

✓ 정규화/표준화

✓ 정규화(normalization)와 표준화(standardization)는 왜 하는걸까?

- 기준을 설정하고 그 기준내에서 각 데이터들을 평가하면 비교가 효과적임

- 머신러닝을 하다보면 기본적으로 많은 양의 데이터를 처리하게 되며 그 데이터 안에 target과 관련있는 특성(feature)을 추출함

- 예를 들어, 오늘 입을 옷을 결정할 때 그날의 온도, 습도, 날씨, 약속의 유무 등을 따지는 것과 같음
 - 온도, 습도, 날씨는 °C, F 등 특성의 단위도 다르고 그 범위도 달라 직접적으로 비교할 수 없음

- 각 특성들의 단위를 무시하고 값으로 단순 비교할 수 있게 만들어 줄 필요가 있음

- 정규화/표준화해주는 것을 특성 스케일링(feature scaling) 또는 데이터 스케일링(data scaling)이라고 함

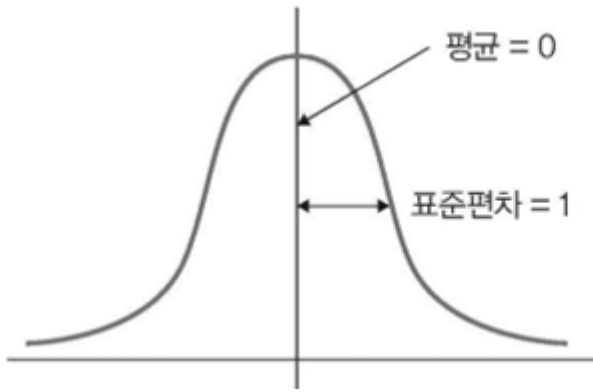
- 가장 중요한 이유는 scale의 범위가 너무 크면 노이즈 데이터가 생성되거나 overfitting이 될 가능성이 높아지기 때문임
-

스케일링 (Scaling)	<ul style="list-style-type: none"> • 수치형 데이터가 여러개 있는데 숫자범위가 다르다면 → 범위를 비슷하게 만들어줄 필요가 있다. • 왜냐하면? 그대로 모델링하면 상대적으로 큰 숫자를 가지는 칼럼의 기여도가 더 크게 반영되어버리기 때문! • 따라서 스케일링하여 상대적인 크기 차이를 없애줘야 함 • 스케일링을 통해 데이터의 불필요한 차원을 줄이고, 분석모델의 학습속도가 향상시킬 수 있음 (효율 ↑)
정규화 (Normalization)	<ul style="list-style-type: none"> • Min-Max Scaling : 데이터의 최솟값, 최댓값을 이용해서 데이터를 일정 범위 내의 값들로 변환 • 일반적으로 0~1 사이의 값으로 변환 ※ 데이터에 이상치가 없고, 분포가 크게 치우쳐 있지 않은 경우에 적합한 방법 (이상치 제거 후 정규화해야함)
표준화 (Standardization)	<ul style="list-style-type: none"> • 데이터의 평균, 표준편차를 이용해서 변환 • 평균=0, 표준편차=1 이 되도록 데이터 값들을 변환 ※ 데이터에 이상치가 있고, 분포가 치우쳐 있는 경우 적합한 방법

✓ 표준화(Standardization) - 정규분포를 따른다고 가정하고 변환하는 표준화 스케일링

- 데이터의 평균을 0 분산 및 표준편차를 1로 만들어 줌
 - 변수의 평균을 빼고 표준편차로 나눠줌
 - 평균이 0이고 분산이 1인 정규분포를 따르는 값으로 변환됨
 - 평균을 기준으로 얼마나 떨어져 있는지 살펴볼 때 사용
- 표준화를 하는 이유
 - 서로 다른 통계 데이터들을 비교하기 용이하기 때문임

- 표준화를 하면 평균은 0, 분산과 표준편차는 1로 만들어 데이터의 분포를 단순화 시키고, 비교를 용이하게 함



표준화 공식

$$z = \frac{(X - \mu)}{\sigma}$$

```
1 import pandas as pd
2 #iris 붓꽃 샘플데이터를 가져옴
3 from sklearn.datasets import load_iris
4 #iris 데이터를 가져와 iris 변수에 대입
5 iris = load_iris()
6 #iris 데이터를 활용하여 DataFrame을 생성
7 df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
8 df['target'] = iris['target']
9 df.head()
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```

1 #4개의 feature 데이터 중 sepal length (cm)의 feature만 임의로 선택하여 X 변수에 저장
2 #sepal length (cm) 컬럼만 X 변수에 저장
3 X = df.iloc[:, 0]

```

```

1 #표준화 코드 구현
2 #X_ 변수에 바로 위에서 만든 X 변수를 표준화를 거친 후 결과를 담음
3 X_ = (X - X.mean()) / X.std()
4 print(X_)

```



```

0    -0.897674
1    -1.139200
2    -1.380727
3    -1.501490
4    -1.018437

...
145    1.034539
146    0.551486
147    0.793012
148    0.430722
149    0.068433
Name: sepal length (cm), Length: 150, dtype: float64

```

```

1 #시각화
2 #시각화로 표준화의 전과 후를 비교
3 import matplotlib.pyplot as plt

```

```
4 import seaborn as sns
5 %matplotlib inline
6 plt.figure(figsize=(12, 6))
7 plt.subplot(1, 2, 1)
8 sns.distplot(X, bins=5, color='b')
9 plt.title('Original', fontsize=16)
10
11 plt.subplot(1, 2, 2)
12 sns.distplot(X_, bins=5, color='r')
13 plt.title('Standardization', fontsize=16)
14 plt.show()
```

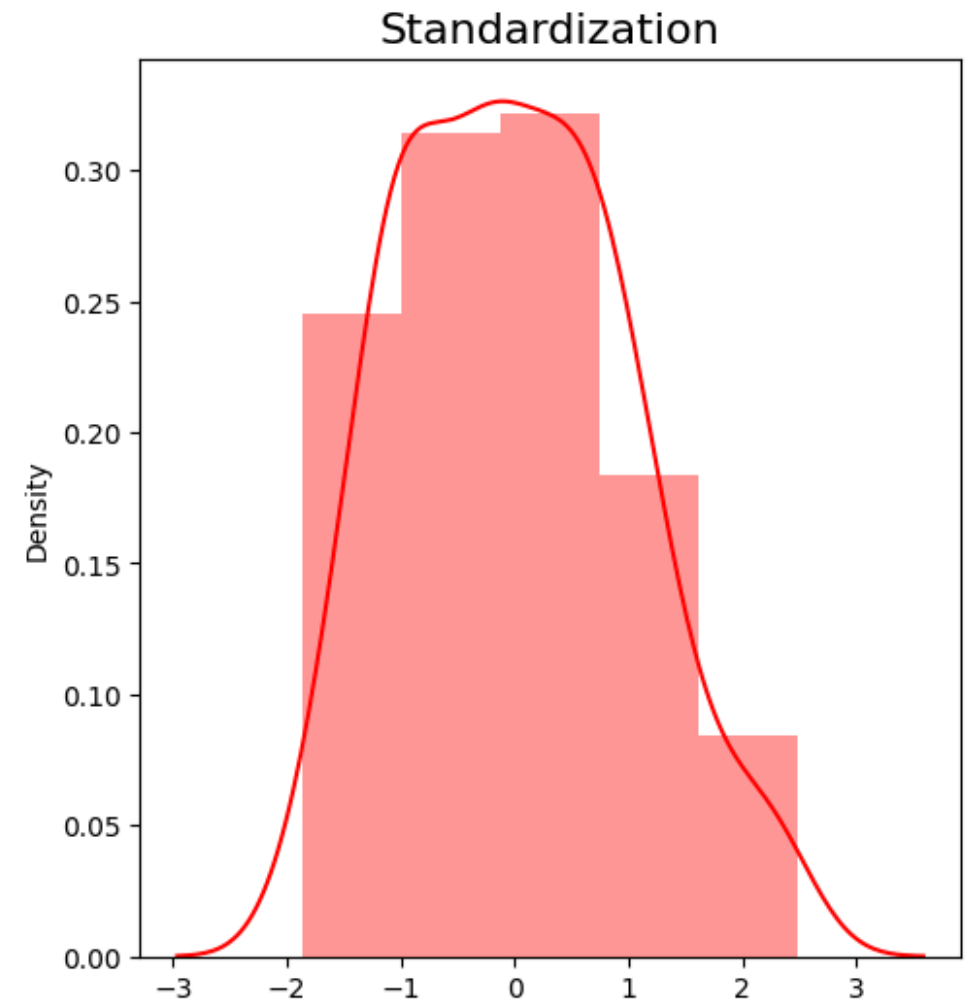
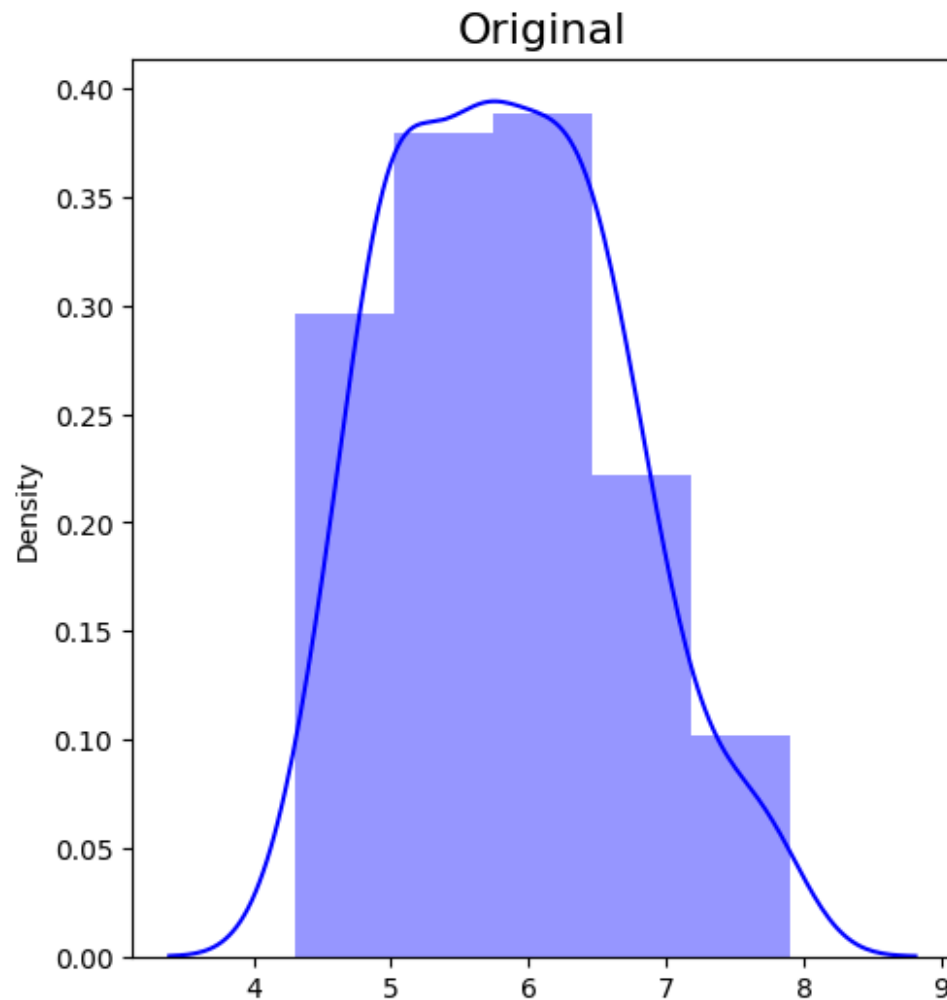
```
<ipython-input-5-79669bfcd8db>:12: UserWarning:
```

``distplot`` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either ``displot`` (a figure-level function with similar flexibility) or ``histplot`` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(X_, bins=5, color='r')
```



sepal length (cm)

sepal length (cm)

```

1 from sklearn.datasets import load_iris
2 iris = load_iris()
3 #데이터 탐색
4 iris.keys()

```

```

➔ dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

```

```

1 #target_name, target: setosa가 0번, versicolor가 1번, virginica가 2번
2 #key 호출 시에는 iris['DESCR'] 혹은 iris.target 형태로 사용
3 iris['DESCR']

```

```

➔ '.. _iris_dataset:~~~~~**Data Set Characteristics:**~~~~~
Iris plants dataset~~~~~
Number of Instances: 150 (50 in each of
three classes)
Number of Attributes: 4 numeric, predictive attributes and the class
Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
        - Iris-Setosa
        - Iris-Versicol
        - Iris-Virginica
~~~~~Summary Statistics:~~~~~
~~~~~=====
n  Max   Mean   SD   Class Correlation
sepal length:  4.3  7.9   5.84   0.83
sepal width:   2.0  4.4   3.05   0.43
petal length:  1.0  6.9   3.76   1.76
petal width:   0.1  2.0   0.94   0.99

```

```

1 import pandas as pd
2 iris_pd = pd.DataFrame(iris.data, columns = iris.feature_names)
3 iris_pd.head()

```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

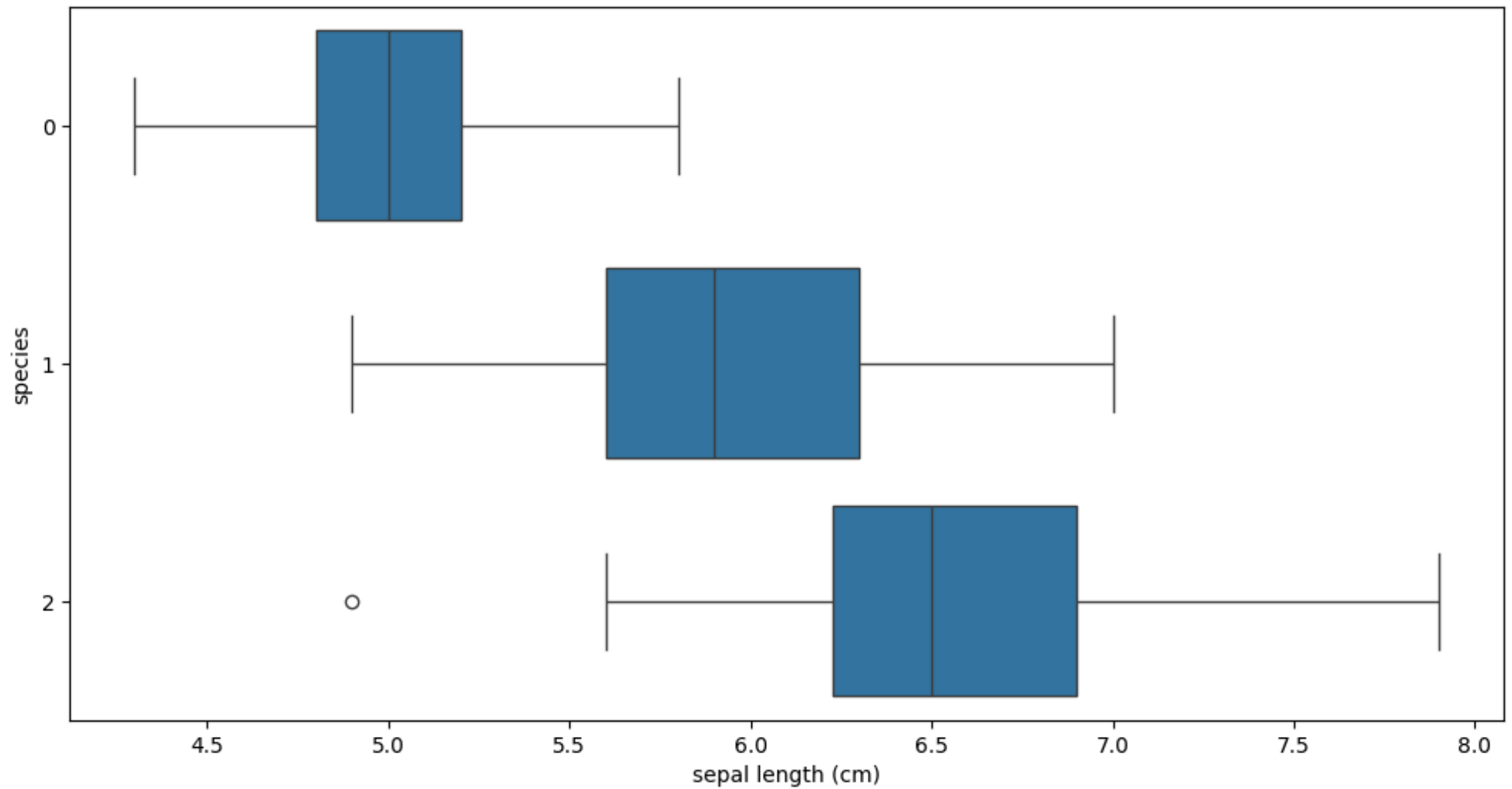
```
1 iris_pd['species'] = iris.target
2 iris_pd.head()
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

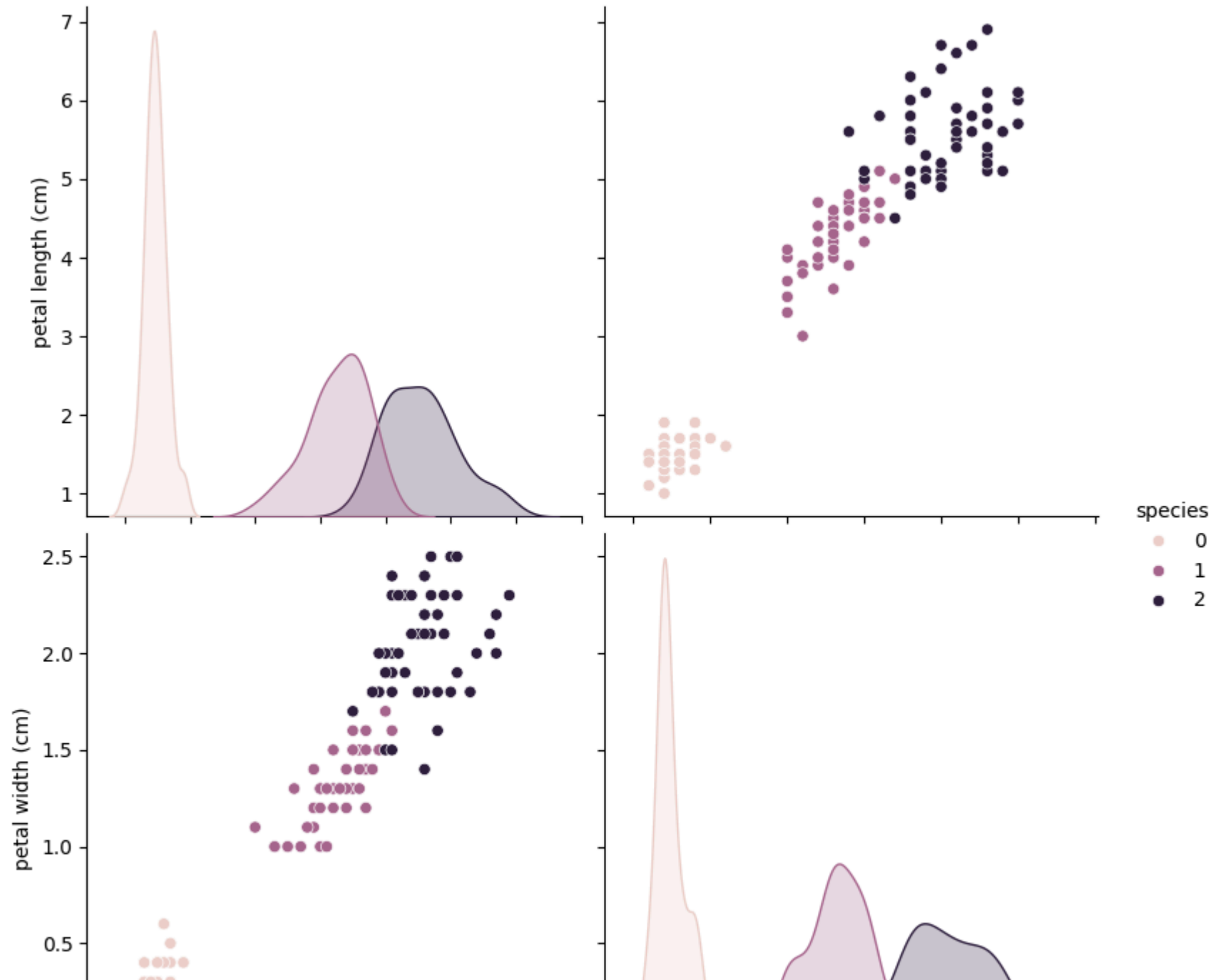
```
1 #https://velog.io/@alicejohn5/Zerobase-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%B7%A8%EC%97%85-%EC%8A%A4%EC%BF%A8-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9C
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 plt.figure(figsize=(12,6))
5 sns.boxplot(x='sepal length (cm)', y='species', data=iris_pd, orient='h') # orient = 'h'는 수평바를 그리는 것
```

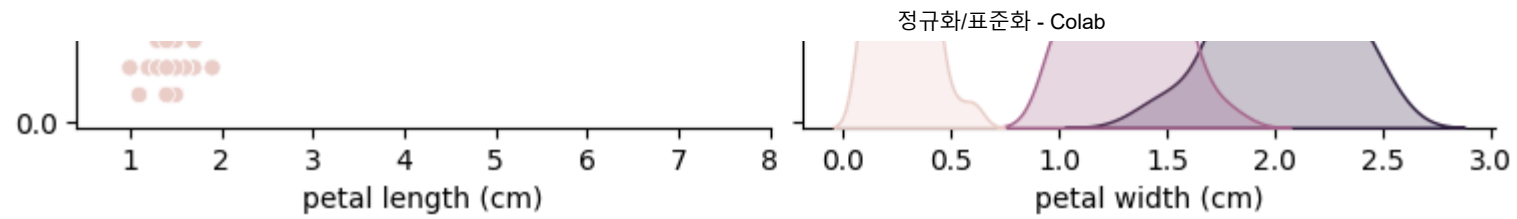

 <Axes: xlabel='sepal length (cm)', ylabel='species'>



```
1 sns.pairplot(data=iris_pd,  
2               vars=['petal length (cm)', 'petal width (cm)'],  
3               hue='species', height=4)
```

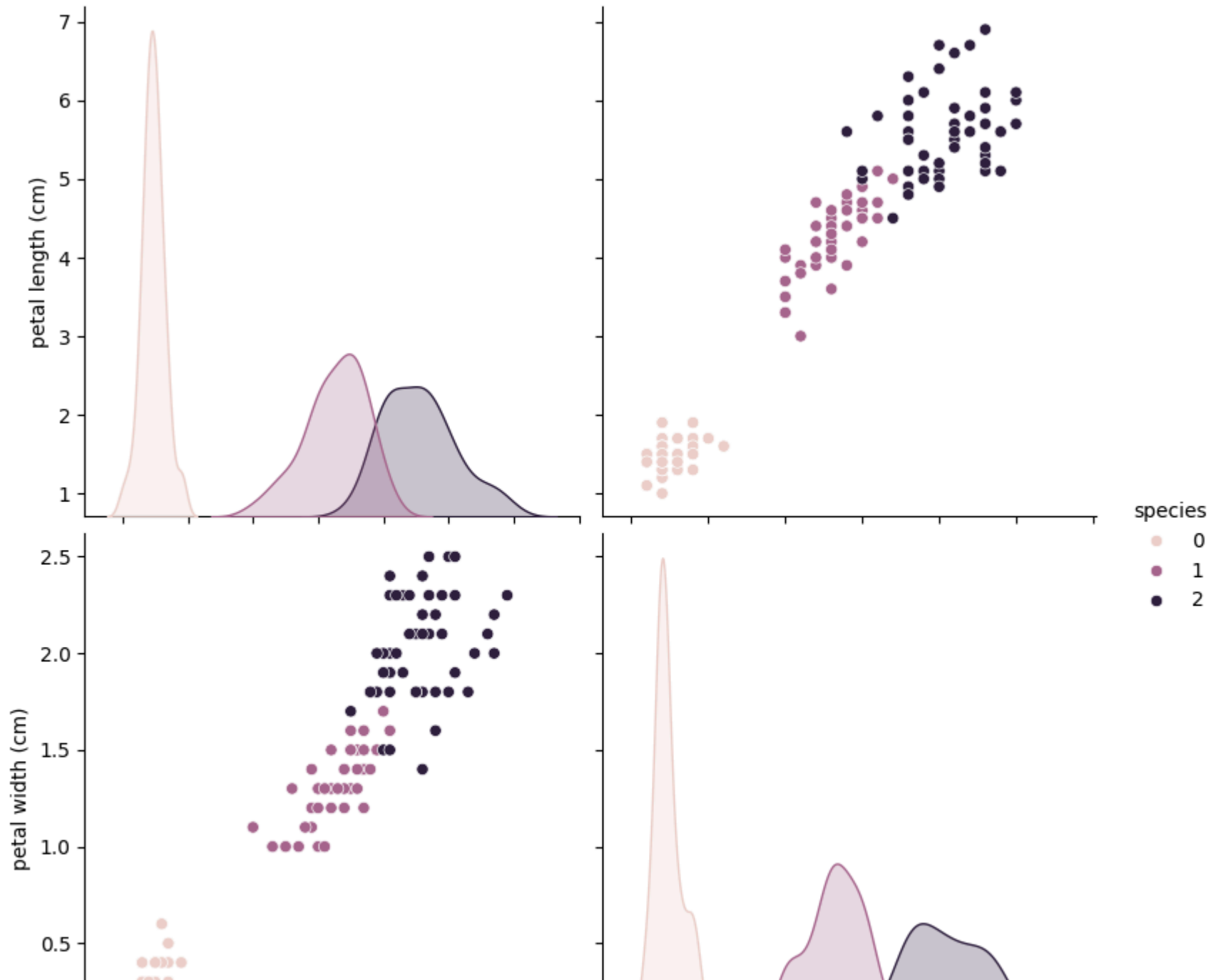
 <seaborn.axisgrid.PairGrid at 0x7805c1684850>

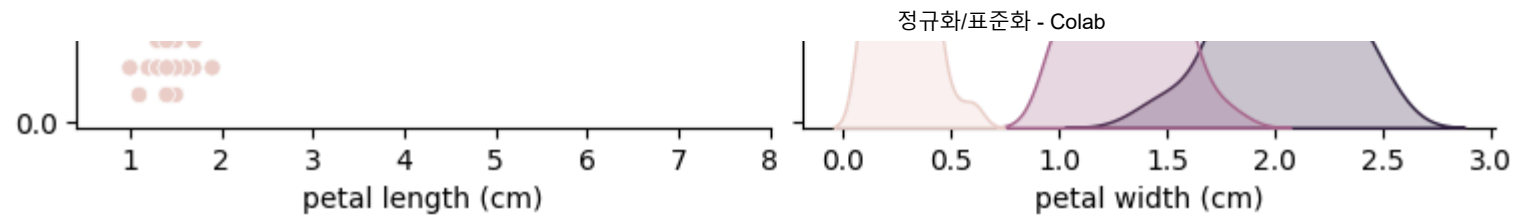




```
1 sns.pairplot(data=iris_pd,  
2               vars=['petal length (cm)', 'petal width (cm)'],  
3               hue='species', height=4)
```

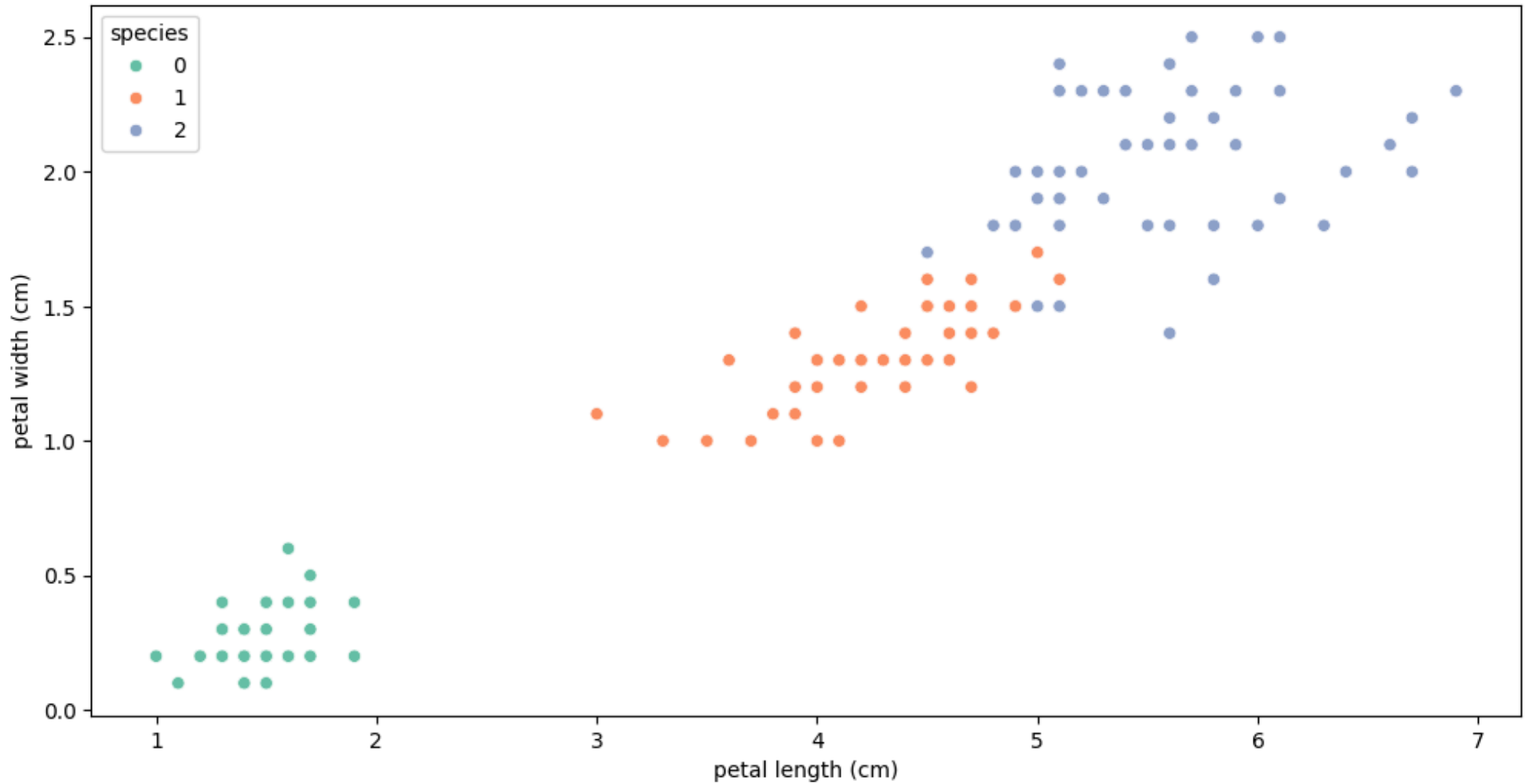
 <seaborn.axisgrid.PairGrid at 0x7805bd1c3a10>





```
1 plt.figure(figsize=(12,6))
2 sns.scatterplot(x="petal length (cm)", y="petal width (cm)",
3                 data=iris_pd, hue='species', palette="Set2")
```

```
<Axes: xlabel='petal length (cm)', ylabel='petal width (cm)'>
```



✓ 정규화(Normalization) - 다른 단위를 따르는 변수들을 동일한 특정 구간으로 조정

- 서로 다른 데이터의 크기를 통일하기 위해 크기를 변환하는 것

◦ 예를들어, 이미지 데이터는 픽셀 정보를 0-255 사이의 값을 갖는데 이를 255로 나누면 0~1.0 사이의 값을 갖게 됨

- 정규화는 실제 값을 특정 구간, 예를 들어 [-1,+1] 혹은 [0,1] 구간으로 변환하는 스케일링 방식
- 최대값과 최소값 범위를 분모로 두어서 전체 구간을 축소시킴
- 수식에서 x_i 값을 최대값이라고 가정하면 전체 값은 1이 되고, x_i 를 최소값으로 두면 전체 값은 0이 됨
- [0,1] 구간을 대부분 많이 사용



$$z = \frac{x - \min(x)}{[\max(x) - \min(x)]}$$

정규화 수식

```

1 #패키지 & 데이터프레임 불러오기
2 #예시가 될 데이터셋은 Kaggle에서 가져온 Flight Price Prediction .
3 #수치형 칼럼인 비행시간(duration), 출발까지 남은일수(days_left), 가격(price)만 가져와서 스케일링
4 # 패키지 불러오기
5 import numpy as np
6 import pandas as pd
7 pd.set_option('float_format', '{:.4f}'.format)
8
9 # 데이터셋 불러오기 (수치형 칼럼만)
10 df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ML/DataSet/Clean_Dataset.csv', usecols=['duration','days_left','price'], encoding='c

```

11
12 df



	duration	days_left	price
0	2.1700	1	5953
1	2.3300	1	5953
2	2.1700	1	5956
3	2.2500	1	5955
4	2.3300	1	5955
...
300148	10.0800	49	69265
300149	10.4200	49	77105
300150	13.8300	49	79099
300151	10.0000	49	81585
300152	10.0800	49	81585

300153 rows × 3 columns

1 #정규화(Normalization) = Min-Max Scaling

2 #데이터프레임의 각 칼럼마다 최솟값, 최댓값을 이용해서 특정 범위의 값으로 정규화할 수 있음

0~1 사이의 값으로 정규화

a~b 사이의 값으로 정규화


$$x_{normalized} = \frac{x_i - \min}{\max - \min}$$

$$x_{normalized} = \frac{x_i - \min}{\max - \min} \times (b - a) + a$$


```


1 # 정규화: 0~1 사이의 값으로 변환
2 df_norm_01 = (df - df.min()) / (df.max() - df.min())
3
4 # 정규화: 1~10 사이의 값으로 변환
5 df_norm_10 = ((df - df.min()) / (df.max() - df.min())) * (10-1) + 1
6
7 df_norm_01.head()

```




	duration	days_left	price
0	0.0273	0.0000	0.0397
1	0.0306	0.0000	0.0397
2	0.0273	0.0000	0.0398
3	0.0290	0.0000	0.0398
4	0.0306	0.0000	0.0398

```
1 df_norm_10.head()
```




	duration	days_left	price
0	1.2461	1.0000	1.3577
1	1.2755	1.0000	1.3577
2	1.2461	1.0000	1.3580
3	1.2608	1.0000	1.3579
4	1.2755	1.0000	1.3579

```
1 df_norm_01.describe()
```



	duration	days_left	price
count	300153.0000	300153.0000	300153.0000
mean	0.2325	0.5209	0.1622
std	0.1468	0.2825	0.1861
min	0.0000	0.0000	0.0000
25%	0.1224	0.2917	0.0302
50%	0.2127	0.5208	0.0518
75%	0.3131	0.7708	0.3396
max	1.0000	1.0000	1.0000

```
1 df_norm_10.describe()
```



	duration	days_left	price
count	300153.0000	300153.0000	300153.0000
mean	3.0922	5.6884	2.4599
std	1.3210	2.5427	1.6749
min	1.0000	1.0000	1.0000
25%	2.1020	3.6250	1.2714
50%	2.9139	5.6875	1.4664
75%	3.8176	7.9375	4.0561
max	10.0000	10.0000	10.0000

```

1 #사이킷런으로 정규화 (Normalization) :: MinMaxScaler()
2 #사이킷런의 MinMaxScaler() 모듈로도 데이터프레임을 정규화시킬 수 있음
3 #다만 array 형태로 결과를 반환하기 때문에, 아래와 같이 다시 데이터프레임으로 바꿔줘야 함
4 # 스케일러 모듈 불러오기
5 from sklearn.preprocessing import MinMaxScaler
6 mms = MinMaxScaler()
7
8 # 정규화하고 데이터프레임으로 바꿔주기
9 arr_norm_sklearn = mms.fit_transform(df)
10 df_norm_sklearn = pd.DataFrame(arr_norm_sklearn, columns=df.columns)

```

```
1 df_norm_sklearn
```



	duration	days_left	price
0	0.0273	0.0000	0.0397
1	0.0306	0.0000	0.0397
2	0.0273	0.0000	0.0398
3	0.0290	0.0000	0.0398
4	0.0306	0.0000	0.0398
...
300148	0.1888	1.0000	0.5588
300149	0.1957	1.0000	0.6231
300150	0.2653	1.0000	0.6395
300151	0.1871	1.0000	0.6599
300152	0.1888	1.0000	0.6599

300153 rows × 3 columns

```
1 df_norm_sklearn.head()
```



	duration	days_left	price
0	0.0273	0.0000	0.0397
1	0.0306	0.0000	0.0397
2	0.0273	0.0000	0.0398
3	0.0290	0.0000	0.0398
4	0.0306	0.0000	0.0398

1 #표준화(Standardization)

2 #아래와 같이 데이터프레임의 각 칼럼마다 평균값, 표준편차를 이용해서 표준화할 수 있음

3 #표준화하면, 평균=0 이고 표준편차=1 인 표준정규분포 상의 값들로 모든 데이터 값들이 변환됨

평균 & 표준편차로 데이터 표준화

$$z_i = \frac{x_i - \text{mean}}{\text{std. deviation}}$$

```
1 df_std = (df - df.mean()) / df.std()
```

```
2 df_std.head()
```

```
df_std.head(1)
df_std
duration  days_left  price
0      1.2075      1.2420  0.6501

1 df_std.describe()
```

```
df_std
duration  days_left  price
```