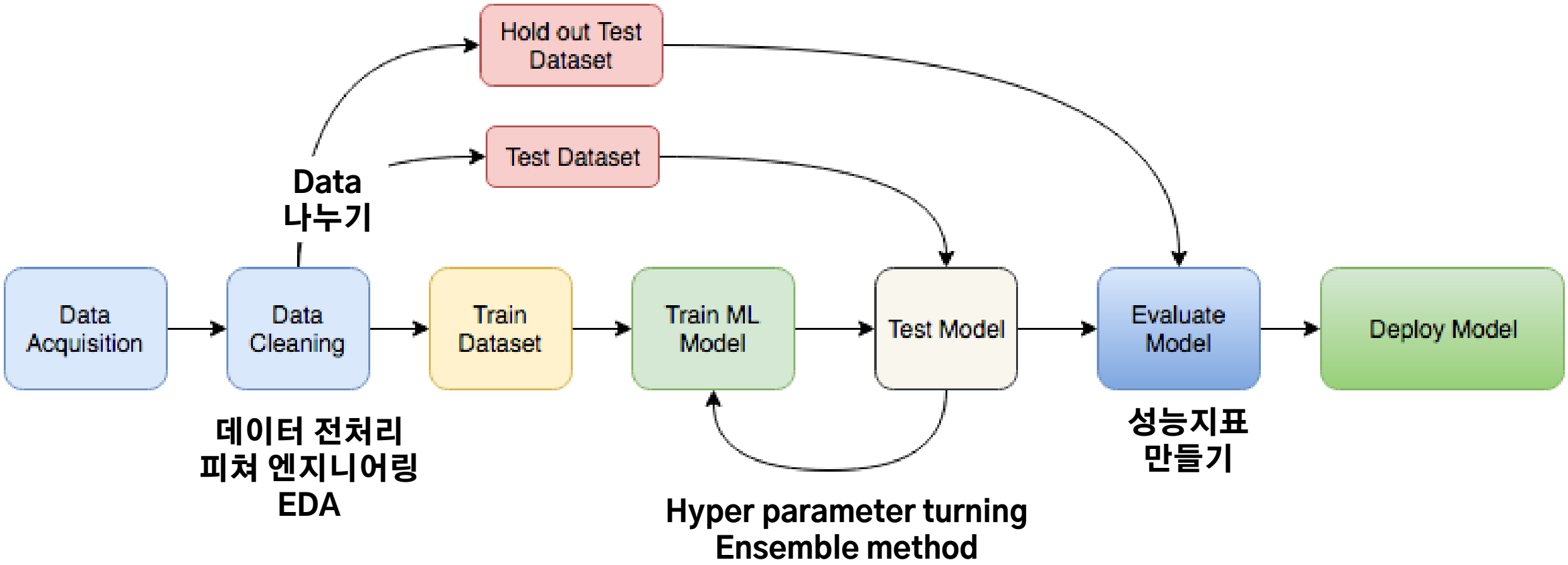
ML Process

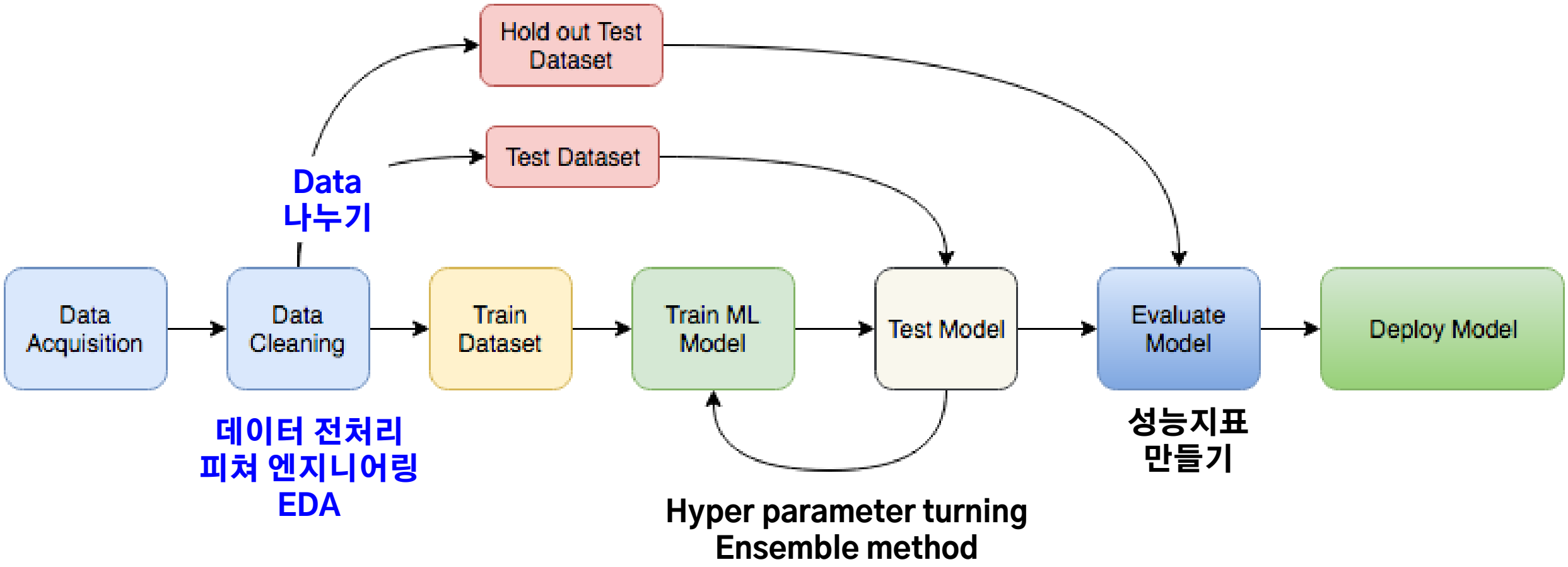
Guide to ML data prepration

Director of TEAMLAB
Sungchul Choi

ML Process


Hold out Test Dataset
Test Dataset
Data 나누기
Data Acquisition
Data Cleaning
Train Dataset
Train ML Model
Test Model
Evaluate Model
Deploy Model
데이터 전처리
피쳐 엔지니어링
EDA
Hyper parameter turning
Ensemble method
성능지표 만들기

ML Process


Hold out Test Dataset
Test Dataset
Data 나누기
Data Acquisition
Data Cleaning
Train Dataset
Train ML Model
Test Model
Evaluate Model
Deploy Model
데이터 전처리
피쳐 엔지니어링
EDA
Hyper parameter turning
Ensemble method
성능지표 만들기

ML을 하기 위해 해야 하는 것들!

- 데이터 나누기 : Train , Validation, Test
- 데이터 전처리 하기
    - 탐색적 자료 분석
    - 피쳐 엔지니어링

Machine Learning SYSTEM

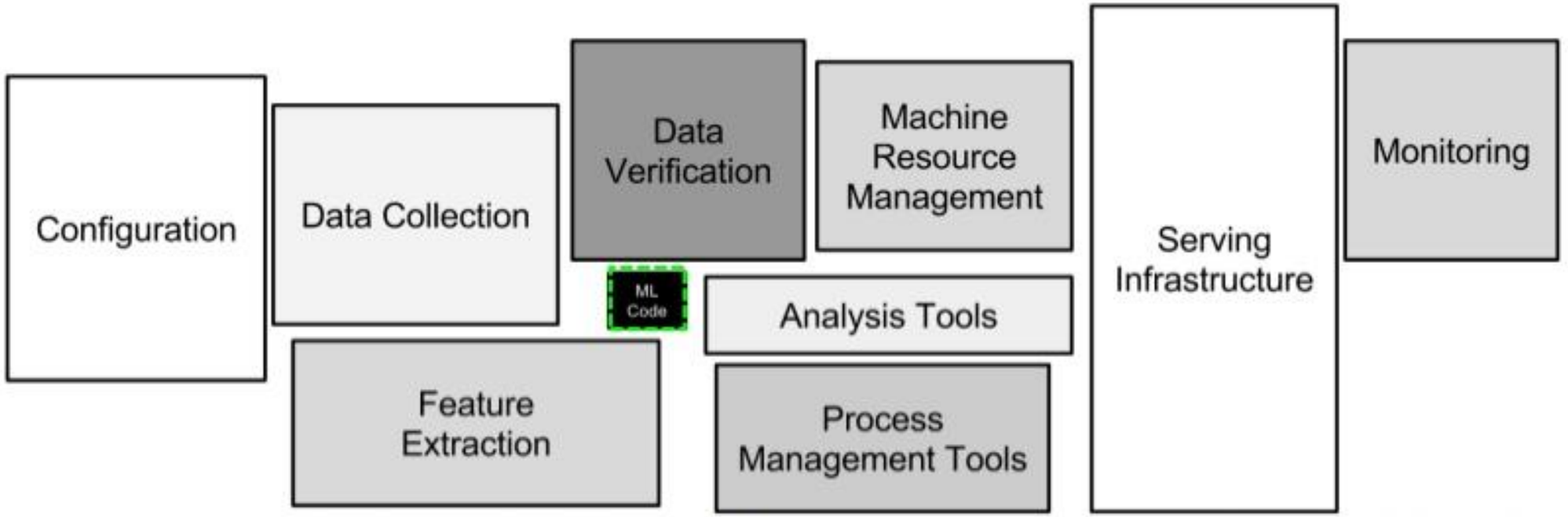


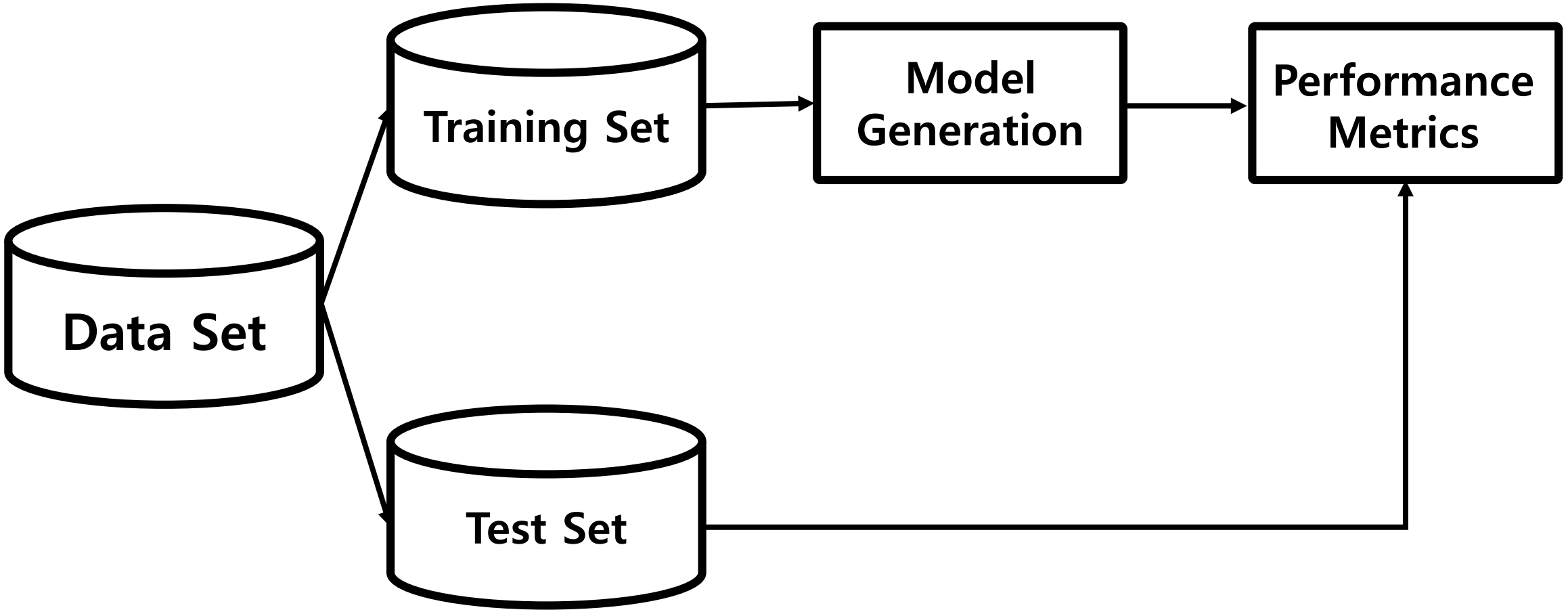
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Hidden Technical Debt in Machine Learning Systems (D Scully et al @Google)

Data Acquisition

From Kaggle

General ML Process
Training / Test Set


Data Set
Training Set
Test Set
Model Generation
Performance Metrics


train.csv
y
test.csv

일단은 한판으로 만든다!

데이터 모판!

Why 데이터 모판!

- Train과 Test 데이터 셋의 각 컬럼에는 같은 전처리 적용
- 데이터에 따라 Train에만 있고 Test에는 존재하지 않음

  → 초기 데이터 전처리시 규칙을 만들어야 함
- 데이터의 분포를 좀 더 넓게 볼 수 있음 (시계열?)

실제 서비스에서는 모델생성시
사용한 전처리를 그대로 활용하여 함

```
DATA_DIR = './titanic'
data_files = reversed(
     [os.path.join(DATA_DIR, filename) \
      for filename in os.listdir(DATA_DIR)])
df_list = []
for filename in data_files:
     df_list.append(pd.read_csv(filename))
df = pd.concat(df_list, sort=False)
df = df.reset_index(drop=True)
```

모판에서 y데이터 만 제거

Train과 Test 데이터의 위치는 기억

```
number_of_train_dataset = df.Survived.notnull().sum()
number_of_test_dataset = df.Survived.isnull().sum()
y_true = df.pop("Survived")[:number_of_train_dataset]
```

Data Preprocessing

데이터는 깨끗한가?

더럽다

어떻게 할까?

깨끗이 하고, 좋게하고

Data Cleansing

Feature Engineering

Exploratory Data Analysis


Data Cleansing
Feature Engineering
Exploratory Data Analysis

데이터 처리의 전략

- 모판은 흔들지 않는다
- 하나의 셀은 다시 실행해도 그 결과가 보장되야 한다.
- 전처리가 완료후 함수화한다 (merge 함수 필수)
- 컬럼이름은 list로 관리하기! 직접 입력 X
- 데이터는 타입별로 분리해서 관리하기!
- 데이터 노트 작성하기!!!

데이터 노트

- 데이터에 대한 처리 내용 및 방향을 정리한 노트
- 기본적인 전처리 방향과 방법들을 정리함
- 데이터에 대한 아이디어를 정리와 지속적인 업데이트

기본적인 데이터 현황 파악 코드

```
df.dtypes
df.info()
df.isnull().sum()
df.describe()
df.head(2).T
```

Data Cleansing issues

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

Missing Value

Missing Value Strategy

- 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)
```

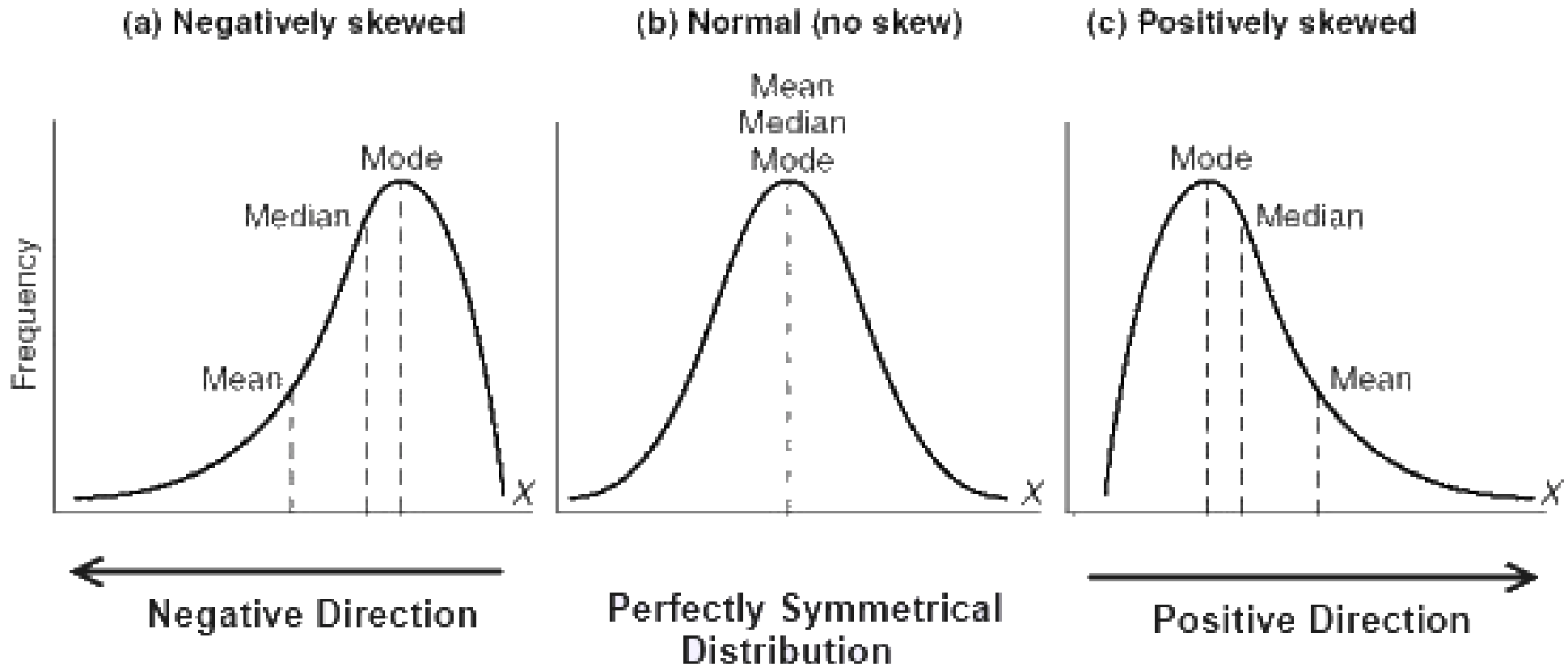
5개 이상 데이터가 있지 않으면 Drop

	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

Missing Value Handling

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

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age            263
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin         1014
Embarked         2
dtype: int64
```

```
pd.options.display.float_format = '{:.2f}%'.format
df.isnull().sum() / len(df) * 100
```

```
PassengerId     0.00%
Pclass          0.00%
Name            0.00%
Sex             0.00%
Age            20.09%
SibSp           0.00%
Parch           0.00%
Ticket          0.00%
Fare            0.08%
Cabin          77.46%
Embarked        0.15%
dtype: float64
```

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

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])
```

Label encoding by sklearn

- 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.],
```

What we did

- Category data encoding – one-hot encoding
- Missing value handling
- Data drop
- Log transformation
- Data binning


Data Cleansing
Feature Engineering
Exploratory Data Analysis

What we will do

- Encoding Families
- Feature Interactions
- Scaling
- Feature Selection
- Data binning

Feature Engineering

Feature

Feature Engineering

가정 적합한 특성을 찾는 것

Feature engineering

Generation

- Binarization, Quantization
- Scaling (normalization)
- Interaction features
- Log transformation
- Dimension reduction
- Clustering

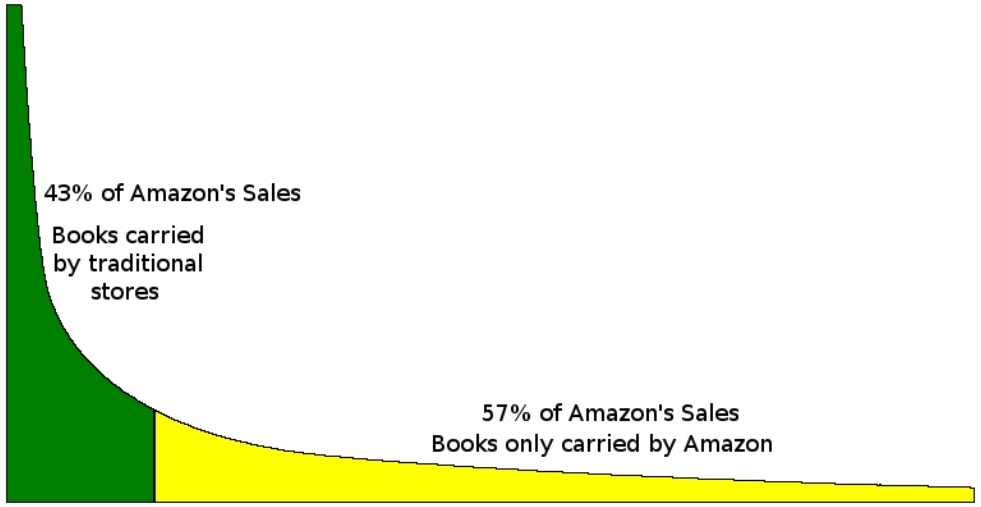
Selection

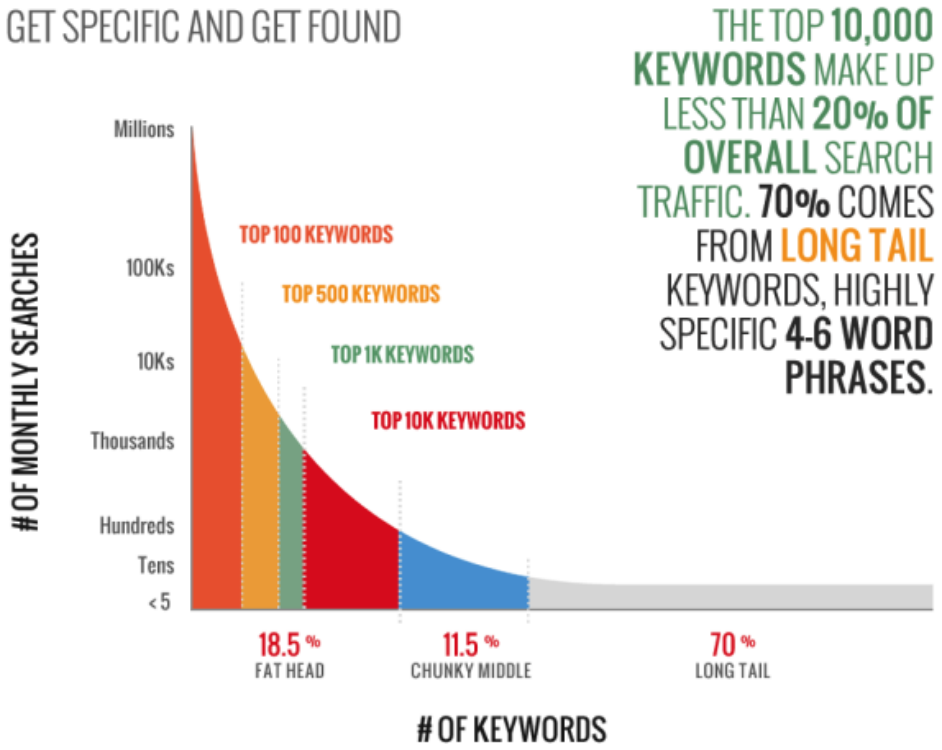
- Univariate statics
- Model-based selection
- Iterative feature selection
- Feature removal

Log transformations

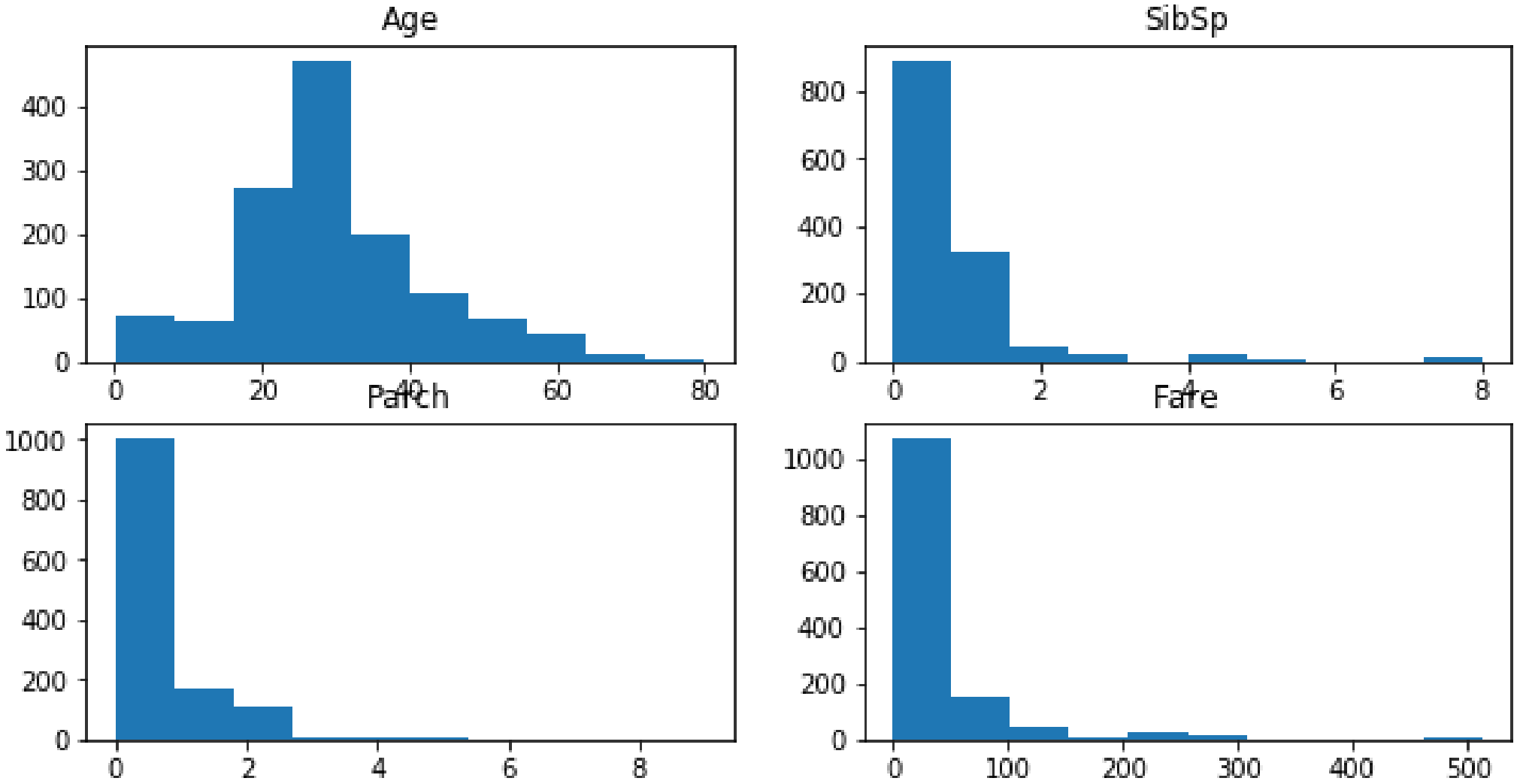
- 데이터의 분포가 극단적으로 모였을 때(poisson)
- 선형 모델은 데이터가 정규분포때 적합
- Poisson → Normal distribution
- 로그인 카운트, 제품 판매량, 검색 단어, 친구수
- np.log or np.exp 등의 함수를 사용

Log transformations



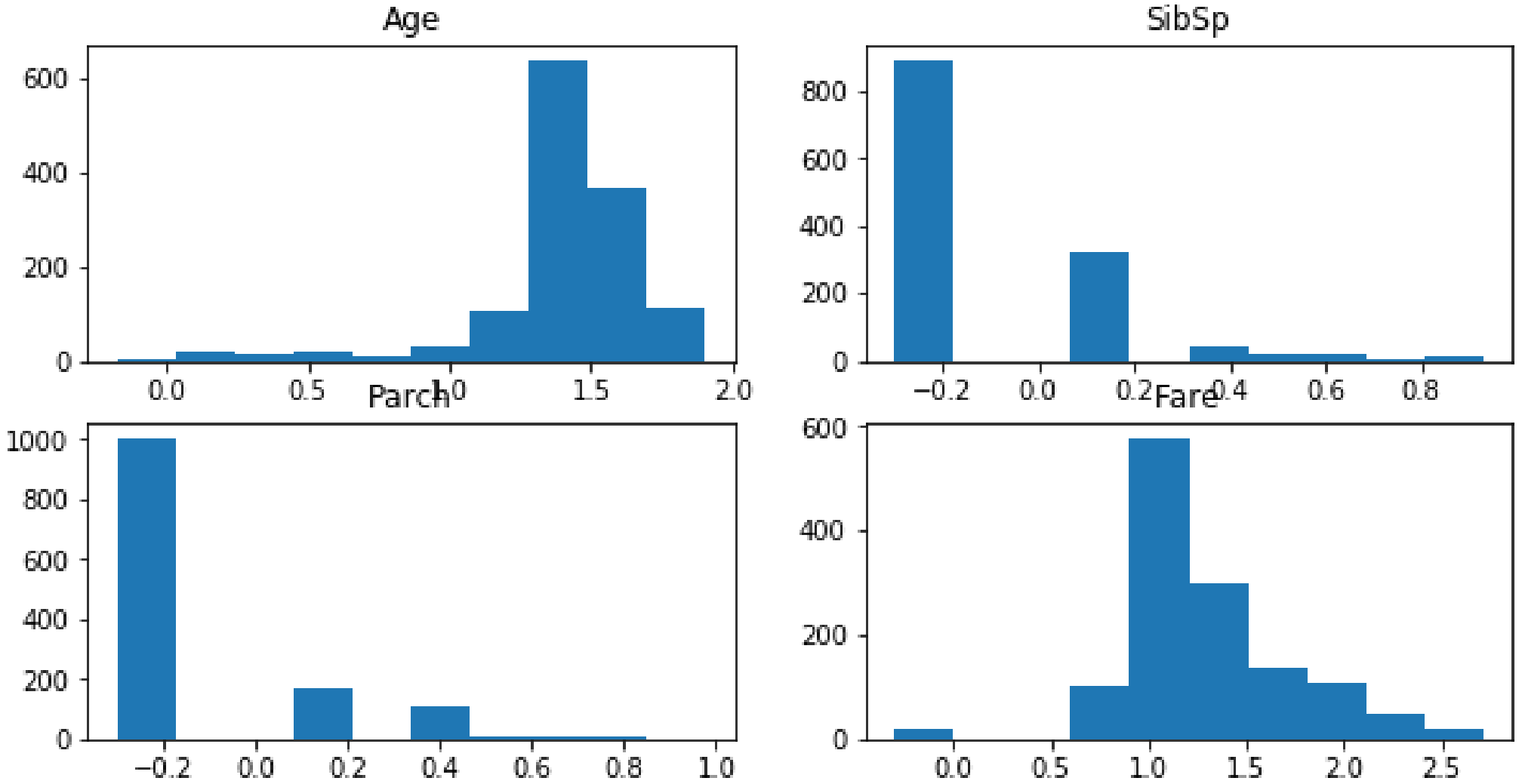


Log transformations



```
fig = plt.figure()
fig.set_size_inches(10,5) # 싸이즈 설정


ax = []
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
for i, col_name in enumerate(numeric_columns):
    ax.append(fig.add_subplot(2,2,i+1))
    X_1 = np.log10(one_hot_df[col_name]+0.5)


    ax[i].hist(X_1)
    ax[i].set_title(col_name)
```

Log transformations



Mean encoding

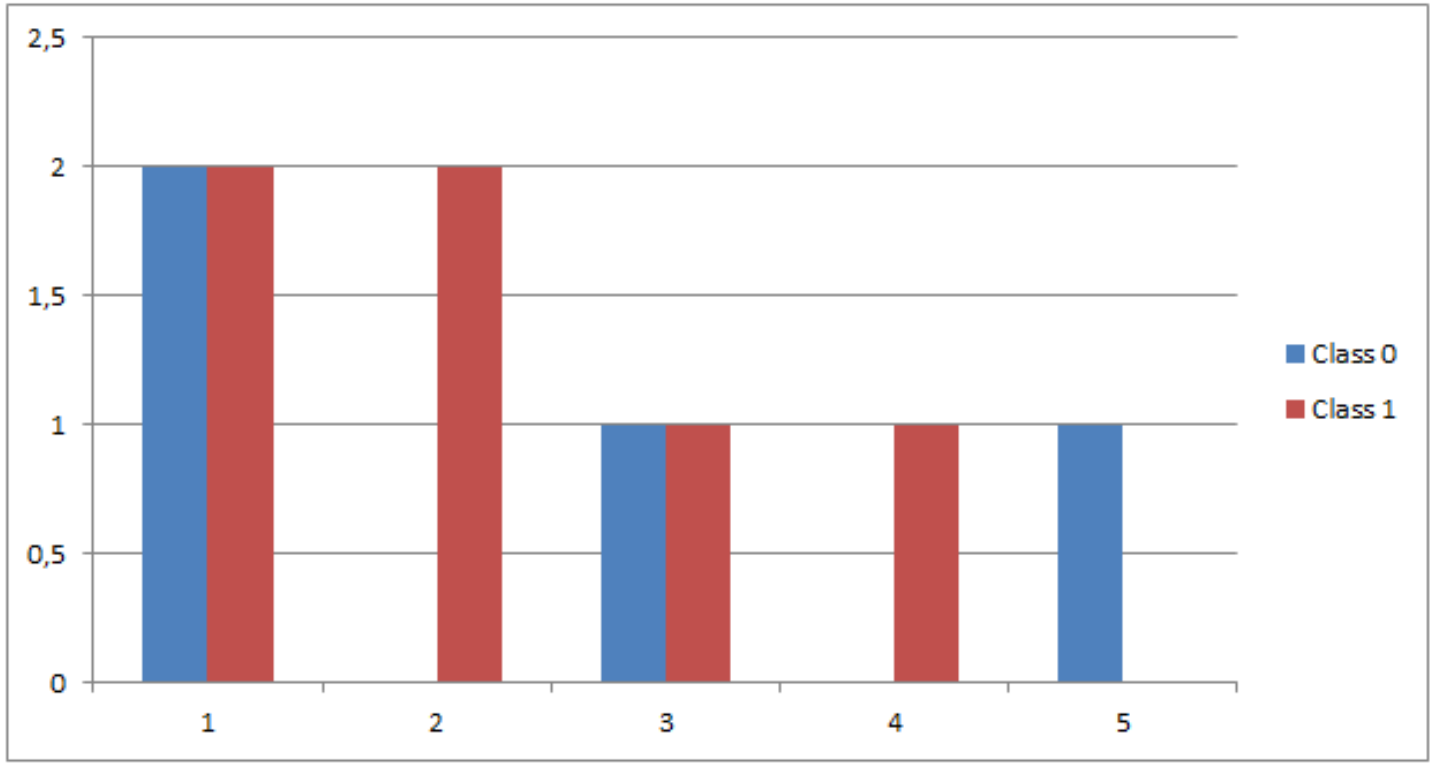
- Category 데이터는 항상 One-hot Encoding?
  → X, 다양한 인코딩 기법이 있음

- 대표적인 방법으로 Y값에 대한 분포를 활용한 Mean Encoding이 사용됨

Mean encoding

- Label 인코딩은 그 자체로 정보가 존재하지 않음

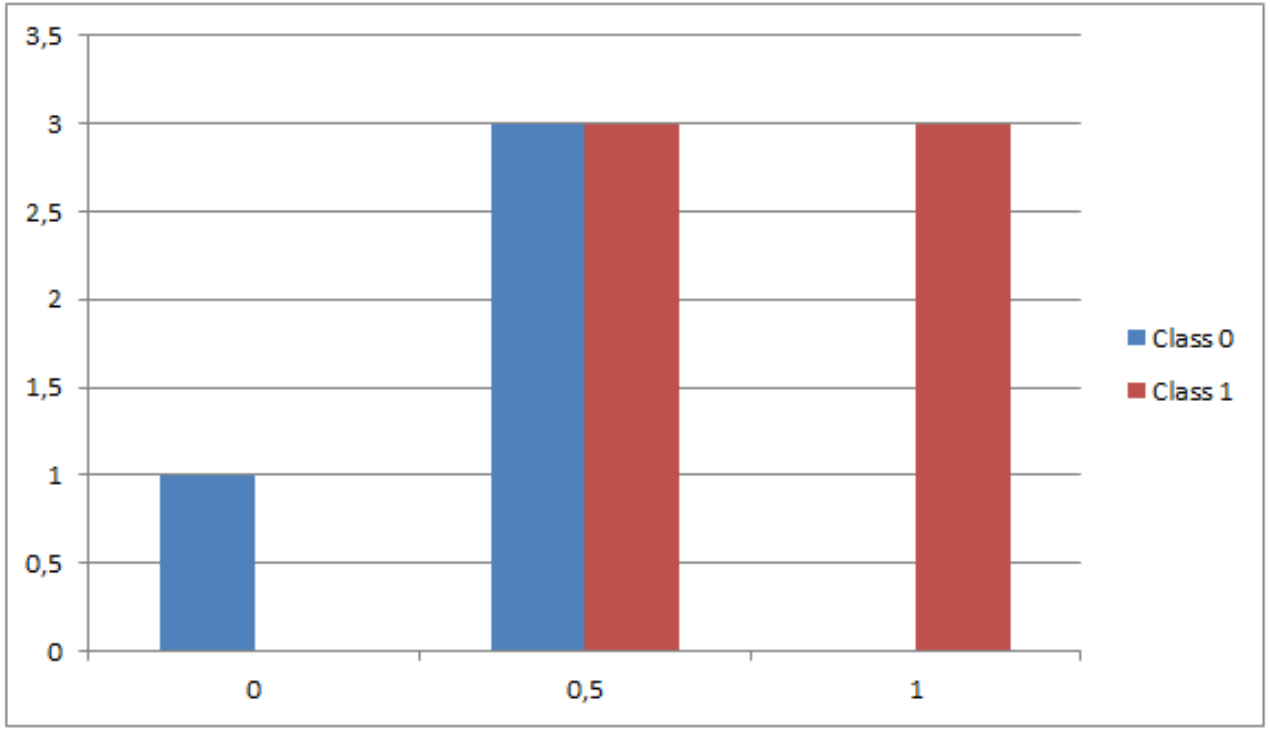
id	job	job_label	target
1	Doctor	1	1
2	Doctor	1	0
3	Doctor	1	1
4	Doctor	1	0
5	Teacher	2	1
6	Teacher	2	1
7	Engineer	3	0
8	Engineer	3	1
9	Waiter	4	1
10	Driver	5	0



Mean encoding

- Mean 인코딩: 분포의 값을 취할 수 있음

id	job	job_mean	target
1	Doctor	0,50	1
2	Doctor	0,50	0
3	Doctor	0,50	1
4	Doctor	0,50	0
5	Teacher	1	1
6	Teacher	1	1
7	Engineer	0,50	0
8	Engineer	0,50	1
9	Waiter	1	1
10	Driver	0	0



Mean encoding

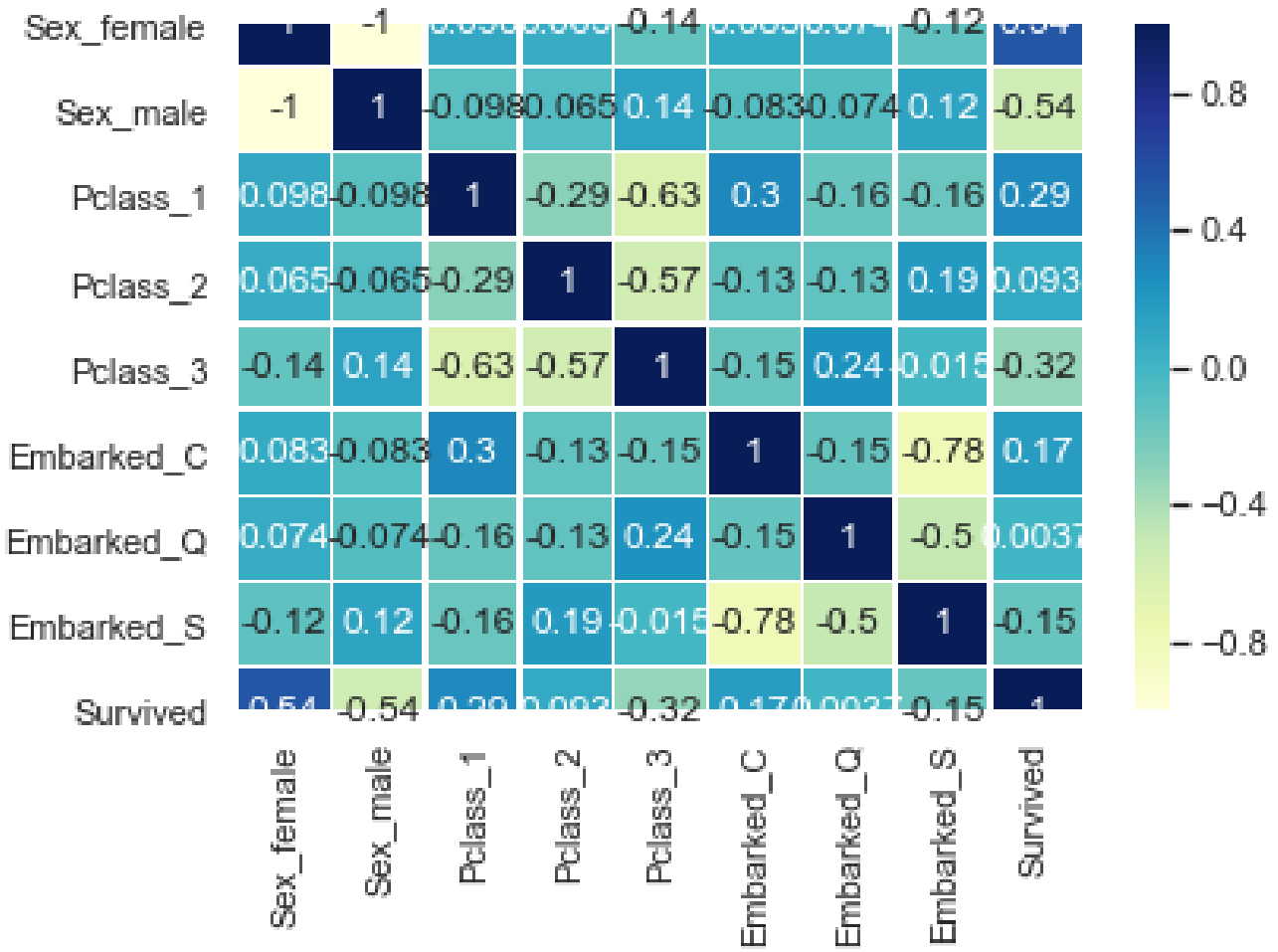
```
temp_df = pd.merge(
    one_hot_df["Pclass"], y_true,
        left_index=True, right_index=True)
temp_df.groupby("Pclass")["Survived"].mean()
```

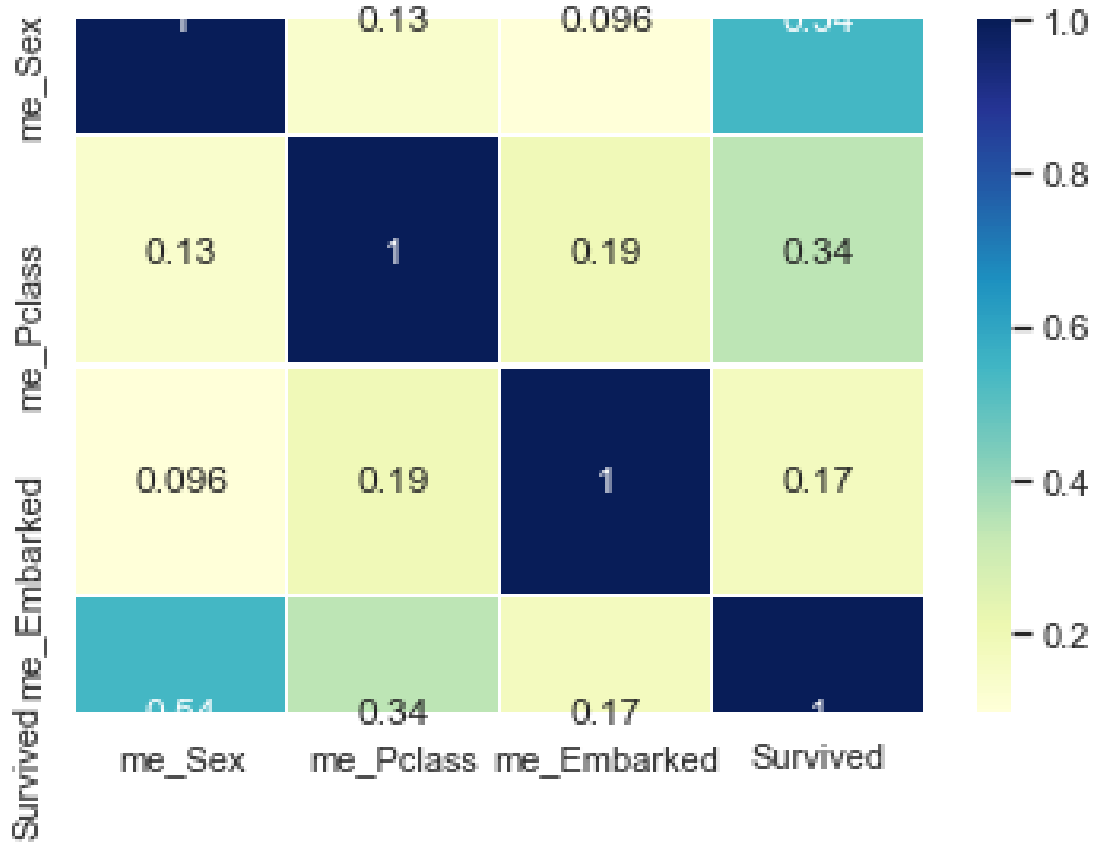
```
Pclass
1    0.63
2    0.47
3    0.24
Name: Survived, dtype: float64
```

```
temp_df["Pclass"].replace(
    temp_df.groupby("Pclass")["Survived"].mean())
```

```
0      0.24
1      0.63
2      0.24
3      0.63
4      0.24
       ...
886    0.47
887    0.63
888    0.24
889    0.63
890    0.24
Name: Pclass, Length: 891, dtype: float64
```

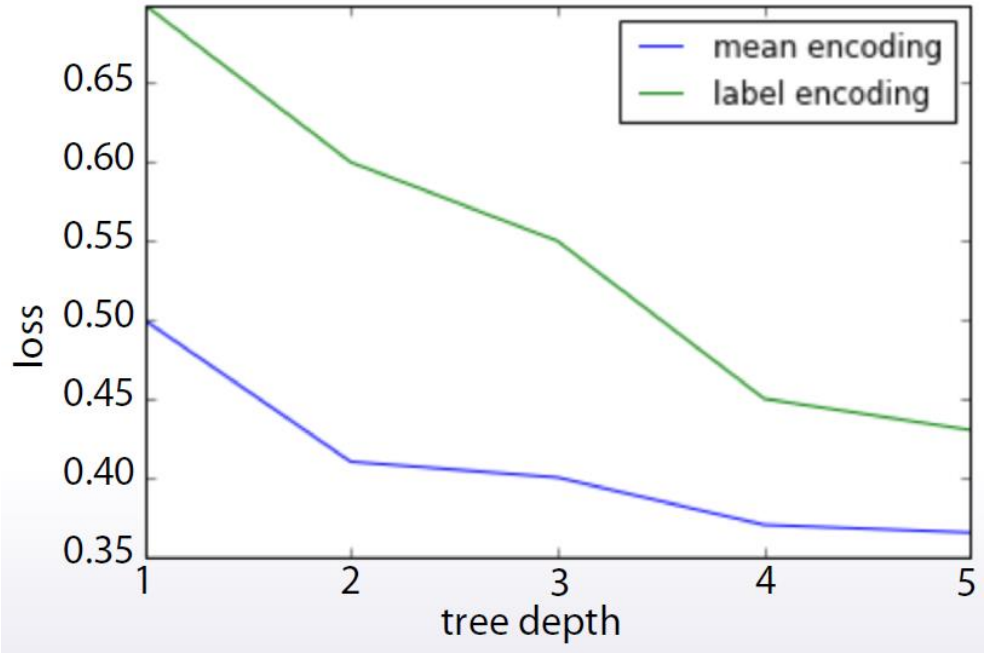
Mean encoding

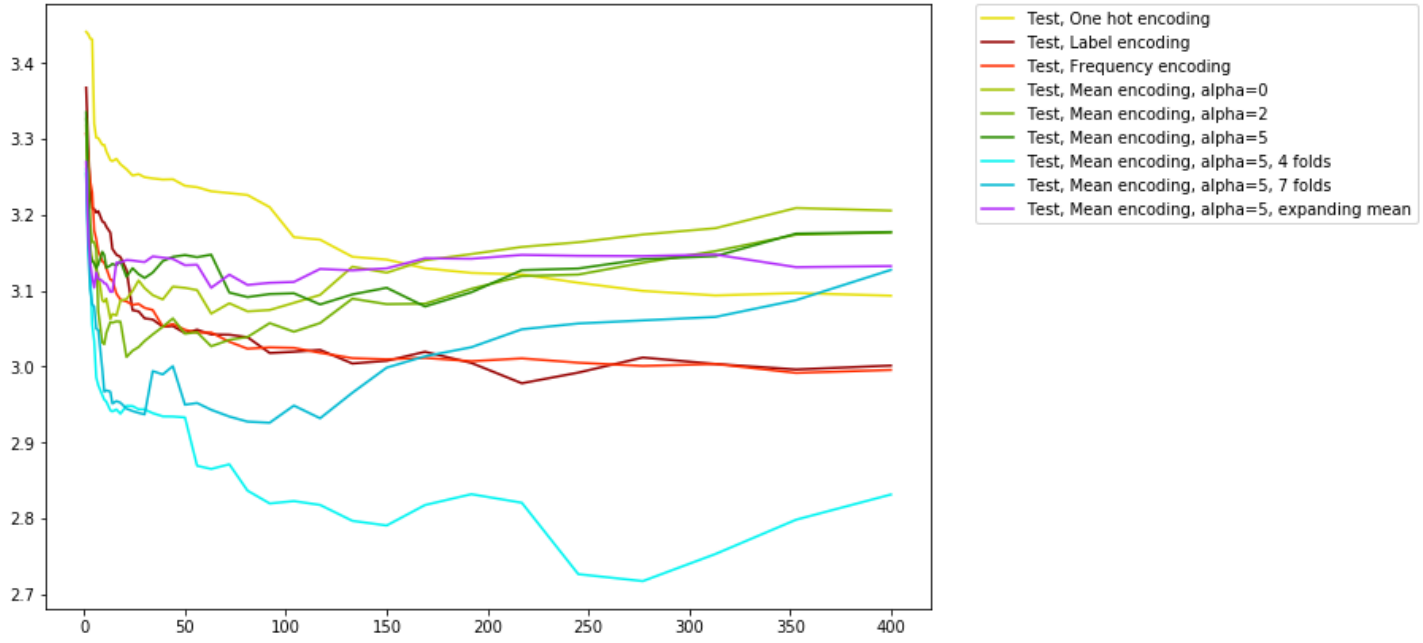




Mean encoding

- 조금 더 빨리 , 조금 더 나은 성능이 나오기 도 함





Mean encoding

- Regression Task는 단순 평균값으로 입력

$$label_c = p_c$$

https://www.kaggle.com/vprokopev/mean-likelihood-encodings-a-comprehensive-study

Mean encoding

- Overfitting을 제거하기 위해 smoothing을 사용함

$$label_c = \frac{(p_c * n_c + p_{global} * \alpha)}{(n_c + \alpha)}$$

```
def calc_smooth_mean(df, by, on, m):
    # Compute the global mean
    mean = df[on].mean()

    # Compute the number of values and the mean of each
    agg = df.groupby(by)[on].agg(['count', 'mean'])
    counts = agg['count']
    means = agg['mean']

    # Compute the "smoothed" means
    smooth = (counts * means + m * mean) / (counts + m)

    # Replace each value by the according smoothed mean
    return df[by].map(smooth)
```

https://www.kaggle.com/vprokopev/mean-likelihood-encodings-a-comprehensive-study

Mean encoding

- 이외에도 많은 Encoding 기법들이 존재함

Encoding Methods

- Backward Difference Contrast [2][3]
- BaseN [6]
- Binary [5]
- Count [10]
- Hashing [1]
- Helmert Contrast [2][3]
- James-Stein Estimator [9]
- LeaveOneOut [4]
- M-estimator [7]
- Ordinal [2][3]
- One-Hot [2][3]
- Polynomial Contrast [2][3]
- Sum Contrast [2][3]
- Target Encoding [7]
- Weight of Evidence [8]

https://github.com/scikit-learn-contrib/categorical-encoding?fbclid=IwAR3b4X2XUuMJWuH0LxTs9Hf4rAzHeS6W-q3DegG1kuZwhKhZejTmznG_nvM

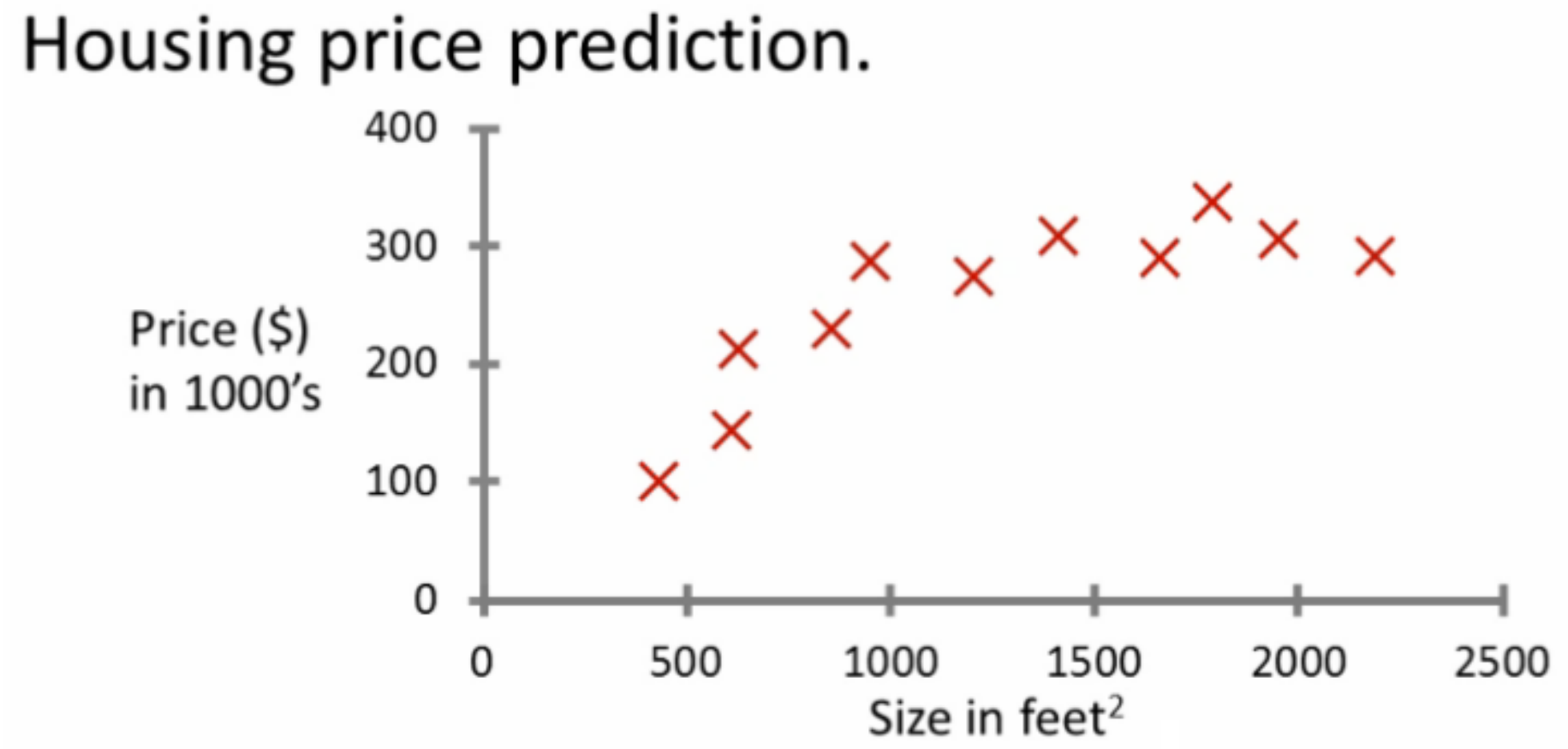
Interaction features

- 기존 feature들의 조합으로 새로운 feature를 생성
- Data에 대한 사전 지식과 이해가 필요
- Polynomial feature를 사용한 자동화 가능 → 높은 비용

sklearn.preprocessing.PolynomialFeatures

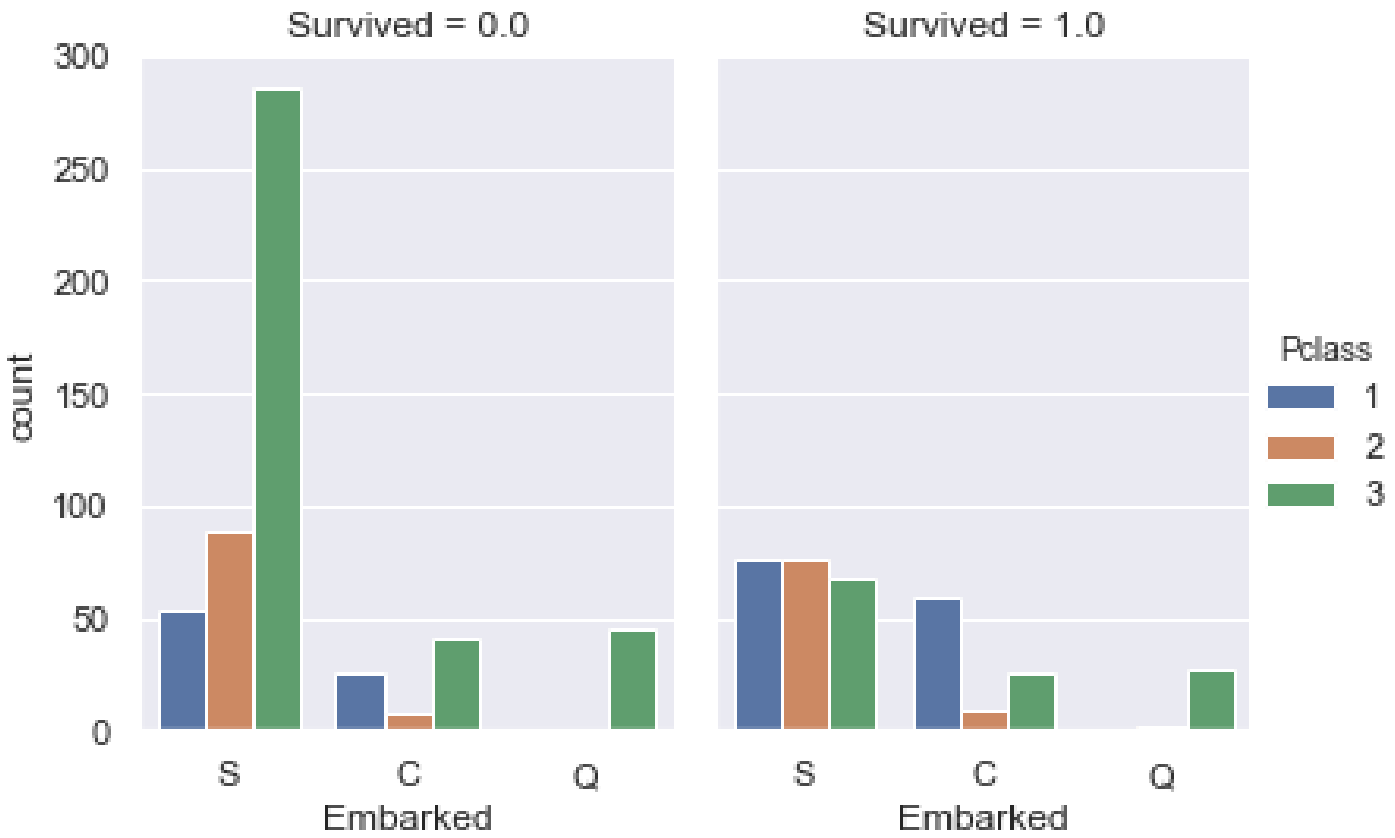
- 실험적으로 접근할 요소들은 있음 → 자동화 코드 중요
- weight + time-period, sensor1 + sensor2

Interaction features



Interaction features

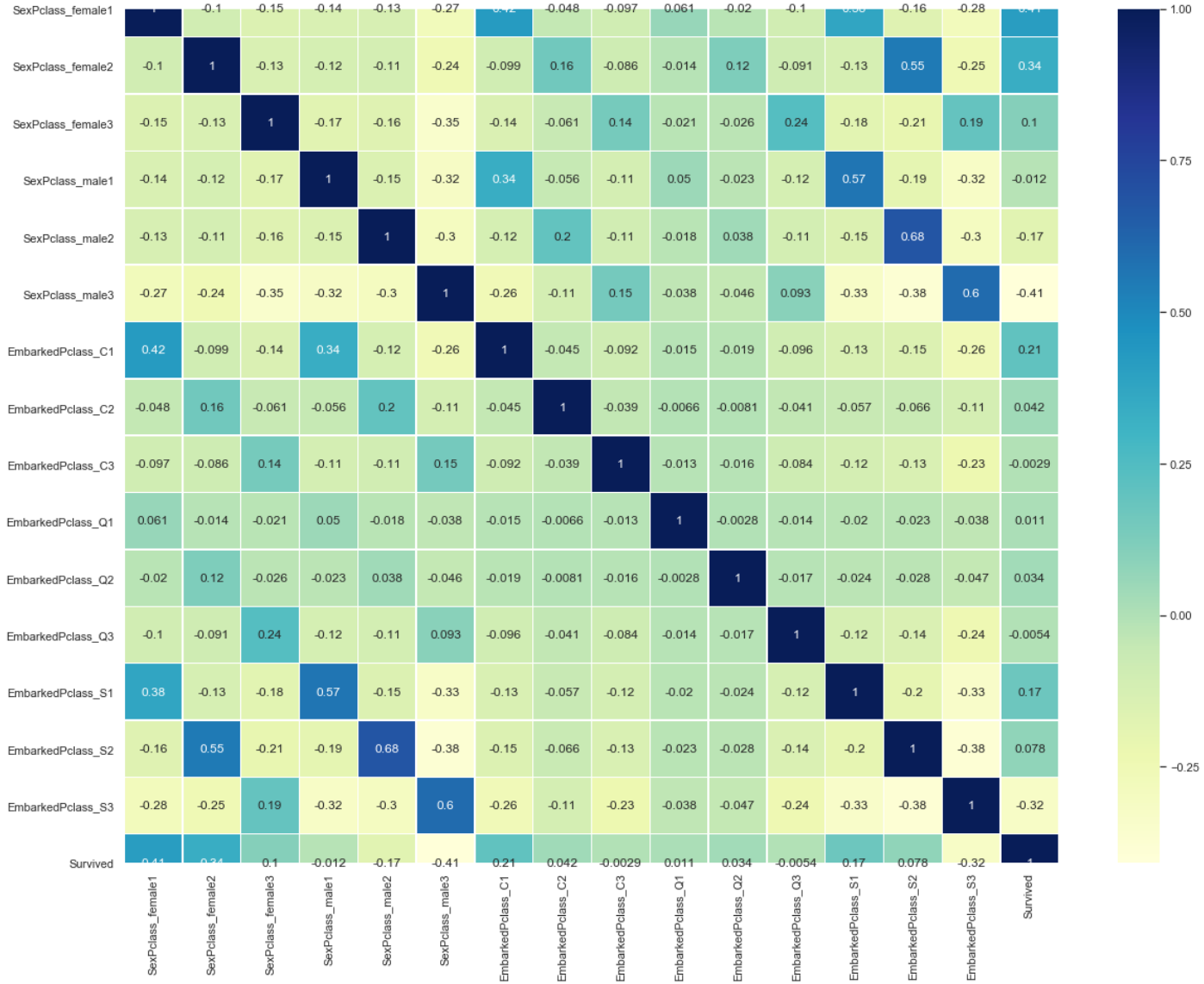
- Category Combination



Interaction features

```
temp_columns = ["Sex", "Pclass", "Embarked"]
one_hot_df["Sex-Pclass"] =
df["Sex"].map(str)+df["Pclass"].map(str)
one_hot_df["Embarked-Pclass"] =
    df["Embarked"].map(str)+df["Pclass"].map(str)
```

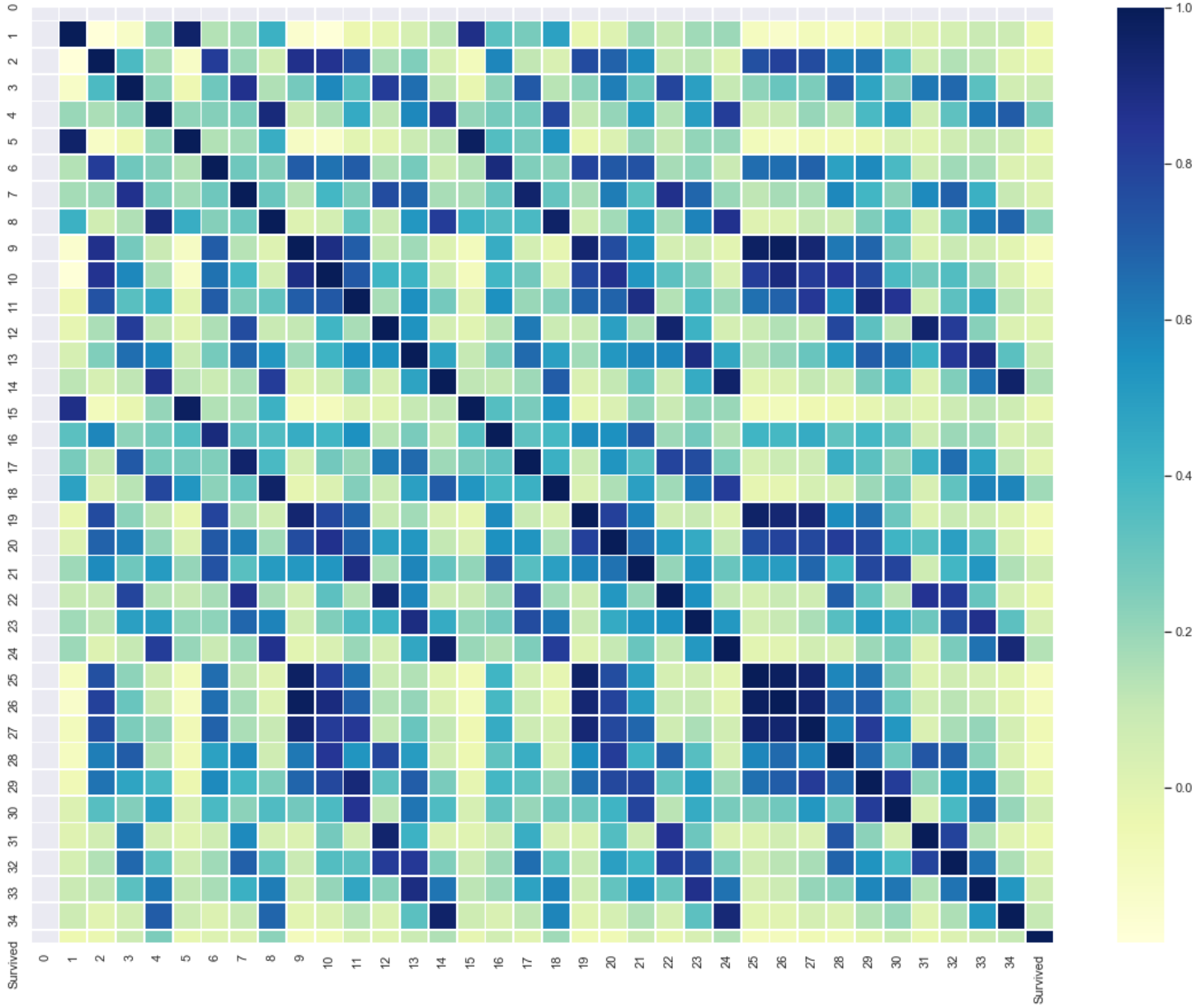
Interaction features



Interaction features

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2)
X_poly = pd.DataFrame(
    poly_features.fit_transform(
    log_bin_one_hot_df[numeric_columns]))
```

Interaction features



Etc

- Feature 끼리 더하기, 곱하기, 나누기 등등
- 왜 잘 되는지 모르는데 잘 되는 경우가 있음
- 도메인 지식과 EDA로 좋은 Feature들을 생성해야함

Feature Selection

가정 적합한 특성을 선택하는 방법

Feature engineering

Generation

- Binarization, Quantization
- Scaling (normalization)
- Interaction features
- Log transformation
- Dimension reduction
- Clustering

Selection

- Univariate feature selection
- Model-based selection
- Iterative feature selection
- Feature removal

Feature selection

- 모든 feature 들이 반드시 model 학습에 필요치 않음
- 어떤 feature들은 성능을 오히려 나쁘게 함
- 너무 많은 feature → overfitting의 원인
- 모델에 따라서 필요한 feature를 선택함
- 필요없는 feature 제거 → 학습 속도와 성능 향상
- 다양한 기법과 코드에 대해 공부

Univariate feature selection

- 통계 모델을 기반으로 한 최적의 feature를 선택
- Chi square, F-test, ANOVA 등의 통계 모델을 사용
- Y값과 하나의 feature간의 통계적 유의미를 분석
- 주로 선형 모델에서 유용하게 사용할 수 있음
- 빠르게 사용할 수 있는 feature selection 기법

SelectKBest

sklearn.feature_selection.SelectKBest¶

class sklearn.feature_selection. **SelectKBest** (*score_func=<function f_classif>, k=10*) [source]

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.feature_selection import chi2
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X.shape
(150, 4)
>>> X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
>>> X_new.shape
(150, 2)
```

SelectKBest

SelectKBest SelectPercentile

- For regression: f_regression, mutual_info_regression
- For classification: chi2, f_classif, mutual_info_classif

http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

`f_classif`

ANOVA F-value between label/feature for classification tasks.

`mutual_info_classif`

Mutual information for a discrete target.

`chi2`

Chi-squared stats of non-negative features for classification tasks.

`f_regression`

F-value between label/feature for regression tasks.

`mutual_info_regression`

Mutual information for a continuous target.

`SelectPercentile`

Select features based on percentile of the highest scores.

`SelectFpr`

Select features based on a false positive rate test.

`SelectFdr`

Select features based on an estimated false discovery rate.

`SelectFwe`

Select features based on family-wise error rate.

`GenericUnivariateSelect`

Univariate feature selector with configurable mode.

Model based feature selection

- 몇몇 모델들은 학습 과정에서 적절한 feature를 찾음
- L1 penalty, Tree-based model
- Feature importance를 기반으로 한 feature 선택이 가능
- 다른 모델의 feature 선택의 전처리 단계로 활용 가능
- 한번에 모든 feature를 고려함 → 시간 증가, 성능 향상
- Tree-based ensemble 계열은 이런 특징들이 이미 있음

sklearn.feature_selection.SelectFromModel

class sklearn.feature_selection. **SelectFromModel** (*estimator, threshold=None, prefit=False, norm_order=1*) [source]

Meta-transformer for selecting features based on importance weights.

```
select  = SelectFromModel(estimator=RandomForestRegressor(n_estimators=100), threshold="median")
```

```
select.fit(X_train, y_train)

# transform training set
X_train_selected = select.transform(X_train)
```

Iterative Feature Selection

- 반복적으로 feature의 수를 조절 → 최적 feature 선택
- 1개 → n개 , 또는 n개 → 1개 (Recursive Feature Elimination, RFE)
- 매우 높은 계산 비용, 성능 보장
- 회귀모델의 stepwise selection 기법이 존재(scikit-learn X)
- Tree 계열 모델을 사용 feature importance를 사용
- 데이터의 context를 모를 때, 사용하기 용이함

Recursive Feature Elimination

sklearn.feature_selection.RFE

Parameters:

estimator : object

A supervised learning estimator with a `fit` method that provides information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

n_features_to_select : int or None (default=None)

The number of features to select. If None, half of the features are selected.

step : int or float, optional (default=1)

If greater than or equal to 1, then step corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then step corresponds to the percentage (rounded down) of features to remove at each iteration.

verbose : int, default=0

Controls verbosity of output.

feature 선택의 주의 사항들

- prediction time에도 쓸 수 있는 feature 인가?
- 실시간 예측이 필요할 때, 생성이 너무 고비용이 아닌가?
- scale은 일정한가? 또는 비율적으로 표현 가능한가?
- 새롭게 등장하는 category data는? 가장 비슷한 것?
- 너무 극단적인 분포 → threshold 기반으로 binarization

feature 선택의 주의 사항들

- prediction time에도 쓸 수 있는 feature 인가?
- 실시간 예측이 필요할 때, 생성이 너무 고비용이 아닌가?
- scale은 일정한가? 또는 비율적으로 표현 가능한가?
- 새롭게 등장하는 category data는? 가장 비슷한 것?
- 너무 극단적인 분포 → threshold 기반으로 binarization

이런 Feature들은 삭제하자!

- Correlation 이 너무 높은 Feature는 삭제
- 전처리가 완료된 str feature들
- ID와 같은 성향을 가진 Feature 들

Feature scaling

두 변수중 하나의 값의 크기가 너무 크다!

몸무게와 키가 변수일때, 키가 영향을 많이 줌

Feature scaling

Feature간의 최대-최소값의 차이를 맞춘다!

$$y = \beta_1 x_1 + \beta_2 x_1 + b$$

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]
```

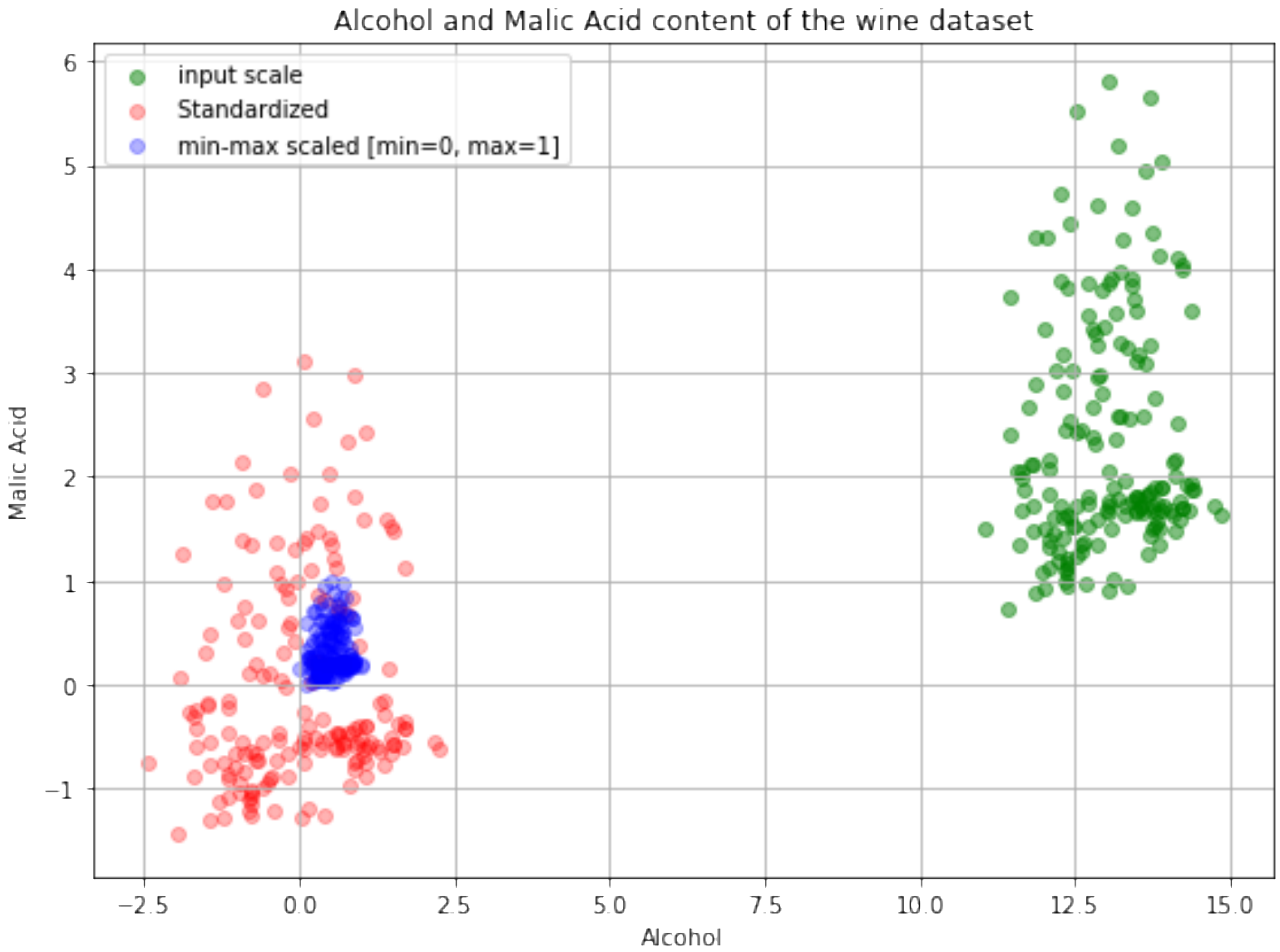
```
array([[ 1.51861254, -0.5622498 ],
       [ 0.24628963, -0.49941338],
       [ 0.19687903,  0.02123125],
```

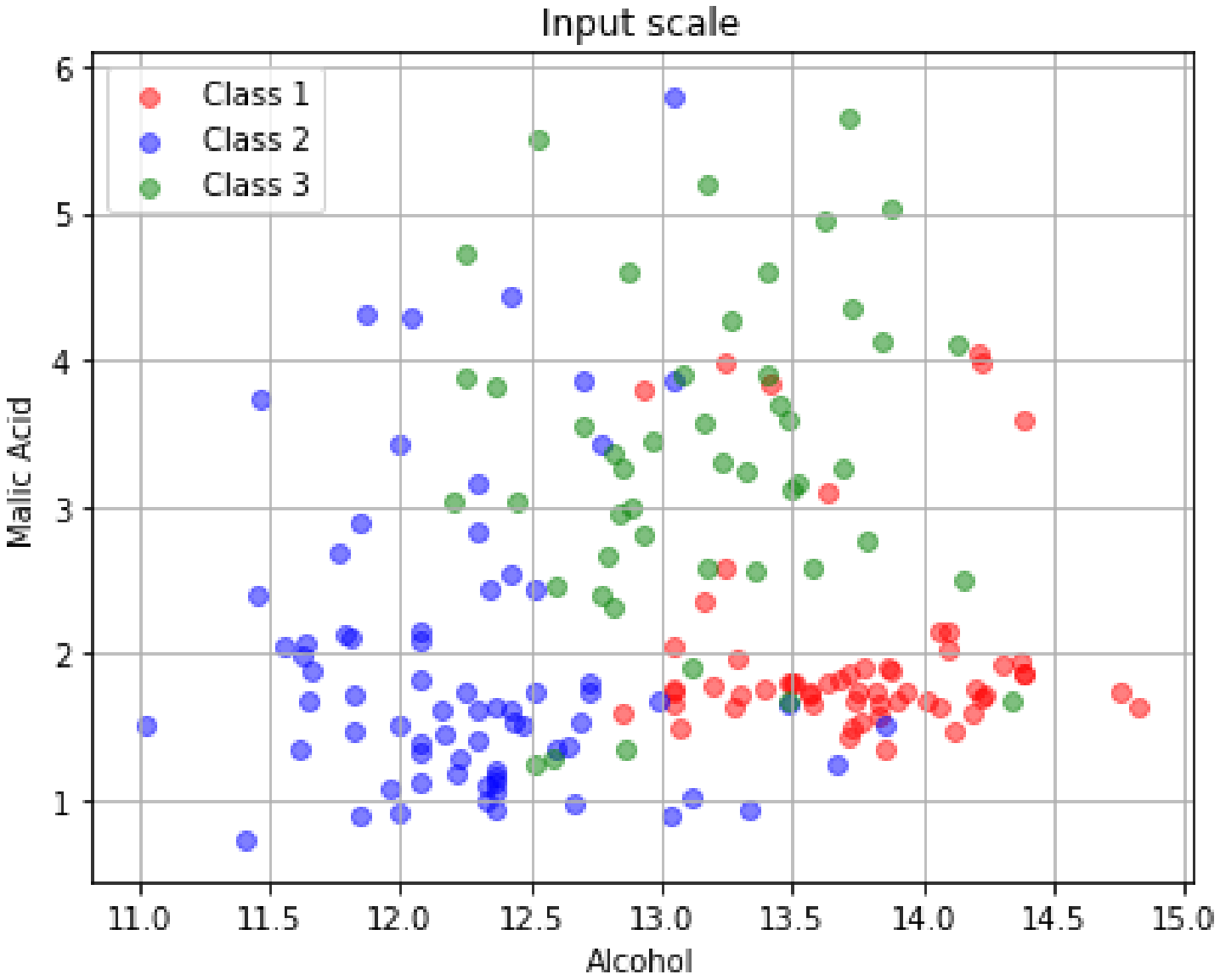
Feature scaling with sklearn

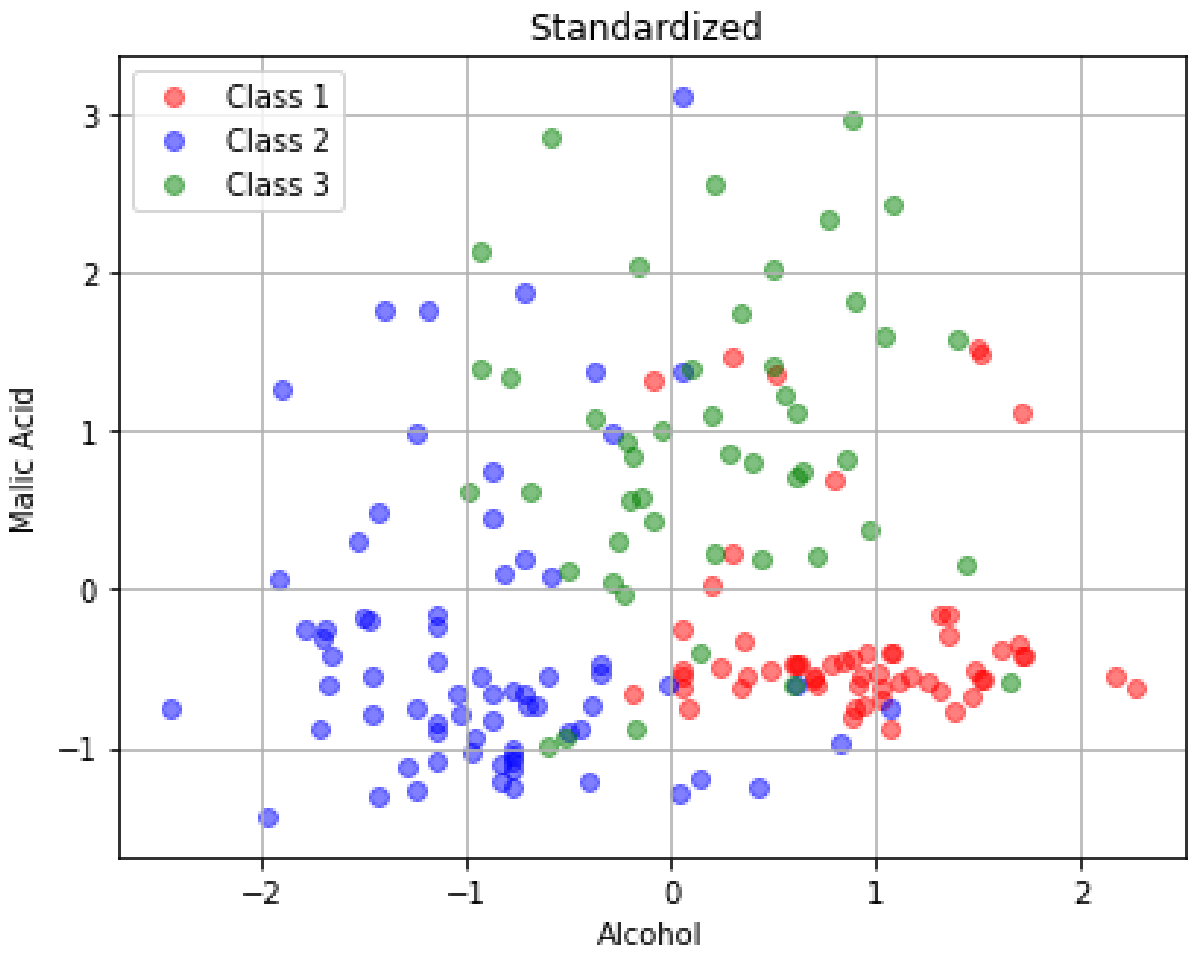
- Preprocessing은 모두 fit → transform의 과정을 거침
- 이유는 label encoder와 동일
- 단, scaler는 한번에 여러 column을 처리 가능

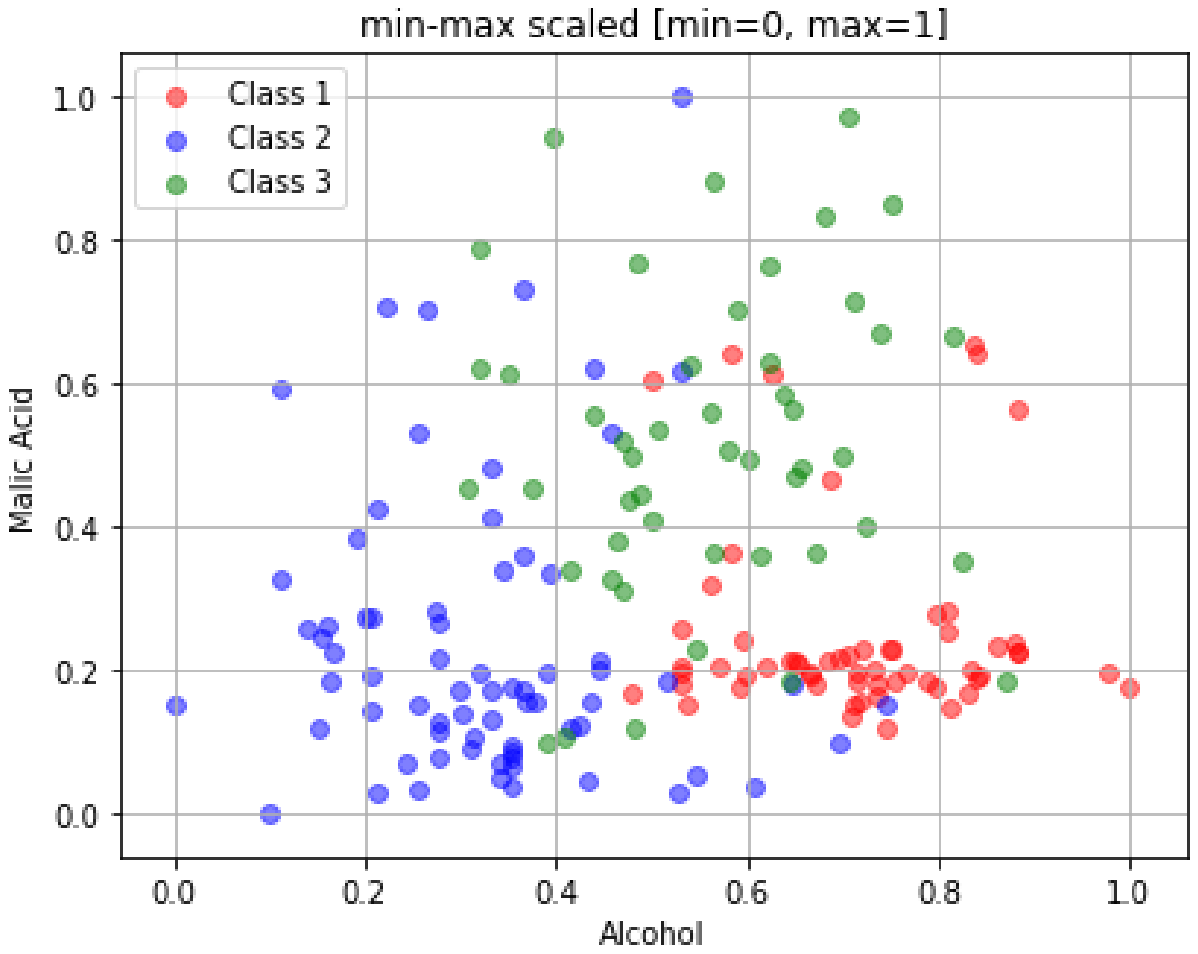
```
minmax_scale = preprocessing.MinMaxScaler().fit(df[['Alcohol', 'Malic acid']])
df_minmax = minmax_scale.transform(df[['Alcohol', 'Malic acid']])
df_minmax[:3]
```

```
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

Model & Trainning

데이터의 정리가 끝나면 학습 하는 방법

- 적합한 모델을 선정한다 (실험)
- 모델에 적합한 하이퍼 파라메터를 선정한다 (실험)
- 다양한 전처리 경우의 수를 입력한다 (실험)
- 학습을 실행한다.
- 성능을 평가한다.

데이터 numpy로 변환

```
X_train = all_df[:number_of_train_dataset].values
X_test = all_df[number_of_train_dataset:].values
y_train = y_true.copy()
```

모델을 선정하여 학습시키기

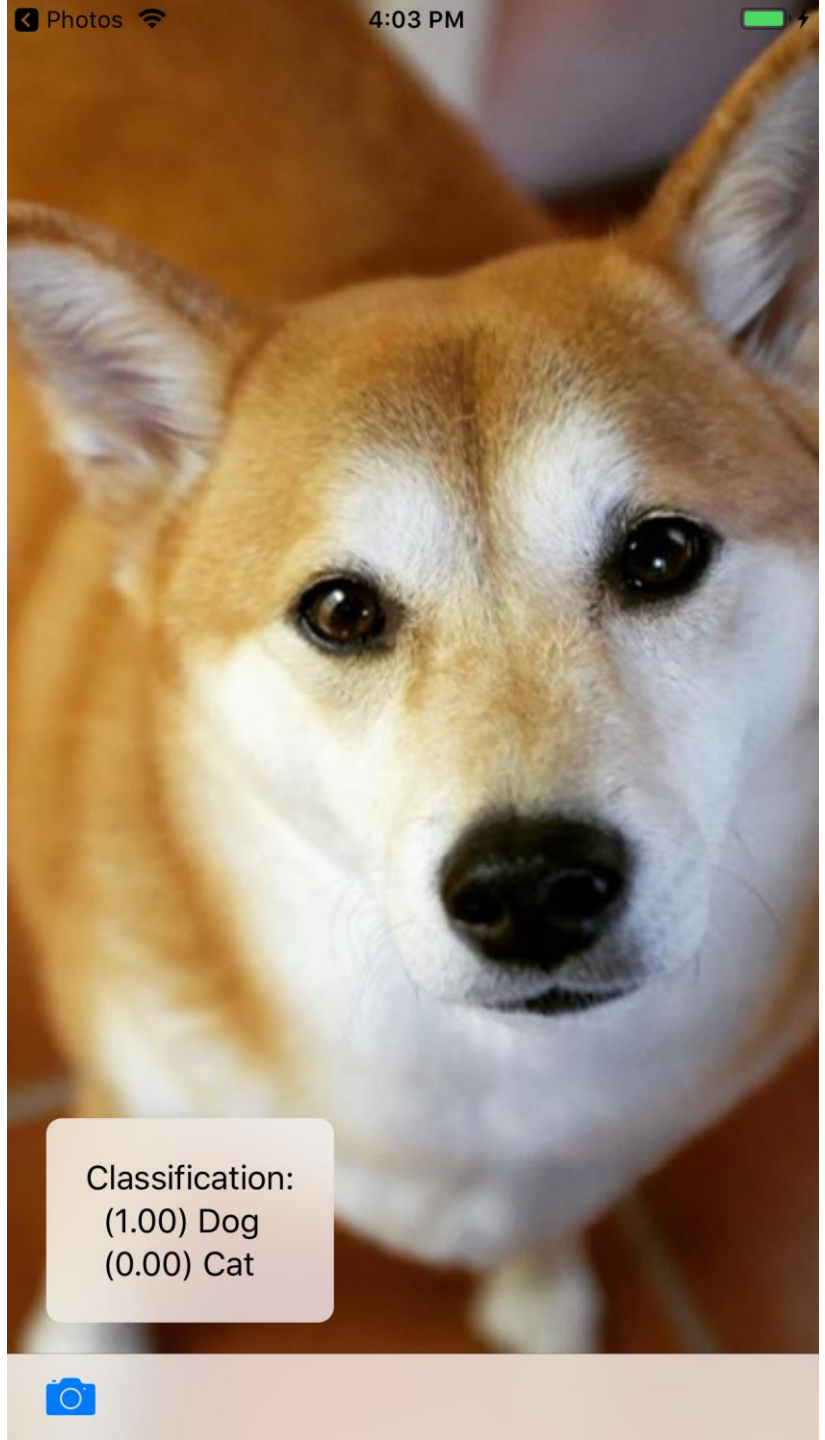
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(
    n_estimators=100, max_depth=20, random_state=0)
clf.fit(X_train, y_train)

y_pre = clf.predict(X_test)
```

Data Split

모의고사 늘 만점 받던 철수는
수능에서 80점 받았다.

왜 그랬을까?

철수는 수능공부안하고
모의고사 공부만함

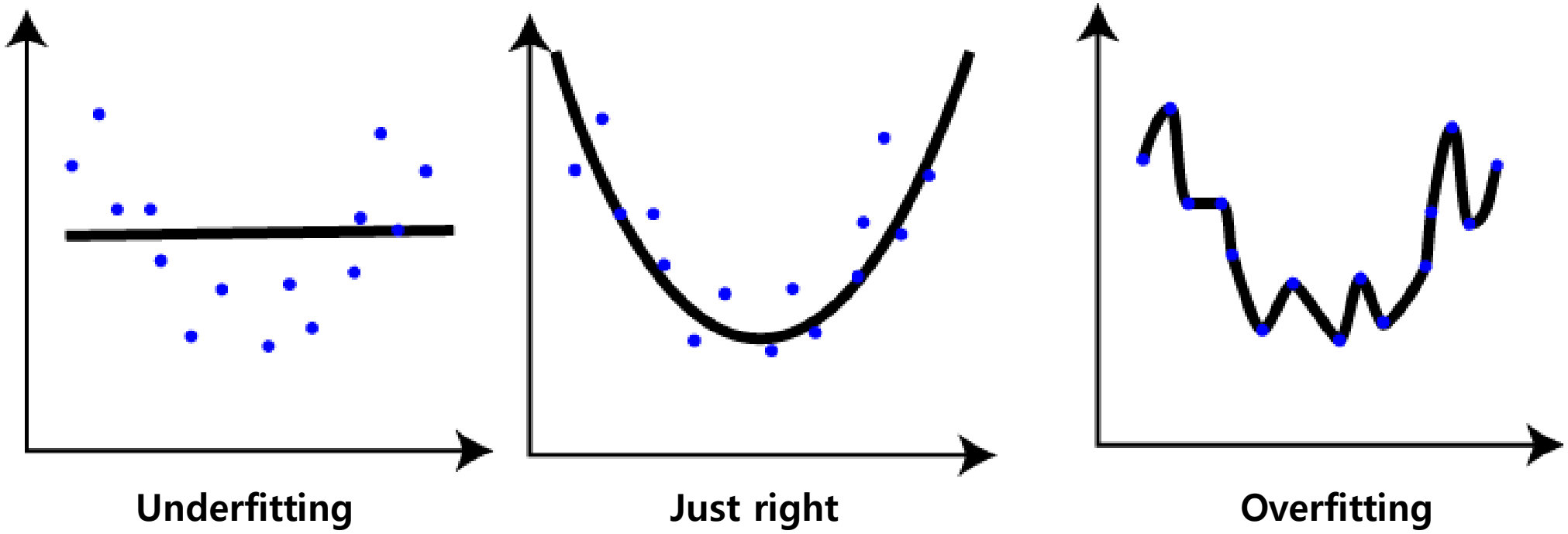

Photos
4:03 PM
Classification:
(1.00) Dog
(0.00) Cat

Overfitting

ML에서 학습 데이터에만 맞춰서 모델을 생성

Overfitting

학습데이터 과다 최적화 → 새로운 데이터의 예측 ↓

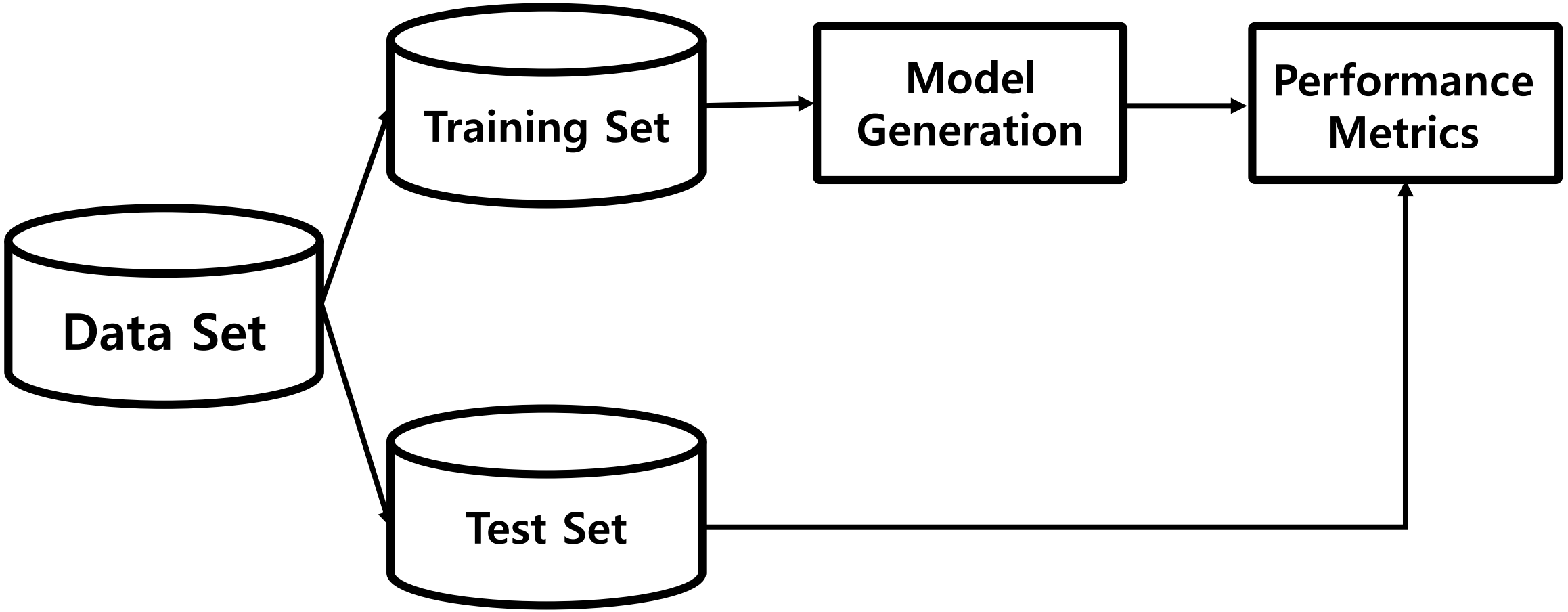


https://goo.gl/aP8iFa

ML 모델은 현실의 데이터를 잘 예측해야 함!

확보된 데이터를 잘 나눠서 평가하자!

그 데이터가 세상을 제대로 반영하도록!


Data Set
Training Set
Test Set
Model Generation
Performance Metrics

Holdout Method (Sampling)

- 데이터를 Training과 Test와 나눠서 모델을 생성하고 테스트하는 기법
- 가장 일반적인 모델 생성을 위한 데이터 램덤 샘플링 기법
- Training과 Test를 나누는 비율은 데이터의 크기에 따라 다름

```
import numpy as np
from sklearn.model_selection import train_test_split

X, y = np.arange(10).reshape((5, 2)), range(5)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

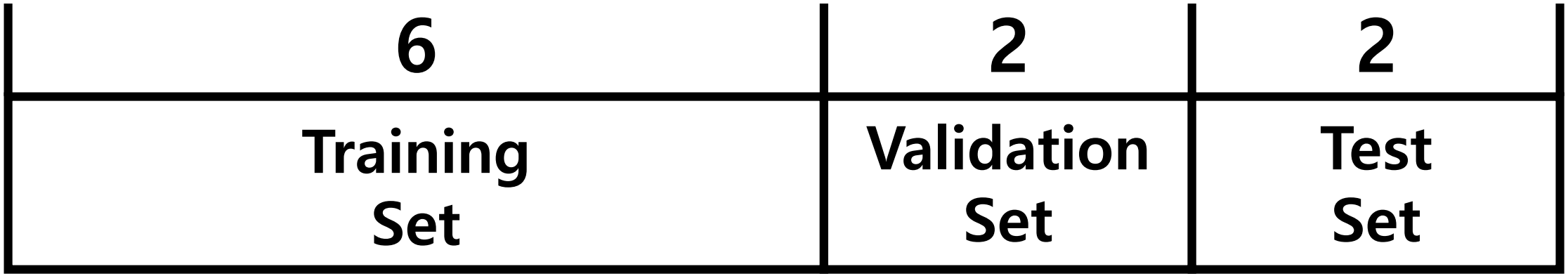
성능 측정을 위해 데이터를 나누는 방법

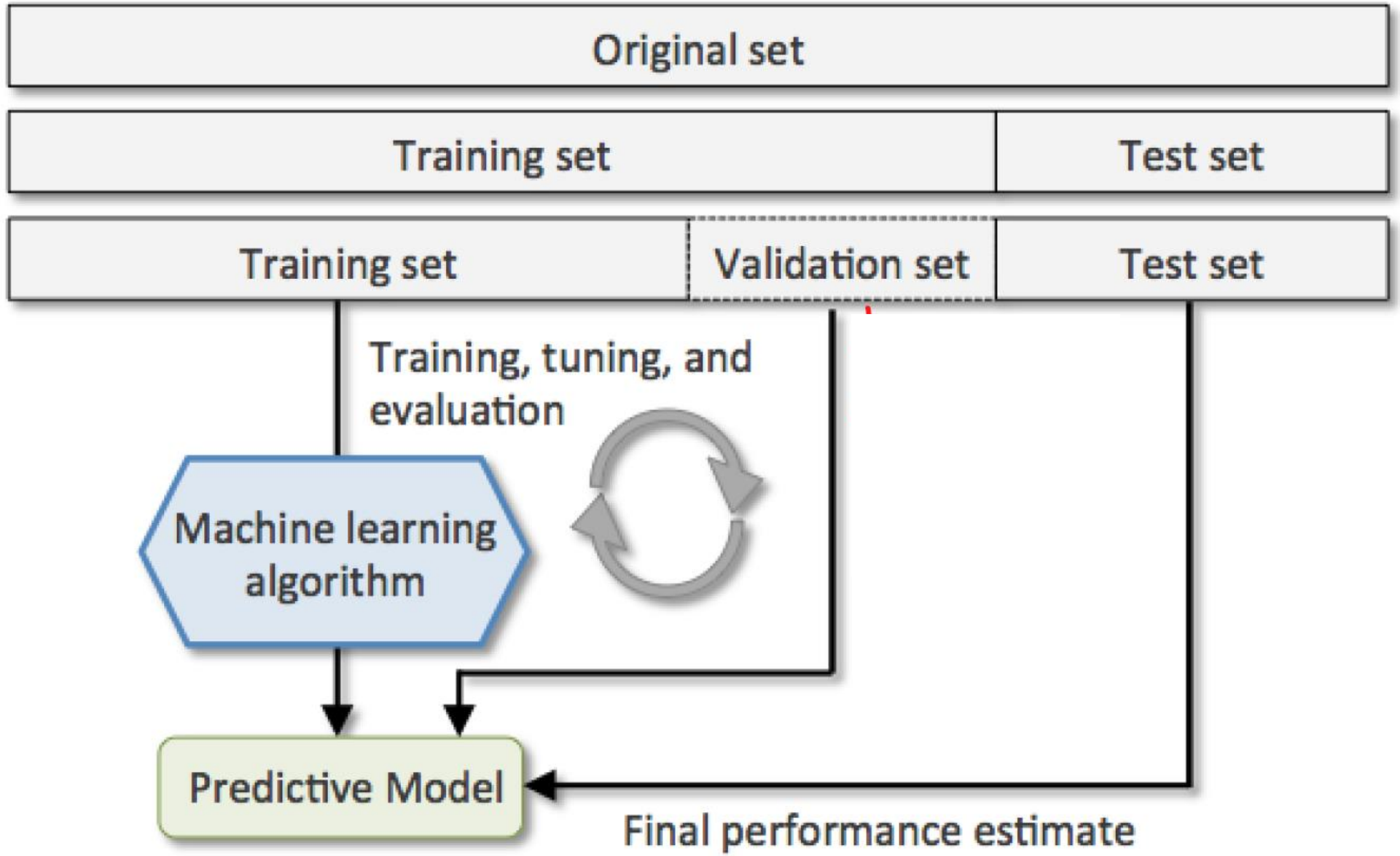
Training - Validation - Test

Training	Validation	Test
Model Building	Model Check	Model Evaluation

Validation Set

- Test Set은 Model이 생성시 절대 Training Set에 포함되지 않아야 함
- Test Set과 달리 Model 생성시 Model에 성능을 평가하기 위해 사용
- Hyper Parameter Turning 시 성능 평가를 통해 Overfitting 방지
- Training 중간에 Model의 성능을 점검

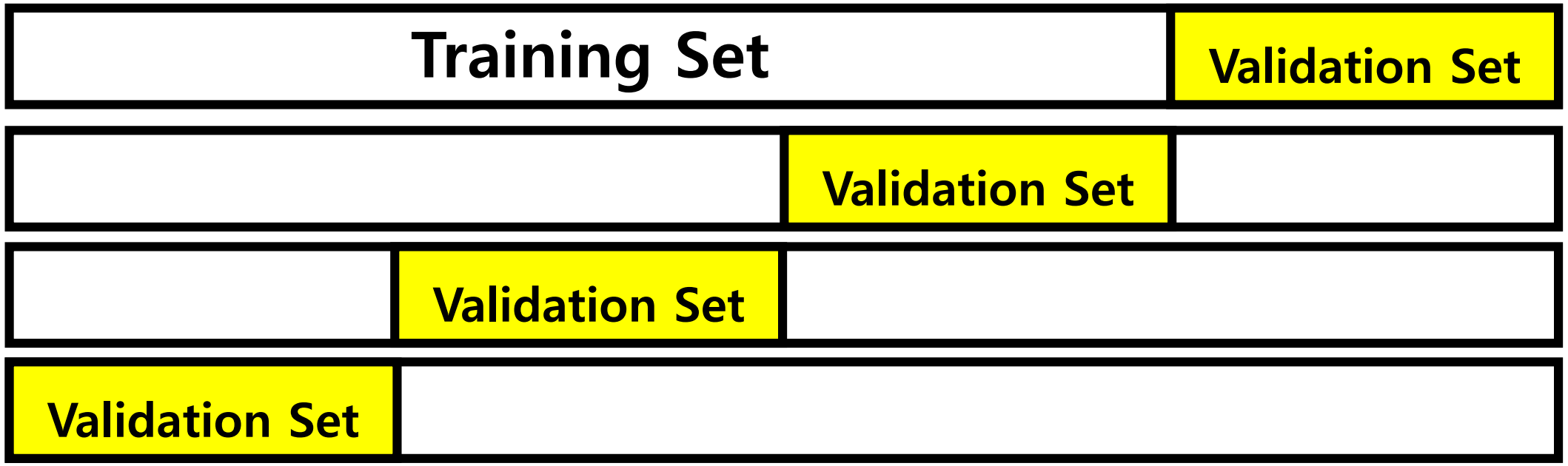

6
2
2
Training Set
Validation Set
Test Set



From: Python Machine Learning, https://goo.gl/JR9vxM

K-fold cross validation

- 학습 데이터를 K번 나눠서 Test와 Train을 실시 → Test의 평균값을 사용
- 모델의 Parameter 튜닝, 간단한 모델의 최종 성능 측정 등 사용



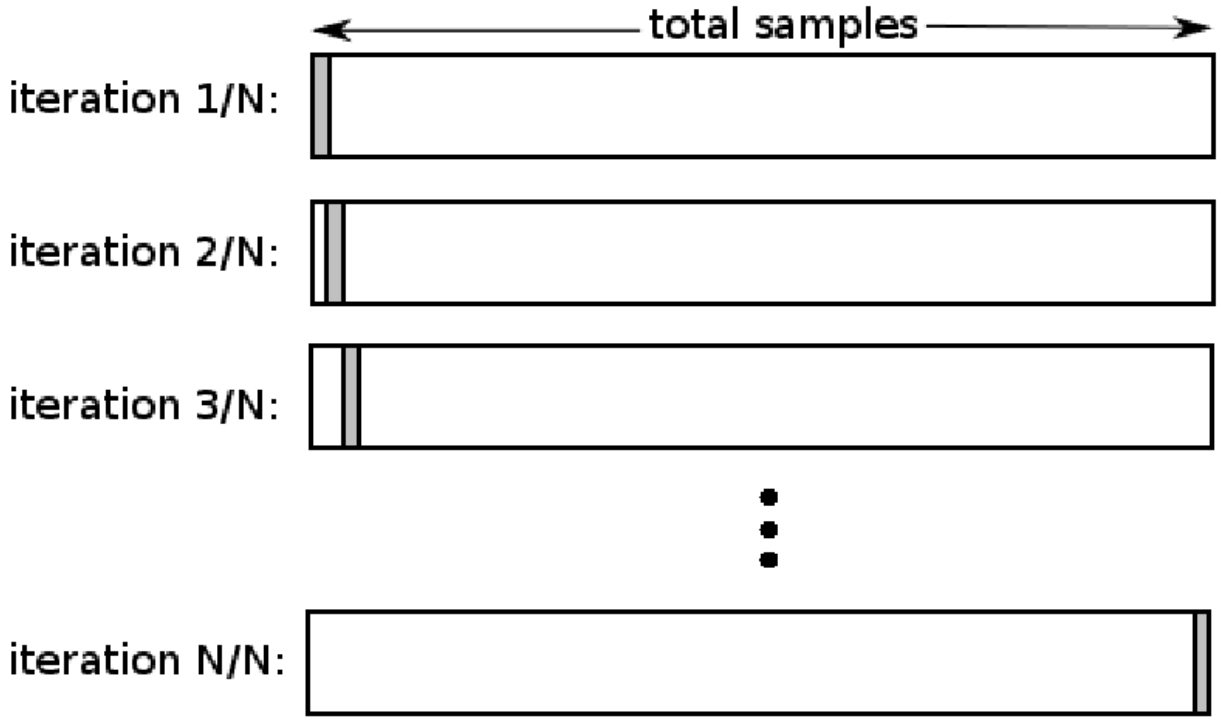
K-fold Cross Validation

```python
from sklearn.model_selection import KFold

kf = KFold(n_splits=10, shuffle=True)

for train_index, test_index in kf.split(X):
    print("TRAIN - ", train_index[:10])
    print("TEST - ", test_index[:10])
```

```
TRAIN -  [0 1 2 3 4 5 6 7 8 9]
TEST -  [ 16  22  24  25  28  58  60  79  92 110]
TRAIN -  [0 1 2 3 4 5 6 7 8 9]
TEST -  [ 23  30  33  56  66  69  72  73  74 107]
TRAIN -  [ 0  1  2  3  4  5  6  7  9 10]
TEST -  [  8  12  39  41  61  78  96  97 100 112]
TRAIN -  [ 0  1  2  3  4  6  7  8  9 10]
TEST -  [  5  15  31  38  46  85  91  95 116 124]
TRAIN -  [0 1 2 3 4 5 6 7 8 9]
TEST -  [ 18  37  40  43  55  57  75  77  90 104]
```

Leave One Out (LOO)

- Simple cross validation → k = data size
- 한번에 한 개의 데이터만 Test set으로 사용함 → 총 k번 iteration



https://www.researchgate.net/profile/Nikolaos_Chlis/publication/266617511/figure/fig11/AS:295705362092036@1447513060277/Leave-One-Out-Cross-Validation.png

Etc…

- RepatedKFold – 중복이 포함된 K-Fold 생성

- LeavePOut – 한번에 P개를 뽑음 (Not LOO for one data)

- ShuffleSplit – 독립적인(중복되는) 데이터 Sampling

- StratifiedKFold – Y 값 비율에 따라 뽑음

- GroupKFold – 그룹별로 데이터를 Sampling

Cross validation
Train-Validation-Test

Imbalanced dataset

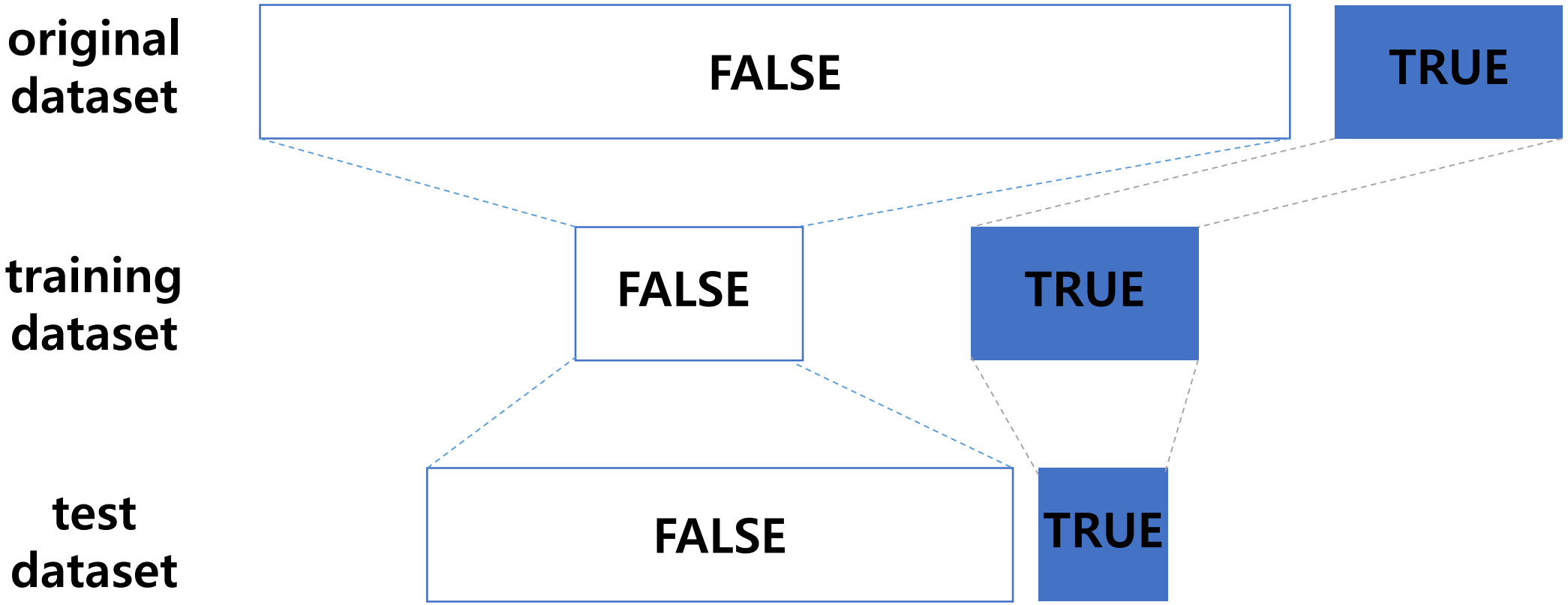
- 유방암 사진 dataset
- 학사 경고자 예측 dataset
- 물건을 구매한 유저의 dataset
- 카드 사기에 관련된 dataset

대부분의 dataset은 imbalanced dataset

How to handle imbalanced dataset

- 적절한 performance metric을 선정 (accuracy X)
  - precision, recall, AUC이 적절
- 적절한 training dataset의 resampling
  - oversampling, under sampling, data augmentation
- Ensemble

Dataset resampling



Dataset resampling

- Imbalanced class가 충분히 많다면

  under sampling → FALSE 데이터를 줄임

- Imbalanced class가 부족하다면

  over sampling → TRUE 데이터를 늘림

imbalanced-learn

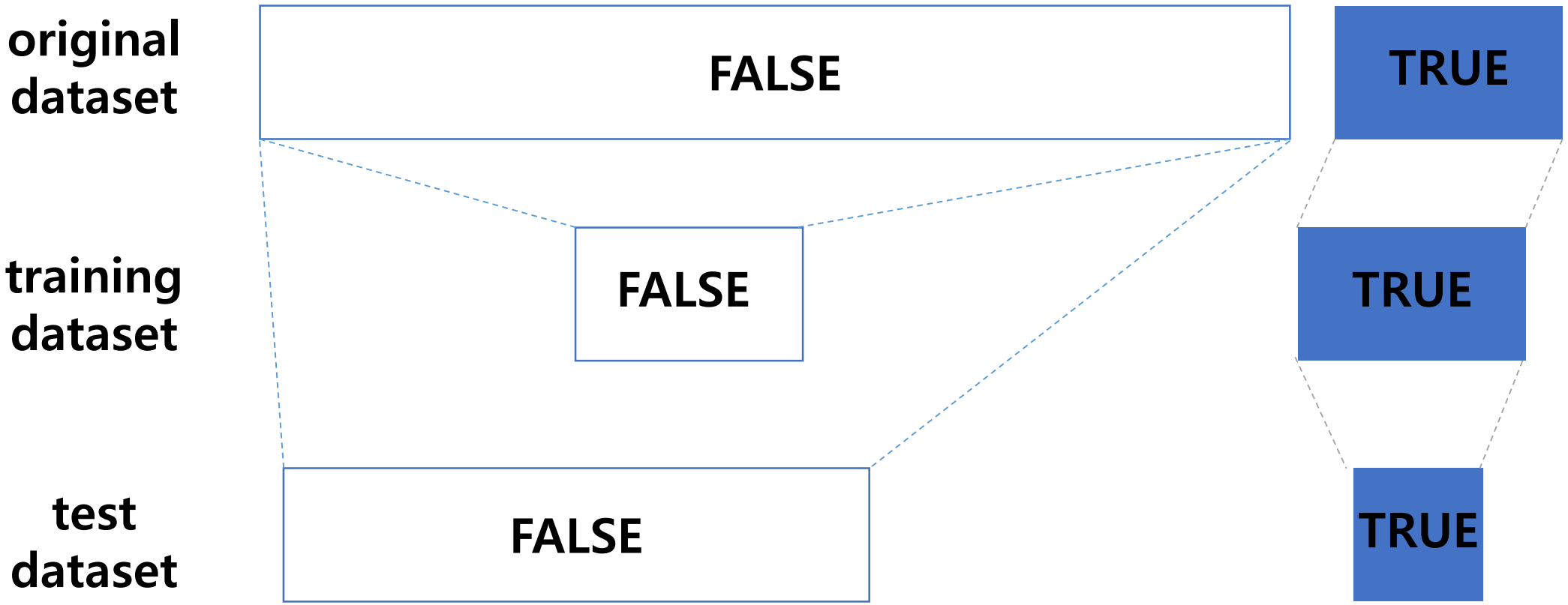
- scikit-learn의 imbalanced dataset 확장 모듈
- under sampling, over sampling, SMOTE 등 제공

https://github.com/scikit-learn-contrib/imbalanced-learn

```
pip install -U imbalanced-learn
```

```
conda install -c conda-forge imbalanced-learn
```

Stratified sampling



Imbalanced dataset handling process

- 전체 dataset에서 test와 dev set을 나눔 (stratified)
- dev set으로 under sampling 또는 oversampling
- 모델의 생성
- Test set으로 모델의 검증

Performance Metrics

만들어진 모델의 성능은 어떻게 평가할 것인가?

평가할 수 있는 Measure가 필요

Regression metrics

- Mean Absolute Error

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n} |e_i| \,.$$

잔차의 절대값의 Sum

```
from sklearn.metrics import median_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
median_absolute_error(y_true, y_pred)
```

Regression metrics

- Root Mean Squared Error (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

잔차 제곱의 sum의 루트

```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
```

Regression metrics

- R squared

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \mu)^2}.$$

0과 1사이 숫자로 크면 클 수록 높은 적합도를 지님

```
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)
```

분류 문제의 정확도 성능

실제 Class 대비
얼마나 잘 맞혔는가?

Confusion Matrix (혼합 행렬)

- 실제 라벨과 예측 라벨의 일치 개수를 Matrix 형태로 표현하는 기법

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

Confusion Matrix (혼합 행렬)

True Positive (TP)

- 실제 결과 참(1)에 대한 예측이 맞음

True – 예측이 맞음

Positive – 참(1) 인 경우

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

Confusion Matrix (혼합 행렬)

True Negative (TN)

- 실제 결과 거짓(0)에 대한 예측이 맞음

True – 예측이 맞음

Negative – 거짓(0) 인 경우

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

Confusion Matrix (혼합 행렬)

False Positive (FP)

- 실제 결과 참(1)에 대한 예측이 틀림

False – 예측이 틀림

Positive – 참(1) 인 경우

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

Confusion Matrix (혼합 행렬)

False Negative (FN)

- 실제 결과 거짓(0)에 대한 예측이 틀림

False – 예측이 틀림

Negative – 거짓(0) 인 경우

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

Confusion Matrix (혼합 행렬)

True Positive (TP)

True Negative (TN)

False Positive (FP)

False Negative (FN)

sklearn.metrics.confusion_matrix

sklearn.metrics. **confusion_matrix** (*y_true, y_pred, labels=None, sample_weight=None*) ¶ [source]

```
from sklearn.metrics import confusion_matrix
y_true = [1, 0, 1, 1, 0, 1]
y_pred = [0, 0, 1, 1, 0, 1]
confusion_matrix(y_true, y_pred)
```

```
array([[2, 0],
       [1, 3]])
```

True Class \ Prediction	0	1
0		
1		

```
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
(tn, fp, fn, tp)
```

```
(2, 0, 1, 3)
```

Metrics for classification performance

- **Accuracy (정확도)** $$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Error Rate (오차율)** $$Errorrate = \frac{FP+FN}{TP+TN+FP+FN} = (1-Accuracy)$$

- **Precision (정밀도)** $$Precision = \frac{TP}{TP+FP} \quad (PPV: \ Positive \ \ Predict \ \ Value)$$

- **Specificity (특이도)** $$Specificity = \frac{TN}{TN+FP} \quad (TNR: \ True \ \ Negative \ \ Rate)$$

- **Sensitivity (민감도)** $$Sensitivity = \frac{TP}{TP+FP} \quad (TPR: \ True \ \ Positive \ \ Rate)$$

실제 Class 대비
얼마나 잘 맞혔는가?

정확도 (Accuracy, ACC)

- 전체 데이터 대비 정확하게 예측한 개수의 비율

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

$$ACC = 1 - ERR$$

		Prediction: 1	Prediction: 0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

오차율 (Error Rate, ERR)

- 전체 데이터 대비 부정확하게 예측한 개수의 비율

$$ERR = \frac{FP + FN}{TP + TN + FP + FN}$$

$$ERR = 1 - ACC$$

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

```
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = np.array([0, 1, 1, 0])
y_true = np.array([0, 1, 0, 0])
```

```
sum(y_true == y_pred) / len(y_true)
```

```
0.75
```

```
accuracy_score(y_true, y_pred)
```

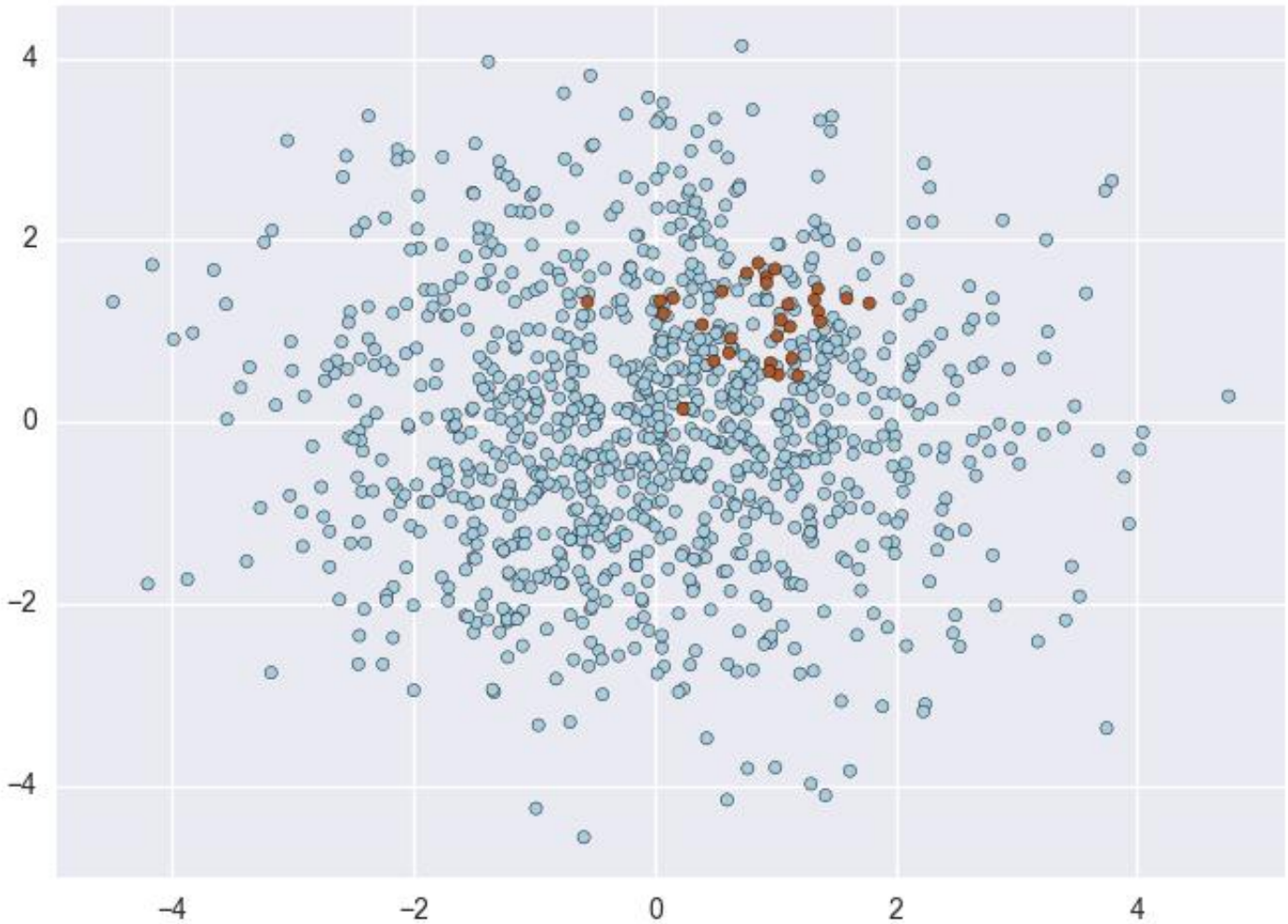
```
0.75
```

불균일한
Dataset의 처리

불균일한 Dataset의 종류

- 14세 이하의 10만명당 암 발병 인원은 14.8, 약 0.015%
- 대학의 학사경고자 평균 비율 3%
- 하버드 입학 지원자의 합격률은 2%
- 이메일 수신자 중 2% 만이 물건을 구매

만약 Accuracy로 구한다면?



https://svds.com/learning-imbalanced-classes/

Metrics for Imbalanced Dataset

정밀도 (Precision, Positive Predictive Value)

- 긍정이라고 예측한 비율 중 진짜 긍정인 비율
- 긍정이라고 얼마나 잘 예측했는가? 긍정 예측 정밀도?

$$PRECISON(PPV) = \frac{TP}{TP + FP}$$

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

```
from sklearn.metrics import precision_score
```

```
y_pred = np.array([0, 1, 1, 0])
y_true = np.array([0, 1, 0, 0])
```

```
sum((y_pred == 1) & (y_pred == y_true)) / sum(y_pred)
```

```
0.5
```

```
precision_score(y_true, y_pred)
```

```
0.5
```

sklearn.metrics.precision_score

```
sklearn.metrics. precision_score (y_true, y_pred, labels=None, pos_label=1, average='binary',
sample_weight=None)                                                                    [source]
```

labels : list, optional
pos_label : str or int, 1 by default
average : string, [None, 'binary' (default), 'micro', 'macro', 'samples', 'weighted']
sample_weight : array-like of shape = [n_samples], optional

average : string, [None, 'binary' (default), 'micro', 'macro', 'samples', 'weighted']

This parameter is required for multiclass/multilabel targets. If `None`, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

`'binary'`:

Only report results for the class specified by `pos_label`. This is applicable only if targets (`y_{true,pred}`) are binary.

`'micro'`:

Calculate metrics globally by counting the total true positives, false negatives and false positives.

`'macro'`:

Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

`'weighted'`:

Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

`'samples'`:

Calculate metrics for each instance, and find their average (only meaningful for multilabel classification where this differs from `accuracy_score`).

```
from sklearn.metrics import precision_score
```

```
y_pred = np.array([0, 1, 1, 0])
y_true = np.array([0, 1, 0, 0])
```

```
sum((y_pred == 1) & (y_pred == y_true)) / sum(y_pred)
```

```
0.5
```

```
precision_score(y_true, y_pred)
```

```
0.5
```

```
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
confusion_matrix(y_true, y_pred)
```

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

전체 평균

'micro':

Calculate metrics globally by counting the total true positives, false negatives and false positives.

'macro': (Label별 값 합)의 평균

Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

```
precision_score(y_true, y_pred, average='macro')
```

```
0.2222222222222221
```

```
precision_score(y_true, y_pred, average=None)
```

```
array([ 0.66666667,  0.        ,  0.        ])
```

```
precision_score(y_true, y_pred, average='micro')
```

```
0.3333333333333331
```

민감도 (Sensitivity, Recall, True Positive Rate)

- 실제 긍정 데이터중 긍정이라고 예측한 비율, 반환율, 재현율
- 얼마나 잘 긍정(예 - 암)이라고 예측하였는가?

$$RECALL(TPR) = \frac{TP}{TP + FN} = \frac{TP}{P}$$

		Prediction: 1	Prediction: 0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

```
from sklearn.metrics import recall_score
```

```
y_pred = np.array([0, 1, 1, 0])
y_true = np.array([0, 1, 0, 0])
```

```
sum((y_true == 1) & (y_pred == y_true)) / sum(y_true)
```

```
1.0
```

$$RECALL(TPR) = \frac{TP}{TP + FN} = \frac{TP}{P}$$

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

```
recall_score(y_true, y_pred)
```

```
1.0
```

```
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
recall_score(y_true, y_pred, average='macro')
```

```
0.3333333333333331
```

```
recall_score(y_true, y_pred, average='micro')
```

```
0.3333333333333331
```

```
recall_score(y_true, y_pred, average=None)
```

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

특이성 (Specificity, True Negative Rate)

- 부정을 얼마나 잘 부정이라고 인식해는가?
- 전제 부정중 부정을 정확히 찾아낸 비율

$$SPC = \frac{TN}{TN + FP} = \frac{TN}{N}$$

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

F1 Score (F-measure, F-score)

- Precision과 Recall의 통합한 측정지표
- Precision과 Recall의 조화평균

$$F_1 = 2\frac{precision * recall}{precision + recall}$$

		Prediction	
		1	0
Actual Class	1	True Positive	False Negative
	0	False Positive	True Negative

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

```
from sklearn.metrics import f1_score
y_pred = np.array([0, 1, 1, 0])
y_true = np.array([0, 1, 0, 0])
```

```
pre = precision_score(y_true, y_pred)
rec = recall_score(y_true, y_pred)
```

```
2 * (pre * rec) / (pre + rec)
```

```
0.6666666666666663
```

$$F_1 = 2\frac{precision * recall}{precision + recall}$$

```
f1_score(y_true, y_pred)
```

```
0.6666666666666663
```

```
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
f1_score(y_true, y_pred, average='macro')
```

```
0.26666666666666666
```

```
f1_score(y_true, y_pred, average='micro')
```

```
0.3333333333333331
```

```
f1_score(y_true, y_pred, average=None)
```

```
array([ 0.8,  0. ,  0. ])
```

Example

		Prediction		
		1	0	
Actual Class	1	90	210	300
	0	140	9560	9700
		230	9770	10000

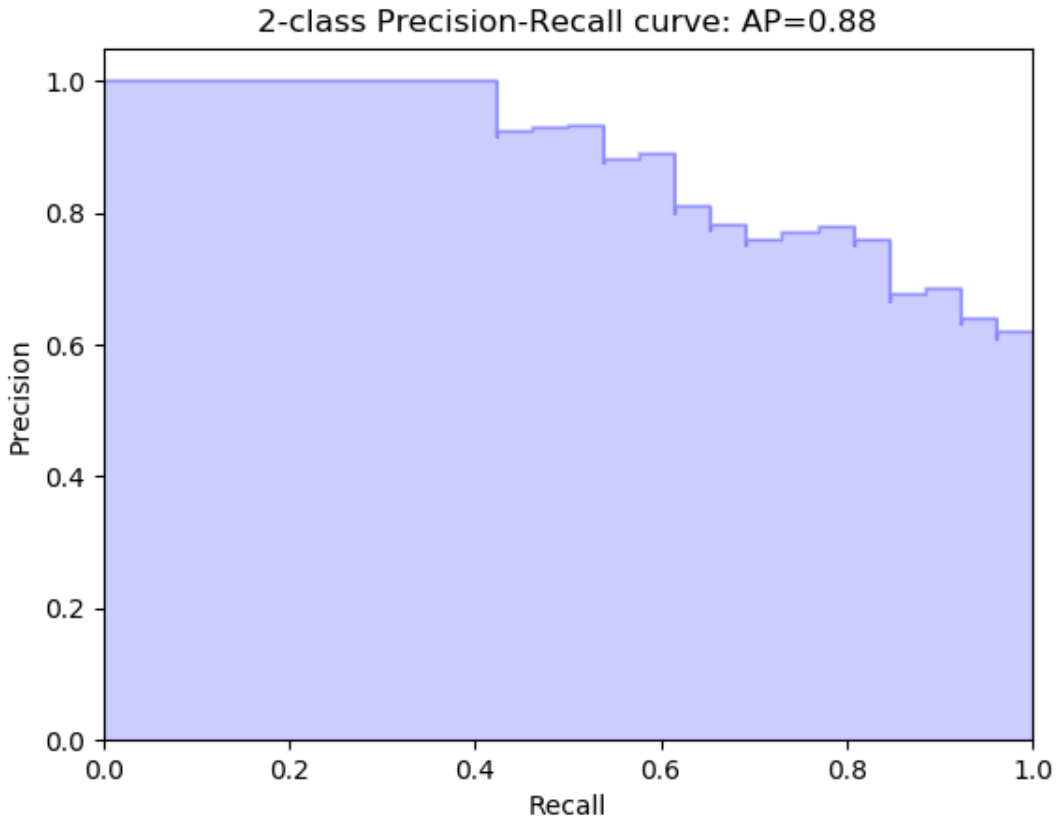
$$PRECISON(PPV) = \frac{TP}{TP + FP}$$

$$RECALL(TPR) = \frac{TP}{TP + FN} = \frac{TP}{P}$$

$$SPC = \frac{TN}{TN + FP} = \frac{TN}{N}$$

Precision - Recall Curve

- 예측 확률 Threshold를 변화시켜 Precision/Recall 측정
- 시각화 할 때 유용하게 사용 가능



```
import numpy as np
from sklearn.metrics import precision_recall_curve
y_true = np.array([0, 0, 1, 1])
y_scores = np.array([0.1, 0.4, 0.35, 0.8])
precision, recall, thresholds = precision_recall_curve(
    y_true, y_scores)
```

```
precision
```

```
array([ 0.66666667,  0.5       ,  1.        ,  1.        ])
```

```
recall
```

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

```
thresholds
```

```
array([ 0.35,  0.4 ,  0.8 ])
```

Precision - Classification Report

- Classification 문제에서 한번에 Precision, Recall, F1 결과 출력

	precision	recall	f1-score	support
class 0	0.50	1.00	0.67	1
class 1	0.00	0.00	0.00	1
class 2	1.00	0.67	0.80	3
avg / total	0.70	0.60	0.61	5

```
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
```

```
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
             precision    recall  f1-score   support

    class 0       0.50      1.00      0.67         1
    class 1       0.00      0.00      0.00         1
    class 2       1.00      0.67      0.80         3

avg / total       0.70      0.60      0.61         5
```

Confusion matrix:

```
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0
 [ 0  0  0  1  0  1  0  0
```

	precision	recall	f1-score	suppo
0	1.00	0.99	0.99	
1	0.99	0.97	0.98	
2	0.99	0.99	0.99	
3	0.98	0.87	0.92	
4	0.99	0.96	0.97	
5	0.95	0.97	0.96	
6	0.99	0.99	0.99	
7	0.96	0.99	0.97	
8	0.94	1.00	0.97	
9	0.93	0.98	0.95	
g / total	0.97	0.97	0.97	8

Human knowledge belongs to the world.