강원지역혁신플랫폼

기계학습

Machine Learning

선형 변환 변수

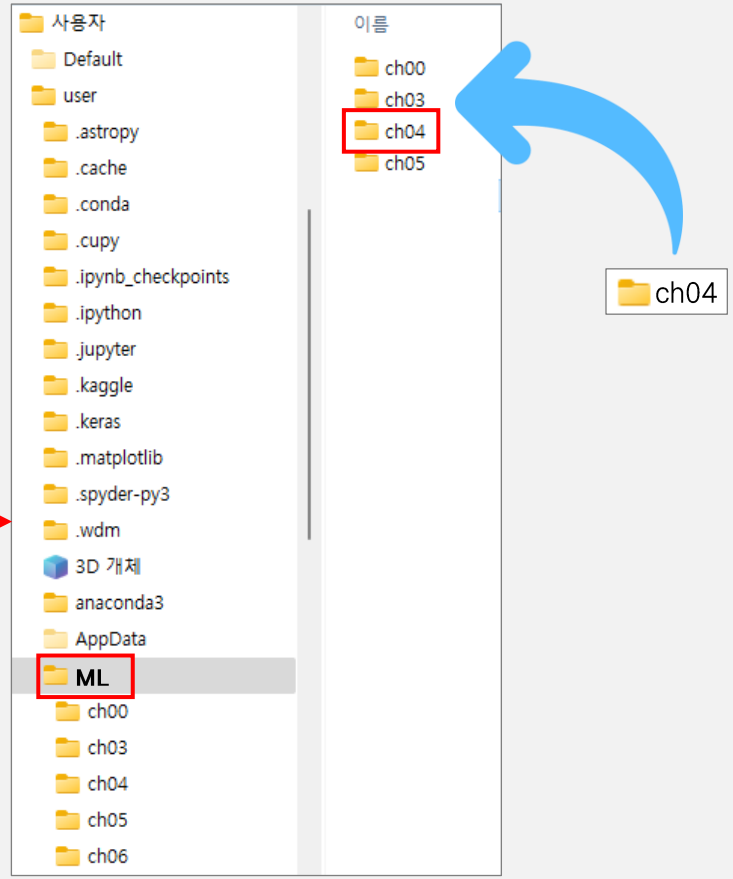
학습목표

- 선형 변환 변수의 개념을 이해하고 구현할 수 있습니다.

01 | 4주차 실습코드 복사하기

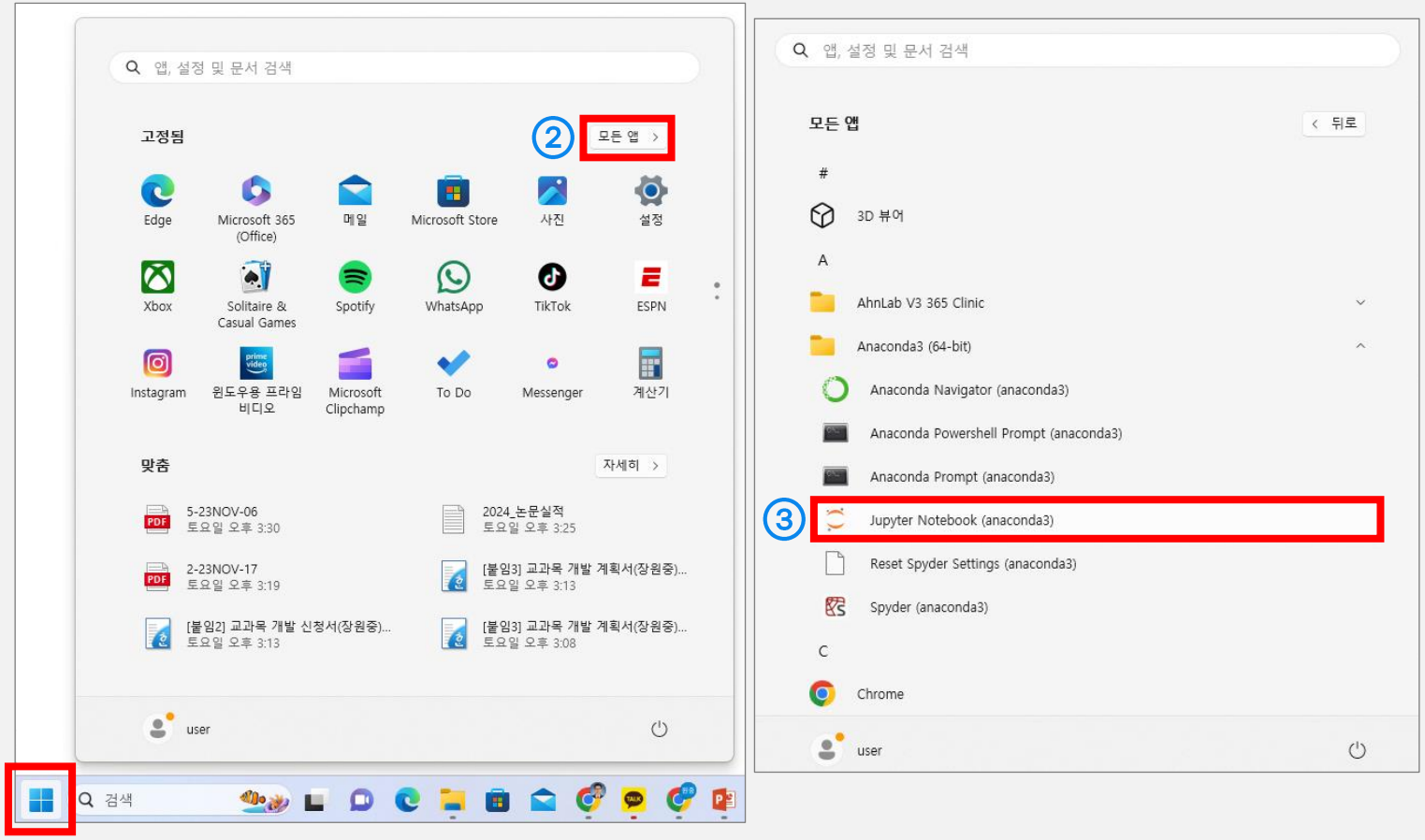
(권장) 아래와 같은 경로에 실행 소스가 존재하면 환경 구축 완료

- 4주차 실습코드 다운로드 → 압축해제 → ch04 폴더를 ML 하위 폴더로 복사
- c:₩Users₩user>ML> 컴퓨터이름 또는 사용자계정



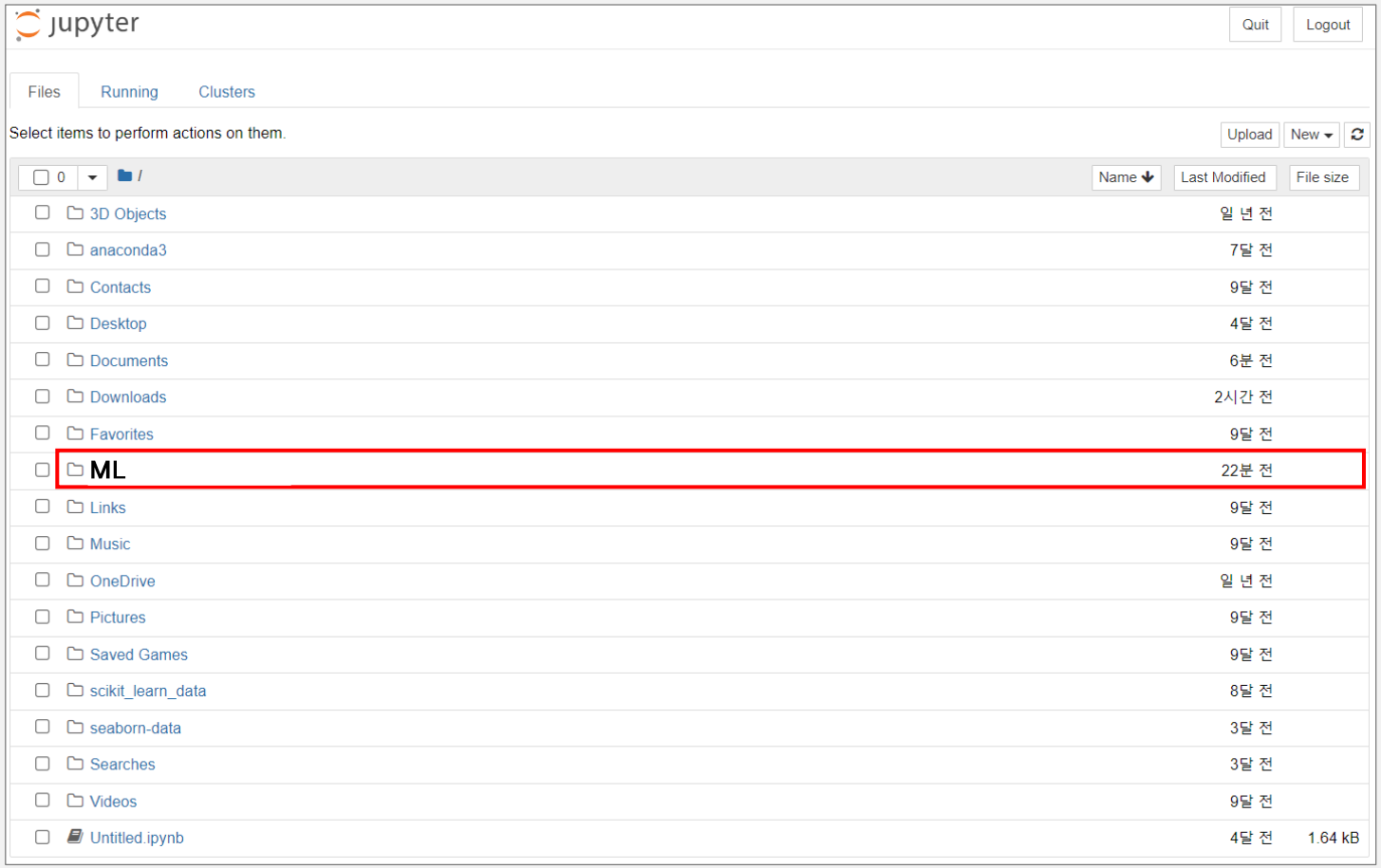
02 | Jupyter Notebook 실행하기

✦ ①시작 메뉴 클릭 > ②모든 앱 버튼 클릭 > ③Anaconda3(64-bit)
> “Jupyter Notebook (anaconda)” 메뉴 클릭하기



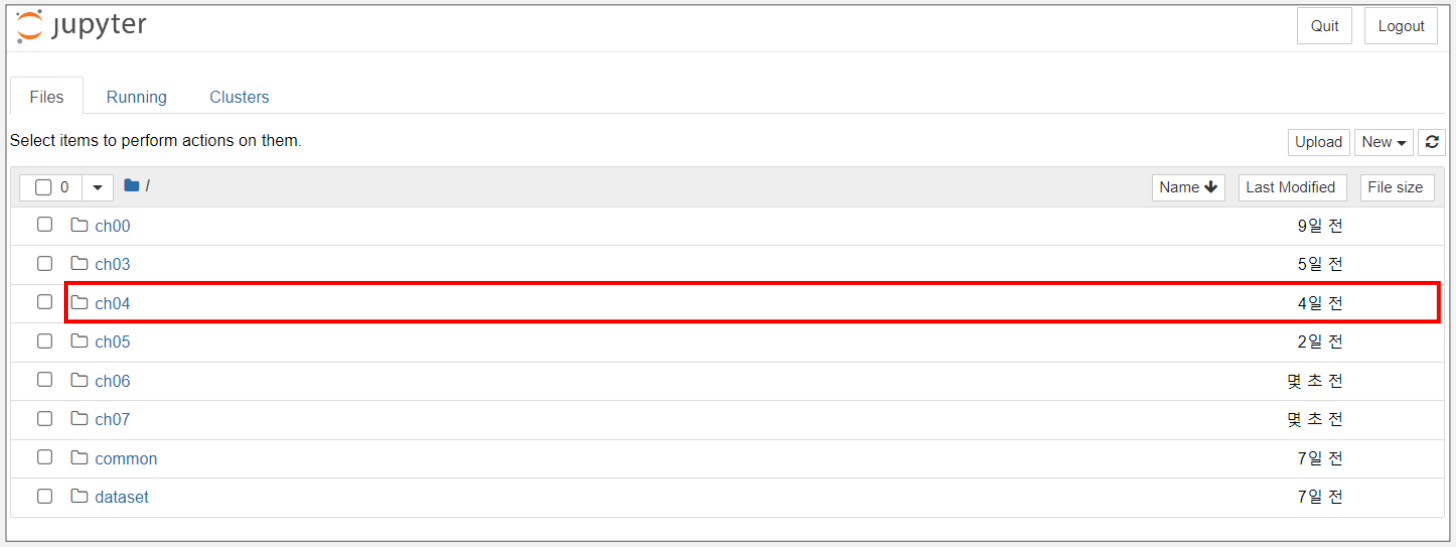
03 | ML 폴더

ML 폴더를 클릭하기



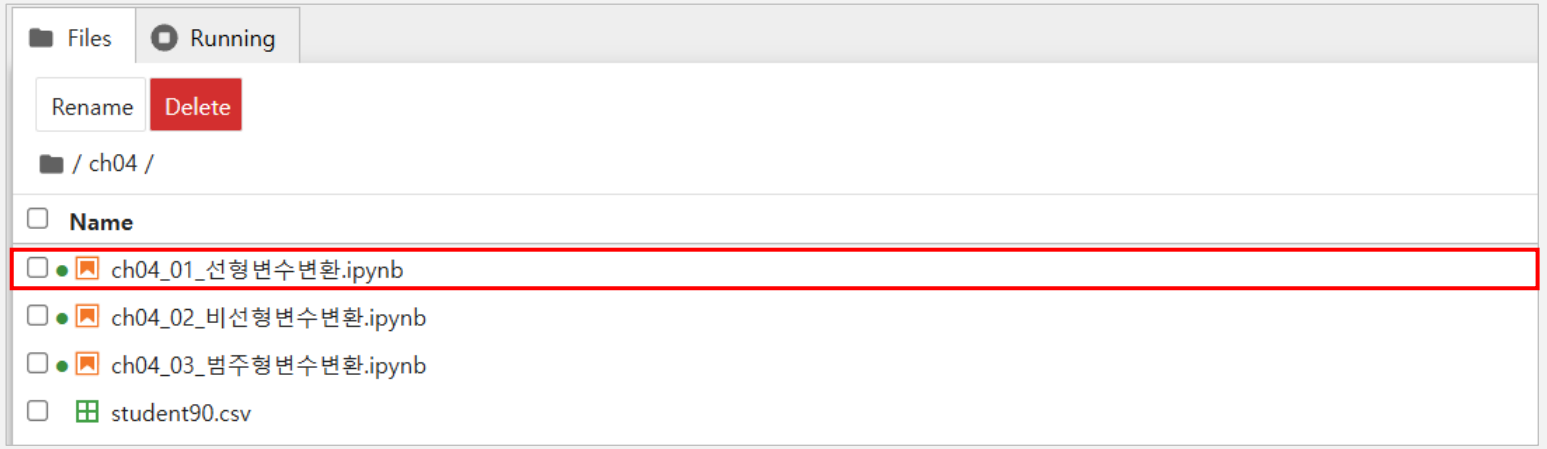
04 | ch04 폴더

ch04 폴더 클릭하기



05 | ch04_01_선형변수변환.ipynb

✦ ch04_01_선형변수변환.ipynb 파일 클릭하기



06 | 변수 변환

변수 변환(Feature Scaling)

- 변수 변환이란, 변수(feature)의 스케일(scale)을 바꾸는 변수 정규화를 의미함
- 입력 변수들의 스케일이 서로 크게 다른 상황에서 유용함
  - 어떤 수치형 변수들은 무한히 증가하기 때문에 평활 함수 모델은 입력의 스케일에 영향을 받음
    - 예를 들어 선형 회귀, 로지스틱 회귀 등의 모델은 입력의 스케일에 영향을 받음
  - 반면, 트리 기반 모델은 입력의 스케일에 그다지 신경 쓰지 않아도 됨
  - 모델이 입력 변수의 스케일에 민감하다면 변수 변환이 도움이 될 수 있음

06 | 변수 변환

- 일반적으로 **변수 변환**은 **각 변수**에 대해 **개별적**으로 **수행**됨
  - **변수의 유형**에 따라 다음과 같은 **변수 변환 방법**들이 있음
    1. **수치형 변수** 변환(괄호는 Scikit Learn 모듈의 함수명)
       - 선형 변환 : Scaler
         - 최소최대 스케일링(MinMaxScaler), 표준화(StandardScaler),
           로버스트 스케일링(RobustScaler), 균등분포/RankGauss(QuantileTransformer)
       - 비선형 변환 : 함수 변환
         - 로그변환, 거듭제곱변환(PowerTransformer – Boxcox, YeoJohnson),
           정규화(Normalizer – L1, L2, Max), 루트변환, 역수변환, 지수변환
       - 기타 변환
         - 구간분할(=이산화, binning), 순위 변환

06 | 변수 변환

2 범주형 변수 변환

- 원핫인코딩 (One-hot encoding)
- 더미코딩 (Dummy coding)
- 이펙트코딩 (Effect coding)
- 숫자로 표현된 범주형 특성
- 레이블인코딩 (Label encoding)
- 특징 해싱 (Feature Hashing)
- 빈도인코딩 (Frequency encoding)

06 | 변수 변환

변수 변환과 스케일 백(표준화한 값을 원래의 값 범위로 복원)

- 사이킷런(Scikir-Learn)의 preprocessing 모듈의 **표준화 함수** (scale(), robust_scale(), minmax_scale(), maxabs_scale() 등) 들을 이용해 **표준화**를 할 수 있음

- 또한, 사이킷런(Scikir-Learn)의 preprocessing 모듈의 StandardScaler, MinMaxScaler, MaxAbsScaler 등 **클래스**를 이용해 **표준화**를 할 수도 있음
  - 이들 **클래스**를 이용하면 **표준화 후** 표준화한 값을 **원래의 값 범위**로 **되돌릴 수 있음**
  - 이들 클래스의 표준화 방법은 **표준화 함수**와 **동일함**

06 | 변수 변환

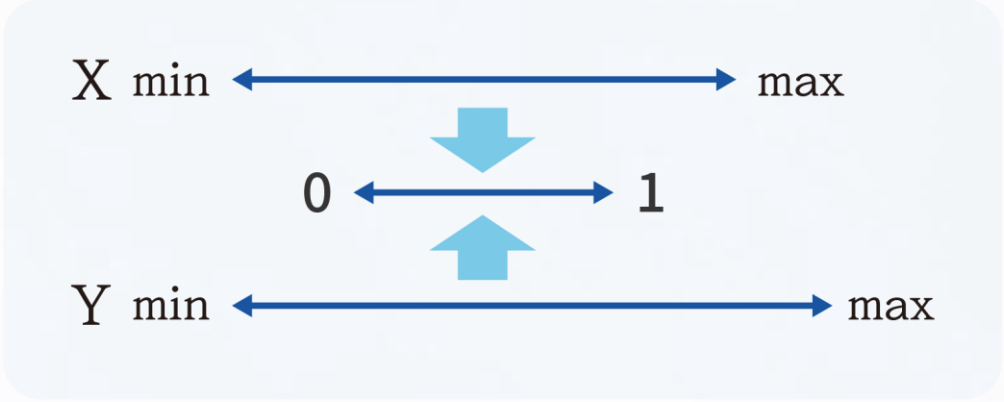
사이킷런(Scikir-Learn)의 preprocessing 모듈의 표준화 클래스 (StandardScaler, MinMaxScaler, MaxAbsScaler 등)는 아래의 함수를 이용함

메서드	설명
fit(X[, y])	스케일링에 사용될 평균 및 표준편차를 계산한다.
transform(X[, y, copy])	표준화를 수행해서 데이터를 변환한다.
fit_transform(X[, y])	평균과 표준편차를 계산하고, 표준화를 수행해서 데이터를 변환한다.
get_params([deep])	파라미터를 가져온다.
inverse_transform(X[, copy])	데이터를 원래 표현으로 스케일 백 한다.
partial_fit(X[, y])	스케일링을 위해 X에 대한 평균 및 표준편차를 계산한다. - RobustScaler 클래스에는 이 메서드가 없다.
set_params(**params)	매개변수를 설정한다.

07 | 선형 변수 변환: MinMaxScaler

최소최대 스케일링(Min-Max Scaling) with MinMaxScaler

- 최소최대 스케일링은 모든 특성이 정확하게 0~1사이에 위치하도록 데이터를 정규화함
  - 즉, 개별 데이터의 크기를 모두 똑같은 단위(scale)로 변경하는 것임
    - 예를 들어, 2차원 데이터 집합일 경우 모든 데이터가 x축의 0~1, y축의 0~1 사이의 사각 영역에 담기게 됨



최소 최대 '0~1' 범위 표준화

07 | 선형 변수 변환: MinMaxScaler

최소최대 정규화 공식은 다음과 같다.

- 새로운 데이터 x'는 원래 값 x_i에서 변수 x의 최솟값을 뺀 값을 변수 x의 최대값과 최솟값의 차이로 나눈 값으로 변환한 것임

$$x' = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

다음의 함수를 이용해 최소최대 정규화를 구현할 수 있음

- sklearn.preprocessing.MinMaxScaler()
- sklearn.preprocessing.minmax_scale()

07 | 선형 변수 변환: MinMaxScaler

- 최소최대 스케일링 변환 후의 평균이 정확히 0.5가 되지 않고 이상치의 악영향을 받기 쉽다는 단점이 있음
  - 그래서 표준정규분포 데이터 표준화(standardization) 방법이 더 자주 쓰임
  - 하지만, 이미지 데이터의 픽셀 값과 같이 처음부터 0~255로 범위가 정해진 변수는 최소최대 스케일링을 이용하는 것이 더 자연스러울 수 있음

07 | 선형 변수 변환: MinMaxScaler

2차원 배열 데이터 세트

- 다음은 2차원 배열의 훈련용 데이터 세트를 생성하는 코드이다.
  - 임의로 생성한 2차원 배열 데이터 세트로 최소최대 스케일링 적용
  - 아래와 같이 형상이 (3,3)인 2차원 배열 데이터 세트가 생성된 것을 볼 수 있음

```
X = np.array([[ 10, -10, 1], [ 5, 0, 2], [ 0, 10, 3]])
X
```

```
array([[ 10, -10,   1],
       [  5,   0,   2],
       [  0,  10,   3]])
```

07 | 선형 변수 변환: MinMaxScaler

다음은 훈련용 데이터 세트에 최소최대 스케일링을 적용하는 코드이다.

✦ 아래와 같이 **최소최대 정규화 공식**을 적용하여 **정규화**함

```
X_MinMax = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_MinMax
```

```
array([[1. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [0. , 1. , 1. ]])
```

✦ 아래 그림과 같이 변경 전과 **변경 후**를 **비교**해보면 **0~1 사이 값**으로 **변환된 것**을 알 수 있음

```
array([[ 10, -10,   1],
       [  5,   0,   2],
       [  0,  10,   3]])
```

변경 전


```
array([[1. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [0. , 1. , 1. ]])
```

변경 후

07 | 선형 변수 변환: MinMaxScaler

- 다음은 훈련용 데이터 세트로 학습을 수행하는 코드이다.
  - 여기서는 MinMaxScaler() 클래스를 이용함
    - fit_transform() 함수는 훈련 데이터를 학습시킴

```
MinMax_scaler = MinMaxScaler()                          # 최소최대 모델 생성
X_MinMax_train = MinMax_scaler.fit_transform(X)  # 훈련 데이터 학습
X_MinMax_train
```

```
array([[1. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [0. , 1. , 1. ]])
```

07 | 선형 변수 변환: MinMaxScaler

- 다음은 시험용 데이터 세트를 생성하고 학습된 모델에 적용시켜 정규화를 수행하는 코드이다.
  - 즉, 훈련 데이터에서 학습하고 학습된 모델을 시험용 데이터에 적용시킴
    - 아래 코드에서 새로운 시험용 데이터 세트를 변수 X_new에 생성함
    - 그리고, 학습된 모델에 적용시켜 정규화를 수행함

```
X_new = np.array([[9., -10., 1.], [5., -5., 3.], [1., 0., 5.]])
X_MinMax_new = MinMax_scaler.transform(X_new)  # 학습된 모델에 시험 데이터를 적용하여 변환시킴
X_MinMax_new
```

```
array([[0.9 , 0.  , 0.  ],
       [0.5 , 0.25, 1.  ],
       [0.1 , 0.5 , 2.  ]])
```

07 | 선형 변수 변환: MinMaxScaler

- 다음은 훈련용 데이터 세트로 minmax_scale() 함수에 적용하여 정규화를 수행하는 코드이다.
  - minmax_scale() 함수는 최소최대 정규화를 수행함

```
from sklearn.preprocessing import minmax_scale

X_MinMax_scaled = minmax_scale(X, axis=0, copy=True)
X_MinMax_scaled
```

```
array([[1. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [0. , 1. , 1. ]])
```

  - 아래 그림과 같이 MinMaxScaler() 클래스와 minmax_scale() 함수의 결과를 비교해보면 동일한 결과인 것을 알 수 있음

```
array([[1. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [0. , 1. , 1. ]])
```

MinMaxScaler()

```
array([[1. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [0. , 1. , 1. ]])
```

Minmax_scale()

07 | 선형 변수 변환: MinMaxScaler

아이리스 (iris) 데이터 세트

다음은 아이리스 데이터 세트를 읽어오는 코드이다.

- 아래와 같이 아이리스 데이터 세트는 150개의 관측치와 5개의 속성으로 구성된 것을 알 수 있음

```
iris = sns.load_dataset("iris")   # iris 데이터셋 읽기
iris
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows x 5 columns

07 | 선형 변수 변환: MinMaxScaler

- 다음은 아이리스 데이터 세트를 훈련 데이터와 시험 데이터로 분리하는 코드이다.
  - 여기서는 species 층 구분 변수로 7:3 비율로 데이터를 분리함

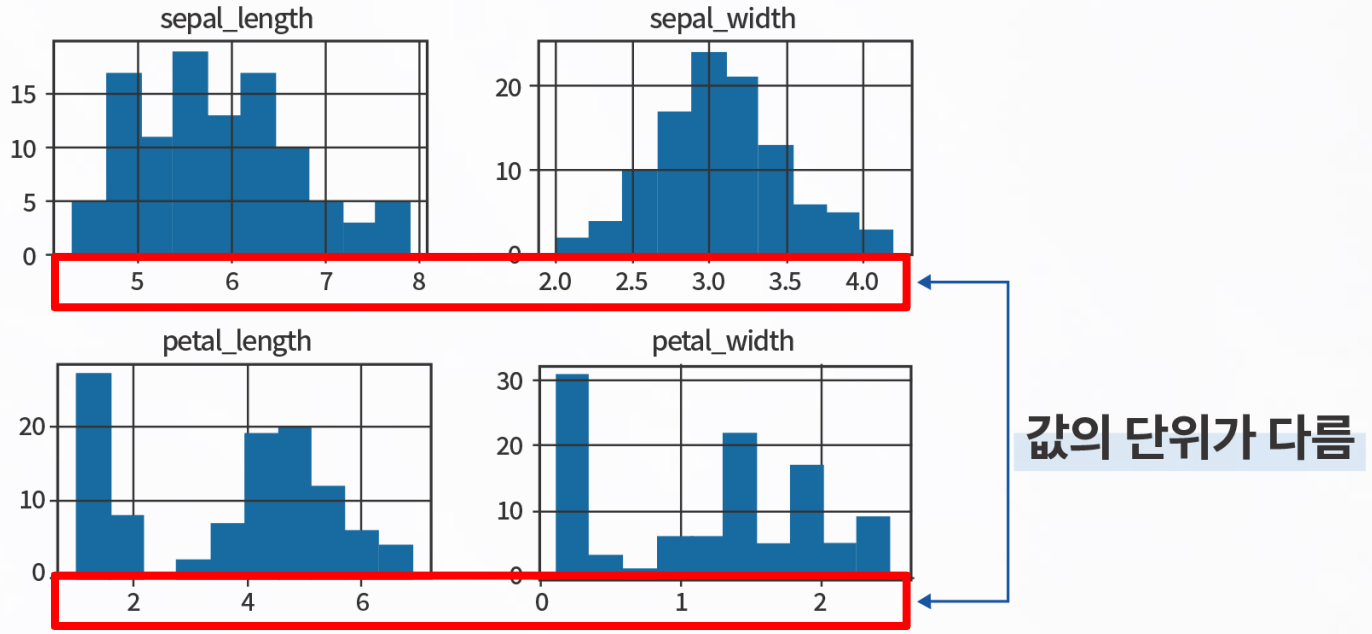
```
# 데스트 데이터셋 비율 = 30%
x_train, x_test, y_train, y_test = train_test_split(iris.iloc[:,0:4],
                                   iris.species,
                                   test_size=0.3,
                                   shuffle=True,              # shuffle=True : 무작위 추출
                                   stratify=iris['species'],  # stratify : 층 구분 변수이름
                                   random_state=1234)
print(x_train.shape, y_train.shape)  # (105, 4) (105,)
print(x_test.shape, y_test.shape)    # 45, 4) (45,)
```

07 | 선형 변수 변환: MinMaxScaler

다음은 아이리스 훈련용 데이터 세트로 히스토그램을 그리는 코드이다.

- 히스토그램은 아래 그림과 같고 변수마다 값의 크기와 분포가 다른 것을 볼 수 있음

```
pd.DataFrame(x_train, columns=['sepal_length','sepal_width','petal_length','petal_width']).hist()
plt.subplots_adjust(hspace=1)
plt.show()
```



07 | 선형 변수 변환: MinMaxScaler

- 다음은 아이리스 훈련용 데이터 세트로 학습을 수행하는 코드이다.
  - 여기서는 MinMaxScaler() 클래스를 이용함
    - fit_transform() 함수는 훈련 데이터를 학습시킴

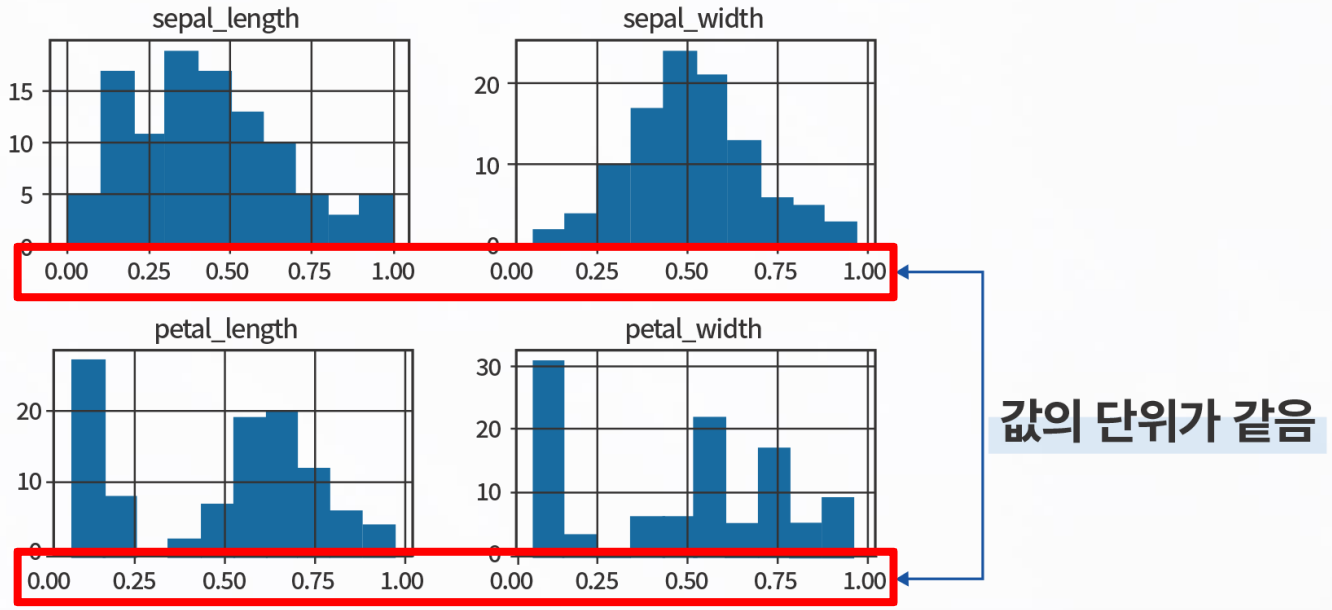
```
MinMax_scaler = MinMaxScaler()                                  # 최소최대 모델 생성
iris_MinMax_train = MinMax_scaler.fit_transform(x_train)        # 훈련 데이터 학습
iris_MinMax_train[:10, :]                                       # 10개 행을 출력
```

```
array([[0.94444444, 0.45454545, 0.86440678, 0.91666667],
       [0.16666667, 0.72727273, 0.06779661, 0.        ],
       [0.55555556, 0.36363636, 0.69491525, 0.58333333],
       [0.72222222, 0.5       , 0.74576271, 0.83333333],
       [0.52777778, 0.63636364, 0.74576271, 0.91666667],
       [0.33333333, 0.18181818, 0.47457627, 0.41666667],
       [0.55555556, 0.40909091, 0.77966102, 0.70833333],
       [0.41666667, 0.31818182, 0.49152542, 0.45833333],
       [0.72222222, 0.5       , 0.66101695, 0.58333333],
       [0.22222222, 0.77272727, 0.08474576, 0.125     ]])
```

07 | 선형 변수 변환: MinMaxScaler

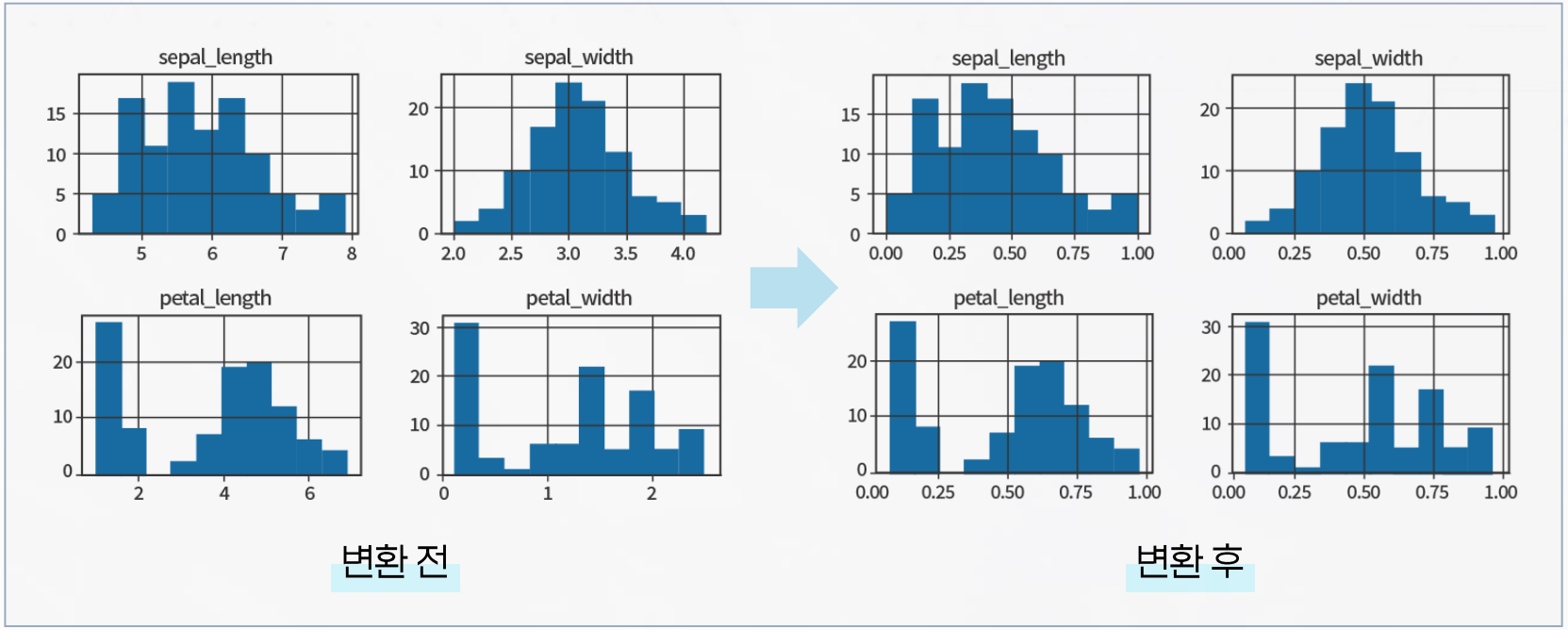
- 다음은 정규화된 아이리스 **훈련용 데이터** 세트로 **히스토그램**을 그리는 코드이다.
  - 히스토그램은 아래 그림과 같고 **변수**마다 **값**의 **크기가 동일한 것**을 볼 수 있음

```
pd.DataFrame(iris_MinMax_train,
columns=['sepal_length','sepal_width','petal_length','petal_width']).hist()
plt.subplots_adjust(hspace=1)
plt.show()
```



07 | 선형 변수 변환: MinMaxScaler

- 다음은 아이리스 훈련용 데이터 세트의 정규화 이전과 이후의 히스토그램을 비교해보자.
  - 비교 결과 그래프에서 데이터 분포의 모양은 동일한 것을 볼 수 있음
    - 여기서 중요한 점은 데이터 값의 크기가 동일하게 0~1사이 값으로 변화된 것을 알 수 있음



07 | 선형 변수 변환: MinMaxScaler

- 다음은 아이리스 훈련용 데이터 세트 정규화 이전 값과 이후 값을 비교하는 코드이다.
  - 아래와 같이 정규화된 경우 0~1 사이 값으로 변경된 것을 볼 수 있음

```
print(x_train[:4])
print('-' * 65)
print(pd.DataFrame(iris_MinMax_train[:4],
columns=['sepal_length','sepal_width','petal_length','petal_width']))
```

	sepal_length	sepal_width	petal_length	petal_width
135	7.7	3.0	6.1	2.3
37	4.9	3.6	1.4	0.1
133	6.3	2.8	5.1	1.5
139	6.9	3.1	5.4	2.1

정규화 이전

	sepal_length	sepal_width	petal_length	petal_width
0	0.944444	0.454545	0.864407	0.916667
1	0.166667	0.727273	0.067797	0.000000
2	0.555556	0.363636	0.694915	0.583333
3	0.722222	0.500000	0.745763	0.833333

정규화 이후

07 | 선형 변수 변환: MinMaxScaler

- 다음은 아이리스 훈련용 데이터 세트에서 변경 전과 변경 후의 최소, 최대값을 확인하는 코드이다.
  - 아래와 같이 정규화된 이후 모든 값이 0~1 사이에 있는 것을 알 수 있음

```
print(x_train.min(axis=0), "\n", x_train.max(axis=0), "\n\n",
    iris_MinMax_train.min(axis=0), "\n", iris_MinMax_train.max(axis=0))
```

```
sepal_length    4.3
sepal_width     2.0
petal_length    1.0
petal_width     0.1
dtype: float64
 sepal_length    7.9
sepal_width     4.2
petal_length    6.9
petal_width     2.5
dtype: float64

 [0. 0. 0. 0.]
 [1. 1. 1. 1.]
```

07 | 선형 변수 변환: MinMaxScaler

- 다음은 학습된 모델로 아이리스 시험용 데이터 세트를 적용하여 정규화를 수행하는 코드이다.
  - 여기서 중요한 것은 데이터의 학습은 훈련 데이터 세트로 수행함 (fit_transform() 함수를 사용함)
    - 시험 데이터 세트는 학습된 모델로 정규화를 수행함 (transform() 함수를 사용함)

```
# 학습된 모델에 시험 데이터를 적용하여 변환시킴
iris_MinMax_test = MinMax_scaler.transform(x_test)
iris_MinMax_test[:10, :]
```

```
array([[0.94444444, 0.45454545, 0.86440678, 0.91666667],
       [0.16666667, 0.72727273, 0.06779661, 0.        ],
       [0.55555556, 0.36363636, 0.69491525, 0.58333333],
       [0.72222222, 0.5       , 0.74576271, 0.83333333],
       [0.52777778, 0.63636364, 0.74576271, 0.91666667],
       [0.33333333, 0.18181818, 0.47457627, 0.41666667],
       [0.55555556, 0.40909091, 0.77966102, 0.70833333],
       [0.41666667, 0.31818182, 0.49152542, 0.45833333],
       [0.72222222, 0.5       , 0.66101695, 0.58333333],
       [0.22222222, 0.77272727, 0.08474576, 0.125     ]])
```

07 | 선형 변수 변환: MinMaxScaler

- 다음은 아이리스 시험용 데이터 세트 정규화 이전 값과 이후 값을 비교하는 코드이다.
  - 아래와 같이 정규화된 경우 0~1 사이 값으로 변경된 것을 볼 수 있음

```
print(x_test[:4])
print('-' * 65)
print(pd.DataFrame(iris_MinMax_test[:4],
columns=['sepal_length','sepal_width','petal_length','petal_width']))
```

	sepal_length	sepal_width	petal_length	petal_width
145	6.7	3.0	5.2	2.3
112	6.8	3.0	5.5	2.1
136	6.3	3.4	5.6	2.4
38	4.4	3.0	1.3	0.2

정규화 이전

	sepal_length	sepal_width	petal_length	petal_width
0	0.666667	0.454545	0.711864	0.916667
1	0.694444	0.454545	0.762712	0.833333
2	0.555556	0.636364	0.779661	0.958333
3	0.027778	0.454545	0.050847	0.041667

정규화 이후

07 | 선형 변수 변환: MinMaxScaler

다음은 아이리스 시험용 데이터 세트에서 변경 전과 변경 후의 최소, 최대값을 확인하는 코드이다.

- 아래와 같이 정규화된 이후 모든 값이 0~1 사이에 있는 것을 알 수 있음

```
print(x_test.min(axis=0), "\n", x_test.max(axis=0), "\n\n",
    iris_MinMax_test.min(axis=0), "\n", iris_MinMax_test.max(axis=0))
```

```
sepal_length    4.4
sepal_width     2.2
petal_length    1.3
petal_width     0.1
dtype: float64
 sepal_length    7.7
sepal_width     4.4
petal_length    6.7
petal_width     2.5
dtype: float64

 [0.02777778 0.09090909 0.05084746 0.        ]
 [0.94444444 1.09090909 0.96610169 1.        ]
```

각 변수의 최소, 최대값이 0~1 사이

07 | 선형 변수 변환: MinMaxScaler

다음은 정규화된 훈련 데이터를 스케일 백 (정규화된 데이터를 원래의 값으로 변환)을 수행하는 코드이다.

- 아래와 같이 정규화된 훈련 데이터를 원래의 값을 잘 변환된 것을 볼 수 있음

```
real_train = MinMax_scaler.inverse_transform(iris_MinMax_train)
print(real_train[:10, :])
print("-" * 60)
print(x_train[:10])
```

```
[[7.7 3.  6.1 2.3]
 [4.9 3.6 1.4 0.1]
 [6.3 2.8 5.1 1.5]
 [6.9 3.1 5.4 2.1]
 [6.2 3.4 5.4 2.3]
 [5.5 2.4 3.8 1.1]
 [6.3 2.9 5.6 1.8]
 [5.8 2.7 3.9 1.2]
 [6.9 3.1 4.9 1.5]
 [5.1 3.7 1.5 0.4]]
```

정규화된 훈련 데이터를 원래의 값으로 변환

	sepal_length	sepal_width	petal_length	petal_width
135	7.7	3.0	6.1	2.3
37	4.9	3.6	1.4	0.1
133	6.3	2.8	5.1	1.5
139	6.9	3.1	5.4	2.1
148	6.2	3.4	5.4	2.3
80	5.5	2.4	3.8	1.1
103	6.3	2.9	5.6	1.8
82	5.8	2.7	3.9	1.2
52	6.9	3.1	4.9	1.5
21	5.1	3.7	1.5	0.4

원래의 값

08 | 선형 변수 변환: StandardScaler

표준화(Standardization) with StandardScaler

- 표준정규분포를 이용한 데이터 표준화(standardization): StandardScaler()
- 표준화는 분산 스케일링(Variance scaling)이라고 불리기도 함
  - 곱셈과 덧셈만으로 변환하는 선형변환을 통해 각 특성의 평균을 0, 분산을 1로 변경하여 모든 특성이 같은 크기를 가지게 함
  - 그러나, 이 방법은 특성은 최소값과 최대값 크기를 제한하지는 않음

08 | 선형 변수 변환: StandardScaler

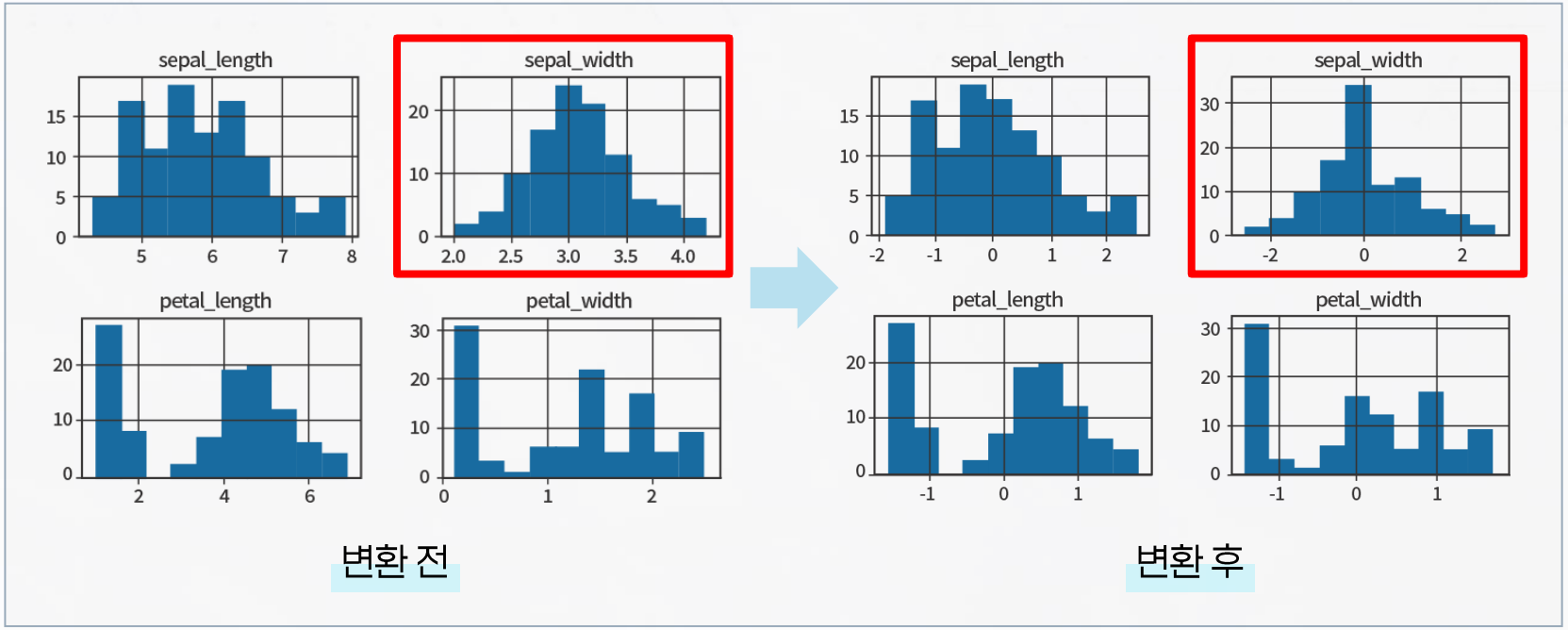
표준화 공식은 다음과 같다.

$$x' = \frac{x_i - mean(x)}{STDEV(x)}$$

- 새로운 데이터 x' 는 원래 값에서 변수 x의 평균을 뺀 값을 변수 x의 표준편차(STDEV, Standard Deviation)로 나눈 값으로 변환한 것임
- 이 값을 표준점수 혹은 Z-점수(Z-Score)라고 함

08 | 선형 변수 변환: StandardScaler

- 다음은 아이리스 훈련용 데이터 세트의 정규화 이전과 이후의 히스토그램을 비교해보자.
  - 비교 결과 그래프의 모양이 조금 달라진 것을 볼 수 있음
    - 여기서 중요한 점은 데이터 값의 크기가 평균 0, 분산 1로 변화된 것을 알 수 있음



08 | 선형 변수 변환: StandardScaler

다음은 아이리스 시험용 데이터 세트에서 변경 전과 변경 후의 최소, 최대값을 확인하는 코드이다.

- 아래와 같이 정규화된 이후 각 변수 값의 최소와 최대값이 다른 것을 알 수 있음

```
print(x_test.min(axis=0), "\n", x_test.max(axis=0), "\n\n",
iris_stdS_test.min(axis=0), "\n", iris_stdS_test.max(axis=0))
```

```
sepal_length    4.4
sepal_width     2.2
petal_length    1.3
petal_width     0.1
dtype: float64
 sepal_length    7.7
sepal_width     4.4
petal_length    6.7
petal_width     2.5
dtype: float64

 [-1.75271225 -2.02942225 -1.38757947 -1.42746349]
 [2.26924003 3.16787863 1.70259162 1.75872606]
```

각 변수의 최소, 최대값이 다름

09 | 선형 변수 변환: RobustScaler

중앙값과 IQR로 표현되는 Robust Scaling

- 중앙값(median)과 IQR로 표현: RobustScaler()
- 특성들이 같은 스케일을 같게 된다는 통계적 측면에서는 표준화와 비슷하지만
  평균과 분산 대신 중앙값(median)과 사분위수(quantile)을 사용함
  - ✦ 이 때문에 RobustScaler는 이상치의 영향을 받지 않음

09 | 선형 변수 변환: RobustScaler

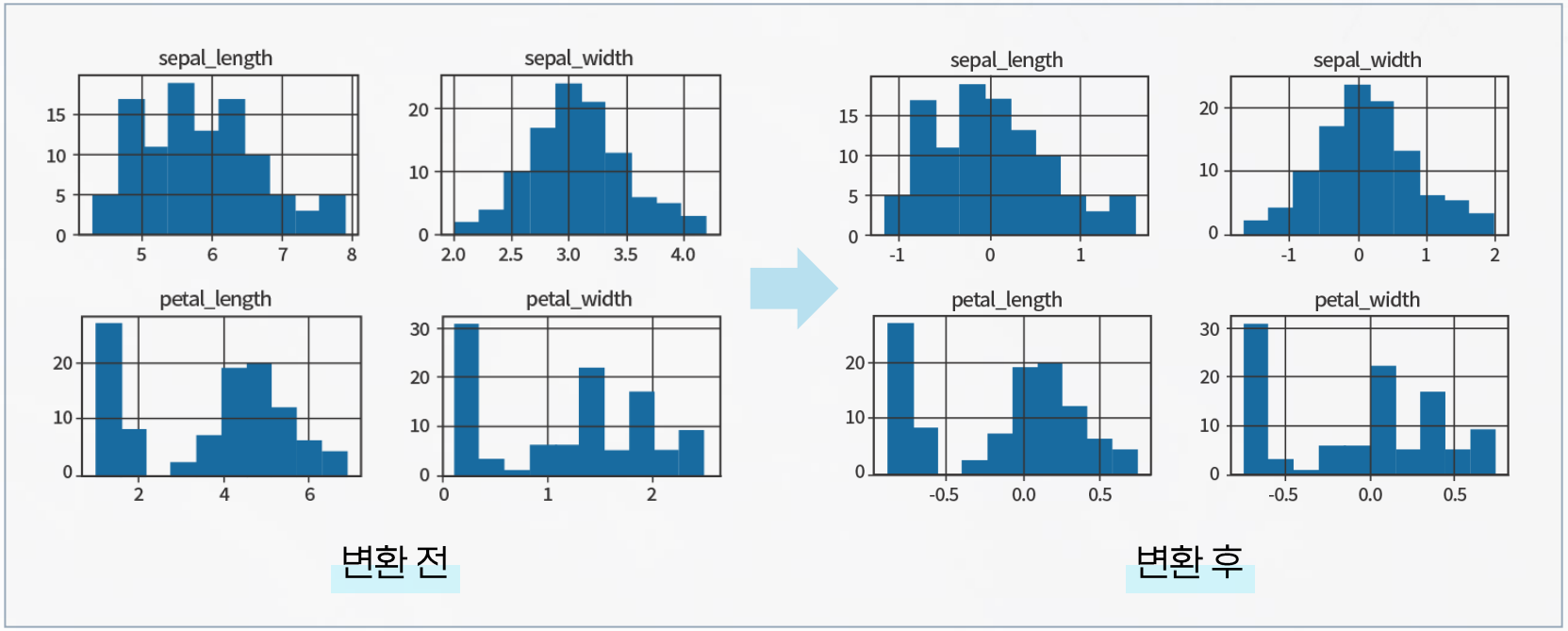
공식은 다음과 같다.

$$x' = \frac{x_i - median(x)}{IQR\ (x)}$$

- 새로운 데이터 x' 는 원래 값에서 변수 x의 중앙값(median)을 뺀 값을 변수 x의 IQR(Inter-Quartile Range)로 나눈 값으로 변환한 것임
  - IQR = Q3(3사 분위수) – Q1(1사 분위수)

09 | 선형 변수 변환: RobustScaler

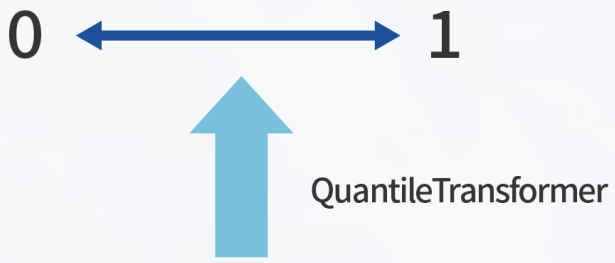
- 다음은 아이리스 훈련용 데이터 세트의 정규화 이전과 이후의 히스토그램을 비교해보자.
  - 비교 결과 그래프의 데이터 분포 모양은 동일한 것을 볼 수 있음
    - 여기서 중요한 점은 데이터 값이 중앙값을 뺀 값을 IQR(Inter-Quartile Range)로 나눈 값으로 변화된 것을 알 수 있음



10 | 선형 변수 변환: QuantileTransformer

균등분포, 정규분포(RankGauss) with QuantileTransformer

- Scikit Learn의 QuantileTransformer로 균등분포와 정규분포로의 선형 변환이 가능함
  - 균등분포는 1000개의 분위를 사용하여 데이터를 균등하게 배포시키는 방법임
  - RobustScaler와 비슷하게 이상치에 민감하지 않으며 전체 데이터를 0~1사이로 압축함



10 | 선형 변수 변환: QuantileTransformer

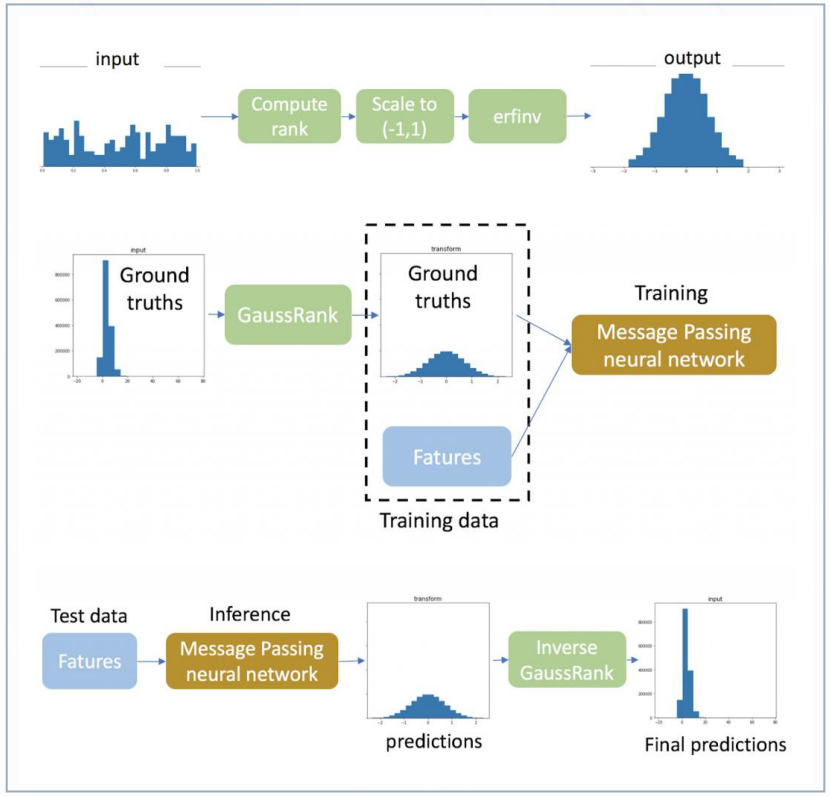
QuantileTransformer() 클래스는 다음 식으로 작성함

```
QuantileTransformer(output_distribution='uniform', n_quantiles=688)
```

- 여기서 output_distribution을 uniform으로 하면 균등분포가 되고, normal로 하면 정규분포가 됨
- 분위수는 n_quantiles 매개변수에서 설정할 수 있으며 기본값은 1000임

10 | 선형 변수 변환: QuantileTransformer

- 순위기반 가우스 분포 변환(RankGauss)는 수치형 변수를 순위로 변환한 뒤 순서를 유지한 채 반강제로 정규분포가 되도록 변환하는 방법임
  - 신경망에서 모델을 구축할 때의 변환으로서 일반적인 표준화보다 좋은 성능을 나타낸다고 함



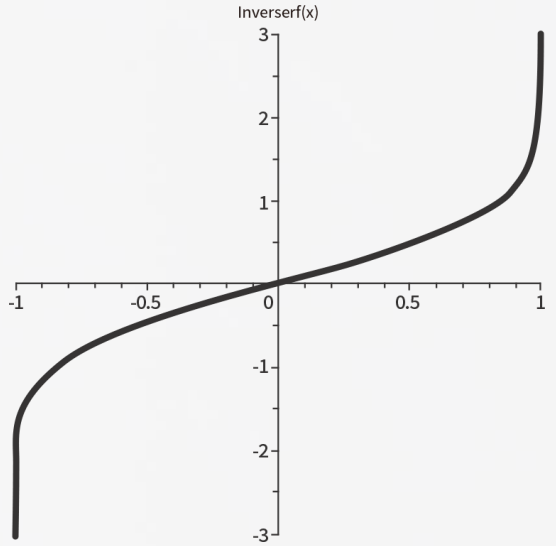
출처: https://developer.nvidia.com/blog/gauss-rank-transformation-is-100x-faster-with-rapids-and-cupy/

10 | 선형 변수 변환: QuantileTransformer

신경망(Neural Network)뿐 아니라 대부분 모델에서 학습을 수행하기 전에 데이터 값의 범위를 정규화(Normalization), 표준화(Standardization)하는 것은 매우 중요함

- 순위기반 가우스 분포 변환(Gaussian Rank) 방법은 숫자 특징 분포 (Numeric Feature Distribution)를 정규 분포(Normal Distribution)로 변형시켜줌

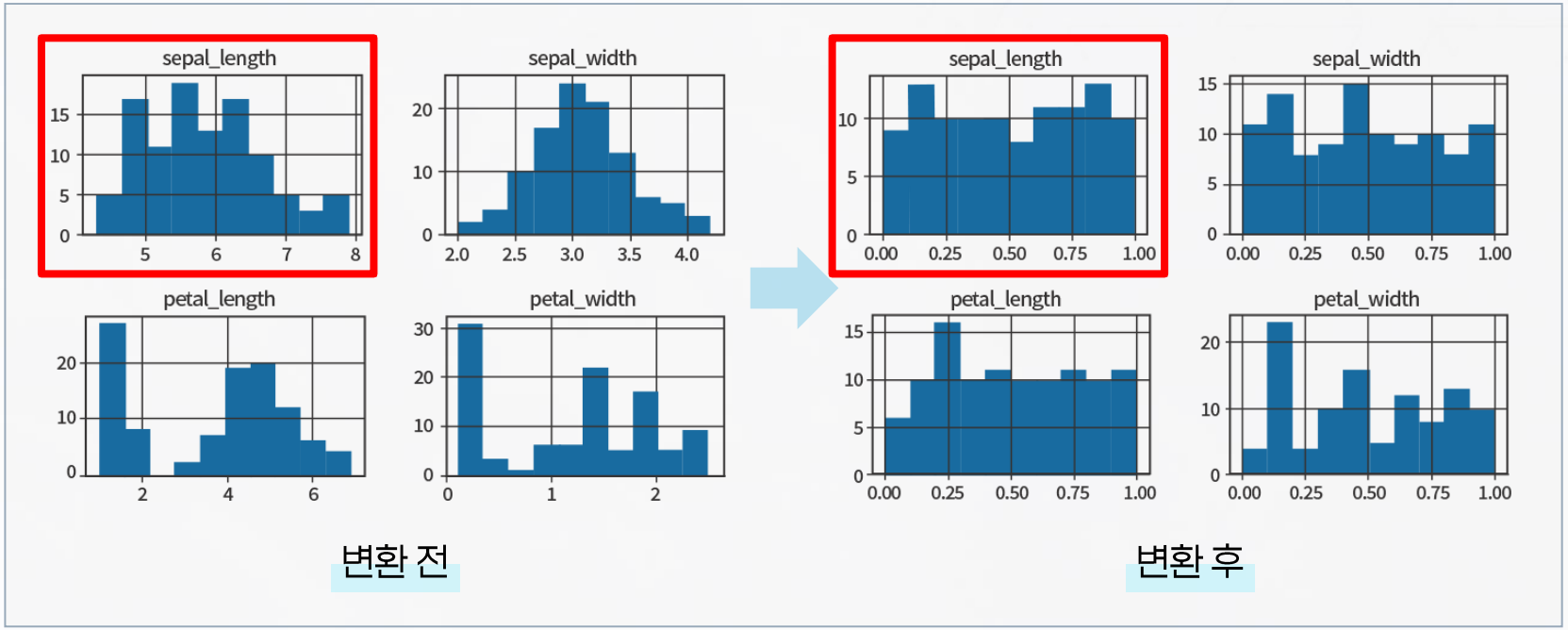
1. -1 ~ 1사이의 값(clipped value)을 순서(sorted)를 매김
2. 순서 값에 오차역함수(inverse error function)을 적용하여 마치 정규분포(Normal Distribution)처럼 만듦



오차역함수 (inverse error function)

10 | 선형 변수 변환: QuantileTransformer

- 다음은 아이리스 훈련용 데이터 세트의 정규화 이전과 이후의 히스토그램을 비교해보자.
  - 비교 결과 그래프의 데이터 분포 모양은 많이 다른 것을 볼 수 있음
    - 여기서 중요한 점은 데이터 값이 0~1 사이 값으로 변화된 것을 알 수 있음



10 | 선형 변수 변환: QuantileTransformer

다음은 아이리스 시험용 데이터 세트에서 변경 전과 변경 후의 최소, 최대값을 확인하는 코드이다.

- 아래와 같이 정규화된 이후 각 변수 값의 최소와 최대값이 0~1 사이인 것을 알 수 있음

```
print(x_test.min(axis=0), "\n", x_test.max(axis=0), "\n\n",
iris_qtS_test.min(axis=0), "\n", iris_qtS_test.max(axis=0))
```

```
sepal_length    4.4
sepal_width     2.2
petal_length    1.3
petal_width     0.1
dtype: float64
 sepal_length    7.7
sepal_width     4.4
petal_length    6.7
petal_width     2.5
dtype: float64

 [0.00671141 0.01226994 0.04444444 0.        ]
 [0.98305085 1.         0.98773006 1.        ]
```

각 변수의 최소, 최대값이 0~1 사이