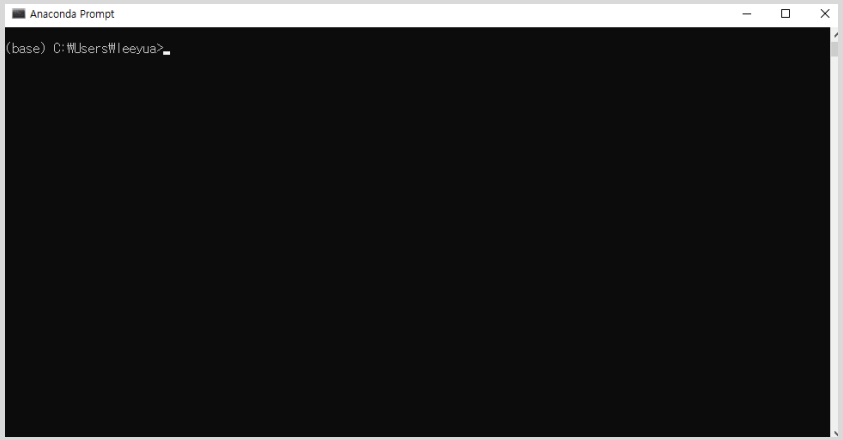
0장 – 라이브러리 설치하기

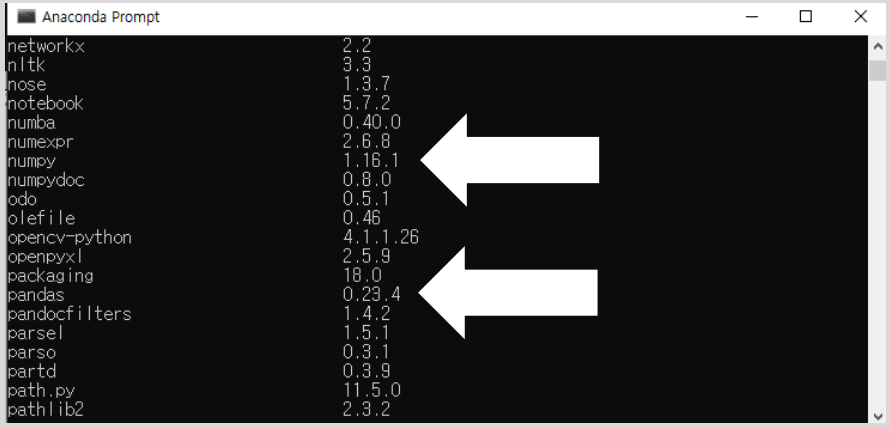
pip 간단 사용법 익히기

- 아나콘다 프롬프트에서 실행할 수 있습니다.
- 돋보기 버튼을 누르고 'anaconda prompt'를 검색한 다음 아나콘다 프롬프트를 실행하세요.



내 컴퓨터에 설치된 파이썬 라이브러리 확인하기

```
pip list
```



파이썬 라이브러리 설치하기

- 'pip install'을 입력한 다음 한 칸을 띄고 라이브러리 이름을 입력하면 파이썬 라이브러리를 설치하거나 제거할 수 있습니다.
- 실습을 위해 pandas / numpy / matplotlib 를 설치해주세요.

```
pip install pandas
```

```
pip install numpy
```

```
pip install matplotlib
```

1장 – 벡터와 행렬(numpy)

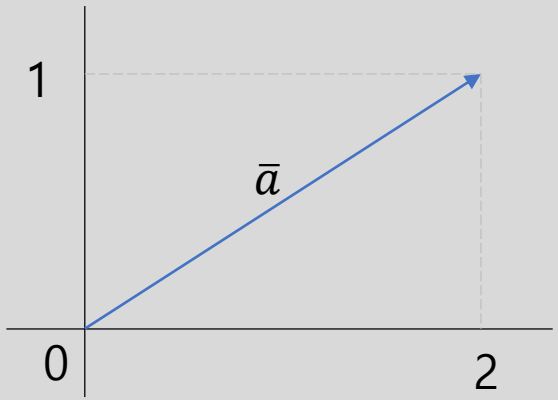
1.1.1 스칼라, 벡터, 행렬 텐서

숫자를 표현하는 데 사용하는 정수와 소수 등으로 크기만을 나타내는 양을 '스칼라' 라고합니다.

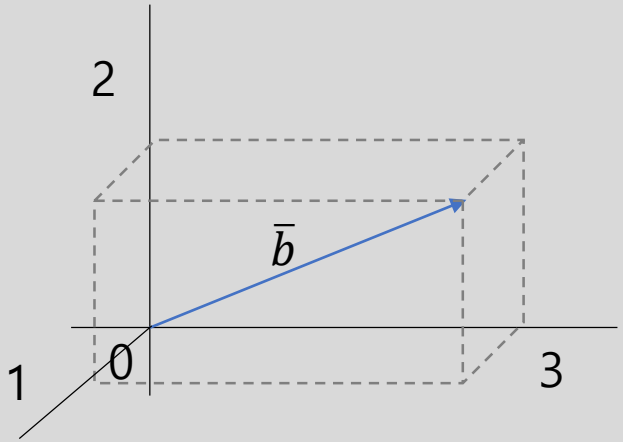
3이나 -2, 1.45 등을 값 하나로 표현합니다.

스칼라끼리 덧셈, 뺄셈, 곱셈 등을 계산한 결과도 마찬가지로 스칼라 입니다.

크기뿐만 아니라 방향도 있는 양을 '벡터' 라고 합니다. 예를 들어 아래 그림같은 평면과 공간에서 크기와 방향을 생각해봅시다.



평면에서의 벡터



공간에서의 벡터

데이터를 일렬로 정리하는 벡터

벡터는 크기 뿐만 아니라 방향이 있으므로 알파벳 위에 화살표를 사용하여 $\vec{a}$, $\vec{b}$ 처럼 표현합니다. 또한 $\boldsymbol{a}, \boldsymbol{b}$ 처럼 굵은 알파벳으로 표현할 수도 있습니다. 우리 기초 파이썬 굵은 글씨로 표현하도록 하겠습니다.

벡터의 크기와 **방향**은 좌표축 성분으로 나눠 표현합니다. 2차원이라면 x좌표와 y좌표, 3차원이면 x 좌표, y좌표, z좌표를 순서대로 적습니다. 예를들어 2차원 좌표 평면은 $\boldsymbol{a}$ = [2,1], 3차원 좌표 평면은 $\boldsymbol{b}$ = [3,2,1] 처럼 적습니다. 벡터의 원소를 표현할 때는 '( )'를 사용하여 $\boldsymbol{a}$ = (2,1)처럼 표현할 수도 있습니다.

원소를 나열하는 방법은 가로로 나열하는 '행 벡터'와 세로로 나열하는 '열 벡터'가 있습니다. 아래 식은 열 벡터를 보여줍니다.

$$\boldsymbol{a} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

행 벡터 혹은 열 벡터를 합해 여러 개 원소를 가로 혹은 세로로 나열한 것을 '행렬' 이라고 합니다. 행렬은 $\boldsymbol{A}$ 처럼 굵은 대문자로 표현하고 원소는 아래의 식처럼 대괄호 안에 넣습니다.

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

데이터를 일렬로 정리하는 벡터

여러 개 벡터를 '곱 연산'을 사용해 복합적으로 연결시킨 구조를 '텐서(tensor)' 라고 합니다. 보통 아래의 식처럼 두 벡터의 행렬 곱셈 결과가 텐서가 되는 텐서곱 연산을 사용하죠.

$$\boldsymbol{a} \otimes \boldsymbol{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} [b_1\ b_2\ b_3] = \begin{bmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{bmatrix}$$

보통 프로그래밍에서 스칼라는 0차원 배열(파이썬은 리스트), 벡터는 1차원 배열, 행렬은 2차원 배열로 표현합니다.
텐서의 개념이라면 스칼라는 0계층 텐서, 벡터는 1계층 텐서, 행렬은 2계층 텐서로 표현합니다.

머신러닝에서는 이러한 행렬 데이터를 쉽게 다루는 라이브러리를 사용하게 됩니다.
넘파이의 ndarray(N Dimension Array)라는 클래스 입니다.
array 함수로 다차원 배열을 처리하는 객체를 만들어 행렬을 쉽게 생성할 수 있습니다.

넘파이에서 행렬을 생성할 때는 다차원 리스트를 인자로 전달합니다. 이제 이것을 코드로 구현해 보도록 하겠습니다.

데이터를 일렬로 정리하는 벡터

```
import numpy as np

a = np.array([2,1])
print(a)

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

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

```
[2 1]

[3 2 1]

[[2 4 1]
 [6 3 5]]
```

이 코드처럼 넘파이의 array 함수를 사용할 때는 인자로 사용하는 리스트 요소를 같은 형태로 구성해야 한다는 점에 주의 해주셔야 합니다.

한편 넘파이 외에도 레이블 및 값을 집합 타임으로 변수에 저장하는 판다스(pandas)도 있으니 이부분은 뒤에서 다뤄보도록 하겠습니다.

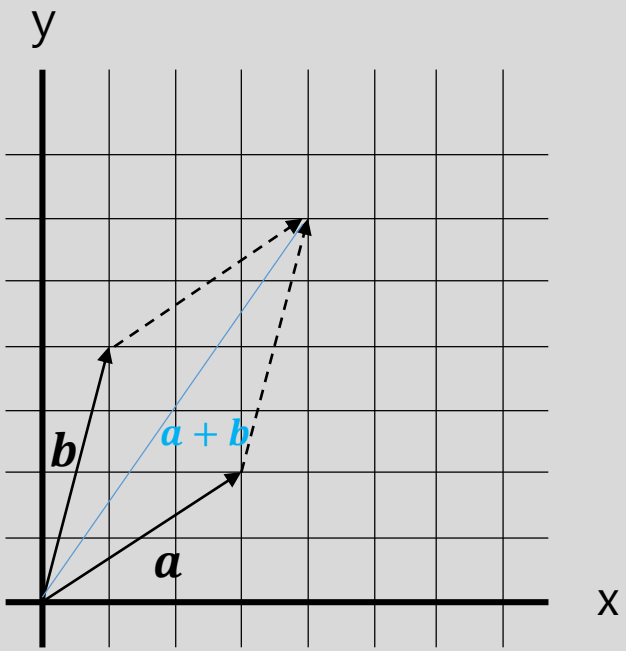
데이터를 일렬로 정리하는 벡터

1.1.2 벡터의 합, 차, 크기

두 벡터가 있을 때 합과 차를 계산해 보겠습니다. 벡터의 합과 차는 서로 대응하는 원소 각각을 더하거나 뺍니다. 예를들어

두 벡터 $\boldsymbol{a} = [3 \quad 2]$, $\boldsymbol{b} = [1 \quad 4]$ 가 있다면 합은 $\boldsymbol{a} + \boldsymbol{b} = [3+1 \quad 2+4] = [4 \quad 6]$, 차는 $\boldsymbol{a} - \boldsymbol{b} = [3-1 \quad 2-4] = [2 \quad -2]$ 처럼 계산합니다.

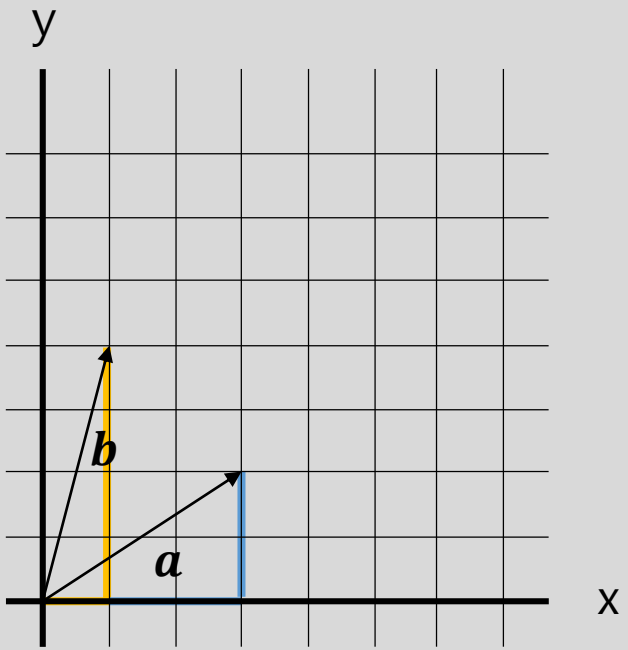
아래의 좌표 평면상의 평행 사변형을 떠올리시면 이해가 쉽습니다.



벡터의 합

데이터를 일렬로 정리하는 벡터

벡터의 크기는 '벡터를 나타내는 화살표 선의 길이' 입니다. 아래 그림처럼 피타고라스의 정리를 사용해 계산할 수 있습니다.



벡터의 크기

벡터의 크기를 식으로 나타낼 때는 벡터 기호에 절댓값을 씌워 표현합니다. 예를들어 $\boldsymbol{a} = [\,3 \quad 2\,], \boldsymbol{b} = [\,1 \quad 4\,]$ 이 있을 때 벡터 $\boldsymbol{a}, \boldsymbol{b}$ 의 크기는 아래처럼 계산할 수 있습니다.

$$|\boldsymbol{a}| = \sqrt{3^2 + 2^2} = \sqrt{13}, \qquad |\boldsymbol{b}| = \sqrt{1^2 + 4^2} = \sqrt{17}$$

데이터를 일렬로 정리하는 벡터

넘파이를 사용해 벡터의 합, 차, 크기를 계산해 봅시다.

```python
import numpy as np
import math

a = np.array([3,2])
b = np.array([1,4])

print(a + b) #벡터의 합
print(a - b) #벡터의 차
print(np.linalg.norm(a)) #벡터 a의 크기
print(np.linalg.norm(b)) #벡터 b의 크기


# 벡터의 크기를 함수로 계산
def norm(x):
    return math.sqrt(sum([i ** 2 for i in x]))
    #조건을 먼저 걸어주고 뒤이어 반복문을 넣어주는 형식

print(norm(a))
print(norm(b))
```

```
[4 6]
[ 2 -2]
3.605551275463989
4.123105625617661
3.605551275463989
4.123105625617661
```

자체함수 norm 함수를 살펴보면

벡터의 크기는 각 리스트를 제곱한 값의 합에

제곱근을 계산하는 것입니다.

이렇게 넘파이를 사용하지 않아도 계산은 가능합니다.

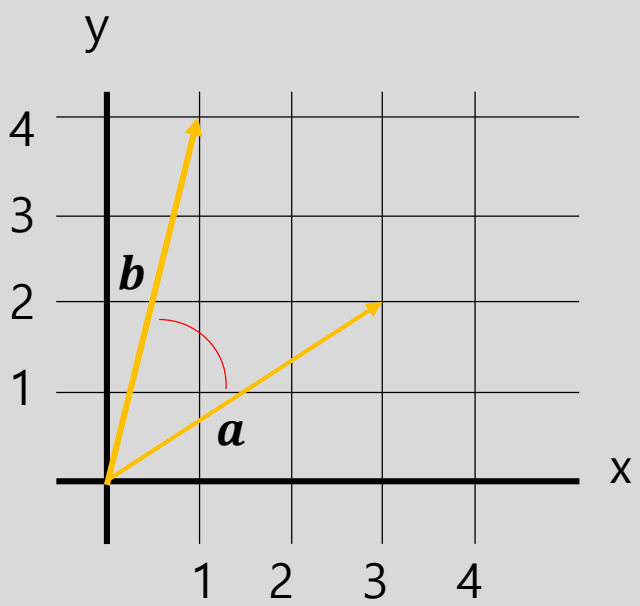
$$|\boldsymbol{a}| = \sqrt{3^2 + 2^2} = \sqrt{13}, \qquad |\boldsymbol{b}| = \sqrt{1^2 + 4^2} = \sqrt{17}$$

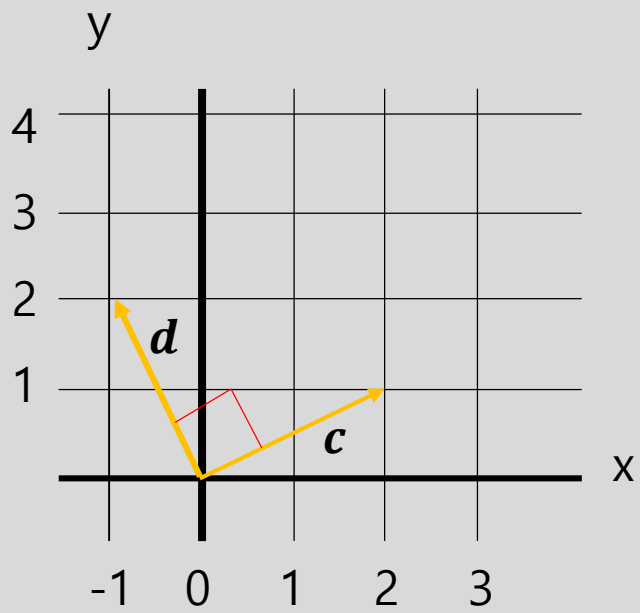
데이터를 일렬로 정리하는 벡터

1.1.3 벡터의 내적

두 벡터의 곱셈은 보통 '내적' 이라고 하며, ' . '이라는 기호로 표현 합니다.

벡터 $\boldsymbol{a} = [a_1 \quad a_2]$와 $\boldsymbol{b} = [b_1 \quad b_2]$ 의 내적은 $\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2$ 로 계산합니다.





왼쪽 두 벡터 $\boldsymbol{a} = [3 \ 2]$, $\boldsymbol{b} = [1 \ 4]$의 내적은 $\boldsymbol{a} \cdot \boldsymbol{b} = 3 * 1 + 2 * 4 = 11$ 로 계산할 수 있습니다.
또한 오른쪽 두 벡터 $\boldsymbol{c} = [2 \ 1]$, $\boldsymbol{d} = [-1 \ 2]$의 내적은 $\boldsymbol{c} \cdot \boldsymbol{d} = 2 * (-1) + 1 * 2 = 0$ 으로 계산할 수 있습니다.

즉, 오른쪽 두 벡터는 직각으로 만나는 상태인 '수직'입니다. 그리고 두 벡터가 수직일 때는 벡터의 내적은 항상 0입니다.
또한 내적이 양수 일때는 두 벡터가 이루는 각이 90도 보다 작고, 내적이 음수 일때는 두 벡터가 이루는 각이 90도 보다 커집니다.

데이터를 일렬로 정리하는 벡터

파이썬으로 벡터의 내적을 계산하는 방법은 두가지가 있습니다. 하나는 방금 살펴본 내적 계산 그대로 단순히 리스트 안 요소를 차례로 곱해서 더하는 것 입니다. 다른 하나는 아래의 코드처럼 넘파이의 dot 함수를 사용하는 것 입니다.

```
import numpy as np

a = np.array([3,2])
b = np.array([1,4])

print(a.dot(b))

c = np.array([2,1])
d = np.array([-1,2])

print(c.dot(d))
```

```
11
0
```

벡터의 내적은 $-|\boldsymbol{a}||\boldsymbol{b}| \leq \boldsymbol{a} \cdot \boldsymbol{b} \leq |\boldsymbol{a}||\boldsymbol{b}|$ 라는 관계가 성립합니다. 이를 '코시-슈바르츠 부등식' 이라고 합니다.

두 벡터 a, b 의 방향이 같을 때 내적을 계산하면 $|\boldsymbol{a}||\boldsymbol{b}|$ 고 내적의 최댓값입니다. 마찬가지로 두 벡터의 방향이 반대일 때 내적을 계산하면 $-|\boldsymbol{a}||\boldsymbol{b}|$ 고 내적의 최솟값입니다. 즉, 두 벡터가 이루는 방향으로 내적이 최댓값인지 최솟값인지 알 수 있는 것입니다.

이는 신경망에서 최솟값을 계산하고 싶을 때 응용됩니다.

어떤 벡터의 내적이 최솟값인 벡터가 필요할 때는 방향이 반대인 벡터를 선택하는 방식으로 사용됩니다.

데이터를 일렬로 정리하는 벡터

1.2.1 행렬의 덧셈과 곱셈

행렬은 1개 이상의 수나 식을 가로 '행' 과 세로 '열' 의 사각형으로 표현하는 것입니다.

보통 행 개수가 m 이고 열 개수가 n 이면 $m * n$ 행렬이라고 합니다. 아래 식은 행 2개, 열 3개인 2 * 3 행렬 이라고 합니다.

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

또한 B,C 처럼 행 개수와 열 개수가 같으면 '정사각행렬' 이라고 합니다. 또한 B는 '2차 정사각행렬', C는 '3차 정사각행렬' 이라고 합니다.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

두 행렬의 합과 차는 각 행렬의 같은 위치 원소를 더하거나 빼서 계산합니다. 아래 식은 행렬 X,Y 가 있을 때 X+Y, X-Y 를 계산한 결과입니다.

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \end{bmatrix} \quad Y = \begin{bmatrix} 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} \qquad X + Y = \begin{bmatrix} 4 & 6 & 8 \\ 7 & 9 & 11 \end{bmatrix} \qquad X - Y = \begin{bmatrix} -2 & -2 & -2 \\ -1 & -1 & -1 \end{bmatrix}$$

※ 행 개수와 열 개수가 다른 두 행렬에 대한 합과 차를 연산할 때는 행과 열 개수가 같아야 합니다.

다차원 데이터를 다루는 행렬

앞에서 행렬은 2차원 배열로 표현한다고 했습니다. 따라서 행렬의 합과 차는 리스트의 각 요소를 더하거나 빼면 됩니다.

물론 아래의 코드 처럼 넘파이를 사용해 행렬을 생성한 후 계산을 할 수 있습니다.

```
import numpy as np

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

print( X + Y )
print( X - Y )
```

```
[[ 4  6  8]
 [ 7  9 11]]
[[-2 -2 -2]
 [-1 -1 -1]]
```

한편 행렬 하나를 계속 더하는 것, 예를들면 X + X + X... 은 각 원소를 2배 , 3배 ,4배 ... 등으로 곱한 것과 결과가 같습니다.

따라서 아래의 식 처럼 정의할 수 있습니다.

$$\boldsymbol{X} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \mathbf{6} \end{bmatrix} \quad \mathbf{2}\boldsymbol{X} = \begin{bmatrix} \mathbf{2} & \mathbf{4} & \mathbf{6} \\ \mathbf{8} & \mathbf{10} & \mathbf{12} \end{bmatrix}$$

일반적인 행렬의 곱을 계산할 때는 왼쪽 행렬의 '행'과 오른쪽 행렬의 '열' 을 곱합니다. 아래의 식은 두 행렬 A,B의 곱 A * B 입니다.

곱한 행렬은 AB로 표기 할 수 있습니다.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a_{11}} & \boldsymbol{a_{12}} & \boldsymbol{a_{13}} \\ \boldsymbol{a_{21}} & \boldsymbol{a_{22}} & \boldsymbol{a_{23}} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{b_{11}} & \boldsymbol{b_{12}} \\ \boldsymbol{b_{21}} & \boldsymbol{b_{22}} \\ \boldsymbol{b_{31}} & \boldsymbol{b_{32}} \end{bmatrix}$$

$$\boldsymbol{AB} = \begin{bmatrix} \boldsymbol{a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}} & \boldsymbol{a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32}} \\ \boldsymbol{a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31}} & \boldsymbol{a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32}} \end{bmatrix}$$

이러한 식을 실제 행렬에서 계산해보록 해보겠습니다.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{2} & \boldsymbol{3} \\ \boldsymbol{3} & \boldsymbol{4} & \boldsymbol{5} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{3} & \boldsymbol{4} \\ \boldsymbol{4} & \boldsymbol{5} \\ \boldsymbol{5} & \boldsymbol{6} \end{bmatrix}$$

$$\boldsymbol{AB} = \begin{bmatrix} \boldsymbol{1 * 3 + 2 * 4 + 3 * 5} & \boldsymbol{1 * 4 + 2 * 5 + 3 * 6} \\ \boldsymbol{3 * 3 + 4 * 4 + 5 * 5} & \boldsymbol{3 * 4 + 4 * 5 + 5 * 6} \end{bmatrix} = \begin{bmatrix} \boldsymbol{26} & \boldsymbol{32} \\ \boldsymbol{50} & \boldsymbol{62} \end{bmatrix}$$

행렬의 곱셈이라면 왼쪽 행렬의 '열 개수'와 오른쪽 행렬의 '행 개수'가 같아야 합니다.

보통 왼쪽 행렬이 m * k 행렬이고 오른쪽 행렬이 k * n 행렬일 때, 행렬의 곱 결과는 m * n 입니다.

다차원 데이터를 다루는 행렬

행렬곱의 특성은 잘 기억해 놔야 합니다. 예를 들어 A ~ F 라는 행렬 6개가 있을 때 *ABCDEF* 라는 행렬의 곱을 계산해야 한다고 생각해봅시다. A 가 3*4 행렬이고 F가 5 * 2 행렬이라면 ABCDEF는 3 * 2 행렬이 됩니다. 이렇게 행렬의 곱을 계산할 때는 결과 행렬의 행과 열 개수를 바로 파악할 수 있도록 합니다.

```
import numpy as np

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

print(A.dot(B)) # 행렬의 곱 계산

C = np.matrix([[1,2,3], [3,4,5]])
D = np.matrix([[3,4], [4,5], [5,6]])

print(C * D) # matrix함수를 사용한 행렬의 곱 계산
```

```
[[26 32] [50 62]]
[[26 32] [50 62]]
```

array 함수로 행렬을 만들었다면 dot 함수로 행렬의 곱을 쉽게 계산할 수 있습니다. 또한 넘파이에서 행렬을 다루는 matrix 함수를 사용해서 행렬을 만들 수도 있습니다. matrix 함수로 행렬을 만들어 변수에 저장하면 곱셈을 의미하는 * 연산자를 사용할 수 있다는 장점이 있습니다.

다차원 데이터를 다루는 행렬

행렬의 곱에서 잊으면 안 되는 사실이 하나 있습니다. AB와 BA의 계산 결과가 일치하지 않는다는 점입니다. 즉, '교환 법칙'이 성립하지 않습니다.

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{3} & \mathbf{4} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \mathbf{3} & \mathbf{4} \\ \mathbf{4} & \mathbf{5} \end{bmatrix}$$

$$\boldsymbol{AB} = \begin{bmatrix} \mathbf{11} & \mathbf{14} \\ \mathbf{25} & \mathbf{32} \end{bmatrix}, \qquad \boldsymbol{BA} = \begin{bmatrix} \mathbf{15} & \mathbf{22} \\ \mathbf{19} & \mathbf{28} \end{bmatrix}$$

참고로 왼쪽 위에서 오른쪽 아래 대각선 방향의 행렬 원소가 1이고 나머지 원소가 0이면 '단위행렬' 리고 합니다.

보통 E 와 I 같은 기호로 표현합니다.

$$\boldsymbol{E} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \qquad \boldsymbol{E} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

다차원 데이터를 다루는 행렬

1.2.2 전치행렬과 역행렬

행렬의 행과 열을 교환한 행렬을 '전치행렬' 이라고 합니다. 보통 $\boldsymbol{A}^T$ 로 표현하며, 이 부분은 책 혹은 다른 강의마다 다를 수 있습니다.

이 수업에서는 A^T 로 표현하도록 하겠습니다.

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \mathbf{7} & \mathbf{8} & \mathbf{9} \end{bmatrix} \qquad \boldsymbol{A}^T = \begin{bmatrix} \mathbf{1} & \mathbf{4} & \mathbf{7} \\ \mathbf{2} & \mathbf{5} & \mathbf{8} \\ \mathbf{3} & \mathbf{6} & \mathbf{9} \end{bmatrix}$$

참고로 전치행렬의 전치행렬은$(A^T)^T = A$ 당연히 원래의 행렬입니다.

전치행렬의 경우 어떠한 행 벡터나 열 벡터의 전치 행렬은 열 벡터나 행 벡터가 됩니다. 이 두 벡터의 내적을 계산해 수직인지 아닌지 판단하는 것 같은 계산에 사용합니다.

행렬 A가 있을 때 AB 혹은 BA가, 단위행렬 E인 행렬 B를 A의 '역행렬' 이라고 합니다. A^{-1} 로 표현합니다. 단, 역행렬이 있으려면 해당 행렬이 정사각형 이어야 합니다. 예를들어 아래의 두 행렬 A,B의 AB와 BA를 계산하면 모두 단위 행렬입니다.

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{2} & \mathbf{5} \\ \mathbf{1} & \mathbf{3} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \mathbf{3} & \mathbf{-5} \\ \mathbf{-1} & \mathbf{2} \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \qquad \boldsymbol{BA} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

즉, $B = A^{-1}$ 이고, $\mathrm{A} = B^{-1}$ 입니다.

```
import numpy as np

A = np.array([[2,5],[1,3]])
print(np.linalg.inv(A))
```

```
[[ 3. -5.]
 [-1.  2.]]
```

모든 정사각형 행렬에 역행렬이 있지는 않습니다. 또한 역행렬은 여러개일 수도 있습니다. 그래서 역행렬이 유일하기 있는 정사각형 행렬을 '가역행렬' 이라고 따로 구분합니다.

다차원 데이터를 다루는 행렬

2장 – 판다스 시작하기(pandas)

```python
import pandas as pd

#gapminder데이터 집합은 콜럼비아 대학의 연구용도로 만든 기초 통계분석용 데이터입니다.
df = pd.read_csv('gapminder.tsv',sep='\t')

# 국가 / 대륙 / 연도 / 기대수명 / 인구 / 1인 gdp
print(df.head())

#변수 타입 확인
print(type(df))

#데이터의 행렬 크기 확인
print(df.shape)

#데이터의 컬럼값 확인
print(df.columns)

#데이터 타입확인
print(df.dtypes)
```

```
 country continent  year  lifeExp       pop   gdpPercap
0  Afghanistan      Asia  1952   28.801   8425333  779.445314
1  Afghanistan      Asia  1957   30.332   9240934  820.853030
2  Afghanistan      Asia  1962   31.997  10267083  853.100710
3  Afghanistan      Asia  1967   34.020  11537966  836.197138
4  Afghanistan      Asia  1972   36.088  13079460  739.981106
<class 'pandas.core.frame.DataFrame'>
(1704, 6)
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'],
dtype='object')
country       object
continent     object
year           int64
lifeExp      float64
pop            int64
gdpPercap    float64
dtype: object
```

데이터 집합 불러오기

```
# 열 단위로 데이터 추출하기
country_df = df['country']
print(country_df.head())
print(country_df.tail())

# 여러 열 단위 추출하기
subset = df[['country','continent','year']]
print(subset.head())
print(subset.tail())

#loc 속성으로 행 데이터 추출하기
print(df.loc[0])
print(df.loc[99])

#마지막 행 데이터 추출하기
print(df.tail(n=1))

#원하는 다중 행 출력하기
print(df.loc[[0,99,999]])
```

```
#데이터 행 추출
print(df.iloc[1])
print(df.iloc[99])
print(df.iloc[-1])

#슬라이싱을 활용한 데이터 추출 (변수명)
subset = df.loc[:, ['year','pop']]
print(subset.head())

#슬라이싱을 활용한 데이터 추출 (정수)
subset = df.iloc[:, [2,4, -1]]
print(subset.head())

#다양한 방식의 데이터 추출
print(df.iloc[[0,99,999],[0,3,5]])
print(df.loc[[0,99,999],['country','lifeExp','gdpPercap']])
```

* loc : 속성은 인덱스 활용 데이터 추출
* iloc: 순서를 의미하는 행 번호사용 추출

데이터 추출하기

```
#groupby를 활용한 그룹화한 데이터 기초 통계계산
#(연도별) [기대수명]의 평균 구하기
#(그룹화) [구하고자 하는 값]
print(df.groupby('year')['lifeExp'].mean())
```

```
#([연도별,대륙별]) [[기대수명,1인 gdp]]의 평균 구하기
#(그룹화) [구하고자 하는 값]
print(df.groupby(['year','continent'])[['lifeExp','gdpPercap']].mean())
```

```
#그룹화한 데이터의 개수 즉, 빈도수 구하기
# (대륙별) [나라].빈도수 확인()
print(df.groupby('continent')['country'].nunique())
```

기초적인 통계 계산하기

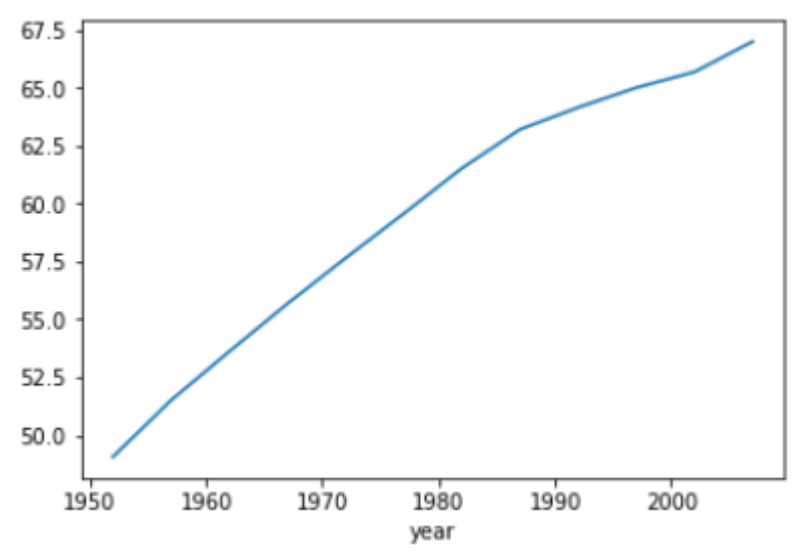
```python
#시각화 도구 matplotlib.pyplot 호출
import matplotlib.pyplot as plt

#(연도별) [기대수명] 평균내어 변수화
global_yearly_life_expectancy = df.groupby('year')['lifeExp'].mean()

#출력
print(global_yearly_life_expectancy)

#그래프 출력
global_yearly_life_expectancy.plot()
plt.show()
```

#시간이 지남에 따라 평균 기대수명은 점차 증가



```
#데이터프레임 만들기
corona_data = pd.DataFrame(
    #실질적으로 들어가게 될 데이터, 컬럼명 : [데이터들]
    data = {'city': ['seoul','gyeong-gi'],
            'patient':[1580,1300],
            'today_patient' : [15,10],
            'date' :['2020-07-10','2020-07-10']},
    #인덱스 설정
    index = ['seoul','gyeong-gi'],
    #컬럼명 설정
    columns = ['patient','today_patient','date'])

#출력
print(corona_data)
```

```
#행 위주의 출력
seoul = corona_data.loc['seoul']
print(seoul)
print(seoul.index)
print(seoul.values)
```

나만의 데이터 만들기

```
scientists = pd.read_csv('scientists.csv')
scientists.head()

#브로드 캐스팅 : 데이터 프레임에 있는 모든 데이터에 대해 한번에 연산하는 것

#변수선택[행추출['컬럼지정'] > ['컬럼지정'].mean()]
print(scientists[scientists['Age'] > scientists['Age'].mean()])

#시리즈와 데이터프레임처리
born_datetime = pd.to_datetime(scientists['Born'], format = '%Y-%m-%d')
print(born_datetime)
died_datetime = pd.to_datetime(scientists['Died'], format = '%Y-%m-%d')
print(died_datetime)

# 변수['새로운 컬럼할당','새로운 컬럼할당'] = (시계열테이터, 시계열 데이터)
scientists['born_dt'], scientists['died_dt'] = (born_datetime, died_datetime)
print(scientists.head())

#새로운 컬럼에 할당 = 시계열 - 시계열
scientists['age_days_dt'] = (scientists['died_dt'] - scientists['born_dt'])
print(scientists)
```

시리즈와 데이터프레임 데이터 처리하기

3장 – 시각화(matplotlib)

기본 셋팅

```
import seaborn as sns

#tips데이터 불러오기 및 저장
#손님들이 지불한 tip 정보
tips = sns.load_dataset("tips")
print(tips.head())

#빈 그래프 그리기
fig = plt.figure()
#add_subplot(행 크기,  열크기, 들어갈 위치)
axes1 = fig.add_subplot(1,1,1)
```

히스토그램

```
# 히스토그램 분포도
#hist(변수['컬럼'], x축의 간격 10)
axes1.hist(tips['total_bill'], bins=10)
#상단의 이름
axes1.set_title('Histogram of total Bill')
#x축의 이름
axes1.set_xlabel('Frequency')
#y축의 이름
axes1.set_ylabel('Total Bill')

fig
```

시각화 처리하기

산점도 그래프

```
#기본 틀 생성
scatter_plot = plt.figure()
#그래프 격자 생성
axes1 = scatter_plot.add_subplot(1,1,1)

#scatter(변수[컬럼_1 : total_bill], 변수[컬럼_2:tip])
axes1.scatter(tips['total_bill'],tips['tip'])
axes1.set_title('전체 지불액 vs 팁') # 한글은 잘 적용되지 않는다.
axes1.set_xlabel('total bill')
axes1.set_ylabel('tip')
plt.show()
```

박스 그래프

```
#기본 틀 생성
boxplot=plt.figure()
#격자 생성
axes1 = boxplot.add_subplot(1,1,1)

#tips데이터 프레임에서 성별이 남자와 여자인 데이터에서 tips 열 데이터만 추출하여
#리스트에 담아 전달.
#데이터프레임[데이터 프레임내의 ['컬렴명'] == 'Female'][열 선택 추출]
axes1.boxplot([tips[tips['sex'] == 'Female']['tip'],
               tips[tips['sex'] == 'Male']['tip']] ,
              #라벨링 인잣값으로 성별을 구분하기 위한 이름 추가
               labels = ['Female', 'Male'])

axes1.set_xlabel('Sex')
axes1.set_ylabel('Tip')
axes1.set_title('Boxplot of Tips by Sex')

plt.show()
```

시각화 처리하기

다변량 그래프

```
#문자열 데이터 치환과정 male, female -> 남자_1, 여자_0
def recode_sex(sex):
    if sex == 'female':
        return 0
    else:
        return 1

#apply함수활용
#tips['sex_color']새로운 컬럼 생성 = tips의 ['sex']에.함수적용(자체함수)
tips['sex_color'] = tips['sex'].apply(recode_sex)
tips.head()#새로운 변수 확인
```

```
scatter_plot = plt.figure()
axes1 = scatter_plot.add_subplot(1,1,1)
axes1.scatter(
x = tips['total_bill'],
y = tips['tip'],
s = tips['size'] * 10, # s는 점의크기를 할당,  점의 크기를 인원수로 설정.
c = tips['sex_color'], # c는 색상을 할당,  남자,여자로 분리하여 할당
alpha = 0.5) #투명도

axes1.set_title('total_bill vs Tip colored by sex sized by size')
axes1.set_xlabel('total bill')
axes1.set_ylabel('tip')

plt.show()
```

시각화 처리하기

다변량 그래프

```
# 산점도
fig, ax = plt.subplots()
ax = tips.plot.hexbin(x ='total_bill', y='tip', ax=ax)

#히스토그램
fig, ax = plt.subplots()
ax = tips[['total_bill', 'tip']].plot.hist(alpha=0.5, bins = 20, ax = ax)
```

결측치 처리하기

```
import pandas as pd

#isnull을 통해 결측값 검사
print(pd.isnull((NaN)))

print(pd.isnull((nan)))

print(pd.isnull((NaN)))

visited= pd.read_csv('survey_visited.csv')
survey = pd.read_csv('survey_survey.csv')

print(visited)
print(survey)
```

```
#데이터 결합을 위한 merge()
vs = visited.merge(survey, left_on='ident', right_on='taken')
print(vs)
print(vs.count()) #열 별로 결측값 출력

#결측값 삭제하기
vs_dropna = vs.dropna()
print(vs_dropna)

#결측값 0 처리
vs_fillna = vs.fillna(0)
print(vs_fillna)
```

4장 – sqlite with python

sqlite – db연결 및 쿼리 수행

```
import sqlite3

#import os
#print(os.getcwd())

#db와 연결
conn = sqlite3.connect('world.db')
#연결 확인
type(conn)

#실행
cursor = conn.cursor()
#쿼리 작성 및 테이블 생성 (삼성 주식관련 내용)
cursor.execute("CREATE TABLE samsung(Date text, Open int, High i
nt, Low int, Closing int, Volumn int)")
#쿼리 작성 행 삽입
cursor.execute("INSERT INTO kakao VALUES('20.07.08', 55800, 5590
0, 53400, 55000, 357)")
cursor.execute("INSERT INTO kakao VALUES('20.07.07', 55800, 5590
0, 53400, 55000, 357)")

#저장 및 닫기
conn.commit()
conn.close()
```

팀 미션

Corona-19 관련 혹은 수업중 활용했던 데이터를 활용하여

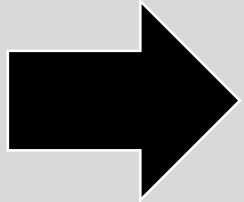
데이터에 숨겨진 인사이트를 발굴하고 분석결과를 도출해 내는 것 입니다.

평가 지표

기획과 분석 과정 (1~5점)

합리적인 결론 도출 (1~5점)

적절한 발표 자료 (1~5점)



모든 인원들이 평가
총 150점 만점
(135점 + 15점)

사용 가능 프로그램 : 파이썬, 엑셀, SQL

Team

강지웅
김대호A
고병표

김성민
김지원
배영섭

고정환
정해앙
김선일

정진우
한승원
이동형

지영석
송현달
유병욱

주하영
이원호
이이삭

임준형
장하영
김나영

김시아
강지수
홍지형

김희아
신혜수
최우혁

최서윤
이상민
이민기