Practical No. 02

Aim: To study and implement various methods of NumPy Library.

S/W Required: Python 3.9, Jupyter Notebook

Theory:

Introduction to NumPy:

NumPy (short for Numerical Python) provides an efficient interface to store and operate on dense data buffers. In some ways, NumPy arrays are like Python's built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python, so time spent learning to use NumPy effectively will be valuable no matter what aspect of data science interests you. Basic array manipulation includes:

1. Attributes of arrays: Determining the size, shape, memory consumption, and data types of arrays
2. Indexing of arrays: Getting and setting the value of individual array elements
3. Slicing of arrays: Getting and setting smaller subarrays within a larger array
4. Reshaping of arrays: Changing the shape of a given array
5. Joining and splitting of arrays: Combining multiple arrays into one, and splitting one array into many

Attributes of NumPy Array:

Each numpy array has attributes ndim (the number of dimensions), shape (the size of each dimension), and size (the total size of the array). Another useful attribute is the dtype, the data type of the array. Other attributes include itemsize, which lists the size (in bytes) of each array element, and nbytes, which lists the total size (in bytes) of the array.

Array Indexing:

Array indexing in NumPy is similar as that of the lists in python. In a one-dimensional array, you can access the i^{th} value (counting from zero) by specifying the desired index in square brackets. In a multidimensional array, you access items using a comma-separated tuple of

Indices.

Array Slicing:

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the slice notation, marked by the colon (:) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array x, use this: x[start:stop:step].

Reshaping and Joining:

Reshaping an array is another useful operation in NumPy, which is used to reshape the array, or you can say to change the shape of the array. Joining is used to merge two arrays together.

Computation on NumPy arrays:

Universal Functions:

Computation on NumPy array is very easy. Using Universal Functions available in NumPy this can be done effectively. Following are universal functions in NumPy.

1. Arithmetic Operations:

a.	np.add	(addition)
b.	np.subtract	(subtraction)
c.	np.negative	(unary negation)
d.	np.multiply	(multiplaction)
e.	np.divide	(division)
f.	np.floor_divide	(floor division)
g.	np.power	(exponentation)
h.	np.mod	(modulus)

2. Trigonometric Functions:
    a. np.sin()
    b. np.cos()
    c. np.tan()
    d. np.arcsin()
    e. np.arccos()
    f. np.arctan()

Aggregations:

The most common summary statistics are the mean and standard deviation, which allow you to summarize the “typical” values in a dataset, but other aggregates are useful as well (the sum, product, median, minimum and maximum, quantiles, etc.). NumPy has fast built-in aggregation functions for working on arrays.

1.	np.sum()	(Compute sum of elements)
2.	np.prod()	(Compute product of elements)
3.	np.mean()	(Compute median of elements)
4.	np.std()	(Compute standard deviation)
5.	np.var()	(Compute variance)
6.	np.min()	(Find minimum value)
7.	np.max()	(Find maximum value)
8.	np.argmin()	(Find index of minimum value)
9.	np.argmax()	(Find index of maximum value)
10.	np.median()	(Compute median of elements)
11.	np.percentile()	(Compute rank-based statistics of elements)
12.	np.any()	(Evaluate whether any elements are true)
13.	np.all()	(Evaluate whether all elements are true)

Code/Program:

Conclusion:

Thus, we have studied how python is used for data science with wide range of libraries and learned how to setup a python environment for data science.