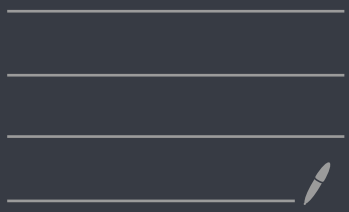

Numpy

수학, 행렬 등을 위한 파이썬 패키지

array

array는 Numpy의 가장 기본 구조이다.
array는 벡터이다.
array는 콤마가 없다.

생성

1. list를 이용한 생성

ex1 = np.array([1,2,3,4]) → 1차원 array 생성
ex2 = np.array([1,2,3,4],[5,6,7,8]) → 2차원 array 생성

⚠ 1차원 array는 [ ] 단위 벡터로 생성된다.

ex1.shape → (4,)
ex2.shape → (2,4)

2. 함수를 이용한 생성

- np.zeros(x) : 길이가 x이고 값이 0 (double)인 array 생성
  ↳ s 주의!!

ex) np.zeros(5) → $\begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}$ (5,)

np.zeros(5, dtype = np.int64) → int형 0으로 생성

- np.ones(x) : np.zeros(x)와 동일 기능, 0을 1로 바꾸기만 함.
  ↳ s주의!!

- np.empty(x) : 길이가 x짜리인 초기화 안된 array 생성
  ⇒ 이전 메모리의 영향을 받는다.

- np.arange(x [,y] [,z]) : range 함수처럼 동작하여 array 생성
  - np.arange(x) : 0 ~ x-1
  - np.arange(x,y) : x ~ y-1
  - np.arange(x,y,z) : x ~ y-1, z 단위로

- np.linspace(x,y,num) : x부터 y까지 구간을 num개 포인트로 나눈다.
  ↳ default: 50 이다

ex) np.linspace(0,9,10) ⇒

```
[0.
 1.
 2.
 3.
 4.
 5.
 6.
 7.
 8.
 9.]
```

9 포함 ★
10개 element

3. random number 생성

np.random.rand(x) : uniform distribution (연속확률분포)에서 랜덤으로 x개 추출

list vs array

list와 array의 차이점

1. 원소의 데이터
   - list : 숫자 문자, list, tuple 등 모든 자료형 (혼합 가능) array로라 넣을수 있다.
   - array : 한가지 자료형으로 통일 ★
2. 목적
   - list : 데이터의 저장
   - array : 연산 (데이터가 숫자인 경우 메모리 사용 적음)
3. 연산 적용 방식
   - list : + 연산 ⇒ list 합치기
   - array : + 연산 ⇒ 원소간 덧셈

vectorization

⇒ 명시적인 반복없이 array로 연산

x = list(range(1, 101))

ex)
```
y = []
for i in range(0, 100):
    y = y + [x[i]*2]

print(y)
```
→ list for문

```
x = np.array(range(1, 101))
y = 2*x

print(y)
```
→ array vectorization

dtype method

⇒ array 원소의 자료형 반환

ex) np.array([1, 2, 3, "a"]).dtype ⇒ <U11 , 유니코드

np.array([1.1, 2.2, 3.3]).dtype ⇒ float64

indexing & slicing

array도 indexing & slicing이 가능하다.

arr = np.array([1, 2, 3])

arr[1] ⇒ [2]

arr[0:2] ⇒ [1 2]

★

boolean indexing

```
arr = np.arange(10)          → [0 1 2 3 4 5 6 7 8 9]

print(arr >= 5)              → [False False .... True True]
```

원소 각각에 연산 적용

```
print(arr[arr >= 5])         → [5 6 7 8 9]
```

↳ 인덱싱

```
# my_list = [0,1,2,3,4,5,6,7,8,9]
# print(my_list >= 5)        → Error!
```

ex) `X = np.array([10,25,67,45,80])`

```
Print(len(x[x<=60]))
print(sum(x<=60))
```

이 코드가 더 효율적이다.

array operation

```
my_list1 = [1,2,3]                  my_list2 = [4,5,6]
my_array1 = np.array([1,2,3])       my_array2 = np.array([4,5,6])
```

+

리스트간 덧셈만 가능★

list : `my_list1 + my_list2` # [1,2,3,4,5,6]

↳ 이어 붙이기

array : `my_array1 + my_array2` # [5 7 9]

↳ 원소간 덧셈

*

list : `my_list1 * 2` # [1,2,3,1,2,3]

↳ 반복 (리스트간 곱셈 안됨)

array : `my_array1 * my_array2` # [4 10 18]

↳ 원소간 곱셈

-, /

list : 안됨
array : array간 빼기, 나누기

Broadcasting

→ array와 상수의 연산

my_array1 + 1 → [2 3 4]
my_array − 1 → [0 1 2]
my_array * 2 → [2 4 6]
my_array / 2 → [0.5 1 1.5]

⚠ list는 sum을 function으로만 사용하지만, array는 method도 가능

ex) sum(my_list) sum(my_array) my_array.sum()

array를 위한 Numpy function

function	input	output
np.random.rand(x)	랜덤 생성할 수 x	0~1인 값 x개 랜덤 생성하여 ndarray 객체로 반환
np.min(x)	array x	x 원소 중 최대/최소 값 반환
np.max(x)	array x	
np.mean(x)	array x	x 원소의 평균값 반환
np.var(x, ddof=0)	array x, 자유도	x 원소의 분산을 자유도 ddof로 계산하여 반환 ⚠ n개 원소를 관측했을 때, 평균을 알고 있다 하자. n-1개를 알면 나머지 1개 값을 알 수 있다. → 자유도 n-1 ddof의 default=0, 표본분산일 경우 1준다.
np.median(x)	array x	x 원소의 중위값 반환
np.quantile(x, q)	array x, 확률 q	x 원소 중 $Pr(x \le y) = q$를 만족하는 원소 y를 반환한다. ∴ q=0.25 → 1사분위수

array의 메소드

function	input	output
x.round(num)	표현할 소수점 아래까지 num	array x의 원소들을 소수점 아래 num까지 반올림
np.sort(x)	array x	x를 정렬 (오름차순)
np.unique(x [,return_index] [,return_counts])	array x [,return_index] [,return_counts]	x의 원소 중 unique한 값을 모두 반환 return_index가 True이면 unique한 값들의 첫번째 index도 반환 (default : False) return_counts가 True이면 unique한 값들의 count 정보도 반환 (default : False)
np.flip(x)	array x	x를 뒤집음
np.size(x)	array x	x의 크기 (원소 수 반환)
np.shape(x)	array x	x의 차원 요소 수 (행렬 n by m) 반환
np.concatenate(x,y)	array x, y	x와 y를 이어붙여서 반환

ex) uni_arr, count_arr = np.unique(x, return_counts = True)

[1. 2. 3. ... 10.] [11 11 13 ... 8]

ex) 표본분산 구하기

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

sum ((x - np.mean(x)) **2 /(len(x) - 1)

또는 np.var(x, ddof = 1)

MATRIX

array는 Matrix처럼 동작한다.

2차원 matrix

$n \times m$ Matrix

$$\begin{array}{c} \\ n \text{ rows} \end{array} \overbrace{\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0\,m-1} \\ a_{10} & a_{11} & & \vdots \\ \vdots & & & \vdots \\ a_{n-1\,0} & & \cdots & a_{n-1\,m-1} \end{bmatrix}}^{m \text{ columns}}$$

```
ex) X_2 = np.array([[1,2,3],[4,5,6],[7,8,9]]);
    print(X_2);
    print(X_2.shape);
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
(3,3)
```

3차원 matrix

$r \times n \times m$ matrix : $n \times m$ 짜리 2차원 행렬이 r개

```
ex) np.zeros([2,2,3])
```

```
[[[0. 0. 0.]
  [0. 0. 0.]]
 [[0. 0. 0.]
  [0. 0. 0.]]]
```

indexing & slicing

array_name [i, j] → 2차원의 indexing → a_{ij}

array_name [: i, j :] → 2차원의 slicing → $\begin{bmatrix} a_{0j} & \cdots & a_{0\,m-1} \\ \vdots & & \\ a_{i-1\,j} & \cdots & a_{i-1\,m-1} \end{bmatrix}$

```
ex) X_2[[True, True, False], :]
```

row 0 row 1 row 2

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

⚠ boolean을 쓸 때, 차원이 망가질수도 있으므로 주의

ex) X_3 : [[[1 2 3]
[4 5 6]]
[[7 8 9]
[10 11 12]]]

print(X_3 > 5) : [[[False False False]
[False False True]]
[[True True True]
[True True True]]]

print(X_3[X_3 > 5]) : [6 7 8 9 10 11] → 망가짐

matrix의 사칙연산

elementwise

↳ 원소끼리 사칙연산이 수행된다.

⚠ 차원이 다르면 오류!

ex) X_1 : [[1 2 3]
[4 5 6]]

X_2 : [[1 0 3]
[4 0 6]]

X_1 + X_2 == [[2 2 6
8 5 12]]

X_1 / X_2 == [[1 INF 1]
1 INF 1]]

↳ array에서 divide by 0은 오류가 아닌 warning
⇒ 실행됨

행렬곱

$\mathbb{R}^{n \times m}$ A와
$\mathbb{R}^{m \times n}$ B는 행렬곱 연산 (@)가 성립한다.

☆ 행렬곱의 조건 : A col의 수 == B row의 수

ex) $\mathbb{R}^{2\times2\times3}$ @ $\mathbb{R}^{3\times2}$ ⇒ $\mathbb{R}^{2\times2\times2}$

행렬곱의 성질

- 교환 법칙이 성립하지 않는다. # A@B != B@A
- 항등원은 상수가 아닌 행렬이다. # A@I == A
- 역원은 0 행렬을 제외하더라도 존재하지 않을 수 있다.

항등행렬

$I = [a_{ij}]$

$$a_{ij} = \begin{cases} 1 & \text{if } i == j \\ 0 & \text{o.w} \end{cases}$$

ex) np.identity(4) # 4x4 I 생성

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

역원

역원이란 $A @ B = I$을 만족시키는 B를 A의 역원이라한다.

$B = A^{-1}$ 으로 표기한다. (T랑 다름에 유의

역행렬은 존재하지 않을 수 있다. ex) [[1 1] [1 1]]

ex) np.linalg.inv(X-1) ⇒ 근사치로 계산한다

matrix function

function	input	output
np.random.rand(x,y)	int x, int y	x행 y열의 행렬 생성 후 0~1값으로 초기화
np.min(x,axis)	matrix x, axis = 0,1	axis : 0 (default) ⇒ row들끼리 비교했을 때 최솟값 axis : 1 ⇒ column들끼리 비교했을 때 최솟값
np.mean(x,axis,ddof)	matrix x, axis, ddof	axis : 0 ⇒ row들끼리의 평균값
np.size(x)	matrix x	x의 전체 원소 수 반환
np.shape(x)	matrix x	x의 구조 반환
np.ndim(x)	matrix x	x의 차원 수 반환
np.transpose(x)	matrix x	x의 전치행렬 반환 // x.T도 가능
np.reshape(x,(a,b))	matrix x, (a,b)	x의 구조를 axb로 변환 ↳ x의 사이즈 == axb 이어야 한다.

ex) $x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 3 & 1 \\ 5 & 6 & 2 \\ 7 & 8 & 9 \end{bmatrix}$

np.min(x, axis=0) ⟶ [1 2 1]

np.mean(x, axis=0, ddof=1) ⟶ [4.xx 4.xx 4.xx]