Numpy

Update : 2021

Content

Content

- 균등분포 데이터 랜덤 추출하기
- 정규분포 데이터 랜덤 추출하기
- Numpy 배열의 Boolean Indexing
- 마스크 적용 후 true 값 갯수 구하기
- Q. 상위 5퍼센트에 해당하는 숫자 값을 추출해라
- 외부 파일 불러오기
- 외부 파일 저장하기

Numpy 개요

- Numeric + Python = Numpy
- 수학 및 과학 연산을 위한 파이썬 패키지
- 배열이나 행렬 계산에 용이한 메서드를 제공
- 한글로 넘파이로 주로 통칭, 넘피/늄파이라고 부르기도 함
- 관련 사이트 : http://www.numpy.org

Numpy 개요

- 일반 List에 비해 빠르고, 메모리 효율적
- 반복문 없이 데이터 배열에 대한 처리를 지원함
- 선형대수와 관련된 다양한 기능을 제공함
- Reference Site
  - cs231 : http://cs231n.github.io/python-numpy-tutorial/#numpy
  - https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
  - 데이터사이언스스쿨(파이썬버전) - https://goo.gl/3hsjbS

Numpy 설치 및 임포트

- Numpy 는 외부 모듈
- 터미널 창을 이용하여 아래와 같이 설치한다.

```
pip install numpy
```

- Numpy 임포트 : 별칭 np 이용

```
import numpy as np
```

일반적으로 numpy는 np라는 alias(별칭) 이용해서 호출함. 세계적인 약속 같은 것

- Numpy 버전 확인 : np.__version__

```
np.___version___
```

Numpy 설치 및 임포트

- Numpy 는 외부 모듈
- 터미널 창을 이용하여 아래와 같이 설치한다.

```
pip install numpy
```

Numpy 배열 만들기 : ndarray 개체로 생성

- 실수형 리스트 → Numpy 배열

```
import numpy as np
data = [6, 7.5, 8, 9, 1]
arr = np.array(data)
arr
```

배열이름 = np.array(리스트)

```
array([6. , 7.5, 8. , 9. , 1. ])
[6. , 7.5, 8. , 9. , 1. ]
```

- 문자형 리스트 → Numpy 배열

```
myList = ['강아지','원숭이','꽃’]
np.array(myList)
```

```
array(['강아지', '원숭이', '꽃'], dtype='<U3')
```

Numpy 배열 Type 확인하기

type(넘파이 배열)

```
import numpy as np
names = [1998, '마리아', 10.5]
arr = np.array(names)
type(arr)
```

numpy.ndarray

Numpy 배열 사이즈 확인하기

```
import numpy as np
data = [6, 7.5, 8, 9, 1]
arr = np.array(data)
arr.shape
```

(5,)

Numpy배열이름.shape

2차원 Numpy 배열 선언하기

np.array([1행 data...], [2행 데이터...]...)

- 각 행의 데이터 수가 같아야 한다.

```
import numpy as np
data2 = [ [1,2,3,4],
          [5,6,7,8]]
arr2 = np.array(data2)
arr2.shape
```

(2,4)

넘파이 배열의 차원 확인하기

넘파이배열.ndim

ndarray dimension. 정수로 표시

```
arr = np.array([1,2,3,4,5])
arr.ndim
```

1

```
arr2d = np.array([[1,2,3],[4,5,6]])
arr2d.ndim
```

2

넘파이 배열의 전체 크기 확인하기

넘파이배열.size

데이터 개수를 표시한다.

```
import numpy as np
data2 = [ [1,2,3,4],
          [5,6,7,8]]
arr2 = np.array(data2)
arr2.size
```

8

초기값을 0 또는 1로 지정하기

```
import numpy as np
np.zeros((3,6))
```

np.zeros((i,j)) : 2차원 i*j

np.ones((i,j)) : 2차원 i*j

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

```
import numpy as np
np.ones(10)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

초기값을 0 또는 1로 지정하기

```
import numpy as np
np.empty(10)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

np. empty(n)
: 초기값을 지정하지 않고 배열 생성

np.empty((i,j)) : 2차원 i*j

```
import numpy as np
np.empty((2,2))
```

```
array([[0., 0.],
       [0., 0.]])
```

초기값을 1씩 증가되는 숫자값으로 지정하기

np.arrange(j)

np.arrange(i,j)

np.arrange(i,j,step)

```
import numpy as np
np.arange(15)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
import numpy as np
np.arange(5,10)
```

```
array([5, 6, 7, 8, 9])
```

초기값을 1씩 증가되는 숫자값으로 지정하기

np.arrange(j)

np.arrange(i,j)

np.arrange(i,j,step)

```
import numpy as np
np.arange(0,5,0.5)
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

```
import numpy as np
np.arange(30).reshape(5,6)
```

1~30 까지 숫자를 5행6열로 생성

```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29]])
```

Numpy 배열의 자료형

```
import numpy as np
data = [6, 7.5, 8, 9, 1]
arr = np.array(data)
arr.dtype
```

Numpy배열명.dtype

dtype('float64')

```
import numpy as np
arr = np.arange(5,10)
arr.dtype
```

dtype('int32')

Numpy 배열의 초기값으로 자료형 지정하기

배열명 = np.array(리스트, dtype=자료형)

자료형 = float, int, '<U2'

```
import numpy as np
arr = np.array([1,2,3,4,5], dtype=float)
arr
arr.dtype
```

```
array([1., 2., 3., 4., 5.])
dtype('float64')
```

Numpy 배열의 초기값으로 자료형 지정하기

배열명 = np.array(리스트, dtype=자료형)

자료형 = float, int, '<U2'

```
import numpy as np
arr_i = np.array([.7, .5, 1.5, 2], dtype=int)
arr_i
arr_i.dtype
```

```
array([0, 0, 1, 2])
dtype('int')
```

Numpy 배열의 초기값으로 자료형 지정하기

배열명 = np.array(리스트, dtype=자료형)

자료형 = float, int, '<U2'

```
import numpy as np
np.array([1,2,3,4,5], dtype='<U2')
```

```
array(['1', '2', '3', '4', '5'],
dtype='<U2')
```

Numpy 배열의 데이터형 변환

배열명2 = 배열명1.astype(자료형)

자료형 = np.float64, np.int32

```
import numpy as np
arr=np.array([1.5 , 3.4 , 7.8])
arr
```

```
array([1.5, 3.4, 7.8])
```

```
arrInt = arr.astype(np.int64)
arrInt.dtype
```

```
dtype('int64')
```

Numpy 배열의 데이터형 변환

배열명2 = 배열명1.astype(자료형)

자료형 = np.float63, np.int32, np.str, '<U11'

```
import numpy as np
test_array = np.array([1.,3.,5.6,7.], np.str,)
test_array
```

```
array(['1.0', '3.0', '5.6', '7.0'], dtype='<U3')
```

Numpy 배열의 Reshape

배열명2 = 배열명1.reshape(행,열)

```
import numpy as np
test_matrix = [[1,2,3,4],[1,2,5,8]]
test_matrix_arr = np.array(test_matrix)
test_matrix_arr
```

```
array([[1, 2, 3, 4],
       [1, 2, 5, 8]])
```

```
test_matrix_arr.shape
```

```
(2, 4)
```

```
test_matrix_arr.reshape(4,2)
```

```
array([[1, 2],
       [3, 4],
       [1, 2],
       [5, 8]])
```

flatten()으로 1차원 배열로 변경하기

배열명.flatten()

```
import numpy as np
arr2d = np.array([[1,4,5,6],[56,23,45,67],[8,4,6,10]])
arr2d.flatten()
```

```
array([ 1,  4,  5,  6, 56, 23, 45, 67,  8,  4,  6, 10])
```

Numpy 배열의 행열 연산

같은 위치의 요소끼리 사칙 연산

```
import numpy as np
arrOne = np.array([[1,2,3], [4,5,6]], dtype=np.float64)
arrTwo = np.array([[7,8,9], [10,11,12]], dtype=np.float64)
arrOne
arrTwo
```

```
array([[1., 2., 3.], [4., 5., 6.]])
array([[ 7.,  8.,  9.],  [10., 11., 12.]])
```

```
arrOne + arrTwo
```

```
array([[ 8., 10., 12.],
       [14., 16., 18.]])
```

Numpy 배열의 행열 연산

같은 위치의 요소끼리 사칙 연산

```
arrOne - arrTwo
```

```
array([[-6., -6., -6.],
       [-6., -6., -6.]])
```

```
arrOne * arrTwo
```

```
array([[ 7., 16., 27.],
       [40., 55., 72.]])
```

```
arrOne / arrTwo
```

```
array([[0.14285714, 0.25      , 0.33333333],
       [0.4       , 0.45454545, 0.5       ]])
```

Numpy 배열의 행열 연산

특정 숫자를 이용한 행열 연산

```
1 / arrOne
```

```
array([[1.        , 0.5       , 0.33333333],
       [0.25      , 0.2       , 0.16666667]])
```

```
arrOne * 2
```

```
array([[ 2.,  4.,  6.],
       [ 8., 10., 12.]])
```

```
arrOne ** 2
```

```
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
```

수학적인 행렬의 곱 (DOT product)

- 배열1.dot(배열2)

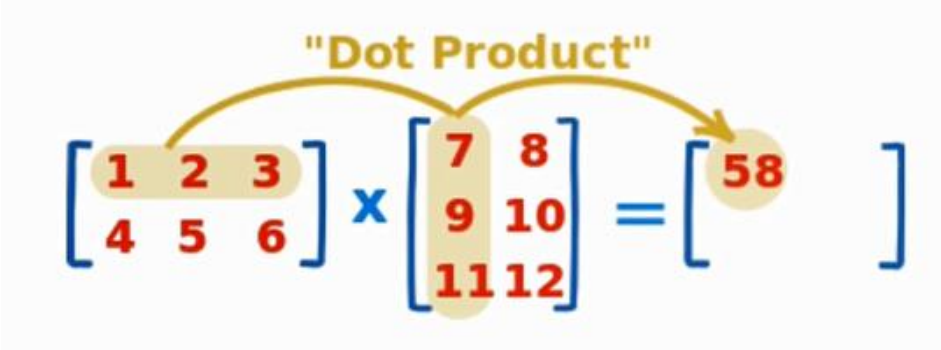
```
a = np.arange(1,7).reshape(2,3)
b = np.arange(7,13).reshape(3,2)
a
b
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
array([[ 7,  8],
       [ 9, 10],
       [11, 12]])
```

```
a.dot(b)
```

```
array([[ 58,  64],
       [139, 154]])
```



Numpy 배열의 인덱싱

리스트 인덱싱 기법과 동일

```
arr = np.arange(10)
arr
```

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

```
arr[5]
```

5

```
arr[5:8]
```

array([5, 6, 7])

Numpy 배열의 인덱싱

리스트 인덱싱 기법과 동일

```
arr = np.arange(10)
arr
```

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

```
arr[:]
```

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

```
arr[:5]
```

array([0, 1, 2, 3, 4])

```
arr[5:]
```

array([5, 6, 7, 8, 9])

Numpy 배열의 인덱싱 - 값 교체하기

리스트 인덱싱 기법과 동일

```
arr = np.arange(10)
arr
```

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

```
arr[0] = 100
arr
```

array([100, 1, 2, 3, 4, 5, 6, 7, 8, 9])

```
arr[5:8]=12
arr
```

array([100, 1, 2, 3, 4, 12, 12, 12, 8, 9])

Numpy 배열의 인덱싱 - 값 교체하기

리스트 인덱싱 기법과 동일

```
arr[7:] = 0
arr
```

```
array([100,   1,   2,   3,   4,  12,  12,   0,   0,   0])
```

Numpy 배열의 인덱싱 - 2차원

배열이름[행,열]

```
arr2d = np.array([ [1,2,3,4],
                   [5,6,7,8],
                   [9,10,11,12],
                   [13,14,15,16]])
arr2d
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

```
arr2d.shape
```

```
(4,4)
```

Numpy 배열의 인덱싱 - 2차원

배열이름[행:열]

```
arr2d[2,:]
```

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

인덱스가 2일 행 데이터 출력

```
arr2d[1:3, :]
```

```
array([[ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

인덱스가 1,2인 행 데이터 출력

```
arr2d[:, 3]
```

```
array([ 4,  8, 12, 16])
```

인덱스가 3인 열 데이터 출력. 4열 출력

Numpy 배열의 인덱싱 - 2차원

배열이름[행:열]

```
arr2d[:, :2]
```

```
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]])
```

인덱스가 0,1 열 인 데이터 출력

```
arr2d[2,2]
```

```
11
```

```
arr2d[3,2]
```

```
15
```

Numpy 배열의 인덱싱 - 2차원 : 특정값으로 세팅하기

배열이름[행:열]

```
arr2d = np.array([ [1,2,3,4],
                   [5,6,7,8],
                   [9,10,11,12],
                   [13,14,15,16]])
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

```
arr2d[:2, 1:3] = 0
arr2d
```

```
array([[ 1,  0,  0,  4],
       [ 5,  0,  0,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

1,2 행에서
2,3열 0으로 지정

:: 를 이용한 Numpy 배열 인덱싱

배열이름[start:end;step, start:end;step]

```
arr1 = np.array([1,2,3,4,5,6,7,8,9,10])
arr1[::2]
```

```
array([1, 3, 5, 7, 9])
```

```
arr2 = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
arr2
arr2[:,::2]
arr2[::2,::]
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

```
array([[ 1,  3,  5],
       [ 6,  8, 10],
       [11, 13, 15]])
```

```
array([[ 1,  2,  3,  4,  5],
       [11, 12, 13, 14, 15]])
```

Numpy 배열의 난수값 지정하기

np.random.randn(행수,열수)

```
arr_random = np.random.randn(5)
arr_random
```

```
array([ 0.59390464,  1.35837555, -0.50661085, -0.62270521,  0.5398455 ])
```

```
arr_random_2rd = np.random.randn(2,2)
arr_random_2rd
```

```
array([[-0.90797725, -0.44135803],
       [ 0.60295452,  2.11935903]])
```

Numpy 배열의 함수 이용하기

np.sqrt(배열명) : 제곱근 구하기

```
arr = np.arange(1,10)
arr
```

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

```
np.sqrt(arr)
```

```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.        ])
```

Numpy 배열의 함수 이용하기

np.log10(배열명) : 로그 구하기

```
arr = np.arange(1,10)
arr
```

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

```
np.log10(arr)
```

```
array([0.        , 0.30103   , 0.47712125, 0.60205999, 0.69897   ,
       0.77815125, 0.84509804, 0.90308999, 0.95424251])
```

Numpy 배열의 함수 이용하기

두개의 배열에서 큰 숫자만 추출하기

np.maximum(배열1, 배열2)

```
x = np.random.randn(8)
y = np.random.randn(8)
```

```
x
```

```
array([-0.62798839, -1.52600327,  1.0060349 ,  0.15035899,  1.88750083,
       -1.40882704,  1.22454134, -1.71651044])
```

```
y
```

```
array([-1.19605599, -0.39659137,  0.69618244,  0.28508404,  1.5963126 ,
       -0.00169748, -0.53328296,  0.58049113])
```

Numpy 배열의 함수 이용하기

두개의 배열에서 큰 숫자만 추출하기

np.maximum(배열1, 배열2)

```
np.maximum(x,y)
```

```
array([-6.27988388e-01, -3.96591375e-01,  1.00603490e+00,  2.85084035e-01,
        1.88750083e+00, -1.69748201e-03,  1.22454134e+00,  5.80491130e-01])
```

Numpy 배열의 함수 이용하기

배열 원소 전체의 합과 평균 구하기

배열이름.sum()
배열이름.mean()

```
arr = np.random.randn(5,4)
arr
```

```
array([[-0.54829013, -0.47042474, -0.0322164 , -0.40253898],
       [-0.30231623,  0.44002564,  0.46200343, -0.04059315],
       [-1.0678957 , -0.52062628, -1.20915093, -1.34887165],
       [-1.18757631, -1.48012777,  0.42301795, -1.96551195],
       [ 0.24021846,  0.90684117, -0.6183727 ,  0.02518732]])
```

Numpy 배열의 함수 이용하기

배열 원소 전체의 합과 평균 구하기

배열이름.sum()
배열이름.mean()

```
arr.sum()
```

-8.697218970013097

```
arr.mean()
```

-0.4348609485006548

Numpy 배열의 함수 이용하기

각 열과 행 단위로 합 구하기
배열이름.sum(axis=0)
배열이름.sum(axis=1)

```
arr = np.random.randn(5,4)
arr
```

```
array([[-0.54829013, -0.47042474, -0.0322164 , -0.40253898],
       [-0.30231623,  0.44002564,  0.46200343, -0.04059315],
       [-1.0678957 , -0.52062628, -1.20915093, -1.34887165],
       [-1.18757631, -1.48012777,  0.42301795, -1.96551195],
       [ 0.24021846,  0.90684117, -0.6183727 ,  0.02518732]])
```

Numpy 배열의 함수 이용하기

각 열과 행 단위로 합 구하기
배열이름.sum(axix=0)
배열이름.sum(axix=1)

```
arr.sum(axis=0)
```

```
array([-2.86585991, -1.12431198, -0.97471865, -3.73232842])
```

```
arr.sum(axis=1)
```

```
array([-1.45347026,  0.55911968, -4.14654456, -4.21019808,  0.55387424])
```

Numpy 배열의 함수 이용하기

배열 합치기
np.vstack((배열명1,배열명2)) : 행으로 합치기
np.hstack((배열명1,배열명2)) : : 열로 합치기. 2차원으로 지정해야한다

```
a = np.array([1,2,3])
b = np.array([4,5,6])
np.vstack((a,b))
np.hstack((a,b))
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
array([1, 2, 3, 4, 5, 6])
```

```
a = np.array([[1],[2],[3]])
b = np.array([[4],[5],[6]])
np.hstack((a,b))
```

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Numpy 배열의 함수 이용하기

배열 합치기
np.vstack((배열명1,배열명2)) : 행으로 합치기
np.hstack((배열명1,배열명2)) : : 열로 합치기. 2차원으로 지정해야한다

```
a = np.array([1,2,3])
b = np.array([4,5,6])
np.vstack((a,b))
np.hstack((a,b))
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
array([1, 2, 3, 4, 5, 6])
```

```
a = np.array([[1],[2],[3]])
b = np.array([[4],[5],[6]])
np.hstack((a,b))
```

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Numpy 배열의 함수 이용하기

배열 합치기 : concatenate()
- 가로, 세로 방향으로 넘파이 배열 합치기
- np.concatenate((배열1, 배열2), axis=0/1)
- axis = 0 : 세로로 합치기
- axis = 1 : 가로로 합치기

```
a = np.array([[1,2,3]])
b = np.array([[4,5,6]])
np.concatenate( (a,b), axis=0)
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
np.concatenate( (a,b), axis=1)
```

```
array([1, 2, 3, 4, 5, 6])
```

Numpy 배열의 함수 이용하기

배열 합치기 : concatenate()
- 가로, 세로 방향으로 넘파이 배열 합치기
- np.concatenate((배열1, 배열2), axis=0/1)
- axis = 0 : 세로로 합치기
- axis = 1 : 가로로 합치기

```
a = np.array([[1,2],[3,4]])
b = np.array([[5,6]])
a
b
b.T
```

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

```
array([[5],
       [6]])
```

```
array([[5, 6]])
```

```
np.concatenate((a,b.T), axis=1)
```

```
array([[1, 2, 5],
       [3, 4, 6]])
```

Numpy 배열의 함수 이용하기

정렬과 관련된 함수 이용하기

np.sort(배열이름)
np.sort(배열이름)[::-1]

```
arr = np.random.randn(8)
arr
```

```
array([-0.15826735,  0.11337688,  0.47224796,  0.36067284, -1.44941238,
        0.78604113, -0.39059043,  0.75602804])
```

```
np.sort(arr)
```

```
array([-1.44941238, -0.39059043, -0.15826735,  0.11337688,  0.36067284,
        0.47224796,  0.75602804,  0.78604113])
```

```
np.sort(arr)[::-1]
```

```
array([ 0.78604113,  0.75602804,  0.47224796,  0.36067284,  0.11337688,
       -0.15826735, -0.39059043, -1.44941238])
```

Numpy 배열의 함수 이용하기

정렬과 관련된 함수 이용하기 – 2차원

np.sort(배열이름, axis=0) – 열방향으로 정렬하기
np.sort(배열이름, axis=1) – 행방향으로 정렬하기

```
arr = np.random.randn(5,3)
arr
```

```
array([[ 1.59759886,  0.89651006,  1.16279832],
       [ 1.39558679,  1.46696994,  0.16154362],
       [ 0.45637333,  0.99947484,  1.73103623],
       [-0.80138537,  0.35605303, -0.04612007],
       [ 1.04954882,  0.43428921, -0.62481861]])
```

Numpy 배열의 함수 이용하기

정렬과 관련된 함수 이용하기 – 2차원

np.sort(배열이름, axis=0) – 열방향으로 정렬하기
np.sort(배열이름, axis=1) – 행방향으로 정렬하기

```
np.sort(arr, axis=0)
```

```
array([[-0.80138537,  0.35605303, -0.62481861],
       [ 0.45637333,  0.43428921, -0.04612007],
       [ 1.04954882,  0.89651006,  0.16154362],
       [ 1.39558679,  0.99947484,  1.16279832],
       [ 1.59759886,  1.46696994,  1.73103623]])
```

Numpy 배열의 함수 이용하기

정렬과 관련된 함수 이용하기 – 2차원

np.sort(배열이름, axis=0) – 열방향으로 정렬하기
np.sort(배열이름, axis=1) – 행방향으로 정렬하기

```
np.sort(arr, axis=1)
```

```
array([[ 0.89651006,  1.16279832,  1.59759886],
       [ 0.16154362,  1.39558679,  1.46696994],
       [ 0.45637333,  0.99947484,  1.73103623],
       [-0.80138537, -0.04612007,  0.35605303],
       [-0.62481861,  0.43428921,  1.04954882]])
```

Numpy 배열의 함수 이용하기

중복값 제거하기

np.unique(배열명)

```
names = np.array(['Charles','Julia', 'Hayoung','Charles',
                  'Hayoung','Julia','Julia'])
ints = np.array([3,3,3,2,2,1,1,4,4])
```

```
np.unique(names)
```

```
array(['Charles', 'Hayoung', 'Julia'], dtype='<U7')
```

```
np.unique(ints)
```

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

단위행렬(Identity)

np.identity(대각선숫자)
np.identity(n=대각선숫자, dtype=데이타형)

```
np.identity(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
np.identity(n=4, dtype=int)
```

```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

대각선이 1인 행렬 만들기

- np.eye(N=1의숫자, M=열수, dtype=데이터형) : 대문자 주의
- np.eye(1의숫자)
- np.eye(1의숫자, 열수, k=startIndex)

```
np.eye(N=3,M=4,dtype=int)
```

```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0]])
```

```
np.eye(5)
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
np.eye(4,7,k=2)
```

```
array([[0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.]])
```

대각행렬의 값을 추출함 : np.diag()

```
np.diag(배열이름)
np.diag(배열이름, k=startIndex)
```

```
matrix = np.arange(9).reshape(3,3)
matrix
```

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

```
np.diag(matrix)
```

```
array([0, 4, 8])
```

```
np.diag(matrix, k=1)
```

```
array([[0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.]])
```

균등분포 데이터 랜덤 추출하기

np.random.uniform(0,1,데이터수)

np.random.uniform(0,1,10)

```
array([0.35378435, 0.65861067, 0.53928702, 0.45567964, 0.57641595,
       0.11758972, 0.60750528, 0.33911796, 0.38538083, 0.60604352])
```

np.random.uniform(0,1,10).reshape(2,5)

```
array([[0.85231164, 0.52047935, 0.58339012, 0.48804021, 0.89076047],
       [0.03187728, 0.1303267 , 0.94740567, 0.60492528, 0.26449567]])
```

정규분포 데이터 랜덤 추출하기

np.random. normal(0,1,데이터수)

```
np.random.normal(0,1,10).reshape(5,2)
```

```
array([[ 0.87583619,  0.46716933],
       [ 0.05300641, -1.0637336 ],
       [ 0.26724267, -0.20972726],
       [-0.7666967 ,  1.20439811],
       [ 1.24212901, -0.85216263]])
```

Numpy 배열의 Boolean Indexing, Mask

다수개의 배열에서 특정 값을 추출할 때 사용한다.

```
data = np.random.randn(7,4)
data
```

```
array([[ 1.05836724, -0.05063985, -1.00077849,  0.13438379],
       [-0.83632777, -0.20234479,  0.93418906,  0.06174163],
       [-1.39750668,  0.49026975, -1.9635103 , -0.32006973],
       [ 0.49610261,  2.10659512,  0.00493886, -0.05564645],
       [-0.80506692,  0.25654448,  0.90196181,  1.07732046],
       [ 2.07382619,  1.22637296, -0.69395325, -0.11657227],
       [ 0.02712782,  1.13544876, -1.1857645 ,  0.10523148]])
```

```
names = np.array(['Charles','Jhon', 'Hayoung','Charles',
                  'Hayoung','Jhon','Elise'])
names
```

```
array(['Charles', 'Jhon', 'Hayoung', 'Charles', 'Hayoung', 'Jhon',
       'Elise'], dtype='<U7')
```

Numpy 배열의 Boolean Indexing, Mask

다수개의 배열에서 특정 값을 추출할 때 사용한다.

배열이름[조건]

```
names == 'Charles'
```

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

```
data[names == 'Charles']
```

array([[1.05836724, -0.05063985, -1.00077849, 0.13438379],
       [0.49610261, 2.10659512, 0.00493886, -0.05564645]])

True 값이 적용된 1, 4행 만 추출

Numpy 배열의 Boolean Indexing, Mask

다수개의 배열에서 특정 값을 추출할 때 사용한다.

배열이름[조건]

```
names == 'Jhon'
```

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

```
data[names == ‘Jhon']
```

array([[1.43925418, -0.53976641, 0.81342517, 1.33233496],
       [0.42590361, 1.70670429, -0.24851528, -1.40718732]])

True 값이 적용된 2, 6행 만 추출

마스크 적용 후 true 값 갯수 구하기

마스크 적용 후 true 값 갯수 구하기

(조건).sum()

```
arr > 0
```

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

```
(arr>0).sum()
```

Q. 상위 5퍼센트에 해당하는 숫자 값을 추출해라

1. 전체 길이를 구한 후 상위 5프로 , 0.05를 곱한 후 정수형으로 변환.

2. 상위 5프로에 해당하는 인덱스 추출

```
large_arr = np.random.randn(50)
large_arr
```

```
array([-0.10543917, -0.14112602,  0.62971083,  0.94422854,  1.25565393,
        0.19716041, -1.30401035,  0.07106028, -0.96213509,  0.88655724,
       -0.04205286, -1.48322018,  1.04704071,  2.3304161 , -0.10710946,
       -0.67649761,  1.35238516, -1.26541795,  0.8402277 ,  1.80190845,
        0.38753024,  1.08541049,  0.54344364, -0.34538154, -0.220738  ,
       -1.45707222,  2.26069601, -1.55366309, -0.14171545, -0.07276405,
        0.14612277, -1.72742487,  0.86415653, -0.22132677,  0.35873793,
        2.39206824, -1.95599484, -0.20574049,  0.36969793,  0.57592836,
        1.98580701, -2.07285264,  0.38649635,  1.6559648 ,  0.73782756,
       -0.85669507, -0.30574773, -0.21633621, -0.69436293, -0.37934845])
```

Q. 상위 5퍼센트에 해당하는 숫자 값을 추출해라

상위 5퍼센트에 해당하는 숫자 값을 추출해라

1. 전체 길이를 구한 후 상위 5프로 , 0.05를 곱한 후 정수형으로 변환.
2. 상위 5프로에 해당하는 인덱스 추출

```
np.sort(large_arr)[::-1]
```

내림차순으로 소팅

```
array([ 2.39206824,  2.3304161 ,  2.26069601,  1.98580701,  1.80190845,
        1.6559648 ,  1.35238516,  1.25565393,  1.08541049,  1.04704071,
        0.94422854,  0.88655724,  0.86415653,  0.8402277 ,  0.73782756,
        0.62971083,  0.57592836,  0.54344364,  0.38753024,  0.38649635,
        0.36969793,  0.35873793,  0.19716041,  0.14612277,  0.07106028,
       -0.04205286, -0.07276405, -0.10543917, -0.10710946, -0.14112602,
       -0.14171545, -0.20574049, -0.21633621, -0.220738  , -0.22132677,
       -0.30574773, -0.34538154, -0.37934845, -0.67649761, -0.69436293,
       -0.85669507, -0.96213509, -1.26541795, -1.30401035, -1.45707222,
       -1.48322018, -1.55366309, -1.72742487, -1.95599484, -2.07285264])
```

Q. 상위 5퍼센트에 해당하는 숫자 값을 추출해라

상위 5퍼센트에 해당하는 숫자 값을 추출해라

1. 전체 길이를 구한 후 상위 5프로 , 0.05를 곱한 후 정수형으로 변환.
2. 상위 5프로에 해당하는 인덱스 추출

```
# 전체 길이 구하기
len(large_arr)
```

 # 전체 길이를 구한 후 상위 5프로 , 0.05를 곱한 후 정수형으로 변환

```
n = int(0.05*len(large_arr))
# 상위 5프로에 해당하는 인덱스 추출
np.sort(large_arr)[::-1][0:n]
```

array([2.6832301 , 2.01383475])

외부 파일 불러오기

```
배열명 = np.loadtxt('파일경로', delimeter='구분자', dtype=데이터형)
```

```
data = np.loadtxt('data/ratings.dat', delimiter='::', dtype=np.int64)
data[:5, :]
```

```
array([[       1,     1193,        5, 978300760],
       [       1,      661,        3, 978302109],
       [       1,      914,        3, 978301968],
       [       1,     3408,        4, 978300275],
       [       1,     2355,        5, 978824291]], dtype=int64)
```

```
data.shape
```

```
(1000209, 4)
```

외부파일 불러오기

외부 파일 불러오기

배열명 = np.loadtxt('파일경로', delimeter='구분자', dtype=데이터형)

```
mean_rating_total = data[:,2].mean()
mean_rating_total
```

3.581564453029317

전체 평균 평점 구하기

외부파일 저장하기

외부 파일 저장하기

np.savetxt('파일경로', 배열이름, fmt='포맷형식', delimeter='구분자')

```
data = np.loadtxt('data/ratings.dat', delimiter='::', dtype=np.int64)
data[:10, :]
np.savetxt("data.csv", data, fmt='%.3f', delimiter=",")
```