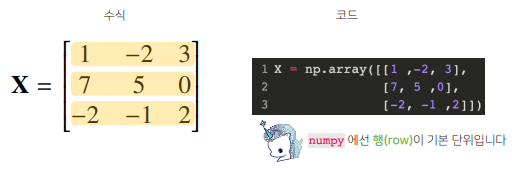
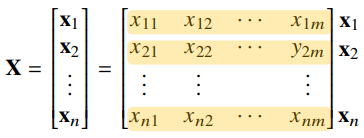
1. 행렬

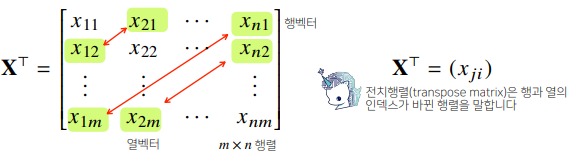
행렬은 벡터를 원소로 가지는 2차원 배열이다. 행과 열이라는 인덱스를 가진다.



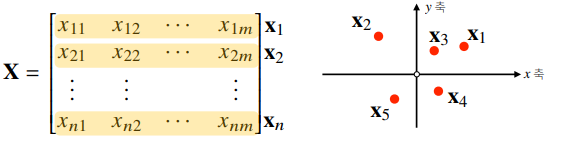
(n개의 행, m개의 열)

 행렬을 xij와 같은 표기로 표현할 수도 있다.

전치행렬을 행과 열의 인덱스가 바뀐 행렬을 의미한다. 즉, n\*m행렬이 m\*n행렬로 바뀐다. 벡터에 적용하면 행 벡터는 열 벡터가 되고 열 벡터는 행 벡터가 된다.

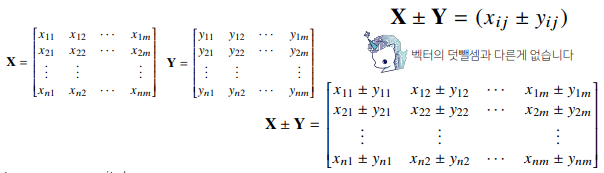


벡터가 공간에서 한 점을 의미했다면 행렬은 여러 점들을 의미한다. 행렬의 행 벡터 xi는 i 번째 데이터를 의미한다. xij는 i 번째 데이터의 j 번째 변수 값을 의미한다.

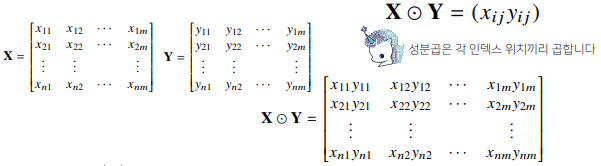


2. 행렬의 연산

행렬은 벡터를 원소로 가지는 2차원 배열이다. 따라서 벡터와 동일하게 행렬끼리 같은 모양을 가지면 **덧셈, 뺄셈을 계산**할 수 있다.

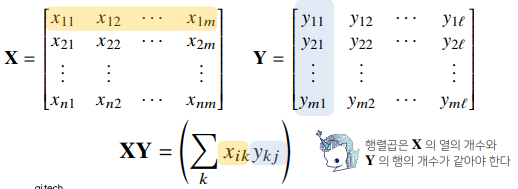


**성분곱도 벡터와 동일하다. 스칼라곱도 마찬가지다.**

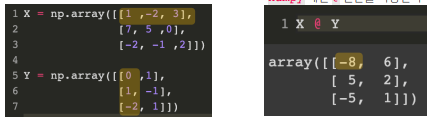


3. 행렬의 곱셈

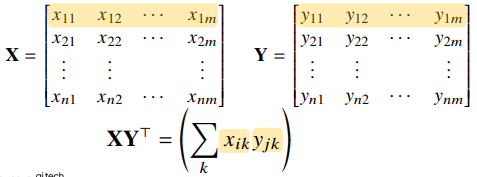
행렬 곱셈(matrix multiplication)은 i번째 행벡터와 j번째 열벡터 사이의 내적 성분으로 가지는 행렬을 계산한다. **x의 행 길이는 y의 열 길이와 같아야한다.**



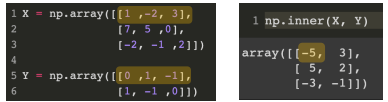
아래에서 -8이라는 값은 1\*0 + (-2)\*1 + 3\*(-2)로 계산된 것이다.



넘파이의 내적 함수인 **np.inner()를 사용하면 i번째 행벡터와 j번째 행벡터 사이의 내적을 성분으로 가지는 행렬을 계산한다**. 즉, x의 행 길이와 y의 행 길이가 같아야한다.

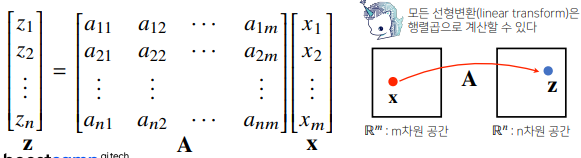


-5는 1\*0 + (-2)\*1 + 3\*(-1)의 결과이다.



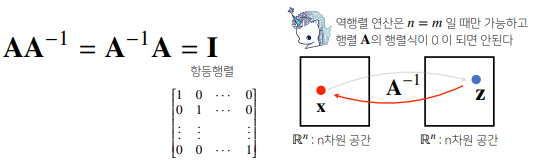
4. 행렬\_2

행렬은 벡터공간에서 사용되는 연산자(operator)로 이해할 수 있다. 행렬곱을 사용하면 **벡터를 다른 차원의 공간으로 보낼 수 있다**. 이런 특징을 이용해 **패턴을 추출**할 수 있고 **데이터를 압축**할 수도 있다.

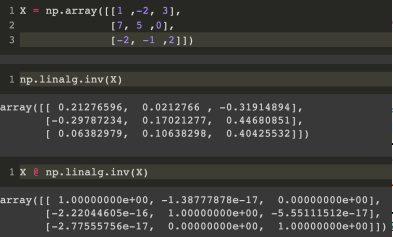


5. 역행렬

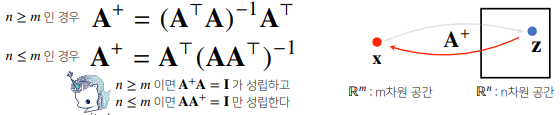
어떤 행렬 A의 연산을 거꾸로 되돌리는 행렬을 **역행렬(inverse matrix)**이라고 부르고 A^(-1)이라 표기한다. **역행렬은 행과 열 숫자가 같고 행렬식이 0이 아닌 경우에만 계산할 수 있다**.



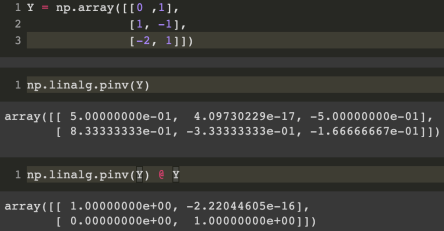
numpy.linalg.inv()로 구할 수 있다.



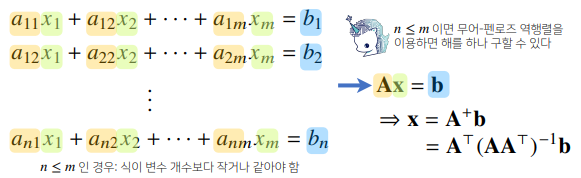
행과 열의 길이가 달라 역행렬을 계산할 수 없다면 **유사역행렬(pseudo-inverse) 또는 무어-펜로즈(Moore-Penrose) 역행렬** A^+을 이용할 수 있다.



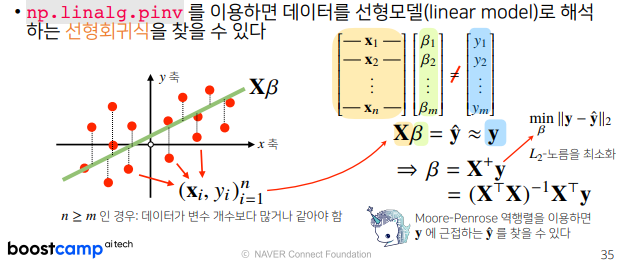
numpy.linalg.pinv()로 구할 수 있다.



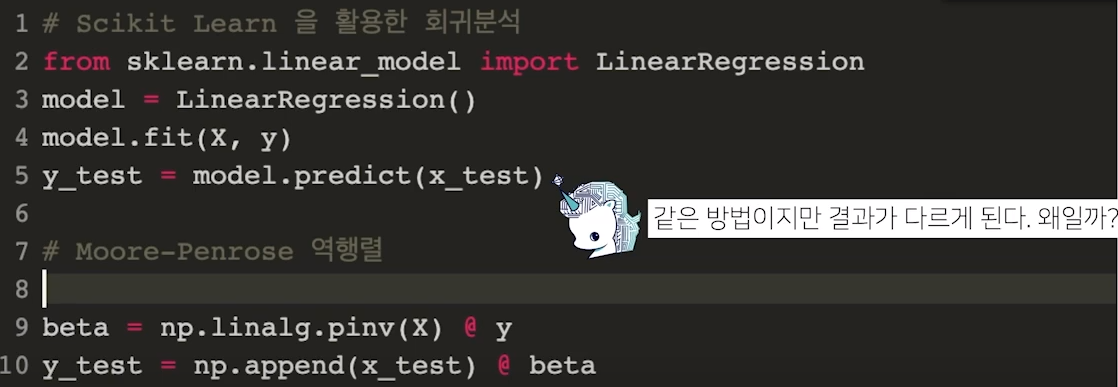
유사역행렬을 사용해 연립 방정식을 풀 수 있다.



유사역행렬을 이용해 선형 회귀 분석도 수행할 수 있다.



코드는 아래와 같이 쓸 수 있다.



두 방식의 결과값 차이는 y절편 때문에 발생한다. sklearn은 y절편을 자동으로 추가해주지만 우리는 직접 추가해야한다. 완성 코드는 아래와 같다.

