파이썬 라이브러리를 활용한 데이터 분석

3장 numpy 기본: 배열과 벡터 연산

2020.06.25 2h

3장 numpy 기본: 배열과 벡터 연산

누적 값 계산

2020.06.25 1h

4.7 계단 오르내리기

- **계단을 오르거나(+1) 내리거나(-1) 값의 누적합**
  - 리스트 walk에는 난수 0, 1의 누적 합이 저장

```
In [143]: np.random.seed(12345)

In [145]: np.random.seed(12345)
          nsteps = 1000
          draws = np.random.randint(0, 2, size=nsteps) #0, 1중 난수 발생
          steps = np.where(draws > 0, 1, -1)
          walk = steps.cumsum()
          walk[:20]

Out[145]: array([-1,  0,  1,  2,  1,  2,  1,  0,  1,  0,  1,  2,  1,  2,  3,  2,  3,
                  4,  5,  4], dtype=int32)

In [146]: walk.min()

Out[146]: -3

In [147]: walk.max()

Out[147]: 31
```

계단 처음에서 10칸 이상 떨어지기까지 소요 횟수

- **np.abs(walk) >= 10**
  - 리스트 walk에서 절대 값이 10 이상인 원소의 논리 배열
    - **처음위치에서 10 칸 이상 떨어진 시점을 알려주는 논리 배열**
- **처음 색인을 반환하는 argmax() 사용**
  - (np.abs(walk) >= 10).argmax()
    - **처음위치에서 10 칸 이상 떨어진 첫 위치 반환**
      - **37**

```
In [146]: walk.min()
Out[146]: -3

In [147]: walk.max()
Out[147]: 31

In [148]: np.array([False, False, True, False]).argmax()
Out[148]: 2

In [150]: np.array([1, 2, 30, 5, 30]).argmax()
Out[150]: 2

In [152]: (np.abs(walk) >= 10).argmax()
Out[152]: 37
```

5000번 모의 실험

- **2차원 배열 walks (5000 x 1000)**

In [164]:
```
np.random.seed(12345)
nwalks = 5000
nsteps = 1000
draws = np.random.randint(0, 2, size=(nwalks, nsteps)) # 0 or 1
steps = np.where(draws > 0, 1, -1)
steps
```

Out[164]:
```
array([[-1,  1,  1, ..., -1,  1,  1],
       [ 1, -1,  1, ...,  1, -1,  1],
       [ 1, -1, -1, ..., -1, -1, -1],
       ...,
       [-1, -1,  1, ..., -1, -1, -1],
       [ 1,  1, -1, ..., -1,  1,  1],
       [ 1,  1,  1, ..., -1, -1,  1]])
```

In [165]:
```
steps.shape
```

Out[165]:
```
(5000, 1000)
```

In [166]:
```
walks = steps.cumsum(1)
walks
```

Out[166]:
```
array([[ -1,   0,   1, ...,  12,  13,  14],
       [  1,   0,   1, ...,   8,   7,   8],
       [  1,   0,  -1, ...,  34,  33,  32],
       ...,
       [ -1,  -2,  -1, ..., -16, -17, -18],
       [  1,   2,   1, ...,  24,  25,  26],
       [  1,   2,   3, ...,  14,  13,  14]], dtype=int32)
```

In [167]:
```
walks.max() #원소의 최대 값
```

Out[167]:
```
138
```

In [168]:
```
walks.min() #원소의 최소 값
```

Out[168]:
```
-133
```

어느 행이 조건(30 칸 이상 떨어지는)을 만족하는 지 검사

- 각 행에서 30 칸 이상 떨어진 점이 있는 지의 논리 배열
  - hits30 = (np.abs(walks) >= 30).any(1)
    - '논리 값이 True인 실험에서 30 칸 이상 떨어진 점이 있다'
      - 논리 값이 False인 실험에서 30 칸 이상 떨어진 점이 없다라는 의미
- 5천 번의 실험 중에 누적 합이 30 또는 -30 이상 모의 실험 횟수
  - hits30.sum() # Number that hit 30 or -30
    - 실험에서 30 칸 이상 떨어진 점이 있는 총 모의실험 횟수
      - 3411

```
In [169]: hits30 = (np.abs(walks) >= 30).any(1)
          hits30

Out[169]: array([ True, False,  True, ..., False, False,  True])

In [170]: len(hits30)

Out[170]: 5000

In [171]: hits30.sum() # Number that hit 30 or -30, 누적 합이 30 또는 -30 이상 되는 수

Out[171]: 3411
```

조건이 맞는 행(30 칸 이상 떨어지는)에서 처음으로 30칸 이상 떨어지는 지점 찾기

- **30 칸 이상 떨어진 점이 있는 행(walks[hits30])에서 처음으로 30칸 이상 떨어지는 지점을 반환**
  - crossing_times = (np.abs(walks[hits30]) >= 30).argmax(1)
    - **walks에서 hits30이 만족하는 행(절대값이 30이 넘는 경우)을 선택한 후**
      - **축 1에 따라(각 행에서) 최대값 첨자를 구하면**
      - **=> 처음으로 30칸 이상 떨어지는 지점을 반환**
  - 절대값이 30이 넘는 실험 행에서 누적 절대값이 30 이상이 되는 최소 횟수의 목록
    - **이 목록의 수도 hits30의 수와 같이 3411개**
- **절대값이 30이 넘는 실험 행에서 누적 절대값이 30 이상이 되는 평균 횟수**
  - crossing_times.mean()
    - **499**

```
In [179]: # 절대값이 30이 넘는 실험 행에서 누적 절대값이 30 이상이 되는 최소 횟수
          crossing_times = (np.abs(walks[hits30]) >= 30).argmax(1)
          crossing_times

Out[179]: array([909, 735, 409, ..., 327, 453, 447], dtype=int64)

In [178]: crossing_times.shape

Out[178]: (3411,)

In [181]: #절대값이 30이 넘는 실험 행에서 누적 절대값이 30 이상이 되는 평균 횟수
          crossing_times.mean()

Out[181]: 499.00996775139254

In [182]: crossing_times[:10]

Out[182]: array([909, 735, 409, 253, 161, 821, 393, 527, 437, 183], dtype=int64)
```

다른 정규 분포에서 실험

- **평균(loc) 0, 표준편차(scale) 0.25인 정규분포에서 난수 방생**
  - steps = np.random.normal(loc=0, scale=0.25, size=(nwalks, nsteps))

3장 numpy 기본:
배열과 벡터 연산

배열 결합과 분리

2020.06.25 1h

배열 결합: concatenate

- **배열 합치기, 기본은 0 축(세로)으로**
  - np.concatenate((a1, a2, ...), axis=0)
  - a1, a2.....: 배열

In [2]:
```
# 데모 배열
a = np.arange(1, 7).reshape((2, 3))
pprint(a)
b = np.arange(7, 13).reshape((2, 3))
pprint(b)
```

```
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[1 2 3]
[4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[ 7  8  9]
[10 11 12]]
```

In [3]:
```
# axis=0 방향으로 두 배열 결합, axis 기본값=0
result = np.concatenate((a, b))
result
```

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

In [4]:
```
# axis=0 방향으로 두 배열 결합, 결과 동일
result = np.concatenate((a, b), axis=0)
result
```

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

In [5]:
```
# axis=1 방향으로 두 배열 결합, 결과 동일
result = np.concatenate((a, b), axis=1)
result
```

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

배열 결합: vstack

- **수직 방향 배열 결합**
- **np.vstack(tup)**
  - tup: 튜플
  - 튜플로 설정된 여러 배열을 수직 방향으로 연결 (axis=0 방향, 세로)
  - np.concatenate(tup, axis=0)와 동일

```
In [6]: # 데모 배열
        a = np.arange(1, 7).reshape((2, 3))
        pprint(a)
        b = np.arange(7, 13).reshape((2, 3))
        pprint(b)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[1 2 3]
[4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[ 7  8  9]
[10 11 12]]

In [7]: np.vstack((a, b))

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

In [8]: # 4개 배열을 튜플로 설정
        np.vstack((a, b, a, b))

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

배열 결합: hstack

- **수평 방향 배열 결합**
- **np.hstack(tup)**
  - tup: 튜플
  - 튜플로 설정된 여러 배열을 수평 방향으로 연결 (axis=1 방향, 가로)
  - np.concatenate(tup, axis=1)와 동일

```
In [9]: # 데모 배열
        a = np.arange(1, 7).reshape((2, 3))
        pprint(a)
        b = np.arange(7, 13).reshape((2, 3))
        pprint(b)

shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[1 2 3]
[4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
Array's Data
[[ 7  8  9]
[10 11 12]]

In [10]: np.hstack((a, b))

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

In [11]: np.hstack((a, b, a, b))

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

배열 분리: hsplit

- **np.hsplit(ary, indices_or_sections)**
  - 지정한 배열을 수평(행) 방향으로 분할

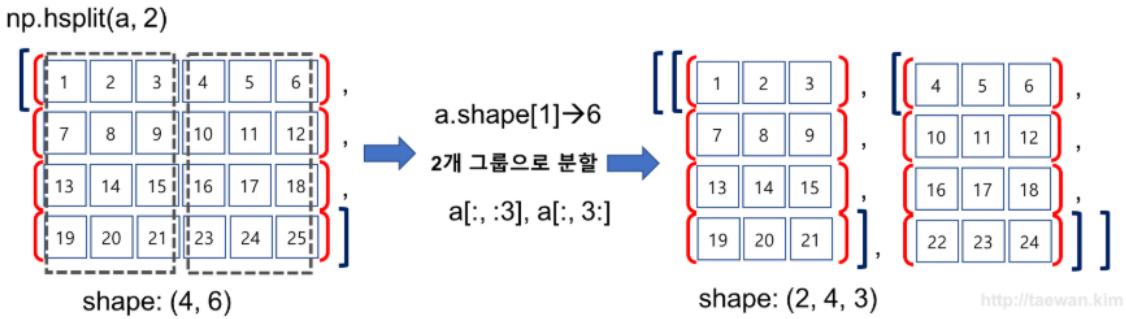
배열 분리

- **hsplit**
  - 결과는 배열의 리스트

In [2]:
```
# 분할 대상 배열 생성
a = np.arange(1, 25).reshape((4, 6))
pprint(a)
```

```
shape: (4, 6), dimension: 2, dtype:int64
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

수평 방향으로 배열을 두 그룹으로 분할

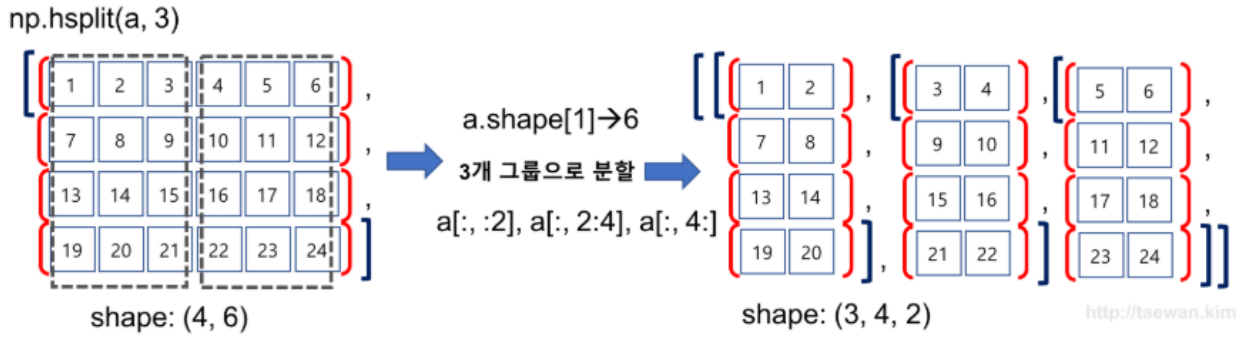


In [3]:
```
# 수평으로 두 그룹으로 분할하는 함수
result = np.hsplit(a, 2)
result
```

Out[3]:
```
[array([[ 1,  2,  3],
        [ 7,  8,  9],
        [13, 14, 15],
        [19, 20, 21]]), array([[ 4,  5,  6],
        [10, 11, 12],
        [16, 17, 18],
        [22, 23, 24]])]
```

배열 분리: hsplit

수평 방향으로 배열을 세 그룹으로 분할



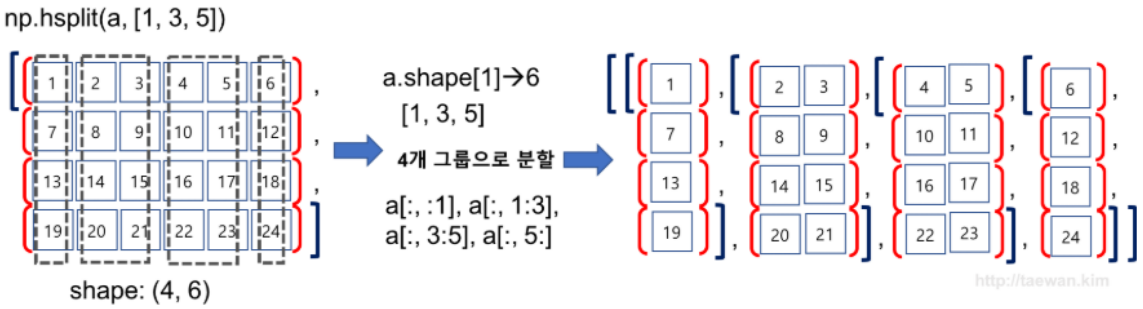
In [4]:
```
# 수평으로 두 그룹으로 분할하는 함수
result = np.hsplit(a, 3)
result
```

Out[4]:
```
[array([[ 1,  2],
        [ 7,  8],
        [13, 14],
        [19, 20]]), array([[ 3,  4],
        [ 9, 10],
        [15, 16],
        [21, 22]]), array([[ 5,  6],
        [11, 12],
        [17, 18],
        [23, 24]])]
```

배열 분리: hsplit

수평 방향으로 여러 구간으로 구분

- np.hsplit의 두 번째 파라미터에 구간 설정 배열을 전달하여 여러 배열로 구분합니다.




```
In [5]: np.hsplit(a, [1, 3 ,5])
```

```
Out[5]: [array([[ 1],
                [ 7],
                [13],
                [19]]), array([[ 2,  3],
                [ 8,  9],
                [14, 15],
                [20, 21]]), array([[ 4,  5],
                [10, 11],
                [16, 17],
                [22, 23]]), array([[ 6],
                [12],
                [18],
                [24]])]
```

배열 분리: vsplit

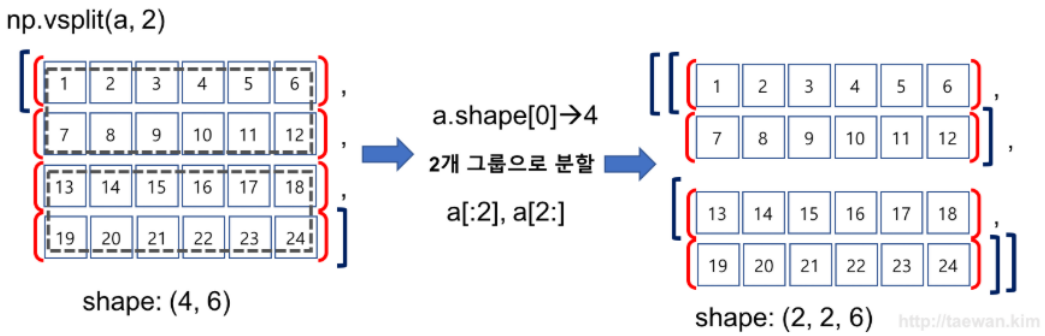
- **배열을 수직 방향(행 방향)으로 분할하는 함수**
  - np.vsplit(ary, indices_or_sections)

배열 분리: vsplit

In [6]:
```
# 분할 대상 배열 생성
a = np.arange(1, 25).reshape((4, 6))
pprint(a)
```

```
shape: (4, 6), dimension: 2, dtype:int64
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

수직 방향으로 배열을 두 개 그룹으로 분할



In [7]:
```
result=np.vsplit(a, 2)
result
```

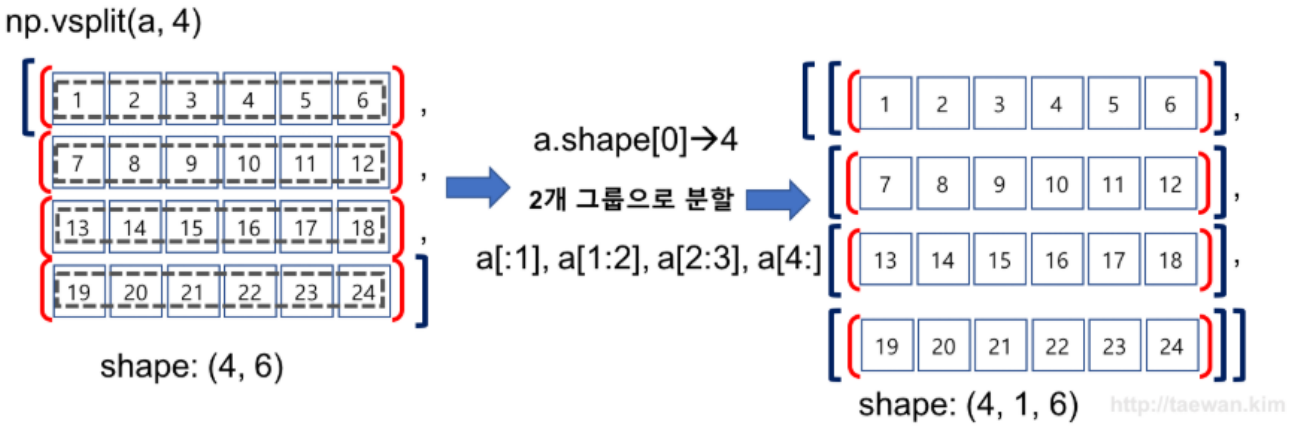
Out[7]:
```
[array([[ 1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 10, 11, 12]]), array([[13, 14, 15, 16, 17, 18],
        [19, 20, 21, 22, 23, 24]])]
```

In [8]:
```
np.array(result).shape
```

Out[8]:
```
(2, 2, 6)
```

배열 분리: vsplit

수직 방향으로 배열을 4 개 그룹으로 분할




```
In [9]: result=np.vsplit(a, 4)
        result
```

```
Out[9]: [array([[1, 2, 3, 4, 5, 6]]),
         array([[ 7,  8,  9, 10, 11, 12]]),
         array([[13, 14, 15, 16, 17, 18]]),
         array([[19, 20, 21, 22, 23, 24]])]
```

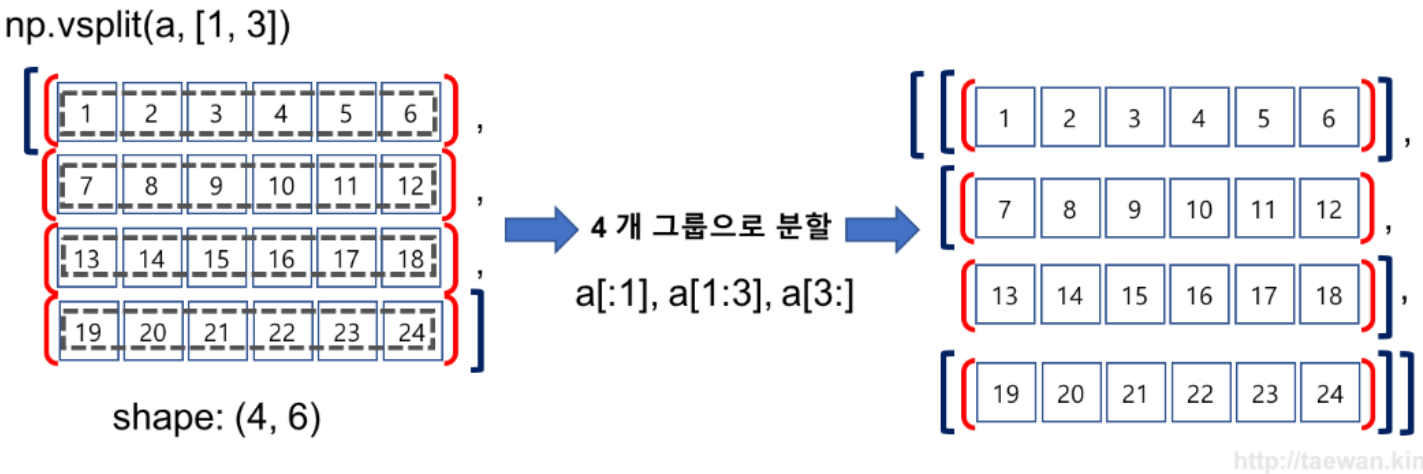
```
In [10]: np.array(result).shape
```

```
Out[10]: (4, 1, 6)
```

배열 분리: vsplit

수직 방향으로 여러 구간으로 구분

- np.hsplit의 두 번째 파라미터에 구간 설정 배열을 전달하여 여러 배열로 구분합니다.

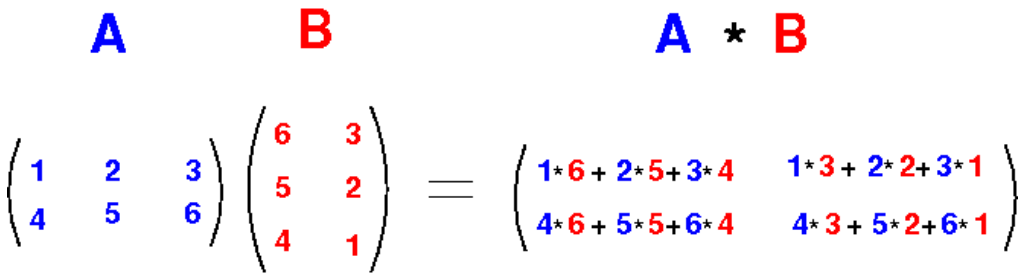



```
In [11]: # row를 1, 2-3, 4번째 라인으로 구분
         np.vsplit(a, [1, 3])

Out[11]: [array([[1, 2, 3, 4, 5, 6]]), array([[ 7,  8,  9, 10, 11, 12],
                 [13, 14, 15, 16, 17, 18]]), array([[19, 20, 21, 22, 23, 24]])]
```

행렬 곱(내적)

- np.dot(a, b)
- a.dot(b)



$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$C_{ij} = \sum_k A_{ik} B_{kj} = A_{ik} B_{kj}$$

3장 numpy 기본: 배열과 벡터 연산

브로드캐스팅

2020.06.25 1h

브로드캐스팅

- **다른 모양의 배열 간의 산술 연산 방법**
  - 배열 + 4(스칼라)
    - **4는 배열의 모든 원소로 브로드캐스트(전파)되어 계산**

```
In [6]: arr = np.arange(5)
        arr
Out[6]: array([0, 1, 2, 3, 4])

In [7]: arr * 4
Out[7]: array([ 0,  4,  8, 12, 16])
```

2차원과 1차원 간의 브로드캐스트

- **1차원을 2차원 모양으로 변환**
  - 1차원을 2차원으로 확대
    - **내부 값을 복사**

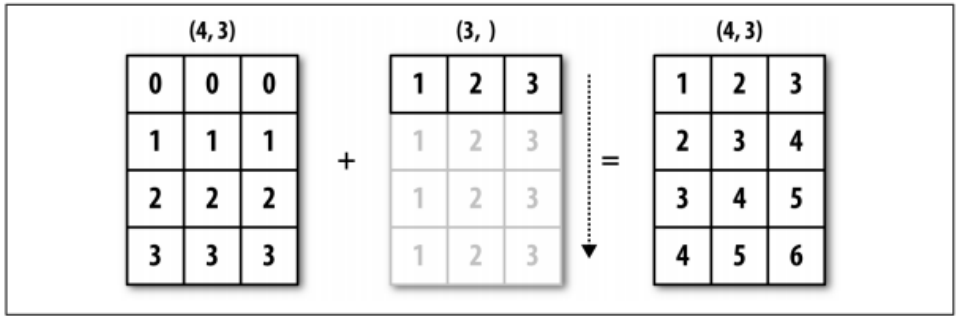


Figure A-4. Broadcasting over axis 0 with a 1D array

```
In [8]: arr = np.random.randn(4, 3)
        arr
Out[8]: array([[-0.2047,  0.4789, -0.5194],
               [-0.5557,  1.9658,  1.3934],
               [ 0.0929,  0.2817,  0.769 ],
               [ 1.2464,  1.0072, -1.2962]])

In [9]: arr.mean(0)
Out[9]: array([0.1447, 0.9334, 0.0867])

In [10]: demeaned = arr - arr.mean(0)
         demeaned
Out[10]: array([[-0.3494, -0.4545, -0.6061],
                [-0.7005,  1.0324,  1.3067],
                [-0.0518, -0.6517,  0.6823],
                [ 1.1017,  0.0738, -1.3829]])

In [11]: demeaned.mean(0)
Out[11]: array([0., 0., 0.])
```

열을 늘리도록 브로드캐스트

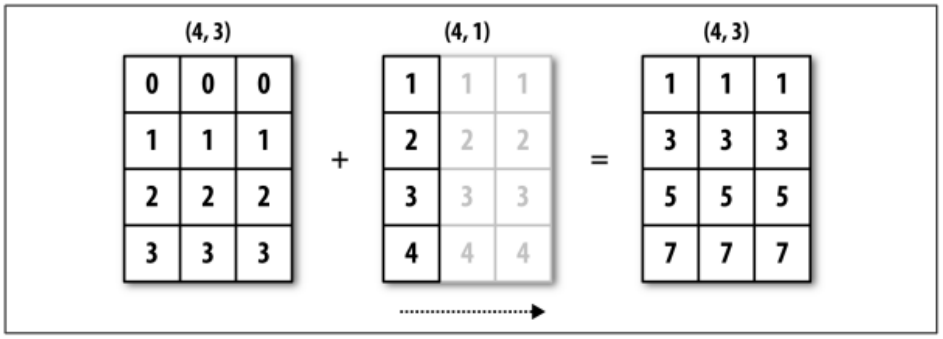


Figure A-5. Broadcasting over axis 1 of a 2D array

```
In [12]: arr
Out[12]: array([[-0.2047,  0.4789, -0.5194],
                [-0.5557,  1.9658,  1.3934],
                [ 0.0929,  0.2817,  0.769 ],
                [ 1.2464,  1.0072, -1.2962]])

In [13]: row_means = arr.mean(1)
         row_means
Out[13]: array([-0.0817,  0.9345,  0.3812,  0.3191])

In [14]: row_means.shape
Out[14]: (4,)

In [15]: row_means.reshape((4, 1))
Out[15]: array([[-0.0817],
                [ 0.9345],
                [ 0.3812],
                [ 0.3191]])

In [16]: demeaned = arr - row_means.reshape((4, 1))
         demeaned
Out[16]: array([[-0.123 ,  0.5607, -0.4377],
                [-1.4902,  1.0313,  0.4589],
                [-0.2883, -0.0995,  0.3878],
                [ 0.9273,  0.6881, -1.6154]])

In [17]: demeaned.mean(1)
Out[17]: array([ 0., -0.,  0.,  0.])
```

다양한 모양의 브로드캐스팅

- **각각 모양을 모두 수정**

```
In [29]: a = np.array(range(4))
         a

Out[29]: array([0, 1, 2, 3])

In [28]: b = np.array(range(4)).reshape(4, 1)
         b

Out[28]: array([[0],
                [1],
                [2],
                [3]])

In [30]: a + b

Out[30]: array([[0, 1, 2, 3],
                [1, 2, 3, 4],
                [2, 3, 4, 5],
                [3, 4, 5, 6]])
```

오류 발생

- **모양 변형이 안되면 오류**

```
In [23]: a = np.array(range(3))
         a

Out[23]: array([0, 1, 2])

In [26]: b = np.array(range(8)).reshape(4, 2)
         b

Out[26]: array([[0, 1],
                [2, 3],
                [4, 5],
                [6, 7]])

In [27]: a + b

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-bd58363a63fc> in <module>
----> 1 a + b

ValueError: operands could not be broadcast together with shapes (3,) (4,2)
```

실습

- **교재 파일**
  - ch04.ipynb
  - Ch04-study.ipynb로 복사해서 연습
- **난수와 실수의 정확도**
  - import numpy as np
  - np.random.seed(12345)
    - **난수를 발생시키기 위한 초기 값 지정**
      - **이후 난수가 동일하게 발생**
  - np.set_printoptions(precision=4, suppress=True)
    - **precision=4: 소수점 이하 반올림해 4개 표시**
    - **np.array(3.123456)**
      - **array(3.1235)**
    - **suppress=True: e-04와 같은 scientific notation을 억제하고 싶으면**
- **Alt + Enter**
  - 현재 셀 실행 후, 다음 셀 삽입
- **Ctrl + shift + enter**
  - 셀 분리