42_1_NP_numpy_operations

December 15, 2023

1 NumPy Arithmetic Array Operations

1.1 NumPy Arithmetic Array Operations

NumPy provides a wide range of operations that can perform on arrays, including arithmetic operations.

NumPy's arithmetic operations are widely used due to their ability to perform simple and efficient calculations on arrays.

In this tutorial, we will explore some commonly used arithmetic operations in NumPy and learn how to use them to manipulate arrays.

1.2 List of Arithmetic Operations

Here's a list of various arithmetic operations along with their associated operators and built-in functions:

Element-wise Operation	Operator	Function
Addition	`+`	`np.add(a, b)`
Subtraction	`-`	`np.subtract(a, b)`
Multiplication	`*`	`np.multiply(a, b)`
Division	`/`	`np.divide(a, b)`
Floor Division	`//`	`np.floor_divide(a, b)`
Power	`**`	`np.power(a, b)`
Modulus	`%`	`np.mod(a, b)`
Trigonometric Functions	`sin`	`np.sin(a)`

To perform each operation, we can either use the associated operator or built-in functions. For example, to perform addition, we can either use the + operator or the add() built-in function.

Next, we will see examples of each of these operations.

1.2.1 NumPy Array Element-Wise Addition

As mentioned earlier, we can use the both + operator and the built-in function add() to perform element-wise addition between two NumPy arrays.

[8]:
```python
first_array = np.array([1, 3, 5, 7])
second_array = np.array([2, 4, 6, 8])
```

```
# using the + operator
result1 = first_array + second_array
print("Using the + operator:",result1)

# using the add() function
result2 = np.add(first_array, second_array)
print("Using the add() function:",result2)
```

```
Using the + operator: [ 3  7 11 15]
Using the add() function: [ 3  7 11 15]
```

In the above example, first we created two arrays named: first_array and second_array.

Then, we used the + operator and the add() function to perform element-wise addition respectively.

As we can see, both + and add() give the same result.

1.2.2 NumPy Array Element-Wise Subtraction

In NumPy, we can either use the - operator or the subtract() function to perform element-wise subtraction between two NumPy arrays.

[9]:
```
first_array = np.array([3, 9, 27, 81])
second_array = np.array([2, 4, 8, 16])

# using the - operator
result1 = first_array - second_array
print("Using the - operator:",result1)

# using the subtract() function
result2 = np.subtract(first_array, second_array)
print("Using the subtract() function:",result2)
```

```
Using the - operator: [ 1  5 19 65]
Using the subtract() function: [ 1  5 19 65]
```

Here, we have performed subtraction between first_array and second_array using both the - operator and the subtract() function.

1.3 NumPy Array Element-Wise Multiplication

For element-wise multiplication, we can use the * operator or the multiply() function.

[10]:
```
first_array = np.array([1, 3, 5, 7])
second_array = np.array([2, 4, 6, 8])

# using the * operator
result1 = first_array * second_array
print("Using the * operator:",result1)
```

```
# using the multiply() function
result2 = np.multiply(first_array, second_array)
print("Using the multiply() function:",result2)
```

```
Using the * operator: [ 2 12 30 56]
Using the multiply() function: [ 2 12 30 56]
```

Here, we have used * and multiply() to demonstrate two ways of multiplying the two arrays element-wise.

1.4 NumPy Array Element-Wise Division

We can use either the / operator or the divide() function to perform element-wise division between two numpy arrays.

[ ]:
```
first_array = np.array([1, 2, 3])
second_array = np.array([4, 5, 6])

# using the / operator
result1 = first_array / second_array
print("Using the / operator:",result1)

# using the divide() function
result2 = np.divide(first_array, second_array)
print("Using the divide() function:",result2)
```

Note : Here, we can see both / and divide() give the same result.

1.5 NumPy Array Element-Wise Exponentiation

Array exponentiation refers to raising each element of an array to a given power.

In NumPy, we can use either the ** operator or the power() function to perform the element-wise exponentiation operation.

[11]:
```
array1 = np.array([1, 2, 3])

# using the ** operator
result1 = array1 ** 2
print("Using the ** operator:",result1)

# using the power() function
result2 = np.power(array1, 2)
print("Using the power() function:",result2)
```

```
Using the ** operator: [1 4 9]
Using the power() function: [1 4 9]
```

In the above example, we have used the ** operator and the power() function to perform exponentiation operation respectively.

Here, ** and power() both raise each element of array1 to the power of 2.

1.6 NumPy Array Element-Wise Modulus

We can perform a modulus operation in NumPy arrays using the % operator or the mod() function.

This operation calculates the remainder of element-wise division between two arrays.

[15]:
```python
first_array = np.array([9, 10, 20])
second_array = np.array([2, 5, 7])

# using the % operator
result1 = first_array % second_array
print("Using the % operator:",result1)

# using the mod() function
result2 = np.mod(first_array, second_array)
print("Using the mod() function:",result2)
```

```
Using the % operator: [1 0 6]
Using the mod() function: [1 0 6]
```

Here, we have performed the element-wise modulus operation using both the % operator and the mod() function.

1.7 NumPy Array Functions

NumPy array functions are the built-in functions provided by NumPy that allow us to create and manipulate arrays, and perform different operations on them.

We will discuss some of the most commonly used NumPy array functions.

Common NumPy Array Functions

There are many NumPy array functions available but here are some of the most commonly used ones.

Array Operations	Functions
Array Creation Functions	np.array(), np.zeros(), np.ones(), np.empty(), etc.
Array Manipulation Functions	np.reshape(), np.transpose(), etc.
Array Mathematical Functions	np.add(), np.subtract(), np.sqrt(), np.power(), etc.
Array Statistical Functions	np.median(), np.mean(), np.std(), and np.var()
Array Input and Output Functions	np.save(), np.load(), etc.

1.7.1 NumPy Array Creation Functions

Array creation functions allow us to create new NumPy arrays.

[ ]:
```python
# create an array using np.array()
array1 = np.array([1, 3, 5])
```

```
print("np.array():\n", array1)

# create an array filled with zeros using np.zeros()
array2 = np.zeros((3, 3))
print("\nnp.zeros():\n", array2)

# create an array filled with ones using np.ones()
array3 = np.ones((2, 4))
print("\nnp.ones():\n", array3)
```

Here,

- np.array() - creates an array from a Python List
- np.zeros() - creates an array filled with zeros of the specified shape
- np.ones() - creates an array filled with ones of the specified shape

1.7.2 NumPy Array Manipulation Functions

NumPy array manipulation functions allow us to modify or rearrange NumPy arrays.

[19]:
```
# create a 1D array
array1 = np.array([1, 3, 5, 7, 9, 11])

# reshape the 1D array into a 2D array
array2 = np.reshape(array1, (2, 3))

# transpose the 2D array
array3 = np.transpose(array2)

print("Original array:\n", array1)
print("\nReshaped array:\n", array2)
print("\nTransposed array:\n", array2.T)
print("\nTransposed array:\n", array3)
```

```
Original array:
 [ 1  3  5  7  9 11]

Reshaped array:
 [[ 1  3  5]
 [ 7  9 11]]

Transposed array:
 [[ 1  7]
 [ 3  9]
 [ 5 11]]

Transposed array:
 [[ 1  7]
 [ 3  9]
 [ 5 11]]
```

In this example,

- np.reshape(array1, (2, 3)) - reshapes array1 into 2D array with shape (2,3)
- np.transpose(array2) - transposes 2D array array2

1.7.3 NumPy Array Mathematical Functions

In NumPy, there are tons of mathematical functions to perform on arrays.

[ ]:
```
# create two arrays
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([4, 9, 16, 25, 36])

# add the two arrays element-wise
arr_sum = np.add(array1, array2)

# subtract the array2 from array1 element-wise
arr_diff = np.subtract(array1, array2)

# compute square root of array2 element-wise
arr_sqrt = np.sqrt(array2)


print("\nSum of arrays:\n", arr_sum)
print("\nDifference of arrays:\n", arr_diff)
print("\nSquare root of first array:\n", arr_sqrt)
```

1.7.4 NumPy Array Statistical Functions

NumPy provides us with various statistical functions to perform statistical data analysis.

These statistical functions are useful to find basic statistical concepts like mean, median, variance, etc. It is also used to find the maximum or the minimum element in an array.

[20]:
```
# create a numpy array
marks = np.array([76, 78, 81, 66, 85])

# compute the mean of marks
mean_marks = np.mean(marks)
print("Mean:",mean_marks)

# compute the median of marks
median_marks = np.median(marks)
print("Median:",median_marks)

# find the minimum and maximum marks
min_marks = np.min(marks)
print("Minimum marks:", min_marks)
```

```
max_marks = np.max(marks)
print("Maximum marks:", max_marks)
```

```
Mean: 77.2
Median: 78.0
Minimum marks: 66
Maximum marks: 85
```

Here, computed the mean, median, minimum, and maximum of the given array marks.

## NumPy Array Input/Output Functions

NumPy offers several input/output (I/O) functions for loading and saving data to and from files.

[ ]:
```
# create an array
array1 = np.array([[1, 3, 5], [2, 4, 6]])

# save the array to a text file
np.savetxt('data.txt', array1)

# load the data from the text file
loaded_data = np.loadtxt('array1.txt')

# print the loaded data
print(loaded_data)
```

In this example, we first created the 2D array named array1 and then saved it to a text file using the np.savetxt() function.

We then loaded the saved data using the np.loadtxt() function.

1.8 Numpy Comparison/Logical Operations

NumPy provides several comparison and logical operations that can be performed on NumPy arrays.

NumPy's comparison operators allow for element-wise comparison of two arrays.

Similarly, logical operators perform boolean algebra, which is a branch of algebra that deals with True and False statements.

First we'll discuss comparison operations and then about logical operations in NumPy.

1.8.1 NumPy Comparison Operators

NumPy provides various element-wise comparison operators that can compare the elements of two NumPy arrays.

Here's a list of various comparison operators available in NumPy.

Operators	Description
< (less than)	Returns `True` if the element in the first array is less than the element in the second array.
<= (less than or equal to)	Returns `True` if the element in the first array is less than or equal to the element in the second array.
> (greater than)	Returns `True` if the element in the first array is greater than the element in the second array.
>= (greater than or equal to)	Returns `True` if the element in the first array is greater than or equal to the element in the second array.
== (equal to)	Returns `True` if the element in the first array is equal to the element in the second array.
!= (not equal to)	Returns `True` if the element in the first array is not equal to the element in the second array.

Example 1: NumPy Comparison Operators

[21]:
```
array1 = np.array([1, 2, 3])
array2 = np.array([3, 2, 1])

# less than operator
result1 = array1 < array2
print("array1 < array2:",result1)    # Output: [ True False False]

# greater than operator
result2 = array1 > array2
print("array1 > array2:",result2)    # Output: [False False  True]

# equal to operator
result3 = array1 == array2
print("array1 == array2:",result3)    # Output: [False  True False]
```

```
array1 < array2: [ True False False]
array1 > array2: [False False  True]
array1 == array2: [False  True False]
```

Here, we can see that the output of the comparison operators is also an array, where each element is either True or False based on the array element's comparison.

### NumPy Comparison Functions

NumPy also provides built-in functions to perform all the comparison operations.

For example, the less() function returns True if each element of the first array is less than the corresponding element in the second array.

Here's a list of all built-in comparison functions.

Function	Description
np.less()	Returns True if the element in the first array is less than the element in the second array.
np.less_equal()	Returns True if the element in the first array is less than or equal to the element in the second array.
np.greater()	Returns True if the element in the first array is greater than the element in the second array.
np.greater_equal()	Returns True if the element in the first array is greater than or equal to the element in the second array.
np.equal()	Returns True if the element in the first array is equal to the element in the second array.
np.not_equal()	Returns True if the element in the first array is not equal to the element in the second array.

Note : al these operations are element-wise

1.8.2 Example 2: NumPy Comparison Functions

[ ]:
```
array1 = np.array([9, 12, 21])
array2 = np.array([21, 12, 9])

# use of less()
result = np.less(array1, array2)
print("Using less():",result)    # Output: [ True False False]

# use of less_equal()
result = np.less_equal(array1, array2)
print("Using less_equal():",result)     # Output: [ True  True False]

# use of greater()
result = np.greater(array1, array2)
print("Using greater():",result)    # Output: [False False  True]

# use of greater_equal()
result = np.greater_equal(array1, array2)
print("Using greater_equal():",result)    # Output: [False  True  True]

# use of equal()
result = np.equal(array1, array2)
print("Using equal():",result)    # Output: [False  True False]

# use of not_equal()
result = np.not_equal(array1, array2)
print("Using not_equal():",result)    # Output: [ True False  True]
```

1.9 NumPy Logical Operations

As mentioned earlier, logical operators perform Boolean algebra; a branch of algebra that deals with True and False statements.

Logical operations are performed element-wise. For example, if we have two arrays x1 and x2 of the same shape, the output of the logical operator will also be an array of the same shape.

Here's a list of various logical operators available in NumPy:

Function	Description
np.logical_and()	Returns `True` if the element in the first array is less than the element in the second array.
np.logical_or()	Returns `True` if the element in the first array is less than the element in the second array.
np.logical_not()	Returns `True` if the element in the first array is less than the element in the second array.

Example 3: NumPy Logical Operations

[22]:
```
x1 = np.array([True, False, True])
x2 = np.array([False, False, True])

# Logical AND
print(np.logical_and(x1, x2)) # Output: [False False  True]

# Logical OR
print(np.logical_or(x1, x2)) # Output: [ True False  True]

# Logical NOT
print(np.logical_not(x1)) # Output: [False  True False]
```

```
[False False  True]
[ True False  True]
[False  True False]
```

Here, we have performed logical operations on two arrays, x1 and x2, containing boolean values.

It demonstrates the logical AND, OR, and NOT operations using np.logical_and(), np.logical_or(), and np.logical_not() respectively.

### NumPy Math Functions

Numpy provides a wide range of mathematical functions that can be performed on arrays.

Let's explore three different types of math functions in NumPy:

- 1 Trigonometric Functions
- 2 Arithmetic Functions
- 3 Rounding Functions

#### 1. Trigonometric Functions

NumPy provides a set of standard trigonometric functions to calculate the trigonometric ratios (sine, cosine, tangent, etc.)

Here's a list of commonly used trigonometric functions in NumPy.

Trigonometric Functiion	Computes (in radians)
np.sin()	Trigonometric sine of an angle
np.cos()	Trigonometric cosine of an angle
np.tan()	Trigonometric tangent of an angle
np.arcsin()	Inverse trigonometric sine of an angle
np.arccos()	Inverse trigonometric cosine of an angle
np.arctan()	Inverse trigonometric tangent of an angle
np.degrees	Converts an angle from radians to degrees
np.radians	Converts an angle from degrees to radians

[25]:
```
# array of angles in radians
angles = np.array([0, 1, 1])
print("Angles:", angles)

# compute the sine of the angles
sine_values = np.sin(angles)
print("Sine values:", sine_values)

# compute the inverse sine of the angles
inverse_sine = np.arcsin(angles)
print("Inverse Sine values:", inverse_sine)
```

```
Angles: [0 1 1]
Sine values: [0.         0.84147098 0.84147098]
Inverse Sine values: [0.         1.57079633 1.57079633]
```

In this example, the sin() and arcsin() functions calculate the sine and inverse sine values, respectively, for each element in the angles array.

The resulting values are in radians.

Now let's see the examples for degrees() and radians().

[ ]:
```
# define an angle in radians
angle =  1.57079633
print("Initial angle in radian:", angle)

# convert the angle to degrees
angle_degree = np.degrees(angle)
print("Angle in degrees:", angle_degree)

# convert the angle back to radians
angle_radian = np.radians(angle_degree)
print("Angle in radians (after conversion):", angle_radian)
```

Here, we first initialized an angle in radians. Then we converted it to degrees using the degrees() function.

Similarly, we used radians() to convert the degrees back to radians.

2. Arithmetic Functions NumPy provides a wide range of arithmetic functions to perform on arrays.

Here's a list of various arithmetic functions along with their associated operators:

Arithmetic Function	Operator	Description
np.add()	+	Addition
np.subtract()	-	Subtraction
np.multiply()	*	Multiplication
np.divide()	/	Division
np.mod()	%	Modulus
np.power()	**	Exponentiation

[ ]:
```
ifirst_array = np.array([1, 3, 5, 7])
second_array = np.array([2, 4, 6, 8])

# using the add() function
result2 = np.add(first_array, second_array)
print("Using the add() function:",result2)
```

In the above example, first we created two arrays named: first_array and second_array. Then, we used the add() function to perform element-wise addition respectively.

3. Rounding Functions We use rounding functions to round the values in an array to a specified number of decimal places.

Here's a list of commonly used NumPy rounding functions:

Rounding Function	Description
np.round()	Rounds the values in an array to a specified number of decimal places
np.floor()	Rounds the values in an array to the nearest integer below the value
np.ceil()	Rounds the values in an array to the nearest integer above the value

[30]:
```
numbers = np.array([1.23456, 2.34567, 3.45678, 4.56789])

# round the array to two decimal places
rounded_array = np.round(numbers, 2)

print(rounded_array)
```

```
# Output: [1.23 2.35 3.46 4.57]
```

```
[1.23 2.35 3.46 4.57]
```

Here, we used the round() function to round the values of array numbers. Notice the line,

`np.round(numbers, 2)`

We've given two arguments to the round() function.

numbers - the array whose values are to be rounded 2 - denotes the number of decimal places to which the array is rounded

[31]:
```
array1 = np.array([1.23456, 2.34567, 3.45678, 4.56789])

print("Array after floor():", np.floor(array1))
print("Array after ceil():", np.ceil(array1))
```

```
Array after floor(): [1. 2. 3. 4.]
Array after ceil(): [2. 3. 4. 5.]
```

In the above example, the floor() function rounds the values of array1 down to the nearest integer that is less than or equal to each element.

Whereas, the ceil() function rounds the values of array1 up to the nearest integer that is greater than or equal to each element.

## Numpy Constants

NumPy constants are the predefined fixed values used for mathematical calculations.

For example, np.pi is a mathematical constant that returns the value of pi ( ), i.e. 3.141592653589793.

Using predefined constants makes our code concise and easier to read.

We'll discuss some common constants provided by Numpy library.

1.10 Numpy Constants

In Numpy, the most commonly used constants are pi and e.

1.10.1 np.pi

np.pi is a mathematical constant that returns the value of pi( ) as a floating point number. Its value is approximately 3.141592653589793.

[33]:
```
radius = 2
circumference = 2 * np.pi * radius
print(circumference)
```

```
12.566368
```

Here, the np.pi returns the value 3.141592653589793, which calculates the circumference.

Instead of the long floating point number, we used the constant np.pi. This made our code look cleaner.

Note : As pi falls under the Numpy library, we need to import and access it with np.pi.

1.10.2 np.e

np.e is widely used with exponential and logarithmic functions. Its value is approximately 2.718281828459045.

```
[34]: y = np.e
      y
```

```
[34]: 2.718281828459045
```

in the previous example, we simply printed the constant np.e. As we can see, it returns its value 2.718281828459045.

However, this example makes no sense because that's not how we use the constant e in real life.

We usually use the constant e with the function exp(). e is the base of exponential function, exp(x), which is equivalent to e^x.

```
[37]: x = 1
      y = np.exp(x)
      y
```

```
[37]: 2.718281828459045
```

### Arithmetic Operations with Numpy Constants and Arrays

We can use arithmetic operators to perform operations such as addition, subtraction, and division between a constant and all elements in an array.

```
[38]: array1 = np.array([1, 2, 3])

      # add each array element with the constant pi
      y = array1 + np.pi
      y
```

```
[38]: array([4.14159265, 5.14159265, 6.14159265])
```

1.11 NumPy Statistical Functions

Statistics involves gathering data, analyzing it, and drawing conclusions based on the information collected.

NumPy provides us with various statistical functions that can perform statistical data analysis.

1.12 Common NumPy Statistical Functions

Here are some of the statistical functions provided by NumPy:

Function	Description
np.mean()	Calculates the mean of the values in an array
np.median()	Calculates the median of the values in an array
np.std()	Calculates the standard deviation of the values in an array
np.var()	Calculates the variance of the values in an array
np.min()	Calculates the minimum value in an array
np.max()	Calculates the maximum value in an array
np.sum()	Calculates the sum of the values in an array
np.percentile()	Calculates the percentile of the values in an array

### Find Median Using NumPy

The median value of a numpy array is the middle value in a sorted array.

In other words, it is the value that separates the higher half from the lower half of the data.

Suppose we have the following list of numbers:

1, 5, 7, 8, 9, 12, 14

Then, median is simply the middle number, which in this case is 8.

It is important to note that if the number of elements is

- **Odd**, the median is the middle element.
- **Even**, the median is the average of the two middle elements.

Now, we will learn how to calculate the median using NumPy for arrays with odd and even number of elements.

[39]:
```python
# create a 1D array with 5 elements
array1 = np.array([1, 2, 3, 4, 5])

# calculate the median
median = np.median(array1)
print(median)
```

```
3.0
```

In the above example, the array named array1 contains an odd number of elements (5 elements).

So, np.median(array1) returns the median of array1 as 3, which is the middle value of the sorted array

1.12.1 Example 2: Compute Median for Even Number of Elements

[40]:
```python
# create a 1D array with 6 elements
array1 = np.array([1, 2, 3, 4, 5, 7])

# calculate the median
median = np.median(array1)
print(median)
```

```
3.5
```

Here, since the array1 array has an even number of elements (6 elements), the median is calculated as the average of the two middle elements (3 and 4) i.e. 3.5.

1.12.2 Median of NumPy 2D Array

Calculation of the median is not just limited to 1D array. We can also calculate the median of the 2D array.

In a 2D array, median can be calculated either along the horizontal or the vertical axis individually, or across the entire array.

When computing the median of a 2D array, we use the `axis` parameter inside `np.median()` to specify the axis along which to compute the median.

If we specify,

- `axis = 0`, median is calculated along vertical axis
- `axis = 1`, median is calculated along horizontal axis
- If we don't use the `axis` parameter, the median is computed over the entire array.

[7]:
```python
import numpy as np
# create a 2D array
array1 = np.array([[10, 4, 12],
                    [2, 8, 6],
                   [14, 16, 18]])

# compute median along horizontal axis
result1 = np.median(array1, axis=1)

print("Median along horizontal axis :", result1)

# compute median along vertical axis
result2 = np.median(array1, axis=0)

print("Median along vertical axis:", result2)

# compute median of entire array
result3 = np.median(array1)

print("Median of entire array:", result3)
```

```
Median along horizontal axis : [10.  6. 16.]
Median along vertical axis: [10.  8. 12.]
Median of entire array: 10.0
```

In this example, we have created a 2D array named array1.

We then computed the median along the horizontal and vertical axis individually and then computed the median of the entire array.

- np.median(array1, axis=1) - median along horizontal axis, which gives [4. 10. 16.]
- np.median(array1, axis=0) - median along vertical axis, which gives [8. 10. 12.]
- np.median(array1) - median over the entire array, which gives 10.0

To calculate the median over the entire 2D array, first we flatten the array to [2, 4, 6, 8, 10, 12, 14, 16, 18] and then find the middle value of the flattened array which in our case is 10.

#### Compute Mean Using NumPy

The mean value of a NumPy array is the average value of all the elements in the array.

It is calculated by adding all elements in the array and then dividing the result by the total number of elements in the array.

We use the np.mean() function to calculate the mean value.

[ ]:
```
# create a numpy array
marks = np.array([76, 78, 81, 66, 85])

# compute the mean of marks
mean_marks = np.mean(marks)

print(mean_marks)

# Output: 77.2
```

Example 3: Mean of NumPy N-d Array

[45]:
```
# create a 2D array
array1 = np.array([[1, 3],
                  [5, 7]])

# calculate the mean of the entire array
result1 = np.mean(array1)
print("Entire Array:",result1)  # 4.0

# calculate the mean along vertical axis (axis=0)
result2 = np.mean(array1, axis=0)
print("Along Vertical Axis:",result2)  # [3. 5.]

# calculate the mean along  (axis=1)
result3 = np.mean(array1, axis=1)
print("Along Horizontal Axis :",result3)  # [2. 6.]
```

```
Entire Array: 4.0
Along Vertical Axis: [3. 5.]
Along Horizontal Axis : [2. 6.]
```

Here, first we have created the 2D array named array1. We then calculated the mean using np.mean().

- np.mean(array1) - calculates the mean over the entire array
- np.mean(array1, axis=0) - calculates the mean along vertical axis
- np.mean(array1, axis=1) calculates the mean along horizontal axis

#### Standard Deviation of NumPy Array

The standard deviation is a measure of the spread of the data in the array. It gives us the degree to which the data points in an array deviate from the mean.

Smaller standard deviation indicates that the data points are closer to the mean Larger standard deviation indicates that the data points are more spread out. In NumPy, we use the np.std() function to calculate the standard deviation of an array.

[46]:
```
# create a numpy array
marks = np.array([76, 78, 81, 66, 85])

# compute the standard deviation of marks
std_marks = np.std(marks)
print(std_marks)
```

```
6.368673331236263
```

In the above example, we have used the np.std() function to calculate the standard deviation of the marks array.

Here, 6.803568381206575 is the standard deviation of marks. It tells us how much the values in the marks array deviate from the mean value of the array.

Standard Deviation of NumPy 2D Array In a 2D array, standard deviation can be calculated either along the horizontal or the vertical axis individually, or across the entire array.

Similar to mean and median, when computing the standard deviation of a 2D array, we use the axis parameter inside np.std() to specify the axis along which to compute the standard deviation.

[47]:
```
# create a 2D array
array1 = np.array([[2, 5, 9],
                  [3, 8, 11],
                  [4, 6, 7]])

# compute standard deviation along horizontal axis
result1 = np.std(array1, axis=1)
print("Standard deviation along horizontal axis:", result1)

# compute standard deviation along vertical axis
result2 = np.std(array1, axis=0)
```

```
print("Standard deviation  along vertical axis:", result2)

# compute standard deviation of entire array
result3 = np.std(array1)
print("Standard deviation of entire array:", result3)
```

```
Standard deviation along horizontal axis: [2.86744176 3.29983165 1.24721913]
Standard deviation  along vertical axis: [0.81649658 1.24721913 1.63299316]
Standard deviation of entire array: 2.7666443551086073
```

Here, we have created a 2D array named array1.

We then computed the standard deviation along horizontal and vertical axis individually and then computed the standard deviation of the entire array.

Compute Percentile of NumPy Array In NumPy, we use the percentile() function to compute the nth percentile of a given array.

[48]:
```
# create an array
array1 = np.array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19])

# compute the 25th percentile of the array
result1 = np.percentile(array1, 25)
print("25th percentile:",result1)

# compute the 75th percentile of the array
result2 = np.percentile(array1, 75)
print("75th percentile:",result2)
```

```
25th percentile: 5.5
75th percentile: 14.5
```

Here,

- 25% of the values in array1 are less than or equal to 5.5.
- 75% of the values in array1 are less than or equal to 14.5.

Find Minimum and Maximum Value of NumPy Array We use the min() and max() function in NumPy to find the minimum and maximum values in a given array.

[49]:
```
# create an array
array1 = np.array([2,6,9,15,17,22,65,1,62])

# find the minimum value of the array
min_val = np.min(array1)

# find the maximum value of the array
max_val = np.max(array1)

# print the results
```

```
print("Minimum value:", min_val)
print("Maximum value:", max_val)
```

```
Minimum value: 1
Maximum value: 65
```

As we can see min() and max() returns the minimum and maximum value of array1 which is 1 and 65 respectively.

1.13 NumPy String Functions

In addition to NumPy's numerical capabilities, it also provides several functions that can be applied to strings represented in NumPy arrays.

For example, we can add two strings, change the contents of a string, case conversion, padding, trimming, and so on.

1.13.1 Common NumPy String Functions

Here are some of the string functions provided by NumPy:

Functions	Descriptions
capitalize()	Converts the first character of each word to uppercase
lower()	Converts the string to lowercase
upper()	Converts the string to uppercase
swapcase()	Swaps the case of the string
add()	concatenate two strings
multiply()	repeat a string
equal()	check if two strings are equal

Note : These are the most commonly used string functions in NumPy.

You can find more string functions in the NumPy strings documentation

1.13.2 NumPy String add() Function

In NumPy, we use the np.char.add() function to perform element-wise string concatenation for two arrays of strings.

[50]:
```
array1 = np.array(['iPhone: ', 'price: '])
array2 = np.array(['15', '$900'])

# perform element-wise array string concatenation
result = np.char.add(array1, array2)

print(result)
```

```
['iPhone: 15' 'price: $900']
```

In this example, we have defined two NumPy arrays: array1 and array2, with string elements ‘iphone:’ and ‘price:’ and ‘15’ and ‘$900’ respectively.

We then used np.char.add(array1, array2) to concatenate the elements of array1 and array2 element-wise.

Finally, we have stored the concatenated string in result, which in our case are ‘iPhone: 15’ and ‘price: $900’.

1.13.3 NumPy String multiply() Function

The np.char.multiply() function is used to perform element-wise string repetition. It returns an array of strings with each string element repeated a specified number of times.

[52]:
```python
#  define array with three string elements
array1 = np.array(['A', 'B', 'C'])

#  repeat each element in array1 two times
result = np.char.multiply(array1, 2)

print(result)
```

```
['AA' 'BB' 'CC']
```

Here, np.char.multiply(array1, 2) repeats each element in array1 two times.

1.13.4 NumPy String capitalize() Function

In NumPy, the np.char.capitalize() function is used to capitalize the first character of each string element in a given array.

[53]:
```python
# define an array with three string elements
array1 = np.array(['eric', 'paul', 'sean'])

# capitalize the first letter of each string in array1
result = np.char.capitalize(array1)

print(result)
```

```
['Eric' 'Paul' 'Sean']
```

In this example, np.char.capitalize(array1) capitalizes the first character of the string element in the array1 array.

1.13.5 NumPy String upper() and lower() Function

NumPy provides two string functions for converting the case of a string: np.char.upper() and np.char.lower().

[ ]:
```python
array1 = np.array(['nEpalI', 'AmeriCAN', 'CaNadIan'])

# convert all string elements to uppercase
```

```
result1 = np.char.upper(array1)

# convert all string elements to lowercase
result2 = np.char.lower(array1)

print("To Uppercase: ",result1)
print("To Lowercase: ",result2)
```

Here, - np.char.upper(array1) - convert all strings in array1 to uppercase - np.char.lower(array1) - convert all strings in array1 to lowercase

1.13.6 NumPy String join() Function

In NumPy, we use the np.char.join() function to join strings in an array with a specified delimiter.

[54]:
```
# create an array of strings
array1 = np.array(['hello', 'world'])

# join the strings in the array using a dash as the delimiter
result = np.char.join('-', array1)

print(result)
```

```
['h-e-l-l-o' 'w-o-r-l-d']
```

In this example, the np.char.join('-', array1) function is used to join the strings in array1 using dash - as the delimiter.

The resulting strings have each character separated by a dash.

1.13.7 NumPy String equal() Function

The np.char.equal() function is used to perform element-wise string comparison between two string arrays.

[ ]:
```
# create two arrays of strings
array1 = np.array(['C', 'Python', 'Swift'])
array2 = np.array(['C++', 'Python', 'Java'])

# check if each element of the arrays is equal
result = np.char.equal(array1, array2)

print(result)

# Output: [False True False]
```

Here, np.char.equal(array1, array2) checks whether the corresponding elements of array1 and array2 are equal or not.

The resulting boolean array [False True False] indicates that the first and third elements of array1 and array2 are not equal, while the second element is equal.