1. 분류 실습 - 산탄데르 고객 만족 예측

초급 세션 2팀 조주현

EWHA EURON

#1. 데이터 전처리

```
import pandas as pd

data = pd.read_csv('data/santander/train.csv', encoding='latin-1')
print("Data Shape: ", data.shape)
```

```
Data Shape:  (76020, 371)
```

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

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

EWHA EURON

#1. 데이터 전처리

```
In [7]: data.describe()
```

Out[7]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op_var40_comer_ult1
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.363067	119.529632	3.559130
std	43781.947379	39033.462364	12.956486	1614.757313	339.315831	546.266294	93.155749
min	1.000000	-999999.000000	5.000000	0.000000	0.000000	0.000000	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000	0.000000	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000	0.000000	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000	0.000000	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000	21024.810000	8237.820000

8 rows × 371 columns

```
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.sh
print(' 학습 세트 레이블 값 분포 비율')
print(y_train.value_counts()/train_cnt)
print('\n 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
```

#2. XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
xgb_clf = XGBClassifier(n_estimators=500, random_state=156)
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))
```

```
[114]   validation_0-auc:0.93793        validation_1-auc:0.82525
ROC AUC: 0.8413
```

```
from sklearn.model_selection import GridSearchCV
xgb_clf = XGBClassifier(n_estimators=100)
params = {'max_depth':[5, 7] , 'min_child_weight':[1,3] ,'colsample_bytree':[0.5, 0.75] }
gridcv = GridSearchCV(xgb_clf, param_grid=params)
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))
```

```
[43]    validation_0-auc:0.88631        validation_1-auc:0.84276
[44]    validation_0-auc:0.88684        validation_1-auc:0.84266
[45]    validation_0-auc:0.88711        validation_1-auc:0.84252
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.5, 'max_depth': 5, 'min_child_weight': 1}
```

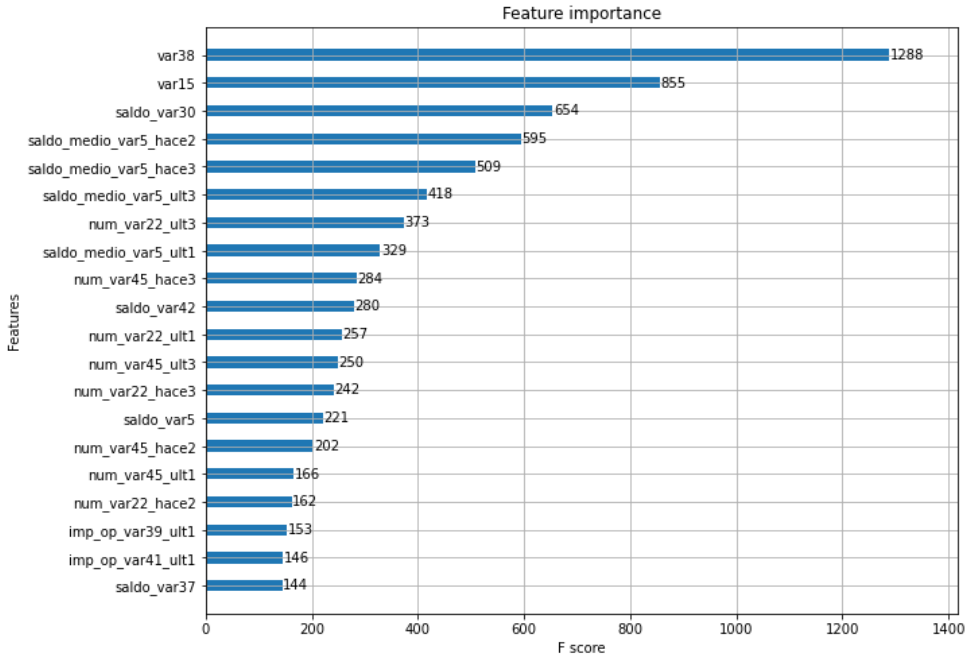
EWHA EURON

#2. XGBoost 모델 학습과 하이퍼 파라미터 튜닝

```
xgb_clf = XGBClassifier(n_estimators=1000, random_state=156, learning_rate=0.02,
max_depth=5,\ min_child_weight=1, colsample_bytree=0.5, reg_alpha=0.03)
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))
```

```
[470]   validation_0-auc:0.88378        validation_1-auc:0.84439
ROC AUC: 0.8457
```

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



#3. LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
from lightgbm import LGBMClassifier
lgbm_clf = LGBMClassifier(n_estimators=500)
eval_set=[(X_tr, y_tr), (X_val, y_val)]
lgbm_clf.fit(X_tr, y_tr, early_stopping_rounds=100, eval_metric="auc", eval_set=eval_set)
lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:,1])
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

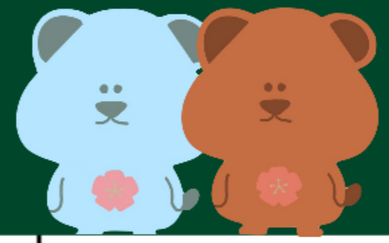
```
[142]   training's auc: 0.946302        training's binary_logloss: 0.0921179    valid_1's auc: 0.829267 v
alid_1's binary_logloss: 0.137482

ROC AUC: 0.8384
```

```
from sklearn.model_selection import GridSearchCV
lgbm_clf = LGBMClassifier(n_estimators=200)
params = {'num_leaves': [32, 64 ],
          'max_depth':[128, 160],
          'min_child_samples':[60, 100],
          'subsample':[0.8, 1])
gridcv = GridSearchCV(lgbm_clf, param_grid=params)
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))
```

#3. LightGBM 모델 학습과 하이퍼 파라미터 튜닝

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=32, sumbsample=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: {0:.4f}'.format(lgbm_roc_score))
```

ROC AUC: 0.8442

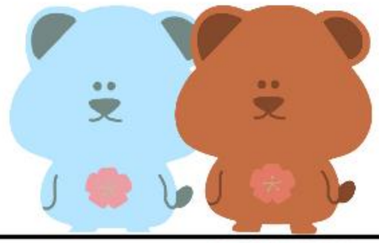


의약품 분류 데이터셋 예제

2팀 이채원

EWHA EURON

목차



EWHA EURON

1. 데이터 검토

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

- 6개의 column, null값 없음
- 숫자형 column 2개, 문자열형 4개

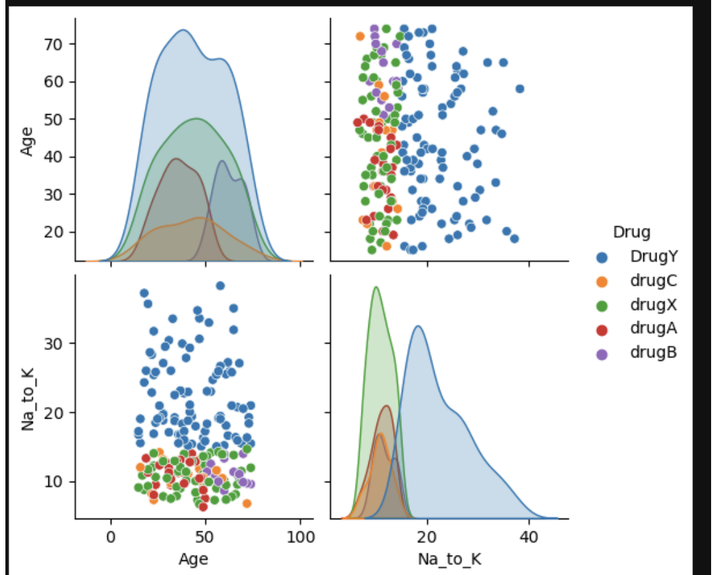
```
data.describe()
```

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

- 숫자형 데이터의 분포 확인

```
sns.pairplot(data, hue = "Drug")
```

```
C:\Users\chwon\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
rsion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\chwon\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
rsion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
<seaborn.axisgrid.PairGrid at 0x1fec9ae7410>
```



- 숫자형 데이터에 따른 레이블 값 분포 시각화

1. 데이터 검토

```
data.columns
```

```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

- 6개의 column

```
data["Age"].value_counts(dropna=False)
```

```
Age
47    8
23    7
28    7
49    7
39    6
32    6
50    5
37    5
58    5
60    5
22    5
34    4
72    4
```

```
data["Sex"].value_counts()
```

```
Sex
M    104
F     96
Name: count, dtype: int64
```

```
data["BP"].value_counts()
```

```
BP
HIGH      77
LOW       64
NORMAL    59
Name: count, dtype: int64
```

- 각 column의 value 확인

```
data["Cholesterol"].value_counts()
```

```
Cholesterol
HIGH      103
NORMAL     97
Name: count, dtype: int64
```

```
data["Drug"].value_counts()
```

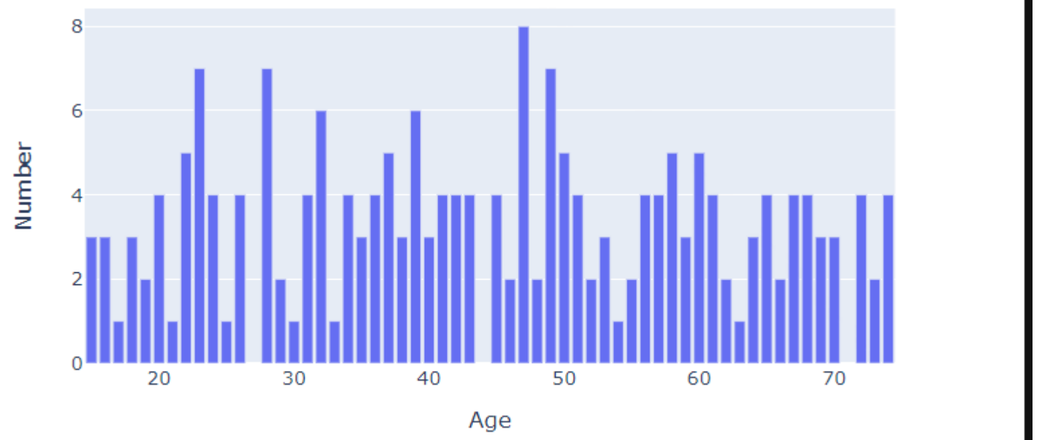
```
Drug
DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: count, dtype: int64
```

```
data["Na_to_K"].value_counts(dropna=False)
```

```
Na_to_K
12.006    2
18.295    2
25.355    1
11.939    1
16.347    1
         ..
24.658    1
24.276    1
13.967    1
19.675    1
11.349    1
Name: count, Length: 198, dtype: int64
```

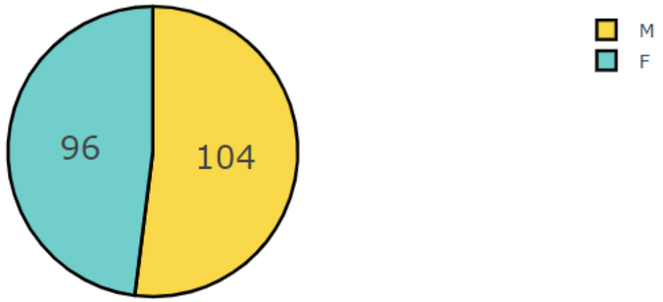
2. 데이터 시각화

```
dataAge = data["Age"].value_counts(dropna = False)
npar_dataAge = np.array(dataAge)
x = list(npar_dataAge)
y = data.Age.value_counts().index

DataAge = {"Age": y, "Number": x}
DataAge = pd.DataFrame(DataAge)

fig = px.bar(DataAge, x = "Age", y = "Number")
fig.show()
```



- Age에 따른 사람 수를 bar 그래프로 표현

```
colors = ['gold', 'mediumturquoise']

fig = go.Figure(data = [go.Pie(labels= ['M', 'F'], values=[104, 96])])

fig.update_traces(hoverinfo = 'label + percent', textinfo = 'value', textfont_size = 20,
                  marker = dict(colors = colors, line = dict(color = '#000000', width = 2)))
fig.show()
```

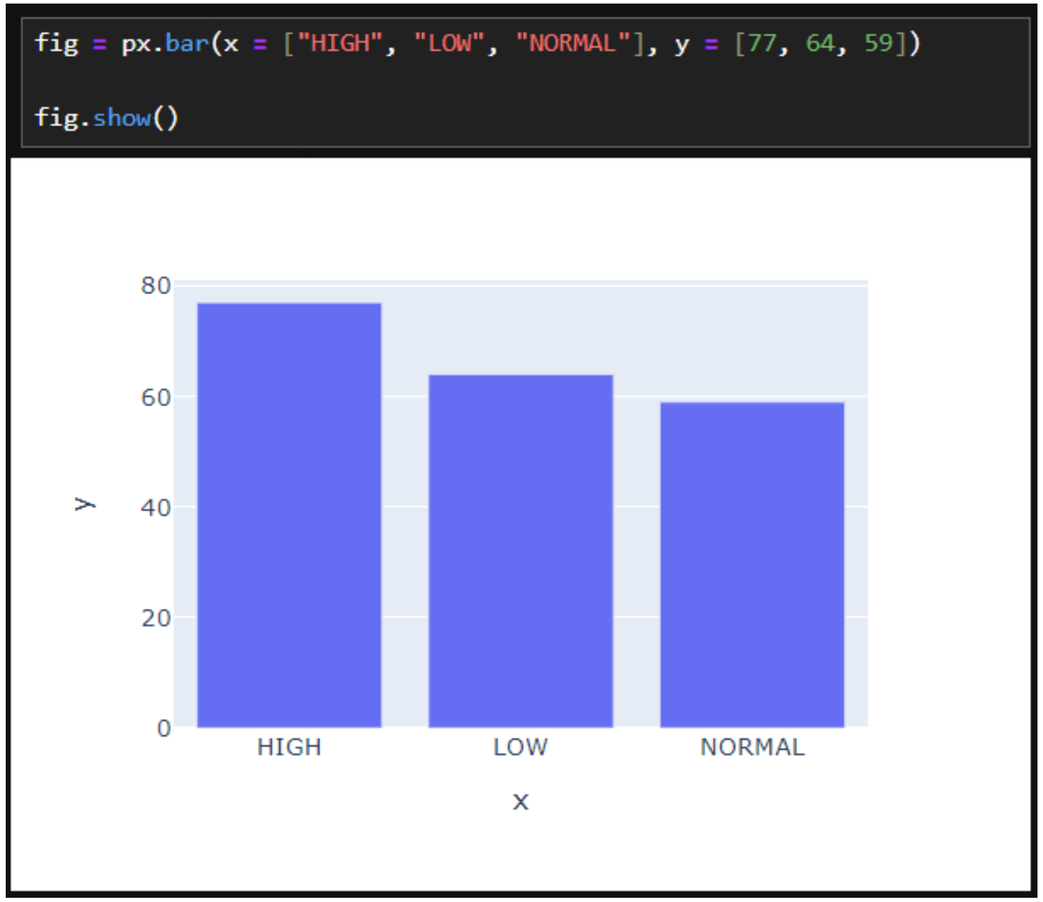


- 성별에 따른 사람 수를 pie 그래프로 표현

EWHA EURON

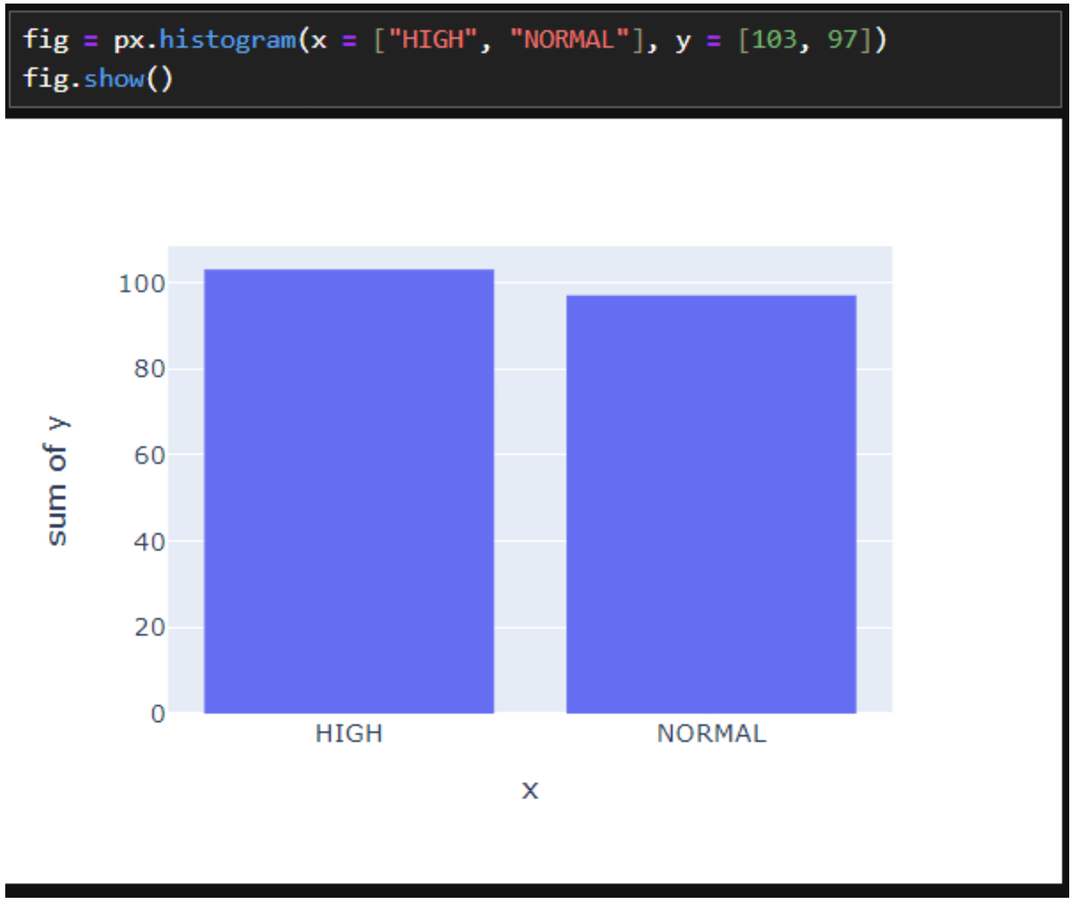
2. 데이터 시각화

```
fig = px.bar(x = ["HIGH", "LOW", "NORMAL"], y = [77, 64, 59])

fig.show()
```



- BP에 따른 사람 수를 bar 그래프로 표현

```
fig = px.histogram(x = ["HIGH", "NORMAL"], y = [103, 97])
fig.show()
```

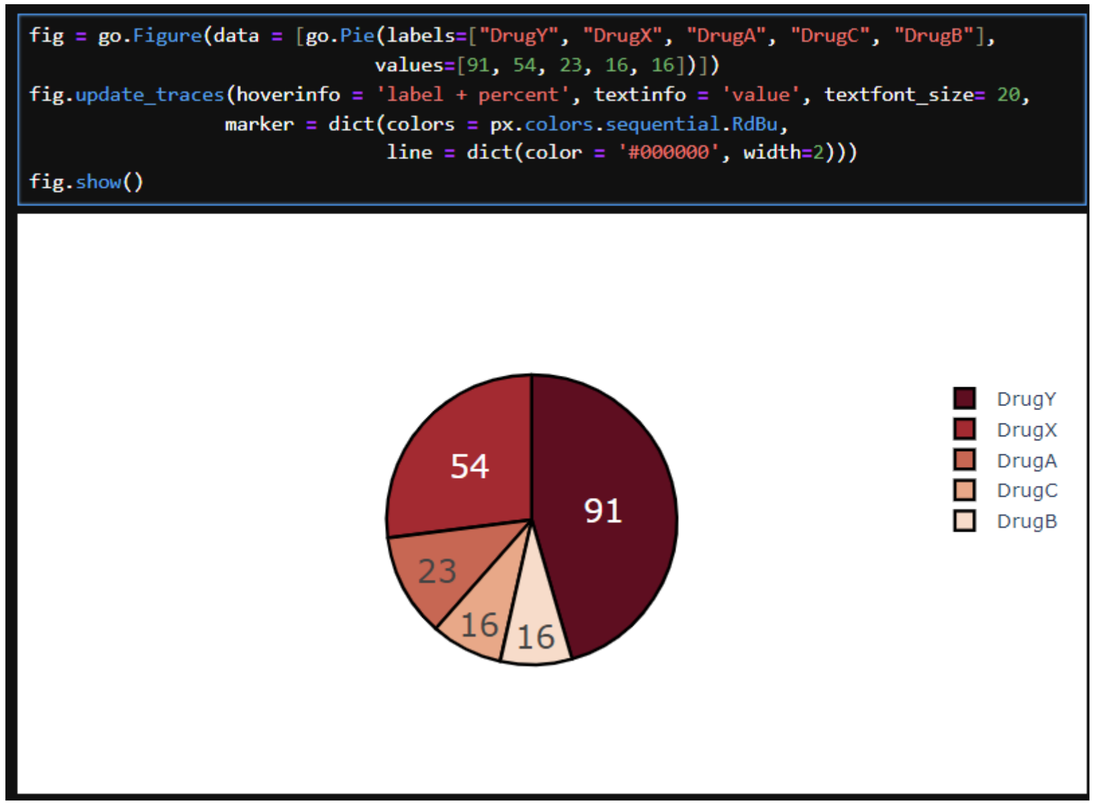


- Cholestrol에 따른 사람 수를 bar 그래프로 표현

EWHA EURON

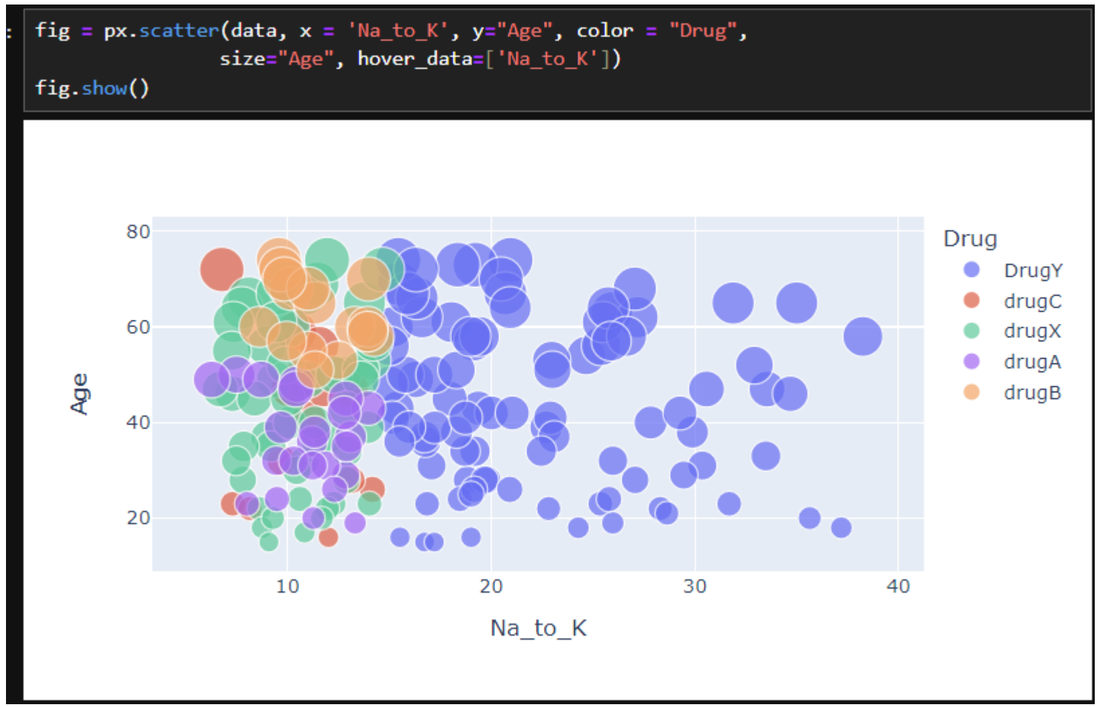
2. 데이터 시각화

```
fig = go.Figure(data = [go.Pie(labels=["DrugY", "DrugX", "DrugA", "DrugC", "DrugB"],
                               values=[91, 54, 23, 16, 16])])
fig.update_traces(hoverinfo = 'label + percent', textinfo = 'value', textfont_size= 20,
                  marker = dict(colors = px.colors.sequential.RdBu,
                                line = dict(color = '#000000', width=2)))
fig.show()
```



- Drug에 따른 사람 수를 pie 그래프로 표현

```
fig = px.scatter(data, x = 'Na_to_K', y="Age", color = "Drug",
                 size="Age", hover_data=['Na_to_K'])
fig.show()
```



- Na to K와 Age의 상관 관계를 산점도 그래프로 표현

3. 데이터 전처리

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

- 문자열 type인 Sex, BP, Cholesterol, Drug column의 숫자형으로의 전환 필요

```
dataclass.Sex = [1 if i == "F" else 0 for i in dataclass.Sex]
```

```
import warnings
warnings.filterwarnings('ignore')

for i in range(0, len(dataclass.BP)):
    if dataclass.BP[i] == "LOW":
        dataclass.BP[i] = 2
    elif dataclass.BP[i] == "NORMAL":
        dataclass.BP[i] = 1
    else :
        dataclass.BP[i] = 0
```

```
dataclass.Cholesterol = [1 if i == "HIGH" else 0 for i in dataclass.Cholesterol]
```

```
for i in range(0, len(dataclass)) :
    if dataclass.Drug[i] == "DrugY" :
        dataclass.Drug[i] = 4
    elif dataclass.Drug[i] == "drugX" :
        dataclass.Drug[i] = 3
    elif dataclass.Drug[i] == "drugA" :
        dataclass.Drug[i] = 2
    elif dataclass.Drug[i] == "drugC" :
        dataclass.Drug[i] = 1
    else:
        dataclass.Drug[i] = 0
```

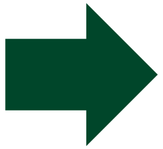
- 각각 숫자형 데이터로 전환

EWHA EURON

3. 데이터 전처리

```
dataclass.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    int64
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    int64
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(3), object(2)
memory usage: 9.5+ KB
```



```
data_types_dict = {"BP":int, "Drug": int}

dataclass = dataclass.astype(data_types_dict)

dataclass.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    int64
 2   BP           200 non-null    int32
 3   Cholesterol  200 non-null    int64
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    int32
dtypes: float64(1), int32(2), int64(3)
memory usage: 7.9 KB
```

- 여전히 문자열형인 칼럼 존재

- 정수형으로 변환

3. 데이터 전처리

```
x_data = dataclass.drop(["Drug"], axis=1)
y_data = dataclass.Drug.values
```

- 피처와 레이블 분리

y_data

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

x_data

	Age	Sex	BP	Cholesterol	Na_to_K
0	23	1	0	1	25.355
1	47	0	2	1	13.093
2	47	0	2	1	10.114
3	28	1	1	1	7.798
4	61	1	2	1	18.043
...	...	...	...	...	...
195	56	1	2	1	11.567
196	16	0	2	1	12.006
197	52	0	1	1	9.894
198	23	0	1	0	14.020
199	40	1	2	0	11.349

- 분리한 결과 확인

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3,
                                                    random_state=1)
```

- 훈련, 테스트 셋 분리

4. 모델링

- DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

dtc = DecisionTreeClassifier()

dtc.fit(X_train, y_train)

predict = dtc.predict(X_test)

print('The accuracy of the DecisionTree is', metrics.accuracy_score(predict, y_test))
```

```
The accuracy of the DecisionTree is 0.9666666666666667
```

- 그대로 결정트리를 적용했을 시 정확도 : 약 **96.7%**

4. 모델링

```
DTC_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)

DTC_gini.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
y_pred_gini = DTC_gini.predict(X_test)
```

```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini  index: {0:.4f}'.format(accuracy_score(y_test, y_pred_gini)))
```

```
Model accuracy score with criterion gini  index: 0.9000
```

-지니 계수를 사용
: 90%의 정확도

```
DTC_en = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)

DTC_en.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

```
y_pred_en = DTC_en.predict(X_test)
```

```
print('Model accuracy score with criterion entropy : {0:.4f}'.format(accuracy_score(y_test, y_pred_en)))
```

```
Model accuracy score with criterion entropy : 0.9000
```

-엔트로피를 사용
: 90%의 정확도

EWHA EURON

4. 모델링

* 지니 계수, 엔트로피 리마인드!

- 결정 트리 : **균일도**가 높은 데이터 세트를 우선적으로 선택하도록 규칙을 만듦.
- 이 때 균일도를 측정하는 지표가 바로 지니 계수와 엔트로피!

참고 :

- 정보 이득은 엔트로피라는 개념을 기반으로 합니다. 엔트로피는 주어진 데이터 집합의 혼잡도를 의미하는데, 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮습니다. 정보 이득 지수는 1에서 엔트로피 지수를 뺀 값입니다. 즉, 1-엔트로피 지수입니다. 결정 트리는 이 정보 이득 지수로 분할 기준을 정합니다. 즉, 정보 이득이 높은 속성을 기준으로 분할합니다.
- 지니 계수는 원래 경제학에서 불평등 지수를 나타낼 때 사용하는 계수입니다. 경제학자인 코라도 지니(Corrado Gini)의 이름에서 딴 계수로서 0이 가장 평등하고 1로 갈수록 불평등합니다. 머신러닝에 적용될 때는 지니 계수가 낮을수록 데이터 균일도가 높은 것으로 해석해 지니 계수가 낮은 속성을 기준으로 분할합니다.

4. 모델링

- RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state = 0)

rfc.fit(X_train, y_train)

predict = rfc.predict(X_test)

print('The accuracy of the RandomForest is', accuracy_score(predict, y_test))
```

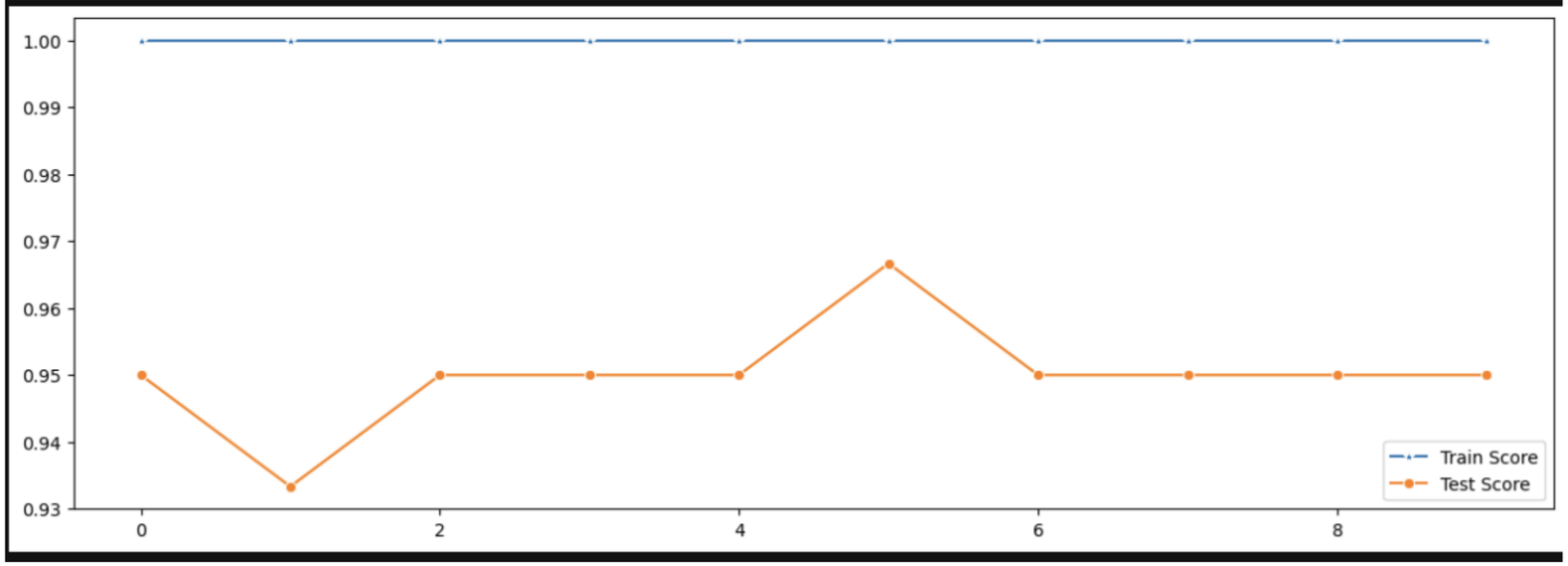
```
The accuracy of the RandomForest is 0.95
```

- 95%의 정확도

만약 최적으로 튜닝한다면?

4. 모델링

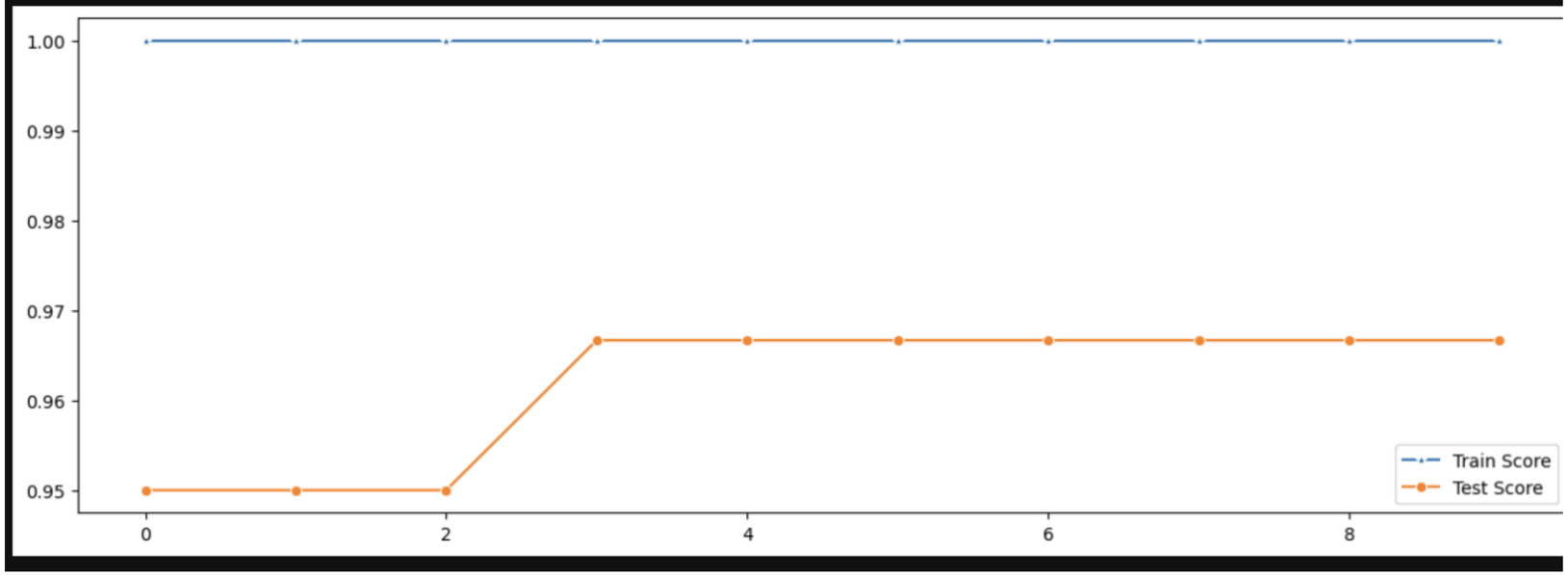
```
test_score_list = []
train_score_list = []

for i in range(10) :
    rfc2 = RandomForestClassifier(random_state=i)
    rfc2.fit(X_train, y_train)
    test_score_list.append(rfc2.score(X_test, y_test))
    train_score_list.append(rfc2.score(X_train, y_train))

plt.figure(figsize=(15,5))
p = sns.lineplot(x=range(0, 10), y=train_score_list, marker='*', label='Train Score')
p = sns.lineplot(x=range(0, 10), y=test_score_list, marker='o', label='Test Score')
```



- random_state: 5가 최적

4. 모델링

```
test_score_list = []
train_score_list = []

list_n_estimators = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

for i in range(len(list_n_estimators)) :
    rfc3 = RandomForestClassifier(n_estimators=list_n_estimators[i], random_state=5)
    rfc3.fit(X_train, y_train)
    test_score_list.append(rfc3.score(X_test, y_test))
    train_score_list.append(rfc3.score(X_train, y_train))
plt.figure(figsize=(15, 5))
p = sns.lineplot(x=range(0, 10), y=train_score_list, marker='*', label='Train Score')
p = sns.lineplot(x=range(0, 10), y=test_score_list, marker='o', label='Test Score')
```



- n_estimators: **40 이상**이 최적

4. 모델링

최적 하이퍼 파라미터를 이용해 분류한 결과 :

```
last_rfc = RandomForestClassifier(n_estimators=100, random_state=5)

last_rfc.fit(X_train, y_train)

predict = last_rfc.predict(X_test)

print('The accuracy of the RandomForest is', accuracy_score(predict, y_test))
```

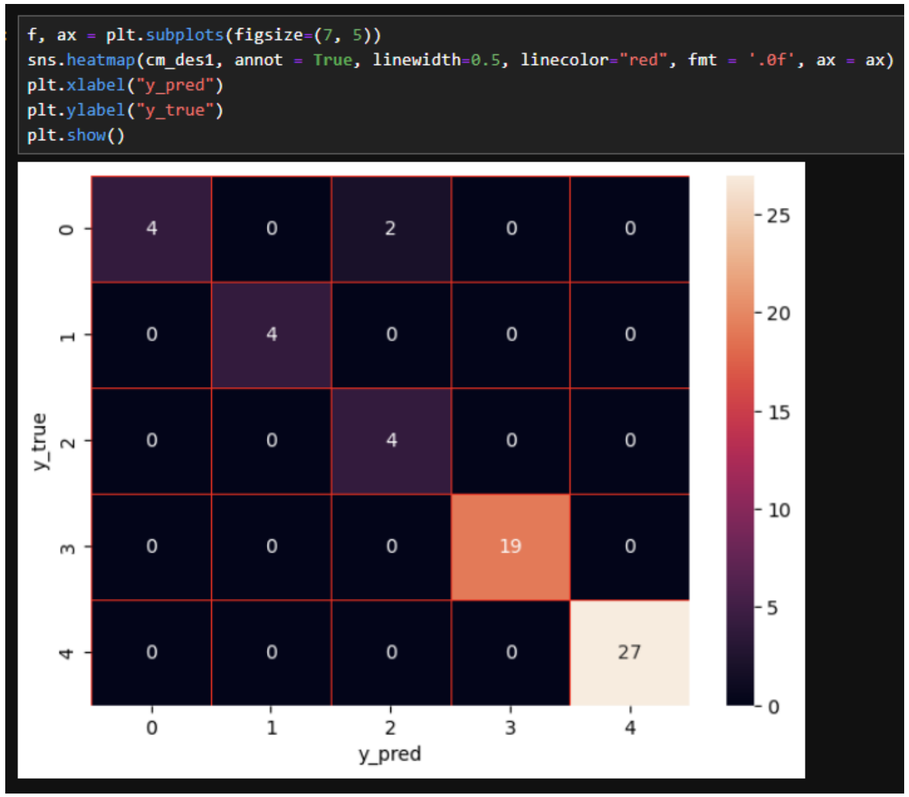
```
The accuracy of the RandomForest is 0.9666666666666667
```

약 96.7%의 정확도를 보임.

5. 평가

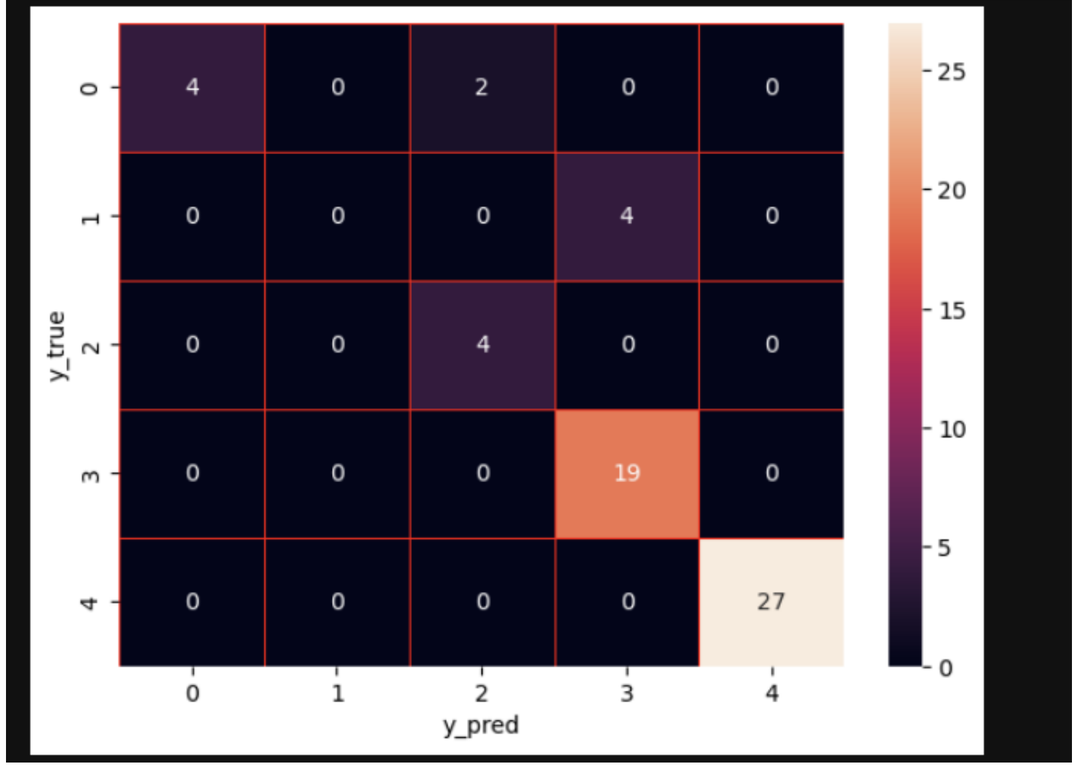
```
cm_des = DecisionTreeClassifier()

cm_des.fit(X_train, y_train)

y_pred_cm = cm_des.predict(X_test)
y_true = y_test

cm_des1 = confusion_matrix(y_true, y_pred_cm)
```

```
f, ax = plt.subplots(figsize=(7, 5))
sns.heatmap(cm_des1, annot = True, linewidth=0.5, linecolor="red", fmt = '.0f', ax = ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



- 결정 트리

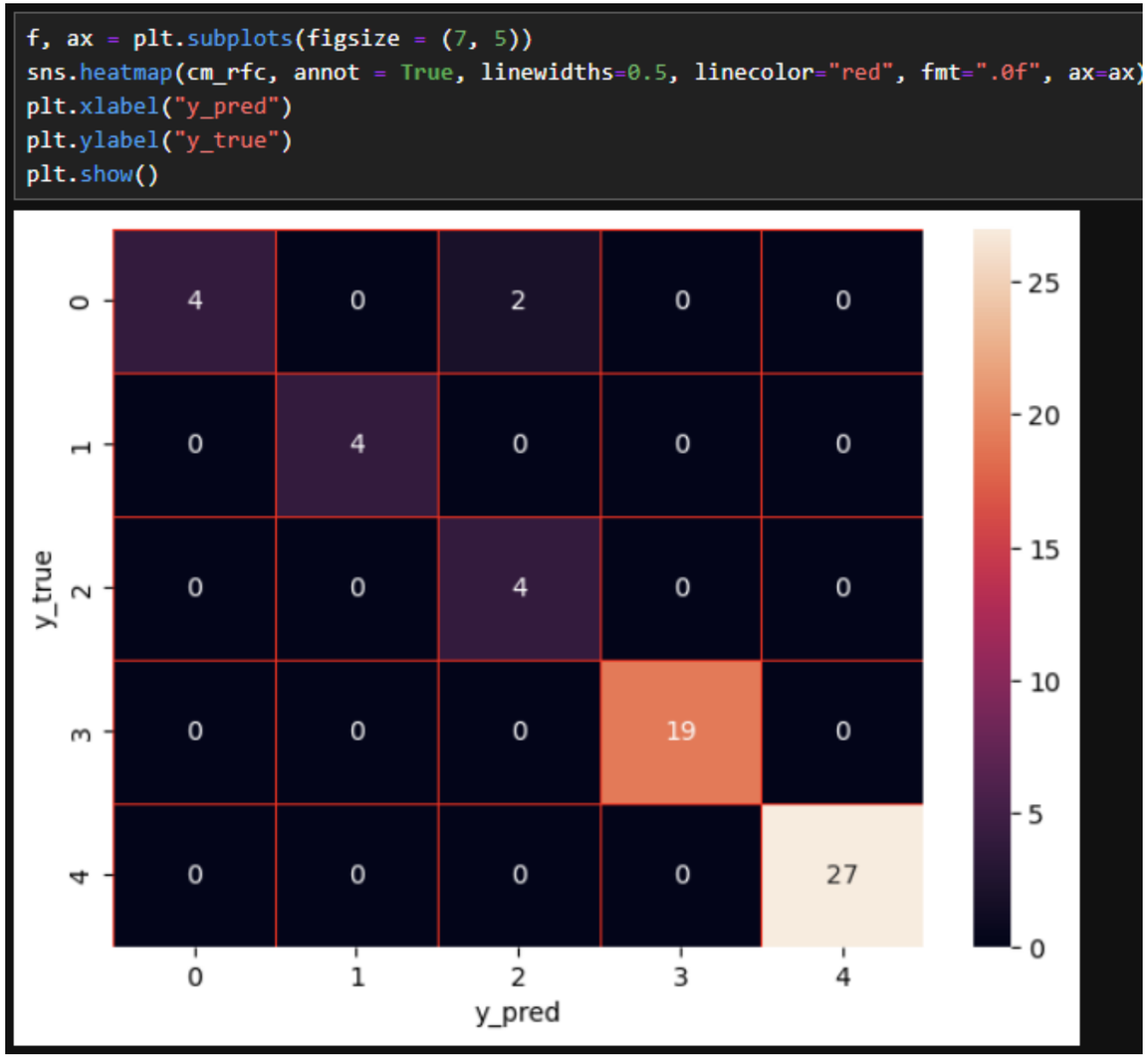
```
f, ax = plt.subplots(figsize = (7, 5))
sns.heatmap(cm_des2, annot = True, linewidths=0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



- 결정 트리 – 지니 계수

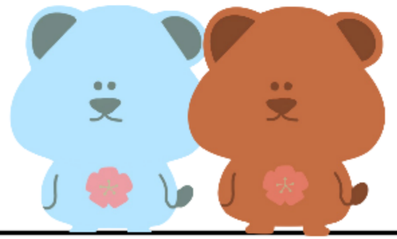
5. 평가

```
f, ax = plt.subplots(figsize = (7, 5))
sns.heatmap(cm_rfc, annot = True, linewidths=0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



- 최적으로 튜닝한 랜덤 포레스트

THANK YOU



EWHA EURON

4.9 분류 실습 – 캐글 신용가드 사기 검출

문원정

EWHA EURON

캐글 신용카드 사기 검출

Credit Card Fraud Detection

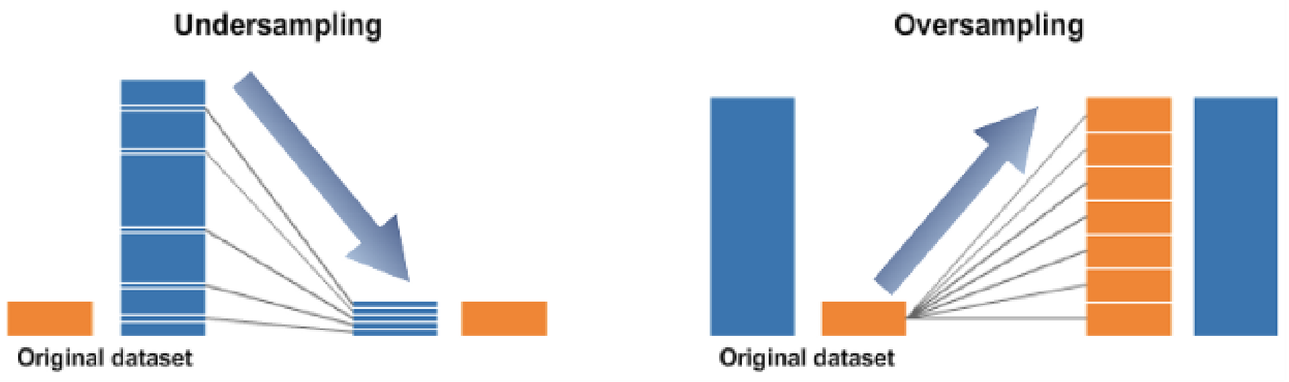
Anonymized credit card transactions labeled as fraudulent or genuine

www.kaggle.com

데이터 세트의 class는 0, 1로 분류되는데 0은 정상적인
신용카드 트랜잭션 데이터, 1은 사기 트랜잭션.

전체 데이터의 0.172%만이 사기 트랜잭션으로 매우 불균형한
분포를 가지고 있다.

언더 샘플링과 오버 샘플링의 이해

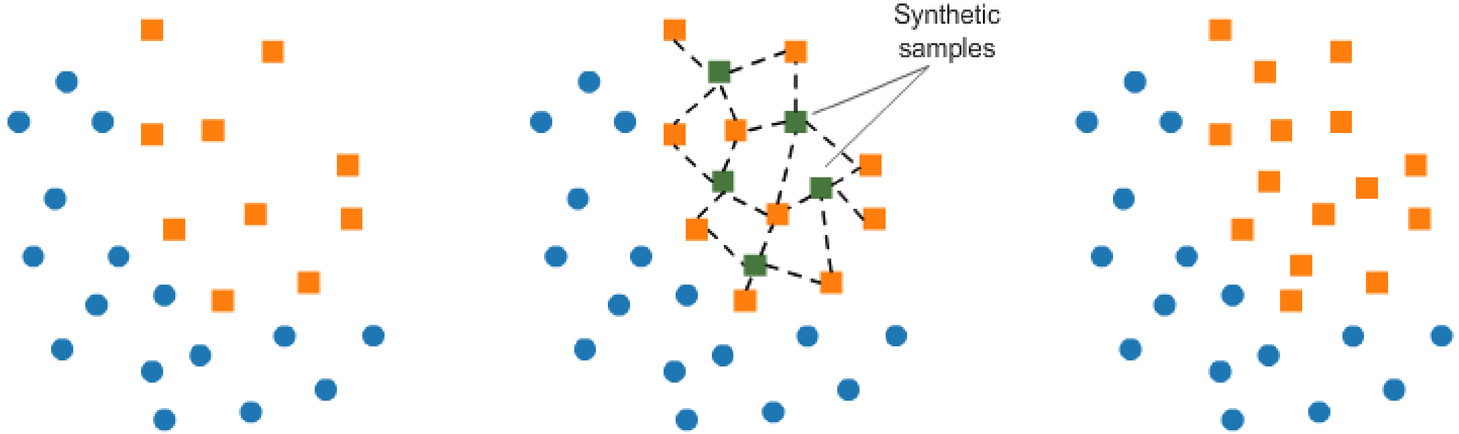


오버 샘플링이 예측 성능상 더 유리한 경우가 많아 주로 사용!

언더 샘플링: 많은 데이터 세트를 적은 데이터 세트 수준으로 감소시키는 방식.

이렇게 학습을 수행하면 과도하게 정상 레이블로 학습/예측하는 부작용을 개선할 수 있지만, 너무 많은 정상 레이블 데이터를 감소시켜 정상 레이블의 경우 오히려 제대로 된 학습을 할 수 없다는 단점이 있어 잘 적용하지 않는다.

언더 샘플링과 오버 샘플링의 이해



오버 샘플링: 이상 데이터와 같은 적은 데이터 세트를 증식하여 학습을 위한 충분한 데이터를 확보하는 방법.

동일한 데이터를 단순히 증식하면 과적합이 되기에 원본 데이터의 피처 값들을 아주 약간만 변경하여 증식.

대표적으로 SMOTE(Synthetic Minority Over-sampling Technique) 방법이 있는데, 이는 적은 데이터 세트에 있는 개별 데이터들의 K Nearest Neighbor 를 찾아서 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어 기존 데이터와 약간 차이가 나는 새로운 데이터를 생성하는 방식.

데이터 일차 가공 및 모델 학습/예측/평가

```
import numpy as np
import pandas as pd
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	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0

3 rows × 31 columns

time 은 데이터 생성 관련한 작업용 속성으로 제거
Amount 는 신용카드 트랜잭션의 금액을 의미,
Class 는 00이면 정상, 10이면 사기 트랜잭션

데이터 일차 가공 및 모델 학습/예측/평가

```
from sklearn.model_selection import train_test_split

def get_preprocessed_df(df=None):
    df_copy = df.copy()
    df_copy.drop('Time', axis=1, inplace=True)
    return df_copy
```

인자로 입력받은 DataFrame을 복사한 뒤 Time 칼럼만 삭제하고 복사된 DataFrame을 반환하는 함수

```
def get_train_test_dataset(df=None):
    df_copy = get_preprocessed_df(df)
    # DataFrame의 맨 마지막 칼럼이 레이블, 나머지는 피처들
    X_features = df_copy.iloc[:, :-1]
    y_target = df_copy.iloc[:, -1]
    # stratify = y_target 으로 stratified 기반 분할
    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)
```

사전 데이터 가공 후 학습과 테스트 데이터 세트를 반환하는 함수

```
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
```

학습 dataset과 테스트 dataset의 label 비율을 보니 서로 비슷하게 분할된 것을 알 수 있음

EWHA EURON

데이터 일차 가공 및 모델 학습/예측/평가

#로지스틱 회귀 이용

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    f1 = f1_score(y_test, pred)
    roc_auc = roc_auc_score(y_test, pred)

    print('오차행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, F1: {3:.4f}, AUC:{4:.4f}'
          .format(accuracy, precision, recall, f1, roc_auc))
```

```
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, lr_pred, lr_pred_proba)
```

```
오차행렬
[[85269    26]
 [   58    90]]
정확도: 0.9990, 정밀도: 0.7759, 재현율: 0.6081, F1: 0.6818, AUC:0.8039
```

데이터 일차 가공 및 모델 학습/예측/평가

#LightGBM 이용

```
def get_model_train_eval(model, ftr_train=None, ftr_test=None,
tgt_train=None, tgt_test=None):
    model.fit(ftr_train, tgt_train)
    pred = model.predict(ftr_test)
    pred_proba = model.predict_proba(ftr_test)[:, 1]
    get_clf_eval(tgt_test, pred, pred_proba)
```

```
# 불균형한 레이블 값 분포도를 가지므로 LGBMClassifier에서
boost_from_average=False로 설정해야한다.
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:0.8783
```

데이터 일차 가공 및 모델 학습/예측/평가

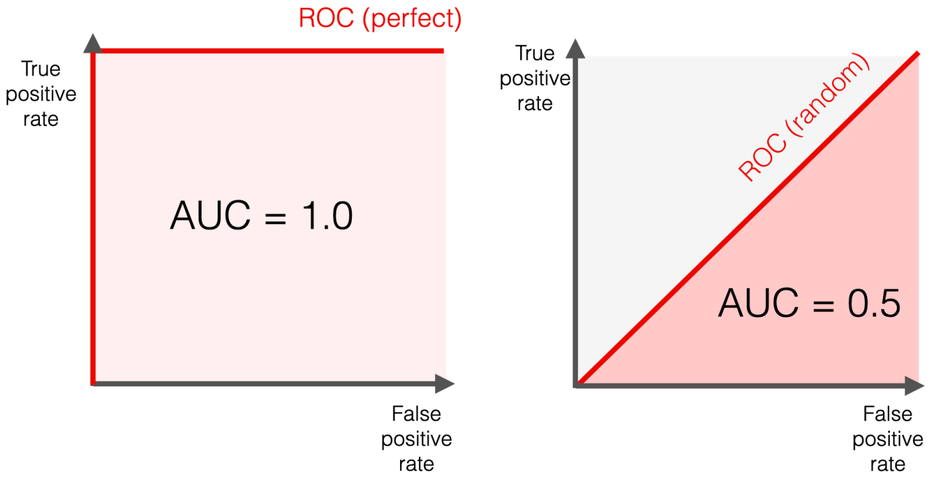
#재현율

$$\text{Recall (재현율)} = \frac{\text{정상으로 올바르게 예측된 데이터 수}}{\text{실제로 정상인 데이터 수}} = \frac{TP}{FN+TP}$$

#ROU-AUC

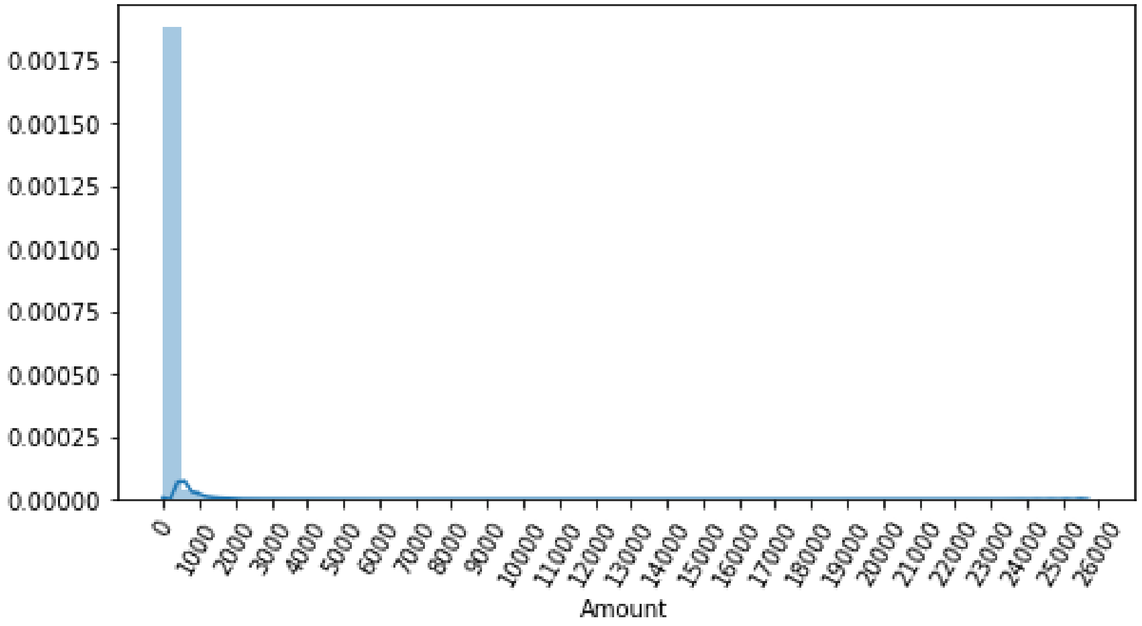
다양한 임계값에서 모델의 분류 성능에 대한 측정 그래프

auc가 높을수록 클래스를 구별하는 모델의 성능이 훌륭하다는 것을 의미.



데이터 분포도 변환 후 모델 학습/예측/평가

#Amount 피처 분포도



1,000불 이하의 데이터가 대부분이며, 26,000불까지 꼬리가 긴 형태의 분포 곡선을 가지고 있음.

데이터 분포도 변환 후 모델 학습/예측/평가

```
from sklearn.preprocessing import StandardScaler
# 사이킷런의 StandardScaler를 이용해 정규 분포 형태로 Amount 피처 값 변환하는 로직으로 수정
def get_preprocessed_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)
    # 기존 Time, Amount 피처 삭제
    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('\n### LightGBM 예측 성능 ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64,
n_jobs=-1, boost_from_average=False)
```

이전 로지스틱 회귀 예측 성능: 정확도: 0.9990, 정밀도: 0.7759, 재현율: 0.6081, F1: 0.6818, AUC:0.8039

현재 로지스틱 회귀 예측 성능: 정확도: 0.9992, 정밀도: 0.8654, 재현율: 0.6081, F1: 0.7143, AUC:0.8040

이전 LightGBM 예측 성능: 정확도: 0.9995, 정밀도: 0.9573, 재현율: 0.7568, F1: 0.8453, AUC:0.8783

현재 LightGBM 예측 성능: 정확도: 0.9995, 정밀도: 0.9569, 재현율: 0.7500, F1: 0.8409, AUC:0.8750

데이터 분포도 변환 후 모델 학습/예측/평가

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

print('\n### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train,
ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

로그 변환은 데이터 분포도가 심하게 왜곡되어 있을 경우 적용하는 중요 기법 중 하나.

원래 큰 값을 상대적으로 작은 값으로 변환하기 때문에 데이터 분포도의 왜곡을 상당 수준 개선해 줌.

```
### 로지스틱 회귀 예측 성능 ###
오차행렬
[[85283    12]
 [   59    89]]
정확도: 0.9992, 정밀도: 0.8812, 재현율: 0.6014,
F1: 0.7149, AUC:0.8006
```

```
### LightGBM 예측 성능 ###
오차행렬
[[85290     5]
 [   35   113]]
정확도: 0.9995, 정밀도: 0.9576, 재현율: 0.7635,
F1: 0.8496, AUC:0.8817
```

이상치 데이터 제거 후 모델 학습/예측/평가

이상치 데이터(Outlier): 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터

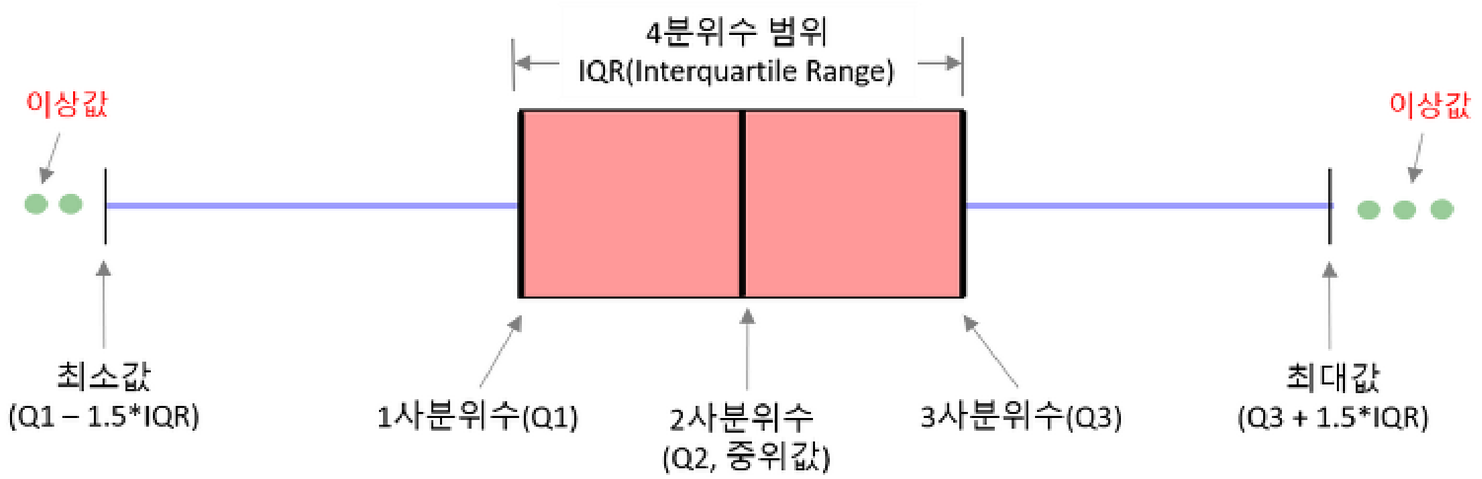
이상치로 인해 머신러닝 모델의 성능에 영향을 받는 경우가 발생하기 쉬움.

매우 많은 피처가 있을 경우 이들 중 결정값(레이블)과 가장 상관성이 높은 피처들을 위주로 이상치를 검출하는 것이 좋음.

IQR(Inter Quantile Range):
사분위 값(Quantile)의 편차를 이용하는 기법으로 흔히 박스 플롯(Box Plot) 방식으로 시각화할 수 있음.

이상치 데이터 제거 후 모델 학습/예측/평가

사분위: 전체 데이터를 오름차순 정렬한 뒤, 이를 1/4(25%)씩으로 구간을 분할하는 것을 지칭.
순서대로 Q1(25%), Q2(50%), Q3(75%), Q4(100%)로 나뉜다.

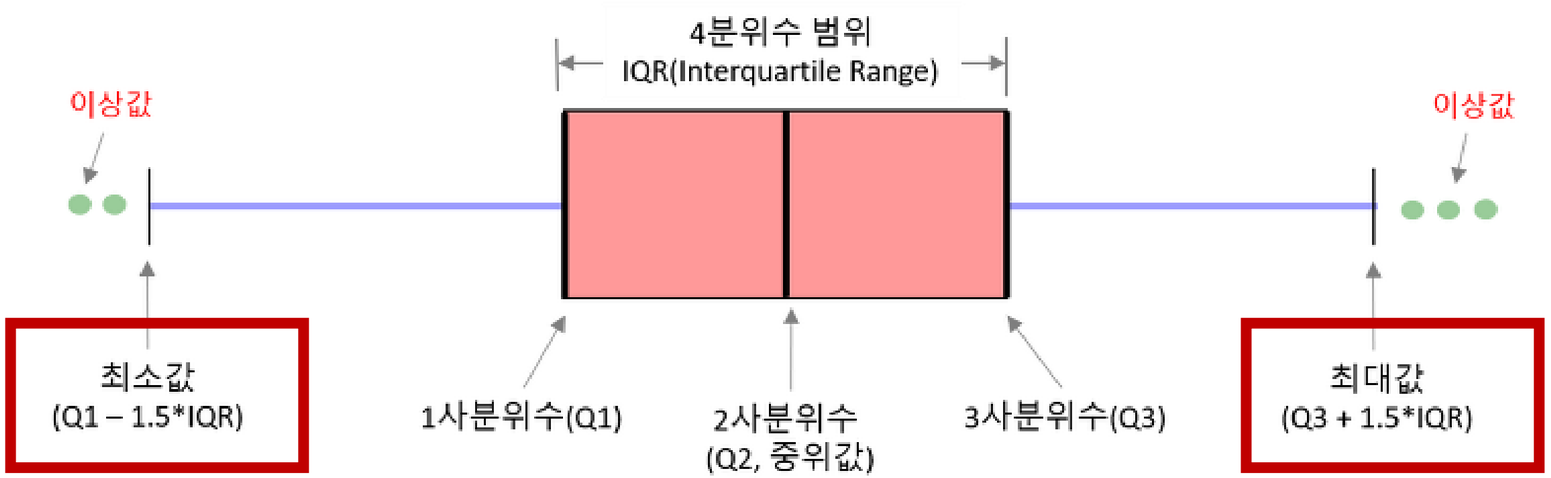


IQR: 25% 구간인 Q1 ~ 75% 구간인 Q3의 범위

이상치 데이터 제거 후 모델 학습/예측/평가

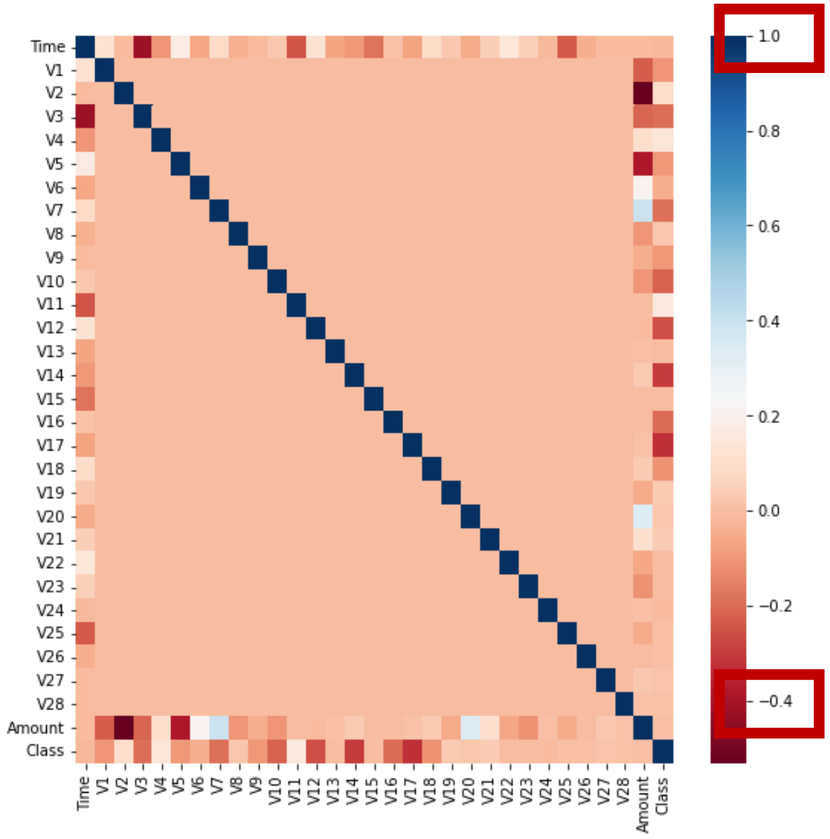
이상치 데이터 검출 방식

보통 IQR에 1.5를 곱해서 생성된 범위를 이용해 최댓값과 최솟값을 결정한 뒤 최댓값을 초과하거나 최솟값을 미달하는 데이터를 이상치로 간주.
1.5가 아닌 다른 값을 적용할 수도 있으며, 보통은 1.5로 적용.



이상치 데이터 제거 후 모델 학습/예측/평가

박스 플롯: IQR 방식을 시각화한 도표로 사분위의 편차와 IQR, 이상치를 나타냄.



양의 상관 관계가 높을수록 색깔이 진한 파란색

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

음의 상관 관계가 높을수록 색깔이 진한 빨간색

결정 레이블인 Class 피처와 음의 상관관계가 가장 높은 피처는 V14, V17

이상치 데이터 제거 후 모델 학습/예측/평가

```
# 이상치 검출 함수

def get_outlier(df=None, column=None, weight=1.5):
    # fraud에 해당하는 column 데이터만 추출, 1/4분위와 3/4분위 지점을 np.percentile로 구함
    fraud = df[df['Class']==1][column]
    quantile_25 = np.percentile(fraud.values, 25)
    quantile_75 = np.percentile(fraud.values, 75)

    # IQR을 구하고, IQR에 1.5를 곱하여 최대값과 최소값 지점 구함
    iqr = quantile_75 - quantile_25
    iqr_weight = iqr * weight
    lowest_val = quantile_25 - iqr_weight
    highest_val = quantile_75 + iqr_weight

    # 최대값보다 크거나, 최소값보다 작은 값을 아웃라이어로 설정하고 DataFrame index 반환
    outlier_index = fraud[(fraud < lowest_val) | (fraud > highest_val)].index
    return outlier_index
```

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

이상치 데이터 제거 후 모델 학습/예측/평가

```
# get_processed_df()를 로그 변환 후 V14 피처의 이상치 데이터를 삭제하는 로직으로 변경
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)
    # 이상치 데이터 삭제하는 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 ###')
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train,
tgt_test=y_test)

print('### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train,
tgt_test=y_test)
```

이상치 데이터 제거 후 모델 학습/예측/평가

로지스틱 회귀 예측 성능
오차행렬
[[85281 14]
 [48 98]]
정확도: 0.9993, 정밀도: 0.8750, 재현율: 0.6712, F1: 0.7597, AUC:0.8355

LightGBM 예측 성능
오차행렬
[[85290 5]
 [25 121]]
정확도: 0.9996, 정밀도: 0.9603, 재현율: 0.8288, F1: 0.8897, AUC:0.9144

이상치 제거 후 로지스틱 회귀의 경우 재현율이 0.6014 에서 0.6712 로, LightGBM의 경우 0.7635 에서 0.8288 로 크게 증가했다.

SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

SMOTE를 적용할 때는 반드시 학습 데이터 세트만 오버 샘플링을 해야함.

검증 데이터 세트나 테스트 데이터 세트를 오버 샘플링할 경우 결국은 원본 데이터 세트가 아닌 데이터 세트에서
검증 또는 테스트를 수행하기 때문에 올바른 검증/테스트가 될 수 없음.

좋은 SMOTE 패키지일수록 재현율 증가율은 높이고 정밀도 감소율은 낮출 수 있도록 효과적으로 데이터를 증식.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
X_train_over, y_train_over = smote.fit_resample(X_train, y_train)

print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', X_train.shape, y_train.shape)
print('SMOTE 적용 후 학습용 피처/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()) # 분포가 동일해진다.
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ((199362, 29), (199362,))
SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ((398040, 29), (398040,))
SMOTE 적용 후 레이블 값 분포:
1    199020
0    199020
Name: Class, dtype: int64
```

2배 가까이 데이터 증식

SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

```
lr_clf = LogisticRegression()
# ftr_train 과 tgt_train 인자 값이 SMOTE 증식된 X_train_over 와 y_train_over로 변경
get_model_train_eval(lr_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over,
tgt_test=y_test)
```

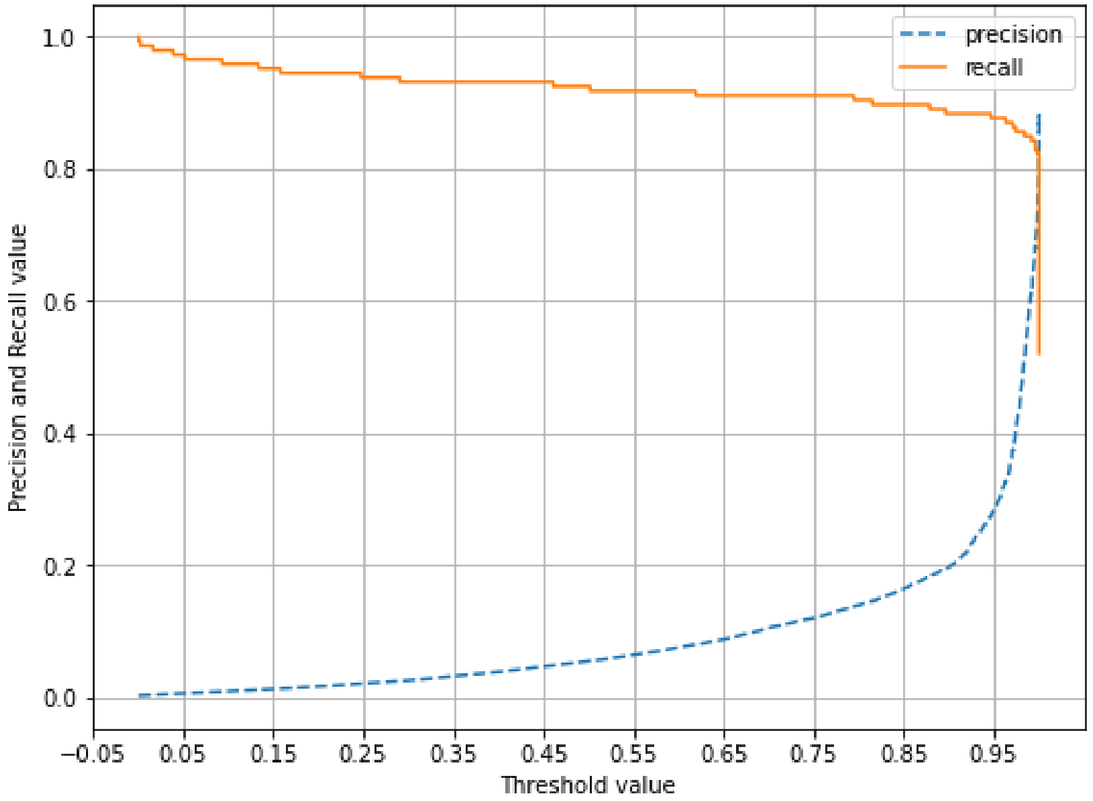
```
오차행렬
[[82937  2358]
 [   11   135]]
정확도: 0.9723, 정밀도: 0.0542, 재현율: 0.9247, F1: 0.1023, AUC:0.9485
```

로지스틱 회귀는 SMOTE 적용 후 재현율이 0.6712 에서 0.9247 로 크게 증가하지만

정밀도가 0.8750 에서 0.0542 로 크게 저하

SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

precision_recall_curve_plot(y_test, lr_clf.predict_proba(X_test)[:, 1]))



0.99 이하에서 재현율이 매우 좋고, 정밀도가 극단적으로 낮다가 0.99 이상에서 반대로 값이 증가하고 감소.

임곗값을 조정하더라도 민감도가 너무 심해 올바른 재현율/정밀도 성능을 얻을 수 없음.

SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

```
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_over, ftr_test=X_test, tgt_train=y_train_over,
tgt_test=y_test)
```

```
오차행렬
[[85283    12]
 [   22   124]]
정확도: 0.9996, 정밀도: 0.9118, 재현율: 0.8493, F1: 0.8794, AUC:0.9246
```

이상치만 제거한 경우와 비교해서 재현율이 0.8288 에서 0.8493 으로 높아졌고,

정밀도는 0.9603 에서 0.9118 로 낮아짐.

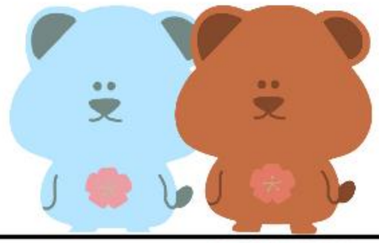
일반적으로 SMOTE 를 적용하면 재현율은 높아지지만 정밀도는 떨어짐.

Beginner Friendly CATBOOST with OPTUNA

2조 김정은

EWHA EURON

목차



EWHA EURON

#01 CatBoost

#1 CatBoost

Boosting 앙상블 기법을 사용하는 모델 중 하나
➔ 대부분이 범주형 변수로 이루어진 데이터셋에서 예측 성능이 우수한 것으로 알려져 있음
➔ Overfitting을 방지
➔ 다른 Boosting 기반 알고리즘과는 달리 범주형 변수를 특별하게 처리

Country	Eye color
Korea	Black
Korea	Black
Korea	Black
Norway	Blue
Norway	Blue
Norway	Blue

➔ Country가 정해지면 Eye Color도 정해짐!
➔ Country, Eye Color 모두 사용할 필요 없이, 둘 중 한 변수만 사용해도 가능
➔ 오히려 둘 다 사용하면 Dimension의 증가로 OverFitting 발생 가능
➔ CatBoost는 모델링에 사용할 변수를 하나만 선택, 이것이 자동적!
➔ *Categorical Feature Combination*

➔ 범주형 변수를 One-hot Encoding, Label Encoding 등 Encoding 작업을 하지 않고도 그대로 모델의 input으로 사용 가능
➔ 범주형 변수를 그대로 모델에 넣어주면 알아서 *Ordered Target Encoding* 진행(과거의 데이터를 이용해 현재의 데이터를 인코딩)
➔ 일반적인 Boosting 대신 *Ordered Boosting* 사용(Random Permutation)

#02 EDA(데이터 파악)

#1 데이터의 구성

In [2]:
```
pd.set_option('max_columns',100)
pd.set_option('max_rows',900)

pd.set_option('max_colwidth',200)

df = pd.read_csv('../input/heart-failure-prediction/heart.csv')
df.head()
```

Out[2]:

Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
M	ATA	140	289	0	Normal	172	N	0.0	Up	0
F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
M	ATA	130	283	0	ST	98	N	0.0	Up	0
F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
M	NAP	150	195	0	Normal	122	N	0.0	Up	0

<- Head( ) 를 이용해 상위 다섯 개의 데이터를 확인

In [3]:
```
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
```

Info( ) 를 이용하여
-> Null 데이터가 있는지 확인, Null 데이터 없음!

#02 EDA(데이터 파악)

#1 데이터의 구성

In [4]:
```
df.duplicated().sum()
```

Out[4]:
```
0
```

<- 중복된 값들의 존재 확인. 중복된 값들 없음!

In [5]:
```
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_Number',
'Missing_Percent'])
    return missing_values

missing(df)
```

Out[5]:

	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

-> NaN 등 누락 데이터가 있는지 확인!

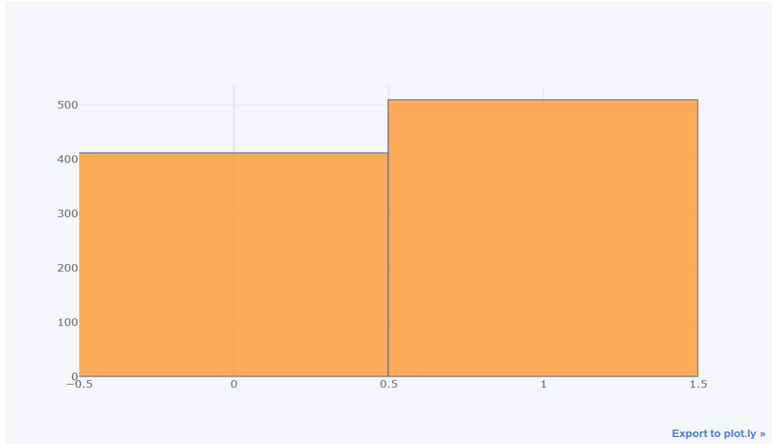
#03 Target 변수

#1 타겟 변수 설정

In [8]:
```
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: {ro
und(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)
```

In [9]:
```
df['HeartDisease'].iplot(kind='hist')
```



➔ 거의 55%의 환자가 심장병을 가지고 있음
➔ 508명의 환자가 심장병을 가지고 있음
➔ 거의 45%의 환자가 심장병을 가지고 있지 않음
➔ 410명의 환자는 심장병을 가지고 있지 않음

➔ 그래프화
➔ 약간의 불균형은 있지만 고려할 만한 수준은 아님
➔ ‘Accuracy’ 를 평가지표로 사용 가능

EWHA EURON

#04 Feature 변수

#1 수치적 특징(Numerical Features)

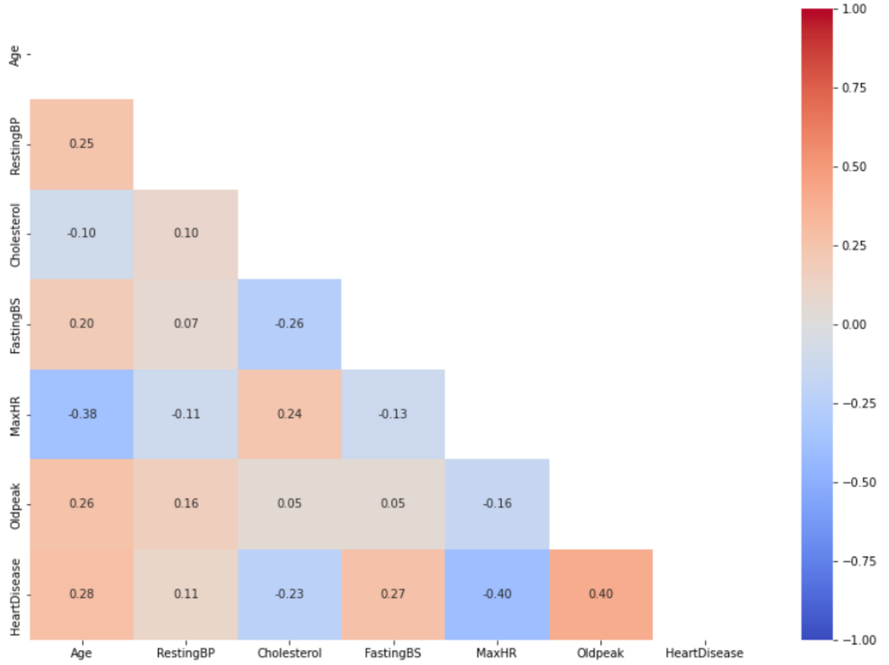
In [13]:
```
skew_limit = 0.75 # This is our threshold-limit to evaluate skewness. Overall below abs(1) seems acc
eptable for the linear models.
skew_vals = df[numerical].drop('FastingBS', axis=1).skew()
skew_cols= skew_vals[abs(skew_vals)> skew_limit].sort_values(ascending=False)
skew_cols
```

Out[13]:
```
Oldpeak    1.022872
dtype: float64
```

➔ Skewness가 높지 않음!
➔ Numerical Feature에 대한 분포는 Normal

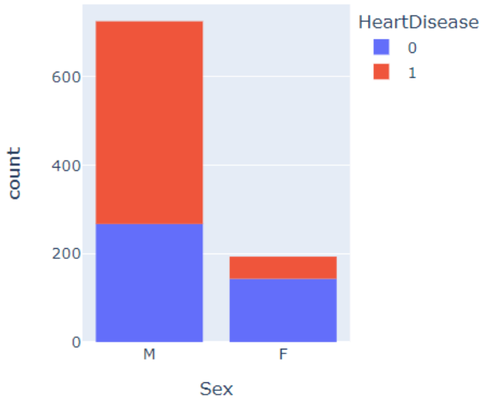
➔ Matrix에 기반하여 Numerical Feature과 Target Variable 사이의 약한 상관관계 관찰 가능
➔ 우울증 관련 수치는 심장 질환과 양의 상관 관계
➔ 최대 심박수는 심장 질환과 음의 상관 관계
➔ 콜레스테롤은 심장 질환과 음의 상관 관계

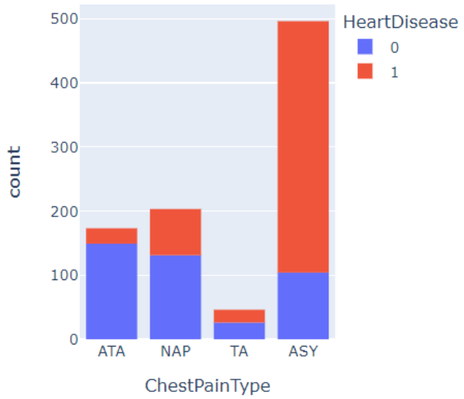
In [14]:
```
numerical1= df.select_dtypes('number').columns


matrix = np.triu(df[numerical1].corr())
fig, ax = plt.subplots(figsize=(14,10))
sns.heatmap (df[numerical1].corr(), annot=True, fmt= '.2f', vmin=-1, vmax=1, center=0, cmap='coolw
arm',mask=matrix, ax=ax);
```

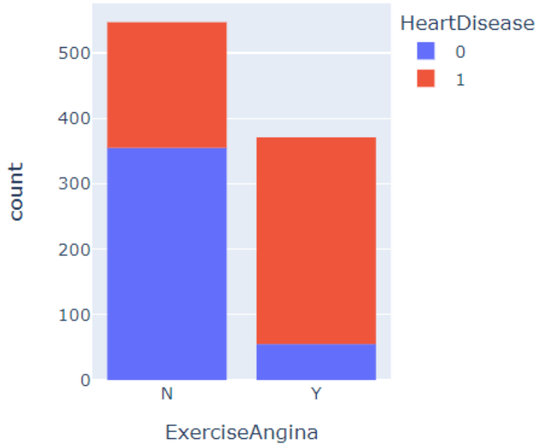


#04 Feature 변수

#1 Categorical Features　　모든 Categorical Feature들에 대하여 심장 질환을 가지고 있는 사람들의 분포







➔ 성별에 따라 남성이 여성보다 심장 질환에 걸릴 확률이 2.44배 높음
➔ ChestPain의 타입에 따라 분명한 차이를 관찰할 수 있음
➔ ASY를 가진 사람 : 무증상 흉통은 ATA 비정형 협심증을 가진 사람보다 심장 질환에 걸릴 확률이 거의 6배나 높음
➔ RestingECG : 휴식 심전도의 결과는 큰 차이가 없음
➔ RestingECG : ST-T파 이상이 있는 사람들은 다른 사람들보다 심장질환에 걸릴 가능성이 높음
➔ ExerciseAngina : Yes 라고 답한 쪽이 심장 질환을 가지고 있을 확률이 No라고 답한 쪽보다 2.4배 높음
➔ ST_Slope : Up쪽은 다른 두 부분에 비해 심장 질환에 걸릴 확률이 낮음

#05 Model Selection

#1 Baseline Model

➔ Dummy Classifier Model을 Base Model로 사용
➔ 그 다음, Logistic & Linear Discrimant & Kneighbors 및 Support Vector Machine 모델을 Scaler 유무에 관계없이 사용
➔ 그 다음, 앙상블 모델, AdaBoost, RandomForest, Gradient Boosting 및 Extra Tree 사용
➔ XGBoost, LightGBM & CatBoost 사용
➔ CatBoost에 대한 하이퍼 파라미터 튜닝 살펴봄

In [26]:

```
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)
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
```

```
model : DummyClassifier(constant=1, strategy='constant') and  accuracy score is : 0.5942
```

Out[26]:

	Accuracy
DummyClassifier	0.5942

-> Baseline Model

#05 Model Selection

#2 Model

-> Logistic & Linear Discriminant & SVC & KNN

In [27]:

```
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)}')

model_names = ['Logistic','LinearDiscriminant','SVM','KNeighbors']
result_df1 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df1
```

```
model : LogisticRegression(solver='liblinear') and  accuracy score is : 0.8841
model : LinearDiscriminantAnalysis() and  accuracy score is : 0.8696
model : SVC() and  accuracy score is : 0.7246
model : KNeighborsClassifier() and  accuracy score is : 0.7174
```

Out[27]:

	Accuracy
Logistic	0.8841
LinearDiscriminant	0.8696
SVM	0.7246
KNeighbors	0.7174

➔ 평가 지표는 Accuracy
➔ Logistic -> LinearDiscriminant -> SVM -> Kneighbors 순서로 Accuracy가 높음

EWHA EURON

#05 Model Selection

#2 Model

-> Logistic & Linear Discriminant & SVC & KNN with Scaler

In [28]:
```
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)}')

model_names = ['Logistic_scl','LinearDiscriminant_scl','SVM_scl','KNeighbors_scl']
result_df2 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df2
```

```
model : LogisticRegression(solver='liblinear') and  accuracy score is : 0.8804
model : LinearDiscriminantAnalysis() and  accuracy score is : 0.8696
model : SVC() and  accuracy score is : 0.8841
model : KNeighborsClassifier() and  accuracy score is : 0.8841
```

Out[28]:

	Accuracy
Logistic_scl	0.8804
LinearDiscriminant_scl	0.8696
SVM_scl	0.8841
KNeighbors_scl	0.8841

➔ 평가 지표는 Accuracy

➔ SVM = Kneighbors -> Logistic -> LinearDiscriminant 순서로 Accuracy가 높음

➔ 예상대로, Scaler을 사용하면 KNN와 SVM 모두 이전보다 성능이 높아진다. -> Scaler 활용에 용이

EWHA EURON

#05 Model Selection

#3 Ensemble

-> Ensemble Models(AdaBoost & Gradient Boosting & Random Forest & Extra Trees

In [29]:

```
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)}')

model_names = ['Ada','Gradient','Random','ExtraTree']
result_df3 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df3
```

```
model : AdaBoostClassifier(random_state=0) and  accuracy score is : 0.8659
model : GradientBoostingClassifier(random_state=0) and  accuracy score is : 0.8768
model : RandomForestClassifier(random_state=0) and  accuracy score is : 0.8877
model : ExtraTreesClassifier(random_state=0) and  accuracy score is : 0.8804
```

Out[29]:

	Accuracy
Ada	0.8659
Gradient	0.8768
Random	0.8877
ExtraTree	0.8804

➔ 평가 지표는 Accuracy
➔ Random -> Extra Tree -> Gradient -> AdaBoost 순서로 Accuracy가 높음
➔ 정확도는 각 앙상블 모델별로 서로 가까움
➔ Random Forest 와 Extra Tree는 하이퍼 파라미터 튜닝으로 개선 가능

EWHA EURON

#05 Model Selection

#4 Boost

-> Famous Trio (XGBoost & LightGBM & CatBoost)

In [30]:

```
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))

model_names = ['XGBoost','LightGBM']
result_df4 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df4
```

```
[19:00:23] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation m
etric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicit
ly set eval_metric if you'd like to restore the old behavior.
```

Out[30]:

	Accuracy
XGBoost	0.8297
LightGBM	0.8732

➔ 평가 지표는 Accuracy
➔ Random -> Extra Tree -> Gradient -> AdaBoost 순서로 Accuracy가 높음
➔ CatBoost는 전처리 없이 범주형 변수를 처리할 수 있는 기능을 사용하여 단독으로 사용할 것
➔ 먼저 XGBoost와 LightGBM을 살펴봄

EWHA EURON

#05 Model Selection

#4 Boost

-> CatBoost

In [31]:

```
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))

model_names = ['Catboost_default']
result_df5 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df5
```

Out[31]:

	Accuracy
Catboost_default	0.8804

➔ Purpose : 분류 문제에 대한 모델 Training 및 Applying, Scikit-learn tools 와의 호환성을 제공. 기본 최적화된 목표는 다양한 조건에 따라 달라짐

➔ Logloss : 대상에 두 개의 다른 값만 있거나 target_border 매개 변수가 None이 아님

➔ MultiClass : 대상에 두 개의 다른 값이 있으며 border_count 매개 변수는 None

➔ Reference : https://catboost.ai/en/docs/concepts/python-reference_catboostclassifier

➔ CatBoost 조정 -> 최고 성능 확인

EWHA EURON

#05 Model Selection

#4 Boost

-> CatBoost

In [52]:
```
def objective(trial):
    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 = {
        "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",
    }

    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_i
ndices,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:")
    trial = study.best_trial

    print("  Value: {}".format(trial.value))

    print("  Params: ")
    for key, value in trial.params.items():
        print("    {}: {}".format(key, value))
```

```
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
```

➔ Objective : 과적합 감지 및 최적 모델 선택을 위한 Supported metrics

➔ Colsample_by level : Training 속도를 높이고 일반적으로 품질에 영향을 미치지 않음

➔ Depth : Tree의 깊이

➔ Bootstrap_type : 기본적으로 객체의 가중치를 샘플링하는 방법으로 설정되어 있음. Bagging의 Sample rate 값이 1보다 작으면 Training이 더 빠르게 수행됨

EWHA EURON

#05 Model Selection

#4 Boost

-> CatBoost (with 최적 파라미터)

In [41]:

```
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))

model_names = ['Catboost_tuned']
result_df6 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df6
```

```
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       112
           1       0.93      0.91      0.92       164

    accuracy                           0.91       276
   macro avg       0.90      0.91      0.91       276
weighted avg       0.91      0.91      0.91       276
```

Out[41]:

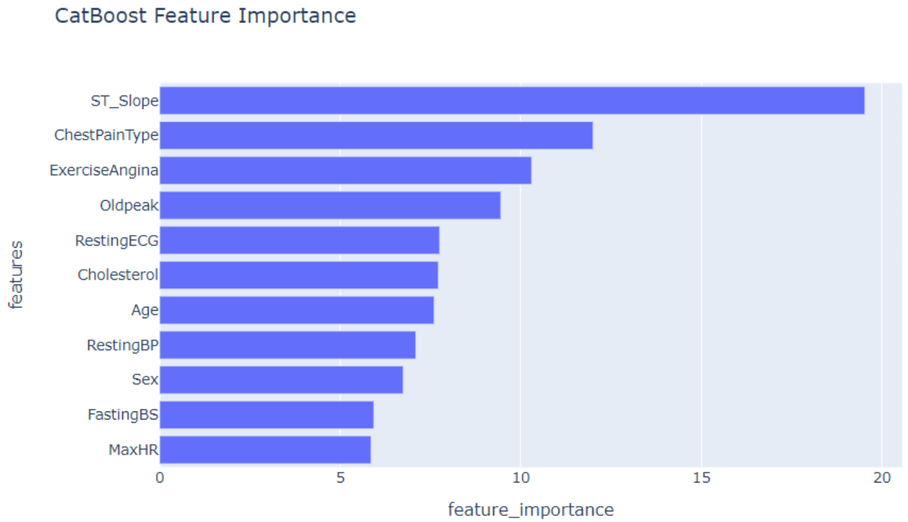
	Accuracy
Catboost_tuned	0.9094

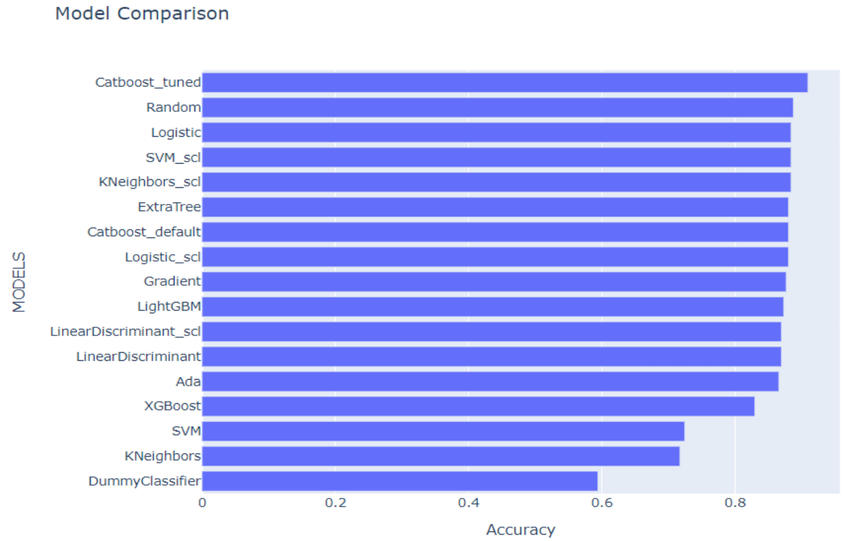
➔ Accuracy가 0.8804 에서 0.9094 까지 상승!

#05 Model Selection

#5 결론

-> CatBoost Feature Importance & Model Comparison & Conclusion

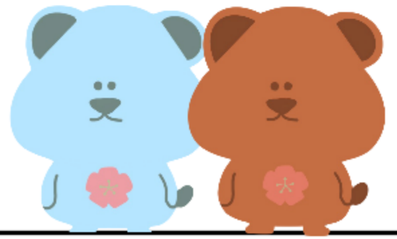




Conclusion

➔ 심장 질환 사례를 분류하기 위한 모델을 개발
➔ Detail 적인 Exploratory Analysis 수행
➔ 어떤 Metric을 사용할 건지 결정, Target, Feature 자세히 분석
➔ 범주형 변수를 숫자로 변환하여 모델에 사용, 데이터 유출을 방지하기 위해 Pipeline 사용
➔ 각 모델의 결과를 보고 가장 적합한 모델을 선택, CatBoost를 자세히 탐색
➔ Optuna를 통해 CatBoost의 하이퍼 파라미터 튜닝을 수행, 개선점 확인

THANK YOU



EWHA EURON