

# **Chapter 7. Preprocessing**

# 맛있는 요리가 나오기 위해서는?



# 신선한 재료 + 재료의 손질 + 레시피



신선한 재료



재료 손질



요리

# 데이터 과학도 마찬가지

- 기계학습 모델이 잘 동작하기 위해서는...
- 1. 양질의 데이터
- 2. 데이터에 대한 적절한 가공
- 3. 적합한 모델의 적용

# Iris dataset

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	4.9	3.0	1.4	0.2	setosa
1	4.7	3.2	1.3	0.2	setosa
2	4.6	3.1	1.5	0.2	setosa
3	6.4	3.2	4.5	1.5	versicolor
4	6.9	3.1	4.9	1.5	versicolor
5	5.5	2.3	4.0	1.3	versicolor
6	7.1	3.0	5.9	2.1	virginica
7	6.3	2.9	5.6	1.8	virginica
8	7.6	3.0	6.6	2.1	virginica

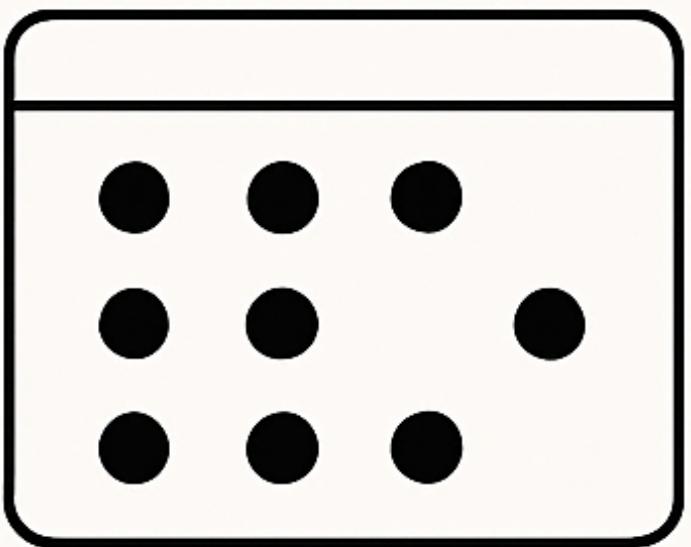


# 현실의 데이터는?

	School ID	Name	Address	City	Subject	Marks	Rank	Grade
0	101.0	Alice	123 Main St	Los Angeles	Math	85.0	2	B
1	102.0	Bob	456 Oak Ave	New York	English	92.0	1	A
2	103.0	Charlie	789 Pine Ln	Houston	Science	78.0	4	C
3	NaN	David	101 Elm St	Los Angeles	Math	89.0	3	B
4	105.0	Eva		Nan	Miami	History	Nan	D
5	106.0	Frank	222 Maple Rd		Nan	Math	95.0	1
6	107.0	Grace	444 Cedar Blvd	Houston	Science	80.0	5	C
7	108.0	Henry	555 Birch Dr	New York	English	88.0	3	B

# 현실의 데이터는?

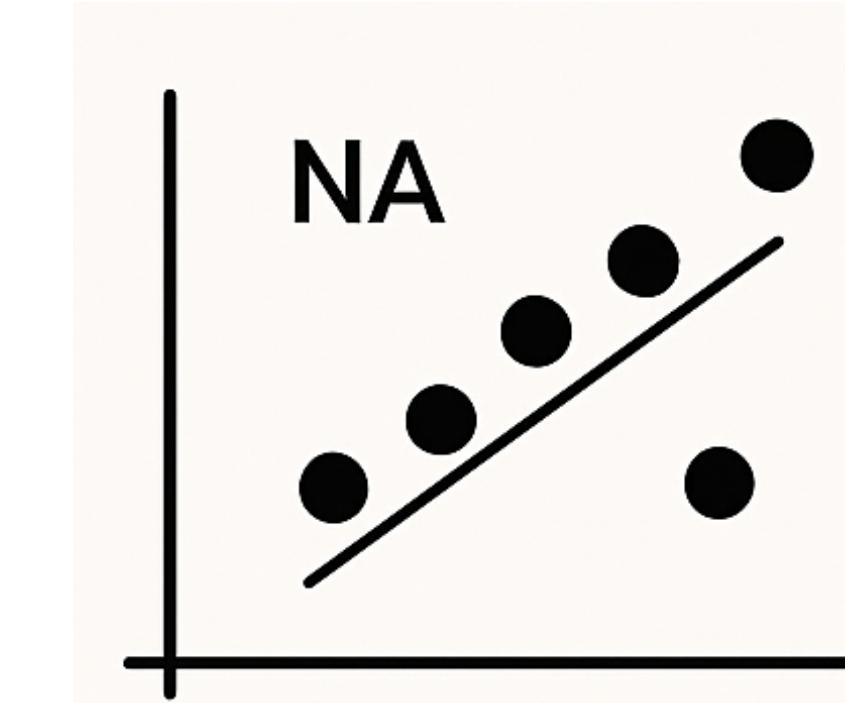
- 현실에서 수집된 데이터는 여러 문제점을 가지고 있음



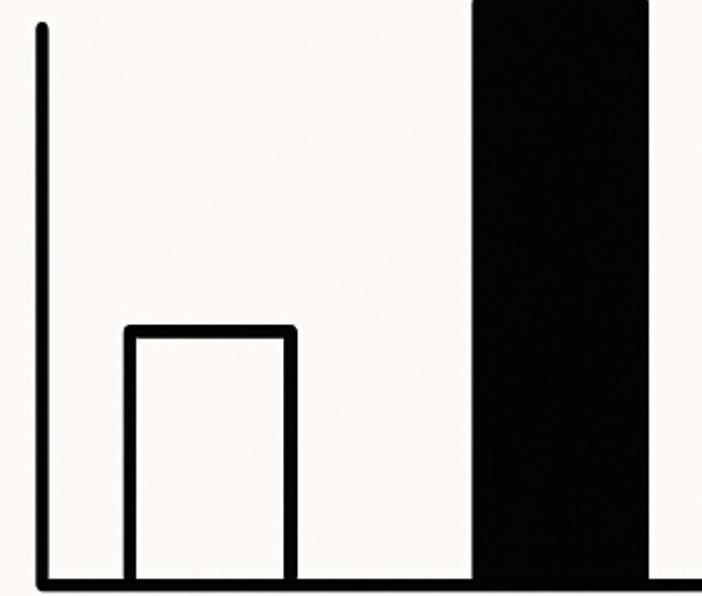
데이터 수 부족

X	$X_2$	$X_3$
•	•••	—
•	•••	이영희
•	•••	박정우

관련없는 정보 포함



누락값, 이상값



데이터 불균형

# 데이터 전처리

- 데이터 전처리 (Data preprocessing) 과정에서 데이터를 정제
- 모델학습에 적합한 형태로 재구성
- 주요 전처리 과정

누락값 / 이상값  
처리

인코딩

정규화  
(스케일링)

불균형 처리

# Titanic data

```
!wget https://raw.githubusercontent.com/shryu8902/KIRD_AUTOML/main/Titanic-Dataset.csv
import pandas as pd
df = pd.read_csv('./Titanic-Dataset.csv')
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

# 누락값 처리

- 누락값 또는 결측치 : 측정해야 할 데이터가 다양한 이유로 인해 기록되지 않은 경우
- 누락값 처리를 위한 주요 전략 3가지
- 누락값 제거 : 결측값이 포함된 행(row)이나 열(column)을 제거
- 누락값 대체 : 누락된 부분을 새로운 값으로 대체 (평균/최빈값/예측값)
- 누락 식별 변수 추가 : 결측값이 존재하는지 여부를 나타내는 binary feature를 새로 추가
  - 단 식별변수 추가는 누락값 대체와 함께 진행되어야함.
  - 일반적으로 ML모델은 NaN값을 다룰 수 없음.

# 누락값 탐색

df.isna()

- isna()를 통해 Null, NaN 값 유무를 확인 가능
- info(), 혹은 isna().sum() 호출을 통해 누락값의 수에 대한 정보 확보

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0		False	False	False	False	False	False	False	False	False	False	True
1		False	False	False	False	False	False	False	False	False	False	False
2		False	False	False	False	False	False	False	False	False	False	True
3		False	False	False	False	False	False	False	False	False	False	False
4		False	False	False	False	False	False	False	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...
886		False	False	False	False	False	False	False	False	False	False	False
887		False	False	False	False	False	False	False	False	False	False	False
888		False	False	False	False	False	True	False	False	False	False	True
889		False	False	False	False	False	False	False	False	False	False	False
890		False	False	False	False	False	False	False	False	False	False	True

891 rows × 12 columns

# 누락값 제거

- 데이터프레임 객체가 df로 주어져있을 때
- df.dropna() : 누락값이 포함된 행제거
- df.dropna(axis=1) : 누락값이 포함된 열제거

```
df.dropna().info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 183 entries, 1 to 889
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId  183 non-null    int64  
 1   Survived     183 non-null    int64  
 2   Pclass       183 non-null    int64  
 3   Name         183 non-null    object  
 4   Sex          183 non-null    object  
 5   Age          183 non-null    float64 
 6   SibSp        183 non-null    int64  
 7   Parch        183 non-null    int64  
 8   Ticket       183 non-null    object  
 9   Fare          183 non-null    float64 
 10  Cabin         183 non-null    object  
 11  Embarked     183 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

```
df.dropna(axis=1).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   SibSp        891 non-null    int64  
 6   Parch        891 non-null    int64  
 7   Ticket       891 non-null    object  
 8   Fare          891 non-null    float64 
dtypes: float64(1), int64(5), object(3)
memory usage: 62.8+ KB
```

# Simple Imputer

```
from sklearn.impute import SimpleImputer

mean_imputer = SimpleImputer(strategy='mean')
median_imputer = SimpleImputer(strategy='median')
constant_imputer = SimpleImputer(strategy='constant', fill_value=99)
most_frequent_imputer = SimpleImputer(strategy='most_frequent')

age_impute_mean = mean_imputer.fit_transform(df[['Age']])
age_impute_med = median_imputer.fit_transform(df[['Age']])
age_impute_const = constant_imputer.fit_transform(df[['Age']])
cabin_impute = most_frequent_imputer.fit_transform(df[['Embarked']])
```

# KNN imputer

```
from sklearn.impute import KNNImputer  
  
knn_imputer = KNNImputer(n_neighbors=5)  
  
age_impute_knn = knn_imputer.fit_transform(df[['Age']])
```

# Indicator

- 기존 imputer 클래스에서 add\_indicator 옵션 설정

```
mean_imputer_with_indicator = SimpleImputer(strategy='mean', add_indicator=True)  
age_impute_mean_with_indicator = mean_imputer_with_indicator.fit_transform(df[['Age']])
```

- 별도의 indicator 활용

```
from sklearn.impute import MissingIndicator  
  
indicator = MissingIndicator()  
  
indicator_array = indicator.fit_transform(df[['Age']])  
indicator_mat = indicator.fit_transform(df)
```

# 이상값 처리

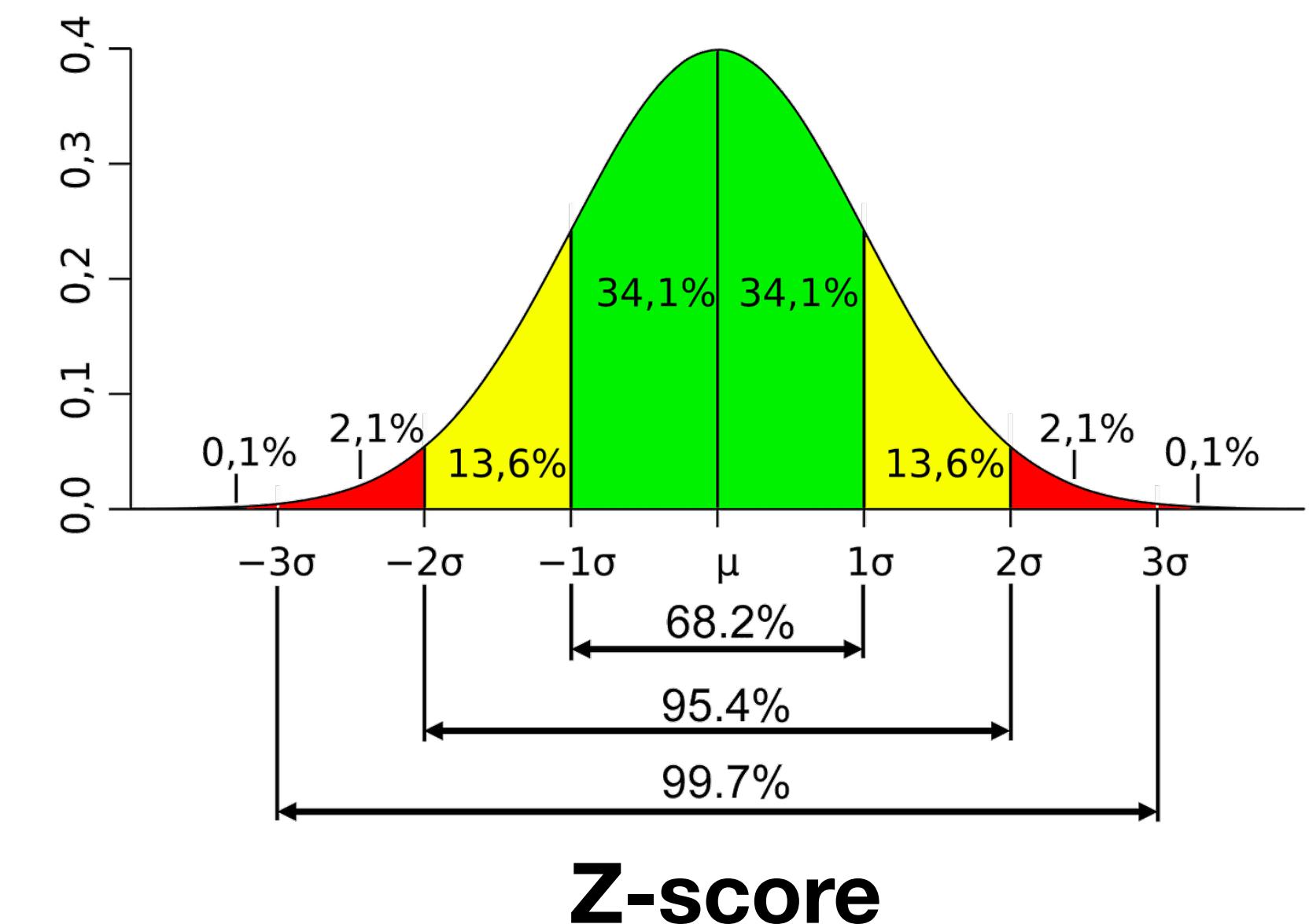
- 이상값 : 측정값의 분포에서 벗어난 outlier
- 기본적으로 이상값 처리 전략은 누락값과 같이 제거, 대체, 식별변수 추가 전략을 따름.
- 단, 어떠한 값을 이상값으로 볼 것인지에 대한 결정이 필요
  - Z-score
  - Box plot
  - EDA를 통한 manual search
  - 기타 ML 기법을 통한 이상치 탐색 - isolation forest, DBSCAN 등

# 이상값 처리 - Z-score

- 데이터가 정규분포를 따르는 것으로 가정

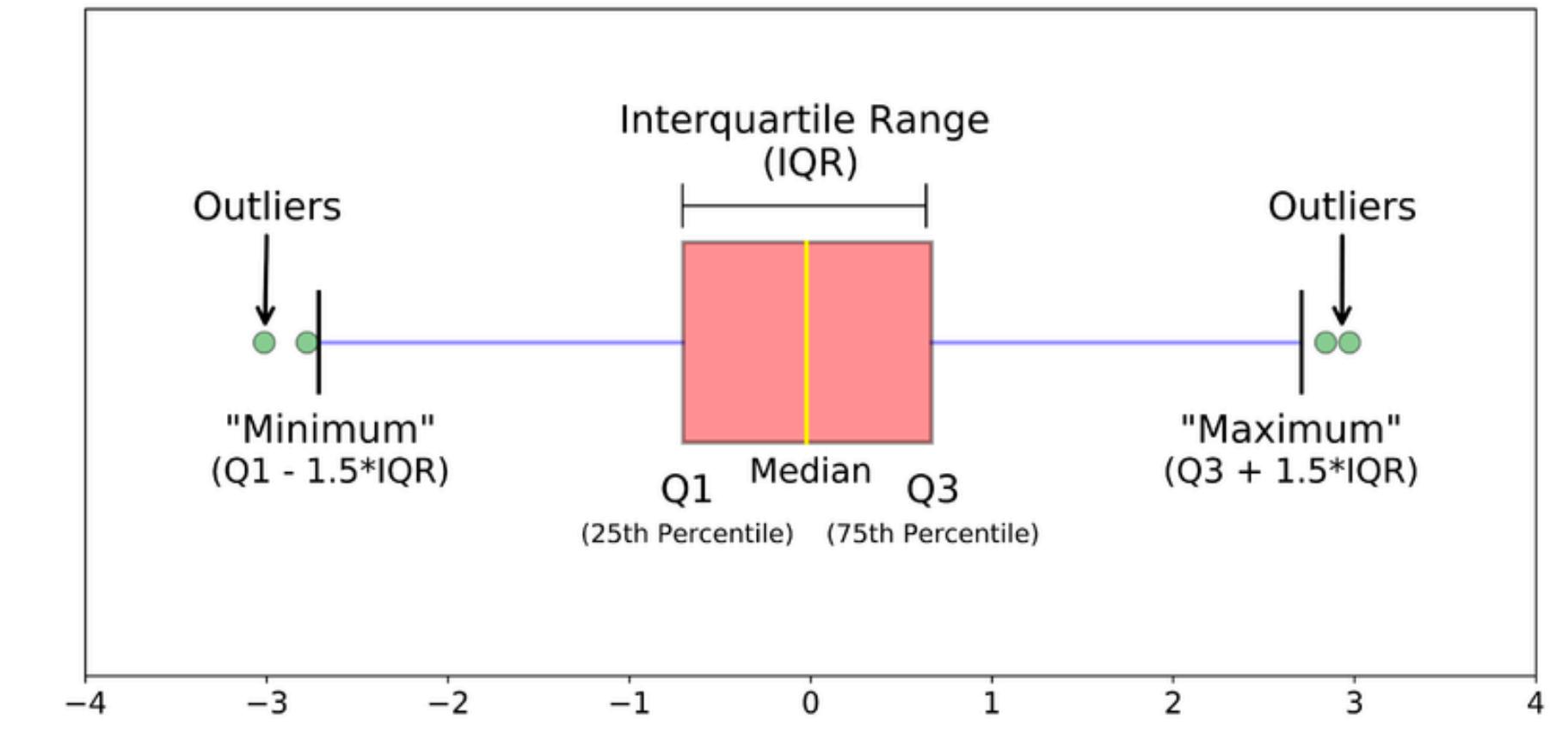
$$\frac{\{x - \mu\}}{\sigma}$$

- 이상치는 확률이 낮은 영역에서 발생한 값으로 볼 수 있음.



# 이상값 처리 - box plot

- Box plot을 통한 이상치 구분
- Empirical distribution을 통해 Q1, Q3를 계산
- Q1 : 전체 데이터의 25%를 포함
- Q3 : 전체 데이터의 75%를 포함
- IQR(Inter quartile range) :  $Q3 - Q1$
- $Q1 - 1.5 \text{ IQR}$  미만, 또는  $Q3 + 1.5 \text{ IQR}$  초과 분을 이상치로 간주



**Box plot**

# 이상값 처리

- Scikit learn 라이브러리 활용시:
  - 별도로 이상치들을 탐색하여 NaN값 처리
  - 이후 imputer 메서드를 활용해 같은 처리 가능
- Feature engine 라이브러리 활용시:
  - Z-score, box plot 기준으로 이상치 기준 설정하고 삭제 혹은 대체 (capping)

# Outlier imputation

```
# Winsorizer by gaussian
from feature_engine.outliers import Winsorizer

winsorizer = Winsorizer(
    variables = ['Age'],
    capping_method = 'gaussian', #gaussian, iqr, mad, quantiles
    tail = 'both', # 'right', 'left', 'both'
    fold = 1, # std scale
    missing_values = 'ignore'
)

df_out = winsorizer.fit_transform(df)

print('Age range before capping: ', df.Age.min(), '~', df.Age.max())
print('Age range after capping: ', round(df_out.Age.min(),1), '~', round(df_out.Age.max(),1))
print(df.shape, df_out.shape) # winsorizer 는 삭제가 아닌 제한!
```

# Outlier imputation

```
# Winsorizer by quantiles

winsorizer = Winsorizer(
    variables = ['Age'],
    capping_method = 'quantiles', # quantile option의 경우
    tail = 'both', # 'right', 'left', 'both'
    fold = 0.05, # std scale, 0.05 = 0.05 ~ 0.95사이의 범위 값으로
    missing_values = 'ignore'
)
|
df_out = winsorizer.fit_transform(df)
print('Age range before capping: ', df.Age.min(), '~', df.Age.max())
print('Age range after capping: ', df_out.Age.min(), '~', df_out.Age.max())
```

# Outlier imputation

```
# Outlier Trimmer by gaussian
from feature_engine.outliers import OutlierTrimmer

trimmer = OutlierTrimmer(
    variables = ['Age'],
    capping_method = 'gaussian',
    tail = 'both', # 'right', 'left', 'both'
    fold = 1, # std scale
    missing_values = 'ignore'
)

df_out = trimmer.fit_transform(df)
print(df.shape, df_out.shape) # trimmer 는 삭제!
```

# 인코딩

- 정형 데이터의 대부분은 수치형 또는 범주형 데이터
- 인코딩: 범주형 데이터를 모델이 인식할 수 있도록 수치형 변수로 변환
- ['CAT', 'DOG', 'BIRD']
- One hot encoding : 클래스 수 만큼의 차원을 갖는 벡터 생성, 특정 클래스에 1 할당
  - CAT = [1, 0, 0], DOG = [0, 1, 0], BIRD = [0, 1, 1]
- Label / Ordinal encoding : 정수로 변환
  - CAT = 0, DOG = 1, BIRD = 2

# 인코딩

- 데이터프레임 객체 이름이 df인 경우
- df.dtypes를 통해 변수별 유형 파악 가능

```
df_sample = df[['Sex', 'Embarked', 'Pclass']].copy()  
print(df_sample.head())  
|
```

# Label / ordinal encoder

```
from sklearn.preprocessing import LabelEncoder  
  
le = LabelEncoder()  
  
encoding = le.fit_transform(df_sample['Sex'])
```

```
from sklearn.preprocessing import OrdinalEncoder  
  
oe = OrdinalEncoder()  
  
encoding = oe.fit_transform(df_sample)
```

# One hot encoder

```
from sklearn.preprocessing import OneHotEncoder  
  
ohe = OneHotEncoder()  
  
encoding = ohe.fit_transform(df_sample)  
  
print(encoding)
```

```
encoding_dense = encoding.toarray() # sparse matrix → dense array  
df_encoded = pd.DataFrame(encoding_dense, columns=ohe.get_feature_names_out())  
print(df_encoded.head())
```

# 중복값 처리

- 누락값, 이상값외에 중복데이터가 존재할 수 있음
- 중복 샘플에 대한 삭제 필요

```
# 예제 데이터프레임 생성
data = {
    '이름': ['철수', '영희', '민수', '철수', '민수', '지영'],
    '나이': [22, 25, 24, 22, 24, 30],
    '도시': ['서울', '부산', '대구', '서울', '대구', '인천']
}
df_dup = pd.DataFrame(data)
df_dup.duplicated()
df_unique = df.drop_duplicates()
```

# 정규화 / 스케일링

- 수치형 데이터의 경우 데이터의 스케일이 서로 다를 수 있음.
- 거리 기반의 방법론의 경우 변수의 스케일이 결과에 큰 영향을 미침
- 스케일링을 통한 변수간 차이 보정

	평형	가격	우편번호
A	34	1억	15141
B	27	2억	15274
C	24	3억	38720
D	39	4억	13920

# 다양한 스케일링 기법들

- Standard Scaler: 평균 0, 표준편차 1로 변환,  $z = \frac{x - \mu}{\sigma}$
- Minmax scaler: 최대 1, 최소 0으로 변환,  $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$
- Robust scaler: 중앙값, IQR 기준으로 변환,  $x' = \frac{x - x_{med}}{IQR}$
- 그외 : MaxAbs scaler, power transformer, quantile transformer

# Sklearn scaler 활용

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.preprocessing import PowerTransformer, QuantileTransformer

age = df[['Age']].dropna().copy()

# 1. StandardScaler (평균 0, 표준편차 1)
standard_scaled = StandardScaler().fit_transform(age)

# 2. MinMaxScaler (0 ~ 1)
minmax_scaled = MinMaxScaler().fit_transform(age)

# 3. RobustScaler (중앙값 기준, 이상치에 덜 민감)
robust_scaled = RobustScaler().fit_transform(age)

# 4. PowerTransformer (Yeo-Johnson: 정규분포에 가깝게)
power_scaled = PowerTransformer(method='yeo-johnson').fit_transform(age)

# 5. QuantileTransformer (정규분포 기반 분위수 정규화)
quantile_scaled = QuantileTransformer(output_distribution='normal',n_quantiles=100).fit_transform(age)
```

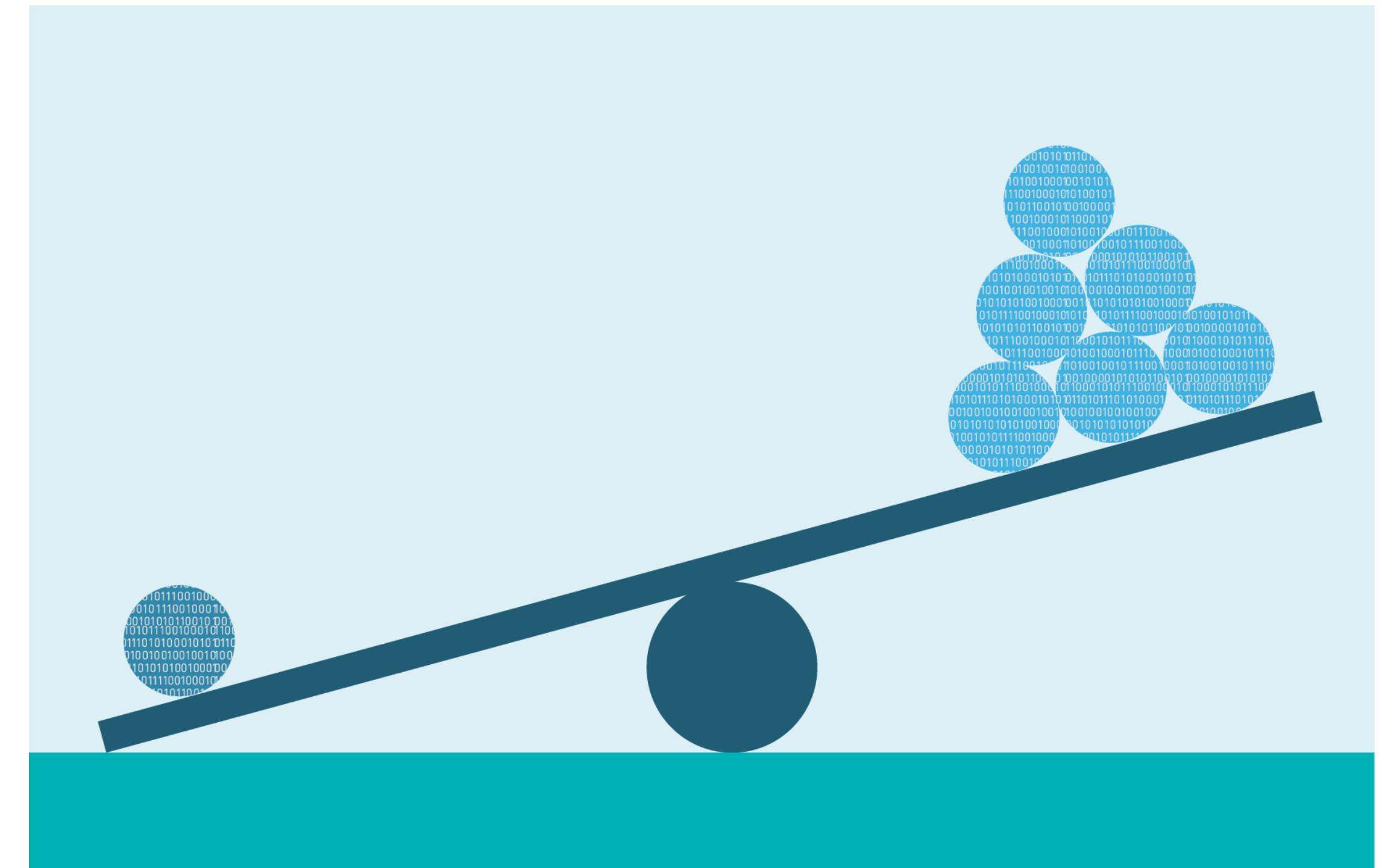
# Sklearn scaler 활용

```
result_df = pd.DataFrame({
    'Original': age.values.flatten(),
    'Standard': standard_scaled.flatten(),
    'MinMax': minmax_scaled.flatten(),
    'Robust': robust_scaled.flatten(),
    'Power': power_scaled.flatten(),
    'Quantile': quantile_scaled.flatten()
})

print(result_df.head())
print(result_df.Original.hist(bins=20))
```

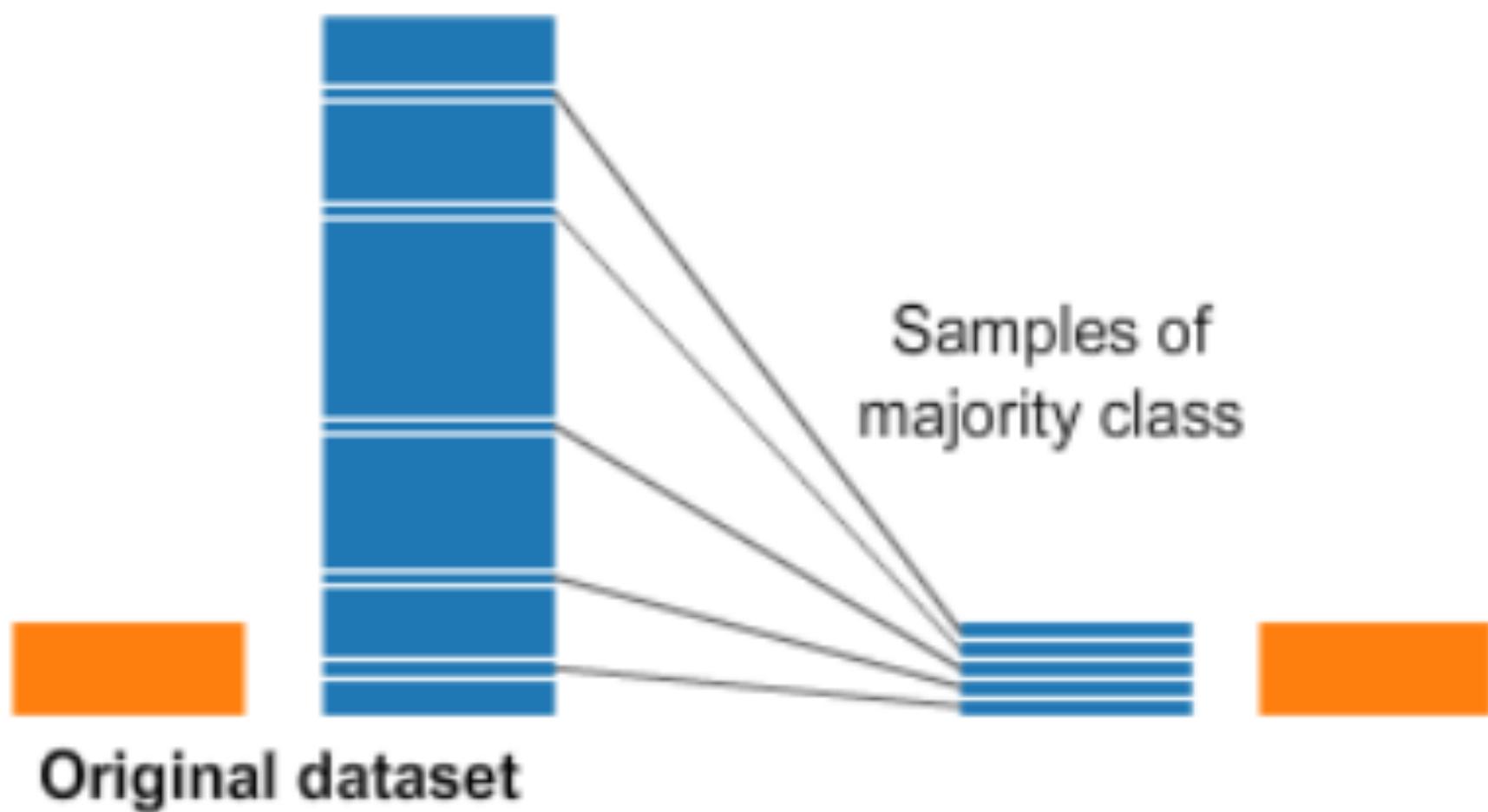
# 데이터 불균형

- 데이터 불균형이란 수집된 샘플내에서 특정 클래스의 비중이 높은 상태를 의미
- 소수의 양성 VS 다수의 음성
- 모델 입장에서는? 양성도 음성으로 예측
- 데이터 불균형이 모델 성능에 부정적 영향을 미침



# 불균형 해소를 위한 두가지 전략

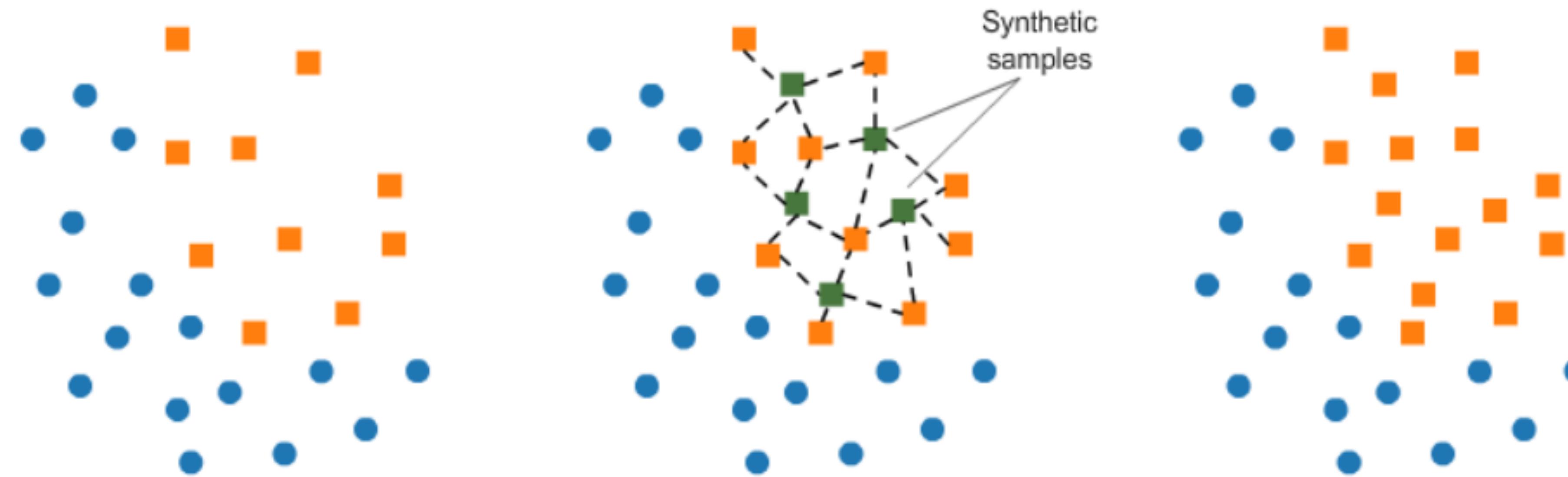
Undersampling



Oversampling



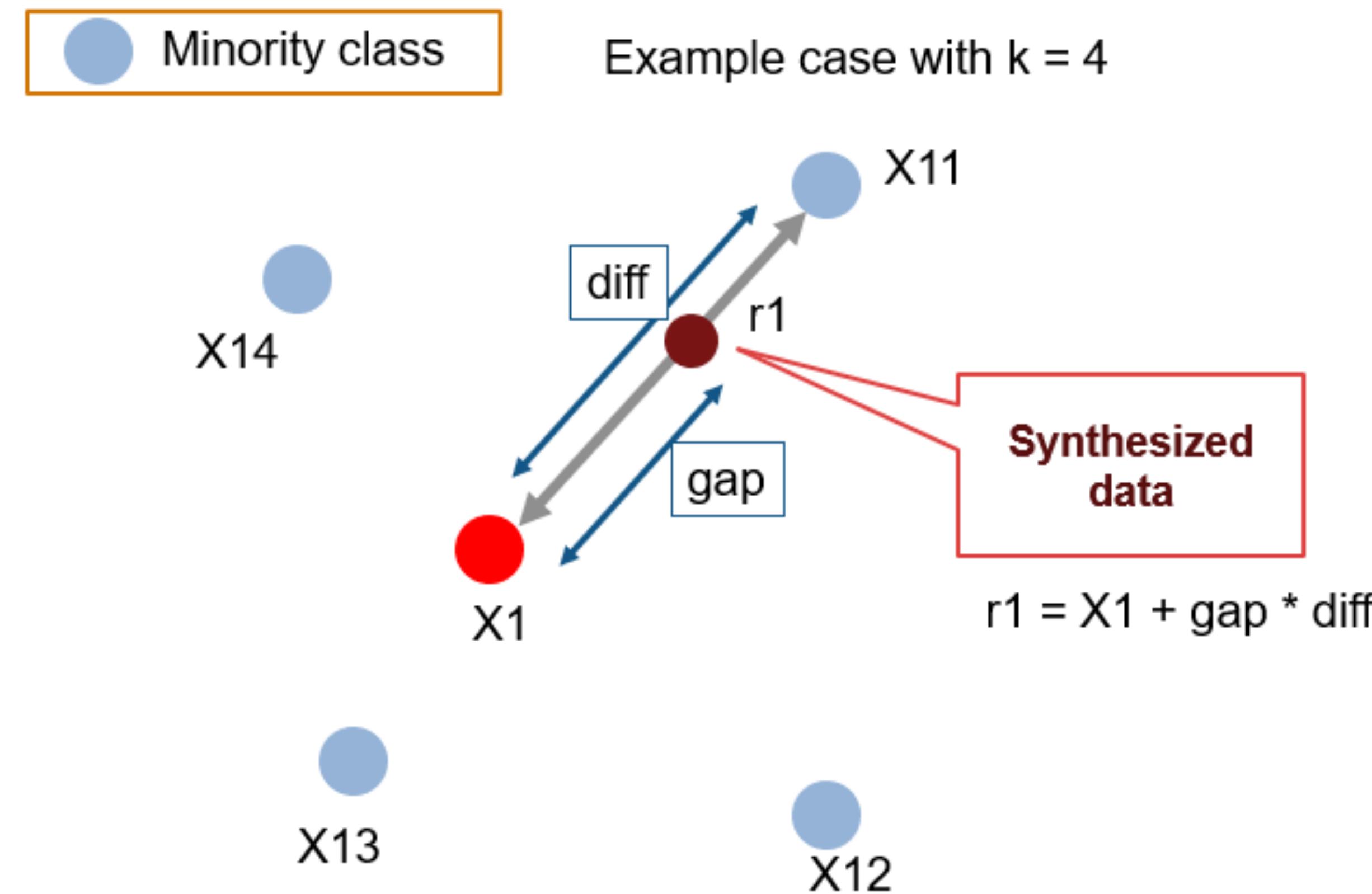
# Over sampling 기법



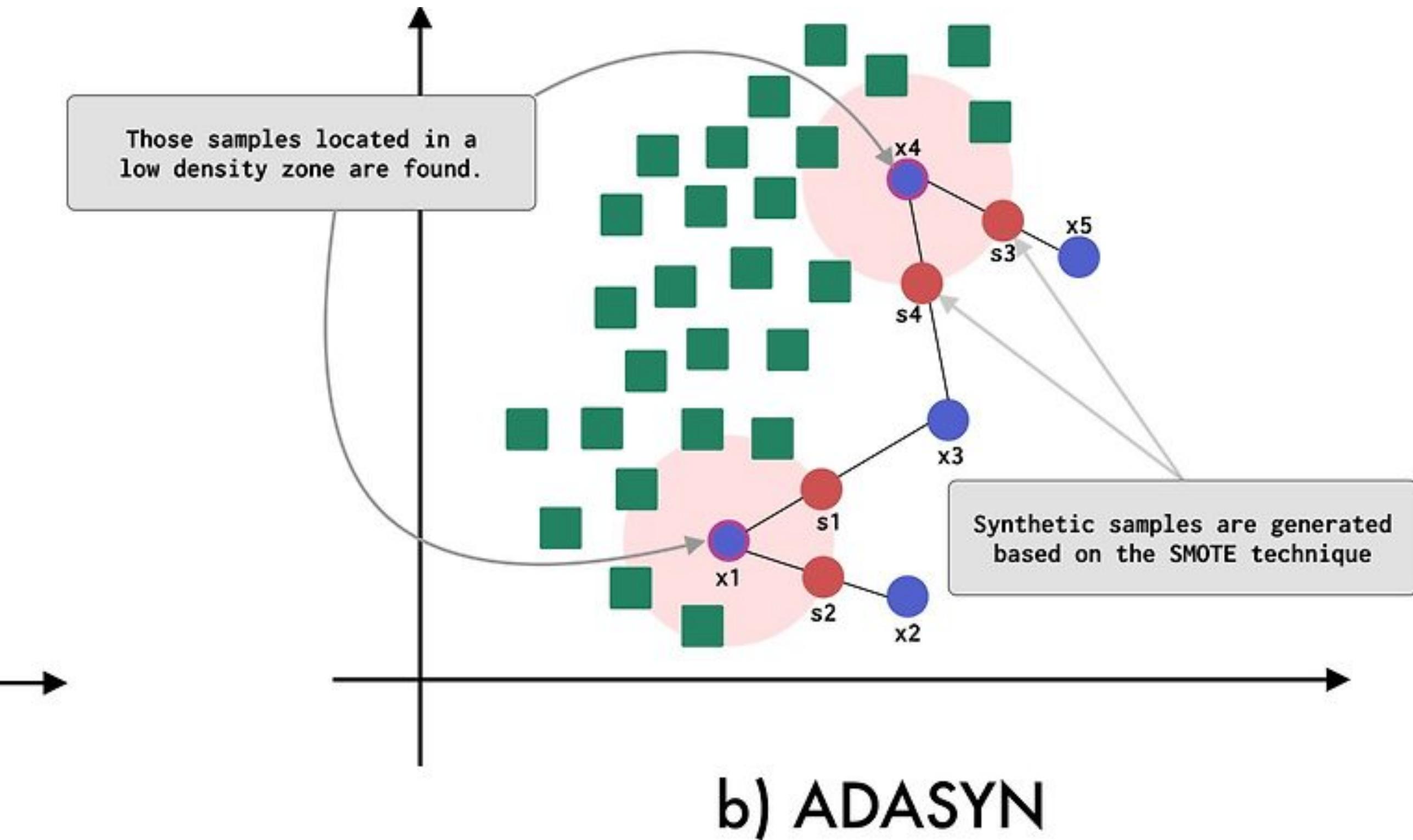
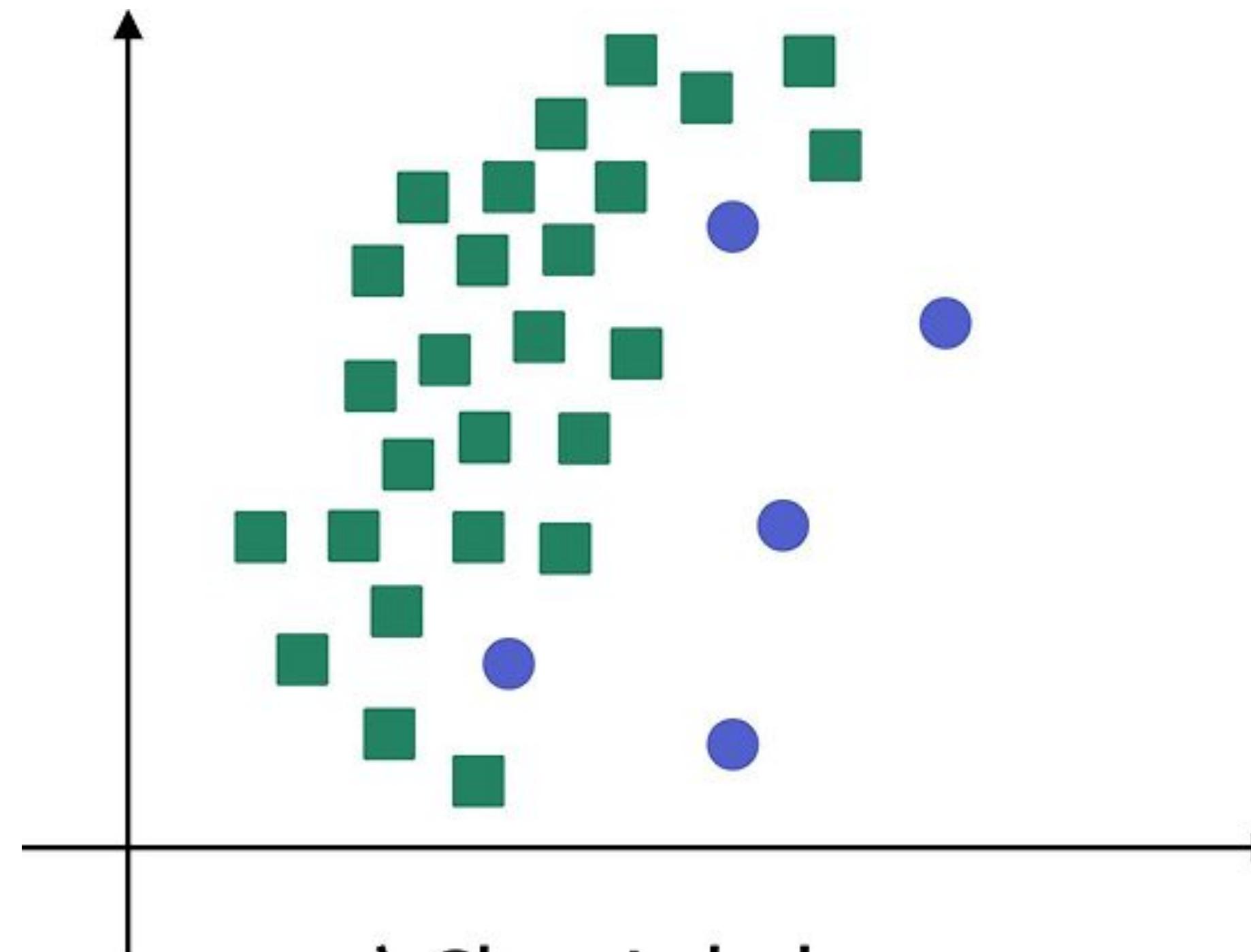
# Over sampling 기법

- Random oversampling : 무작위로 샘플들을 선택하여 복제
- SMOTE (Synthetic Minority Over-sampling Technique) : 최근접이웃에 대한 선형보간
- ADASYN(adaptive synthetic sampling approach for imbalanced learning) : 소수 클래스 주변의 다수 클래스를 고려하여 선형 보간

# Over sampling - SMOTE



# Over sampling - ADASYN



# Imbalanced-learn

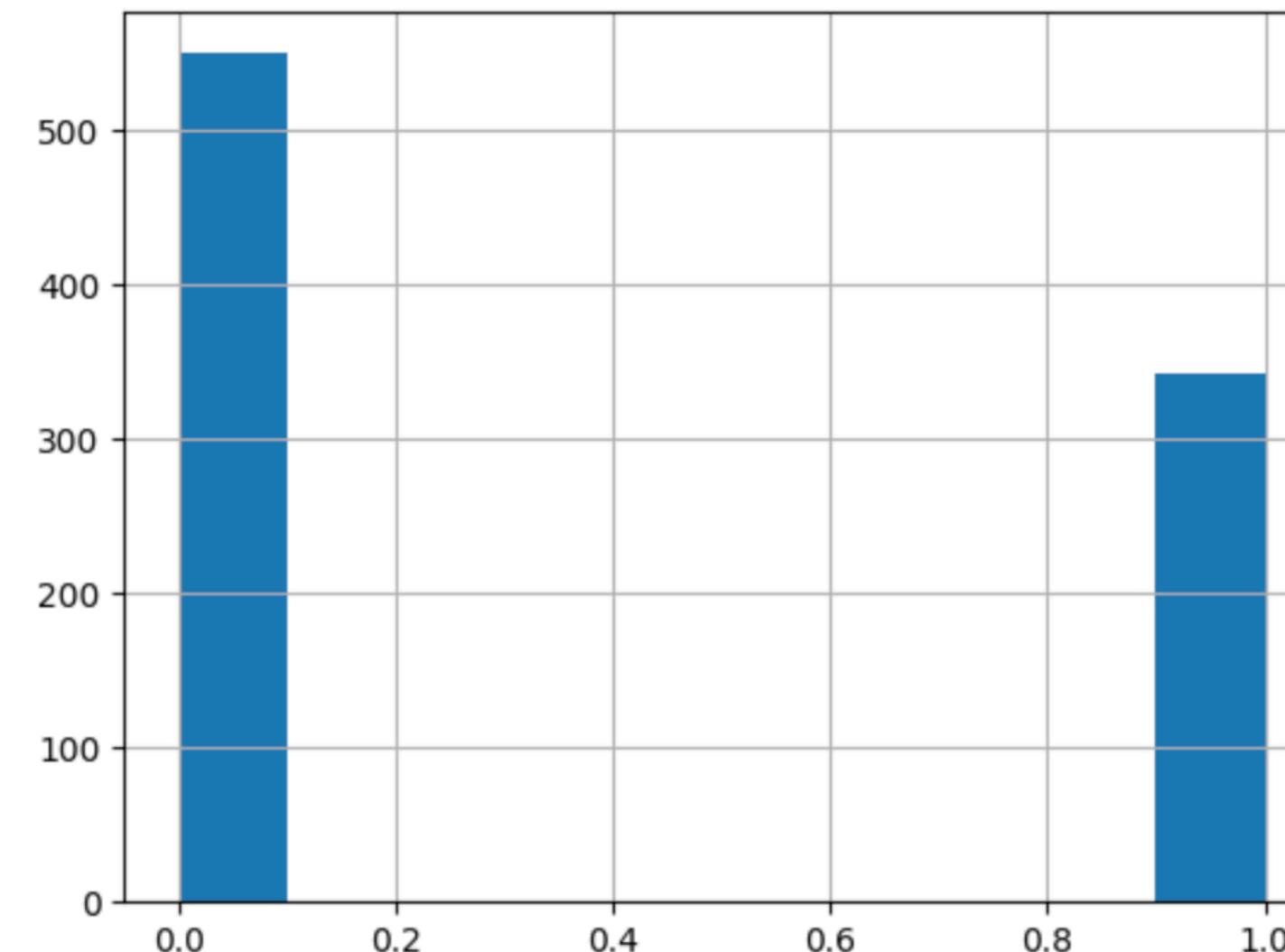
- Imbalanced-learn 라이브러리를 활용
- 언더샘플링/오버샘플링 처리 가능
- !pip install imbalanced-learn

# Imbalanced-learn

- Imbalanced-learn 라이브러리를 활용
- 언더샘플링/오버샘플링 처리 가능
- !pip install imbalanced-learn

```
df_sample = df[['Fare', 'SibSp', 'Survived']]  
df_sample.Survived.hist()
```

<Axes: >



# Random oversampling

```
from imblearn.over_sampling import RandomOverSampler  
  
sampler = RandomOverSampler(random_state=0)  
sampled_df, sampled_lable = sampler.fit_resample(df_sample, df_sample.Survived)  
  
print(sampled_df.groupby('Survived').count())
```

# SMOTE

```
from imblearn.over_sampling import SMOTE  
  
sampler = SMOTE(random_state=0, k_neighbors = 3)  
sampled_df, sampled_lable = sampler.fit_resample(df_sample, df_sample.Survived)
```

```
print(sampled_df.groupby('Survived').count())
```

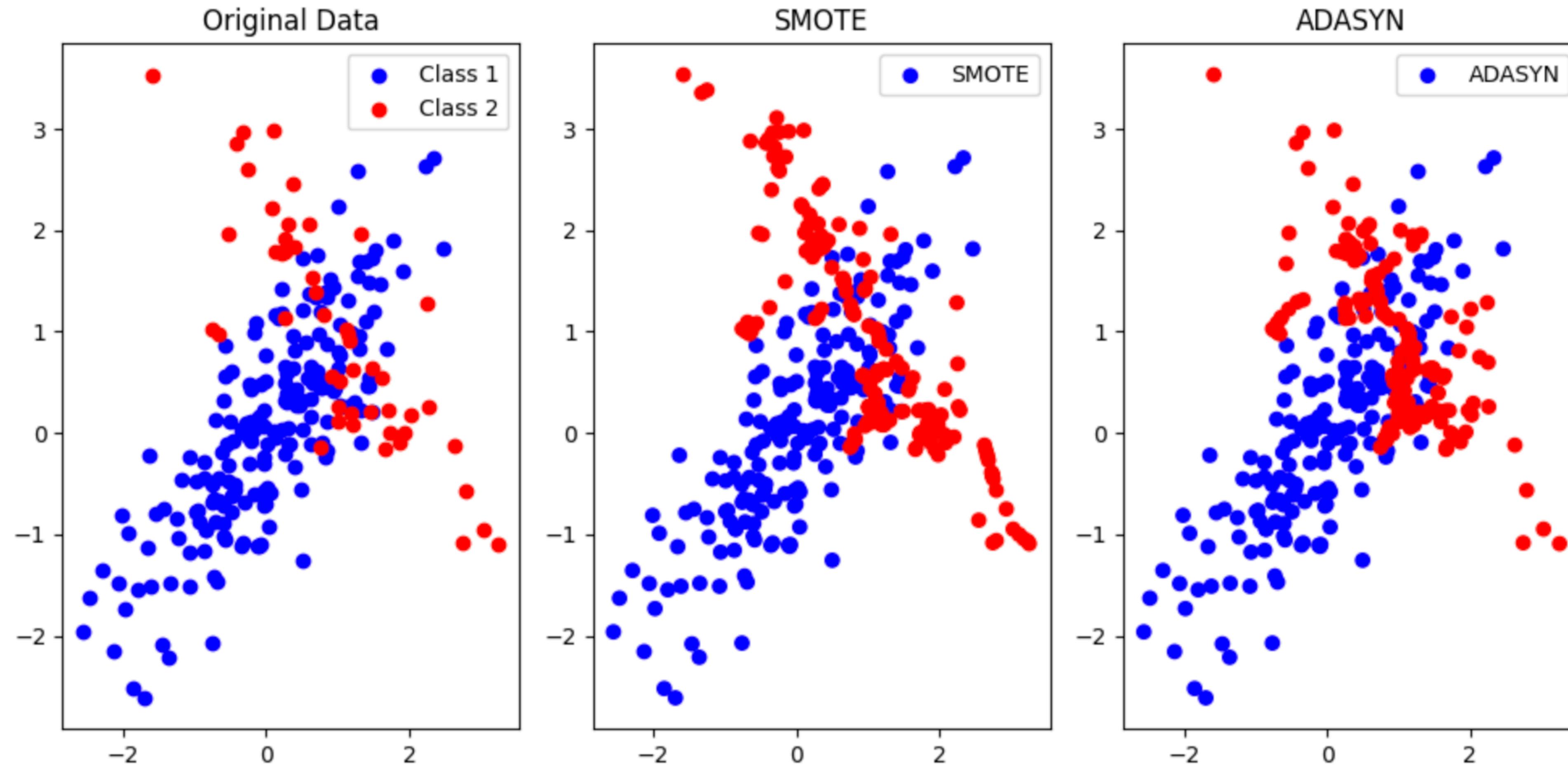
	Fare	SibSp
Survived		
0	549	549
1	549	549

# ADASYN

```
from imblearn.over_sampling import ADASYN

sampler = ADASYN(random_state=0, n_neighbors = 5)
sampled_df, sampled_lable = sampler.fit_resample(df_sample, df_sample.Survived)
print(sampled_df.groupby('Survived').count())
```

# SMOTE vs ADASYN



# Data augmentation

- 데이터 증강을 통해 학습에 사용가능한 데이터를 늘릴 수 있음.
- SMOTE/ADASYN 등
- Gaussian noise injection
- 생성형 모델
- Mixup

# 특성 공학 (Feature engineering)

- 다항 변환 (Polynomial feature) : 기존 변수들의 다항식 조합 활용, (e.g.,  $x^2$ ,  $xy$ ,  $y^3$ ).
- 그룹 특성 (Group feature) : 연관된 변수 그룹이 존재 시 통계값 활용, (e.g.,  $t_1, t_2, t_3, \dots, t_k \rightarrow \mu_t, \sigma_t$ )
- 구간화 (Binning) : 연속형 변수를 구간을 나누어 범주형 변수로 변환
  - (e.g., 0~200까지 속도값, → stop, slow, fast, very fast 클래스로 구분)
- 변수 분할 (Feature split) : 복합적으로 구성된 변수를 분할하여 활용
  - (e.g., 20220301 → (연도) 2022, (월) 3, (일) 1.
- 푸리에 및 웨이브렛 변환 (시계열 데이터), 변수 추가 (e.g., 날짜데이터 → 공휴일 변수), PCA 등

# Feature-engine

- Feature engine을 활용한 특성 공학 적용
- 값 대체 외에 다양한 변환 적용 가능
- Scikit-learn의 경우 numpy array 형태로 데이터 변화
- Feature engine은 데이터프레임에 대한 직접 가공 가능

# 기본 인코딩 - OneHot

```
from feature_engine.encoding import OneHotEncoder

encoder = OneHotEncoder(
    variables=['Sex', 'Pclass'],
    #ignore_format=False,
    ignore_format=True, # object, categorical 이외의 데이터 적용시
)

trans_df = encoder.fit_transform(df)
print(trans_df.head(3))
```

# 기본 인코딩 - Ordinal

```
from feature_engine.encoding import OrdinalEncoder

# One hot vector가 아닌 정수로 인코딩
encoder = OrdinalEncoder(
    encoding_method='arbitrary',
    variables=['Sex', 'Ticket'],
)

trans_df = encoder.fit_transform(df)
print(trans_df.head(3))
```

# 이산화 / Binning

- 연속형, 수치형 변수들을 정수 클래스로 이산화
- EqualFrequencyDiscretiser :
  - 주어진 bin에 따라서 같은 비율로 맞추도록 정수 클래스로 변환.
- EqualWidthDiscretiser :
  - 등간격으로 bin 생성 후 정수 클래스로 변환
- ArbitraryDiscretiser :
  - 주어진 임의의 범위 순으로 정수 클래스로 변환

# 이산화 / Binning - equal frequency

```
from feature_engine.discretisation import EqualFrequencyDiscretiser  
  
discretizer = EqualFrequencyDiscretiser(  
    variables=['Fare'],  
    q=10  
)  
trans_df = discretizer.fit_transform(df)  
print(trans_df.head(3))
```

- 히스토그램을 비교하면?
- df.Fare.hist() vs trans\_df.Fare.hist()

# 이산화 / Binning - equal width

```
from feature_engine.discretisation import EqualWidthDiscretiser
# 등간격 구간 생성
discretizer = EqualWidthDiscretiser(
    variables=['Fare'],
    bins=10
)

trans_df = discretizer.fit_transform(df)
print(trans_df.head(3))
```

- 히스토그램을 비교하면?
- df.Fare.hist() vs trans\_df.Fare.hist()

# 이산화 / Binning - arbitrary

```
from feature_engine.discretisation import ArbitraryDiscretiser

discretizer = ArbitraryDiscretiser(
    ## 기존 variable의 binning_dict안으로 들어감
    binning_dict = {'Fare' : [0,10,100,1000]}, 
    )

trans_df = discretizer.fit_transform(df)
print(trans_df.head(3))
```

- 히스토그램을 비교하면?
- df.Fare.hist() vs trans\_df.Fare.hist()

# Feature Creation

- 기존의 변수들의 수치값을 바탕으로 새로운 feature 생성
- MathFeatures : numerical variables에 대해 주어진 수학적 연산 수행 (커스텀 함수 적용 가능)
- RelativeFeatures : 주어진 변수와 레퍼런스 변수 사이에서의 연산 사칙연산 등 수행 (사용가능한 함수가 정해져있음, 레퍼런스가 존재)
- CyclicalFeatures :  $\sin/\cos(\text{variable} * (2. * \pi / \text{max\_value}))$  변환 수행
- (Scikit-learn) PolynomialFeatures : numerical 변수들에 대한 polynomial 조합 생성

# Feature Creation - MathFeatures

```
from feature_engine.creation import MathFeatures

feature_creator = MathFeatures(
    variables = ['SibSp', 'Parch'],
    func = "sum", # sum, prod, mean, max, min, etc.
    missing_values = 'ignore'
)

new_df = feature_creator.fit_transform(df)
print(new_df.columns)
print(new_df.head(3))
```

# Feature Creation - Relative Feature

```
from feature_engine.creation import RelativeFeatures

feature_creator = RelativeFeatures(
    variables = ['SibSp'],
    reference = ['Parch', 'Age'],
    func = ["add", 'sub'], # add, sub, mul, div, truediv, etc.
    missing_values = 'ignore'
)

new_df = feature_creator.fit_transform(df)
print(new_df.columns)
print(new_df.head(3))
```

# Feature Creation - Relative Feature

```
# Numerical feature만 조합가능.  
# !!scikit-learn 라이브러리!! 활용  
from sklearn.preprocessing import PolynomialFeatures  
  
feature_creator = PolynomialFeatures()  
    degree = 3,  
    interaction_only = False, # False인 경우와 True인 경우의 차이는?  
)  
  
new_df = feature_creator.fit_transform(df[['Fare', 'SibSp', 'Parch']])  
print(feature_creator.get_feature_names_out(['Fare', 'SibSp', 'Parch']))  
print(new_df[:3,:])
```