Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

Python for Data Analytics

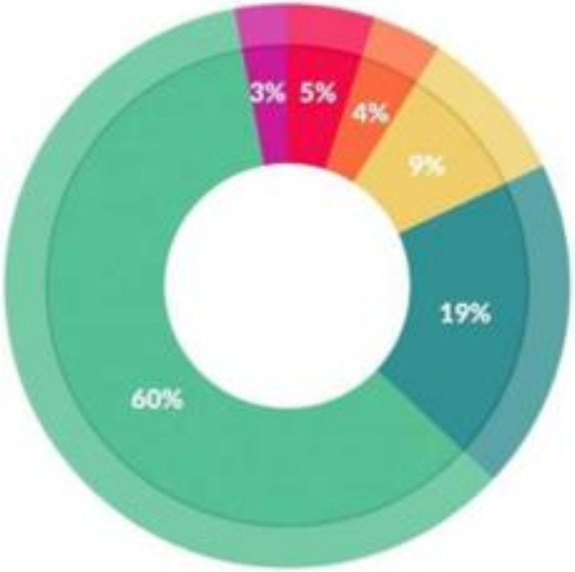
Data Preprocessing

Outline

- Preprocessing with Pandas
- Preprocessing with SK-Learn
- Sampling with Imbalanced Learn (Imblearn)

Data Preprocessing

- The process of cleaning up the messy raw data for analysis
- Repetitive and tedious work
- Direct impact on analysis result and model performance
- Data collection and preprocessing occupy up to 80% of the analysis time

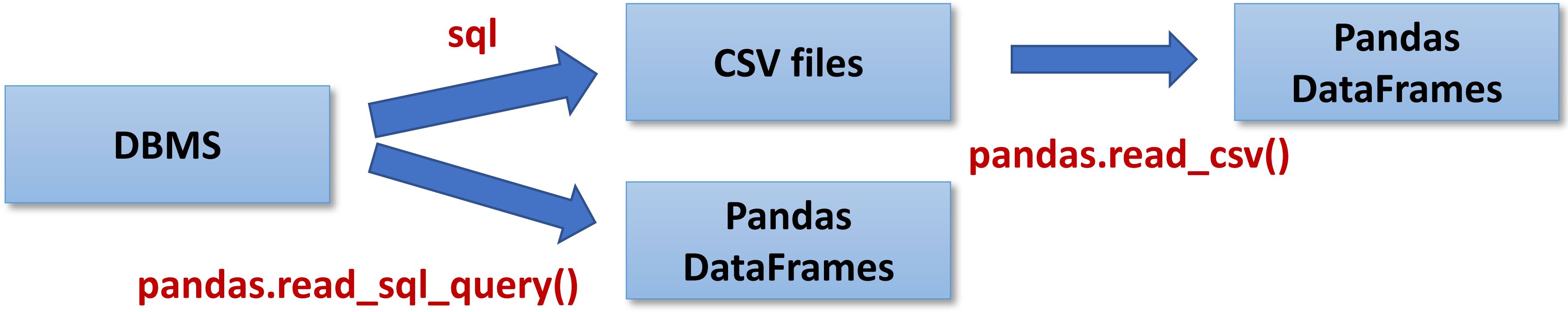


What data scientists spend the most time doing

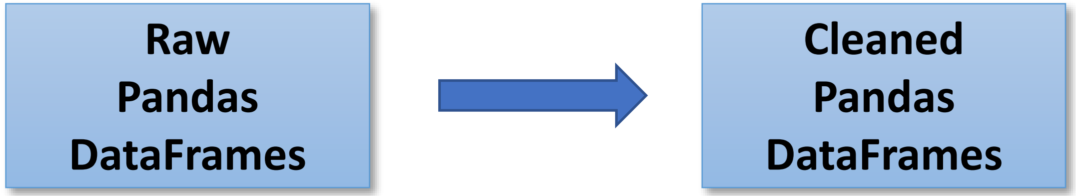
- Building training set: 3%
- Cleaning and organizing data: 60%
- Collecting data set: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

Data Preprocessing Scenarios

- From Database tables to Pandas DataFrame



- From raw DataFrames to cleaned DataFrames
  - Preprocessing with Pandas
  - Preprocessing with SK-Learn



Preprocessing with Pandas

- Handling Missing Values
- Handling Outliers

From MySQL to CSV Files

Database
Table A

id	score
1	90
2	60
3	70
4	50
5	80

```
SELECT *
FROM A
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/A.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ','
LINES TERMINATED BY '\n';
```

CSV file

A.csv

```
1  "1","90"
2  "2","60"
3  "3","70"
4  "4","50"
5  "5","80"
```

Text Format

	A	B
1	1	90
2	2	60
3	3	70
4	4	50
5	5	80

Excel Format

- `INTO OUTFILE`
  - Specify the pathname for the result of the `SELECT` query
  - `FIELDS ENCLOSED BY '"'`
    - Enclose each column with "
  - `FIELDS TERMINATED BY ','`
    - Separate columns using the , character
  - `LINES TERMINATED BY '\n'`
    - Separate rows by adding \n at the end of each row

From MySQL to CSV Files (2)

- testset.csv
  - "test" DB table with 5 columns: idx, iduser, mdutype, group, viewCount
  - MySQL's NULL value is converted to '\N' in the CSV file
  - Pandas recognizes '\N' as a normal string
  - Convert numerical data 'viewcount' into strings

- Preprocessing needed!

Database Table "**test**"

idx	iduser	mdutype	grou	viewcount
0	10100018739106		sdu	12
1	10100037810674		sdu	23
2	10100036273719		sdu	4
3	10100027752244		sdu	6
4	10100000624840		sdu	

column header added

```
(SELECT '', 'iduser', 'mdutype', 'group', 'viewcount')
UNION
SELECT idx, iduser, mdutype, grou, viewcount
FROM test
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/testset.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ','
LINES TERMINATED BY '\n';
```

CSV file

```
"","iduser","mdutype","group","viewcount"
"0","10100018739106","","sdu","12"
"1","10100037810674","","sdu","23"
"2","10100036273719","","sdu","4"
"3","10100027752244","","sdu","6"
"4","10100000624840","","sdu",\N
```

From MySQL to CSV Files (3)

- Using SQL utility function

- IFNULL(*column_name*, *value*)
  - Replace the NULL value into "*value*"

Database Table "**test**"

idx	iduser	mdutype	grou	viewcount
0	10100018739106		sdu	12
1	10100037810674		sdu	23
2	10100036273719		sdu	4
3	10100027752244		sdu	6
4	10100000624840		sdu	

```sql
(SELECT '', 'iduser', 'mdutype', 'group', 'viewcount')
UNION
SELECT idx, iduser, mdutype, grou, IFNULL(viewcount, '')
FROM test
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/testset.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ','
LINES TERMINATED BY '\n';
```

CSV file

```
"","iduser","mdutype","group","viewcount"
"0","10100018739106","","sdu","12"
"1","10100037810674","","sdu","23"
"2","10100036273719","","sdu","4"
"3","10100027752244","","sdu","6"
"4","10100000624840","","sdu",""
```

From MySQL to Pandas (1)

- *pandas*.read_sql_query(*sql*, *con*)
  - Read SQL query into a Pandas dataframe
  - *sql*: SQL query to be executed
  - *con*: DBMS connector

- MySQL connector

```
import MySQLdb
conn = MySQLdb.connect(host=[host], user=[user name], passwd=[password], db=[db name])

#import pymysql
#conn = pymysql.connect(host=[host], user=[user name], password=[password], db=[db name])
```

- Oracle connector

```
import cx_Oracle
conn = cx_Oracle.connect('[user]/[password]@[host]:[port]/[sid]')
```

From MySQL to Pandas (2)

```
data = pd.read_sql("select idx, iduser, mdutype, grou, IFNULL(viewcount, '') from test", conn, index_col='idx')
data.columns = ['iduser', 'mdutype', 'group', 'viewCount']
data.head()
```

Database "**test**" Table

idx	iduser	mdutype	grou	viewcount	editcount	sharecount
0	10100018739106		sdu	12	0	0
1	10100037810674		sdu	23	0	0
2	10100036273719		sdu	4	0	0
3	10100027752244		sdu	6	0	1
4	10100000624840		sdu	NULL	NULL	NULL

…

opencount	savecount	exportcount	viewtraffic	edittraffic	exporttraffic	traffic
12	0	0	3504812	0	0	3504812
23	0	0	17123098	0	0	17123098
4	0	0	2234363	0	0	2234363
6	2	0	602361	210114	0	812475
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Pandas "**data**" Data Frame

idx	iduser	mdutype	group	viewCount
0	10100018739106		sdu	12
1	10100037810674		sdu	23
2	10100036273719		sdu	4
3	10100027752244		sdu	6
4	10100000624840		sdu	

Dataset for Handling Missing Values

- User behavior and payment data in a shared file system
- 200,000 entries with 22 columns
- Column information
  - rowid
  - iduser: User ID
  - mdutype: payment type
  - group: payment info (mdu: paid user, sdu: free user)
  - view/edit/share/search/cowork counts
  - add/del/move/rename counts
  - other user behaviors

```
df = pd.read_csv('testset.csv', index_col=0)
```

https://github.com/songhunhwa/songhunhwa.github.com/tree/master/tutorial/tutorial_03/testset.csv

Dataset Head & Tail

```
df.head()
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
0	10100018739106	NaN	sdu	12.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
1	10100037810674	NaN	sdu	23.0	0.0	0.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
2	10100036273719	NaN	sdu	4.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
3	10100027752244	NaN	sdu	6.0	0.0	1.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
4	10100000624840	NaN	sdu	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 22 columns

```
df.tail()
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
199995	10100014533282	NaN	sdu	37.0	0.0	2.0	...	7.0	0.0	13064406.0	1922364.0	0.0	14986770.0
199996	10100037382422	a2p	mdu	6.0	0.0	0.0	...	0.0	0.0	15936676.0	0.0	0.0	15936676.0
199997	10100024157271	NaN	sdu	32.0	0.0	0.0	...	0.0	0.0	7305871.0	0.0	0.0	7305871.0
199998	10100022150627	NaN	sdu	18.0	0.0	0.0	...	0.0	0.0	53352144.0	0.0	0.0	53352144.0
199999	10100021804275	NaN	sdu	3.0	0.0	0.0	...	0.0	0.0	95232.0	0.0	0.0	95232.0

5 rows × 22 columns

Set Index

```python
df.set_index('iduser', inplace=True)
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
0	10100018739106	NaN	sdu	12.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
1	10100037810674	NaN	sdu	23.0	0.0	0.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
2	10100036273719	NaN	sdu	4.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
3	10100027752244	NaN	sdu	6.0	0.0	1.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
4	10100000624840	NaN	sdu	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 22 columns

	mdutype	group	viewCount	editCount	shareCount	searchCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	tra
iduser													
10100018739106	NaN	sdu	12.0	0.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	35048
10100037810674	NaN	sdu	23.0	0.0	0.0	1.0	...	0.0	0.0	17123098.0	0.0	0.0	1712309
10100036273719	NaN	sdu	4.0	0.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	223436
10100027752244	NaN	sdu	6.0	0.0	1.0	0.0	...	2.0	0.0	602361.0	210114.0	0.0	81247
10100000624840	NaN	sdu	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N

5 rows × 21 columns

Handling Missing Values

- Missing value 처리 방법
  - Missing value 포함 데이터 제거 (가장 쉽지만, 데이터가 충분히 많아야 가능)
  - 수치형인 경우 mean이나 median, 범주형인 경우 mode(최빈값)로 대체
  - 간단한 예측 모델로 예측 값을 추정하여 대체
- 고려 사항
  - 도메인 지식 활용 필요
    - 인적, 기계적 원인인 경우 missing value 발생을 사전에 방지 가능
    - 수치형인 경우, 0, mean, median 중 어떤 값으로 대체하는 것이 맞는지 판단
  - Missing value의 다양한 의미
    - NA: Not Available (missing), Null: empty object, NaN: Not a Number (Pandas)
    - (숫자 0과 missing value는 완전히 다른 개념)
  - Label 데이터에 missing value가 있다면 대체하지 말고 제거

Handling Missing Values in Pandas

- Column별 정보 확인
  - 데이터 타입이 맞지 않는 경우 전처리 필요

예를 들어 ',' 같은 특수문자가 포함되면, 수치형 데이터를 string으로 인식함

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
mdutype          9328 non-null object
group            200000 non-null object
viewCount        165369 non-null float64
editCount        165369 non-null float64
shareCount       165369 non-null float64
searchCount      165369 non-null float64
coworkCount      165369 non-null float64
add              63166 non-null float64
del              63166 non-null float64
move             63166 non-null float64
rename           63166 non-null float64
adddir           63166 non-null float64
movedir          63166 non-null float64
visdays          184306 non-null float64
openCount        149090 non-null float64
saveCount        149090 non-null float64
exportCount      149090 non-null float64
viewTraffic      149090 non-null float64
editTraffic      149090 non-null float64
exportTraffic    149090 non-null float64
traffic          149090 non-null float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

- Column별 missing value 수 확인

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

```
mdutype          190672
group                 0
viewCount         34631
editCount         34631
shareCount        34631
searchCount       34631
coworkCount       34631
add              136834
del              136834
move             136834
rename           136834
adddir           136834
movedir          136834
visdays           15694
openCount         50910
saveCount         50910
exportCount       50910
viewTraffic       50910
editTraffic       50910
exportTraffic     50910
traffic           50910
dtype: int64
```

Replacing Missing Values in Pandas

- Missing value를 평균값으로 대체

```
mean = df.viewCount.mean()
print(mean)
df.viewCount =
df.viewCount.fillna(mean)
```

```
24.576208358277547
```

```
df.viewCount[:20]
```

```
iduser
10100018739106    12.0
10100037810674    23.0
10100036273719     4.0
10100027752244     6.0
10100000624840     NaN
10100006151000    33.0
10100036301327    25.0
10100038731798     NaN
10100039037854     4.0
10100038701419    27.0
10100034746743     5.0
10100016781863    18.0
10100023986518     3.0
10100006498305     3.0
10100038316936    13.0
10100017870216    11.0
10100019387233     8.0
10100038615816     5.0
10100037701731    70.0
10100026531335    18.0
Name: viewCount, dtype: float64
```

```
df.viewCount[:20]
```

```
iduser
10100018739106    12.000000
10100037810674    23.000000
10100036273719     4.000000
10100027752244     6.000000
10100000624840    24.576208
10100006151000    33.000000
10100036301327    25.000000
10100038731798    24.576208
10100039037854     4.000000
10100038701419    27.000000
10100034746743     5.000000
10100016781863    18.000000
10100023986518     3.000000
10100006498305     3.000000
10100038316936    13.000000
10100017870216    11.000000
10100019387233     8.000000
10100038615816     5.000000
10100037701731    70.000000
10100026531335    18.000000
Name: viewCount, dtype: float64
```

Removing Rows with Missing Values

- Missing value가 있는 row를 제거
- *pandas.DataFrame.*dropna([*axis*], [*how*], ...):
  - *how*: either 'any' or 'all'.
  - 'any': if any NA values are present, drop that row or column
  - 'all': if all values are NA, drop that row or column

```
df.head()
```

	mdutype	group	viewCount
iduser			
10100018739106	NaN	sdu	12.0
10100037810674	NaN	sdu	23.0
10100036273719	NaN	sdu	4.0
10100027752244	NaN	sdu	6.0
10100000624840	NaN	sdu	NaN

```
df = df.dropna(how='any')
df.head()
```

	mdutype	group	viewCount
iduser			
10100022538111	a2p	mdu	35.0
10100039309679	a2p	mdu	40.0
10100037687198	a2p	mdu	44.0
10100017371337	a2p	mdu	95.0
10100013627062	a2p	mdu	75.0

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

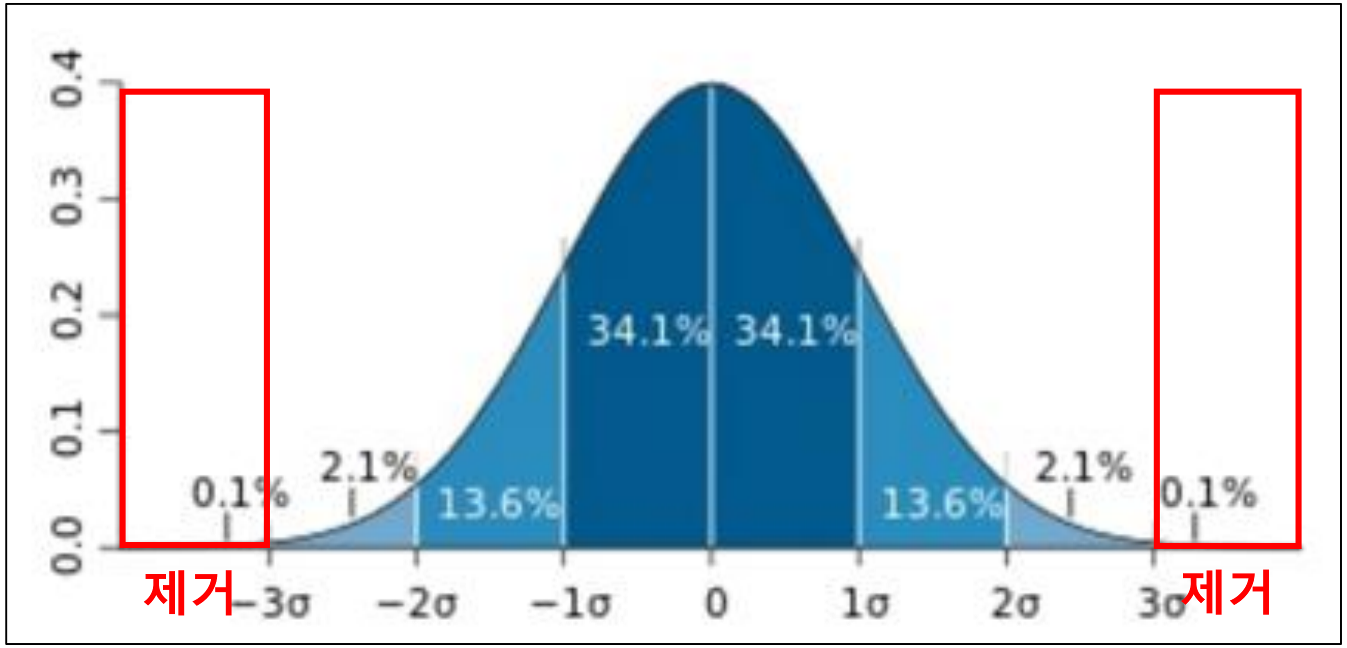
	mdutype	group	viewCount
iduser			
10100018739106	NaN	sdu	12.0
10100037810674	NaN	sdu	23.0
10100036273719	NaN	sdu	4.0
10100027752244	NaN	sdu	6.0
10100000624840	NaN	sdu	NaN

Preprocessing with Pandas

- Handling Missing Values
- Handling Outliers

Handling Outliers in Pandas

- 표준 점수 방식
  - 수치형 데이터의 경우
  - 표준 점수 (μ = 0, σ = 1)로 변환 후, ±3σ 넘는 값 제거



정규 분포 밀도 함수에서 $Z = \frac{X-\mu}{\sigma}$를 통해 X를 Z로 정규화 함으로써 평균이 0, 표준편차가 1인 표준정규분포를 얻을 수 있음. (z-분포라고도 불림)

Handling Outliers in Pandas

```
df = df.dropna(how='any')
df2 = df.select_dtypes(include=[np.number])
df2.head()
```

iduser	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
10100022538111	35.0	68.0	1.0	0.0	0.0	...	0.0	934912.0	92672.0	0.0	1027584.0
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0
10100017371337	95.0	19.0	0.0	12.0	0.0	...	0.0	25970473.0	8772492.0	0.0	34742965.0
10100013627062	75.0	15.0	0.0	3.0	0.0	...	0.0	1289983.0	271360.0	0.0	1561343.0

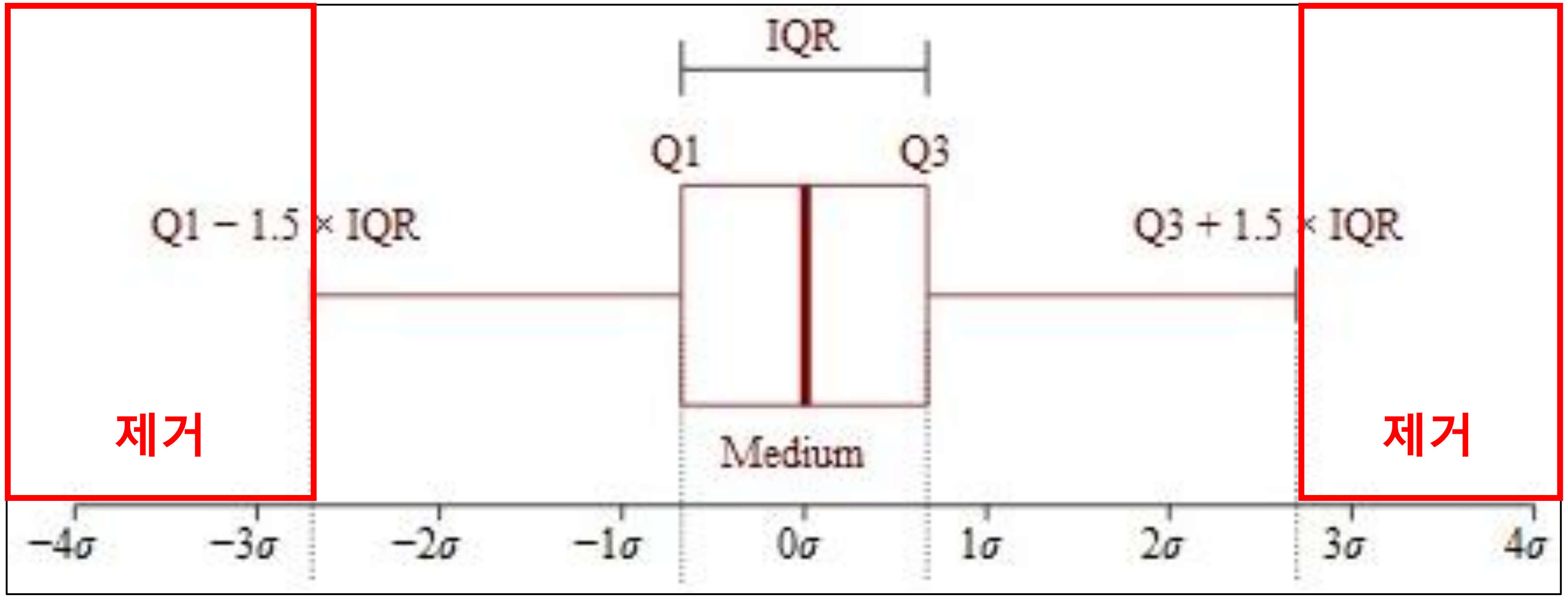
```
for i in range(len(df2.iloc[0])):
    df2 = df2[np.abs(df2.iloc[:,i] - df2.iloc[:,i].mean()) <= 3 * df2.iloc[:,i].std()]
df2.head()
```

$= |x - \mu| \leq 3\sigma$

iduser	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0
10100017371337	95.0	19.0	0.0	12.0	0.0	...	0.0	25970473.0	8772492.0	0.0	34742965.0
10100013627062	75.0	15.0	0.0	3.0	0.0	...	0.0	1289983.0	271360.0	0.0	1561343.0
10100012989173	49.0	0.0	2.0	13.0	0.0	...	0.0	2071600.0	51129.0	0.0	2122729.0

Handling Outliers using IQR (1)

- IQR (InterQuartile Range)
  - Q_1 = 25%, Q_3 = 75%, IQR = Q_3 - Q_1
  - Q_1 - 1.5IRQ 이하, Q_3 + 1.5IQR 이상인 값 제거



Handling Outliers using IQR (2)

- *numpy*.`percentile`(*a*, *q*, [*axis*], ...)
  - Compute the q-th percentile of the data along the specified axis
  - *a*: input array
  - *q*: percentile or sequence of percentiles to compute in [0, 100]
  - *axis*: axis or axes along which the percentiles are computed

```
a = np.random.randint(0, 20, 10)
a = np.sort(a)
```

```
array([ 4,  5,  5,  9, 13, 13, 14, 14, 17, 18])
```

```
np.percentile(a, 50)
```

```
13.0
```

```
np.percentile(a, [25, 75])
```

```
array([ 6., 14.])
```

```
a[np.where(a > 14)]
```

```
array([17, 18])
```

Handling Outliers using IQR (3)

```
def outliers_iqr(a):
    Q1, Q3 = np.percentile(a, [25, 75])
    IQR = Q3 - Q1
    lower_bound = Q1 - (IQR * 1.5)
    upper_bound = Q3 + (IQR * 1.5)
    return a[np.where(np.logical_or(a > upper_bound, a < lower_bound))]

x = np.arange(20)
x = np.hstack((x, np.arange(4) + 50))
print('data: ', x)
print('outliers: ', outliers_iqr(x))
```

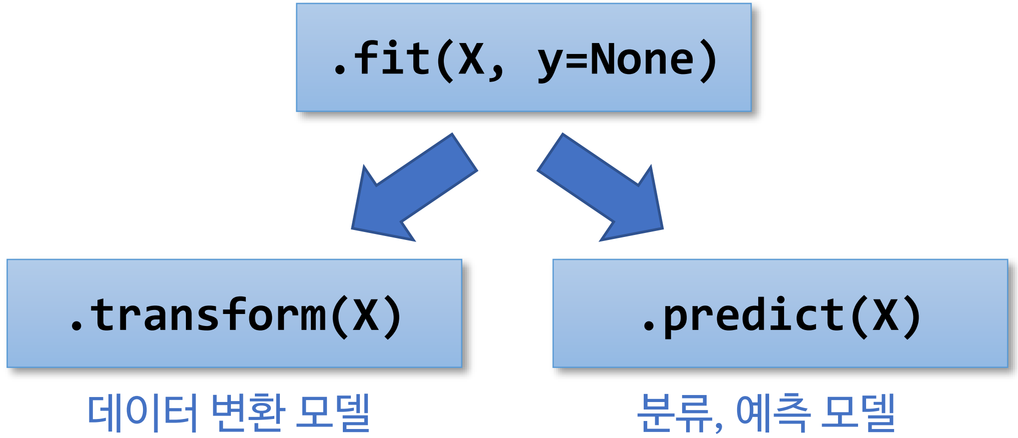
```
data:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 50 51 52 53]
outliers:  [50 51 52 53]
```

Preprocessing with SK-Learn

- Distribution Transformation
- Encoding
- Data Scaling

Preprocessing Steps of SK-Learn

- 일반적인 SK-Learn 모델 실행 과정



- `.fit()` 함수
  - 모델을 데이터에 맞게 학습하는 과정
  - 분류, 예측 모델의 경우 y 값 필요
- `.transform()` 함수
  - 데이터 변환이 필요한 모델에서 제공 (e.g., PCA, scaling, etc.)
  - 학습(fit)된 모델에 맞게 입력 데이터 변형
- `.predict()` 함수
  - 분류 및 예측 모델에서 제공 (e.g., regression, classification, etc.)
  - 학습된 모델을 기반으로 test data의 결과를 예측하는 함수

Example: StandardScaler()

- StandardScaler는 아래 수식을 적용하여 데이터를 변환하는 모델

$$\widetilde{x}_i = \frac{x_i - mean(x)}{std(x)}$$

- `scaler.fit()`
  - 생성한 데이터를 이용해 StandardScaler를 학습. mean, var(σ2) 값 계산
- `scaler.transform()`
  - 학습한 모델을 이용하여 X를 X_로 변환

```
import numpy as np
import sklearn.preprocessing as prep
```

```
X = np.arange(5, dtype='float').reshape(5, 1)
X
```

```
array([[0.],
       [1.],
       [2.],
       [3.],
       [4.]])
```

```
scaler = prep.StandardScaler()
scaler.fit(X)
scaler.mean_, scaler.var_
```

```
(array([2.]), array([2.]))
```

```
X_ = scaler.transform(X)
X_
```

```
array([[-1.41421356],
       [-0.70710678],
       [ 0.        ],
       [ 0.70710678],
       [ 1.41421356]])
```

Example: LinearRegression()

- Linear regression

```
import numpy as np
from sklearn import linear_model

X = np.random.random((5,3))
y = np.random.random((5,1))
X, y
```

```
array([[0.34523274, 0.03465153, 0.49879222],
       [0.34154268, 0.558655  , 0.05047529],
       [0.55781272, 0.93527388, 0.59078667],
       [0.7568254 , 0.57266255, 0.90788885],
       [0.42153745, 0.09800326, 0.69636864]])
array([[0.02940408],
       [0.41239473],
       [0.1845568 ],
       [0.78603924],
       [0.33245886]])
```

- `regr.fit()`
  - Linear regression 학습
  - $y = -0.42+3.58x_1-0.69x_2-1.22x_3$

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
regr.coef_, regr.intercept_
```

```
(array([[ 3.58372982, -0.68867795, -1.21704883]]), array
([-0.41676285]))
```

- `regr.predict()`
  - 학습한 linear regression 식을 기반으로 test 데이터의 예측 값 계산

```
test = np.random.random((2,3))
print('test\n', test)
regr.predict(test)
```

```
test
 [[0.1394362  0.8556813  0.437153  ]
 [0.56918605 0.05113164 0.42308297]]

array([[-1.03838658],
       [ 1.07292029]])
```

Dataset for Distribution Transformation

- 1970년 Boston 지역별 주택 가격 데이터셋
  - 주택 관련 여러 속성과 집 값을 정리한 데이터
- 506개 데이터, 14개 column (13개 속성 + 집 값)

CRIM: 범죄율
ZN: 25,000ft^2 초과 거주지역 비율
INDUS: 비소매상업지역 면접 비율
CHAS: 찰스강 경계에 위치한 경우 1
NOX: 일산화질소 농도
AGE: 1940년 이전 건축된 주택 비율
RM: 주택당 방 수
RAD: 방사형 고속도로까지의 거리
LSTAT: 인구 중 하위 계층 비율
DIS: 직업 센터의 거리
B: 인구 중 흑인 비율
TAX: 재산세율
PTRATIO: 학생/교사 비율
MEDV: 주택 가격의 median (단위: $1,000)

```
from sklearn import datasets
import pandas as pd
boston = datasets.load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Distribution Transformation: scale()

- 대부분 모델은 변수가 특정 분포를 따른다고 가정
  - 선형 모델은 변수가 정규 분포와 유사할수록 성능 향상
- 분포의 특성에 따라 다양한 함수 사용 가능
  - Left shift: X^3
  - Mild left shift: X^2
  - Mild right shift: sqrt(X)
  - Right shift: ln(X)

```
from sklearn import preprocessing

# For specific column(s)
df['LSTAT_log'] = preprocessing.scale(np.log(df['LSTAT']+1))
df['LSTAT_sqrt'] = preprocessing.scale(np.sqrt(df['LSTAT']+1))

# For all the dataframe (numeric data only)
df_log = df.apply(lambda x: x*x)
```

```
df[['LSTAT','LSTAT_log', 'LSTAT_sqrt']].head()
```

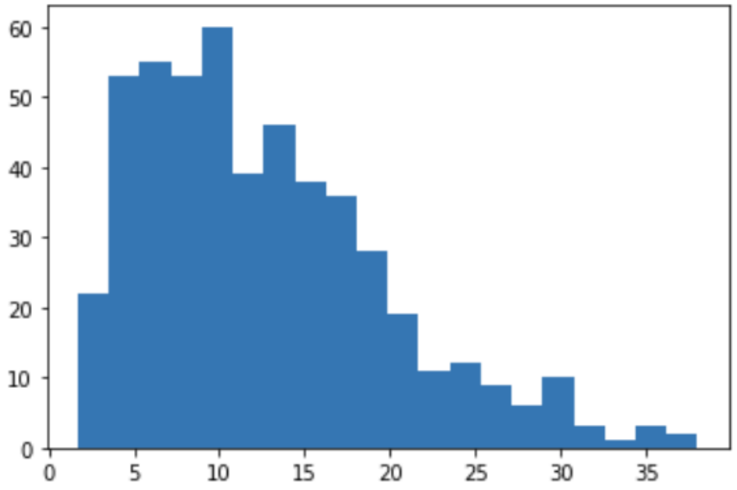
	LSTAT	LSTAT_log	LSTAT_sqrt
0	4.98	-1.276118	-1.195731
1	9.14	-0.295491	-0.411945
2	4.03	-1.597382	-1.410669
3	2.94	-2.050937	-1.684142
4	5.33	-1.170492	-1.120904

Distribution Transformation: scale()

- 변환 결과 정규 분포에 근접, 결과 확인 후 적절한 분포 선택

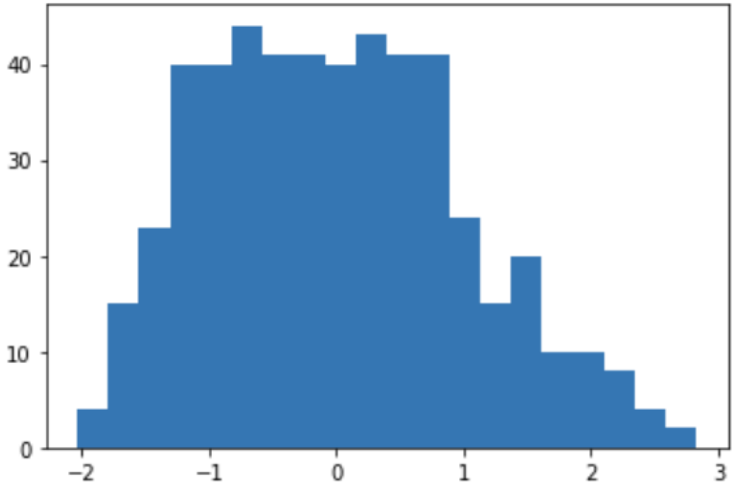
Original data

```python
import matplotlib.pyplot as plt

plt.hist(df['LSTAT'], bins=20)
plt.show()
```



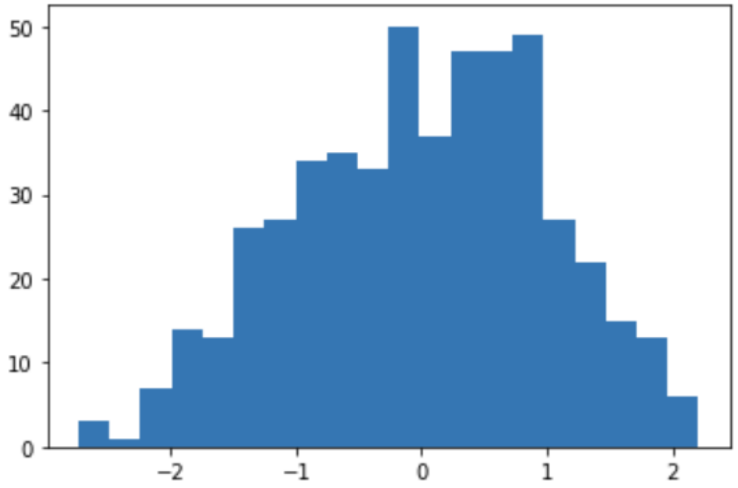
제곱근 변환시

```python
import matplotlib.pyplot as plt

plt.hist(df['LSTAT_sqrt'], bins=20)
plt.show()
```



Log 변환시

```python
import matplotlib.pyplot as plt

plt.hist(df['LSTAT_log'], bins=20)
plt.show()
```



Preprocessing with SK-Learn

- Distribution Transformation
- Encoding
- Data Scaling

One Hot Encoding

- 범주형 값이나 텍스트 정보를 처리 쉬운 정수로 변환하는 과정
- OneHotEncoder()
  - 0 ~ K-1 값을 가지는 정수 값을 0 또는 1의 값을 가지는 K-차원 벡터로 변환
  - 벡터 입력 시, 각 벡터의 변환 결과를 하나의 배열에 모두 연결하여 표현

정수 입력

```
array([[0],
       [1],
       [2],
       [3],
       [4]])
```

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

벡터 입력

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

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

벡터 간 구분 지점

SK-Learn OneHotEncoder()

- *sklearn.preprocessing.*`OneHotEncoder`([*sparse*], ...)
  - Encode categorial features using a one-hot numeric array
  - *sparse*: if True, return sparse matrix else return an array (default: True)

- Attributes
  - *categories_*: the categories of each feature determined during fitting

- Methods
  - `fit`(X, [y]): fit OneHotEncoder to X
  - `transform`(X): transform X using one-hot encoding
  - `inverse_transform`(X): convert the data back to the original representation

OneHotEncoder(): 1D data

- OneHotEncoder의 결과는 대부분이 0으로 구성된 sparse matrix 이므로, transform시 SciPy의 sparse matrix 형태로 변환됨
- 직관적으로 확인하기 위해서 toarray() 함수를 이용하여 dense matrix로 변환 가능

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()
X = np.array(['a', 'b', 'a', 'c']).reshape(-1, 1)
ohe.fit(X)
X_encode = ohe.transform(X)
type(X_encode)
```

```
scipy.sparse.csr.csr_matrix
```

```
X_encode.toarray()
```

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

```
ohe.inverse_transform([[0., 1., 0.]])
```

```
array([['b']], dtype='<U1')
```

OneHotEncoder(): 2D data

```
X = np.array([[0, 0, 4],
              [1, 1, 0],
              [0, 2, 1],
              [1, 0, 2]])
ohe.fit(X)
X_encode = ohe.transform(X)
X_encode.toarray()
```

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

```
ohe.inverse_transform([[1., 0., 0., 0., 1., 0., 0., 0., 1.]])
```

```
array([[0, 2, 4]], dtype=int32)
```

SK-Learn SimpleImputer()

- *sklearn.impute.*`SimpleImputer`([*missing_values*], [*strategy*], [*fill_value*], ...)
  - Imputation transformer for completing missing values
  - *missing_values*: the placeholder for the missing values (default: np.nan)
  - *strategy*: 'mean', 'median', 'most_frequent', or 'constant' (default: 'mean')
  - *fill_value*: the value used to replace all occurrences of missing_values when strategy == 'constant'

- Methods
  - `fit`(X, [y]): fit the imputer on X
  - `transform`(X): Impute all missing values in X

SimpleImputer() Example

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer()
X = np.array([[1, 2], [np.nan, 3], [7, np.nan]])
imp.fit_transform(X)
```

```
array([[1. , 2. ],
       [4. , 3. ],
       [7. , 2.5]])
```

average of the corresponding column

```
imp = SimpleImputer(strategy='constant', fill_value = -1)
imp.fit_transform(X)
```

```
array([[ 1.,  2.],
       [-1.,  3.],
       [ 7., -1.]])
```

SK-Learn Binarizer()

- *sklearn.preprocessing.*`Binarizer(`[*threshold*], ...`)`
  - Binarizer data (set feature values to 0 or 1) according to a threshold
  - *threshold*: feature values below or equal to this are replaced by 0, above it by 1 (default: 0.0)

- Methods
  - `fit`(X, [y]): do nothing
  - `transform`(X): binarize each element of X

Binarizer() Example

```
from sklearn.preprocessing import Binarizer
X = np.array([[1., -1.,  2.],
              [2., -3.,  1.],
              [0.,  1., -1.]])
bin = Binarizer()
bin.transform(X)
```

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

```
bin = Binarizer(threshold = 1)
bin.transform(X)
```

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

SK-Learn: PolynomialFeatures()

- *sklearn.preprocessing.*`PolynomialFeatures`([*degree*], [*interaction_only*], [*include_bias*], …)
  - Generate polynomial and interaction features
  - *degree*: the degree of the polynomial features (default: 2)
  - *interaction_only*: if True, only interaction features are produced (default: False)
  - *include_bias*: if True (default), include a bias column (polynomial powers are zero)

$$x \rightarrow [1, x, x^2, x^3, \ldots]$$

- Methods
  - `fit`(X, [y]): compute number of output features
  - `transform`(X): transform data to polynomial features

PolynomialFeatures() Example

```
from sklearn.preprocessing import PolynomialFeatures
X = np.array([1, 2, 3]).reshape(3, 1)
poly = PolynomialFeatures(degree = 4)
poly.fit_transform(X)
```

```
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  2.,  4.,  8., 16.],
       [ 1.,  3.,  9., 27., 81.]])
```

[1, a, aa, aaa, aaaa]

```
X = np.array([1, 2, 3, 4, 5, 6]).reshape(3, 2)
poly = PolynomialFeatures()
poly.fit_transform(X)
```

```
array([[ 1.,  1.,  2.,  1.,  2.,  4.],
       [ 1.,  3.,  4.,  9., 12., 16.],
       [ 1.,  5.,  6., 25., 30., 36.]])
```

[1, a, b, aa, ab, bb]

SK-Learn FunctionTransformer()

- *sklearn.preprocessing.*`FunctionTransformer`([*func*], [*inverse_func*], ...)
  - Constructs a transformer from an arbitrary callable
  - *func*: the callable to use for the transformation
  - *inverse_func*: the callable to use for the inverse transformation

$$\boxed{x \rightarrow [\, f_1(x), f_2(x), f_3(x), \ldots \,]}$$

- Methods
  - `fit`(X, [y]): fit transformer by checking X
  - `transform`(X): transform X using the forward function

FunctionTransformer() Example

```
from sklearn.preprocessing import FunctionTransformer
def kernel(X):
    x0 = X[:, :1]
    x1 = X[:, 1:2]
    x2 = X[:, 2:3]
    X_new = np.hstack([x0, 2*x1, x2**2, np.log(x2)])
    return X_new

X = np.arange(12).reshape(4, 3)
ft = FunctionTransformer(kernel)
ft.transform(X)
```

```
array([[ 0.        ,  2.        ,  4.        ,  0.69314718],
       [ 3.        ,  8.        , 25.        ,  1.60943791],
       [ 6.        , 14.        , 64.        ,  2.07944154],
       [ 9.        , 20.        , 121.       ,  2.39789527]])
```

x_0 $2x_1$ x_2^2 $\ln(x_2)$

SK-Learn LabelEncoder()

- *sklearn.preprocessing.*`LabelEncoder()`
  - Encode target labels with value between 0 and n_classes-1
- Methods
  - `fit`(X, [y]): fit label encoder
  - `transform`(X): transform labels to normalized encoding
  - `inverse_transform`(y): transform labels back to original encoding

```
from sklearn.preprocessing import LabelEncoder
X = ['A', 'B', 'A', 'A', 'B', 'C', 'C', 'A', 'C', 'B']
le = LabelEncoder()
le.fit_transform(X)
```

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

Preprocessing with SK-Learn

- Distribution Transformation
- Encoding
- Data Scaling

Data Scaling

- 데이터 측정 단위가 다를 경우 모델에 부정적 영향
  - 단위를 일정하게 통일해야 함

- 일반적인 Scaling의 의미
  - 모든 데이터에 선형 변환을 적용하여 $\mu = 0, \sigma = 1$로 변환

- Scaling의 효과
  - Overflow 및 underflow 방지
  - 최적화 과정의 안정성 및 수렴 속도 향상

Data Scaling: Taxonomy

- Standard scaling: $\mu = 0, \sigma = 1$인 분포로 변환
- Min-Max scaling: 특정 범위(0~1)로 데이터 변환
- Max-Abs scaling: 최대 절대값이 1이 되도록 변환
- Robust scaling: Median, IQR 사용.
  Outlier 영향 최소화

$$\tilde{x}_i = \frac{x_i - mean(x)}{std(x)}$$

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

$$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$

$$\tilde{x}_i = \frac{x_i - median(x)}{Q3(x) - Q1(x)}$$

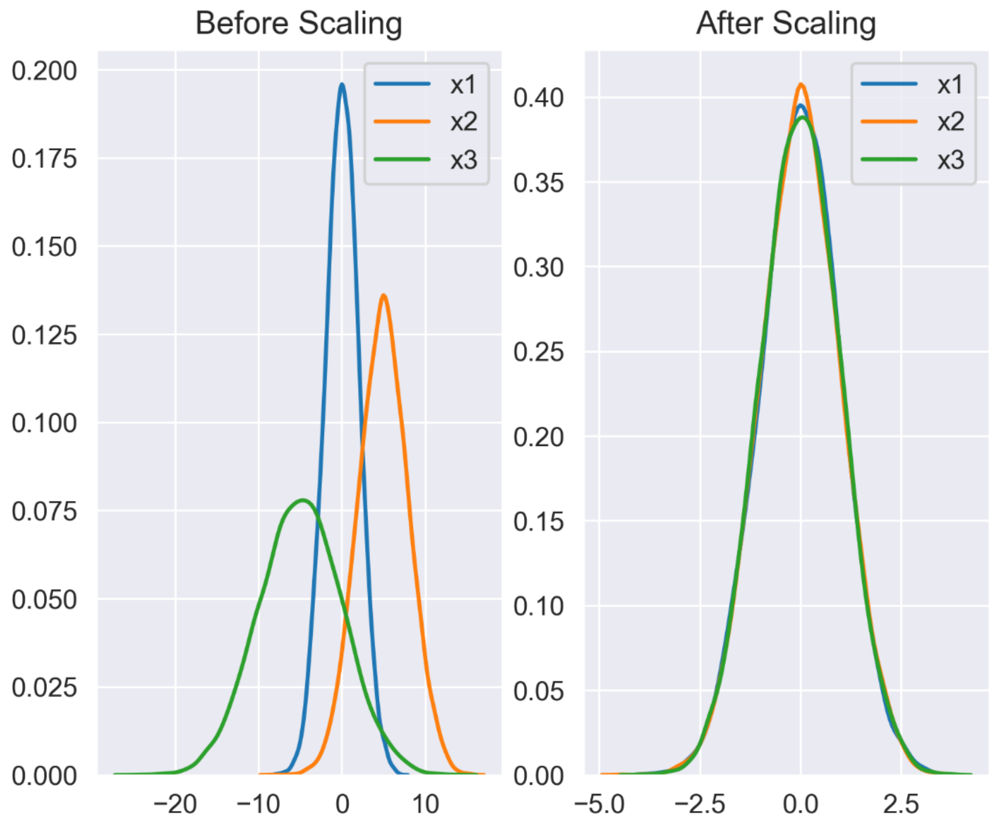
- SK-Learn에서의 scaling

Scaling 방식	Function	Class
Standard	`scale(x)`	`StandardScaler`
Min-Max	`minmax_scale()`	`MinMaxScaler`
Max-Abs	`maxabs_scale()`	`MaxAbsScaler`
Robust	`robust_scale()`	`RobustScaler`

Standard Scaling

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing

df = pd.DataFrame({
        'x1': np.random.normal(0, 2, 10000),
        'x2': np.random.normal(5, 3, 10000),
        'x3': np.random.normal(-5, 5, 10000)
})
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

sns.set_style('darkgrid')
_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

$$\tilde{x}_i = \frac{x_i - mean(x)}{std(x)}$$

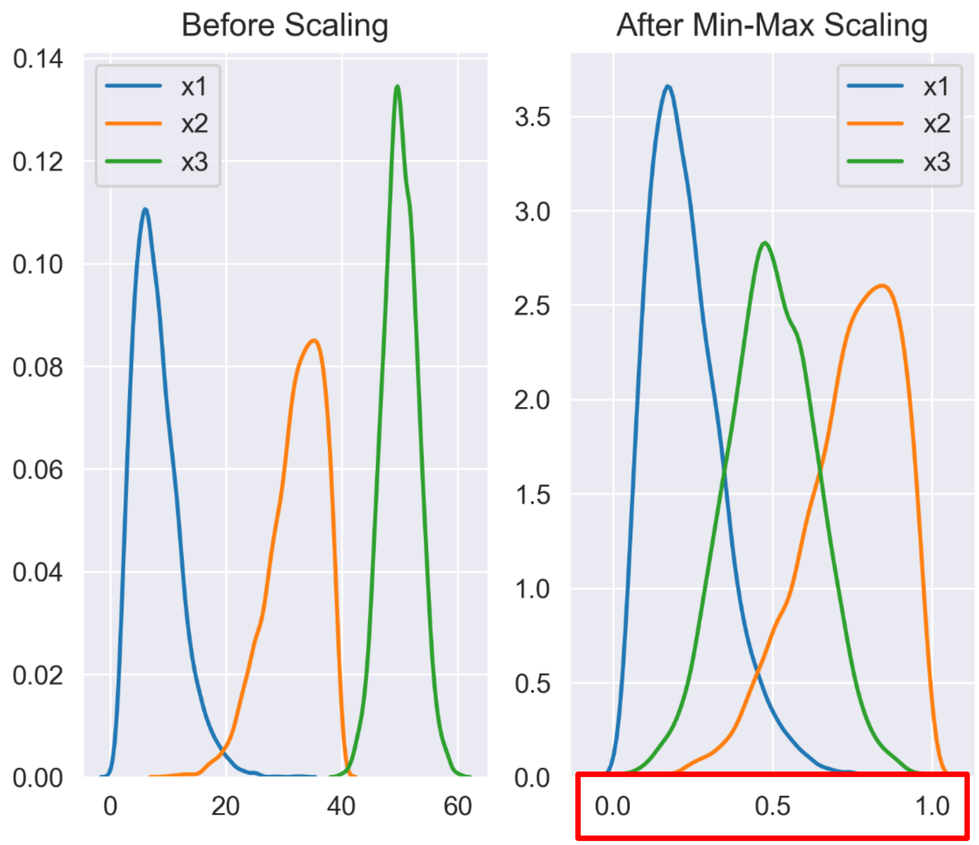


Min-Max Scaling

- 정규분포가 아닌 경우 유용
- 최대, 최소값 정보를 이용하므로 outlier에 취약

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

```
df = pd.DataFrame({
        'x1': np.random.chisquare(8, 10000),     # positive skew
        'x2': np.random.beta(8, 2, 10000)*40,    # negative skew
        'x3': np.random.normal(50, 3, 10000)     # no skew
})

scaler = preprocessing.MinMaxScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Min-Max Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

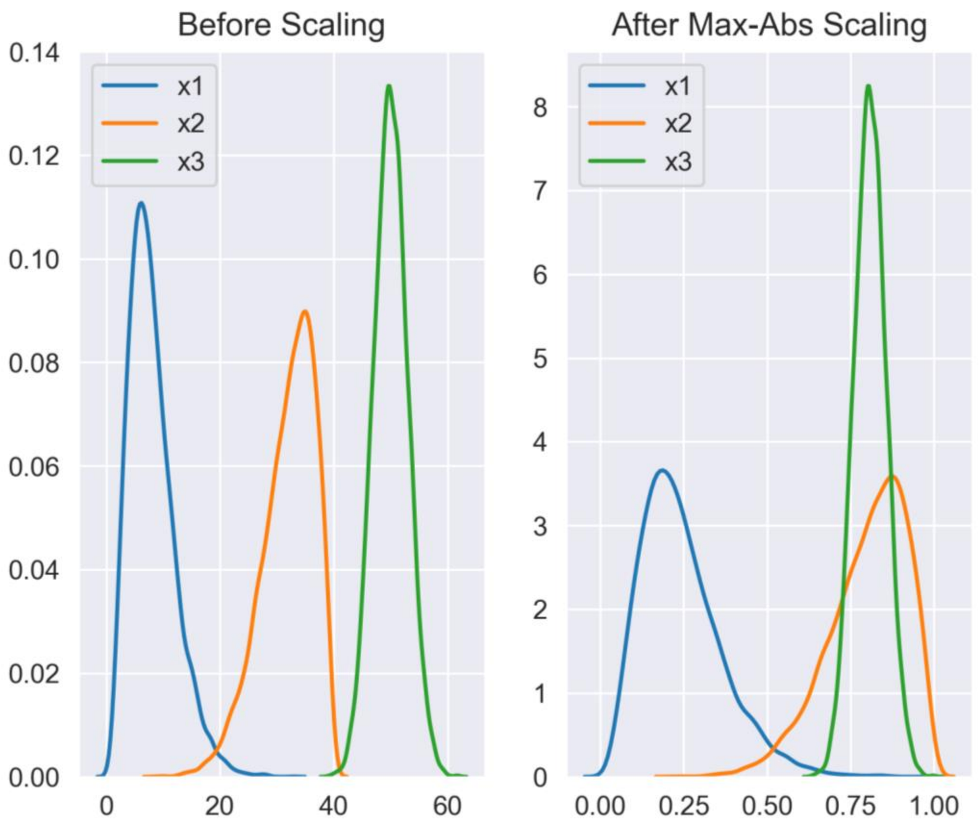


Max-Abs Scaling

- Shift나 center 작업없이 기존 분포를 온전히 유지
- Outlier에 취약

$$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$

```python
df = pd.DataFrame({
        'x1': np.random.chisquare(8, 10000),     # positive skew
        'x2': np.random.beta(8, 2, 10000)*40,    # negative skew
        'x3': np.random.normal(50, 3, 10000)     # no skew
})

scaler = preprocessing.MaxAbsScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Max-Abs Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```



Robust Scaling (1)

- Median이나 IQR같이 outlier의 영향을 크게 받지 않는 값들을 이용하므로 outlier 영향 최소화

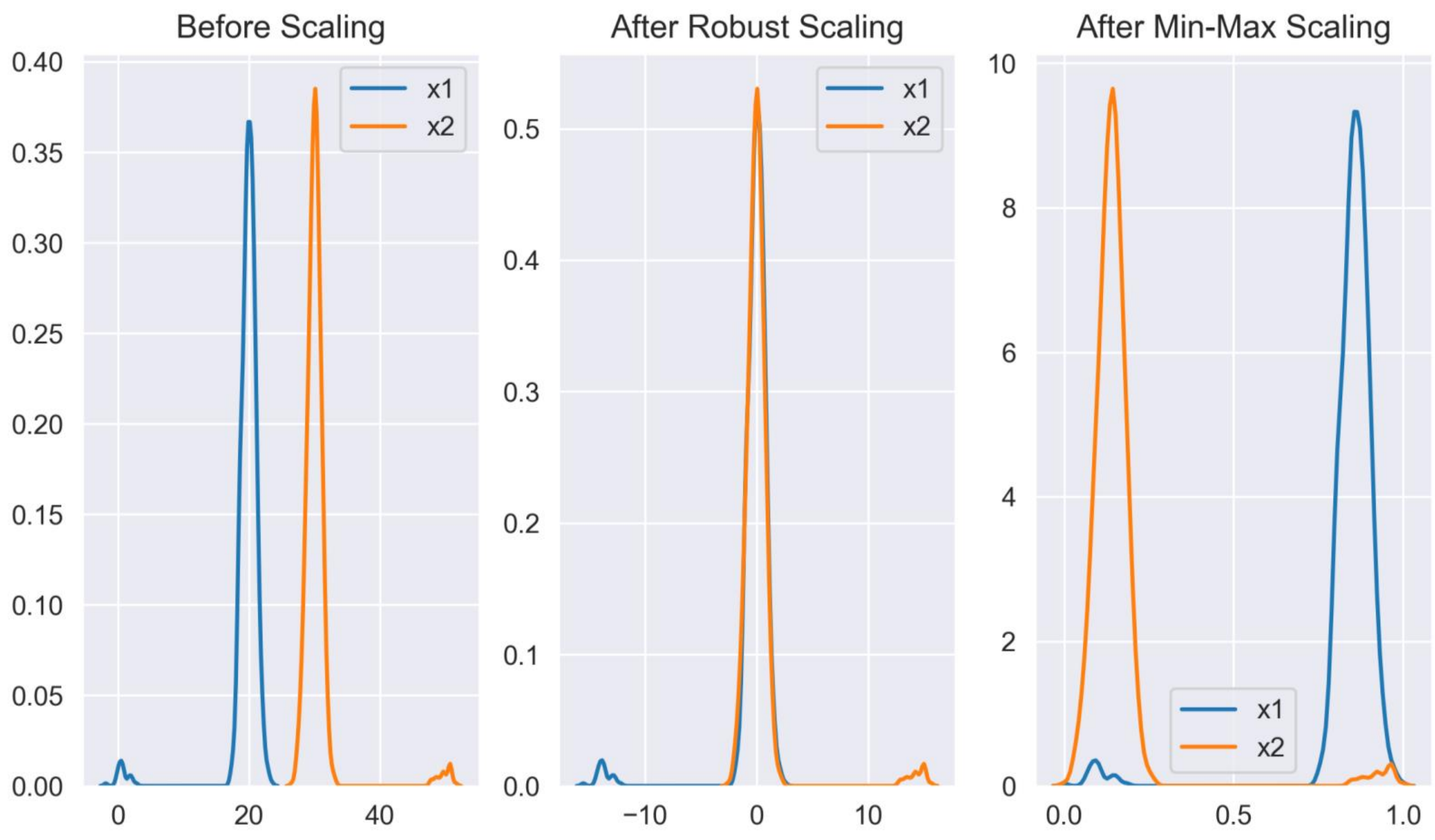
$$\tilde{x}_i = \frac{x_i - median(x)}{Q3(x)\ - Q1(x)}$$

```
df = pd.DataFrame({
        # distribution with lower outliers
        'x1': np.hstack((np.random.normal(20,1,1000),
                         np.random.normal(1,1,25))),
        # distribution with upper outliers
        'x2': np.hstack((np.random.normal(30,1,1000),
                         np.random.normal(50,1,25)))
})

robust_scaler = preprocessing.RobustScaler()
robust_df = robust_scaler.fit_transform(df)
robust_df = pd.DataFrame(robust_df, columns=['x1', 'x2'])

minmax_scaler = preprocessing.MinMaxScaler()
minmax_df = minmax_scaler.fit_transform(df)
minmax_df = pd.DataFrame(minmax_df, columns=['x1', 'x2'])
```

```
_, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(9,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df['x1'], ax=ax2)
sns.kdeplot(robust_df['x2'], ax=ax2)

ax3.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df['x1'], ax=ax3)
sns.kdeplot(minmax_df['x2'], ax=ax3)
```

Robust Scaling (2)

- Min-Max Scaling
  - 두 정규 분포가 outlier에 의해 완전히 분리되어 버림
  - Inlier들은 [0, 0.25]에 위치
- Robust Scaling
  - 두 분포를 같은 기준으로 scale
  - Outlier는 여전히 변환된 분포의 바깥쪽에 위치
  - Inlier들은 [-2, 2]에 위치



Data Scaling vs. Normalization

- Data normalization
  - 각 sample의 크기를 unit norm으로 만드는 과정
  - Scaling과 다르게 개별 데이터의 크기를 동일하게 만드는 과정
  - 다차원 독립변수 벡터의 각 원소들의 상대적 크기만 중요한 경우 사용 (e.g, text classification and clustering)
  - sklearn.preprocessing.normalize()

$$\hat{x}_i = \frac{x_i}{||x||\ (= norm)}$$

- Data scaling vs. normalization
  - Data scaling은 column에 포함된 데이터들의 변환
  - Normalization은 row에 포함된 데이터들의 변환

Sampling with Imbalanced Learn

Imbalanced Data

- 클래스 비율이 너무 차이가 나는 경우
  - Majority Class를 예측하는 모델의 정확도가 너무 높아져 성능 판별 어려움
- 예제
  - Class A, B의 비율이 9:1인 dataset
  - 만약 test dataset의 Class A, B 정답비율이 9:1
  - Model M은 test 과정에서 무조건 Class A라고 판정
    → Class B 데이터를 하나도 분류하지 못해도 Model M의 정확도는 90%

- Sampling을 사용하여 데이터의 비율 조정
  - Oversampling: Minority Class의 데이터를 증가
  - Undersampling: Majority Class에서 데이터의 일부만 사용

imbalanced-learn module

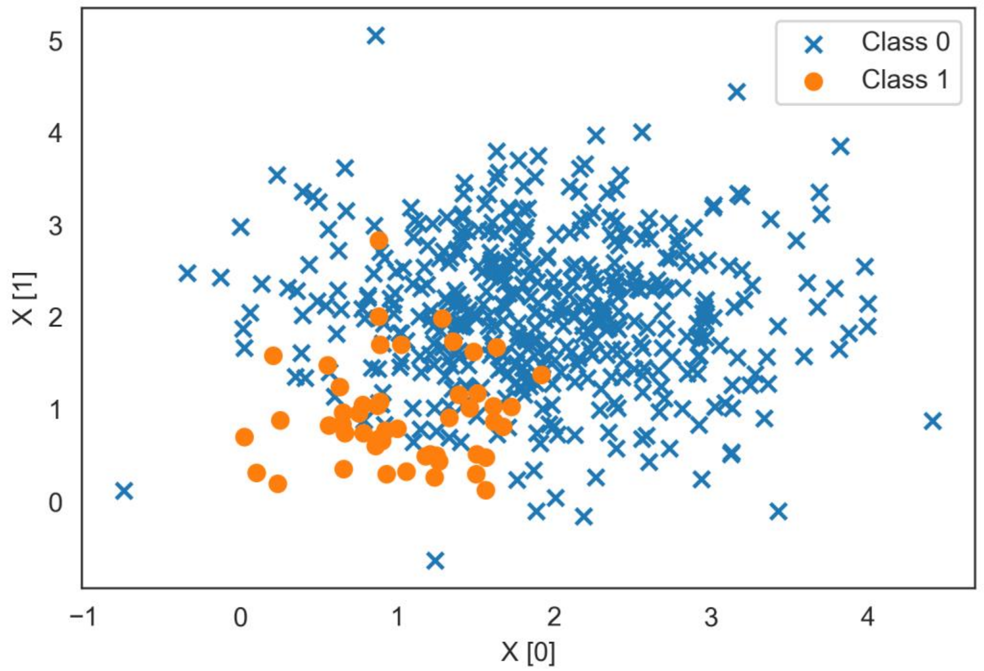
- A python package offering a number of re-sampling techniques
- Commonly used for datasets showing strong between-class imbalance
- Part of scikit-learn-contrib projects
- https://github.com/scikit-learn-contrib/imbalanced-learn

- Installation
  - pip install -U imbalanced-learn
  - conda install -c conda-forge imbalanced-learn

```
>>> import imblearn.under_sampling
>>> import imblearn.over_sampling
```

Creating Imbalanced Data

```
def plot(X, y):
    plt.scatter(X[y==0, 0], X[y==0, 1], marker='x', label='Class 0')
    plt.scatter(X[y==1, 0], X[y==1, 1], marker='o', label='Class 1')
    plt.xlabel('X [0]')
    plt.ylabel('X [1]')
    plt.legend()

n0 = 450
n1 = 50

a = np.random.randn(n0, 2)*0.8 + 2  # N(2, 0.8)
b = np.random.randn(n1, 2)*0.5 + 1  # N(1, 0.5)

X = np.vstack([a, b])
y = np.hstack([np.zeros(n0), np.ones(n1)])

plot(X, y)
```

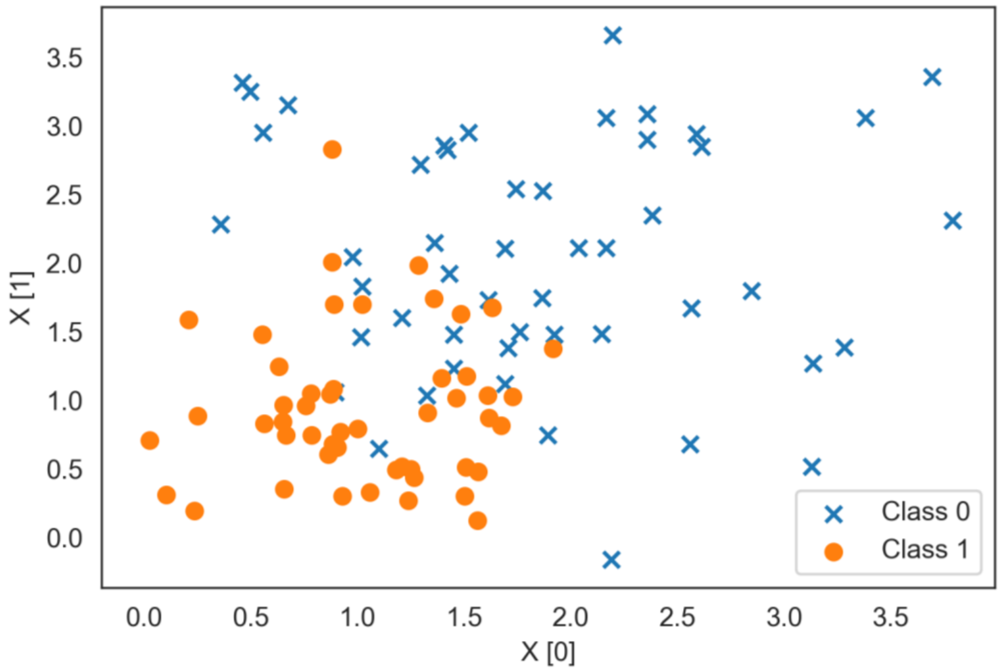


Undersampling: RandomUnderSampler()

- 임의로 다수 클래스 데이터의 일부를 버림

```
from imblearn.under_sampling import RandomUnderSampler

X_samp, y_samp = RandomUnderSampler(random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(100, 2) (100,)
```

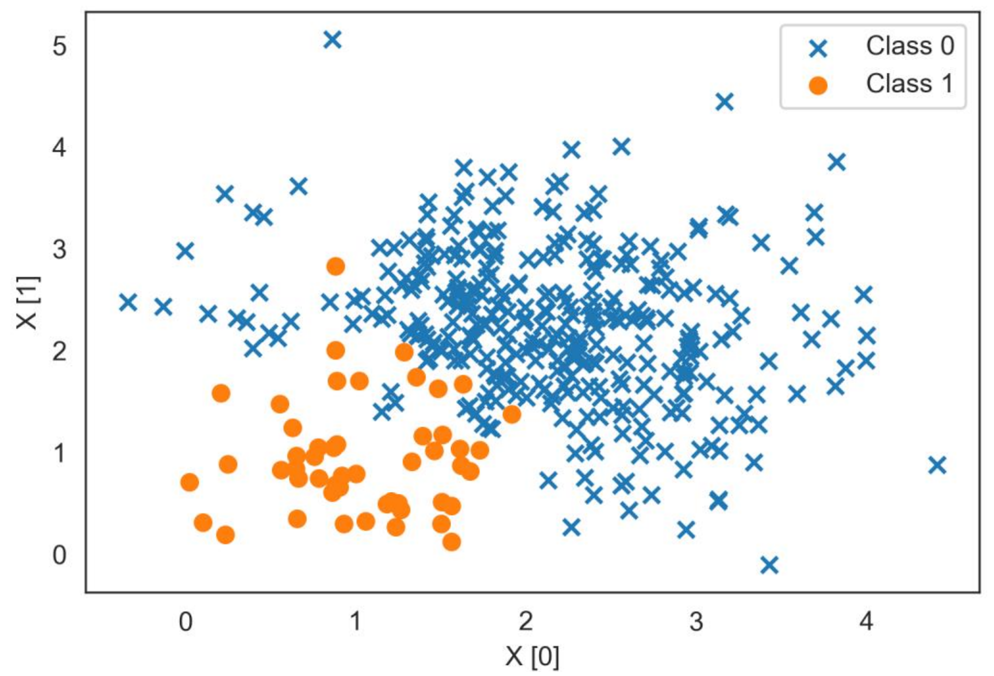


Undersampling: EditedNearestNeighbours()

- 다수 클래스 데이터 중 가까운 k개가 과반수 이상이 아닌 경우 삭제

```
from imblearn.under_sampling import EditedNearestNeighbours

X_samp, y_samp = EditedNearestNeighbours(kind_sel='all', n_neighbors=10,
                    random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(388, 2) (388,)
```



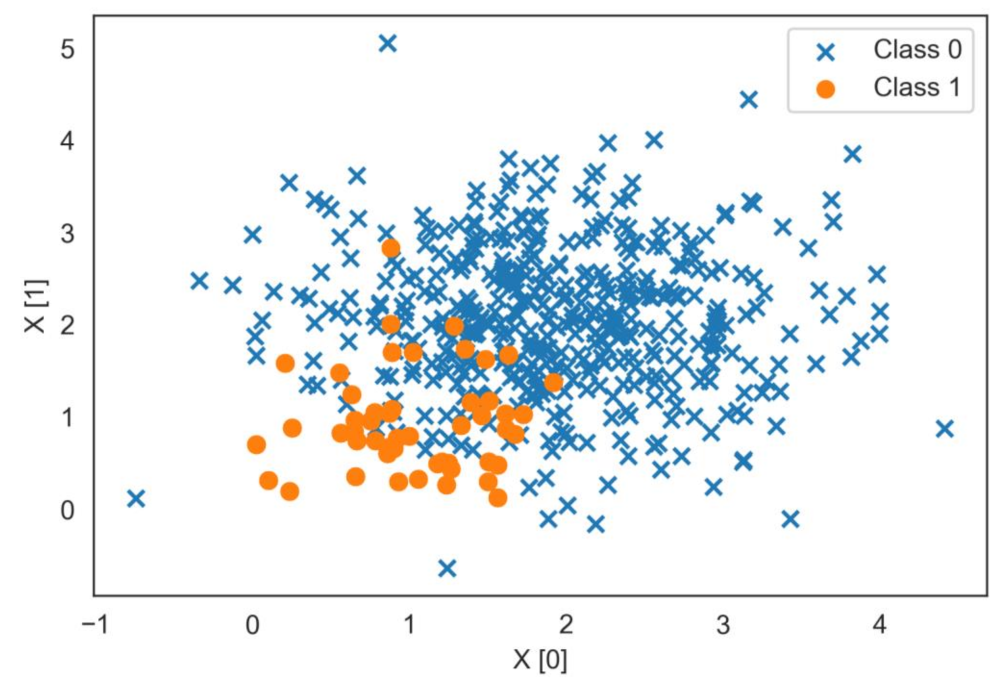
- *n_neighbors*: 비교할 k 개의 데이터 수
- *kind_sel*: 'all' (모두가 아니면 삭제),
  'mode' (과반수 이상이 아니면 삭제)

Oversampling: RandomOverSampler()

- 기존 소수 클래스 데이터를 여러 번 중복해서 선택

```
from imblearn.over_sampling import RandomOverSampler

X_samp, y_samp = RandomOverSampler(random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(900, 2) (900,)
```



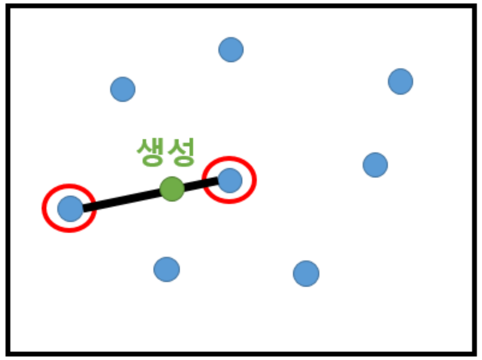
- 그래프는 동일

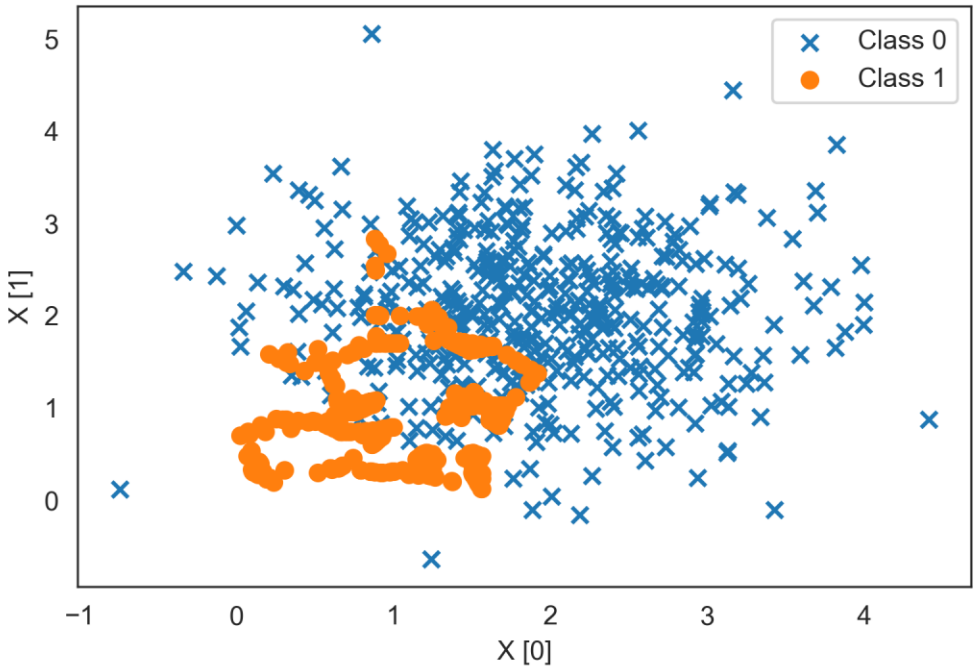
Oversampling: SMOTE()

- 소수 클래스 데이터와 인접한 k개 데이터 중 임의 선택된 데이터 간의 차에 0~1 사이의 값을 곱하여 기존 데이터에 더해주는 방식으로 생성

```
from imblearn.over_sampling import SMOTE

X_samp, y_samp = SMOTE(k_neighbors=3).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(900, 2) (900,)
```





Oversampling: ADASYN()

- SMOTE에 약간의 randomness 추가
- 밀도 분포를 고려하여 생성할 샘플의 수를 결정

```python
from imblearn.over_sampling import ADASYN

X_samp, y_samp = ADASYN(n_neighbors=3, random_
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(908, 2) (908,)
```