 4 5 6 7 8 9]
Element at index 5: 5
Sliced Array: [2 3 4 5 6]
Reshaped Array (2x5):
 [[0 1 2 3 4]
 [5 6 7 8 9]]
Squared Array: [ 0  1  4  9 16 25 36 49 64 81]
Exponential Array: [1.00000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e+01
 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e+03
 2.98095799e+03 8.10308393e+03]
```

```
In [11]: #3. Data Aggregation with NumPy
```

```
In [12]: # Sample data
data = np.random.randn(1000)  # Generate 1000 random numbers from a normal distribution
```

```
In [14]: # Summary statistics
mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)
total_sum = np.sum(data)
```

```
In [15]: print("Mean:", mean)
print("Median:", median)
print("Standard Deviation:", std_dev)
print("Sum:", total_sum)

# Grouping and aggregation
grouped_data = np.array([np.mean(data[:500]), np.mean(data[500:])])
print("Mean of first half vs second half of data:", grouped_data)
```

```
Mean: -0.018233901641072463
Median: -0.01569645779579632
Standard Deviation: 1.0118017895096243
Sum: -18.233901641072464
Mean of first half vs second half of data: [-0.02003888 -0.01642893]
```

```
In [16]: #4. Data Analysis with NumPy
```

```
In [17]: # Generating data
data = np.random.randn(1000)
```

```
In [18]: # Identifying outliers
z_scores = (data - np.mean(data)) / np.std(data)
outliers = data[np.abs(z_scores) > 3]
```

```
In [19]: # Calculating percentiles
percentile_25 = np.percentile(data, 25)
percentile_50 = np.percentile(data, 50)  # 50th percentile is the median
percentile_75 = np.percentile(data, 75)
```

```
In [20]: print("Outliers:", outliers)
print("25th Percentile:", percentile_25)
print("50th Percentile (Median):", percentile_50)
print("75th Percentile:", percentile_75)
```

```
Outliers: [-3.08726646 -3.0559519   2.99821415]
25th Percentile: -0.6599084771826766
50th Percentile (Median): 0.07841222659245681
75th Percentile: 0.6885590621314583
```

```
In [ ]: #5. Application in Data Science
```

```
In [25]: print('''Numerical Computation: NumPy’s array operations are optimized for performance, making it significantly faster than Python lists, especially when working with large datasets.
Data Analysis: Functions for statistical operations (mean, median, std, etc.) are simple to use and run efficiently on large datasets.
Data Reshaping: The ability to reshape arrays without copying data is crucial when manipulating datasets in machine learning.
Outlier Detection and Correlations: Quickly identify patterns, correlations, and anomalies in data.
Real-World Examples:
Machine Learning: NumPy is often used to handle numerical data before feeding it into machine learning algorithms. Libraries like TensorFlow and PyTorch rely on concepts similar to NumPy arrays.
Financial Analysis: Calculating portfolio risk and returns using large time series data is made efficient with NumPy.
Scientific Research: In fields like astronomy and physics, where large datasets are common, NumPy is used for simulations and data analysis due to its performance and ease of use.
Conclusion
By incorporating NumPy into your workflow, you can enhance the efficiency of your data processing and analysis tasks, allowing for more complex and large-scale data manipulation than would be feasible with traditional Python data stru
```

```
Numerical Computation: NumPy’s array operations are optimized for performance, making it significantly faster than Python lists, especially when working with large datasets.
Data Analysis: Functions for statistical operations (mean, median, std, etc.) are simple to use and run efficiently on large datasets.
Data Reshaping: The ability to reshape arrays without copying data is crucial when manipulating datasets in machine learning.
Outlier Detection and Correlations: Quickly identify patterns, correlations, and anomalies in data.
Real-World Examples:
Machine Learning: NumPy is often used to handle numerical data before feeding it into machine learning algorithms. Libraries like TensorFlow and PyTorch rely on concepts similar to NumPy arrays.
Financial Analysis: Calculating portfolio risk and returns using large time series data is made efficient with NumPy.
Scientific Research: In fields like astronomy and physics, where large datasets are common, NumPy is used for simulations and data analysis due to its performance and ease of use.
Conclusion
```

By incorporating NumPy into your workflow, you can enhance the efficiency of your data processing and analysis tasks, allowing for more complex and large-scale data manipulation than would be feasible with traditional Python data structures.

In [ ]:

```

```