Numpy Mathematical Operations

-MUKESH KUMAR

AGENDA

- Min, max, argmax, argmin
- Rounding Functions
- Arithmetic operations
- Aggregate Functions
- Array Manipulation
- Array Sorting

Min

- Numpy.min: Return the minimum of an array or minimum along an axis.

```
a = np.arange(4).reshape((2,2))
a
array([[0, 1],
       [2, 3]])
np.min(a)           # Minimum of the flattened array
0
np.min(a, axis=0)   # Minima along the first axis
array([0, 1])
np.min(a, axis=1)   # Minima along the second axis
array([0, 2])
```

Max

- **numpy.max :** Return the maximum of an array or maximum along an axis.

```
a = np.arange(4).reshape((2,2))
a
array([[0, 1],
       [2, 3]])
np.max(a)           # Maximum of the flattened array
3
np.max(a, axis=0)   # Maxima along the first axis
array([2, 3])
np.max(a, axis=1)   # Maxima along the second axis
array([1, 3])
```

Argmin

- **numpy.argmin** : Returns the indices of the minimum values along an axis.

```
a = np.arange(6).reshape(2,3) + 10
a
array([[10, 11, 12],
       [13, 14, 15]])
np.argmin(a)
0
np.argmin(a, axis=0)
array([0, 0, 0])
np.argmin(a, axis=1)
array([0, 0])
```

Argmax

- **numpy.argmax** : Returns the indices of the maximum values along an axis.

```
a = np.arange(6).reshape(2,3) + 10
a
array([[10, 11, 12],
       [13, 14, 15]])
np.argmax(a)
5
np.argmax(a, axis=0)
array([1, 1, 1])
np.argmax(a, axis=1)
array([2, 2])
```

Rounding Functions

- Round: `np.round(x)`
- Floor: `np.floor(x)`
- Ceil: `np.ceil(x)`

Round

- **numpy.round :** Evenly round to the given number of decimals.
  - Round Half to Even
  - The "round half to even" rule states that when a number is exactly halfway between two integers (e.g., 2.5 is halfway between 2 and 3, and 5.5 is halfway between 5 and 6), it is rounded to the nearest even integer.

```
np.round([0.37, 1.64])
array([0., 2.])
np.round([0.37, 1.64], decimals=1)
array([0.4, 1.6])
np.round([.5, 1.5, 2.5, 3.5, 4.5]) # rounds to nearest even value
array([0., 2., 2., 4., 4.])
```

Floor

- **numpy.floor :** Converts a number to the lowest possible integer that is less than or equal to the original number. It always rounds down, even if the number is negative, making it the closest lower integer.

```
a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
np.floor(a)
array([-2., -2., -1.,  0.,  1.,  1.,  2.])
```

Ceil

- **numpy.ceil**
- If the input is a positive number with a decimal part, np.ceil() will round it up to the next whole number.
- If the input is a negative number, np.ceil() will still round it up (toward positive infinity).

```
a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
np.ceil(a)
array([-1., -1., -0.,  1.,  2.,  2.,  2.])
```

Aggregation Functions

- Sum: `np.sum(x)`
- Product: `np.prod(x)`
- Cumulative Sum: `np.cumsum(x)`
- Cumulative Product: `np.cumprod(x)`

Sum

- **numpy.sum :** Sum of array elements over a given axis.

```
np.sum([0.5, 1.5])
2.0
np.sum([0.5, 0.7, 0.2, 1.5], dtype=np.int32)
1
np.sum([[0, 1], [0, 5]])
6
np.sum([[0, 1], [0, 5]], axis=0)
array([0, 6])
np.sum([[0, 1], [0, 5]], axis=1)
array([1, 5])
```

Product

- **numpy.prod :** Return the product of array elements over a given axis.

```
By default, calculate the product of all elements:

import numpy as np
np.prod([1.,2.])
2.0
Even when the input array is two-dimensional:

a = np.array([[1., 2.], [3., 4.]])
np.prod(a)
24.0
But we can also specify the axis over which to multiply:

np.prod(a, axis=1)
array([  2.,  12.])
np.prod(a, axis=0)
array([3., 8.])
```

Cumulative Sum

- **numpy.cumsum :** Return the cumulative sum of the elements along a given axis.

```
a = np.array([[1,2,3], [4,5,6]])
a
array([[1, 2, 3],
       [4, 5, 6]])
np.cumsum(a)
array([ 1,  3,  6, 10, 15, 21])
np.cumsum(a, dtype=float)     # specifies type of output value(s)
array([  1.,   3.,   6.,  10.,  15.,  21.])
np.cumsum(a,axis=0)      # sum over rows for each of the 3 columns
array([[1, 2, 3],
       [5, 7, 9]])
np.cumsum(a,axis=1)      # sum over columns for each of the 2 rows
array([[ 1,  3,  6],
       [ 4,  9, 15]])
```

Cumulative Product

- **numpy.cumprod :** Return the cumulative product of elements along a given axis.

```
a = np.array([1,2,3])
np.cumprod(a) # intermediate results 1, 1*2
              # total product 1*2*3 = 6
array([1, 2, 6])
a = np.array([[1, 2, 3], [4, 5, 6]])
np.cumprod(a, dtype=float) # specify type of output
array([   1.,    2.,    6.,   24.,  120.,  720.])
The cumulative product for each column (i.e., over the rows) of a:

np.cumprod(a, axis=0)
array([[ 1,  2,  3],
       [ 4, 10, 18]])
The cumulative product for each row (i.e. over the columns) of a:

np.cumprod(a,axis=1)
array([[  1,   2,   6],
       [  4,  20, 120]])
```

Reshape

- **numpy.reshape :** Gives a new shape to an array without changing its data.

```
a = np.arange(6).reshape((3, 2))
a
array([[0, 1],
       [2, 3],
       [4, 5]])
-------------------------------
a = np.array([[1,2,3], [4,5,6]])
np.reshape(a, 6)
array([1, 2, 3, 4, 5, 6])

-------------------------------
a = np.arange(6).reshape((3, 2))
a
array([[0, 1],
       [2, 3],
       [4, 5]])


np.reshape(a, (2, 3)) ]
array([[0, 1, 2],
       [3, 4, 5]])
```

Flatten

- **numpy.ndarray.flatten** : Return a copy of the array collapsed into one dimension.

```
a = np.array([[1,2], [3,4]])
a.flatten()
array([1, 2, 3, 4])
a.flatten('F')
array([1, 3, 2, 4])
```

Concatenate

- **numpy.concatenate** : Join a sequence of arrays along an existing axis.

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
np.concatenate((a, b), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])
np.concatenate((a, b.T), axis=1)
array([[1, 2, 5],
       [3, 4, 6]])
np.concatenate((a, b), axis=None)
array([1, 2, 3, 4, 5, 6])
```

Arithmetic Operations

- Addition: `np.add(x, y)` or `x + y`
- Subtraction: `np.subtract(x, y)` or `x - y`
- Multiplication: `np.multiply(x, y)` or `x * y`
- Division: `np.divide(x, y)` or `x / y`
- Modulus: `np.mod(x, y)` or `x % y`
- Power: `np.power(x, y)` or `x ** y`

Statistical Functions

- Mean: `np.mean(x)`
- Median: `np.median(x)`
- Standard Deviation: `np.std(x)`
- Variance: `np.var(x)`
- Min: `np.min(x)`
- Max: `np.max(x)`

- Note: Covered in Notebook