**1. Numpy란**

Numpy는 C언어로 구현된 파이썬 라이브러리로써, 고성능의 수치계산을 위해 제작되었습니다. Numerical Python의 줄임말이기도 한 Numpy는 벡터 및 행렬 연산에 있어서 매우 편리한 기능을 제공합니다.

또한 이는 데이터분석을 할 때 사용되는 라이브러리인 pandas와 matplotlib의 기반으로 사용되기도 합니다.

numpy에서는 기본적으로 array라는 단위로 데이터를 관리하며 이에 대해 연산을 수행합니다. array는 말그대로 행렬이라는 개념으로 생각하시면 됩니다.

먼저 numpy를 사용하기 위해서는 아래와 같은 코드로 numpy를 import해야 합니다.

***import numpy as np***

사실상, numpy를 설치하고 단순히 import numpy 만 해도 되지만, 이를 코드에서 보다 편하게 사용하기 위해 as np 를 붙임으로써 np라는 이름으로 numpy를 사용합니다.

## Numpy 기초

### 1. Numpy 란?

*# numpy 사용하기*

**import** **numpy** **as** **np**

### 2. Array 정의 및 사용하기

|  |
| --- |
| data1 = [1,2,3,4,5]  print(data1) |
| [1, 2, 3, 4, 5] |

|  |
| --- |
| data2 = [1,2,3,3.5,4]  print(data2) |
| [1, 2, 3, 3.5, 4] |

|  |
| --- |
| *# numpy를 이용하여 array 정의하기*  *# 1. 위에서 만든 python list를 이용*  arr1 = np.array(data1)  print(arr1)  print(arr1.shape) *# array의 형태(크기)를 확인할 수 있다.* |
| [1 2 3 4 5]  arr1 : (5,) |

|  |
| --- |
| *# 2. 바로 python list를 넣어 줌으로써 만들기*  arr2 = np.array([1,2,3,4,5]) print(arr2) print(arr2.shape)  print(arr2.dtype) |
| [1 2 3 4 5]  (5,)  int32 |

|  |
| --- |
| *# array의 자료형을 확인할 수 있다.*  Data3 = [1,2,3,3.5,4] arr3 = np.array(data3) print(arr3) print(arr3.shape) print(arr3.dtype) |
| [1. 2. 3. 3.5 4. ]  (5,)  float64 |

|  |
| --- |
| arr4 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]]) print(arr4) print(arr4.shape) print(arr4.dtype) |
| [[ 1 2 3]  [ 4 5 6]  [ 7 8 9]  [10 11 12]]  (4, 3)  int32 |

**numpy shape**

numpy에서는 해당 array의 크기를 알 수 있다. shape 을 확인함으로써 몇개의 데이터가 있는지, 몇 차원으로 존재하는지 등을 확인할 수 있다. 위에서 arr1.shape의 결과는 (5,) 으로써, 1차원의 데이터이며 총 5라는 크기를 갖고 있음을 알 수 있다. arr4.shape의 결과는 (4,3) 으로써, 2차원의 데이터이며 4 \* 3 크기를 갖고 있는 array 이다.

**numpy 자료형**

arr1이나 arr2는 int64라는 자료형을 갖는 것에 반해 arr3는 float64라는 자료형을 갖는다. 이는 arr3내부 데이터를 살펴보면 3.5라는 실수형 데이터를 갖기 때문임을 알 수 있다. numpy에서 사용되는 자료형은 아래와 같다. (자료형 뒤에 붙는 숫자는 몇 비트 크기인지를 의미한다.)

* 부호가 있는 정수 int(8, 16, 32, 64)
* 부호가 없는 정수 uint(8 ,16, 32, 54)
* 실수 float(16, 32, 64, 128)
* 복소수 complex(64, 128, 256)
* 불리언 bool
* 문자열 string\_
* 파이썬 오프젝트 object
* 유니코드 unicode\_

#### 2-1. np.zeros(), np.ones(), np.arange() 함수

numpy에서 array를 정의할 때 사용되는 함수들이다.

아래 실습을 통해 각각이 어떻게 다른지 살펴본다.

|  |
| --- |
| array1 = np.zeros(10) print(array1)  array2 = np.zeros((3,5)) print(array2) |
| [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  [[0. 0. 0. 0. 0.]  [0. 0. 0. 0. 0.]  [0. 0. 0. 0. 0.]] |

|  |
| --- |
| *# np.zeros() 함수는 인자로 받는 크기만큼, 모든요소가 0인 array를 만든다.*  array3 = np.ones(9) print(array3) print(array3.shape)  array4 = np.ones((2,10)) print(array4) print(array4.shape) |
| [1. 1. 1. 1. 1. 1. 1. 1. 1.]  (9,)  [[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]  (2, 10)  *# np.ones() 함수는 인자로 받는 크기만큼, 모든요소가 1인 array를 만든다.* |

|  |
| --- |
| array5 = np.arange(10)  print(array5)  array6 = np.arange(3,10) print(array6) |
| [0 1 2 3 4 5 6 7 8 9]  [3 4 5 6 7 8 9]  *# np.arange() 함수는 인자로 받는 값 만큼 1씩 증가하는 1차원 array를 만든다.*  *# 이때 하나의 인자만 입력하면 0 ~ 입력한 인자, 값 만큼의 크기를 가진다.* |

### 3. Array 연산

기본적으로 numpy에서 연산을 할때는 크기가 서로 동일한 array 끼리 연산이 진행된다.

이때 같은 위치에 있는 요소들 끼리 연산이 진행된다.

arr1 = np.array([[1,2,3],[4,5,6]])

arr1

array([[1, 2, 3],

[4, 5, 6]])

arr1.shape

(2, 3)

arr2 = np.array([[10,11,12],[13,14,15]])

arr2

array([[10, 11, 12],

[13, 14, 15]])

arr2.shape

(2, 3)

#### 3-1. array 덧셈

arr1 + arr2

array([[11, 13, 15],

[17, 19, 21]])

#### 3-2. array 뺄셈

arr1 - arr2

array([[-9, -9, -9],

[-9, -9, -9]])

#### 3-3. array 곱셈

주의하자.

행렬의 곱처럼 곱셈이 진행되는 것이 아니라 각 요소별로 곱셈이 진행된다.

arr1 \* arr2

array([[10, 22, 36],

[52, 70, 90]])

#### 3-4. array 나눗셈

arr1 / arr2

array([[0.1 , 0.18181818, 0.25 ],

[0.30769231, 0.35714286, 0.4 ]])

#### 3-5. array의 브로드 캐스트

위에서는 array가 같은 크기를 가져야 서로 연산이 가능하다고 했지만,

numpy에서는 브로드캐스트라는 기능을 제공한다.

브로드캐스트란, 서로 크기가 다른 array가 연산이 가능하게끔 하는 것이다.

arr1

array([[1, 2, 3],

[4, 5, 6]])

arr1.shape

(2, 3)

arr3 = np.array([10,11,12])

arr3

array([10, 11, 12])

arr3.shape

(3,)

arr1 + arr3

array([[11, 13, 15],

[14, 16, 18]])

arr1 \* arr3

array([[10, 22, 36],

[40, 55, 72]])

위와 같이 서로 크기가 다른 arr1과 arr3의 연산이 가능하다.

연산결과를 살펴보면 arr3이 [10,11,12] 에서 [[10,11,12],[10,11,12]]로 확장되어 계산되었음을 확인할 수 있다.

동일한 방식으로 하나의 array에 스칼라 연산도 가능하다.

arr1 \* 10

array([[10, 20, 30],

[40, 50, 60]])

*# 요소에 대해 제곱처리*

arr1 \*\* 2

array([[ 1, 4, 9],

[16, 25, 36]])

### 4. Array 인덱싱

numpy에서 사용되는 인덱싱은 기본적으로 python 인덱싱과 동일하다.

이때, python에서와 같이 1번째로 시작하는 것이 아니라 0번째로 시작하는 것에 주의한다.

arr1 = np.arange(10)

arr1

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

*# 0번째 요소*

arr1[0]

0

*# 3번째 요소*

arr1[3]

3

*# 3번째 요소부터 8번째 요소*

arr1[3:9]

array([3, 4, 5, 6, 7, 8])

arr1[:]

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

물론 1차원이 아닌 그 이상의 차원에서도 인덱싱이 가능하다.

arr2 = np.array([[1,2,3,4],

[5,6,7,8],

[9,10,11,12]])

arr2

array([[ 1, 2, 3, 4],

[ 5, 6, 7, 8],

[ 9, 10, 11, 12]])

*# 2차원의 array에서 인덱싱을 하기 위해선 2개의 인자를 입력해야 합니다.*

arr2[0,0]

1

*# 2행의 모든 요소 꺼내기*

arr2[2,:]

array([ 9, 10, 11, 12])

*# 2행의 3번째 요소 꺼내기*

arr2[2,3]

12

*# 모든 열의 3번째 요소 꺼내기*

arr2[:,3]

array([ 4, 8, 12])

### 5. Array boolean 인덱싱(마스크)

위에서 이용한 다차원의 인덱싱을 응용하여 boolean 인덱싱을 할 수 있다.

해당 기능은 주로 마스크라고 이야기하는데, boolean인덱싱을 통해 만들어낸 array를 통해

우리가 원하는 행 또는 열의 값만 뽑아낼 수 있다.

즉, 마스크처럼 우리가 가리고 싶은 부분은 가리고, 원하는 요소만 꺼낼 수 있다.

names = np.array(['Beomwoo','Beomwoo','Kim','Joan','Lee','Beomwoo','Park','Beomwoo'])

names

array(['Beomwoo', 'Beomwoo', 'Kim', 'Joan', 'Lee', 'Beomwoo', 'Park',

'Beomwoo'], dtype='<U7')

names.shape

(8,)

*# 아래에서 사용되는 np.random.randn() 함수는 기대값이 0이고, 표준편차가 1인 가우시안 정규 분포를 따르는 난수를 발생시키는 함수이다.*

*# 이 외에도 0~1의 난수를 발생시키는 np.random.rand() 함수도 존재한다.*

data = np.random.randn(8,4)

data

array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],

[ 1.93097843, -1.8885167 , 1.99767454, 0.31524498],

[-0.22633642, -0.76385264, 0.16368804, 0.91204438],

[ 1.34321923, 1.54532121, 0.28814921, 0.50688776],

[ 0.4126606 , -0.52356522, 0.27124037, -0.66383264],

[ 0.88575452, -1.39205929, 0.91019739, -1.04676349],

[-0.05673648, -1.63408607, -2.29844338, -0.3662913 ],

[ 0.45963024, 0.35662128, 0.18307525, 1.46992167]])

data.shape

(8, 4)

위와 같은 names와 data라는 array가 있다.

이때, names의 각 요소가 data의 각 행과 연결된다고 가정해보자.

그리고 이 때, names가 Beomwoo인 행의 data만 보고 싶을 때 다음과 같이 마스크를 사용한다.

*# 요소가 Beomwoo인 항목에 대한 mask 생성*

names\_Beomwoo\_mask = (names == 'Beomwoo')

names\_Beomwoo\_mask

array([ True, True, False, False, False, True, False, True])

data[names\_Beomwoo\_mask,:]

array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],

[ 1.93097843, -1.8885167 , 1.99767454, 0.31524498],

[ 0.88575452, -1.39205929, 0.91019739, -1.04676349],

[ 0.45963024, 0.35662128, 0.18307525, 1.46992167]])

위의 결과를 보면, 요소가 Beomwoo인 것은 0번째, 1번째, 5번째, 7번째 이므로

data에서 0,1,5,7행의 모든 요소를 꺼내와야 한다.

이를 위해 요소가 Beomwoo인 것에 대한 boolean 값을 가지는 mask를 만들었고

마스크를 인덱싱에 응용하여 data의 0,1,5,7행을 꺼냈다.

*# 요소가 Kim인 행의 데이터만 꺼내기*

data[names == 'Kim',:]

array([[-0.22633642, -0.76385264, 0.16368804, 0.91204438]])

*# 논리 연산을 응용하여, 요소가 Kim 또는 Park인 행의 데이터만 꺼내기*

data[(names == 'Kim') | (names == 'Park'),:]

array([[-0.22633642, -0.76385264, 0.16368804, 0.91204438],

[-0.05673648, -1.63408607, -2.29844338, -0.3662913 ]])

물론 data array 자체적으로도 마스크를 만들고, 이를 응용하여 인덱싱이 가능하다.

data array에서 0번째 열의 값이 0보다 작은 행을 구해보자.

*# 먼저 마스크를 만든다.*

*# data array에서 0번째 열이 0보다 작은 요소의 boolean 값은 다음과 같다.*

data[:,0] < 0

array([ True, False, True, False, False, False, True, False])

*# 위에서 만든 마스크를 이용하여 0번째 열의 값이 0보다 작은 행을 구한다.*

data[data[:,0]<0,:]

array([[-1.07099572, -0.85382063, -1.42474621, -0.05992846],

[-0.22633642, -0.76385264, 0.16368804, 0.91204438],

[-0.05673648, -1.63408607, -2.29844338, -0.3662913 ]])

이를 통해 특정 위치에만 우리가 원하는 값을 대입할 수 있다.

위에서 얻은, 0번째 열의 값이 0보다 작은 행의 2,3번째 열값에 0을 대입해보자.

*# 0번째 열의 값이 0보다 작은 행의 2,3번째 열 값*

data[data[:,0]<0,2:4]

array([[-1.42474621, -0.05992846],

[ 0.16368804, 0.91204438],

[-2.29844338, -0.3662913 ]])

data[data[:,0]<0,2:4] = 0

data

array([[-1.07099572, -0.85382063, 0. , 0. ],

[ 1.93097843, -1.8885167 , 1.99767454, 0.31524498],

[-0.22633642, -0.76385264, 0. , 0. ],

[ 1.34321923, 1.54532121, 0.28814921, 0.50688776],

[ 0.4126606 , -0.52356522, 0.27124037, -0.66383264],

[ 0.88575452, -1.39205929, 0.91019739, -1.04676349],

[-0.05673648, -1.63408607, 0. , 0. ],

[ 0.45963024, 0.35662128, 0.18307525, 1.46992167]])

### 6. Numpy 함수

numpy에서는 array에 적용되는 다양한 함수가 있다.

#### 6-1. 하나의 array에 적용되는 함수

arr1 = np.random.randn(5,3)

arr1

array([[-1.28394941, -1.38235479, 0.3676742 ],

[ 0.91707237, 0.45364032, 0.00683315],

[ 0.51191795, 0.39014894, -0.15396686],

[ 0.75541648, -3.0457677 , 0.83785171],

[ 0.36609986, 1.2300834 , 0.51764117]])

*# 각 성분의 절대값 계산하기*

np.abs(arr1)

array([[1.28394941, 1.38235479, 0.3676742 ],

[0.91707237, 0.45364032, 0.00683315],

[0.51191795, 0.39014894, 0.15396686],

[0.75541648, 3.0457677 , 0.83785171],

[0.36609986, 1.2300834 , 0.51764117]])

*# 각 성분의 제곱근 계산하기 ( == array \*\* 0.5)*

np.sqrt(arr1)

/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/\_\_main\_\_.py:2: RuntimeWarning: invalid value encountered in sqrt

from ipykernel import kernelapp as app

array([[ nan, nan, 0.60636144],

[0.95763896, 0.67352826, 0.08266285],

[0.71548442, 0.62461903, nan],

[0.86914699, nan, 0.9153424 ],

[0.60506187, 1.10909125, 0.71947284]])

*# 각 성분의 제곱 계산하기*

np.square(arr1)

array([[1.64852609e+00, 1.91090476e+00, 1.35184315e-01],

[8.41021739e-01, 2.05789543e-01, 4.66919060e-05],

[2.62059989e-01, 1.52216194e-01, 2.37057951e-02],

[5.70654061e-01, 9.27670091e+00, 7.01995494e-01],

[1.34029109e-01, 1.51310516e+00, 2.67952378e-01]])

*# 각 성분을 무리수 e의 지수로 삼은 값을 계산하기*

np.exp(arr1)

array([[0.27694138, 0.25098684, 1.44437138],

[2.50195487, 1.57403175, 1.00685655],

[1.66848821, 1.47720079, 0.85730043],

[2.12849782, 0.04755979, 2.3113961 ],

[1.44209925, 3.42151487, 1.6780647 ]])

*# 각 성분을 자연로그, 상용로그, 밑이 2인 로그를 씌운 값을 계산하기*

np.log(arr1)

/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/\_\_main\_\_.py:2: RuntimeWarning: invalid value encountered in log

from ipykernel import kernelapp as app

array([[ nan, nan, -1.00055807],

[-0.08656889, -0.79045064, -4.98596986],

[-0.66959092, -0.94122672, nan],

[-0.28048605, nan, -0.17691415],

[-1.00484914, 0.20708197, -0.65847301]])

np.log10(arr1)

/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/\_\_main\_\_.py:1: RuntimeWarning: invalid value encountered in log10

if \_\_name\_\_ == '\_\_main\_\_':

array([[ nan, nan, -0.43453685],

[-0.03759639, -0.34328835, -2.1653792 ],

[-0.29079964, -0.40876957, nan],

[-0.12181354, nan, -0.07683284],

[-0.43640043, 0.08993456, -0.28597119]])

np.log2(arr1)

/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/\_\_main\_\_.py:1: RuntimeWarning: invalid value encountered in log2

if \_\_name\_\_ == '\_\_main\_\_':

array([[ nan, nan, -1.44350016],

[-0.1248925 , -1.14037921, -7.193234 ],

[-0.9660155 , -1.35790312, nan],

[-0.40465583, nan, -0.25523316],

[-1.44969086, 0.29875613, -0.94997574]])

*# 각 성분의 부호 계산하기(+인 경우 1, -인 경우 -1, 0인 경우 0)*

np.sign(arr1)

array([[-1., -1., 1.],

[ 1., 1., 1.],

[ 1., 1., -1.],

[ 1., -1., 1.],

[ 1., 1., 1.]])

*# 각 성분의 소수 첫 번째 자리에서 올림한 값을 계산하기*

np.ceil(arr1)

array([[-1., -1., 1.],

[ 1., 1., 1.],

[ 1., 1., -0.],

[ 1., -3., 1.],

[ 1., 2., 1.]])

*# 각 성분의 소수 첫 번째 자리에서 내림한 값을 계산하기*

np.floor(arr1)

array([[-2., -2., 0.],

[ 0., 0., 0.],

[ 0., 0., -1.],

[ 0., -4., 0.],

[ 0., 1., 0.]])

*# 각 성분이 NaN인 경우 True를, 아닌 경우 False를 반환하기*

np.isnan(arr1)

array([[False, False, False],

[False, False, False],

[False, False, False],

[False, False, False],

[False, False, False]])

np.isnan(np.log(arr1))

/Users/doorbw/anaconda3/envs/tensorflow/lib/python3.6/site-packages/ipykernel/\_\_main\_\_.py:1: RuntimeWarning: invalid value encountered in log

if \_\_name\_\_ == '\_\_main\_\_':

array([[ True, True, False],

[False, False, False],

[False, False, True],

[False, True, False],

[False, False, False]])

*# 각 성분이 무한대인 경우 True를, 아닌 경우 False를 반환하기*

arr1 = np.random.randn(5,3)

array = np.isinf(arr1)

print(array)

[[False False False]

[False False False]

[False False False]

[False False False]

[False False False]]

*# 각 성분에 대해 삼각함수 값을 계산하기(cos, cosh, sin, sinh, tan, tanh)*

np.cos(arr1)

array([[ 0.28292939, 0.18732825, 0.93316587],

[ 0.60814678, 0.89885772, 0.99997665],

[ 0.8718066 , 0.92485242, 0.9881705 ],

[ 0.72798607, -0.9954123 , 0.66906099],

[ 0.9337306 , 0.33415913, 0.86898882]])

np.tanh(arr1)

array([[-0.85753362, -0.88147747, 0.35195568],

[ 0.72450949, 0.42488675, 0.00683304],

[ 0.47143823, 0.37148862, -0.15276165],

[ 0.63836924, -0.99548634, 0.68466951],

[ 0.3505756 , 0.84260351, 0.47587741]])

#### 6-2. 두 개의 array에 적용되는 함수

arr1

array([[-1.28394941, -1.38235479, 0.3676742 ],

[ 0.91707237, 0.45364032, 0.00683315],

[ 0.51191795, 0.39014894, -0.15396686],

[ 0.75541648, -3.0457677 , 0.83785171],

[ 0.36609986, 1.2300834 , 0.51764117]])

arr2 = np.random.randn(5,3)

arr2

array([[-1.08072661, 0.49305711, -1.2341793 ],

[ 0.72539264, -0.17482108, -2.29144412],

[ 0.33676285, -0.44206124, 0.68359426],

[-1.3367298 , -0.62530265, 0.04842608],

[ 0.46567612, -2.53008109, 0.80771562]])

*# 두 개의 array에 대해 동일한 위치의 성분끼리 연산 값을 계산하기(add, subtract, multiply, divide)*

np.multiply(arr1,arr2)

array([[ 1.3875983 , -0.68157986, -0.45377588],

[ 0.66523755, -0.07930589, -0.01565778],

[ 0.17239495, -0.17246972, -0.10525086],

[-1.00978772, 1.90452663, 0.04057388],

[ 0.17048396, -3.11221075, 0.41810686]])

*# 두 개의 array에 대해 동일한 위치의 성분끼리 비교하여 최대값 또는 최소값 계산하기(maximum, minimum)*

np.maximum(arr1,arr2)

array([[-1.08072661, 0.49305711, 0.3676742 ],

[ 0.91707237, 0.45364032, 0.00683315],

[ 0.51191795, 0.39014894, 0.68359426],

[ 0.75541648, -0.62530265, 0.83785171],

[ 0.46567612, 1.2300834 , 0.80771562]])

#### 6-3. 통계 함수

통계 함수를 통해 array의 합이나 평균등을 구할 때,

추가로 axis라는 인자에 대한 값을 지정하여 열 또는 행의 합 또는 평균등을 구할 수 있다.

arr1

array([[-1.28394941, -1.38235479, 0.3676742 ],

[ 0.91707237, 0.45364032, 0.00683315],

[ 0.51191795, 0.39014894, -0.15396686],

[ 0.75541648, -3.0457677 , 0.83785171],

[ 0.36609986, 1.2300834 , 0.51764117]])

*# 전체 성분의 합을 계산*

np.sum(arr1)

0.4883407866652476

*# 열 간의 합을 계산*

np.sum(arr1, axis=1)

array([-2.29863 , 1.37754584, 0.74810003, -1.45249951, 2.11382443])

*# 행 간의 합을 계산*

np.sum(arr1, axis=0)

array([ 1.26655726, -2.35424983, 1.57603336])

*# 전체 성분의 평균을 계산*

np.mean(arr1)

0.032556052444349844

*# 행 간의 평균을 계산*

np.mean(arr1, axis=0)

array([ 0.25331145, -0.47084997, 0.31520667])

*# 전체 성분의 표준편차, 분산, 최소값, 최대값 계산(std, var, min, max)*

np.std(arr1)

1.0840662273348296

np.min(arr1, axis=1)

array([-1.38235479, 0.00683315, -0.15396686, -3.0457677 , 0.36609986])

*# 전체 성분의 최소값, 최대값이 위치한 인덱스를 반환(argmin, argmax)*

np.argmin(arr1)

10

np.argmax(arr1,axis=0)

array([1, 4, 3])

*# 맨 처음 성분부터 각 성분까지의 누적합 또는 누적곱을 계산(cumsum, cumprod)*

np.cumsum(arr1)

array([-1.28394941, -2.6663042 , -2.29863 , -1.38155763, -0.92791731,

-0.92108416, -0.40916621, -0.01901727, -0.17298413, 0.58243235,

-2.46333535, -1.62548364, -1.25938378, -0.02930038, 0.48834079])

np.cumsum(arr1,axis=1)

array([[-1.28394941, -2.6663042 , -2.29863 ],

[ 0.91707237, 1.3707127 , 1.37754584],

[ 0.51191795, 0.90206689, 0.74810003],

[ 0.75541648, -2.29035122, -1.45249951],

[ 0.36609986, 1.59618326, 2.11382443]])

np.cumprod(arr1)

array([-1.28394941e+00, 1.77487362e+00, 6.52575230e-01, 5.98458715e-01,

2.71485005e-01, 1.85509711e-03, 9.49657512e-04, 3.70507870e-04,

-5.70459347e-05, -4.30934393e-05, 1.31252606e-04, 1.09970221e-04,

4.02600826e-05, 4.95232591e-05, 2.56352776e-05])

#### 6-4. 기타 함수

arr1

array([[-1.28394941, -1.38235479, 0.3676742 ],

[ 0.91707237, 0.45364032, 0.00683315],

[ 0.51191795, 0.39014894, -0.15396686],

[ 0.75541648, -3.0457677 , 0.83785171],

[ 0.36609986, 1.2300834 , 0.51764117]])

*# 전체 성분에 대해서 오름차순으로 정렬*

np.sort(arr1)

array([[-1.38235479, -1.28394941, 0.3676742 ],

[ 0.00683315, 0.45364032, 0.91707237],

[-0.15396686, 0.39014894, 0.51191795],

[-3.0457677 , 0.75541648, 0.83785171],

[ 0.36609986, 0.51764117, 1.2300834 ]])

*# 전체 성분에 대해서 내림차순으로 정렬*

np.sort(arr1)[::-1]

array([[ 0.36609986, 0.51764117, 1.2300834 ],

[-3.0457677 , 0.75541648, 0.83785171],

[-0.15396686, 0.39014894, 0.51191795],

[ 0.00683315, 0.45364032, 0.91707237],

[-1.38235479, -1.28394941, 0.3676742 ]])

*# 행 방향으로 오름차순으로 정렬*

np.sort(arr1,axis=0)

array([[-1.28394941, -3.0457677 , -0.15396686],

[ 0.36609986, -1.38235479, 0.00683315],

[ 0.51191795, 0.39014894, 0.3676742 ],

[ 0.75541648, 0.45364032, 0.51764117],

[ 0.91707237, 1.2300834 , 0.83785171]])