1.GETTING FAMILIARITY WITH NUMPY

```python
#importing numpy package
import numpy as np

# Creating a 1D array
array_1d = np.array([1, 2, 3, 4, 5])
print("1D Array:", array_1d)

# Creating a 2D array (matrix)
array_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("\n2D Array:\n", array_2d)

# Creating a 3D array
array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("\n3D Array:\n", array_3d)
```

```
1D Array: [1 2 3 4 5]

2D Array:
 [[1 2 3]
 [4 5 6]]

3D Array:
 [[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

```python
#basic properties of np array
#Shape: The dimensions of the array.
print("Shape of 1D array:", array_1d.shape)
print("Shape of 2D array:", array_2d.shape)
print("Shape of 3D array:", array_3d.shape)
#Size: The total number of elements.
print("\nSize of 2D array:", array_2d.size)
#Data Type: The type of elements stored in the array.
print("Data type of 1D array:", array_1d.dtype)
```

```
Shape of 1D array: (5,)
Shape of 2D array: (2, 3)
Shape of 3D array: (2, 2, 2)

Size of 2D array: 6
Data type of 1D array: int64
```

```
# basic operations
array_a = np.array([10, 20, 30])
array_b = np.array([1, 2, 3])

print("\nAddition:", array_a + array_b)
print("Subtraction:", array_a - array_b)
print("Multiplication:", array_a * array_b)
print("Division:", array_a / array_b)
```

```
Addition: [11 22 33]
Subtraction: [ 9 18 27]
Multiplication: [10 40 90]
Division: [10. 10. 10.]
```

2.DATA MANIPULATION

```
# Indexing and Slicing

# Indexing a 1D array
print("\nIndexing 1D Array (element at index 2):")
print(array_1d[2])
# Indexing a 2D array
print("\nIndexing 2D Array (element at [1, 2]):")
print(array_2d[1, 2])

# Slicing a 1D array
print("\nSlicing 1D Array (elements from index 1 to 4):")
print(array_1d[1:4])
# Slicing a 2D array
print("\nSlicing 2D Array (first two rows, last two columns):")
print(array_2d[:2, 1:3])

#Reshaping Arrays

# Reshaping a 1D array to a 2D array
reshaped_array = array_1d.reshape((5, 1))
print("\nReshaped 1D Array to 2D Array:")
print(reshaped_array)

#Mathematical Operations

# Element-wise addition
added_array = array_1d + 10
print("\nElement-wise Addition:")
print(added_array)

# Element-wise multiplication
```

```
multiplied_array = array_1d * 2
print("\nElement-wise Multiplication:")
print(multiplied_array)

# Element-wise square
squared_array = array_1d ** 2
print("\nElement-wise Square:")
print(squared_array)

# Matrix multiplication (dot product)
dot_product = np.dot(array_2d, array_2d.T)
print("\nMatrix Multiplication (Dot Product):")
print(dot_product)
```

```
Indexing 1D Array (element at index 2):
3

Indexing 2D Array (element at [1, 2]):
6

Slicing 1D Array (elements from index 1 to 4):
[2 3 4]

Slicing 2D Array (first two rows, last two columns):
[[2 3]
 [5 6]]

Reshaped 1D Array to 2D Array:
[[1]
 [2]
 [3]
 [4]
 [5]]

Element-wise Addition:
[11 12 13 14 15]

Element-wise Multiplication:
[ 2  4  6  8 10]

Element-wise Square:
[ 1  4  9 16 25]

Matrix Multiplication (Dot Product):
[[14 32]
 [32 77]]
```

3.DATA AGGREGATION

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

#Summary Statistics

# Mean
mean_value = np.mean(data)
print("Mean:", mean_value)

# Median
median_value = np.median(data)
print("Median:", median_value)

# Standard Deviation
std_dev = np.std(data)
print("Standard Deviation:", std_dev)

# Variance
print("Variance of the array")
print(np.var(data))

# Sum
sum_value = np.sum(data)
print("Sum:", sum_value)

# Min value in the array
print("Min value of the array")
print(np.min(data))

# Max value of the array
print("Max value of the array")
print(np.max(data))

#Cumulative Sum of the array
print("Cumulative Sum of the array")
print(np.cumsum(data))

#Cumulative Product of the array
print("Cumulative Product of the array")
print(np.cumprod(data))

#Percentile of the array
print("Percentile of the array")
np.percentile(data,50)

# Aggregation by axis
column_sum=np.sum(data, axis=0)
print("Sum along columns:",column_sum)

# Aggregation by axis
```

```
row_mean=np.mean(array_2d, axis=1)
print("Mean along rows:",row_mean)
```

```
Mean: 55.0
Median: 55.0
Standard Deviation: 28.722813232690143
Variance of the array
825.0
Sum: 550
Min value of the array
10
Max value of the array
100
Cumulative Sum of the array
[ 10  30  60 100 150 210 280 360 450 550]
Cumulative Product of the array
[               10               200              6000
240000
          12000000         720000000       50400000000
4032000000000
   362880000000000 3628800000000000]
Percentile of the array
Sum along columns: 550
Mean along rows: [2. 5.]
```

4.DATA ANALYSIS

```
# Correlation matrix
data2=np.random.rand(5,4)*100
print("Data for analysis:")
print(data2)

cor_arr=np.corrcoef(data2,rowvar=False)
print("Correlation matrix:")
print(cor_arr)

# Identifying outliers
outliers=data2[data2 > 95]
print("Outliers (values greater than 95):")
print(outliers)

# Calculating percentiles
percentile_90=np.percentile(data2,90)
print("\n90th percentile of the data:", percentile_90)
```

```
Data for analysis:
[[34.16930512 75.59231753 63.26063335 41.41646475]
 [ 5.42524509 80.66693457 81.1166432  57.80295798]
 [97.94254998 58.57363463 55.40231119 91.52494117]
 [83.84735297 34.45815431 71.82227343 98.35713428]
 [43.55753188 62.01795784 69.87368409 22.27126344]]
Correlation matrix:
[[ 1.         -0.78953511 -0.68955433  0.69736587]
 [-0.78953511  1.          0.16968348 -0.62383884]
 [-0.68955433  0.16968348  1.         -0.21225544]
 [ 0.69736587 -0.62383884 -0.21225544  1.        ]]
Outliers (values greater than 95):
[97.94254998 98.35713428]

90th percentile of the data: 92.16670204677972
```

5.ADVANTAGES OF NUMPY

The Role of NumPy in Data Science Key Advantages of Using NumPy: Performance and Efficiency:

Vectorized Operations: Faster execution by applying operations to entire arrays, avoiding Python loops. Memory Efficiency: Less memory usage compared to Python lists due to homogeneous data types. Broadcasting: Simplifies operations on arrays of different shapes. Mathematical Functionality:

Comprehensive Library: Optimized mathematical functions for efficient computation. Complex Operations: Simplifies tasks like matrix multiplication and Fourier transforms. Integration with Other Libraries:

Compatibility: Seamless integration with libraries like Pandas, SciPy, and scikit-learn. Data Handling: Efficient handling of large datasets, crucial for data science tasks. Real-World Applications of NumPy: Machine Learning:

Model Training: Efficient handling of datasets and matrix operations for algorithms. Data Preprocessing: Used in normalization, feature scaling, and dimensionality reduction. Financial Analysis:

Risk Management: Supports Monte Carlo simulations to model financial uncertainties. Portfolio Optimization: Helps in optimizing investment portfolios using matrix operations. Scientific Research:

Numerical Simulations: Used in solving equations and simulating physical systems. Data Analysis: Assists researchers in handling and analyzing large datasets.