vstack vs hstack

1차원 벡터끼리의 결합

```
import numpy as np

a = np.random.randint(0, 10, (4,)) # 1차 벡터
b = np.random.randint(0, 10, (4,)) # 1차 벡터

print(f"a: {a.shape}\n{a}")
print(f"a: {b.shape}\n{b}\n")

vstack = np.vstack([a, b])
hstack = np.hstack([a, b])

print(f"vstack: {vstack.shape}\n{vstack}")
print(f"hstack: {hstack.shape}\n{hstack}\n")
```

output :

```
a: (4,)
[7 6 8 3]
a: (4,)
[2 3 9 7]

vstack: (2, 4)
[[7 6 8 3]
 [2 3 9 7]]
hstack: (8,)
[7 6 8 3 2 3 9 7]
```

a, b는 각각 4개의 원소를 가진 1차원 벡터 입니다.

우리는 여기서 vstack과 hstack를 이용해 두 배열을 합치려고 합니다.

행방향

여기서 **vstack**은 **vertical stack** 즉 수직방향으로 쌓는다는 의미이고, **hstack**은 **horizontal stack** 즉 수평방향으로 쌓는 다는 의미입니다.

열방향

사용방법은 vstack/hstack 인자로 **리스트**나 **튜플**과 함께 값을 넣어주면됩니다. 여기선 **리스트**안에 각각의 벡터를 a , b 변수로 받아 집어 넣었습니다.

참고사항으로 수학에서는 벡터를 column 벡터를 기본으로 간주하지만 데이터사이언스에는 row벡터를 기본으로 설정합니다.

이러한 이유로 넘파이에선 기본적으로 1차원벡터를 row 벡터라고 가정을 하지요.

즉 vstack을 이용해 수직으로 쌓는다면 오른쪽이 아닌 왼쪽같이 쌓인다는 말이죠.

이번엔 **컬럼 벡터**를 만들어볼까요?

```python
import numpy as np

a = np.random.randint(0, 10, (4,1)) # column 벡터
b = np.random.randint(0, 10, (4,1)) # column 벡터

print(f"a: {a.shape}\n{a}")
print(f"a: {b.shape}\n{b}\n")

vstack = np.vstack((a, b))
hstack = np.hstack((a, b))

print(f"vstack: {vstack.shape}\n{vstack}")
print(f"hstack: {hstack.shape}\n{hstack}\n")
#
```

output :

```
a: (4, 1)
[[5]
 [6]
 [1]
 [4]]
a: (4, 1)
[[9]
 [1]
 [5]
 [8]]

vstack: (8, 1)
[[5]
 [6]
 [1]
 [4]
 [9]
 [1]
 [5]
 [8]]
hstack: (4, 2)
[[5 9]
 [6 1]
 [1 5]
 [4 8]]
```

dstack 명령은 제3의 축 즉, 행이나 열이 아닌 깊이(depth) 방향으로 배열을 합친다. 가장 안쪽의 원소의 차원이 증가한다. 즉 가장 내부의 숫자 원소가 배열이 된다. shape 정보로 보자면 가장 끝에 값이 2인 차원이 추가되는 것이다.이 예제의 경우에는 shape 변화가 2개의 (3 x 4) -> 1개의 (3 x 4 x 2)가 된다.

```
c1 = np.ones((3, 4))
c1
```

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

```
c2 = np.zeros((3, 4))
c2
```

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

```
np.dstack([c1, c2])
```

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

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

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

```
(np.dstack([c1, c2])).shape
```

```
(3, 4, 2)
```