



4주차 발표

DA팀 박보영 손소현 오수진

목차

#01 캐글 산탄데르 고객 만족 예측

#02 캐글 신용카드 사기 검출

#03 캐글 심장병 발병 예측



1. 캐글 산탄데르 고객 만족 예측



1.1 대회 소개

- 2016년에 열린 캐글 대회
- 은행에서 초기에 불만족 고객을 예측하고 사전에 개선하기 위해 캐글에 요청한 대회
- 산탄데르 은행의 고객 만족 유무를 예측하는 것이 목적
- 평가는 만족하지 못한 고객일 확률을 예측하고, ROC curve 아래부분, 즉 AUC score으로 평가


Featured Prediction Competition

Santander Customer Satisfaction

Which customers are happy customers?

\$60,000

Prize Money

 Banco Santander · 5,115 teams · 6 years ago

Overview

Data

Code

Discussion

Leaderboard

Rules

Team

My Submissions

Late Submission

...

Overview

Description

Evaluation



Prizes

Timeline

From frontline support teams to C-suites, customer satisfaction is a key measure of success. Unhappy customers don't stick around. What's more, unhappy customers rarely voice their dissatisfaction before leaving.

Santander Bank is asking Kagglers to help them identify dissatisfied customers early in their relationship. Doing so would allow Santander to take proactive steps to improve a customer's happiness before it's too late.

In this competition, you'll work with hundreds of anonymized features to predict if a customer is satisfied or dissatisfied with their banking experience.



1.2 Data Description

- train, test 2개의 csv파일이 제공됨
- train으로 훈련하고 test로 최종평가
- train 데이터를 살펴보면, 76020개의 행과 371개의 칼럼으로 구성됨
- 맨 앞의 ID, 맨 뒤의 TARGET(불만족 : 1, 만족 : 0)을 제외한 나머지 369개의 피쳐들을 이용하여 예측 알고리즘을 수행
- TARGET의 0.04만이 1(불만족)
- 이 대회에서 특이한 점은 369개의 각 피쳐들의 피쳐 이름이 모두 익명 처리되어있어, 각 피쳐가 어떤 의미를 가지고 있는지 모른다는 점!

1.3 핵심 아이디어/모델링

- 오버샘플링 전 고객 만족 여부를 lightGBM , XGBoost로 예측
- Imbalance data 해결 (ROSE 오버샘플링)
- 오버샘플링 후 고객 만족 여부를 lightGBM , XGBoost로 예측

1.3 핵심 아이디어/모델링

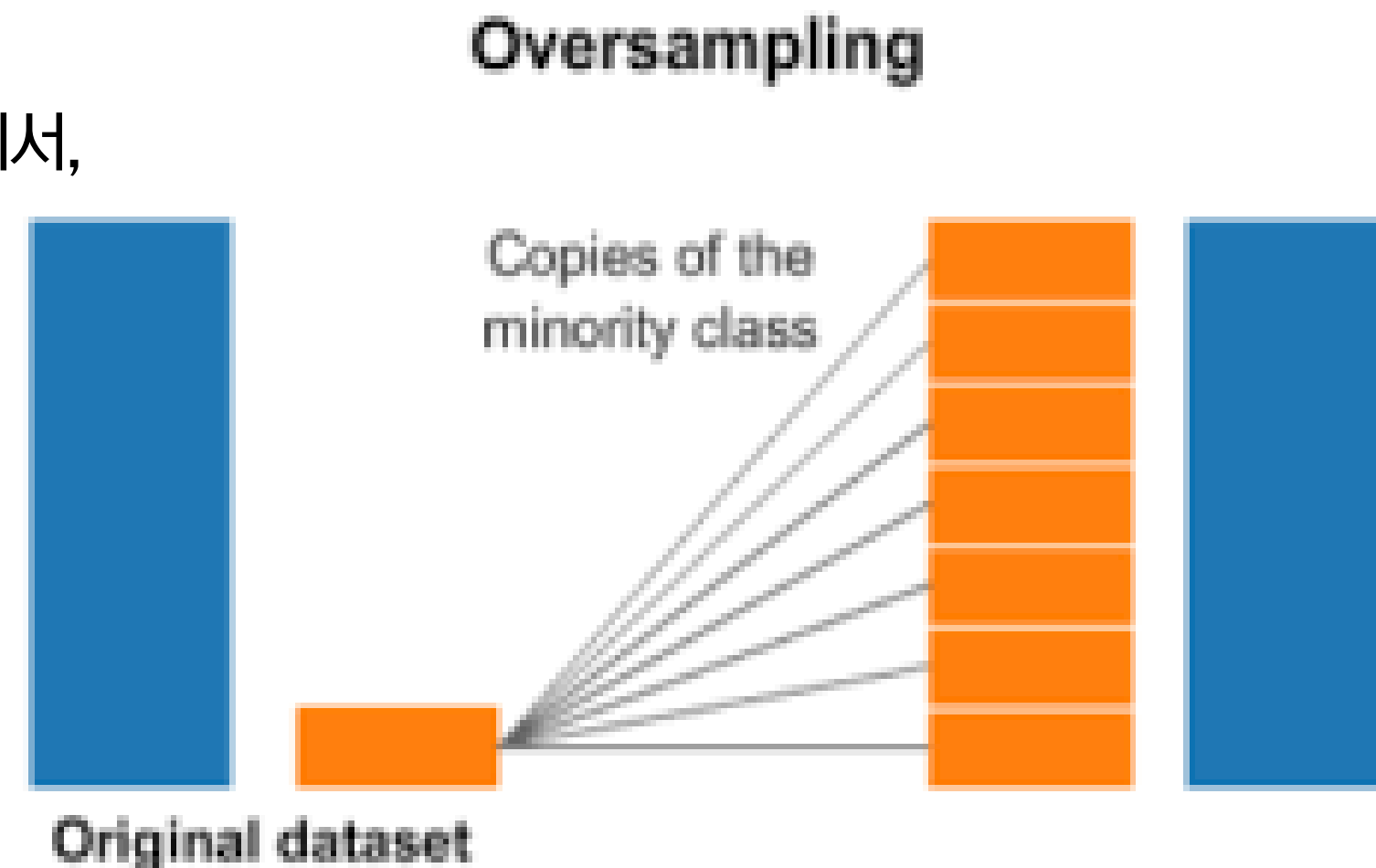
- Imbalance data 해결

Q. 불균형 데이터에서 왜 오버샘플링 필요한가?

A. 불균형 데이터를 사용하면 모델이 major 클래스를 예측하는 편향을 만들

참고) 3장 평가방법에 대해 배우면서 사용한 예시 :

100개의 데이터중에 90개가 0이고, 10개가 1인 상황에서,
모두 0으로 예측한다면 모델의 정확도는 90%가 된다!



1.3 핵심 아이디어/모델링

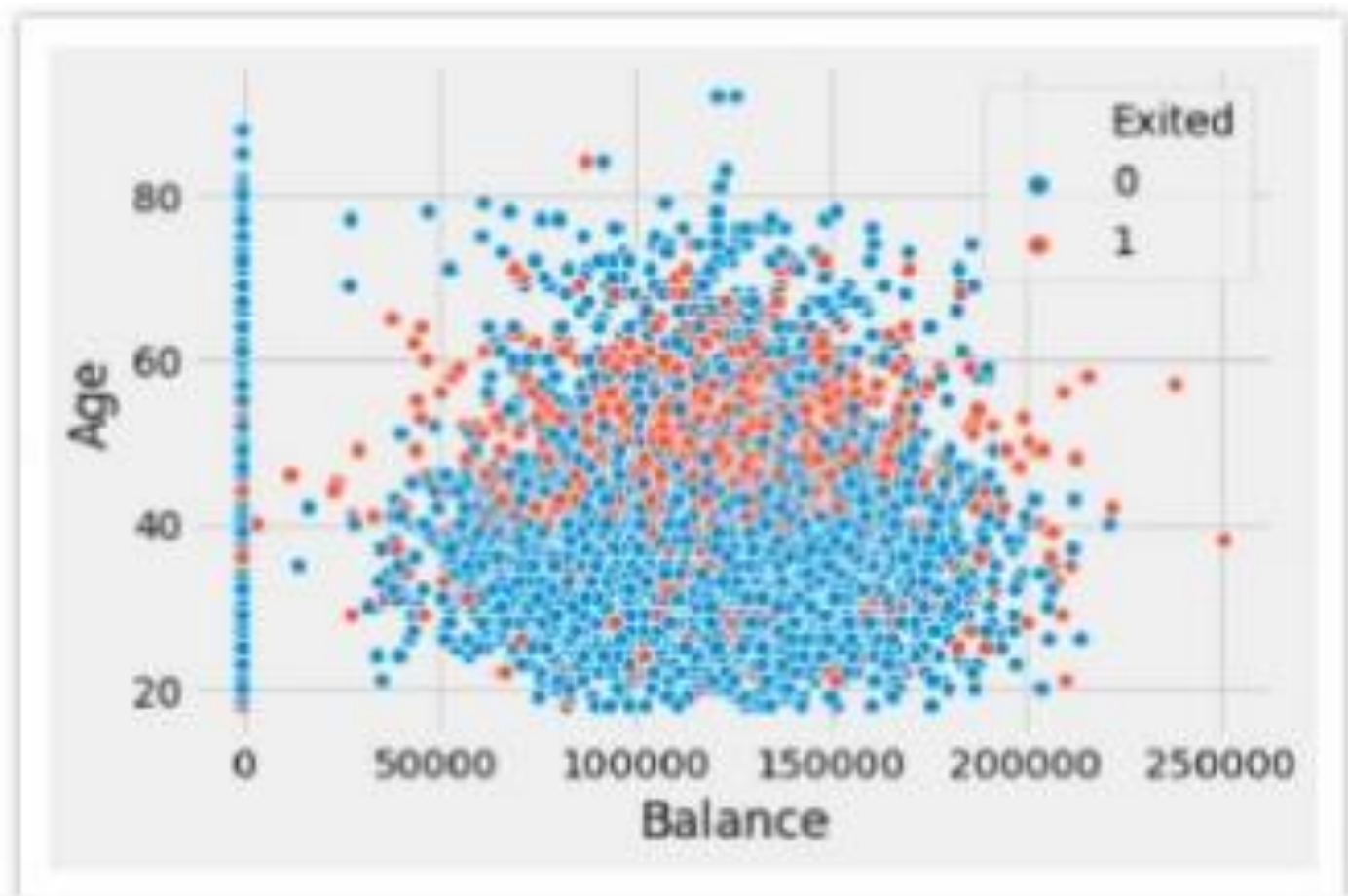
- Imbalance data 해결

1. Random Over Sampling (ROS or ROSE)

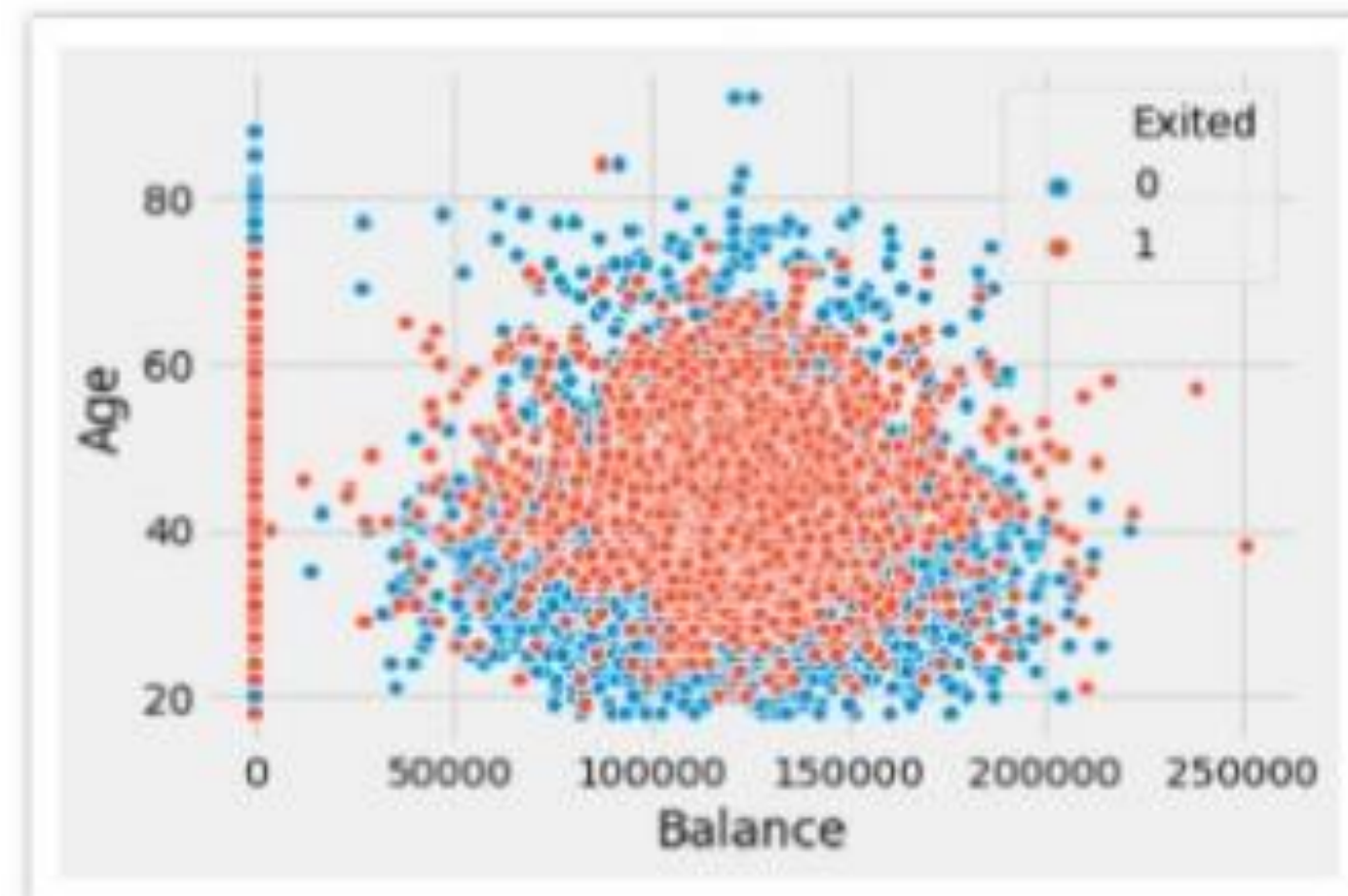
랜덤 오버샘플링은 데이터 세트의 불균형 특성의 균형을 맞추기 위한 가장 간단한 오버샘플링 기술

소수 클래스 샘플을 복제하여 데이터 균형을 유지

이로 인해 정보가 손실되지는 않지만 동일한 정보가 복사되기 때문에 데이터가 과적합되기 쉬움



▲원본데이터



▲샘플링 후 데이터

1.3 핵심 아이디어/모델링

- Imbalance data 해결

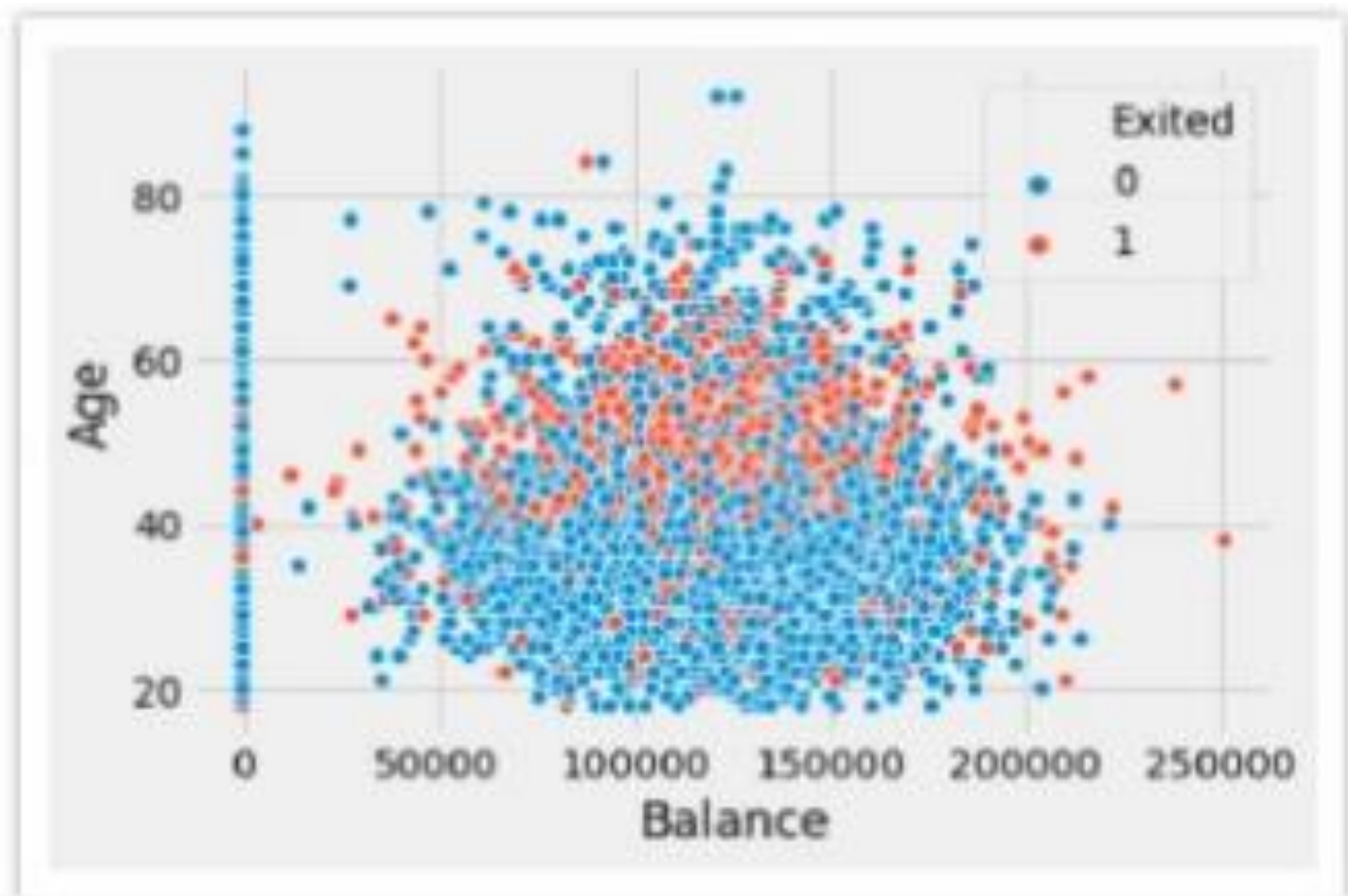
2. SMOTE

Synthetic Minority Oversampling Technique의 약자

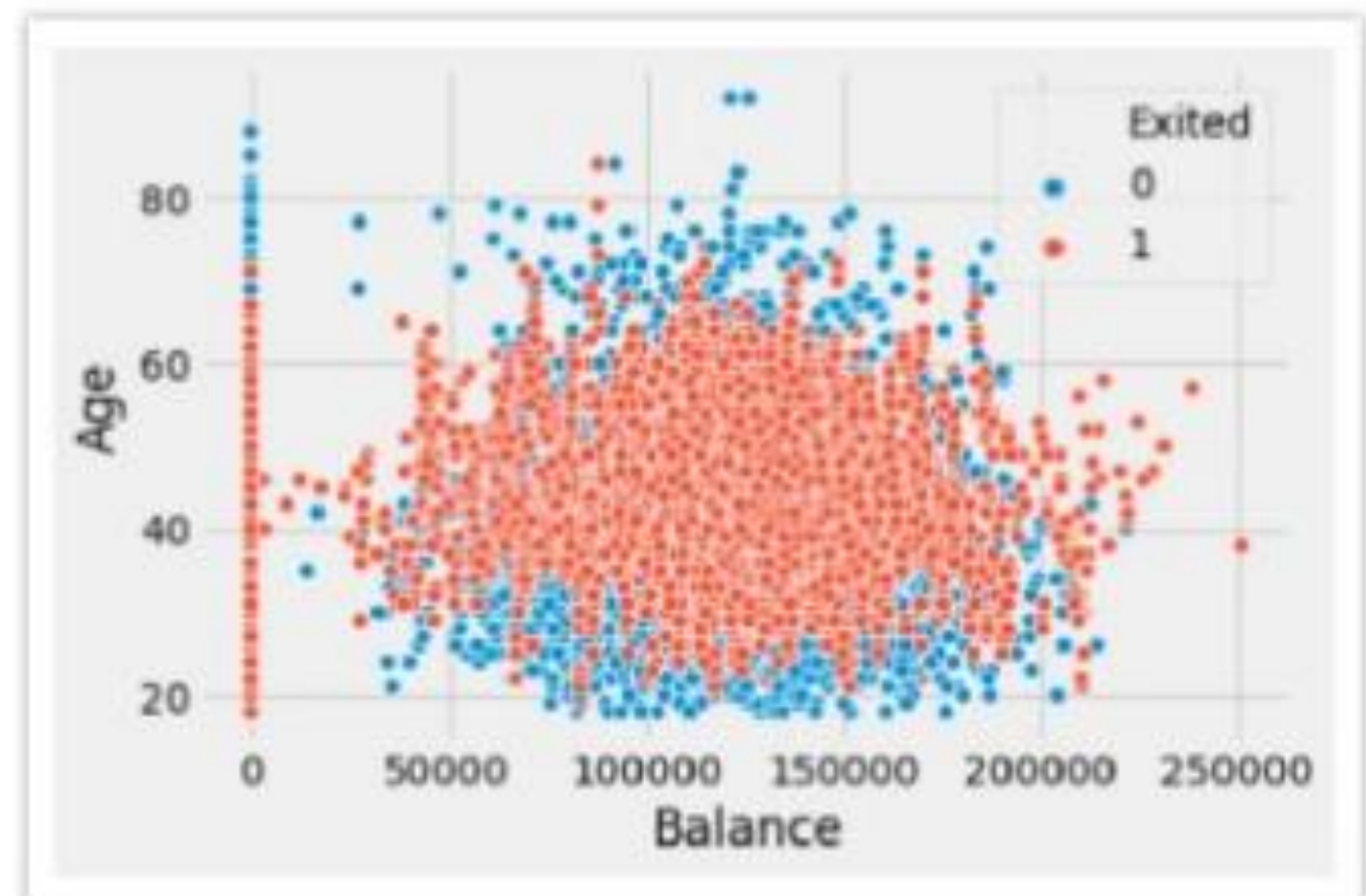
랜덤 오버샘플링의 경우 소수 클래스 샘플이 복제되기 때문에 과적합되는 경향이 있음.

SMOTE는 데이터 세트의 균형을 맞추기 위해 새로운 합성 샘플을 생성. SMOTE는 **k-최근접 이웃** 알고리즘을 활용하여 합성 데이터를 생성.

뒤에 보영님이 SMOTE 오버샘플링 예시를 보여주실 예정입니다.



▲원본데이터



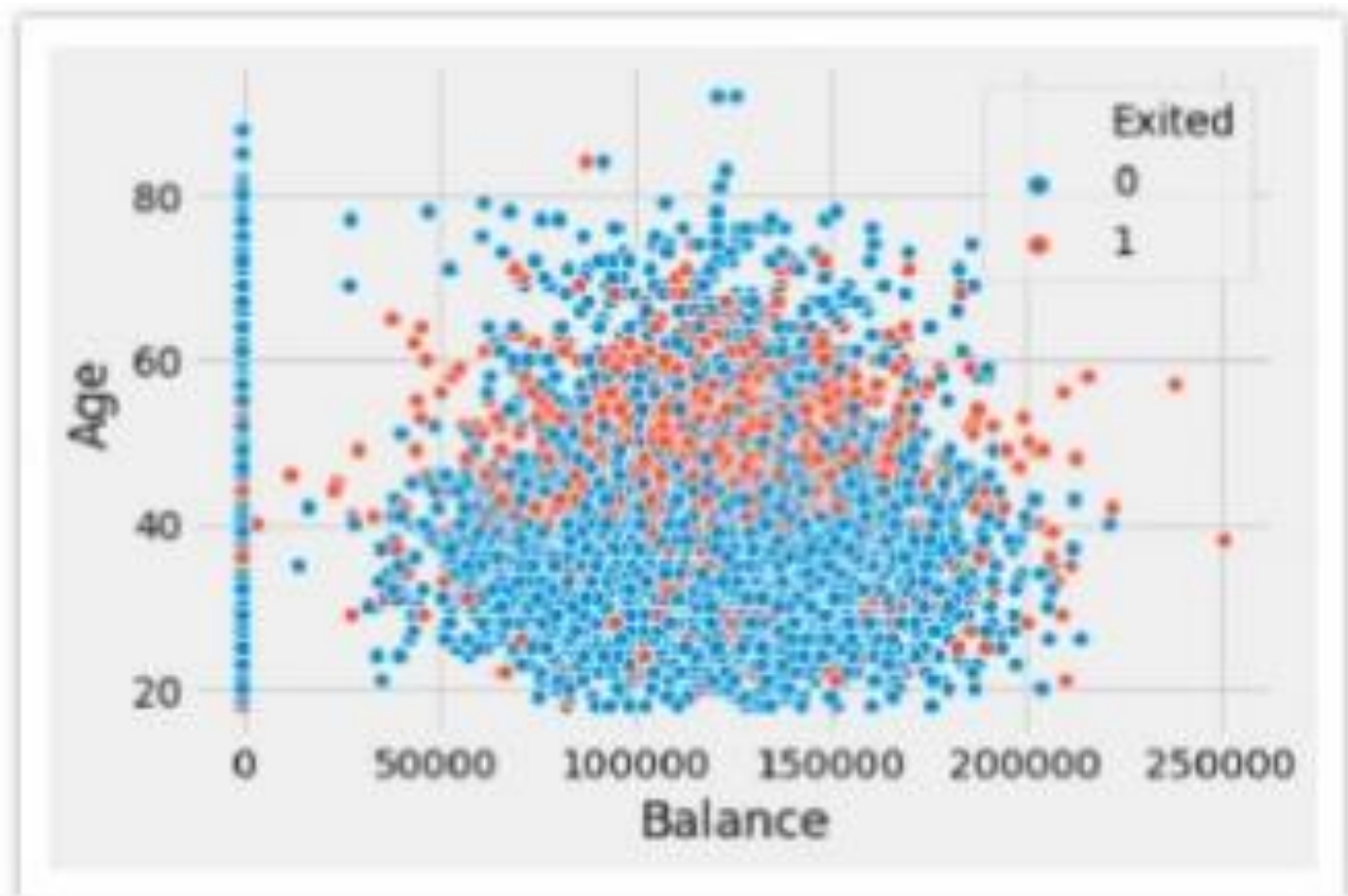
▲샘플링 후 데이터

1.3 핵심 아이디어/모델링

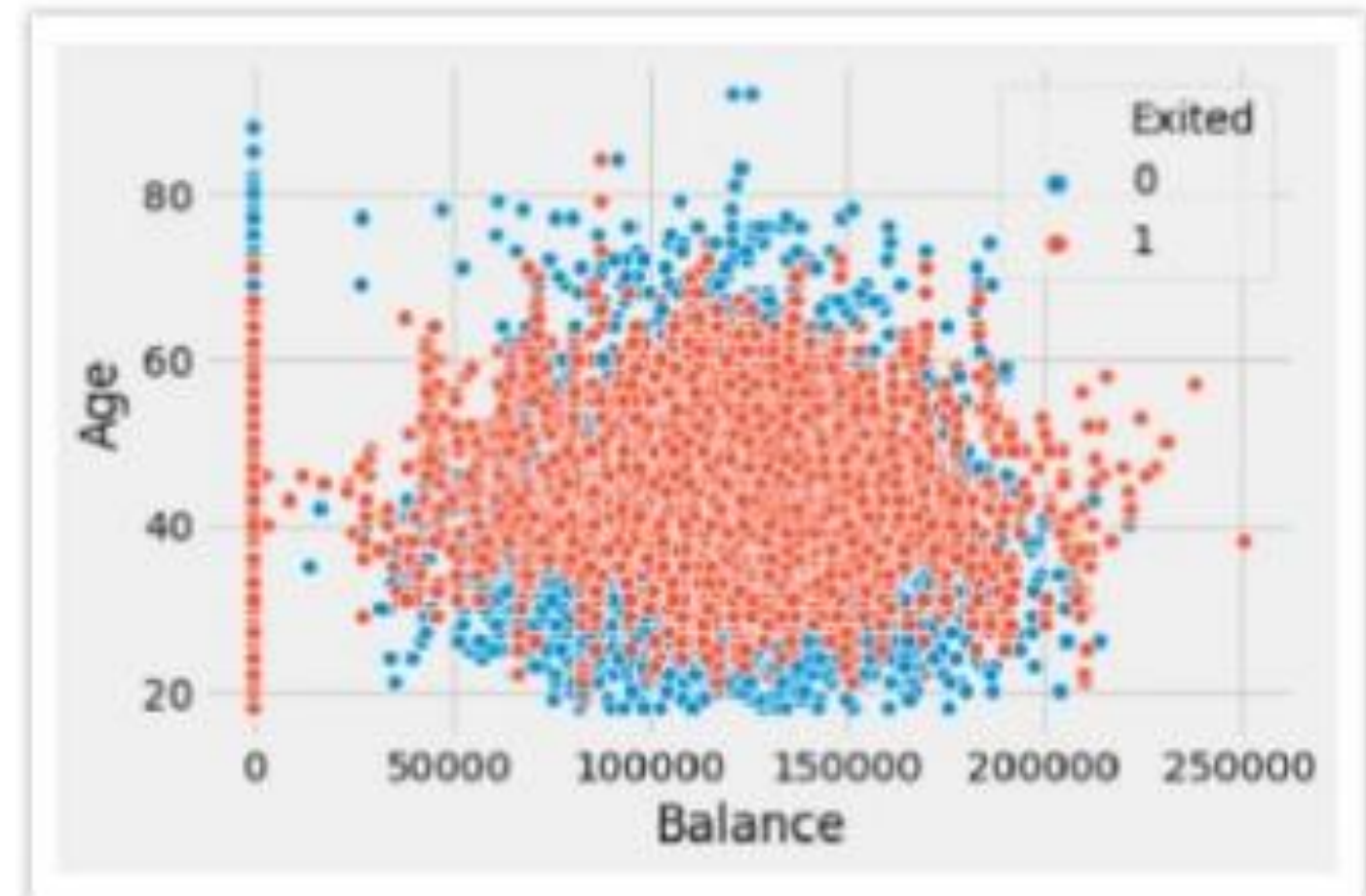
- Imbalance data 해결

3. Borderline SMOTE

Borderline SMOTE 방법은 Borderline 부분에 대해서만 SMOTE 방식을 사용



▲원본데이터



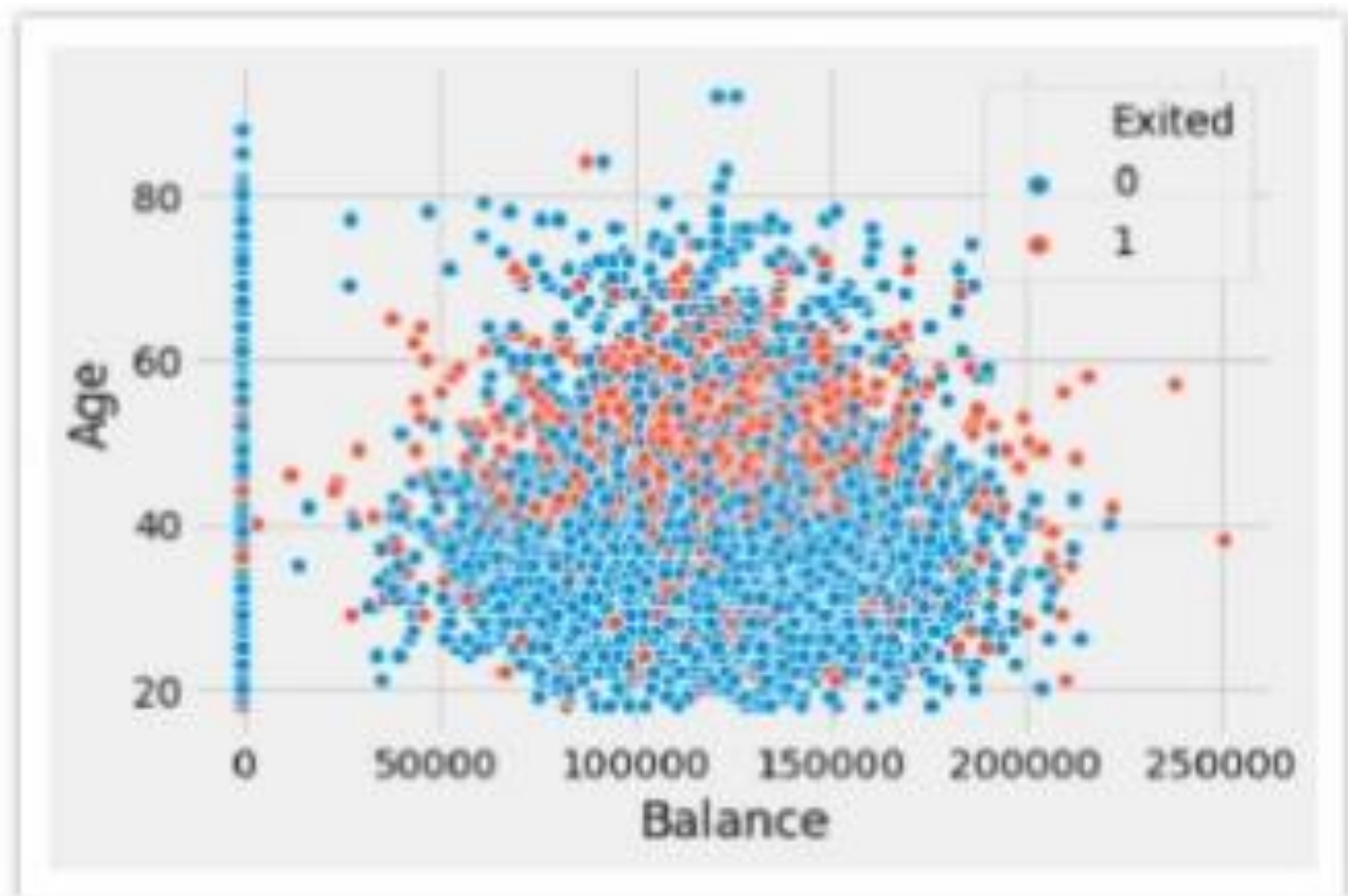
▲샘플링 후 데이터

1.3 핵심 아이디어/모델링

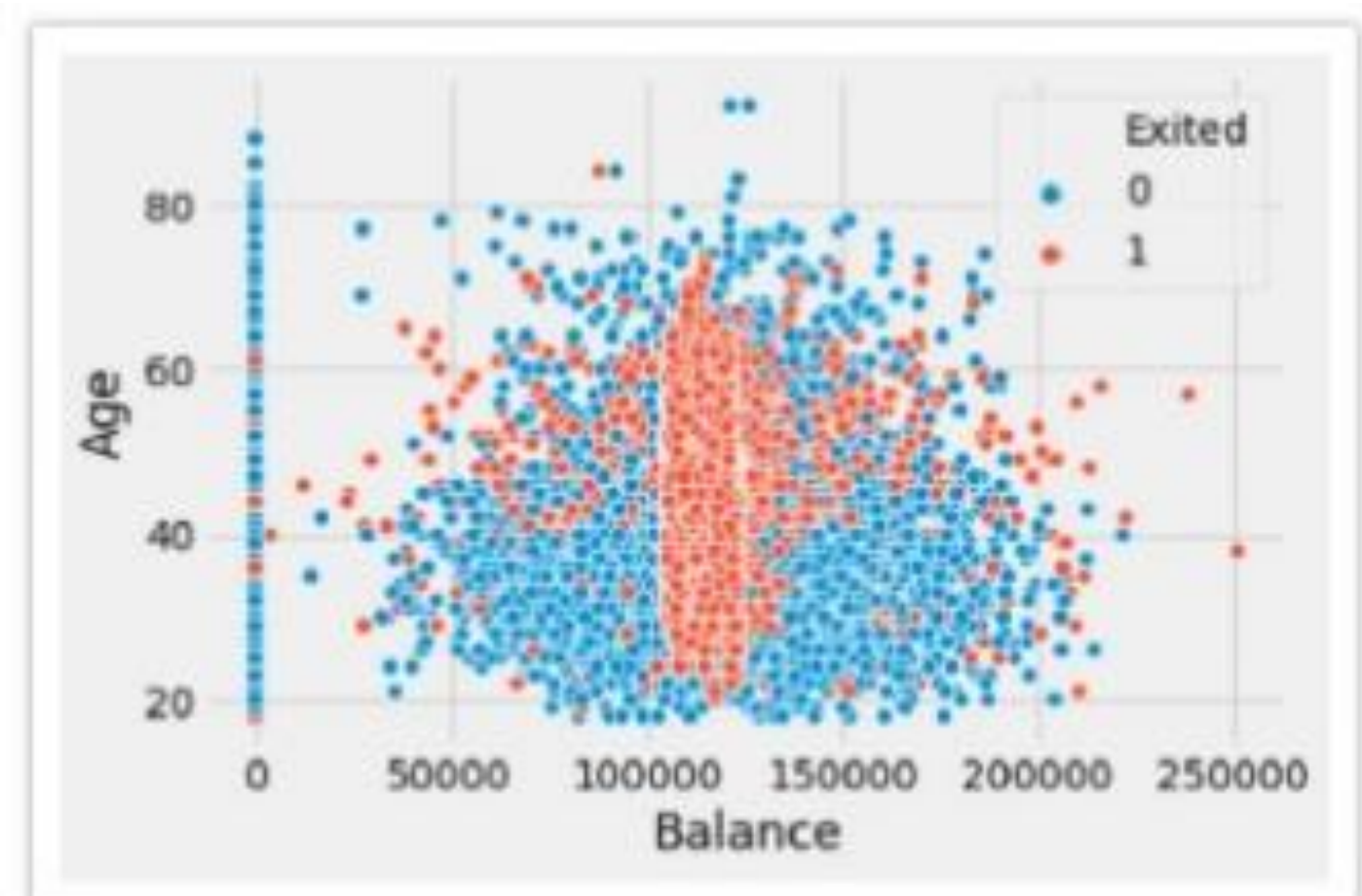
- Imbalance data 해결

4. KMeans Smote

K-Means SMOTE는 k-means클러스터링에 기반한 오버샘플링 방법
노이즈 생성을 방지



▲원본데이터



▲샘플링 후 데이터

1.3 핵심 아이디어/모델링

- Imbalance data 해결 (ROSE 오버샘플링)

이외에도 다음과 같은 오버샘플링 기법들이 있음

5. SVM Smote:

6. Adaptive Synthetic Sampling — ADASYN:

7. Smote-NC

- 오버 샘플링의 장단점

- 장점

데이터를 증가시키기 때문에 정보 손실이 없음

대부분의 경우 언더 샘플링에 비해 높은 분류 정확도를 보임

- 단점

데이터 증가로 인해 계산 시간이 증가할 수 있으며 과적합 가능성이 존재

노이즈 또는 이상치에 민감

1.4 Data analysis processing에 따른 코드/결과 분석

- ① 데이터 전처리
- ② 오버샘플링 전 XGBoost 모델 학습과 하이퍼 파라미터 튜닝
- ③ 오버샘플링 전 LightGBM 모델 학습과 하이퍼 파라미터 튜닝
- ④ Imbalance data 해결 (ROSE 오버샘플링)
- ⑤ 오버샘플링 후 XGBoost 모델 학습과 하이퍼 파라미터 튜닝
- ⑥ 오버샘플링 후 LightGBM 모델 학습과 하이퍼 파라미터 튜닝

1.4 Data analysis processing에 따른 코드/결과 분석

① 데이터 전처리

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib

cust_df = pd.read_csv("D:/Downloads/santander-customer-satisfaction/train_santander.csv")
print('dataset shape:', cust_df.shape)
cust_df.head(3)
```

dataset shape: (76020, 371)

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3
0	1	2	23	0.0	0.0	0.0
1	3	2	34	0.0	0.0	0.0
2	4	2	23	0.0	0.0	0.0

3 rows × 371 columns

- 371칼럼 (ID, TARGET포함)
- 불만족 비율이 0.04

```
cust_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76020 entries, 0 to 76019
Columns: 371 entries, ID to TARGET
dtypes: float64(111), int64(260)
memory usage: 215.2 MB
```

```
print(cust_df['TARGET'].value_counts())
unsatisfied_cnt = cust_df[cust_df['TARGET'] == 1].TARGET.count()
total_cnt = cust_df.TARGET.count()
print('unsatisfied 비율은 {0:.2f}'.format((unsatisfied_cnt / total_cnt)))
```

```
0    73012
1     3008
Name: TARGET, dtype: int64
unsatisfied 비율은 0.04
```

```
cust_df.describe()
```

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.363000
std	43781.947379	39033.462364	12.956486	1614.757313	339.315800
min	1.000000	-999999.000000	5.000000	0.000000	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000

8 rows × 371 columns

1.4 Data analysis processing에 따른 코드/결과 분석

① 데이터 전처리

```
# var3 피쳐 값 대체 및 ID 피쳐 드롭
cust_df['var3'].replace(-999999,2, inplace=True)
cust_df.drop('ID',axis=1, inplace=True)

# 피쳐 세트와 레이블 세트 분리. 레이블 컬럼은 DataFrame의 맨 마지막에 위치해 컬럼 위치
X_features = cust_df.iloc[:, :-1]
y_labels = cust_df.iloc[:, -1]
print('피쳐 데이터 shape:{0}'.format(X_features.shape))
```

피쳐 데이터 shape:(76020, 369)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_features, y_labels,
                                                    test_size=0.2, random_state=0)

train_cnt = y_train.count()
test_cnt = y_test.count()
print('학습 세트 Shape:{0}, 테스트 세트 Shape:{1}'.format(X_train.shape, X_test.shape))

print(' 학습 세트 레이블 값 분포 비율')
print(y_train.value_counts()/train_cnt)
print(' 테스트 세트 레이블 값 분포 비율')
print(y_test.value_counts()/test_cnt)
```

학습 세트 Shape:(60816, 369), 테스트 세트 Shape:(15204, 369)

학습 세트 레이블 값 분포 비율

0 0.960964

1 0.039036

Name: TARGET, dtype: float64

테스트 세트 레이블 값 분포 비율

0 0.9583

1 0.0417

Name: TARGET, dtype: float64

- Var3에 -9999999값을 2로 대체

- 학습세트와 테스트세트의 불만족 비율이 둘 다 0.04에 가까움

1.4 Data analysis processing에 따른 코드/결과 분석

② 오버샘플링 전 XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score

# n_estimators는 500으로, random state는 예제 수행 시마다 동일 예측 결과를 위해 설정.
xgb_clf = XGBClassifier(n_estimators=500, random_state=156)

# 성능 평가 지표를 auc로, 조기 중단 파라미터는 100으로 설정하고 학습 수행.
xgb_clf.fit(X_train, y_train, early_stopping_rounds=100,
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])

xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

```
[96] validation_0-auc:0.93094 validation_1-auc:0.83090
[97] validation_0-auc:0.93102 validation_1-auc:0.83091
[98] validation_0-auc:0.93179 validation_1-auc:0.83066
[99] validation_0-auc:0.93255 validation_1-auc:0.83058
[100] validation_0-auc:0.93296 validation_1-auc:0.83029
[101] validation_0-auc:0.93370 validation_1-auc:0.82955
[102] validation_0-auc:0.93369 validation_1-auc:0.82962
[103] validation_0-auc:0.93448 validation_1-auc:0.82893
[104] validation_0-auc:0.93460 validation_1-auc:0.82837
[105] validation_0-auc:0.93494 validation_1-auc:0.82815
[106] validation_0-auc:0.93594 validation_1-auc:0.82744
[107] validation_0-auc:0.93598 validation_1-auc:0.82728
[108] validation_0-auc:0.93625 validation_1-auc:0.82651
[109] validation_0-auc:0.93632 validation_1-auc:0.82650
[110] validation_0-auc:0.93673 validation_1-auc:0.82621
[111] validation_0-auc:0.93678 validation_1-auc:0.82620
[112] validation_0-auc:0.93726 validation_1-auc:0.82591
[113] validation_0-auc:0.93797 validation_1-auc:0.82498
ROC AUC: 0.8413
```

```
from sklearn.model_selection import GridSearchCV

# 하이퍼 파라미터 테스트의 수행 속도를 향상시키기 위해 n_estimators를 100으로 감소
xgb_clf = XGBClassifier(n_estimators=100)

params = {'max_depth':[5, 7], 'min_child_weight':[1,3], 'colsample_bytree':[0.5, 0.75]}

# cv는 3으로 지정
gridcv = GridSearchCV(xgb_clf, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="auc",
            eval_set=[(X_train, y_train), (X_test, y_test)])

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)

xgb_roc_score = roc_auc_score(y_test, gridcv.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

```
[27] validation_0-auc:0.87543 validation_1-auc:0.84245
[28] validation_0-auc:0.87604 validation_1-auc:0.84225
[29] validation_0-auc:0.87617 validation_1-auc:0.84249
[30] validation_0-auc:0.87673 validation_1-auc:0.84269
[31] validation_0-auc:0.87712 validation_1-auc:0.84240
[32] validation_0-auc:0.87736 validation_1-auc:0.84280
[33] validation_0-auc:0.87779 validation_1-auc:0.84267
[34] validation_0-auc:0.87801 validation_1-auc:0.84254
[35] validation_0-auc:0.87864 validation_1-auc:0.84245
[36] validation_0-auc:0.87932 validation_1-auc:0.84217
[37] validation_0-auc:0.87963 validation_1-auc:0.84184
[38] validation_0-auc:0.87984 validation_1-auc:0.84149
[39] validation_0-auc:0.88080 validation_1-auc:0.84133
[40] validation_0-auc:0.88106 validation_1-auc:0.84151
[41] validation_0-auc:0.88129 validation_1-auc:0.84146
[42] validation_0-auc:0.88226 validation_1-auc:0.84156
[43] validation_0-auc:0.88255 validation_1-auc:0.84143
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.5, 'max_depth': 5, 'min_child_weight': 3}
ROC AUC: 0.8445
```

- 파라미터 튜닝후 AUC 향상됨

1.4 Data analysis processing에 따른 코드/결과 분석

② 오버샘플링 전 XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
# n_estimators는 1000으로 증가시키고, learning_rate=0.02로 감소, reg_alpha=0.03으로 추가함.  
xgb_clf = XGBClassifier(n_estimators=1000, random_state=156, learning_rate=0.02, max_depth=5, #  
                        min_child_weight=3, colsample_bytree=0.5, reg_alpha=0.03)
```

```
# evaluation metric을 auc로, early stopping은 200 으로 설정하고 학습 수행.  
xgb_clf.fit(X_train, y_train, early_stopping_rounds=200,  
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])
```

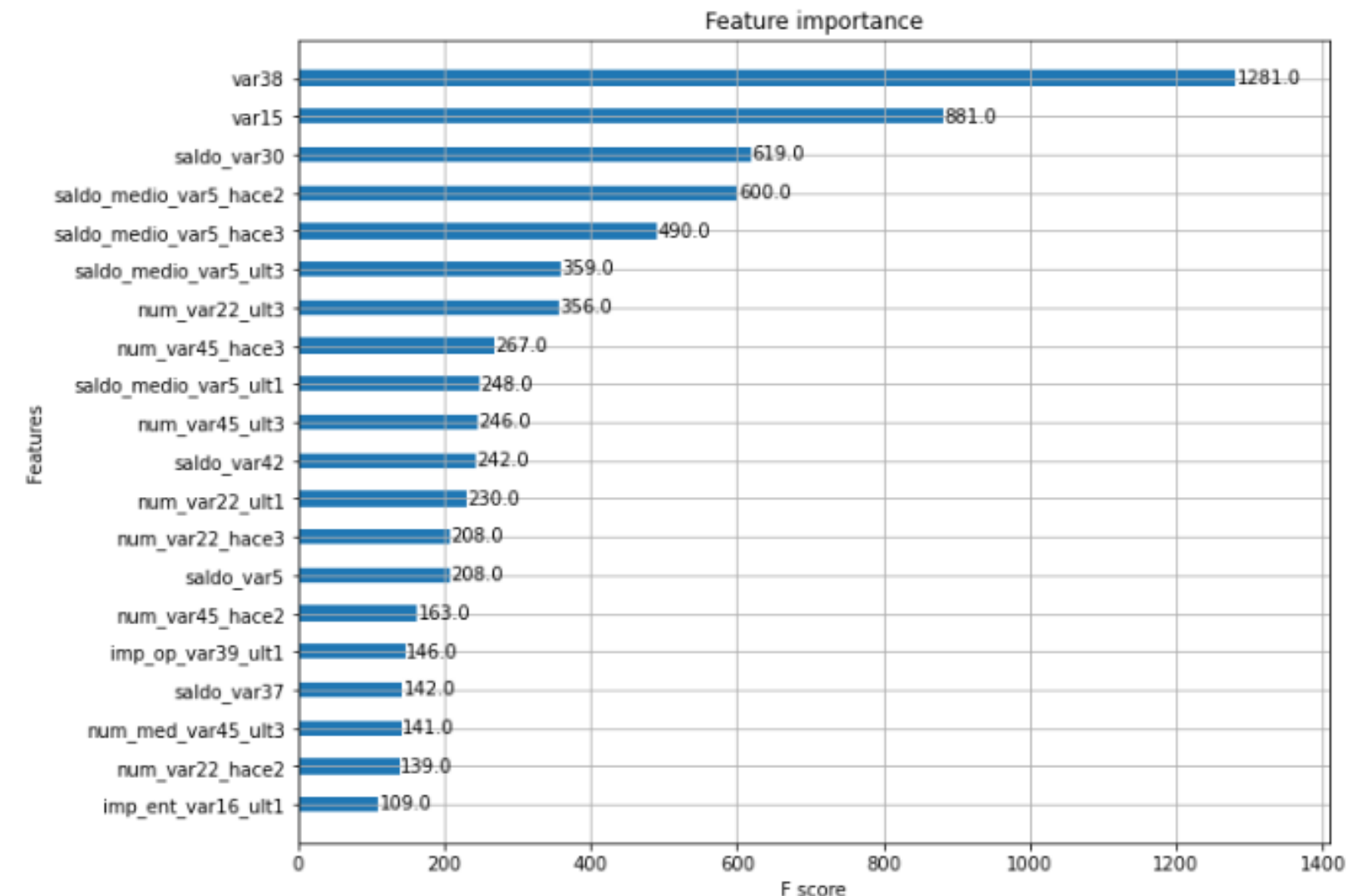
```
xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[: ,1], average='macro')  
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

```
[456] validation_0-auc:0.87925 validation_1-auc:0.84506  
[457] validation_0-auc:0.87932 validation_1-auc:0.84504  
[458] validation_0-auc:0.87935 validation_1-auc:0.84505  
[459] validation_0-auc:0.87942 validation_1-auc:0.84503  
[460] validation_0-auc:0.87945 validation_1-auc:0.84502  
[461] validation_0-auc:0.87950 validation_1-auc:0.84503  
[462] validation_0-auc:0.87951 validation_1-auc:0.84504  
[463] validation_0-auc:0.87956 validation_1-auc:0.84505  
[464] validation_0-auc:0.87957 validation_1-auc:0.84506  
[465] validation_0-auc:0.87964 validation_1-auc:0.84507  
[466] validation_0-auc:0.87968 validation_1-auc:0.84504  
[467] validation_0-auc:0.87969 validation_1-auc:0.84507  
[468] validation_0-auc:0.87971 validation_1-auc:0.84506  
[469] validation_0-auc:0.87973 validation_1-auc:0.84502  
[470] validation_0-auc:0.87974 validation_1-auc:0.84502  
[471] validation_0-auc:0.87976 validation_1-auc:0.84507  
[472] validation_0-auc:0.87984 validation_1-auc:0.84504  
[473] validation_0-auc:0.87986 validation_1-auc:0.84504  
ROC AUC: 0.8463
```

```
from xgboost import plot_importance  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
fig, ax = plt.subplots(1,1,figsize=(10,8))  
plot_importance(xgb_clf, ax=ax, max_num_features=20, height=0.4)
```

<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>



- Var38, var15가 중요 변수임을 확인

1.4 Data analysis processing에 따른 코드/결과 분석

③ 오버샘플링 전 LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=500)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
            verbose=True)

lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

```
[98] valid_0's auc: 0.836675 valid_0's binary_logloss: 0.140361
[99] valid_0's auc: 0.83655 valid_0's binary_logloss: 0.14039
[100] valid_0's auc: 0.836518 valid_0's binary_logloss: 0.1404
[101] valid_0's auc: 0.836998 valid_0's binary_logloss: 0.140294
[102] valid_0's auc: 0.836778 valid_0's binary_logloss: 0.140366
[103] valid_0's auc: 0.83694 valid_0's binary_logloss: 0.140333
[104] valid_0's auc: 0.836749 valid_0's binary_logloss: 0.14039
[105] valid_0's auc: 0.836752 valid_0's binary_logloss: 0.140391
[106] valid_0's auc: 0.837197 valid_0's binary_logloss: 0.140305
[107] valid_0's auc: 0.837141 valid_0's binary_logloss: 0.140329
[108] valid_0's auc: 0.8371 valid_0's binary_logloss: 0.140344
[109] valid_0's auc: 0.837136 valid_0's binary_logloss: 0.14033
[110] valid_0's auc: 0.837102 valid_0's binary_logloss: 0.140388
[111] valid_0's auc: 0.836957 valid_0's binary_logloss: 0.140426
[112] valid_0's auc: 0.836779 valid_0's binary_logloss: 0.14051
[113] valid_0's auc: 0.836831 valid_0's binary_logloss: 0.140526
[114] valid_0's auc: 0.836783 valid_0's binary_logloss: 0.14055
[115] valid_0's auc: 0.836672 valid_0's binary_logloss: 0.140585
ROC AUC: 0.8409
```

```
from sklearn.model_selection import GridSearchCV

# 하이퍼 파라미터 테스트의 수행 속도를 향상시키기 위해 n_estimators를 100으로 감소
lgbm_clf = LGBMClassifier(n_estimators=200)

params = {'num_leaves': [32, 64],
          'max_depth': [128, 160],
          'min_child_samples': [60, 100],
          'subsample': [0.8, 1]}

# cv는 3으로 지정
gridcv = GridSearchCV(lgbm_clf, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="auc",
          eval_set=[(X_train, y_train), (X_test, y_test)])

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)
lgbm_roc_score = roc_auc_score(y_test, gridcv.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

```
[34] training's auc: 0.882181 training's binary_logloss: 0.122567 valid_1's au
25 valid_1's binary_logloss: 0.139275
[35] training's auc: 0.883237 training's binary_logloss: 0.122275 valid_1's au
533 valid_1's binary_logloss: 0.139208
[36] training's auc: 0.884433 training's binary_logloss: 0.121989 valid_1's au
446 valid_1's binary_logloss: 0.139217
[37] training's auc: 0.885423 training's binary_logloss: 0.121707 valid_1's au
379 valid_1's binary_logloss: 0.139221
[38] training's auc: 0.88628 training's binary_logloss: 0.121411 valid_1's auc: 0.838
d_1's binary_logloss: 0.139254
[39] training's auc: 0.886985 training's binary_logloss: 0.121175 valid_1's au
432 valid_1's binary_logloss: 0.139181
[40] training's auc: 0.887543 training's binary_logloss: 0.120933 valid_1's au
247 valid_1's binary_logloss: 0.139215
[41] training's auc: 0.888425 training's binary_logloss: 0.120677 valid_1's au
26 valid_1's binary_logloss: 0.139218
GridSearchCV 최적 파라미터: {'max_depth': 128, 'min_child_samples': 100, 'num_leaves': 32, '
e': 0.8}
ROC AUC: 0.8417
```

- 파라미터 튜닝후 AUC 향상됨

1.4 Data analysis processing에 따른 코드/결과 분석

③ 오버샘플링 전 LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=32, subsample=0.8, min_child_samples=100,
                          max_depth=128)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
            verbose=True)

lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:,-1], average='macro')
print('ROC AUC: {:.4f}'.format(lgbm_roc_score))
```

```
[94] valid_0's auc: 0.83596 valid_0's binary_logloss: 0.139981
[95] valid_0's auc: 0.835924 valid_0's binary_logloss: 0.140014
[96] valid_0's auc: 0.835947 valid_0's binary_logloss: 0.140001
[97] valid_0's auc: 0.835798 valid_0's binary_logloss: 0.140069
[98] valid_0's auc: 0.835699 valid_0's binary_logloss: 0.140112
[99] valid_0's auc: 0.835598 valid_0's binary_logloss: 0.140139
[100] valid_0's auc: 0.835567 valid_0's binary_logloss: 0.140156
[101] valid_0's auc: 0.83541 valid_0's binary_logloss: 0.140183
[102] valid_0's auc: 0.835235 valid_0's binary_logloss: 0.140234
[103] valid_0's auc: 0.835304 valid_0's binary_logloss: 0.140213
[104] valid_0's auc: 0.834946 valid_0's binary_logloss: 0.140301
[105] valid_0's auc: 0.834578 valid_0's binary_logloss: 0.140374
[106] valid_0's auc: 0.834617 valid_0's binary_logloss: 0.140395
[107] valid_0's auc: 0.834575 valid_0's binary_logloss: 0.14042
[108] valid_0's auc: 0.834393 valid_0's binary_logloss: 0.140467
[109] valid_0's auc: 0.834307 valid_0's binary_logloss: 0.14051
[110] valid_0's auc: 0.834382 valid_0's binary_logloss: 0.14051
[111] valid_0's auc: 0.83436 valid_0's binary_logloss: 0.14054
ROC AUC: 0.8417
```

1.4 Data analysis processing에 따른 코드/결과 분석

④ Imbalance data 해결 (ROSE 오버샘플링)

random over sampling

```
from imblearn.over_sampling import RandomOverSampler  
X_train, y_train = RandomOverSampler(random_state=0).fit_resample(X_features, y_labels)
```

```
X_train.shape
```

```
(146024, 369)
```

```
y_train.shape
```

```
(146024,)
```

```
y_train.value_counts()  
#비율이 1:10이 됨을 확인
```

```
0    73012
```

```
1    73012
```

```
Name: TARGET, dtype: int64
```

- 오버샘플링 후 0과 1의 비율이 1:10이 됨

1.4 Data analysis processing에 따른 코드/결과 분석

⑤ 오버샘플링 후 XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score

# n_estimators는 500으로, random state는 예제 수행 시마다 동일 예측 결과를 위해 설정.
xgb_clf = XGBClassifier(n_estimators=500, random_state=156)

# 성능 평가 지표를 auc로, 조기 중단 파라미터는 100으로 설정하고 학습 수행.
xgb_clf.fit(X_train, y_train, early_stopping_rounds=100,
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])

xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {:.4f}'.format(xgb_roc_score))
```

```
[482] validation_0-auc:0.98872 validation_1-auc:0.98838
[483] validation_0-auc:0.98892 validation_1-auc:0.98854
[484] validation_0-auc:0.98899 validation_1-auc:0.98858
[485] validation_0-auc:0.98906 validation_1-auc:0.98861
[486] validation_0-auc:0.98906 validation_1-auc:0.98862
[487] validation_0-auc:0.98921 validation_1-auc:0.98873
[488] validation_0-auc:0.98925 validation_1-auc:0.98877
[489] validation_0-auc:0.98928 validation_1-auc:0.98880
[490] validation_0-auc:0.98935 validation_1-auc:0.98889
[491] validation_0-auc:0.98940 validation_1-auc:0.98894
[492] validation_0-auc:0.98943 validation_1-auc:0.98899
[493] validation_0-auc:0.98946 validation_1-auc:0.98901
[494] validation_0-auc:0.98948 validation_1-auc:0.98905
[495] validation_0-auc:0.98965 validation_1-auc:0.98921
[496] validation_0-auc:0.98966 validation_1-auc:0.98920
[497] validation_0-auc:0.98967 validation_1-auc:0.98920
[498] validation_0-auc:0.98967 validation_1-auc:0.98920
[499] validation_0-auc:0.98967 validation_1-auc:0.98920
ROC AUC: 0.9892
```

1.4 Data analysis processing에 따른 코드/결과 분석

⑤ 오버샘플링 후 XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
from sklearn.model_selection import GridSearchCV

# 하이퍼 파라미터 테스트의 수행 속도를 향상시키기 위해 n_estimators를 50으로 감소
xgb_clf = XGBClassifier(n_estimators=50)

params = {'max_depth':[5, 7] , 'min_child_weight':[1,3] , 'colsample_bytree':[0.5, 0.75] }

# cv는 3으로 지정
gridcv = GridSearchCV(xgb_clf, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="auc",
          eval_set=[(X_train, y_train), (X_test, y_test)])

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)

xgb_roc_score = roc_auc_score(y_test, gridcv.predict_proba(X_test)[:,-1], average='macro')
print('ROC AUC: {:.4f}'.format(xgb_roc_score))
```

```
[33] validation_0-auc:0.91871 validation_1-auc:0.91918
[34] validation_0-auc:0.91974 validation_1-auc:0.92067
[35] validation_0-auc:0.92091 validation_1-auc:0.92194
[36] validation_0-auc:0.92214 validation_1-auc:0.92297
[37] validation_0-auc:0.92233 validation_1-auc:0.92305
[38] validation_0-auc:0.92246 validation_1-auc:0.92335
[39] validation_0-auc:0.92276 validation_1-auc:0.92377
[40] validation_0-auc:0.92402 validation_1-auc:0.92493
[41] validation_0-auc:0.92606 validation_1-auc:0.92673
[42] validation_0-auc:0.92749 validation_1-auc:0.92790
[43] validation_0-auc:0.92768 validation_1-auc:0.92806
[44] validation_0-auc:0.92795 validation_1-auc:0.92833
[45] validation_0-auc:0.92879 validation_1-auc:0.92887
[46] validation_0-auc:0.92970 validation_1-auc:0.92994
[47] validation_0-auc:0.92976 validation_1-auc:0.93000
[48] validation_0-auc:0.93054 validation_1-auc:0.93065
[49] validation_0-auc:0.93077 validation_1-auc:0.93070
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.5, 'max_depth': 7, 'min_child_weight': 1}
ROC AUC: 0.9307
```


1.4 Data analysis processing에 따른 코드/결과 분석

⑤ 오버샘플링 후 XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
# n_estimators는 1000으로 증가시키고, learning_rate=0.02로 감소, reg_alpha=0.03으로 추가함.
xgb_clf = XGBClassifier(n_estimators=1000, random_state=156, learning_rate=0.02, max_depth=7, #
                        min_child_weight=1, colsample_bytree=0.5, reg_alpha=0.03)
```

```
# evaluation metric을 auc로, early stopping은 200 으로 설정하고 학습 수행.
xgb_clf.fit(X_train, y_train, early_stopping_rounds=200,
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])
```

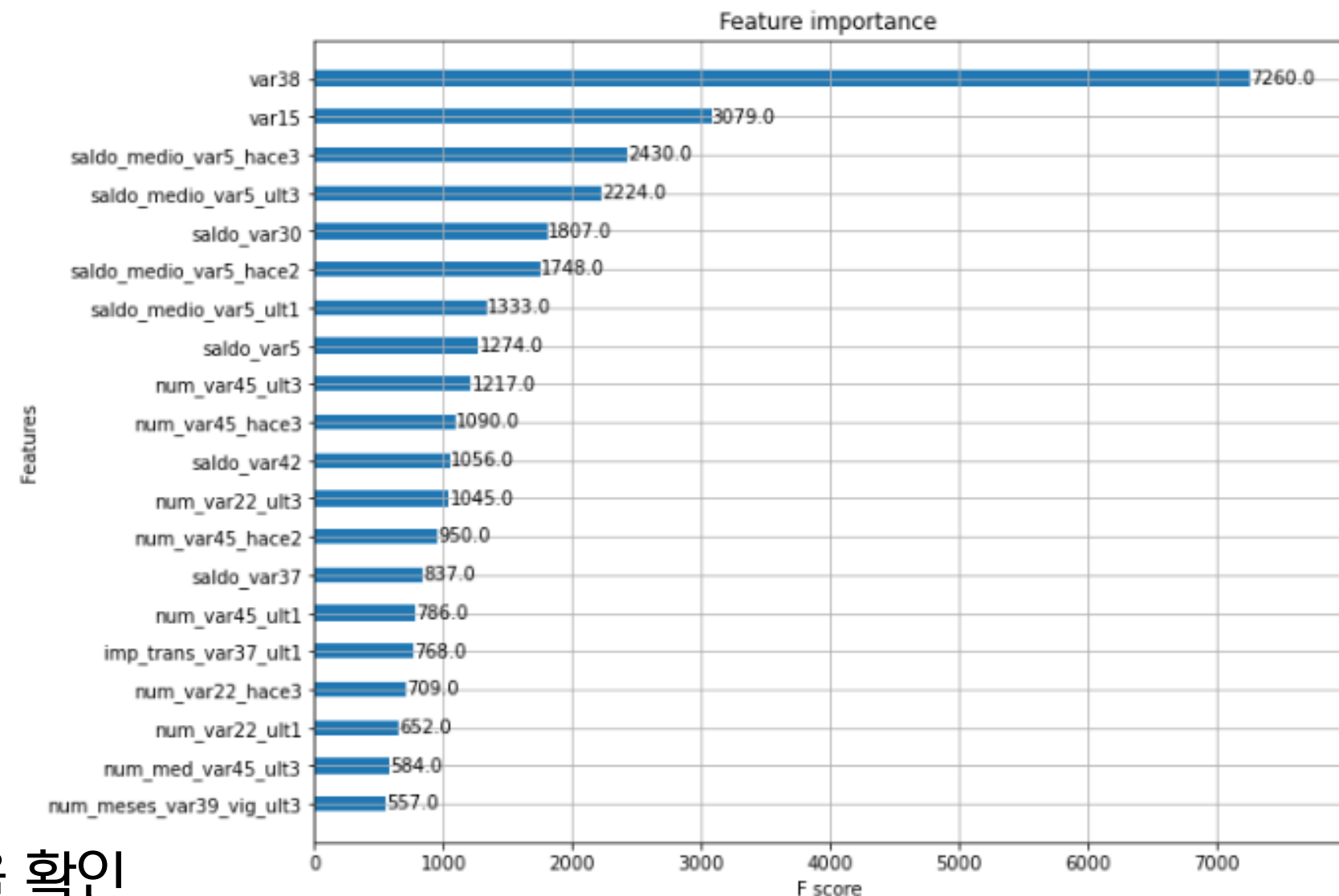
```
xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

```
[982] validation_0-auc:0.93685 validation_1-auc:0.93718
[983] validation_0-auc:0.93686 validation_1-auc:0.93719
[984] validation_0-auc:0.93687 validation_1-auc:0.93721
[985] validation_0-auc:0.93689 validation_1-auc:0.93723
[986] validation_0-auc:0.93689 validation_1-auc:0.93724
[987] validation_0-auc:0.93700 validation_1-auc:0.93730
[988] validation_0-auc:0.93710 validation_1-auc:0.93749
[989] validation_0-auc:0.93714 validation_1-auc:0.93752
[990] validation_0-auc:0.93718 validation_1-auc:0.93754
[991] validation_0-auc:0.93719 validation_1-auc:0.93755
[992] validation_0-auc:0.93721 validation_1-auc:0.93759
[993] validation_0-auc:0.93721 validation_1-auc:0.93759
[994] validation_0-auc:0.93723 validation_1-auc:0.93761
[995] validation_0-auc:0.93726 validation_1-auc:0.93763
[996] validation_0-auc:0.93730 validation_1-auc:0.93771
[997] validation_0-auc:0.93733 validation_1-auc:0.93774
[998] validation_0-auc:0.93734 validation_1-auc:0.93775
[999] validation_0-auc:0.93736 validation_1-auc:0.93778
ROC AUC: 0.9378
```

```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline
```

```
fig, ax = plt.subplots(1,1,figsize=(10,8))
plot_importance(xgb_clf, ax=ax, max_num_features=20, height=0.4)
```

```
<AxesSubplot:title={ 'center': 'Feature importance'}, xlabel='F score', ylabel='Features'>
```



- Var38, var15가 중요 변수임을 확인

1.4 Data analysis processing에 따른 코드/결과 분석

⑥ 오버샘플링 후 LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=500)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
            verbose=True)

lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {:.4f}'.format(lgbm_roc_score))
```

```
[483] valid_0's auc: 0.961986 valid_0's binary_logloss: 0.280507
[484] valid_0's auc: 0.962072 valid_0's binary_logloss: 0.280184
[485] valid_0's auc: 0.962109 valid_0's binary_logloss: 0.279965
[486] valid_0's auc: 0.962133 valid_0's binary_logloss: 0.279837
[487] valid_0's auc: 0.962228 valid_0's binary_logloss: 0.279532
[488] valid_0's auc: 0.962255 valid_0's binary_logloss: 0.279432
[489] valid_0's auc: 0.962279 valid_0's binary_logloss: 0.279291
[490] valid_0's auc: 0.96233 valid_0's binary_logloss: 0.279044
[491] valid_0's auc: 0.962363 valid_0's binary_logloss: 0.278892
[492] valid_0's auc: 0.962443 valid_0's binary_logloss: 0.278712
[493] valid_0's auc: 0.962576 valid_0's binary_logloss: 0.278311
[494] valid_0's auc: 0.962694 valid_0's binary_logloss: 0.277861
[495] valid_0's auc: 0.962706 valid_0's binary_logloss: 0.277779
[496] valid_0's auc: 0.962712 valid_0's binary_logloss: 0.277724
[497] valid_0's auc: 0.962734 valid_0's binary_logloss: 0.27756
[498] valid_0's auc: 0.962754 valid_0's binary_logloss: 0.277484
[499] valid_0's auc: 0.962761 valid_0's binary_logloss: 0.277368
[500] valid_0's auc: 0.962799 valid_0's binary_logloss: 0.277281
ROC AUC: 0.9628
```


1.4 Data analysis processing에 따른 코드/결과 분석

⑥ 오버샘플링 후 LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
from sklearn.model_selection import GridSearchCV

# 하이퍼 파라미터 테스트의 수행 속도를 향상시키기 위해 n_estimators를 100으로 감소
lgbm_clf = LGBMClassifier(n_estimators=200)

params = {'num_leaves': [32, 64],
          'max_depth': [128, 160],
          'min_child_samples': [60, 100],
          'subsample': [0.8, 1]}

# cv는 3으로 지정
gridcv = GridSearchCV(lgbm_clf, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="auc",
          eval_set=[(X_train, y_train), (X_test, y_test)])

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)
lgbm_roc_score = roc_auc_score(y_test, gridcv.predict_proba(X_test)[:,:1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

```
s: 0.29202
[193] training's auc: 0.959502 training's binary_logloss: 0.263668 valid_1's auc: 0.95877 valid_1's binary_logloss: 0.291437
[194] training's auc: 0.959574 training's binary_logloss: 0.263419 valid_1's auc: 0.958868 valid_1's binary_logloss: 0.291073
[195] training's auc: 0.95972 training's binary_logloss: 0.262818 valid_1's auc: 0.959051 valid_1's binary_logloss: 0.291575
[196] training's auc: 0.95978 training's binary_logloss: 0.262585 valid_1's auc: 0.959096 valid_1's binary_logloss: 0.291333
[197] training's auc: 0.959862 training's binary_logloss: 0.262257 valid_1's auc: 0.959173 valid_1's binary_logloss: 0.290021
[198] training's auc: 0.959935 training's binary_logloss: 0.261967 valid_1's auc: 0.95925 valid_1's binary_logloss: 0.289758
[199] training's auc: 0.959973 training's binary_logloss: 0.261776 valid_1's auc: 0.959275 valid_1's binary_logloss: 0.289568
[200] training's auc: 0.960027 training's binary_logloss: 0.261539 valid_1's auc: 0.959335 valid_1's binary_logloss: 0.289326

GridSearchCV 최적 파라미터: {'max_depth': 128, 'min_child_samples': 60, 'num_leaves': 64, 'subsample': 0.8}
ROC AUC: 0.9593
```

1.4 Data analysis processing에 따른 코드/결과 분석

⑥ 오버샘플링 후 LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, subsample=0.8, min_child_samples=60,
                          max_depth=128)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
            verbose=True)

lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:,1], average='macro')
print('ROC AUC: {:.4f}'.format(lgbm_roc_score))
```

```
[983] valid_0's auc: 0.993068 valid_0's binary_logloss: 0.130344
[984] valid_0's auc: 0.993085 valid_0's binary_logloss: 0.130168
[985] valid_0's auc: 0.993104 valid_0's binary_logloss: 0.130013
[986] valid_0's auc: 0.993128 valid_0's binary_logloss: 0.129818
[987] valid_0's auc: 0.993135 valid_0's binary_logloss: 0.129712
[988] valid_0's auc: 0.99314 valid_0's binary_logloss: 0.129619
[989] valid_0's auc: 0.993148 valid_0's binary_logloss: 0.129555
[990] valid_0's auc: 0.993147 valid_0's binary_logloss: 0.12952
[991] valid_0's auc: 0.993166 valid_0's binary_logloss: 0.129359
[992] valid_0's auc: 0.993169 valid_0's binary_logloss: 0.12932
[993] valid_0's auc: 0.993169 valid_0's binary_logloss: 0.129299
[994] valid_0's auc: 0.993167 valid_0's binary_logloss: 0.129262
[995] valid_0's auc: 0.99319 valid_0's binary_logloss: 0.129106
[996] valid_0's auc: 0.99319 valid_0's binary_logloss: 0.129085
[997] valid_0's auc: 0.993227 valid_0's binary_logloss: 0.128849
[998] valid_0's auc: 0.993227 valid_0's binary_logloss: 0.128819
[999] valid_0's auc: 0.993258 valid_0's binary_logloss: 0.128663
[1000] valid_0's auc: 0.993264 valid_0's binary_logloss: 0.128599
ROC AUC: 0.9933
```

1.5 노트의 결론

데이터 가공 유형	머신러닝 알고리즘	정확도	정밀도	재현율	F1-score	ROC-AUC
원본데이터 (가공없음)	XGBoost	0.8115	0.1626	0.8486	0.2729	0.8463
	LightGBM	0.9023	0.2954	0.9700	0.4529	0.8417
ROSE 오버샘플링	XGBoost	0.8487	0.2029	0.8975	0.3310	0.9378
	LightGBM	0.9389	0.4043	0.9826	0.5729	0.9933

- ROSE 오버샘플링 결과, 모든 지수에서 향상된 결과를 보임

02. 캐글 신용카드 사기 검출

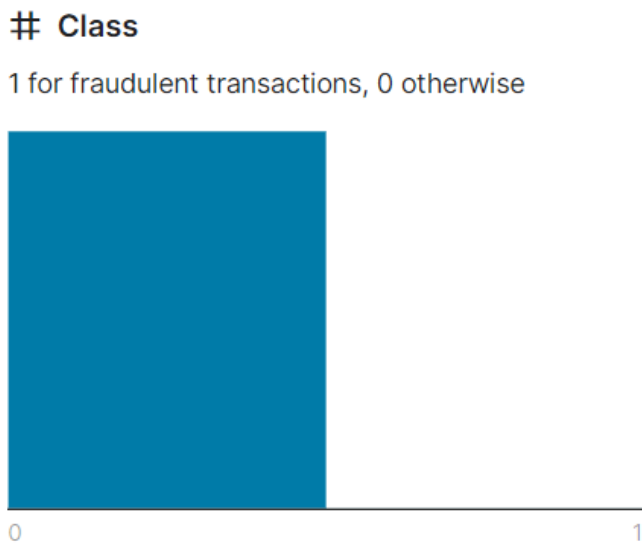


2.1 신용카드사기검출

☞ Credit Card Fraud Detection(신용카드사기검출)-유럽의 2013년 9월 신용카드거래

<목적>신용 카드 회사는 사기성 신용 카드 거래를 인식하여 고객이 구매하지 않은 항목에 대해 요금을 청구하지 않도록 하는 것

<칼럼> V1, V2, ... V28 ,시간,Amount(거래 금액),Class(레이블,사기1 or 정상0)
데이터 세트는 284,807건의 거래 중 492건의 사기로 매우 불균형하며 1-클래스(사기) 0.172%



<문제>
이상레이블 가지는 데이터 건수가 상대적으로 너무 적기 때문에 다양한 유형을 학습 못하게 되고, 이상데이터 검출 어려움
높은 정확도-낮은 재현률
<해결방법>
적절한 학습 데이터 확보를 위해 오버샘플링, 언더샘플링 방법 적용

2.2 오버샘플링

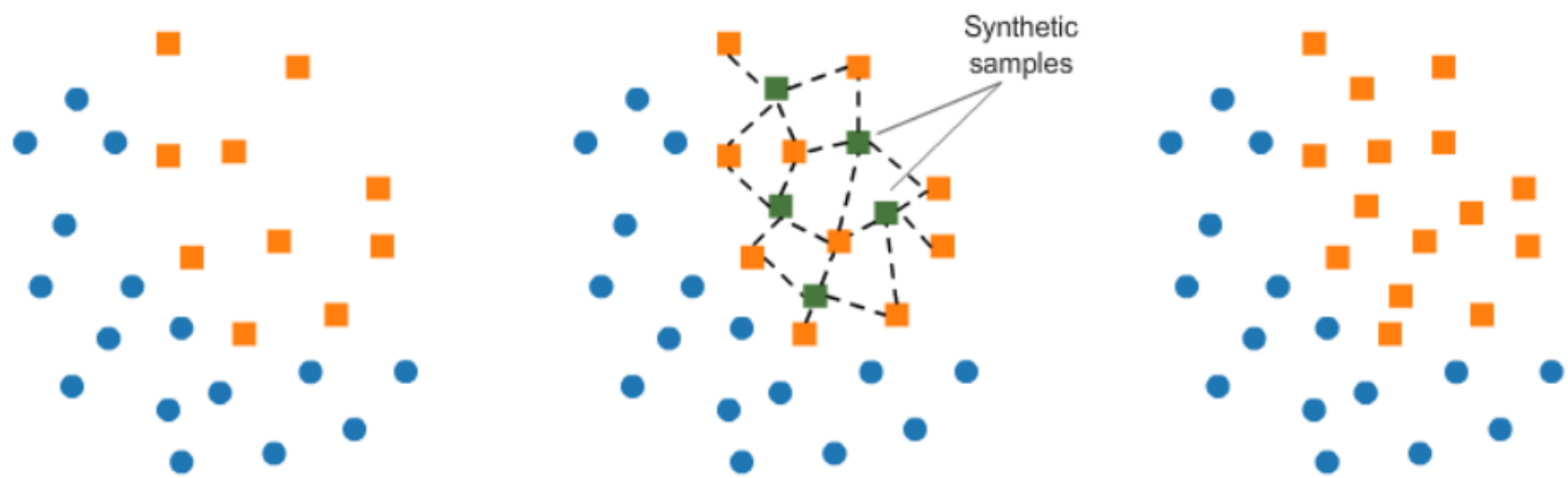
오버샘플링

minor적은 데이터 세트를 증식하여 학습을 위한 충분한 데이터를 확보함

단순증식(완전 동일)은 과적합 문제가 있으므로 변형증식을 해야 함

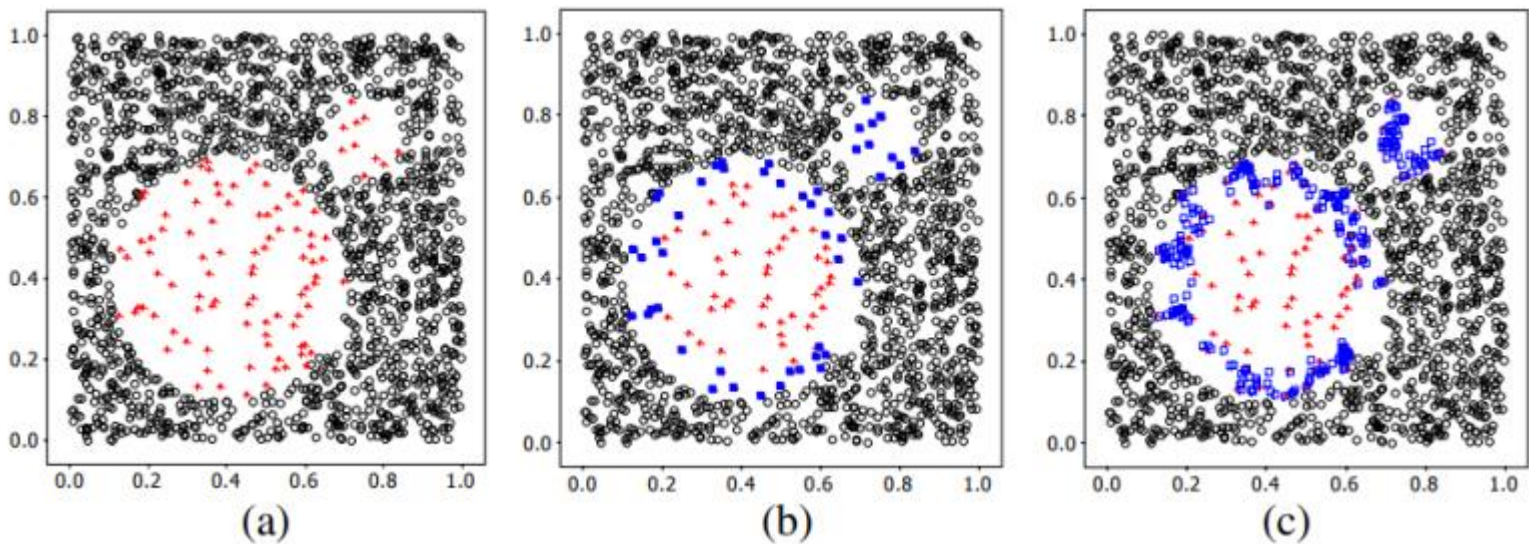


SMOTE(Synthetic Minority Over-sampling Technique)



k개 이웃간의 차difference를 구하고, 0 ~ 1 사이의 임의의 값을 곱하여 원래 샘플에 더함

Borderline-SMOTE



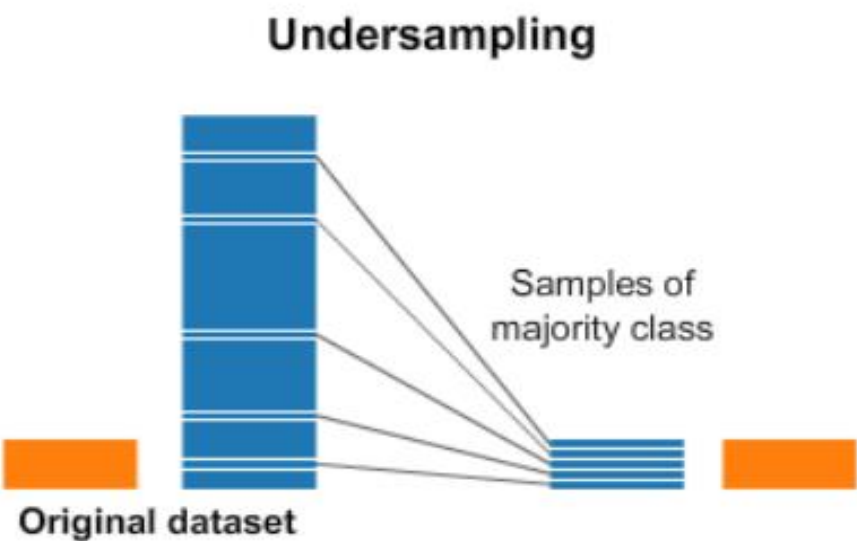
다른 클래스와의 경계borderline에 있는 minor샘플들을 늘림

2.3 언더샘플링

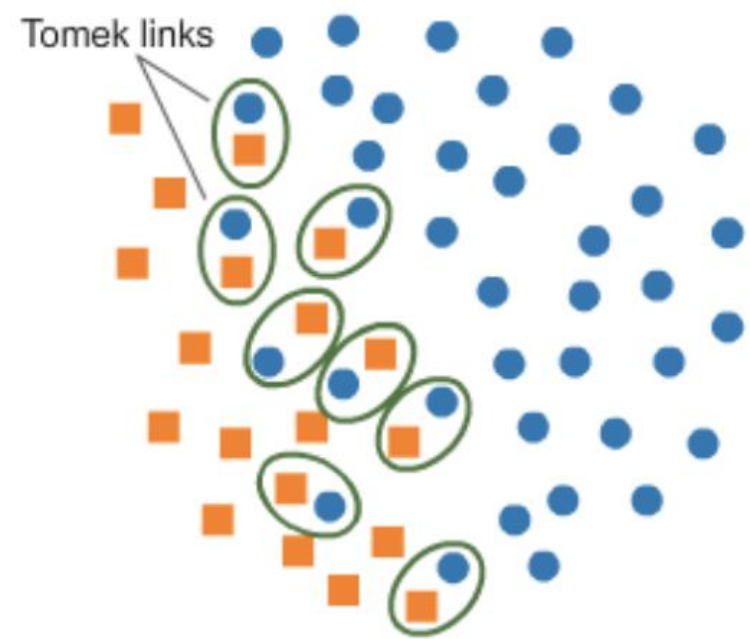
☞ 언더샘플링

major많은 데이터셋을 감소시켜 데이터 불균형 해소, 계산시간 감소

BUT! 학습에 사용되는 전체 데이터 수를 급격하게 감소시켜 오히려 성능이 떨어질 수 있음

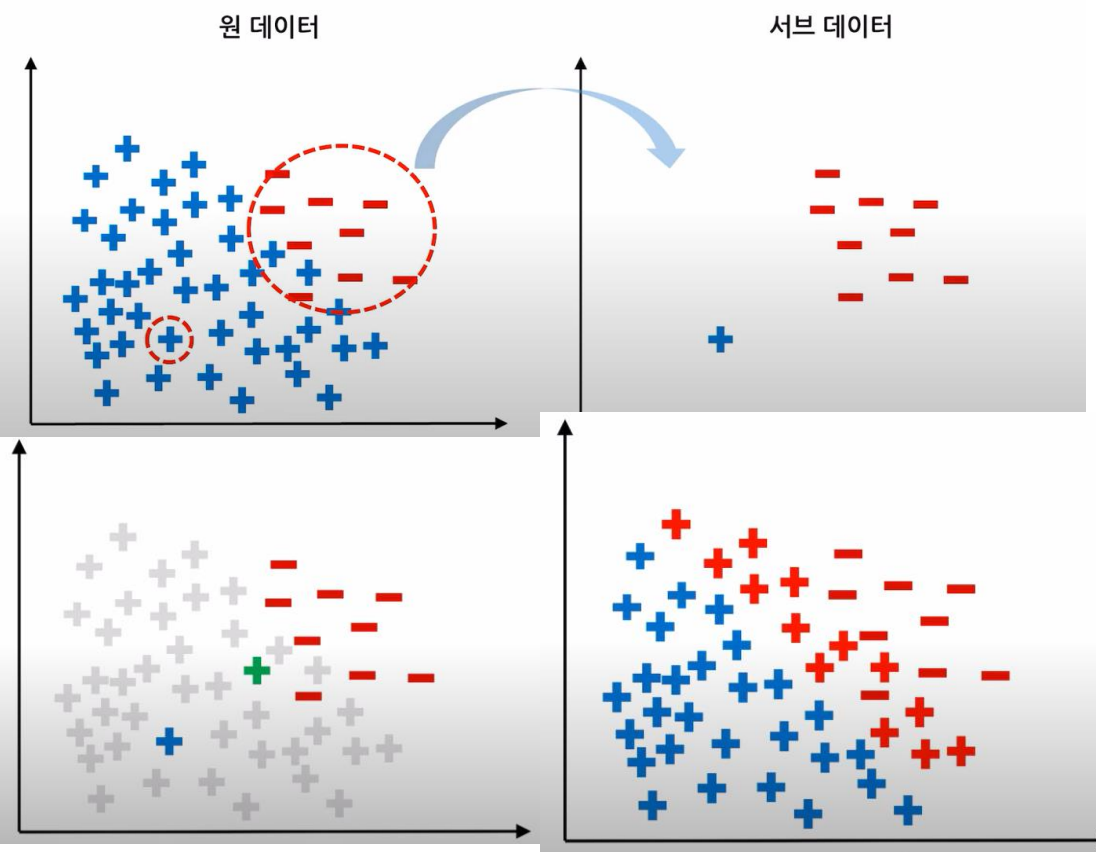


TomekLinks



토멕링크에서 major데이터 삭제

CNN

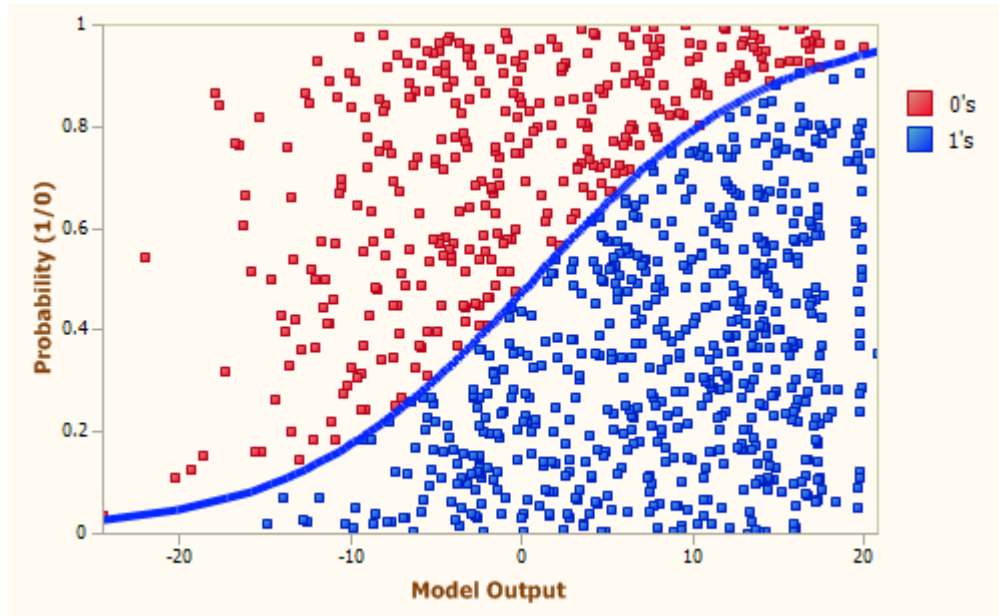
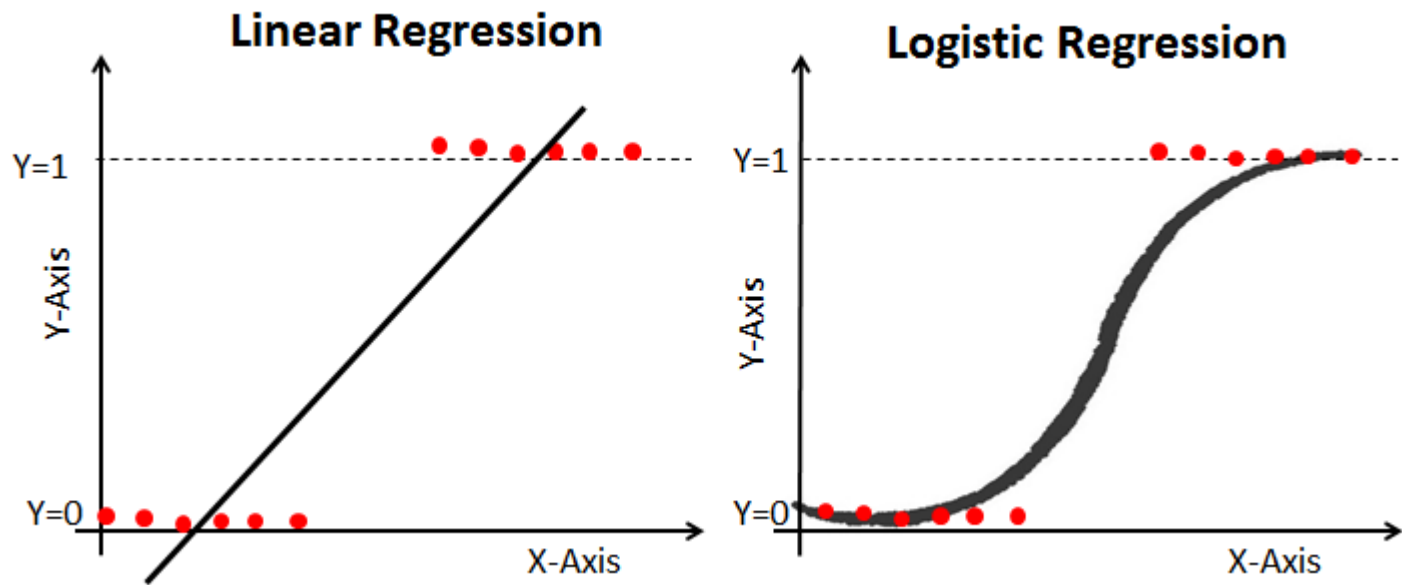


OSS

RandomSampling

2.4 로지스틱 회귀, LightGBM

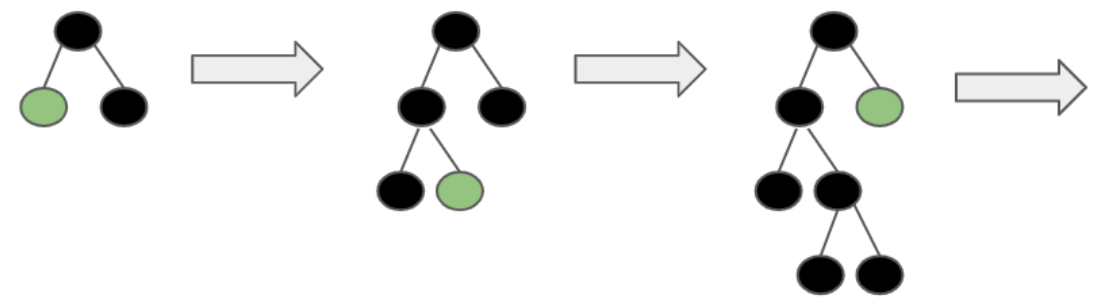
로지스틱 회귀



LightGBM

양상블 학습-부스팅-그래디언트 부스트-LGBM

LightGBM leaf-wise



2.5 데이터 일차가공 및 학습/예측/평가

👉 데이터

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline

card_df = pd.read_csv('creditcard.csv')
card_df.head(3)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...

	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0

```
card_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Time        284807 non-null float64
1   V1          284807 non-null float64
2   V2          284807 non-null float64
3   V3          284807 non-null float64
4   V4          284807 non-null float64
5   V5          284807 non-null float64
6   V6          284807 non-null float64
7   V7          284807 non-null float64
8   V8          284807 non-null float64
9   V9          284807 non-null float64
10  V10         284807 non-null float64
11  V11         284807 non-null float64
12  V12         284807 non-null float64
13  V13         284807 non-null float64
14  V14         284807 non-null float64
15  V15         284807 non-null float64
16  V16         284807 non-null float64
17  V17         284807 non-null float64
18  V18         284807 non-null float64
19  V19         284807 non-null float64
20  V20         284807 non-null float64
21  V21         284807 non-null float64
22  V22         284807 non-null float64
23  V23         284807 non-null float64
24  V24         284807 non-null float64
25  V25         284807 non-null float64
26  V26         284807 non-null float64
27  V27         284807 non-null float64
28  V28         284807 non-null float64
29  Amount      284807 non-null float64
30  Class       284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

2.5 데이터 일차가공 및 학습/예측/평가

☞ 데이터 일차가공

```
from sklearn.model_selection import train_test_split
#time 칼럼 삭제하고 복사된 dataframe 반환
def get_preprocessed_df(df=None):
    df_copy=df.copy()
    df_copy.drop('Time',axis=1,inplace=True)
    return df_copy
```

```
def get_train_test_dataset(df=None):
    df_copy=get_preprocessed_df(df)
    X_features=df_copy.iloc[:, :-1]
    y_target=df_copy.iloc[:, -1]
    X_train, X_test, y_train, y_test=train_test_split(X_features, y_target, test_size=0.3, random_state=0, stratify=y_target)
    return X_train, X_test, y_train, y_test
```

```
X_train, X_test, y_train, y_test=get_train_test_dataset(card_df)
```

```
print('학습 데이터 레이블 값 비율')
print(y_train.value_counts()/y_train.shape[0]*100)
print('테스트 데이터 레이블 값 비율')
print(y_test.value_counts()/y_test.shape[0]*100)
```

```
학습 데이터 레이블 값 비율
0    99.827451
1     0.172549
Name: Class, dtype: float64
테스트 데이터 레이블 값 비율
0    99.826785
1     0.173215
Name: Class, dtype: float64
```

Stratify

분류에서 중요한 옵션

Default=None.

Target으로 지정해주면 class
비율을 train/test에
유지시켜줌(쏠림 방지)

2.5 데이터 일차가공 및 학습/예측/평가

☞ 학습/예측/평가

```
from sklearn.linear_model import LogisticRegression
```

```
lr_clf=LogisticRegression()  
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

오차행렬

```
[[85279 16]  
 [ 60 88]]
```

정확도:0.9991, 정밀도:0.8462, 재현율:0.5946, F1:0.6984, AUC:0.9601

```
from lightgbm import LGBMClassifier
```

```
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=False)  
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

오차행렬

```
[[85290 5]  
 [ 36 112]]
```

정확도:0.9995, 정밀도:0.9573, 재현율:0.7568, F1:0.8453, AUC:0.9790

```
from lightgbm import LGBMClassifier
```

```
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=True)  
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

오차행렬

```
[[85224 71]  
 [ 83 65]]
```

정확도:0.9982, 정밀도:0.4779, 재현율:0.4392, F1:0.4577, AUC:0.7225

Boost_from_average

Default=True

처음 가중치 업데이트 할 때, 그 값이 평균을 향하도록 하는 것.
속도를 빠르게 함.

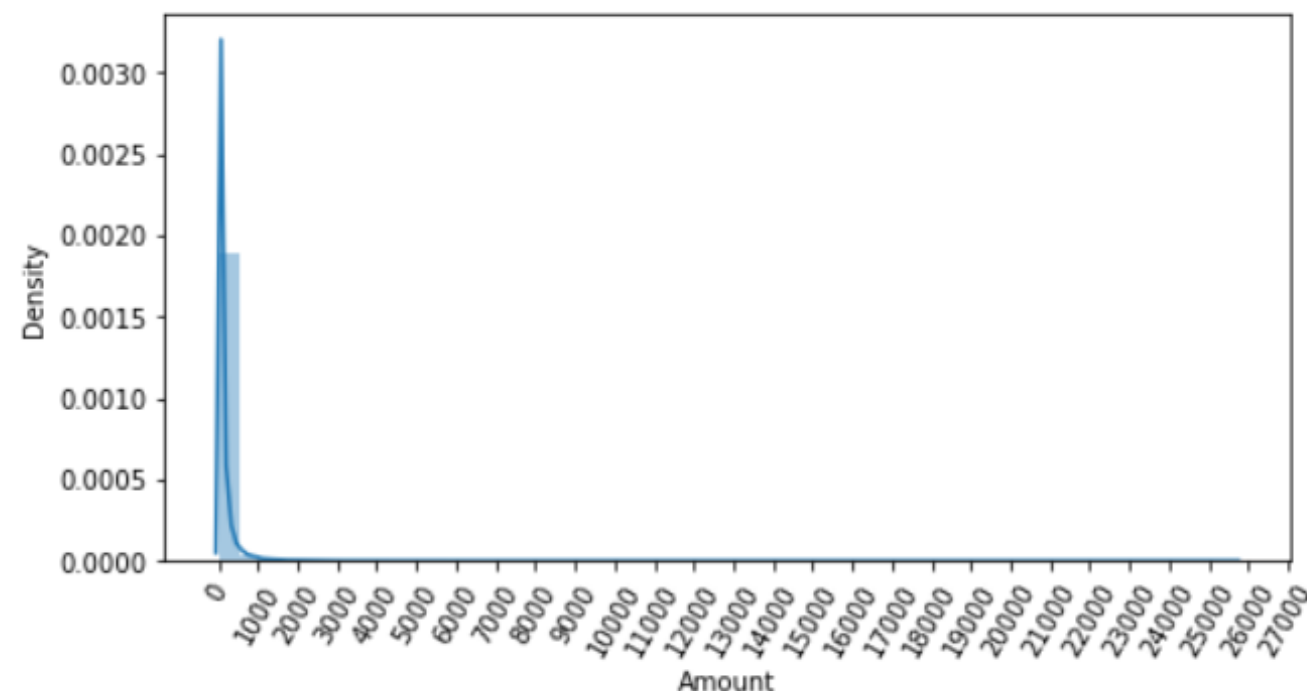
불균형한 데이터 세트에서 처음 학습율이 평균이 되도록 하면 예측
성능이 매우 저조해지므로 False로 설정해야 함

2.6 데이터 분포도 변환 후 및 학습/예측/평가

데이터 분포도

```
import seaborn as sns
plt.figure(figsize=(8,4))
plt.xticks(range(0,30000,1000), rotation=60)
sns.distplot(card_df['Amount'])
```

<AxesSubplot:xlabel='Amount', ylabel='Density'>



로지스틱 회귀는 선형 모델로, **중요 피쳐들의 값이 정규분포** 선호

여기서 피쳐 **Amount**(신용카드사용금액)으로 정상/사기 결정에 중요!

이때, Amount는 데이터 분포도가 심하게 왜곡되어 있으므로 스케일링/정규화 작업을 해야함

StandardScaler, MinMaxScaler, **log변환**

2.6 데이터 분포도 변환 후 및 학습/예측/평가

변환 후 및 학습/예측/평가

표준 정규 분포로 변환

```
from sklearn.preprocessing import StandardScaler
def get_processed_df(df=None):
    df_copy=df.copy()
    scaler=StandardScaler()
    amount_n=scaler.fit_transform(df_copy['Amount'].values.reshape(-1,1))
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)
    return df_copy
```

```
X_train, X_test, y_train, y_test=get_train_test_dataset(card_df)
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf=LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
print('### LigtGBM 예측 성능 ###')
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

로지스틱 회귀 예측 성능

오차행렬

```
[[85279    16]
 [    60    88]]
```

정확도:0.9991, 정밀도:0.8462, 재현율:0.5946, F1:0.6984, AUC:0.9601

LigtGBM 예측 성능

오차행렬

```
[[85224    71]
 [    83    65]]
```

정확도:0.9982, 정밀도:0.4779, 재현율:0.4392, F1:0.4577, AUC:0.7225

로그 변환

```
def get_preprocessed_df(df=None):
    df_copy=df.copy()
    amount_n=np.log1p(df_copy['Amount'])
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)
    return df_copy
```

```
X_train, X_test, y_train, y_test=get_train_test_dataset(card_df)
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf=LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
print('### LigtGBM 예측 성능 ###')
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

로지스틱 회귀 예측 성능

오차행렬

```
[[85283    12]
 [    59    89]]
```

정확도:0.9992, 정밀도:0.8812, 재현율:0.6014, F1:0.7149, AUC:0.9727

LigtGBM 예측 성능

오차행렬

```
[[85238    57]
 [    77    71]]
```

정확도:0.9984, 정밀도:0.5547, 재현율:0.4797, F1:0.5145, AUC:0.7395

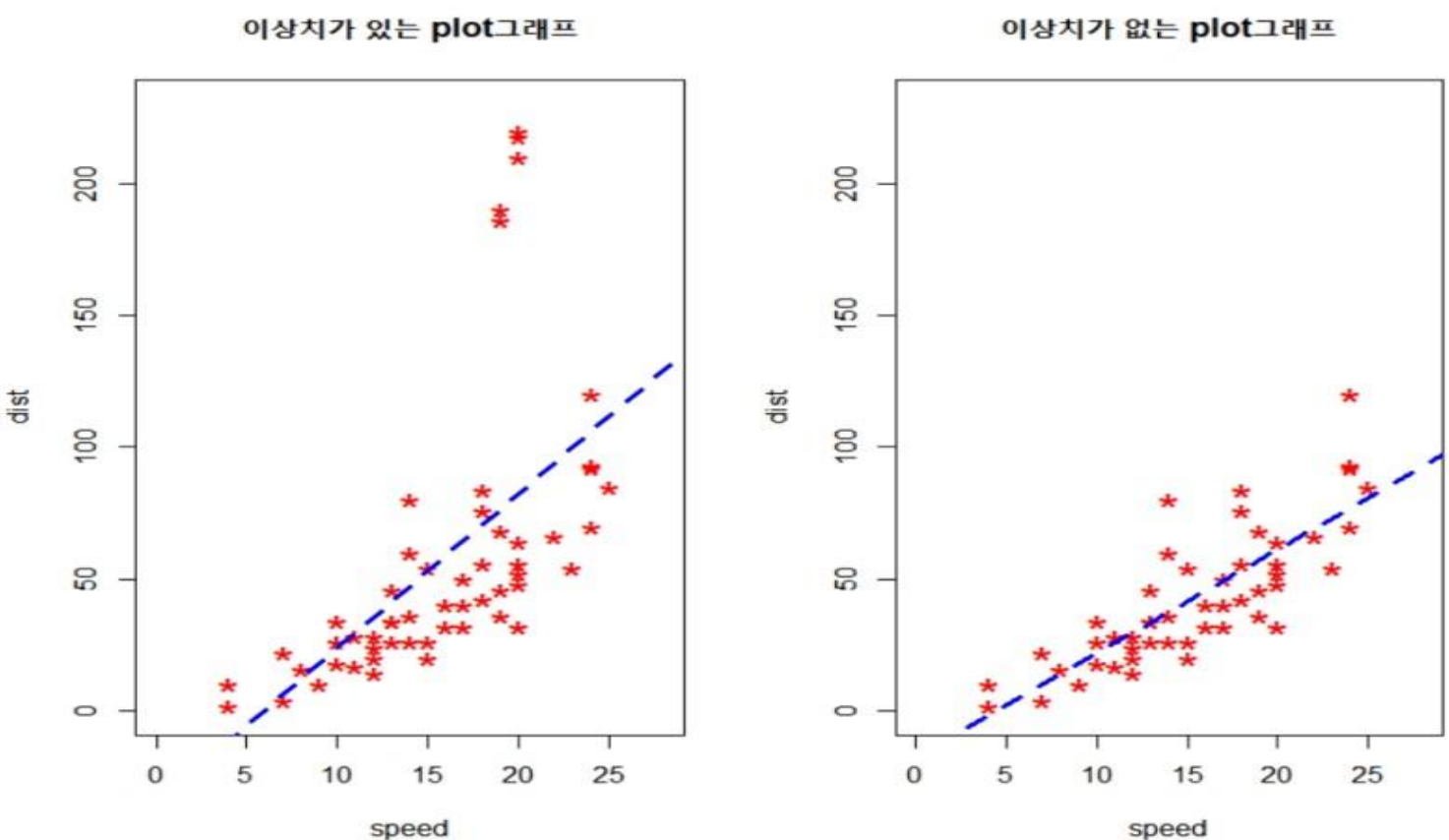
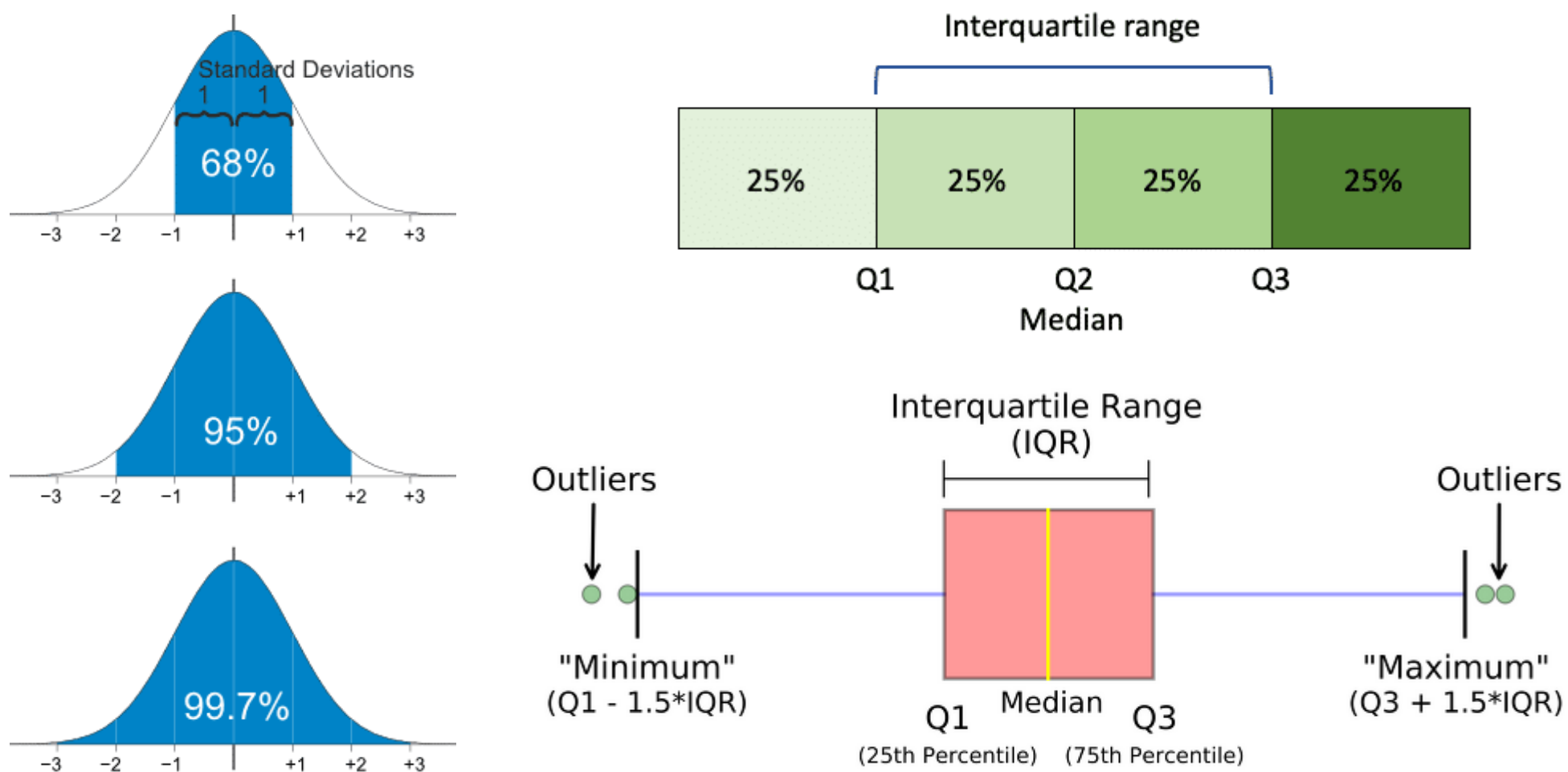
로그 변환이 예측 성능 더 좋음

2.7 이상치 데이터 제거 후 학습/예측/평가

☞ 이상치 데이터와 IQR

이상치(outlier) : 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터
이상치가 의사결정에 큰 영향을 줄 수 있으므로 이상치 처리 필수적

이상치 탐지: 정규분포, IQR(Inter Quantile Range)



크기순으로 정렬한 데이터를 4분할 하고, Q1~Q3 범위를 IQR

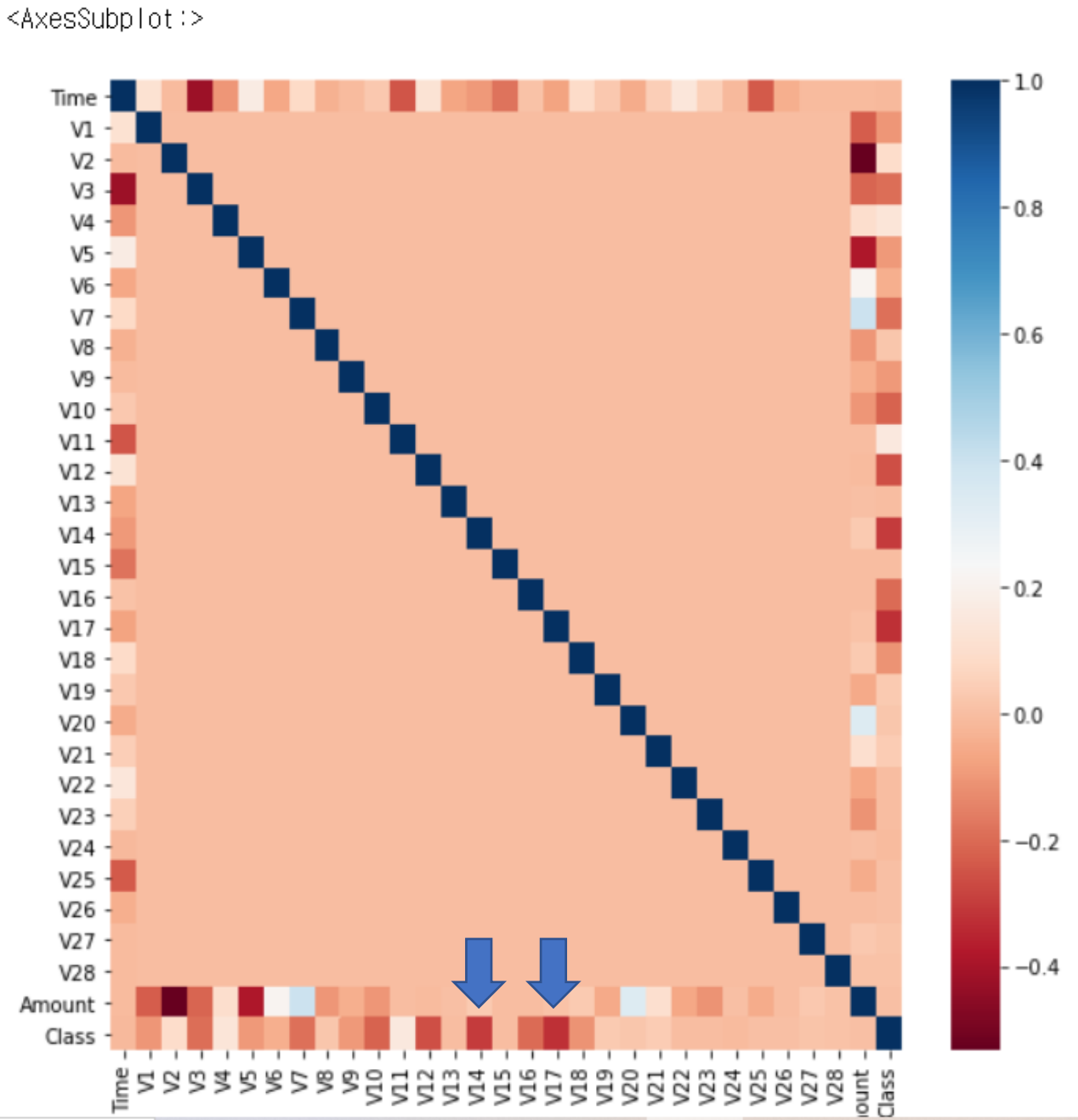
- 중앙값은 Q2
- 최댓값은 $Q3 + 1.5 * IQR$
- 최솟값은 $Q1 - 1.5 * IQR$

이상치는 최댓값 이상, 최솟값 이하 지점

2.7 이상치 데이터 제거 후 학습/예측/평가

피쳐별 상관도

```
plt.figure(figsize=(9,9))
corr=card_df.corr()
sns.heatmap(corr,cmap='RdBu')
```



각 피쳐별로 상관도를 구한 뒤 heatmap cmap='RdBu'로 시각화하여 레이블(class)와 가장 상관성 높은 피쳐들의 이상치를 검출해야 함

상관계수는 -1에서 1의 값을 가지고,
0.7보다 클 경우 강한 양의 상관관계가, -0.7보다 작을 경우 강한 음의 상관관계

2.7 이상치 데이터 제거 후 학습/예측/평가

☞ 이상치 데이터 제거 후 학습/예측/평가

```
def get_outlier(df=None, column=None, weight=1.5):
    fraud=df[df['Class']==1][column]
    quantile_25=np.percentile(fraud.values,25)
    quantile_75=np.percentile(fraud.values,75)
    iqr=quantile_75-quantile_25
    iqr_weight=iqr*weight
    lowest_val=quantile_25-iqr_weight
    highest_val=quantile_75+iqr_weight
    outlier_index=fraud[(fraud<lowest_val)|(fraud>highest_val)].index
    return outlier_index
```

```
outlier_index=get_outlier(df=card_df, column='V14', weight=1.5)
print('이상치 데이터 인덱스:',outlier_index)
```

이상치 데이터 인덱스: Int64Index([8296, 8615, 9035, 9252], dtype='int64')

```
outlier_index=get_outlier(df=card_df, column='V17', weight=1.5)
print('이상치 데이터 인덱스:',outlier_index)
```

이상치 데이터 인덱스: Int64Index([], dtype='int64')

```
def get_processed_df(df=None):
    df_copy=df.copy()
    amount_n=np.log1p(df_copy['Amount'])
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)
    #이상치 데이터 삭제 로직 추가
    outlier_index=get_outlier(df=df_copy, column='V14', weight=1.5)
    df_copy.drop(outlier_index, axis=0, inplace=True)
    return df_copy
```

```
X_train, X_test, y_train, y_test=get_train_test_dataset(card_df)
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf=LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
print('### LigtGBM 예측 성능 ###')
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

로지스틱 회귀 예측 성능

오차행렬

[[85283 12]
[59 89]]

정확도:0.9992, 정밀도:0.8812, 재현율:0.6014, F1:0.7149, AUC:0.9727

LigtGBM 예측 성능

오차행렬

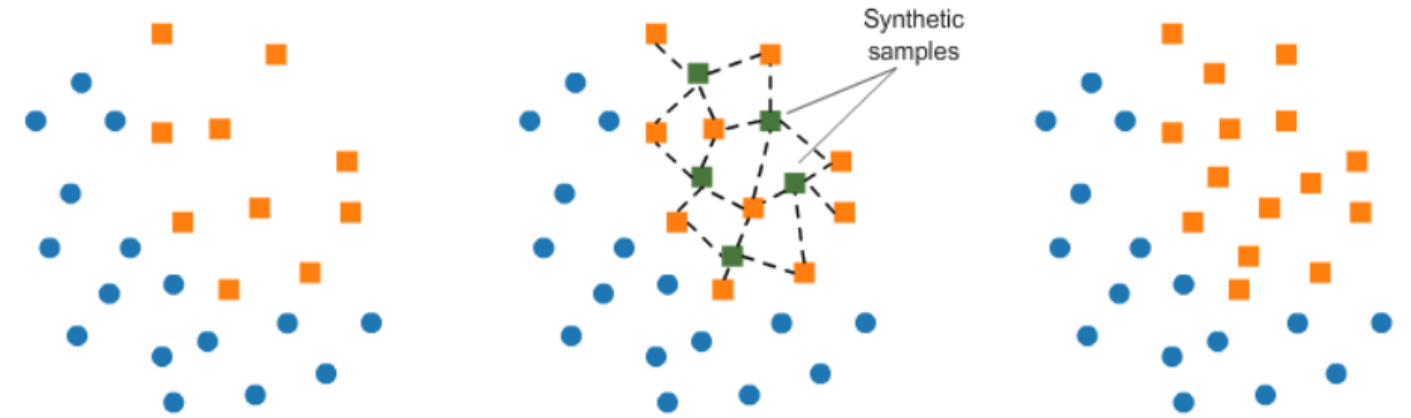
[[85238 57]
[77 71]]

정확도:0.9984, 정밀도:0.5547, 재현율:0.4797, F1:0.5145, AUC:0.7395

이상치 제거 예측 성능 더 좋음

2.8 오버 샘플링 적용 후 모델 학습/예측/평가

☞ SMOTE(Synthetic Minority Over-sampling Technique)



KNN(k 최근접 이웃 k nearest neighbor)

Minor 데이터 샘플과 이들 k개 이웃 간의 차(difference)를 구하고, 이 차이에 0 ~ 1 사이의 임의의 값을 곱하여 원래 샘플에 더하여 추가
결과적으로 SMOTE는 기존의 샘플을 주변의 이웃을 고려해 약간씩 이동시킨 점들을 추가하는 방식으로 동작

학습 데이터 세트만 오버샘플링 해야함

검증, 테스트 데이터 세트를 오버 샘플링하면, 원본이 아니기 때문에 올바른 검증, 테스트 불가

2.8 오버 샘플링 적용 후 모델 학습/예측/평가

SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

```
from imblearn.over_sampling import SMOTE
```

```
smote=SMOTE(random_state=0)
X_train_over, y_train_over=smote.fit_sample(X_train, y_train)
print('SMOTE 적용 전 학습용 피쳐/레이블 데이터 세트: ', X_train.shape, y_train.shape)
print('SMOTE 적용 후 학습용 피쳐/레이블 데이터 세트: ', X_train_over.shape, y_train_over.shape)
print('SMOTE 적용 후 레이블 값 분포: \n', pd.Series(y_train_over).value_counts())
```

SMOTE 적용 전 학습용 피쳐/레이블 데이터 세트: (199364, 29) (199364,)

SMOTE 적용 후 학습용 피쳐/레이블 데이터 세트: (398040, 29) (398040,)

SMOTE 적용 후 레이블 값 분포:

0 199020

1 199020

Name: Class, dtype: int64

```
print('### 로지스틱 회귀 예측 성능 ###')
```

```
lr_clf=LogisticRegression()
```

```
get_model_train_eval(lr_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over, tgt_test=y_test)
```

```
print('### LigtGBM 예측 성능 ###')
```

```
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
```

```
get_model_train_eval(lgbm_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over, tgt_test=y_test)
```

로지스틱 회귀 예측 성능

오차행렬

[[83317 1978]

[15 133]]

정확도:0.9767, 정밀도:0.0630, 재현율:0.8986, F1:0.1178, AUC:0.9803

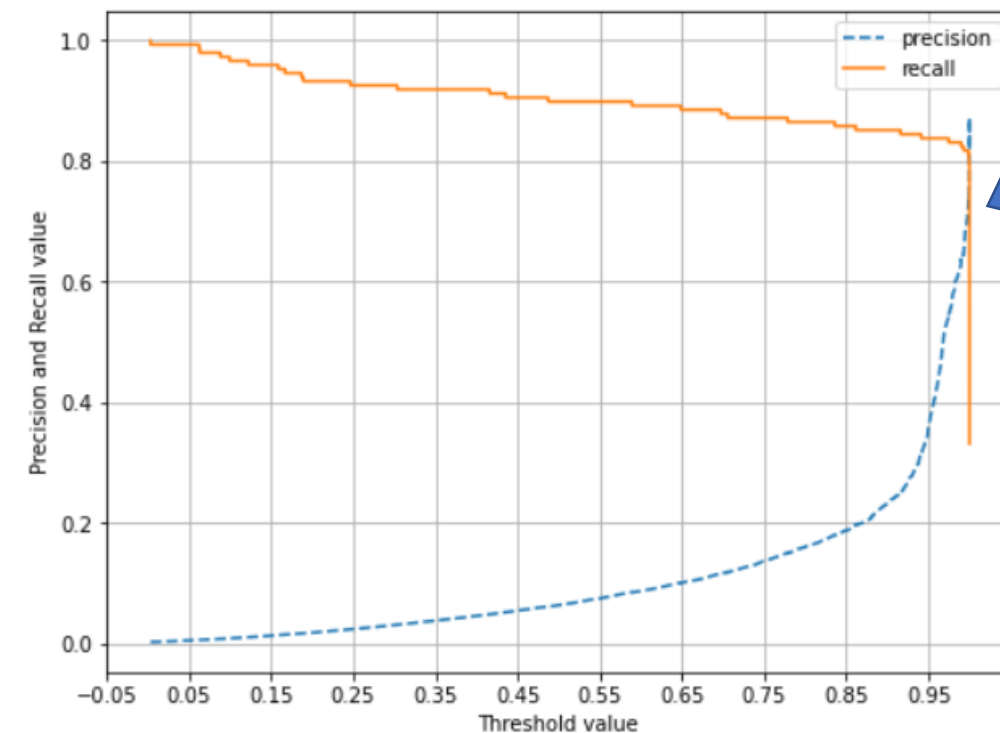
LigtGBM 예측 성능

오차행렬

[[85277 18]

[36 112]]

정확도:0.9994, 정밀도:0.8615, 재현율:0.7568, F1:0.8058, AUC:0.9780



임계값 0.99 부근에서 재현율과 정밀도가 극단적으로 변화함

임계값의 민감도가 심해 올바른 재현율/정밀도 성능을 얻을 수 없음

재현율 높고, 정밀도 낮아짐
로지스틱 회귀보다는 높은 성능

2.8 오버 샘플링 적용 후 모델 학습/예측/평가

Borderline-SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가 두 클래스 간의 결정 경계를 따라 합성 데이터 생성

```
X_train, X_test, y_train, y_test=get_train_test_dataset(card_df)
```

```
from imblearn.over_sampling import BorderlineSMOTE  
bsmote = BorderlineSMOTE(random_state = 101, kind = 'borderline-1')  
X_train_over_borderline, y_train_over_borderline = bsmote.fit_resample(X_train, y_train)
```

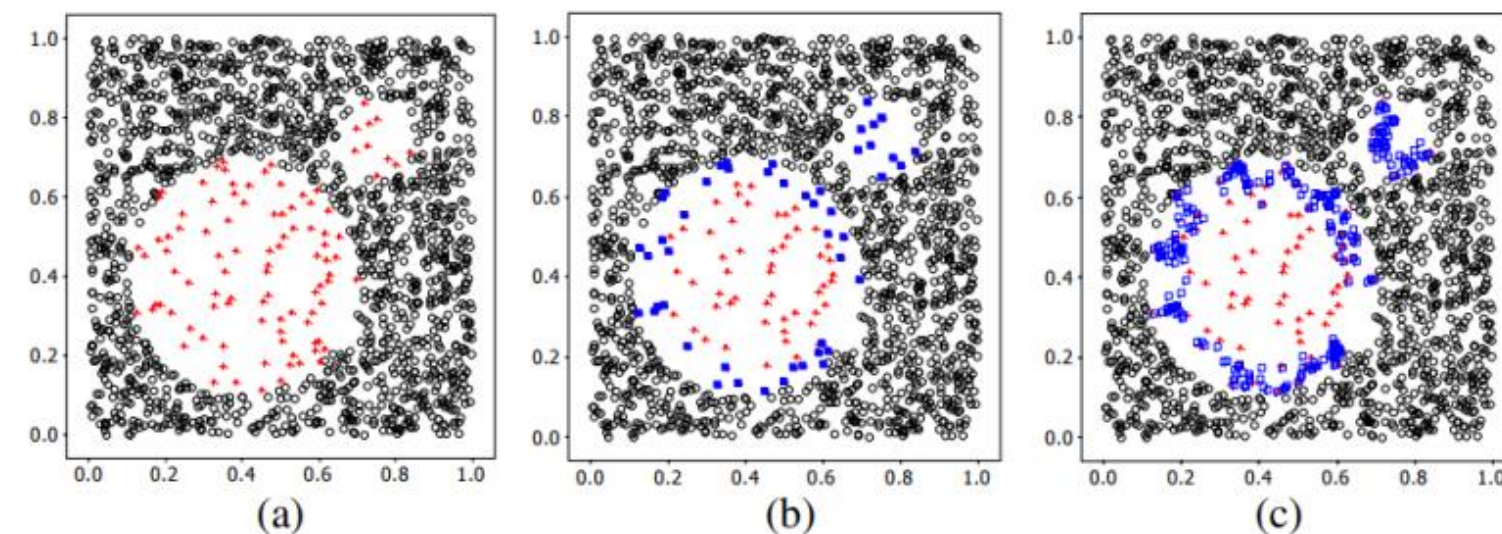
```
print('### 로지스틱 회귀 예측 성능 ###')  
lr_clf=LogisticRegression()  
get_model_train_eval(lr_clf, ftr_train=X_train_over_borderline, ftr_test=X_test, tgt_train=y_train_over_borderline, tgt_test=y_test)  
print('### LightGBM 예측 성능 ###')  
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)  
get_model_train_eval(lgbm_clf, ftr_train=X_train_over_borderline, ftr_test=X_test, tgt_train=y_train_over_borderline, tgt_test=y_test)
```

```
### 로지스틱 회귀 예측 성능 ###  
오차행렬  
[[84623  672]  
 [  22  126]]  
정확도:0.9919, 정밀도:0.1579, 재현율:0.8514, F1:0.2664, AUC:0.9501  
### LightGBM 예측 성능 ###  
오차행렬  
[[85286    9]  
 [   36  112]]  
정확도:0.9995, 정밀도:0.9256, 재현율:0.7568, F1:0.8327, AUC:0.9812
```

Borderline-SMOTE1은 다수 클래스

```
### 로지스틱 회귀 예측 성능 ###  
오차행렬  
[[84087 1208]  
 [   18  130]]  
정확도:0.9857, 정밀도:0.0972, 재현율:0.8784, F1:0.1750, AUC:0.9577  
### LightGBM 예측 성능 ###  
오차행렬  
[[85287    8]  
 [   33  115]]  
정확도:0.9995, 정밀도:0.9350, 재현율:0.7770, F1:0.8487, AUC:0.9846
```

Borderline-SMOTE2는 소수 클래스



학습 데이터 세트만 오버샘플링 해야함/ SMOTE에 비해 정밀도 높아짐

2.9 데이터 가공 정리

결론

데이터 가공 유형	머신러닝 알고리즘	평가지표		
		정밀도	재현율	ROC-AUC
원본 데이터 가공 없음	로지스틱 회귀	0.8738	0.6081	0.9707
	LightGBM	0.9492	0.7568	0.9797
데이터 로그 변환	로지스틱 회귀	0.8824	0.6081	0.9721
	LightGBM	0.9575	0.7635	0.9786
이상치 데이터 제거	로지스틱 회귀	0.8829	0.6712	0.9747
	LightGBM	0.9690	0.8288	0.9831
SMOTE 오버 샘플링	로지스틱 회귀	0.0540	0.9247	0.9737
	LightGBM	0.9323	0.8493	0.9789

데이터 불균형의 문제를 해결하기 위해 3가지의 방법으로 데이터 가공을 했음

(0. 데이터 일차 가공=시간 삭제)

1. 데이터 변환=표준정규분포, 로그

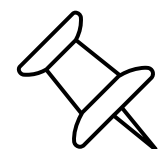
2. 이상치 데이터 제거=IQR,

3. 오버샘플링=SMOTE K최근접이웃, Borderline-SMOTE

03. 캐글 심장병 발병 예측



3.1 대회 소개



Heart Failure Prediction Dataset

Heart Failure Prediction Dataset

11 clinical features for predicting heart disease events.



Data Code (513) Discussion (16) Metadata

About Dataset

Similar Datasets

- Hepatitis C Dataset: [LINK](#)
- Body Fat Prediction Dataset: [LINK](#)
- Cirrhosis Prediction Dataset: [LINK](#)
- Stroke Prediction Dataset: [LINK](#)
- Stellar Classification Dataset - SDSS17: [LINK](#)

Context

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Four out of 5 CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Usability ⓘ

10.00

License

[Database: Open Database, Cont...](#)

Expected update frequency

Never

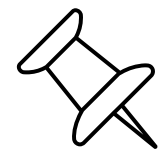
- 심혈관질환(CVD) 전 세계 사망 원인 1위
- 일반적으로 심부전(Heart Failure)은 CVD로 인해 발생
- 심장 질환 예측에 사용할 수 있는 11가지 위험 요소 분석
- 새로운 환자의 심장병 유/무 예측
- CVD 있거나, 위험이 높은 사람 조기 발견 및 관리 가능해짐

3.2 Data Description

Heart Failure Prediction Dataset

칼럼명		
Age	age of the patient	years
Sex	sex of the patient	M: Male, F: Female
ChestPainType(흉통)	chest pain type	TA(전형적 가슴통증), ATA(비전형적 가슴 통증), NAP(비 심장성 흉통), ASY(무증상)
RestingBP(혈압)	resting blood pressure	mm Hg
Cholesterol(혈청 콜레스테롤)	serum cholesterol	mm/dl
FastingBS(공복혈당)	fasting blood sugar	1: if FastingBS>120mg/dl, 0: otherwise
RestingECG(심전도)	resting electrocardiogram results	- Normal: Normal - ST: having ST-T wave abnormality(T파 비정상) - LVH: left ventricular hypertrophy (좌심실 비대)
MaxHR(최대 심박수)	maximum heart rate achieved	Numeric value between 60 and 202
ExerciseAngina (운동유도협심증)	exercise-induced angina	Y: Yes, N: No
Oldpeak	oldpeak = ST	Numeric value measured in depression
ST_Slope	the slope of the peak exercise ST segment	Up: upsloping, Flat: flat, Down: downsloping
HeartDisease	output class	1: heart disease, 0: Normal

3.3 Notebook 소개



Beginner Friendly CATBOOST with OPTUNA

1. EDA

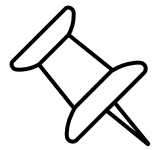
- Target Variable
- 수치형 변수
- 범주형 변수

2. Model Selection

- Baseline Model
- Logistic & Linear Discriminant & SVC & KNN(without, with scaler)
- Ensemble Models (AdaBoost & Gradient Boosting & Random Forest & Extra Trees)
- Famous Trio (XGBoost & LightGBM & Catboost)
- CATBOOST
- Catboost HyperParameter Tuning with OPTUNA
- Feature Importance
- Model Comparison

3. Conclusion

3.3 Notebook 소개



문제 정의 및 Metric

- 분류 문제(심장병 발병 유무 예측)
- 'HeartDisease' 를 타겟변수로 하는 분류 문제
- 새로운 환자의 데이터가 들어왔을 때 해당 환자가 심장병 있을지 예측하는 모델
- 우선 타겟변수의 balance 살펴볼 것 → balanced data
- Accuracy score 사용

3.4 EDA(Exploratory Data Analysis)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Age              918 non-null   int64
1   Sex              918 non-null   object
2   ChestPainType    918 non-null   object
3   RestingBP        918 non-null   int64
4   Cholesterol       918 non-null   int64
5   FastingBS        918 non-null   int64
6   RestingECG       918 non-null   object
7   MaxHR            918 non-null   int64
8   ExerciseAngina    918 non-null   object
9   Oldpeak          918 non-null   float64
10  ST_Slope         918 non-null   object
11  HeartDisease      918 non-null   int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

→ 전체적인 데이터 타입 관촬아 보임!
+ 모든 컬럼 결측치 x

```
df.duplicated().sum()
```

```
0
```

```
def missing(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_N
umber', 'Missing_Percent'])
    return missing_values

missing(df)
```

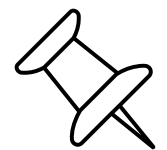
```
]:
```

	Missing_Number	Missing_Percent
Age	0	0.0
Sex	0	0.0
ChestPainType	0	0.0
RestingBP	0	0.0
Cholesterol	0	0.0
FastingBS	0	0.0
RestingECG	0	0.0
MaxHR	0	0.0
ExerciseAngina	0	0.0
Oldpeak	0	0.0
ST_Slope	0	0.0
HeartDisease	0	0.0

duplicated(): row마다의 중복값
검사해주는 함수 → 중복 없음

→ 결측치 없음

3.4 EDA



수치형 변수 / 범주형 변수 구분

```
numerical= df.drop(['HeartDisease'], axis=1).select_dtypes('number').columns

categorical = df.select_dtypes('object').columns

print(f'Numerical Columns: {df[numerical].columns}')
print('\n')
print(f'Categorical Columns: {df[categorical].columns}')
```

Numerical Columns: Index(['Age', 'RestingBP', 'Cholesterol', 'FastingBS',
'MaxHR', 'Oldpeak'], dtype='object')

Categorical Columns: Index(['Sex', 'ChestPainType', 'RestingECG', 'ExerciseA
ngina', 'ST_Slope'], dtype='object')

<수치형 변수 / 범주형 변수 구분>

select_dtypes() - object형 데이터와 nonobject형(숫자형) 데이터
구분해서 호출해주는 함수

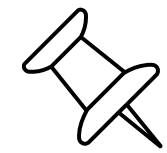
<각 범주형 변수의 고유 값 개수 확인>

```
: df[categorical].nunique()
:
Sex                2
ChestPainType      4
RestingECG         3
ExerciseAngina     2
ST_Slope           3
dtype: int64
```

nunique() - 변수의 고유 값 개수 출력 함수

→ 각 카테고리의 (고유)데이터 개수,
너무 많지도 않고 적절해 보임

3.4 EDA – Target 변수



Target 변수: HeartDisease

〈HeartDisease 변수의 데이터 비율 확인〉

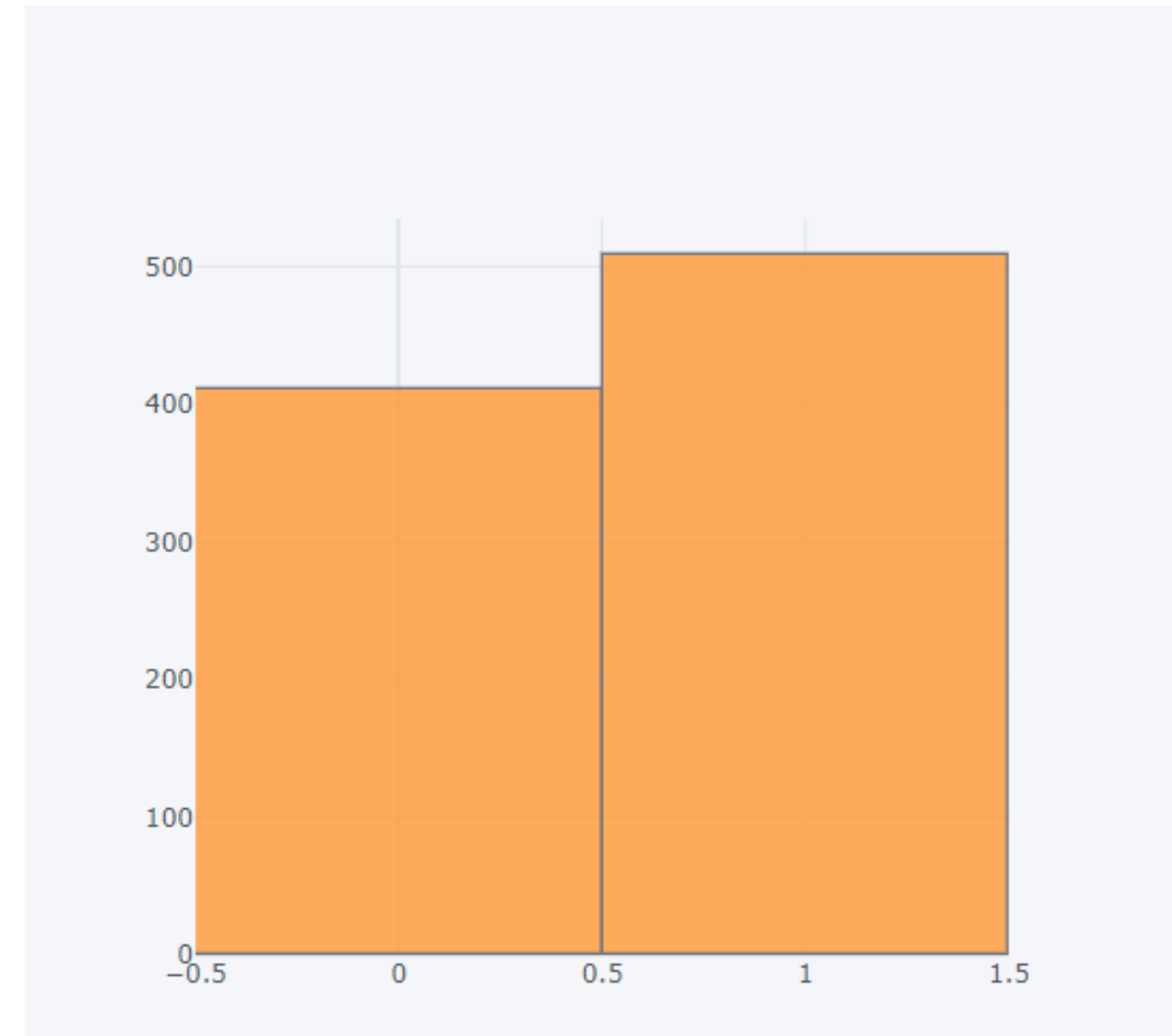
```
y = df['HeartDisease']
print(f'Percentage of patient had a HeartDisease: {round(y.value_counts(normalize=True)[1]*100,2)} % --> ({y.value_counts()[1]} patient)\nPercentage of patient did not have a HeartDisease: {round(y.value_counts(normalize=True)[0]*100,2)} % --> ({y.value_counts()[0]} patient)')
```

Percentage of patient had a HeartDisease: 55.34 % --> (508 patient)

Percentage of patient did not have a HeartDisease: 44.66 % --> (410 patient)

- 55%의 환자 심장병 있음
- 508명의 환자 심장병 있음
- 45%의 환자 심장병 없음
- 410명의 환자 심장병 없음

```
df['HeartDisease'].plot(kind='hist')
```

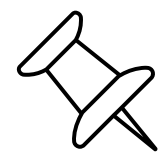


- 약간의 imbalance, 그러나 문제될 정도는 아님
- Accuracy로 평가 가능한 데이터 셋

*파이썬 시각화 라이브러리 Plotly. 쉽고 인터랙티브한 자료 만들 수 있어 유용.

<https://dailyheumsi.tistory.com/118>

3.4 EDA – 수치형 변수



수치형 변수

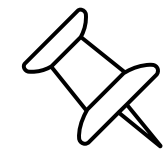
```
df[numerical].describe()
```

:

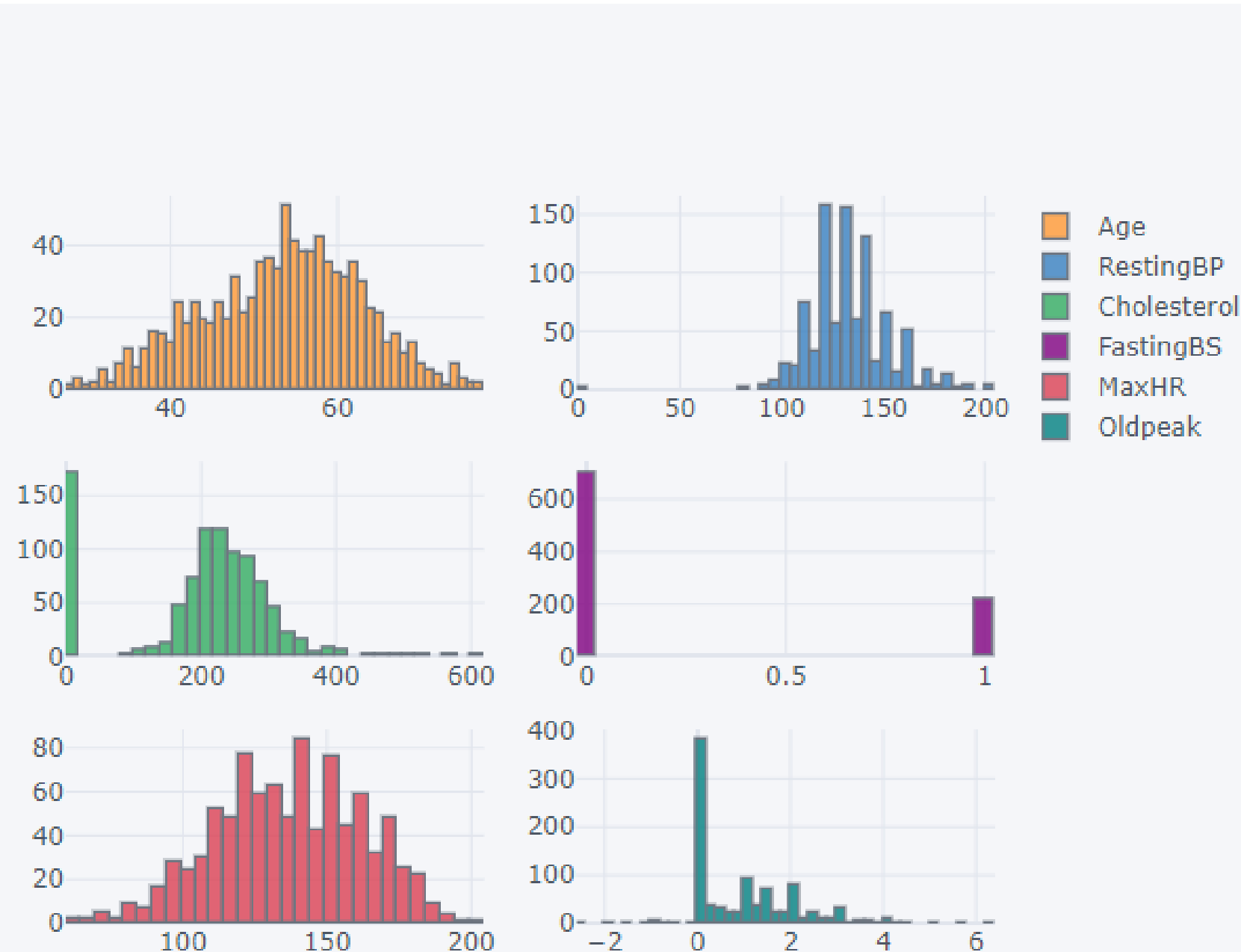
	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000

→ 수치형 변수들의 각종 통계량 요약해서 출력

3.4 EDA – 수치형 변수



수치형 변수 – 왜도(비대칭도) 측정



```
skew_limit = 0.75 # This is our threshold-limit to evaluate skewness. 임계값
```

```
skew_vals = df[numerical].drop('FastingBS', axis=1).skew() 왜도
```

```
# FastingBS는 0과 1로만 구성되어 제외
```

```
skew_cols= skew_vals[abs(skew_vals)> skew_limit].sort_values(ascending=False)
```

```
skew_cols
```

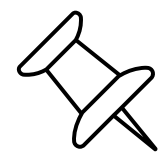
→ Oldpeak 1.022872

개별 변수의 왜도 절대값이 임계값보다 큰 지 확인했음

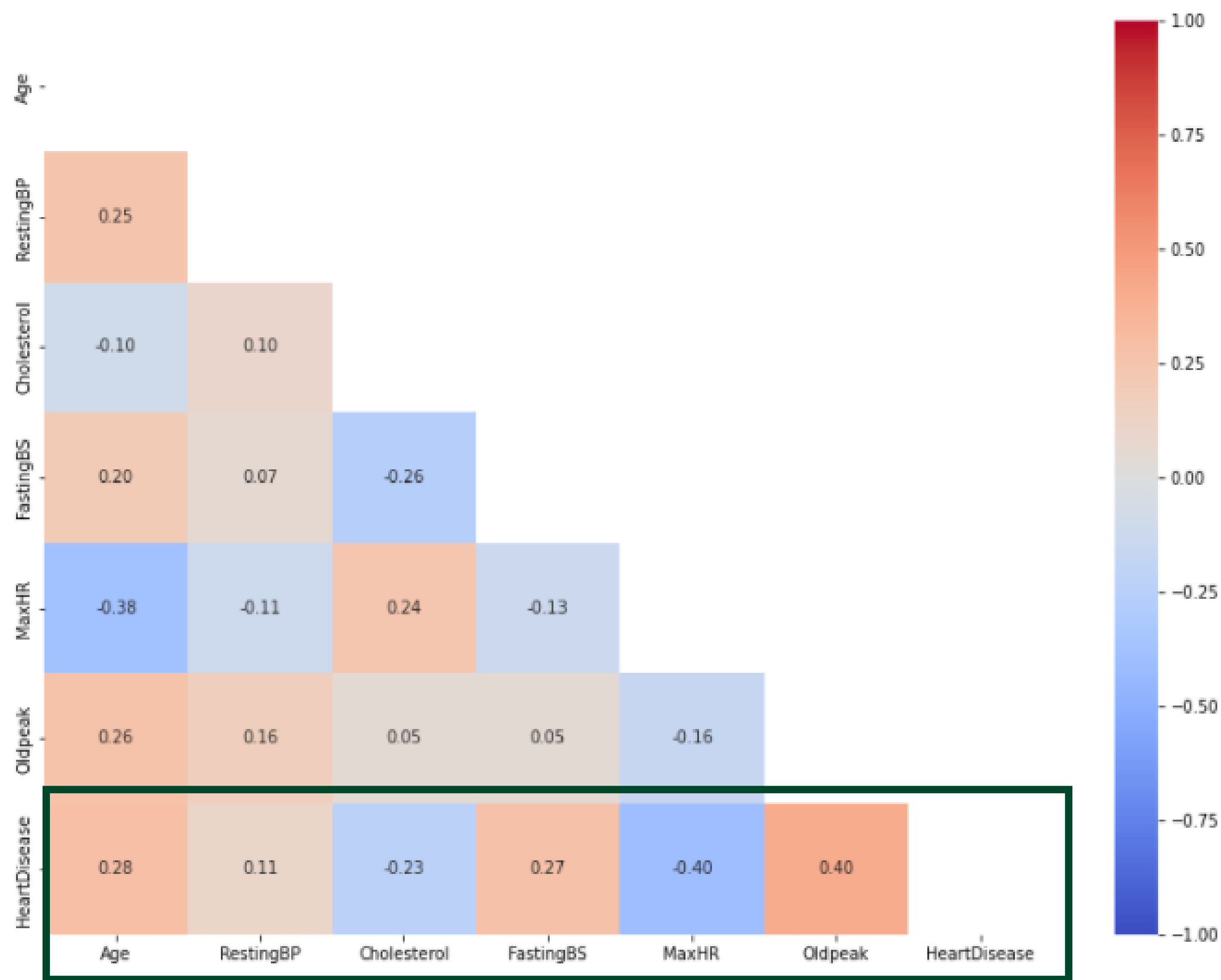
→ Oldpeak밖에 없음

➔ 왜도 측면에서 별 문제 없고, 정규분포와 유사함
(예측 변수와 목표 변수가 정규 분포를 따를 때
더 신뢰할 수 있는 예측 이뤄짐)

3.4 EDA – 수치형 변수

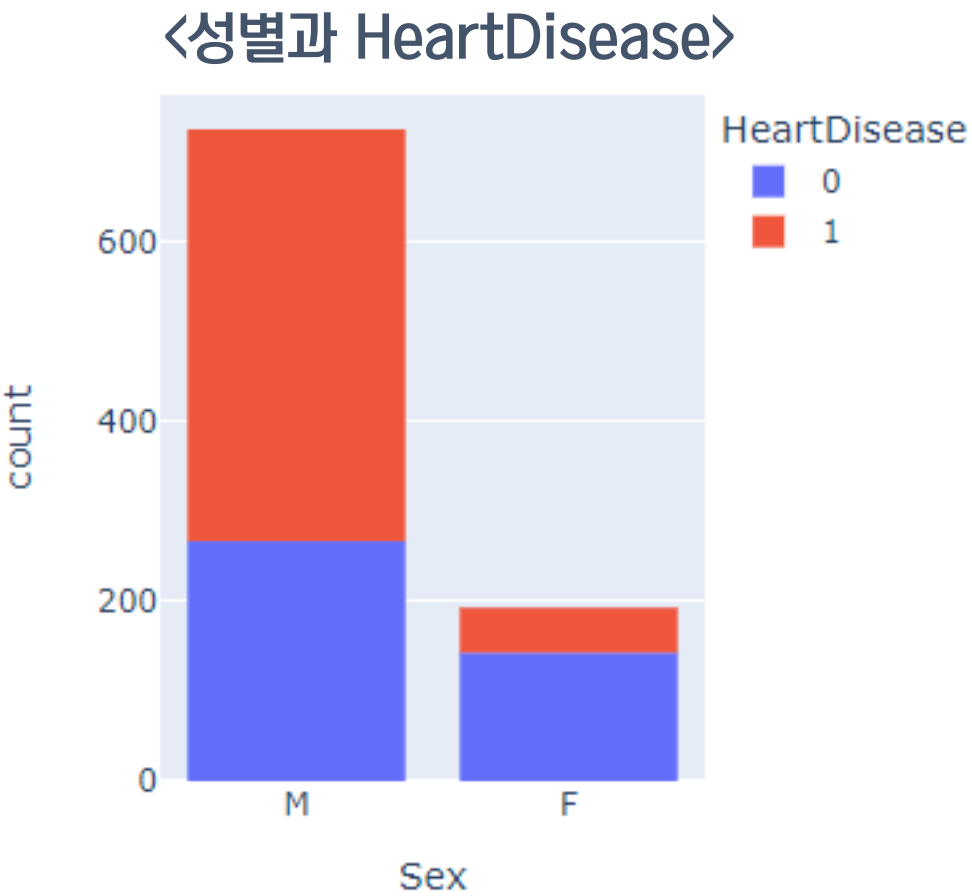


수치형 변수 – 상관관계

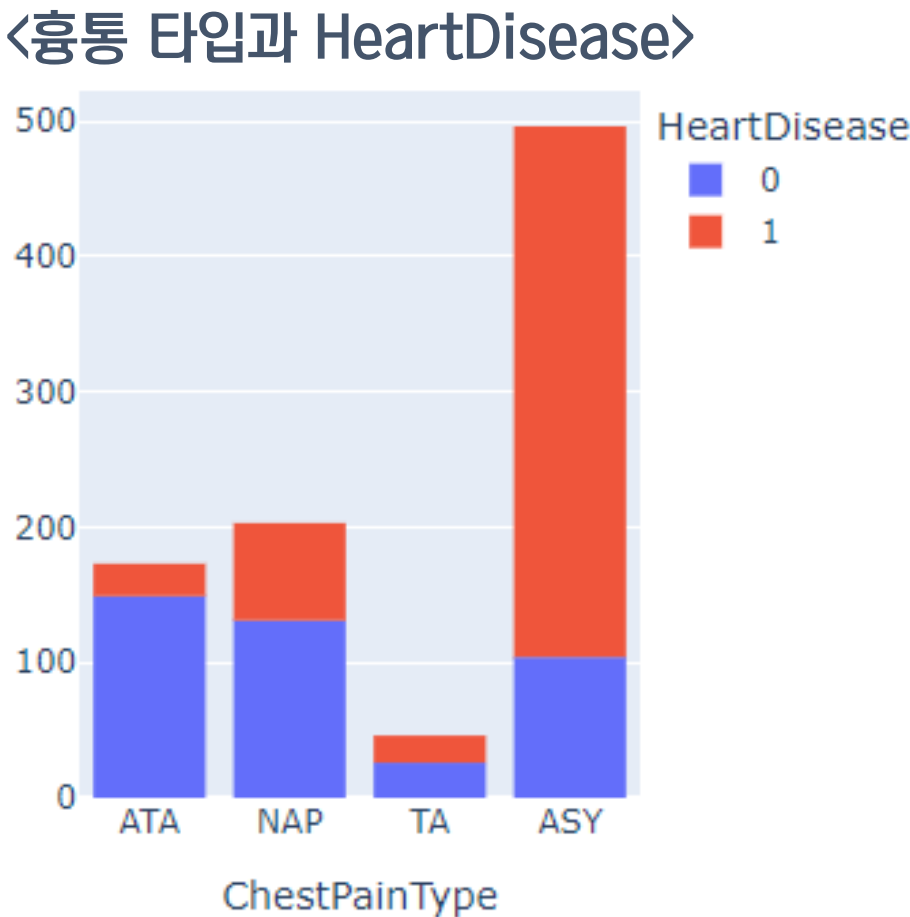


- 수치형 변수와 타겟 변수간의 약한 상관관계
- Oldpeak와 HeartDisease (+)상관관계(0.4)
- MaxHR과 HeartDisease (-)상관관계(-0.4)
- 의외로 콜레스테롤 HeartDisease와 (-)상관관계

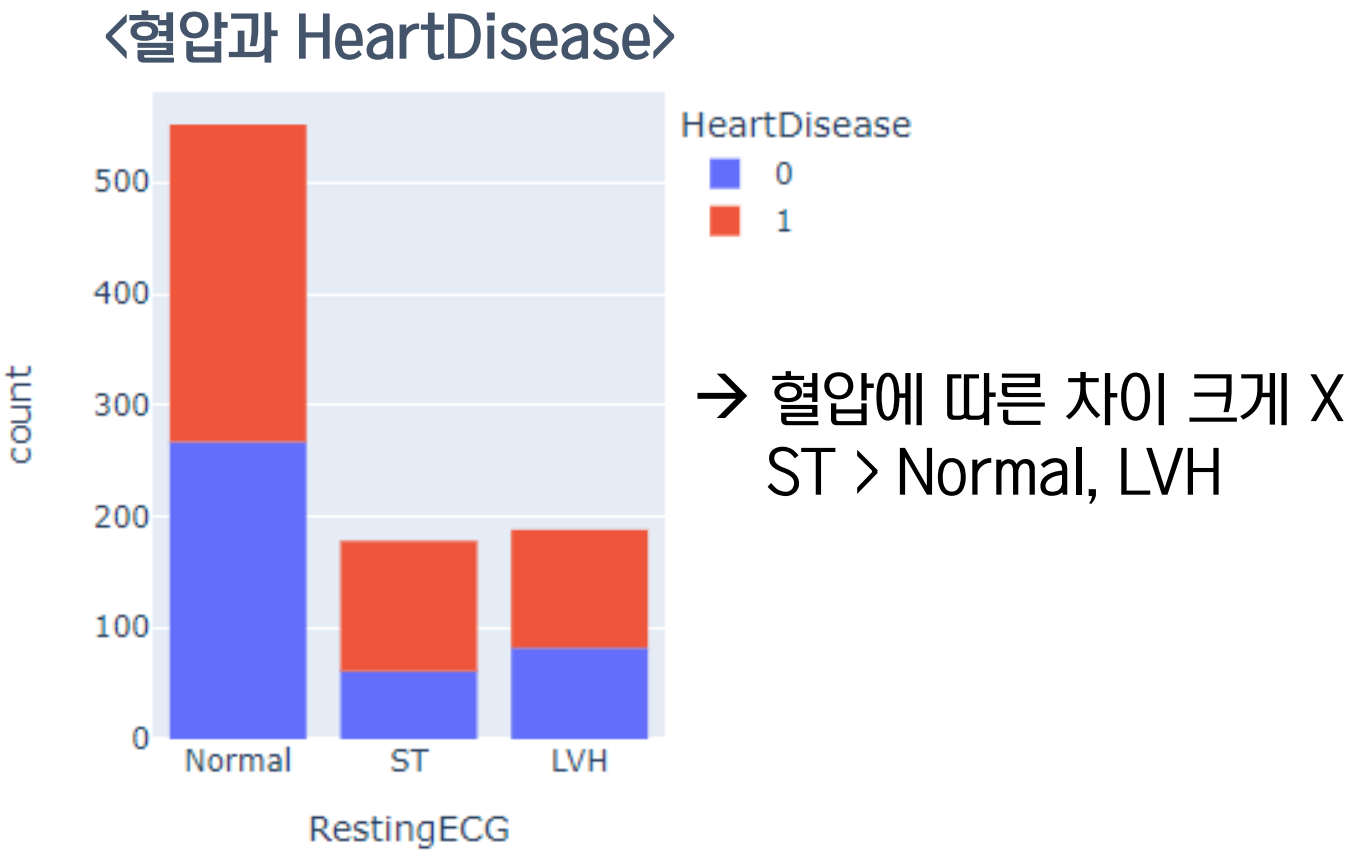
3.4 EDA – 범주형 변수



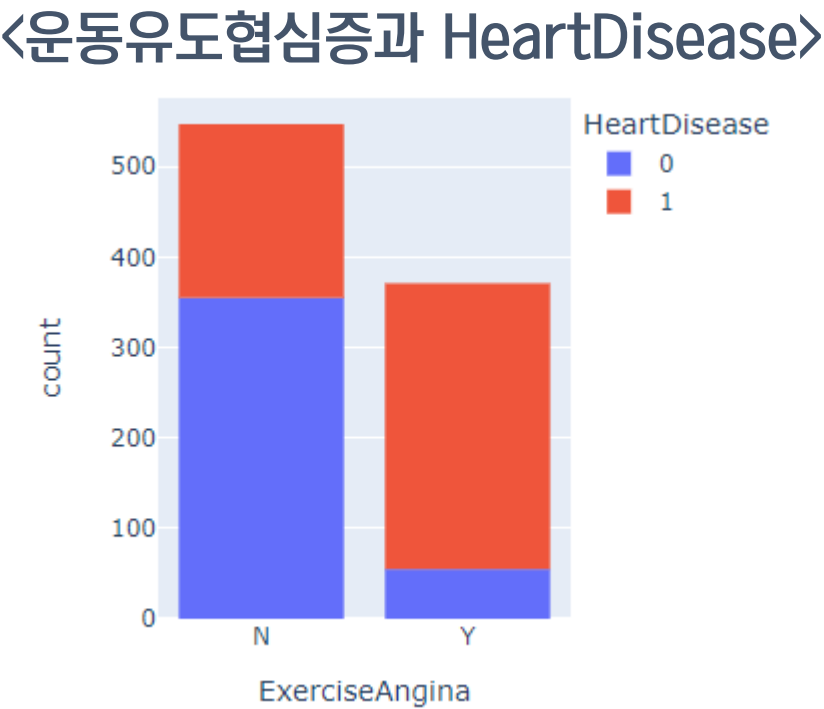
→ 남성 > 여성 (2.44배)



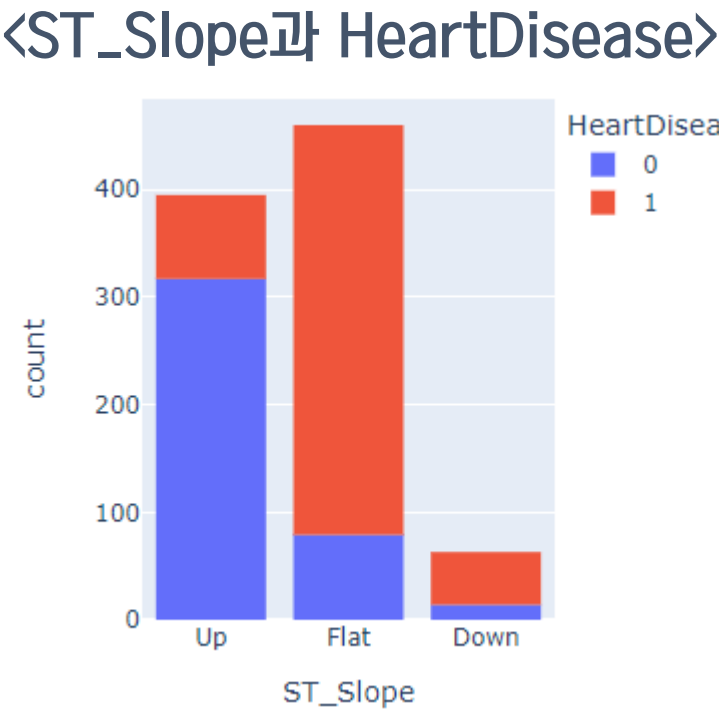
→ 흉통 종류에 따른 차이 분명
ASY > ATA (6배)



→ 혈압에 따른 차이 크게 X
ST > Normal, LVH

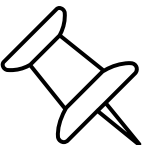


→ Y > N (2.4배)



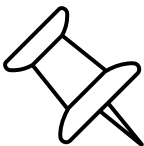
→ 기울기별 차이 존재, Up <<< Flat, down

3.4 EDA 결론



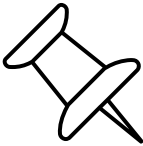
Target 변수

balanced data → 평가 지표로 accuracy 사용 가능



수치형 변수

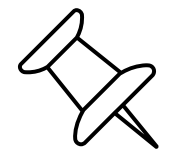
- 타겟 변수와의 약한 상관관계 보임
- Oldpeak (+) 상관관계
- MaxHR (-) 상관관계
- Cholesterol (-) 상관관계



범주형 변수

Sex	남성 > 여성 (2.44배)
ChestPainType(흉통)	분명한 차이 존재, ASY > ATA (6배)
RestingBP(혈압)	큰 차이 x, ST > Normal, LVH
ExerciseAngina(운동유도협심증)	Y > N (2.4배)
ST_Slope	차이 존재: Up <<< Flat, down

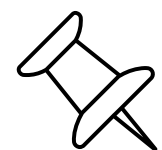
3.5 Model Selection



model selection 순서

- Baseline Model → dummy classifier 사용
- Logistic & Linear Discriminant & SVC & KNN(without, with scaler)
- Ensemble Models (AdaBoost & Gradient Boosting & Random Forest & Extra Trees)
- Famous Trio (XGBoost & LightGBM & Catboost)
- CATBOOST
- Catboost HyperParameter Tuning with OPTUNA
- Feature Importance
- Model Comparison

3.5 Model Selection – baseline model



Baseline Model : 모델 성능 비교의 기준이 되는 모델을 baseline model이라고 함
(즉, 베이스라인 모델은 모델 성능에 대한 최소 하한선 제공)

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ohe= OneHotEncoder()
ct= make_column_transformer((ohe,categorical),remainder='passthrough')

model = DummyClassifier(strategy='constant', constant=1) 항상 상수 1로 예측
pipe = make_pipeline(ct, model)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
accuracy.append(round(accuracy_score(y_test, y_pred),4))
print (f'model : {model} and accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')

model_names = ['DummyClassifier']
dummy_result_df = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
dummy_result_df
```

	Accuracy
DummyClassifier	0.5942

3.5 Model Selection – pipeline

pipeline

:전처리 단계, 모델 생성, 학습 등을 포함하는
여러 단계의 머신러닝 프로세스를 한 번에 처리할 수 있는 사이킷런 내장 라이브러리

```
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_transformer
```

ColumnTransformer - make_column_transformer()

:변수 특성에 따른 다른 종류의 전처리 가능하게 함.

```
ohe= OneHotEncoder()      (변환기, 변환기 적용될 변수) → 범주형 변수에 원-핫 인코딩 적용
ct= make_column_transformer((ohe, categorical), remainder='passthrough')
                                         작업할 때 걸리는 시간
```

```
model = DummyClassifier(strategy='constant', constant=1)
pipe = make_pipeline(ct, model) (변환기, 사용할 모델)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
accuracy.append(round(accuracy_score(y_test, y_pred),4))
print (f'model : {model} and accuracy score is : {round(accuracy_score(y_t
est, y_pred),4)}')
```

3.5 Model Selection

Logistic & Linear Discriminant & SVC & KNN

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ohe= OneHotEncoder()
ct= make_column_transformer((ohe, categorical), remainder='passthrough')

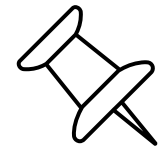
lr = LogisticRegression(solver='liblinear')
lda= LinearDiscriminantAnalysis()
svm = SVC(gamma='scale')
knn = KNeighborsClassifier()

models = [lr,lda,svm,knn]

for model in models:
    pipe = make_pipeline(ct, model)
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    accuracy.append(round(accuracy_score(y_test, y_pred),4))
    print (f'model : {model} and accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')
```

	Accuracy
Logistic	0.8841
LinearDiscriminant	0.8696
SVM	0.7246
KNeighbors	0.7174

3.5 Model Selection



Logistic & Linear Discriminant & SVC & KNN with Scaler

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ohe= OneHotEncoder()
s= StandardScaler()
ct1= make_column_transformer((ohe,categorical),(s,numerical))

lr = LogisticRegression(solver='liblinear')
lda= LinearDiscriminantAnalysis()
svm = SVC(gamma='scale')
knn = KNeighborsClassifier()

models = [lr,lda,svm,knn]

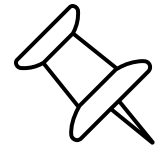
for model in models:
    pipe = make_pipeline(ct1, model)
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    accuracy.append(round(accuracy_score(y_test, y_pred),4))
    print(f'model : {model} and accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')
```

수치형 변수에 대한 스케일링 추가

	Accuracy
Logistic_scl	0.8804
LinearDiscriminant_scl	0.8696
SVM_scl	0.8841
KNeighbors_scl	0.8841

→ scaler 사용 후, KNN과 SVM 성능 향상

3.5 Model Selection - Ensemble Models



AdaBoost & Gradient Boosting & Random Forest & Extra Trees

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ohe= OneHotEncoder()
ct= make_column_transformer((ohe, categorical), remainder='passthrough')

ada = AdaBoostClassifier(random_state=0)
gb = GradientBoostingClassifier(random_state=0)
rf = RandomForestClassifier(random_state=0)
et= ExtraTreesClassifier(random_state=0)

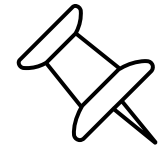
models = [ada,gb,rf,et]

for model in models:
    pipe = make_pipeline(ct, model)
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    accuracy.append(round(accuracy_score(y_test, y_pred),4))
    print(f'model : {model} and accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')
```

	Accuracy
Ada	0.8659
Gradient	0.8768
Random	0.8877
ExtraTree	0.8804

- 4개 모델의 정확도 비슷함
- 특히, Random Forest와 Extra tree 매우 비슷한 점수 나옴
- 두 모델 모두 하이퍼파라미터 튜닝을 통해 성능 개선 가능

3.5 Model Selection - XGBoost & LightGBM



XGBoost & LightGBM

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ohe= OneHotEncoder()
ct= make_column_transformer((ohe, categorical), remainder='passthrough')

xgbc = XGBClassifier(random_state=0)
lgbmc=LGBMClassifier(random_state=0)

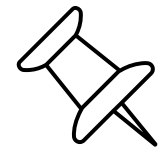
models = [xgbc, lgbmc]

for model in models:
    pipe = make_pipeline(ct, model)
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    accuracy.append(round(accuracy_score(y_test, y_pred), 4))
```

	Accuracy
XGBoost	0.8297
LightGBM	0.8732

- Catboost는 이 두 모델보다 더 나은 작업 수행 가능함

3.5 Model Selection - Catboost



Catboost

- 분류 문제를 위한 모델 학습과 적용

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
categorical_features_indices = np.where(X.dtypes != np.float)[0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = CatBoostClassifier(verbose=False, random_state=0)

model.fit(X_train, y_train, cat_features=categorical_features_indices, eval_set=(X_test, y_test))
y_pred = model.predict(X_test)
accuracy.append(round(accuracy_score(y_test, y_pred), 4))
```

카테고리형 컬럼

검증 세트 지정

	Accuracy
Catboost_default	0.8804

- 기본 값만으로도 앞선 두 모델보다 정확도 약간 상승
- 모델의 최대 성능 확인 위해 파라미터 튜닝

3.5 Model Selection - Catboost

Catboost HyperParameter Tuning with **Optuna**

 Optuna `import optuna`

- 하이퍼파라미터 최적화 프레임워크
- 최신 Automl 기법
- 파라미터의 범위, 목록을 설정하면 매 Trial마다 파라미터를 변경하면서 최적화

GridSearchCV	Optuna
<ul style="list-style-type: none">- 파라미터 값 직접 지정- 수행시간 오래 걸림	<ul style="list-style-type: none">- 파라미터 범위 지정<ul style="list-style-type: none">→ 범위 내에서 자동탐색 통해 최적의 하이퍼파라미터 도출- 수행시간 비교적 빠름- 학습 절차 확인 가능한 시각화 툴 제공

3.5 Model Selection - Catboost



Catboost HyperParameter Tuning with Optuna

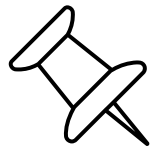
```
def objective(trial):  
    trial - 조정해야하는 하이퍼파라미터를 지정하기 위해  
    objective 함수에 전달  
    X= df.drop('HeartDisease', axis=1)  
    y= df['HeartDisease']  
    categorical_features_indices = np.where(X.dtypes != np.float)[0]  
  
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)  
  
    param = {  
        조정할 하이퍼파라미터 정의  
        "objective": trial.suggest_categorical("objective", ["Logloss", "CrossEntropy"]),  
        "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.1),  
        "depth": trial.suggest_int("depth", 1, 12),  
        "boosting_type": trial.suggest_categorical("boosting_type", ["Ordered", "Plain"]),  
        "bootstrap_type": trial.suggest_categorical(  
            "bootstrap_type", ["Bayesian", "Bernoulli", "MVS"]  
        ),  
        "used_ram_limit": "3gb",  
    }  
    trial.suggest_categorical('파라미터_이름', [파라미터 값])  
    → List 내의 데이터 중 선택  
    trial.suggest_float/int('파라미터_이름', 범위)  
    → 범위 내의 실수형/정수형 값 선택  
  
    if param["bootstrap_type"] == "Bayesian":  
        param["bagging_temperature"] = trial.suggest_float("bagging_temperature", 0, 10)  
    elif param["bootstrap_type"] == "Bernoulli":  
        param["subsample"] = trial.suggest_float("subsample", 0.1, 1)  
  
    cat_cls = CatBoostClassifier(**param)  
  
    cat_cls.fit(X_train, y_train, eval_set=[(X_test, y_test)], cat_features=categorical_features_indices,  
        verbose=0, early_stopping_rounds=100)  
  
    preds = cat_cls.predict(X_test)  
    pred_labels = np rint(preds)  
    accuracy = accuracy_score(y_test, pred_labels)  
    return accuracy
```

```
if __name__ == "__main__":  
    study = optuna.create_study(direction="maximize")  
    study.optimize(objective, n_trials=50, timeout=600)  
  
    print("Number of finished trials: {}".format(len(study.trials)))  
  
    print("Best trial:") 최적의 파라미터 확인  
    trial = study.best_trial  
  
    print("  Value: {}".format(trial.value))  
  
    print("  Params: ")  
    for key, value in trial.params.items():  
        print("    {}: {}".format(key, value))
```

optuna 적용

- 최적화할 함수: objective
- direction: score값을 최대('maximize')또는
최소('minimize')로하는 방향으로 지정
- n_trials: 시도 횟수

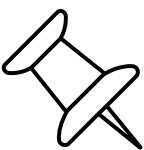
3.5 Model Selection - Catboost



Catboost Parameter

objective	모델 평가 metrics	"Logloss" , "CrossEntropy"
colsample_bylevel(rsm)	랜덤으로 피쳐 샘플링 (각각의 트리 depth 마다 사용할 피쳐의 비율) -> 성능에는 영향주지 않으면서 훈련 속도 높임	float (0;1], 기본값 1
boosting_type	부스팅 방법 설정 (작은 데이터셋에 설정, 오버피팅 방지, 시간 오래 걸림)	["Ordered", "Plain"]
bootstrap_type	객체의 가중치에 대한 샘플링 방법	["Bayesian", "Bernoulli", "MVS"]
used_ram_limit	Attempt to limit the amount of used CPU RAM (cpu ram 제한)	정수, 기본값 없음

3.5 Model Selection - Catboost



Catboost 최적 모델

best parameter 확인

```
Number of finished trials: 50
Best trial:
Value: 0.9021739130434783
Params:
  objective: CrossEntropy
  colsample_bylevel: 0.07461412258635804
  depth: 12
  boosting_type: Plain
  bootstrap_type: MVS
```

최적 모델 학습 결과

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
categorical_features_indices = np.where(X.dtypes != np.float)[0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

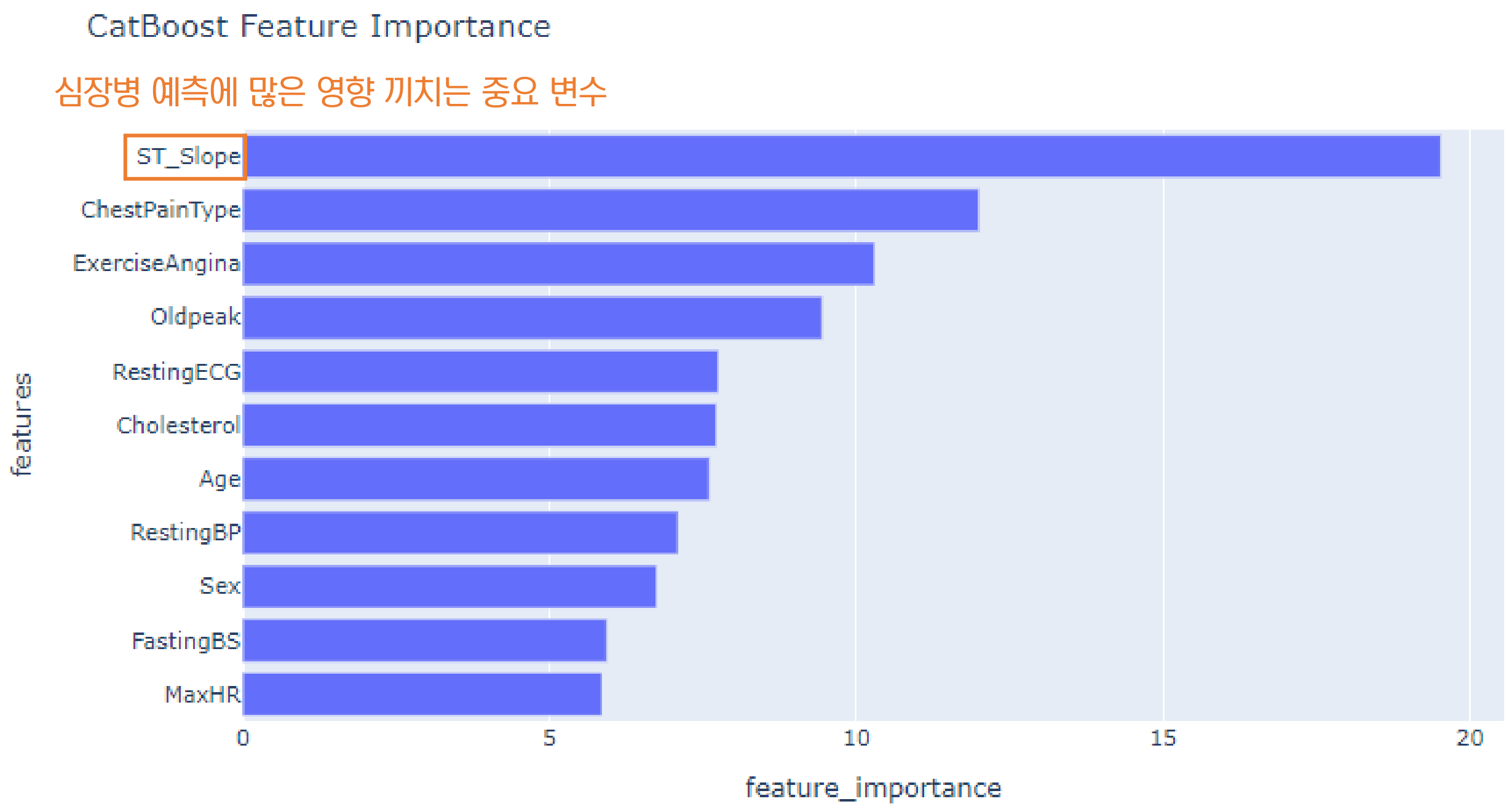
model = CatBoostClassifier(verbose=False, random_state=0,
                           objective= 'CrossEntropy',
                           colsample_bylevel= 0.04292240490294766,
                           depth= 10,
                           boosting_type= 'Plain',
                           bootstrap_type= 'MVS')

model.fit(X_train, y_train, cat_features=categorical_features_indices, eval_set=(X_test, y_test))
y_pred = model.predict(X_test)
accuracy.append(round(accuracy_score(y_test, y_pred),4))
print(classification_report(y_test, y_pred))
```

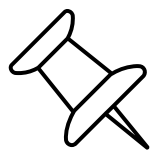
	Accuracy			Accuracy
Catboost_default	0.8804	➡	Catboost_tuned	0.9094

3.5 Model Selection - Feature Importance

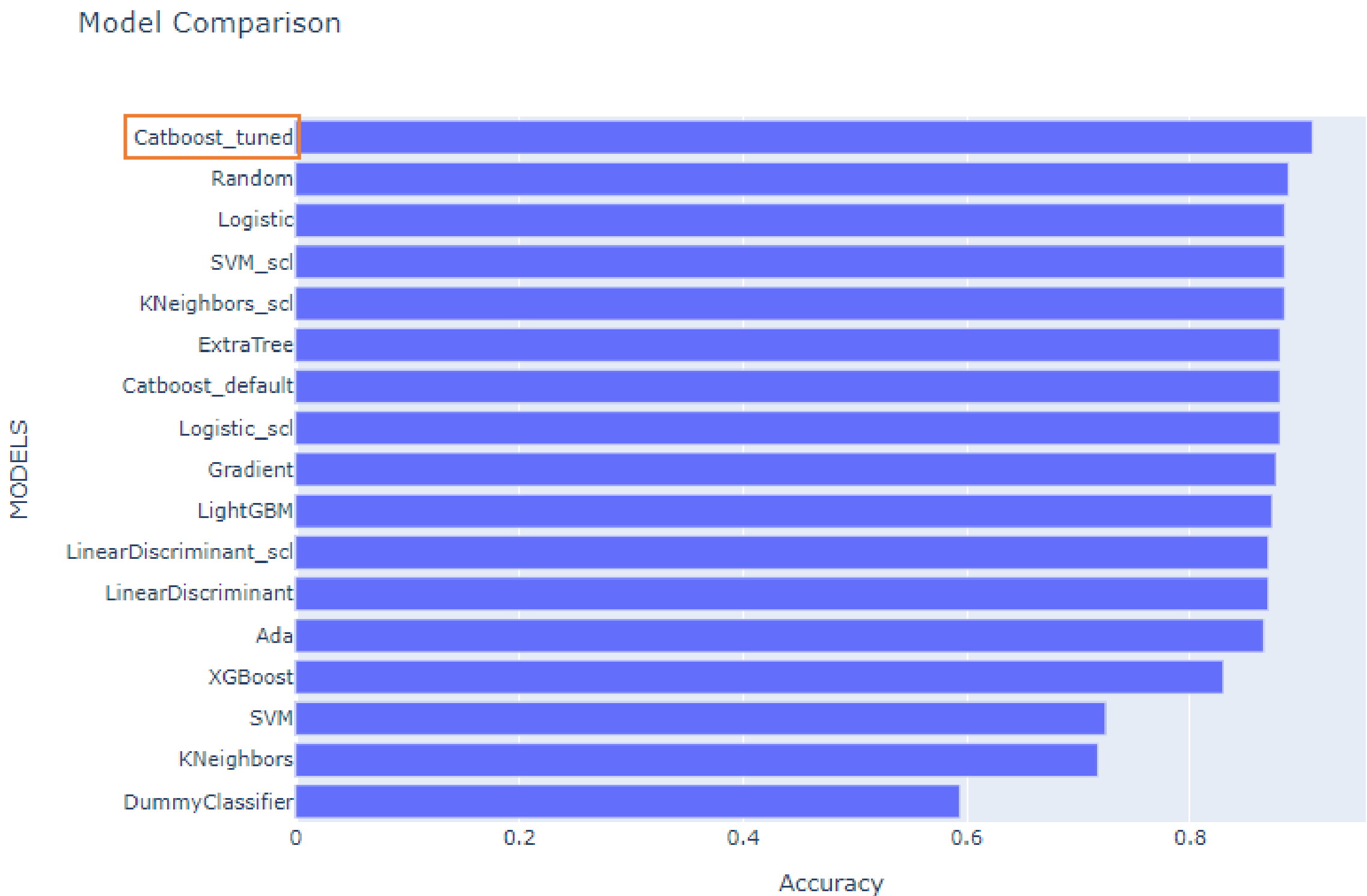
Catboost Feature Importance



3.5 Model Selection - Model Comparison



Model Comparison



3.6 결론

📌 심장병 사례 분류를 위한 모델을 개발했음

1 EDA

- 어떤 metric로 모델 평가할지 결정
- target 변수와 피처변수(수치형 변수, 범주형 변수로 구분)를 아주 자세히 분석했음

2 Model selection

- 범주형 변수를 수치형 변수로 변환(원-핫 인코딩)
- pipeline 사용 → data leakage 방지
- 각 모델의 정확도 확인

3 Model selection – Catboost

- 튜닝 없이 Catboost 사용
- 성능 향상 위해, Optuna를 이용해 Catboost의 하이퍼파라미터 튜닝
- feature importance 확인

THANK YOU

