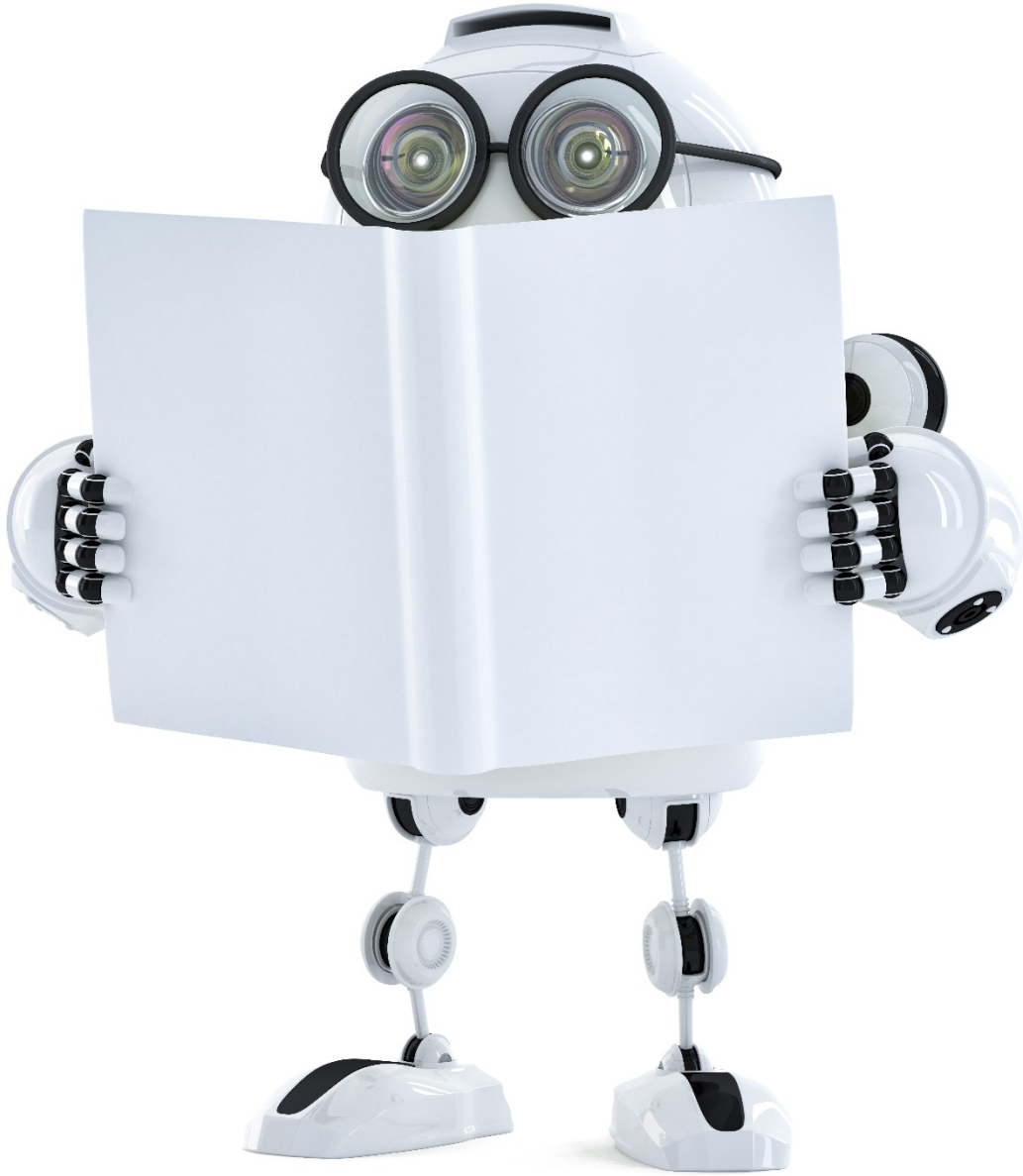
Data cleansing

Data handling



Director of TEAMLAB
Sungchul Choi

Data problems

결국 우리는
데이터 부터...

Data quality problems

- 데이터의 최대/최소가 다름 → Scale에 따른 y값에 영향
- Ordinary 또는 Nominal 한 값 들의 표현은 어떻게?
- 잘 못 기입된 값들에 대한 처리
- 값이 없을 경우는 어떻게?
- 극단적으로 큰 값 또는 작은 값들은 그대로 놔둬야 하는가?

어떤 것을
해결할 것인가?

Data preprocessing issues

- 데이터가 빠진 경우 (결측치의 처리)
- 라벨링된 데이터(category) 데이터의 처리
- 데이터의 scale의 차이가 매우 크게 날 경우

Missing Values

데이터가 없을 때 할 수 있는 전략

- 데이터가 없으면 sample을 drop
- 데이터가 없는 최소 개수를 정해서 sample을 drop
- 데이터가 거의 없는 feature는 feature 자체를 drop
- 최빈값, 평균값으로 비어있는 데이터를 채우기

Data

```
# Eaxmple from - https://chrisalbon.com/python/pandas_missing_data.html
raw_data = {'first_name': ['Jason', np.nan, 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', np.nan, 'Ali', 'Milner', 'Cooze'],
        'age': [42, np.nan, 36, 24, 73],
        'sex': ['m', np.nan, 'f', 'm', 'f'],
        'preTestScore': [4, np.nan, np.nan, 2, 3],
        'postTestScore': [25, np.nan, np.nan, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'sex', 'preTestScore', 'postTestScore'])
df
```

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

Data drop

```
df.isnull().sum()
```

```
first_name        1
last_name         1
age               1
sex               1
preTestScore      2
postTestScore     2
dtype: int64
```

NaN이 데이터를 column별로 합계

```
df_no_missing = df.dropna()
df_no_missing
```

drop nan → 데이터들이 사라짐

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

Data drop

```
df_cleaned = df.dropna(how='all')
df_cleaned
```

모든 데이터가 비어 있으면 drop

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

Data drop

```
df['location'] = np.nan
df
```

NAN을 생성 column

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

```
df.dropna(axis=1, how='all')
```

column 기준으로 삭제

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	3.0	NaN
2	Tina	Ali	36.0	f	3.0	70.0
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

데이터가 최소 4개 이상 없을 때 drop

```
df.dropna(axis=1, thresh=3)
```

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	3.0	NaN
2	Tina	Ali	36.0	f	3.0	70.0
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

Data drop

```
df.dropna(thresh=5)
```

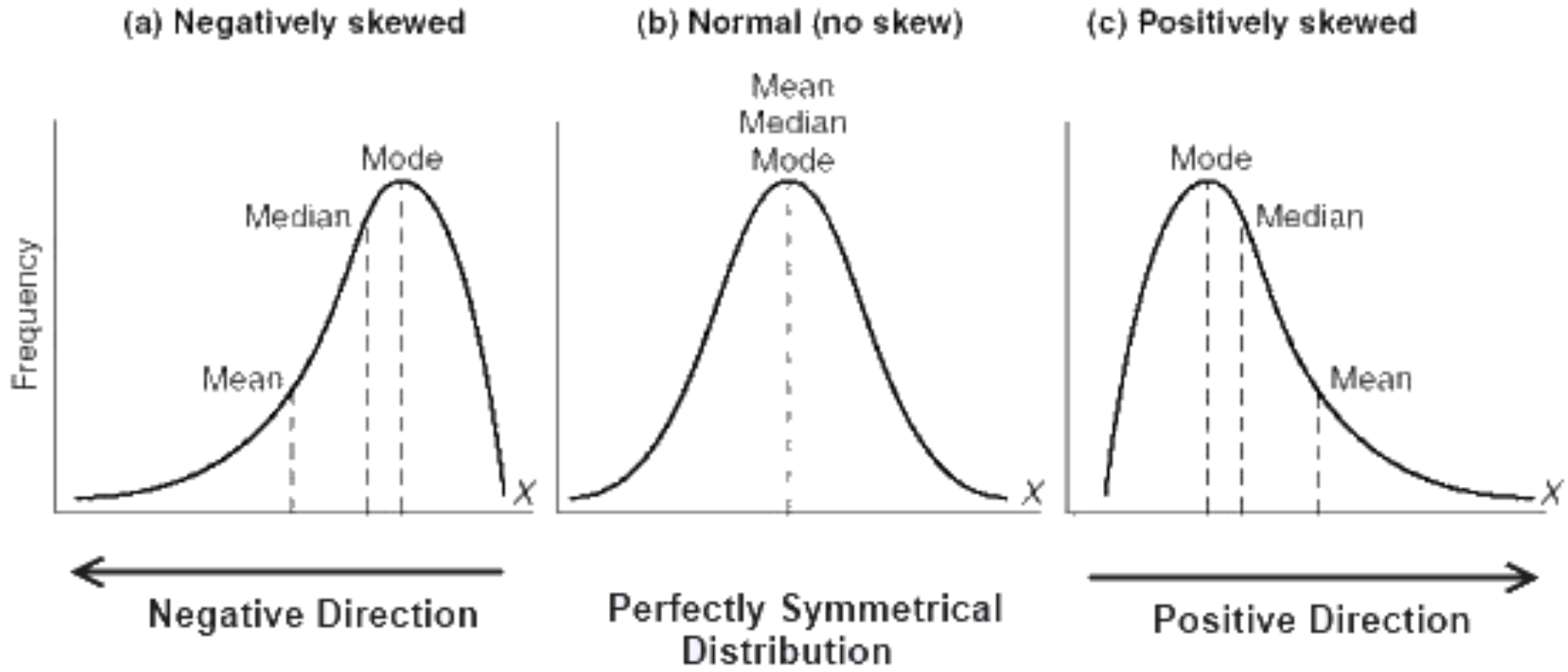
5개 이상 데이터가 있지 않으면 Drop

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

데이터 값 채우기

- 평균값, 중위값, 최빈값을 활용

https://goo.gl/i8iuL9



데이터가 채우기

- 평균값 – 해당 column의 값의 평균을 내서 채우기

$$\overline{X} = \frac{\sum_{i=1}^{n} x_i}{n}$$

```
df["preTestScore"].mean()
```

```
3.0
```

- 중위값 – 값을 일렬로 나열했을 때 중간에 위치한 값

1, 3, 3, **6**, 7, 8, 9 $x_{\frac{(n-1)}{2}}$

```
df["postTestScore"].median()
```

```
66.0
```

- 최빈값 – 가장 많이 나오는 값

1, 2, 2, **3**, **3**, 4, 4, **3**

```
df["postTestScore"].mode()
```

```
0    70.0
dtype: float64
```

Data Fill

```
df.fillna(0)
```

데이터가 없는 곳은 0으로 집어넣어라

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	0.0
1	0	0	0.0	0	0.0	0.0	0.0
2	Tina	Ali	36.0	f	0.0	0.0	0.0
3	Jake	Milner	24.0	m	2.0	62.0	0.0
4	Amy	Cooze	73.0	f	3.0	70.0	0.0

```
df["preTestScore"].fillna(df["preTestScore"].mean(), inplace=True)
df
```

preTestScore의 평균값을 집어넣어라

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
1	NaN	NaN	NaN	NaN	3.0	NaN	NaN
2	Tina	Ali	36.0	f	3.0	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

Data Fill

```
df["postTestScore"].fillna(df.groupby("sex")["postTestScore"].transform("mean"), inplace=True)
df
```

성별로 나눠서 평균 값을 집어 넣어라

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
1	NaN	NaN	NaN	NaN	3.0	NaN	NaN
2	Tina	Ali	36.0	f	3.0	70.0	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

```
df[df['age'].notnull() & df['sex'].notnull()]
```

Age와 sex가 모두 notnull인 경우에만 표시해라

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
2	Tina	Ali	36.0	f	3.0	70.0	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

Category data

이산형 데이터를 어떻게 처리할까?

{Green, Blue, Yellow}

이산형 데이터를 어떻게 처리할까?

One-Hot Encoding

{Green, Blue, Yellow} 데이터 집합

{Green} ➔ [1, 0, 0]
{Green} ➔ [1, 0, 0]
{blue} ➔ [0, 1, 0]

실제 데이터 set의 크기만큼
Binary Feature를 생성

Data type

```
import pandas as pd
import numpy as np
```

```
edges = pd.DataFrame({'source': [0, 1, 2],
                      'target': [2, 2, 3],
                         'weight': [3, 4, 5],
                         'color': ['red', 'blue', 'blue']})
```

```
edges["source"]
```

Data의 type = int64

```
0    0
1    1
2    2
Name: source, dtype: int64
```

```
edges["color"]
```

```
0     red
1    blue
2    blue
Name: color, dtype: object
```

Data의 type = object

One Hot Encoding

```
pd.get_dummies(edges)
```

	source	target	weight	color_blue	color_red
0	0	2	3	0	1
1	1	2	4	1	0
2	2	3	5	1	0

```
pd.get_dummies(edges["color"])
```

	blue	red
0	0	1
1	1	0
2	1	0

```
pd.get_dummies(edges[["color"]])
```

	color_blue	color_red
0	0	1
1	1	0
2	1	0

One Hot Encoding

```
weight_dict = {3:"M", 4:"L", 5:"XL"}
edges["weight_sign"] = edges["weight"].map(weight_dict)
edges
```

Ordinary data → One Hot Encoding

	color	source	target	weight	weight_sign
0	red	0	2	3	M
1	blue	1	2	4	L
2	blue	2	3	5	XL

```
edges = pd.get_dummies(edges)
edges.as_matrix()
```

```
array([[0, 2, 3, 0, 1, 0, 1, 0],
       [1, 2, 4, 1, 0, 1, 0, 0],
       [2, 3, 5, 1, 0, 0, 0, 1]], dtype=int64)
```

데이터의 구간을 나눠보자

Data Binning!

- **Data :** 0, 4, 12, 16, 16, 18, 24, 26, 28
- **Equal width**
  - Bin 1: 0, 4 [-,10)
  - Bin 2: 12, 16, 16, 18 [10,20)
  - Bin 3: 24, 26, 28 [20,+)
- **Equal frequency**
  - Bin 1: 0, 4, 12 [-, 14)
  - Bin 2: 16, 16, 18 [14, 21)
  - Bin 3: 24, 26, 28 [21,+)

Data binning

```
# Example from - https://chrisalbon.com/python/pandas_binning_data.html

raw_data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts', 'S
        'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd','1st', '1st', '2nd', '2nd'],
        'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'],
        'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],
        'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['regiment', 'company', 'name', 'preTestScore', 'postTestScore'])
df
```

	regiment	company	name	preTestScore	postTestScore
0	Nighthawks	1st	Miller	4	25
1	Nighthawks	1st	Jacobson	24	94
2	Nighthawks	2nd	Ali	31	57
3	Nighthawks	2nd	Milner	2	62
4	Dragoons	1st	Cooze	3	70
5	Dragoons	1st	Jacon	4	25
6	Dragoons	2nd	Ryaner	24	94
7	Dragoons	2nd	Sone	31	57
8	Scouts	1st	Sloan	2	62
9	Scouts	1st	Piger	3	70
10	Scouts	2nd	Riani	2	62
11	Scouts	2nd	Ali	3	70

데이터의 구간을 나눌 수 있음

Data binning

구간 기준

```
bins = [0, 25, 50, 75, 100] # Define bins as 0 to 25, 25 to 50, 60 to 75, 75 to 100
group_names = ['Low', 'Okay', 'Good', 'Great']
categories = pd.cut(df['postTestScore'], bins, labels=group_names)
categories
```

구간명

Cut 후 categories에 할당

```
0       Low
1     Great
2      Good
3      Good
4      Good
5       Low
6     Great
7      Good
8      Good
9      Good
10     Good
11     Good
Name: postTestScore, dtype: category
Categories (4, object): [Low < Okay < Good < Great]
```

Data binning

```
df['categories'] = pd.cut(df['postTestScore'], bins, labels=group_names)
pd.value_counts(df['categories'])
```

기존 dataframe에 할당

```
Good     8
Great    2
Low      2
Okay     0
Name: categories, dtype: int64
```

```
df
```

	regiment	company	name	preTestScore	postTestScore	categories
0	Nighthawks	1st	Miller	4	25	Low
1	Nighthawks	1st	Jacobson	24	94	Great
2	Nighthawks	2nd	Ali	31	57	Good
3	Nighthawks	2nd	Milner	2	62	Good
4	Dragoons	1st	Cooze	3	70	Good
5	Dragoons	1st	Jacon	4	25	Low
6	Dragoons	2nd	Ryaner	24	94	Great
7	Dragoons	2nd	Sone	31	57	Good
8	Scouts	1st	Sloan	2	62	Good
9	Scouts	1st	Piger	3	70	Good
10	Scouts	2nd	Riani	2	62	Good
11	Scouts	2nd	Ali	3	70	Good

Label encoding by sklearn

- Scikit-learn의 preprocessing 패키지도 label, one-hot 지원

```
raw_example = df.as_matrix()
raw_example[:3]
```

```
array([['Nighthawks', '1st', 'Miller', 4, 25, 'Low'],
       ['Nighthawks', '1st', 'Jacobson', 24, 94, 'Great'],
       ['Nighthawks', '2nd', 'Ali', 31, 57, 'Good']], dtype=object)
```

```
data = raw_example.copy()
```

Label encoding by sklearn

- Scikit-learn의 preprocessing 패키지도 label, one-hot 지원

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(raw_example[:,0])
le.transform(raw_example[:,0])
```

Encoder 생성

Data에 맞게 encoding fitting

실제 데이터 → labelling data

```
array([1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 2, 2])
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(raw_example[:,0])
le.transform(raw_example[:,0])
```

```
array([1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 2, 2])
```

- Label encoder의 fit과 transform의 과정이 나눠진 이유는
- 새로운 데이터 입력시, 기존 labelling 규칙을 그대로 적용할 필요가 있음
- Fit 은 규칙을 생성하는 과정
- Transform은 규칙을 적용하는 과정
- Fit을 통해 규칙이 생성된 labelencoder는 따로 저장하여
- 새로운 데이터를 입력할 경우 사용할 수 있음
- Encoder들을 실제 시스템에 사용할 경우 pickle화 필요

```
label_column = [0,1,2,5]
label_enconder_list = []
for column_index in  label_column:
    le = preprocessing.LabelEncoder()
    le.fit(raw_example[:,column_index])
    data[:,column_index] = le.transform(raw_example[:,column_index])
    label_enconder_list.append(le)
    del le
data[:3]
```

기존 label encoder를 따로 저장

```
array([[1, 0, 4, 4, 25, 2],
       [1, 0, 2, 24, 94, 1],
       [1, 1, 0, 31, 57, 0]], dtype=object)
```

```
label_enconder_list[0].transform(raw_example[:10,0])
```

```
array([1, 1, 1, 1, 0, 0, 0, 0, 2, 2])
```

저장된 le로 새로운 데이터에 적용

One-hot encoding by sklearn

- Numeric labelling이 완료된 데이터에 one-hot 적용
- 데이터는 1-dim 으로 변환하여 넣어 줄 것을 권장

```
one_hot_enc = preprocessing.OneHotEncoder()
one_hot_enc.fit(data[:,0].reshape(-1,1))
```

1-dim 변환하여 fit

```
onehotlabels = one_hot_enc.transform(data[:,0].reshape(-1,1)).toarray()
onehotlabels
```

1-dim 변환후 transform → ndarray

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

Feature scaling

두 변수중 하나의 값의 크기가 너무 크다!

몸무게와 키가 변수일때, 키가 영향을 많이 줌

Feature scaling

Feature간의 최대-최소값의 차이를 맞춘다!

$$y = \beta_1 x_1 + \beta_2 x_2 + x_0$$

Feature scaling 전략

- **Min-Max Normalization**

  기존 변수에 범위를 **새로운 최대-최소로 변경**

  일반적으로 **0과 1 사이 값으로 변경**함

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}(new\_max - new\_low) + new\_low$$

최소 12,000 / 최대 98,000 → 기존 값 73,600

Feature scaling 전략

- Standardization (Z-score Normalization)

  기존 변수에 범위를 정규 분포로 변환

  실제 Mix-Max의 값을 모를 때 활용가능

$$x_{std\_norm}^{(i)} = \frac{x^{(i)} - \mu}{s_i}$$

평균 54,000 / 표준편자 16,000 → 73,600

주의 사항

실제 사용할 때는 반드시

정규화 Parameter(최대/최소, 평균/표준편차) 등을

기억하여 새로운 값에 적용해야함

Min-Max Normalization

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}(new\_max - new\_low) + new\_low$$

```
( df["A"] - df["A"].min() )
/ (df["A"].max() - df["A"].min()) * (5 - 1) + 1
```

	A	B	C
0	14.00	103.02	big
1	90.20	107.26	small
2	90.95	110.35	big
3	96.27	114.23	small
4	91.21	114.68	small

Z-Score Normalization

$$x^{(i)}_{std\_norm} = \frac{x^{(i)} - \mu}{s_i}$$

```
df["B"] = ( df["B"] - df["B"].mean() )  \
/ (df["B"].std() )
```

	A	B	C
0	14.00	103.02	big
1	90.20	107.26	small
2	90.95	110.35	big
3	96.27	114.23	small
4	91.21	114.68	small

Feature Scaling Function

```
def feture_scaling(df, scaling_strategy="min-max", column=None):
    if column == None:
        column = [column_name for column_name in df.columns]
    for column_name in column:
        if scaling_strategy == "min-max":
            df[column_name] = ( df[column_name] - df[column_name].min() ) /
                              (df[column_name].max() - df[column_name].min())
        elif scaling_strategy == "z-score":
            df[column_name] = ( df[column_name] - \
                                df[column_name].mean() ) /\
                              (df[column_name].std() )
    return df
```

Feature scaling with sklearn

- Label encoder와 마찬가지로, sklearn도 feature scale 지원
- MinMaxScaler와 StandardScaler 사용

```
from sklearn import preprocessing

std_scale = preprocessing.StandardScaler().fit(
    df[['Alcohol', 'Malic acid']])
df_std = std_scale.transform(df[['Alcohol', 'Malic acid']])
df_std[:5]
```

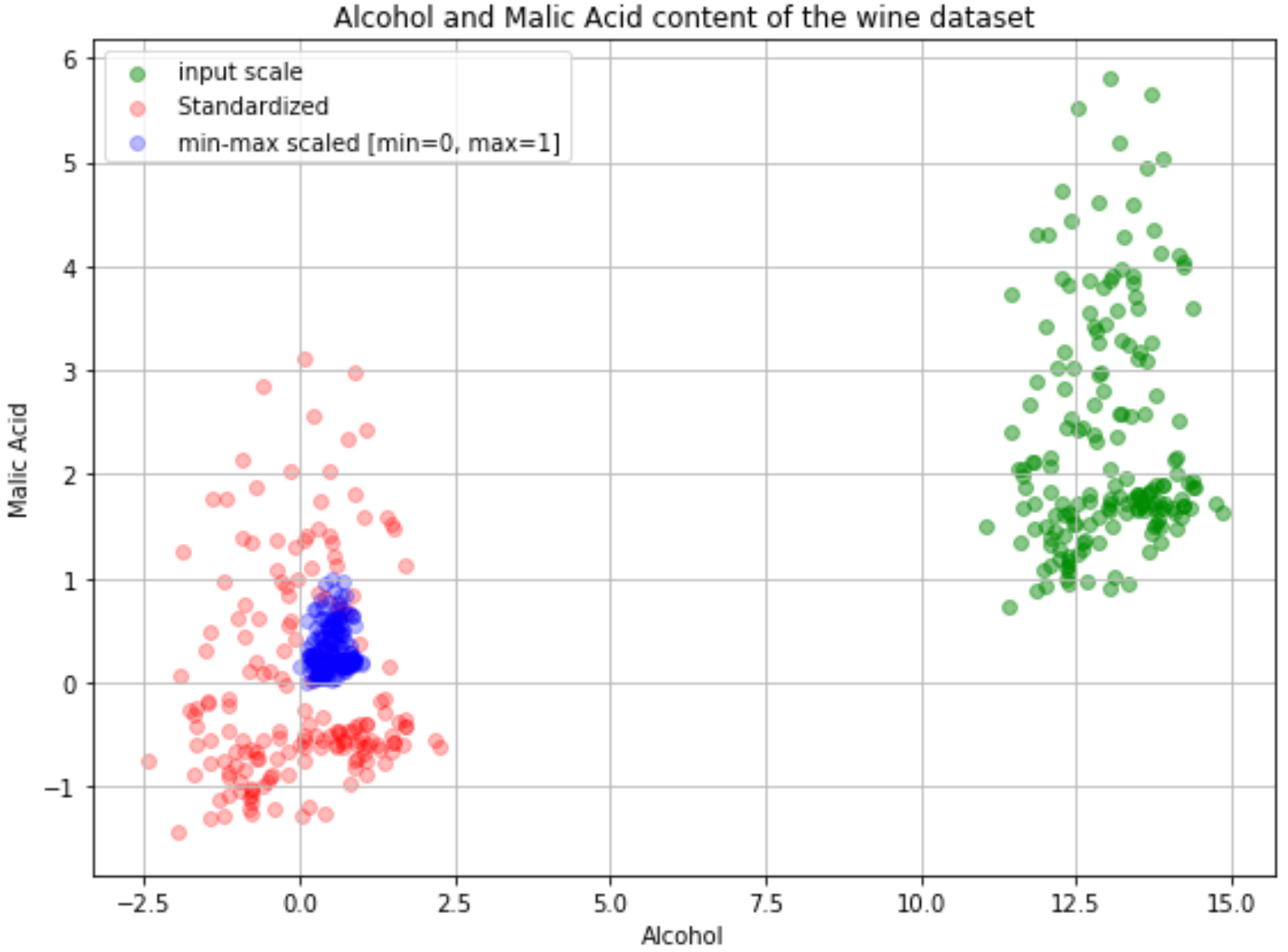
```
array([[ 1.51861254, -0.5622498 ],
       [ 0.24628963, -0.49941338],
       [ 0.19687903,  0.02123125],
```

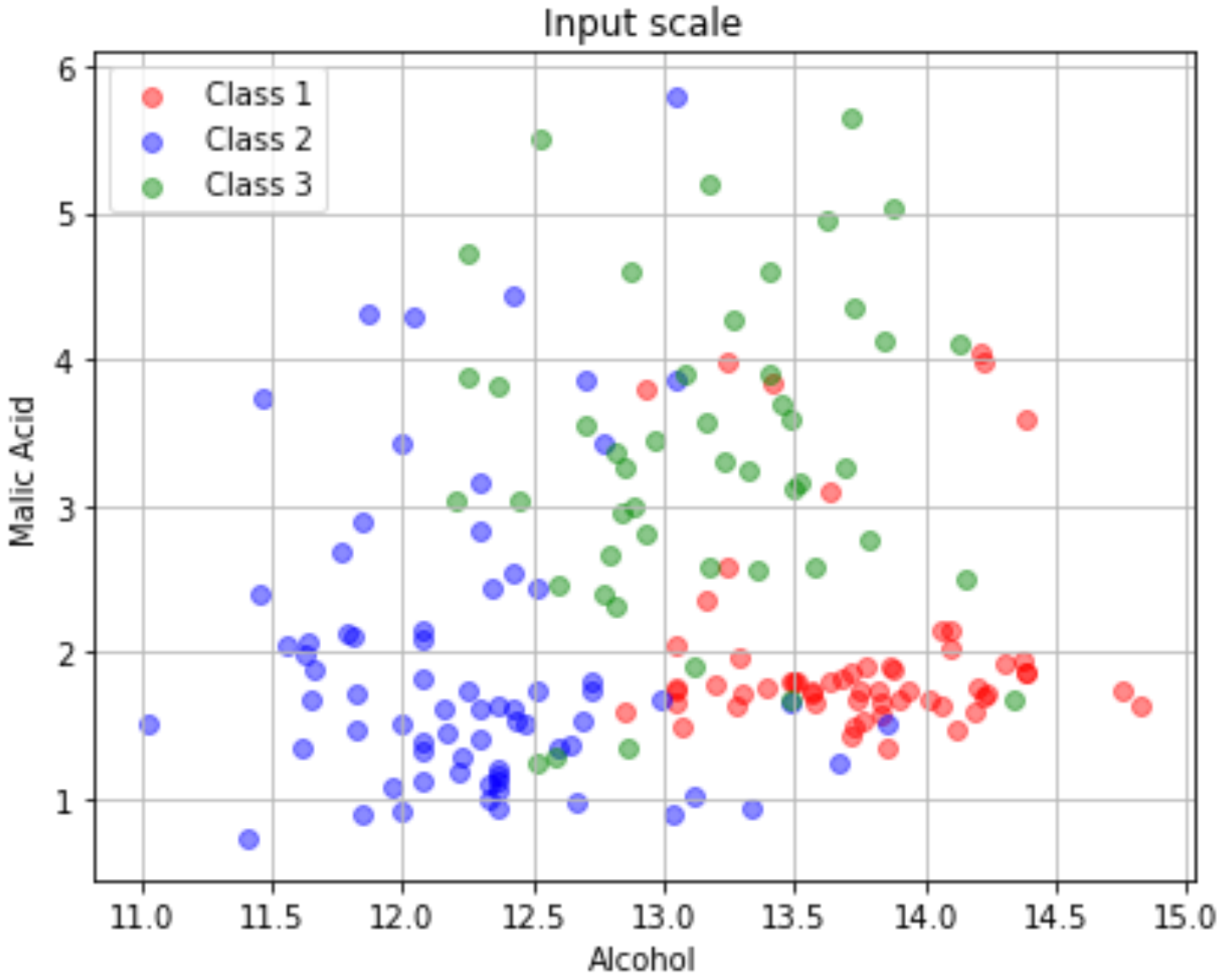
Feature scaling with sklearn

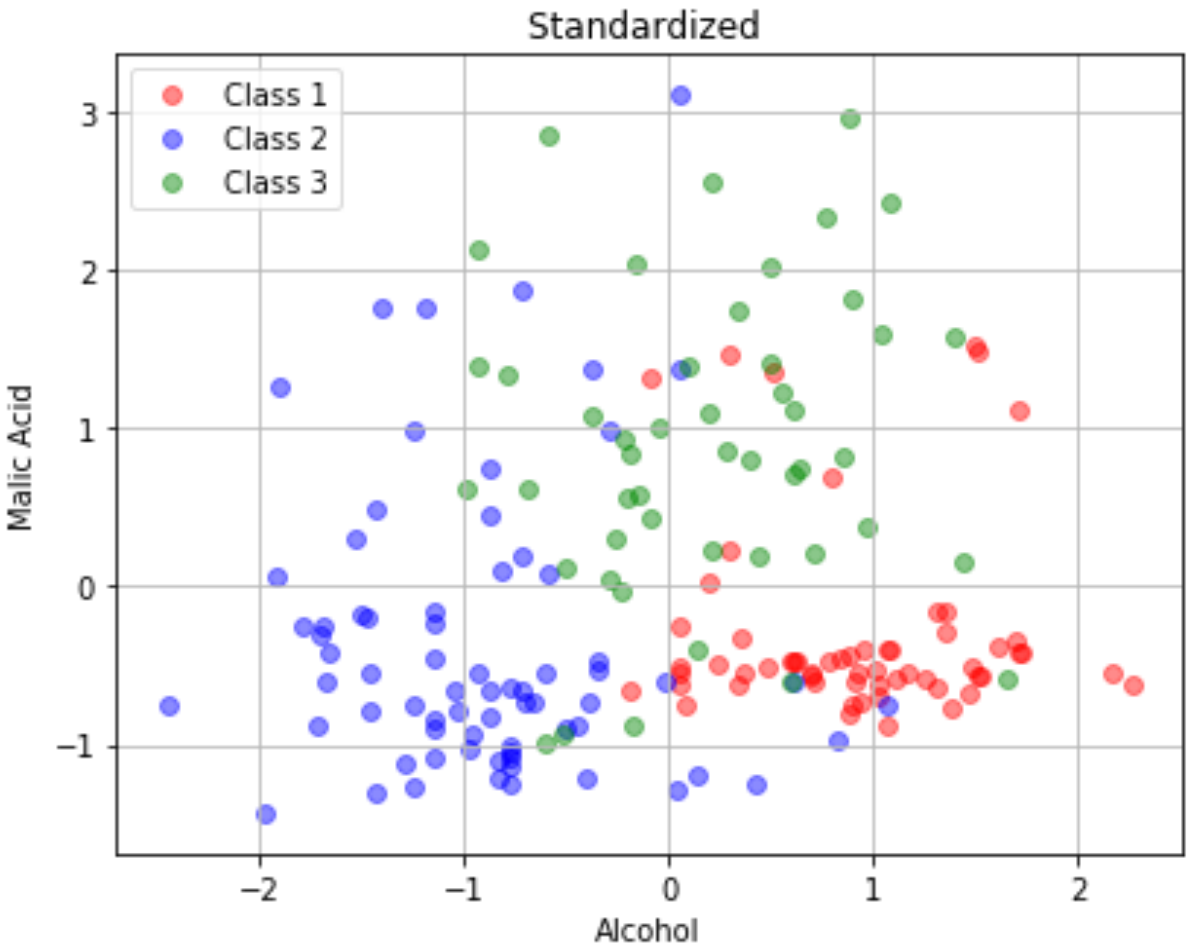
- Preprocessing은 모두 fit → transform의 과정을 거침
- 이유는 label encoder와 동일
- 단, scaler는 한번에 여러 column을 처리 가능

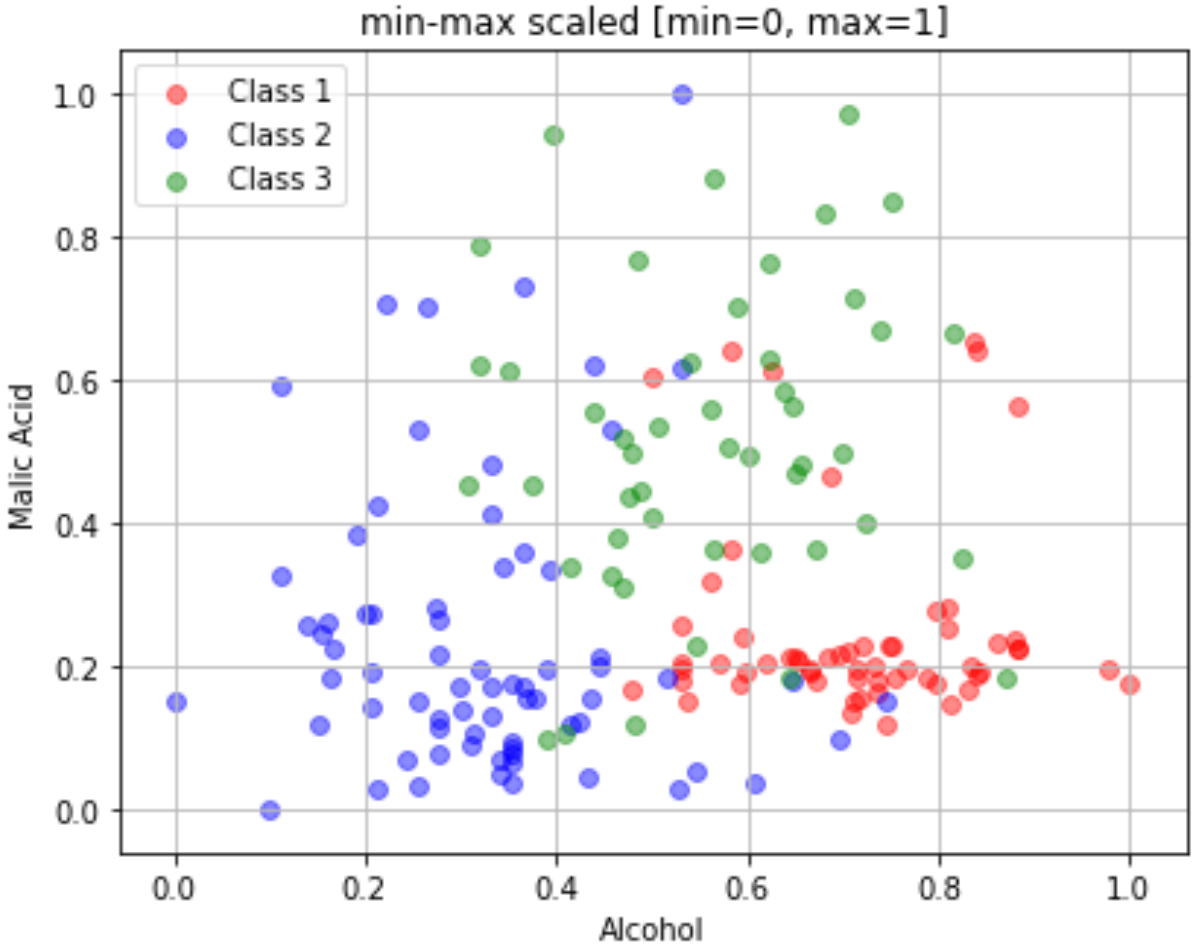
```
minmax_scale = preprocessing.MinMaxScaler().fit(df[['Alcohol', 'Malic acid']])
df_minmax = minmax_scale.transform(df[['Alcohol', 'Malic acid']])
df_minmax[:3]
```

```
array([[ 0.84210526,  0.1916996 ],
       [ 0.57105263,  0.2055336 ],
       [ 0.56052632,  0.3201581 ]])
```


Alcohol and Malic Acid content of the wine dataset
input scale
Standardized
min-max scaled [min=0, max=1]
Malic Acid
Alcohol
6
5
4
3
2
1
0
−1
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0


Input scale
Class 1
Class 2
Class 3
Malic Acid
Alcohol
6
5
4
3
2
1
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0


Standardized
Class 1
Class 2
Class 3
Malic Acid
Alcohol
3
2
1
0
−1
−2
−1
0
1
2


min-max scaled [min=0, max=1]
Class 1
Class 2
Class 3
Malic Acid
Alcohol
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Human knowledge belongs to the world.