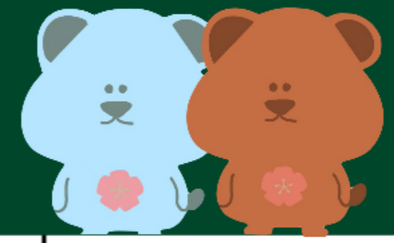


05. 분류 실습

1팀 박혜린, 엄지민, 조승연

EWHA EURON

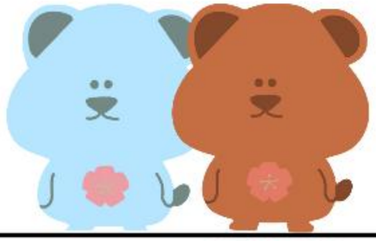
목차

#01 파머완 4.9장 '산탄테르 고객 만족 예측 '

1-1 이진 분류
1-2 데이터 전처리
1-3 모델링 – XGBoost
1-4 모델링 – LightGBM

#02 'Decision Tree and Random Forest Classifier Models'

2-1 Data Review
2-2 Checking the data inside & Visualization
2-3 Models
    1) Data Preparing
    2) Decision Tree
    3) Random Forst
2-4 Evaluation

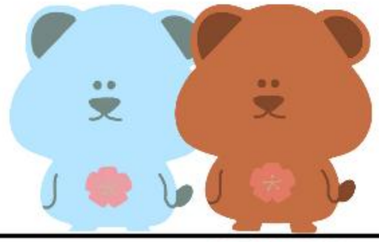


EWHA EURON

목차

#03 'Beginner Friendly Catboost with Optuna'

3-1Data Review
3-2 Exploratory Data Analysis& Visualization
3-3 Models – Base model
3-4 Models
3-5 feature_importance
3-6 Model Comparison



EWHA EURON

01. 파머완 4.9장 ‘산탄테르 고객 만족 예측’

EWHA EURON

#1-1 이진 분류 (binary classification)

#1-1 이진 분류

Dataset Description

You are provided with an anonymized dataset containing a large number of numeric variables. The "TARGET" column is the variable to predict. It equals one for unsatisfied customers and 0 for satisfied customers.

The task is to predict the probability that each customer in the test set is an unsatisfied customer.

클래스 레이블 명: TARGET
TARGET = 1 => 불만족,
TARGET = 0 => 만족

#1-2 데이터 전처리

#1 필요한 모듈 로딩하고 학습 데이터를 DataFrame으로 로딩하기

```
from google.colab import drive
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import warnings

# Google Drive 마운트
drive.mount('/content/drive')
file_path = '/content/drive/MyDrive/train_santander.csv'

warnings.filterwarnings('ignore')
cust_df = pd.read_csv(file_path, encoding='latin-1')
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	imp_op_var40_comer_ult1	imp_op_var40_comer_ult3	imp_op_var40_efect_ult1
0	1	2	23	0.0	0.0	0.0	0.0	0.0	0.0
1	3	2	34	0.0	0.0	0.0	0.0	0.0	0.0
2	4	2	23	0.0	0.0	0.0	0.0	0.0	0.0

3 rows × 371 columns

imp_op_var40_efect_ult3	...	saldo_medio_var33_hace2	saldo_medio_var33_hace3	saldo_medio_var33_ult1	saldo_medio_var33_ult3	saldo_medio_var44_hace2
0.0	...	0.0	0.0	0.0	0.0	0.0
0.0	...	0.0	0.0	0.0	0.0	0.0
0.0	...	0.0	0.0	0.0	0.0	0.0

saldo_medio_var44_hace3	saldo_medio_var44_ult1	saldo_medio_var44_ult3	var38	TARGET
0.0	0.0	0.0	39205.17	0
0.0	0.0	0.0	49278.03	0
0.0	0.0	0.0	67333.77	0

클래스 값 칼럼을 포함한 피처: 371개

#1-2 데이터 전처리

#2 info()로 데이터 분석

```
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
```

- 모든 feature 숫자형임
- NULL값 없음

#3 TARGET 속성값의 분포 - 고객의 만족, 불만족 비율

```
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))
```

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

대부분이 만족하고 불만족은 4%

EWHA EURON

#1-2 데이터 전처리

#4 피처값 분포 확인

```
cust_df.describe( )
```

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op_var40_comer_ult1	imp_op_var40_comer_ult3	imp_op_var40_efect_ult1	imp_op_var40_efect_ult3	...	saldo_medio_var33_hace2
count	76020.000000	76020.000000	76020.000000	76020.000000	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	6.472698	0.412946	0.567352	...	7.935824
std	43781.947379	39033.462364	12.956486	1614.757313	339.315831	546.266294	93.155749	153.737066	30.604864	36.513513	...	455.887218
min	1.000000	-999999.000000	5.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000	21024.810000	8237.820000	11073.570000	6600.000000	6600.000000	...	50003.880000

saldo_medio_var33_hace3	saldo_medio_var33_ult1	saldo_medio_var33_ult3	saldo_medio_var44_hace2	saldo_medio_var44_hace3	saldo_medio_var44_ult1	saldo_medio_var44_ult3	var38	TARGET
76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000	7.602000e+04	76020.000000
1.365146	12.215580	8.784074	31.505324	1.858575	76.026165	56.614351	1.172358e+05	0.039569
113.959637	783.207399	538.439211	2013.125393	147.786584	4040.337842	2852.579397	1.826646e+05	0.194945
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.163750e+03	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.787061e+04	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.064092e+05	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.187563e+05	0.000000
20385.720000	138831.630000	91778.730000	438329.220000	24650.010000	681462.900000	397884.300000	2.203474e+07	1.000000

va3 칼럼에서 min값이 -999999인 이유: NaN값이나 예외값을 -999999로 변환해서 발생한 일

#1-2 데이터 전처리

#5-1 var3에서 -999999 개수

```
cust_df['var3'].value_counts()
```

var3	count
2	74165
8	138
-999999	116
9	110
3	108
...	...
63	1
194	1
40	1
57	1
87	1

208 rows × 1 columns

dtype: int64

-999999 값이 116개 있음
var3은 숫자형이고,
다른 값에 비해 -999999는 편차가 너무 큼
2는 값이 74165개
=> 2로 바꾸는 것이 적절

#5-2 ID 피처 드롭 및 클래스 데이터셋과 피처 데이터셋 분리

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

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

```
피처 데이터 shape:(76020, 369)
```

ID feature는 단순 식별자 -> feature 드롭
학습, 성능 평가를 위해 피처 데이터셋과 레이블 데이터셋 분리

EWHA EURON

#1-2 데이터 전처리

#6 TARGET 값 분포도 확인

```
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('\n 테스트 세트 레이블 값 분포 비율')
print(y_test.value_counts()/test_cnt)
```

비대칭한 데이터셋이니까
Target 값 분포도가
비슷하게 추출되었는지 확인

```
학습 세트 Shape:(60816, 369), 테스트 세트 Shape:(15204, 369)
 학습 세트 레이블 값 분포 비율
TARGET
0    0.960964
1    0.039036
Name: count, dtype: float64

 테스트 세트 레이블 값 분포 비율
TARGET
0    0.9583
1    0.0417
Name: count, dtype: float64
```

학습 데이터셋과 테스트 데이터셋 모두 원본 데이터셋과 유사하게
불만족 4%로 만들어짐

EWHA EURON

#1-2 데이터 전처리

#7 early stopping의 검증 데이터셋으로 사용하기 위해 데이터셋 분리

```
# X_train, y_train을 다시 학습과 검증 데이터 세트로 분리.
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train,
                                            test_size=0.3, random_state=0)
```

- 학습용 데이터셋을 다시 학습, 검증 데이터셋으로 분리
- 학습용 데이터: 70%, 검증용 데이터셋: 30%

#1-3-1 모델링: XGBoost

#1 XGBoost

- 사이킷런래퍼 XGBClassifier 기반으로 학습 수행
- n_estimators = 500
- learning_rate = 0.05 (예제 수행 시마다 동일 예측 결과를 위해 설정)
- 성능 평가 지표: auc로
- 조기 중단 파라미터 = 100

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

# n_estimators는 500으로, learning_rate 0.05, random state는 예제 수행 시마다 동일 예측 결과를 위해 설정.
# 성능 평가 지표를 auc로, 조기 중단 파라미터는 100으로 설정하고 학습 수행... 버전에 맞게 수정했습니다;
XGBClassifier 생성시에 early_stopping_rounds와 eval_metric 선언
xgb_clf = XGBClassifier(n_estimators=500, learning_rate=0.05, random_state=156,
early_stopping_rounds = 100, eval_metric='auc')
xgb_clf.fit(X_tr, y_tr, eval_set=[(X_tr, y_tr), (X_val, y_val)])

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

```
[207]   validation_0-auc:0.91002        validation_1-auc:0.83262
[208]   validation_0-auc:0.91004        validation_1-auc:0.83261
[209]   validation_0-auc:0.91010        validation_1-auc:0.83256
[210]   validation_0-auc:0.91019        validation_1-auc:0.83259
[211]   validation_0-auc:0.91022        validation_1-auc:0.83258
ROC AUC: 0.8415
```

EWHA EURON

#1-3-1 모델링: XGBoost

#2 하이퍼 파라미터 검색 공간 설정

- max_depth: 5 ~ 15, 1간격
- min_child_weight: 1 ~ 6, 1간격
- colsample_bytree: 0.5 ~ 0.95
- learning_rate: 0.01 ~ 0.2, 정규 분포된 값으로 검색

```
from hyperopt import hp
xgb_search_space = {'max_depth': hp.quniform('max_depth', 5, 15, 1),
                    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
                    'colsample_bytree': hp.uniform('colsample_bytree', 0.5, 0.95),
                    'learning_rate': hp.uniform('learning_rate', 0.01, 0.2)
}
```

#1-3 모델링: XGBoost

#3 목적 함수 생성

```python
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score

# 목적 함수 설정.
# 추후 fmin()에서 입력된 search_space값으로 XGBClassifier 교차 검증 학습 후 -1* roc_auc 평균 값을 반환.
def objective_func(search_space):
    xgb_clf = XGBClassifier(n_estimators=100, max_depth=int(search_space['max_depth']),
                            min_child_weight=int(search_space['min_child_weight']),
                            colsample_bytree=search_space['colsample_bytree'],
                            learning_rate=search_space['learning_rate'],
                            early_stopping_rounds=30, eval_metric='auc' # 버전에 맞게 수정
                           )
    # 3개 k-fold 방식으로 평가된 roc_auc 지표를 담는 list
    roc_auc_list= []

    # 3개 k-fold방식 적용
    kf = KFold(n_splits=3)
    # X_train을 다시 학습과 검증용 데이터로 분리
    for tr_index, val_index in kf.split(X_train):
        # kf.split(X_train)으로 추출된 학습과 검증 index값으로 학습과 검증 데이터 세트 분리
        X_tr, y_tr = X_train.iloc[tr_index], y_train.iloc[tr_index]
        X_val, y_val = X_train.iloc[val_index], y_train.iloc[val_index]
        # early stopping은 30회로 설정하고 추출된 학습과 검증 데이터로 XGBClassifier 학습 수행.
        xgb_clf.fit(X_tr, y_tr, eval_set=[(X_tr, y_tr), (X_val, y_val)])

        # 1로 예측한 확률값 추출후 roc auc 계산하고 평균 roc auc 계산을 위해 list에 결과값 담음.
        score = roc_auc_score(y_val, xgb_clf.predict_proba(X_val)[:, 1])
        roc_auc_list.append(score)

    # 3개 k-fold로 계산된 roc_auc값의 평균값을 반환하되,
    # HyperOpt는 목적함수의 최소값을 위한 입력값을 찾으므로 -1을 곱한 뒤 반환.
    return -1 * np.mean(roc_auc_list)
```

- XGBoost와 LightGBM 에서는 교차 검증 시 early_stopping 미지원
  => K Fold 방식으로 구현
- 3 Fold 교차 검증으로 ROC-AUC 값 반환
- HyperOpt는 목적 함수의 반환값이 최소가 될 수 있는 최적의 값을 찾음
  => -1 * roc_auc 평균값을 반환함

EWHA EURON

#1-3 모델링: XGBoost

#4 fmin 함수를 호출해 max_eval = 50 까지 반복하면서 최적의 하이퍼 파라미터 도출

```
from hyperopt import fmin, tpe, Trials

trials = Trials()

# fmin()함수를 호출. max_evals지정된 횟수만큼 반복 후 목적함수의 최소값을 가지는 최적 입력값 추출.
best = fmin(fn=objective_func,
            space=xgb_search_space,
            algo=tpe.suggest,
            max_evals=50, # 최대 반복 횟수를 지정합니다.
            trials=trials, rstate=np.random.default_rng(seed=30))

print('best:', best)
```

```
100%|██████████| 50/50 [49:02<00:00, 58.85s/trial, best loss: -0.8377355574041188]
best: {'colsample_bytree': np.float64(0.8461753280946572), 'learning_rate': np.float64(0.11761398464887646), 'max_depth': np.float64(6.0), 'min_child_weight': np.float64(6.0)}
```

EWHA EURON

#1-3 모델링: XGBoost

#5 XGBClassifier 재학습시키고, 테스트 데이터셋에서 ROC AUC 측정

```
# n_estimators를 500증가 후 최적으로 찾은 하이퍼 파라미터를 기반으로 학습과 예측 수행.
xgb_clf = XGBClassifier(n_estimators=500, learning_rate=round(best['learning_rate'],
5),
                        max_depth=int(best['max_depth']),
min_child_weight=int(best['min_child_weight']),
                        colsample_bytree=round(best['colsample_bytree'], 5),
                        early_stopping_rounds=100, eval_metric="auc"
                       )

# evaluation metric을 auc로, early stopping은 100 으로 설정하고 학습 수행.
xgb_clf.fit(X_tr, y_tr,eval_set=[(X_tr, y_tr), (X_val, y_val)])

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

```
[120]   validation_0-auc:0.90184        validation_1-auc:0.83259
[121]   validation_0-auc:0.90235        validation_1-auc:0.83264
[122]   validation_0-auc:0.90246        validation_1-auc:0.83252
[123]   validation_0-auc:0.90297        validation_1-auc:0.83251
[124]   validation_0-auc:0.90316        validation_1-auc:0.83264
ROC AUC: 0.8431
```

- 0.8415에서 0.8431로 약간 개선됨
- XGBoost는 GBM보다는 빠르지만 수행시간이 오래 걸림
- 앙상블 계열 알고리즘은 overfitting, noise에 뛰어남

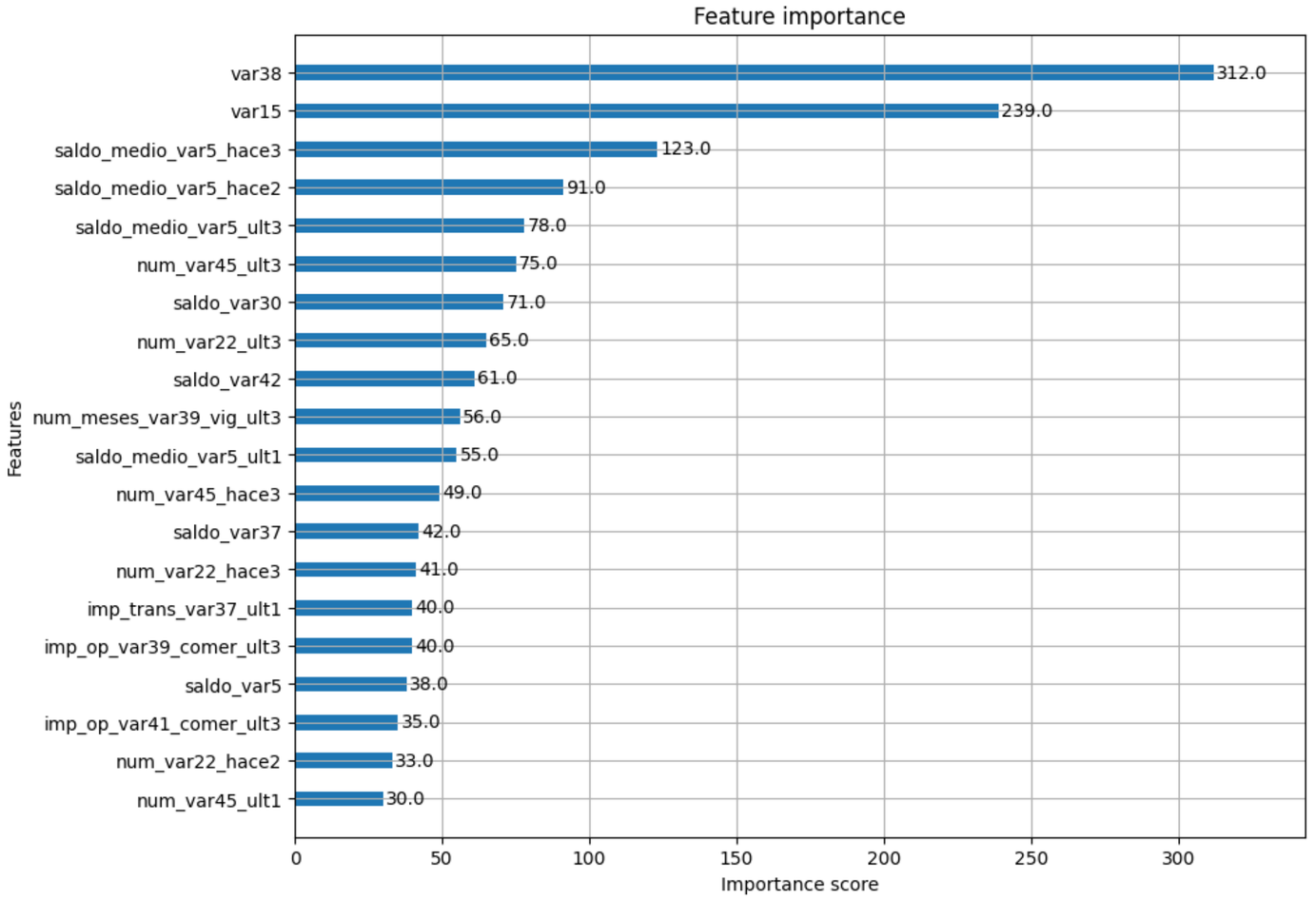
EWHA EURON

#1-3 모델링: XGBoost

#6 튜닝된 모델에서 각 피처의 중요도 시각화

```
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)
```

Xgboost 모듈의 plot_importance() 메소드 사용



#1-4 모델링: LightGBM

#1 학습, 검증 데이터셋 이용해 eval_set으로 학습 후 테스트 데이터텟으로 평가된 ROC-AUC 값 확인하기

```
from lightgbm import LGBMClassifier
# 버전에 맞게 수정
lgbm_clf = LGBMClassifier(n_estimators=500, early_stopping_rounds=100, eval_metric="auc")

eval_set=[(X_tr, y_tr), (X_val, y_val)]
lgbm_clf.fit(X_tr, y_tr, 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))
```

```
[LightGBM] [Warning] Unknown parameter: eval_metric
[LightGBM] [Warning] early_stopping_round is set=100, early_stopping_rounds=100 will be ignored. Current value: early_stopping_round=100
[LightGBM] [Warning] Unknown parameter: eval_metric
[LightGBM] [Info] Number of positive: 1658, number of negative: 40913
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.157291 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 13308
[LightGBM] [Info] Number of data points in the train set: 42571, number of used features: 242
[LightGBM] [Warning] Unknown parameter: eval_metric
[LightGBM] [Warning] early_stopping_round is set=100, early_stopping_rounds=100 will be ignored. Current value: early_stopping_round=100
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.038947 -> initscore=-3.205836
[LightGBM] [Info] Start training from score -3.205836
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[42]    training's binary_logloss: 0.112183     valid_1's binary_logloss: 0.13527
[LightGBM] [Warning] Unknown parameter: eval_metric
ROC AUC: 0.8384
```

- XGBoost와 동일한 n_estimator, early_stopping_rounds
- leaf-wise로 트리 분리

=> XGBoost보다 학습 시간 빠름

EWHA EURON

#1-4 모델링: LightGBM

#2 하이퍼 파라미터 공간 설정

```
lgbm_search_space = {'num_leaves': hp.quniform('num_leaves', 32, 64, 1),
                     'max_depth': hp.quniform('max_depth', 100, 160, 1),
                     'min_child_samples': hp.quniform('min_child_samples', 60, 100, 1),
                     'subsample': hp.uniform('subsample', 0.7, 1),
                     'learning_rate': hp.uniform('learning_rate', 0.01, 0.2)
                    }
```

num_leaves = 2^(max_depth)이면, depth_wise tree와 같은 수의 leaves를 가짐
=> depth_wise tree보다 더 적게 해야 overfitting 줄일 수 있음

#1-4 모델링: LightGBM

#3 목적 함수 생성

```
def objective_func(search_space):
    lgbm_clf =  LGBMClassifier(n_estimators=100, num_leaves=int(search_space['num_leaves']),
                               max_depth=int(search_space['max_depth']),
                               min_child_samples=int(search_space['min_child_samples']),
                               subsample=search_space['subsample'],
                               learning_rate=search_space['learning_rate'],
                                # early stopping은 30회로 설정
                               early_stopping_rounds=30, eval_metric="auc",)
    # 3개 k-fold 방식으로 평가된 roc_auc 지표를 담는 list
    roc_auc_list = []

    # 3개 k-fold방식 적용
    kf = KFold(n_splits=3)
    # X_train을 다시 학습과 검증용 데이터로 분리
    for tr_index, val_index in kf.split(X_train):
        # kf.split(X_train)으로 추출된 학습과 검증 index값으로 학습과 검증 데이터 세트 분리
        X_tr, y_tr = X_train.iloc[tr_index], y_train.iloc[tr_index]
        X_val, y_val = X_train.iloc[val_index], y_train.iloc[val_index]

        # 추출된 학습과 검증 데이터로 XGBClassifier 학습 수행.
        lgbm_clf.fit(X_tr, y_tr,eval_set=[(X_tr, y_tr), (X_val, y_val)])

        # 1로 예측한 확률값 추출후 roc auc 계산하고 평균 roc auc 계산을 위해 list에 결과값 담음.
        score = roc_auc_score(y_val, lgbm_clf.predict_proba(X_val)[:, 1])
        roc_auc_list.append(score)

    # 3개 k-fold로 계산된 roc_auc값의 평균값을 반환하되,
    # HyperOpt는 목적함수의 최소값을 위한 입력값을 찾으므로 -1을 곱한 뒤 반환.
    return -1*np.mean(roc_auc_list)
```

- XGBoost와의 차이점: LGBMCLassifier 객체 생성 부분

#1-4 모델링: LightGBM

#4 fmin()을 호출해 최적 하이퍼 파라미터 도출

```
from hyperopt import fmin, tpe, Trials

trials = Trials()

# fmin()함수를 호출. max_evals지정된 횟수만큼 반복 후 목적함수의 최소값을 가지는 최적 입력값 추출.
best = fmin(fn=objective_func, space=lgbm_search_space, algo=tpe.suggest,
            max_evals=50, # 최대 반복 횟수를 지정합니다.
            trials=trials, rstate=np.random.default_rng(seed=30))

print('best:', best)
```

```
Early stopping, best iteration is:
[45]    training's binary_logloss: 0.113185     valid_1's binary_logloss: 0.136743
[LightGBM] [Warning] Unknown parameter: eval_metric
100%|██████████| 50/50 [08:20<00:00, 10.01s/trial, best loss: -0.835974683012816]
best: {'learning_rate': np.float64(0.060410380599969726), 'max_depth': np.float64(122.0), 'min_child_samples': np.float64(87.0), 'num_leaves': np.float64(38.0), 'subsample': np.float64(0.7023057985957527)}
```

#1-4 모델링: LightGBM

#5 하이퍼 파라미터를 이용해 LightGBM 학습 후 테스트데이터셋에서 ROC-AUC 평가하기

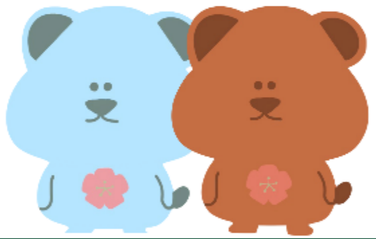
```
from hyperopt import fmin, tpe, Trials

trials = Trials()

# fmin()함수를 호출. max_evals지정된 횟수만큼 반복 후 목적함수의 최소값을 가지는 최적 입력값 추출.
best = fmin(fn=objective_func, space=lgbm_search_space, algo=tpe.suggest,
            max_evals=50, # 최대 반복 횟수를 지정합니다.
            trials=trials, rstate=np.random.default_rng(seed=30))

print('best:', best)
```

```
[LightGBM] [Warning] Unknown parameter: eval_metric
[LightGBM] [Warning] early_stopping_round is set=100, early_stopping_rounds=100 will be ignored. Current value: early_stopping_round=100
[LightGBM] [Warning] Unknown parameter: eval_metric
[LightGBM] [Info] Number of positive: 1658, number of negative: 40913
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.060481 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 12898
[LightGBM] [Info] Number of data points in the train set: 42571, number of used features: 192
[LightGBM] [Warning] Unknown parameter: eval_metric
[LightGBM] [Warning] early_stopping_round is set=100, early_stopping_rounds=100 will be ignored. Current value: early_stopping_round=100
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.038947 -> initscore=-3.205836
[LightGBM] [Info] Start training from score -3.205836
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[62]    training's binary_logloss: 0.115277     valid_1's binary_logloss: 0.134855
[LightGBM] [Warning] Unknown parameter: eval_metric
ROC AUC: 0.8409
```

- 정확도: 0.8409
- 튜닝 전인 0.8384보다 약간 상승
- 정확도가 0.8431인 XGBoost과
  성능 차이는 작지만 수행 시간은 훨씬 단축됨

EWHA EURON

02. ‘Decision Tree and Random Forest Classifier Models’



EWHA EURON

#2-1 Data Review

#1 불러오고 읽기

```
data = pd.read_csv("drug200.csv")
data
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY
...	...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

200 rows × 6 columns

#2 Information

```
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
```

#2-1 Data Review

#3 데이터 통계 확인

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

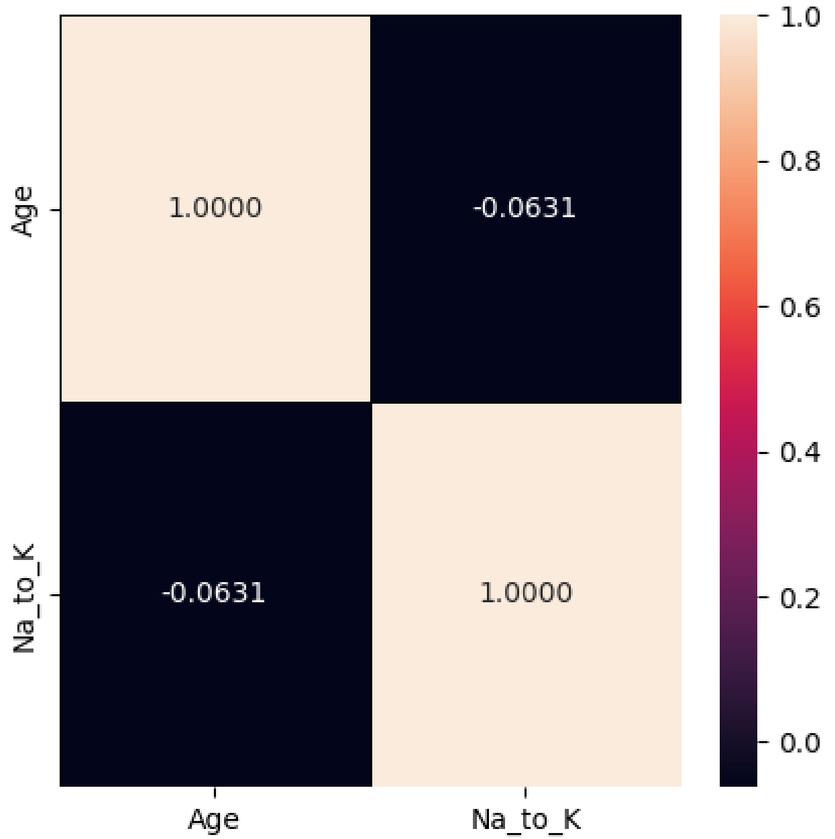
#2-1 Data Review

#4 상관관계 보기

```
data.corr()
data.corr(numeric_only=True)
```

	Age	Na_to_K
Age	1.000000	-0.063119
Na_to_K	-0.063119	1.000000

```
f, ax = plt.subplots(figsize = (5,5))
sns.heatmap(data.corr(numeric_only=True), annot = True,
linewidths=0.5, linecolor = "black", fmt = ".4f", ax = ax)
plt.show()
```



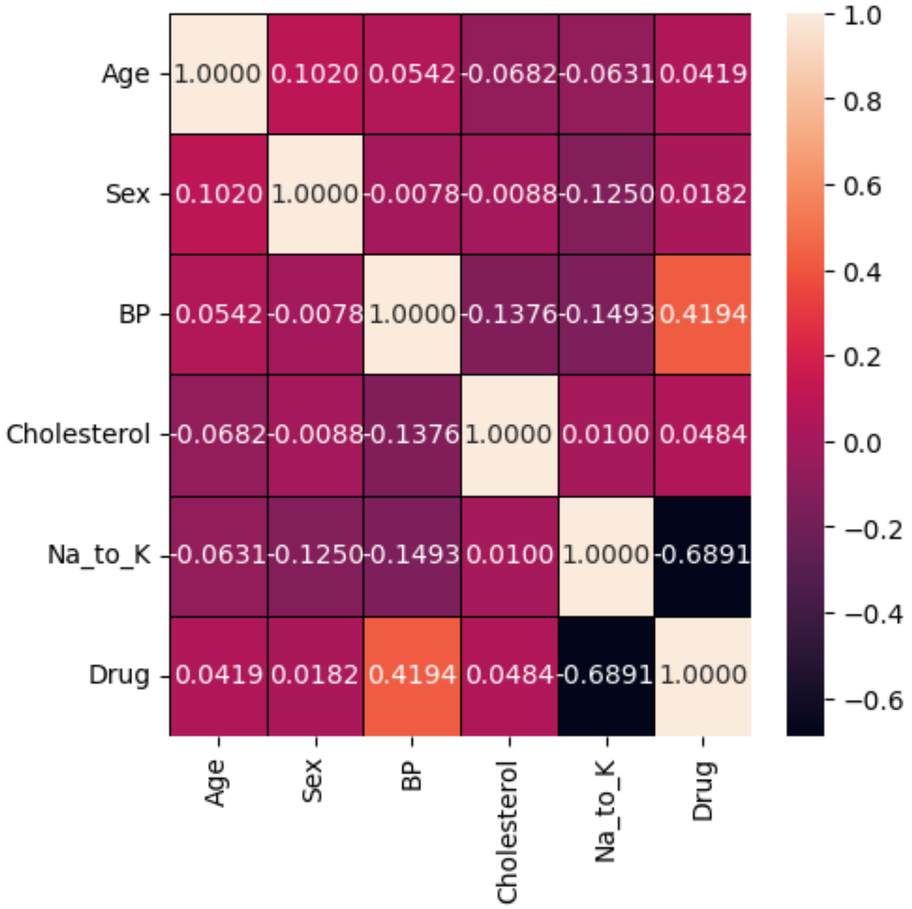
#2-1 Data Review

#4 상관관계 보기

```
df_encoded = data.copy()

from sklearn.preprocessing import LabelEncoder
for col in df_encoded.columns:
    if df_encoded[col].dtype == 'object':
        le = LabelEncoder()
        df_encoded[col] =
le.fit_transform(df_encoded[col])

df_encoded.corr()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
Age	1.000000	0.102027	0.054212	-0.068234	-0.063119	0.041856
Sex	0.102027	1.000000	-0.007814	-0.008811	-0.125008	0.018239
BP	0.054212	-0.007814	1.000000	-0.137552	-0.149312	0.419397
Cholesterol	-0.068234	-0.008811	-0.137552	1.000000	0.010000	0.048415
Na_to_K	-0.063119	-0.125008	-0.149312	0.010000	1.000000	-0.689051
Drug	0.041856	0.018239	0.419397	0.048415	-0.689051	1.000000

```
f, ax = plt.subplots(figsize = (5,5))
sns.heatmap(df_encoded.corr(), annot = True, linewidths=0.5,
linecolor = "black", fmt = ".4f", ax = ax)
plt.show()
```



#2-2 Checking the data inside & Visualization

#1 Columns 목록

```
data.columns
```

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

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY
...	...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX

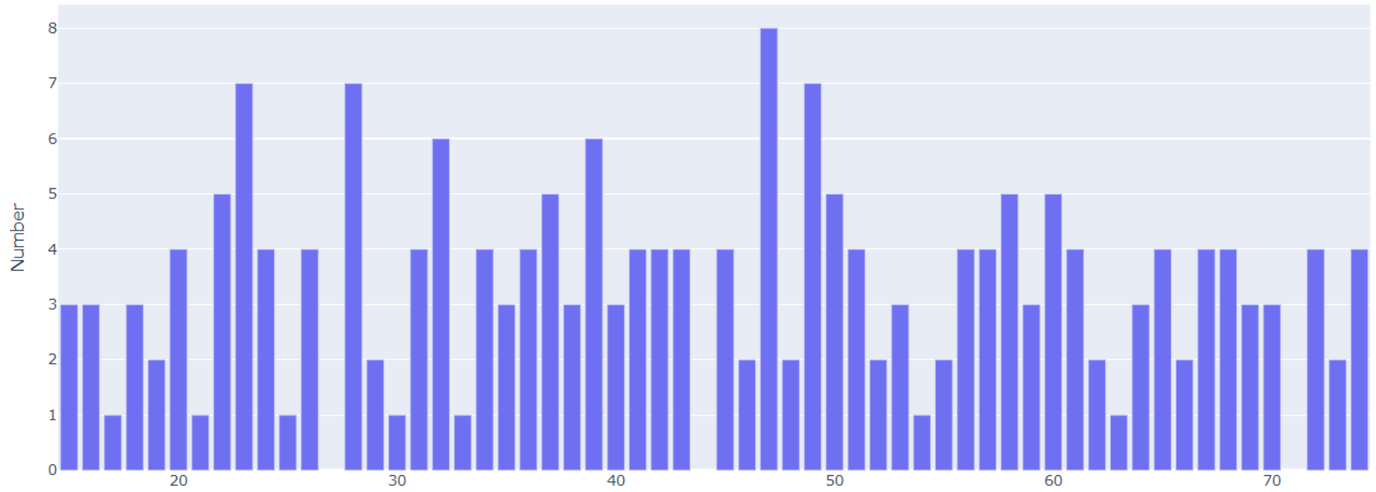
#2-2 Checking the data inside & Visualization

#2 Age

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

Age	count
47	8
23	7
28	7
49	7
32	6
39	6
50	5
60	5
22	5
37	5
58	5
43	4
41	4
57	4
65	4
61	4
74	4
45	4
56	4
24	4
26	4
67	4
68	4
31	4
34	4
51	4
42	4
20	4
72	4
36	4
69	3
18	3
16	3
35	3
59	3
64	3
53	3
38	3
70	3
15	3
40	3
55	2
52	2
48	2
29	2
73	2
46	2
66	2
62	2
19	2
63	1
33	1
17	1

```
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()
```



#2-2 Checking the data inside & Visualization

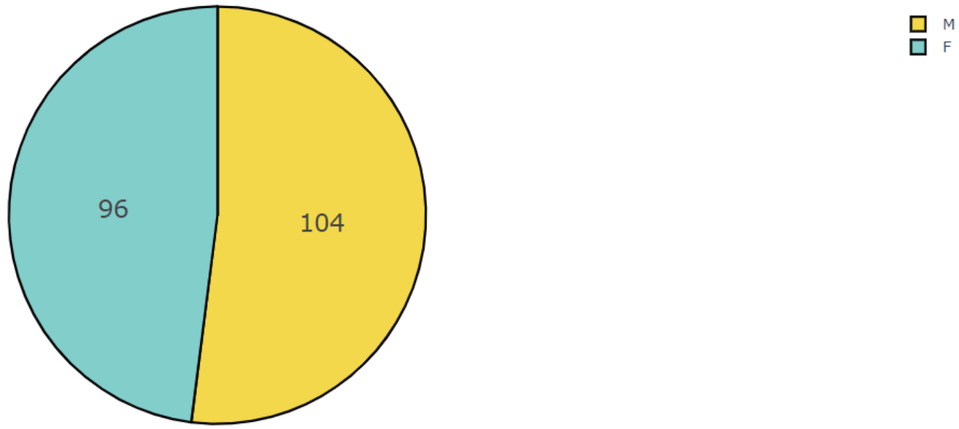
#3 Sex

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

Sex	count
M	104
F	96

dtype: int64

```
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()
```



#2-2 Checking the data inside & Visualization

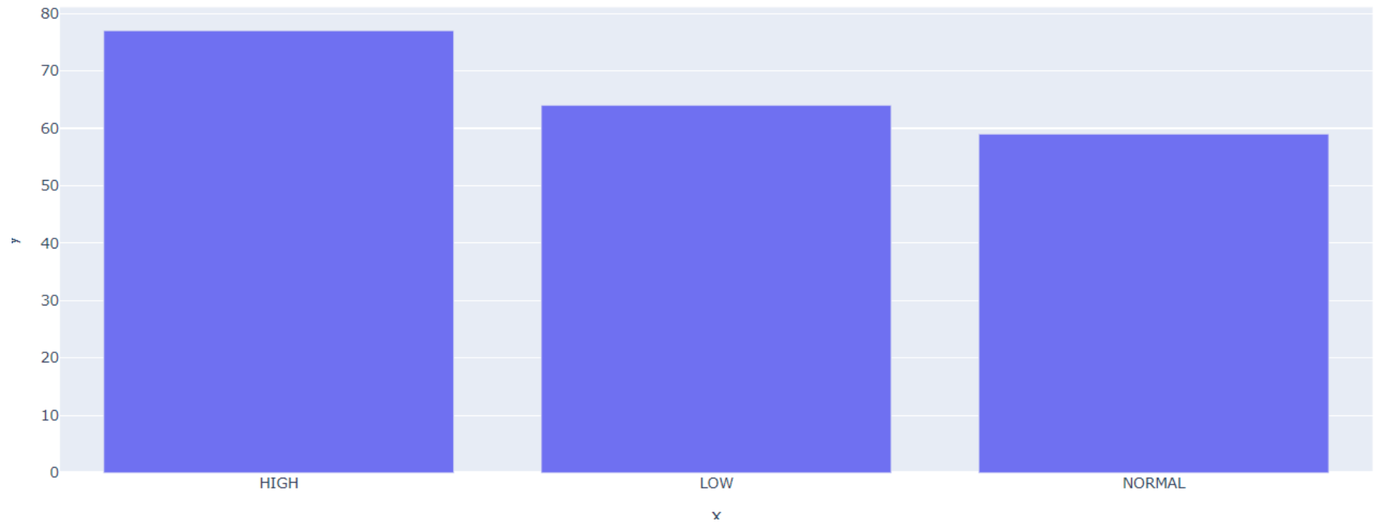
#4 Blood Pressure

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

BP	
HIGH	77
LOW	64
NORMAL	59

dtype: int64

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

fig.show()
```



#2-2 Checking the data inside & Visualization

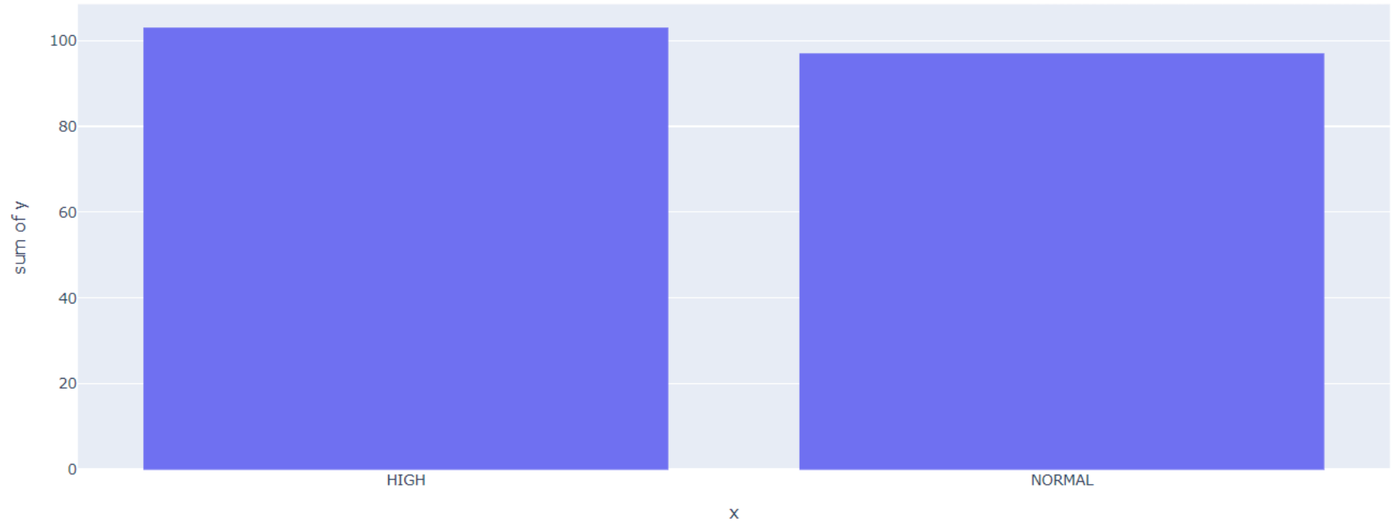
#5 Cholesterol

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

Cholesterol	
HIGH	103
NORMAL	97

dtype: int64

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



#2-2 Checking the data inside & Visualization

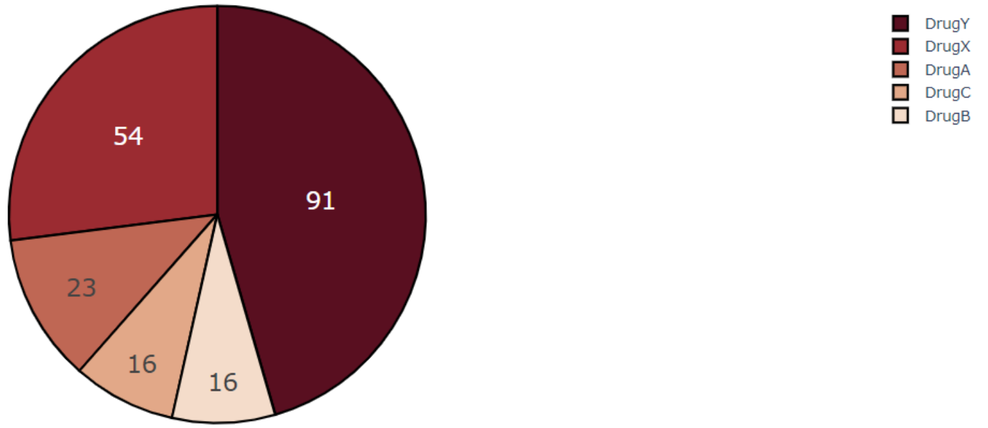
#6 Drug

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

Drug	
DrugY	91
drugX	54
drugA	23
drugC	16
drugB	16

dtype: int64

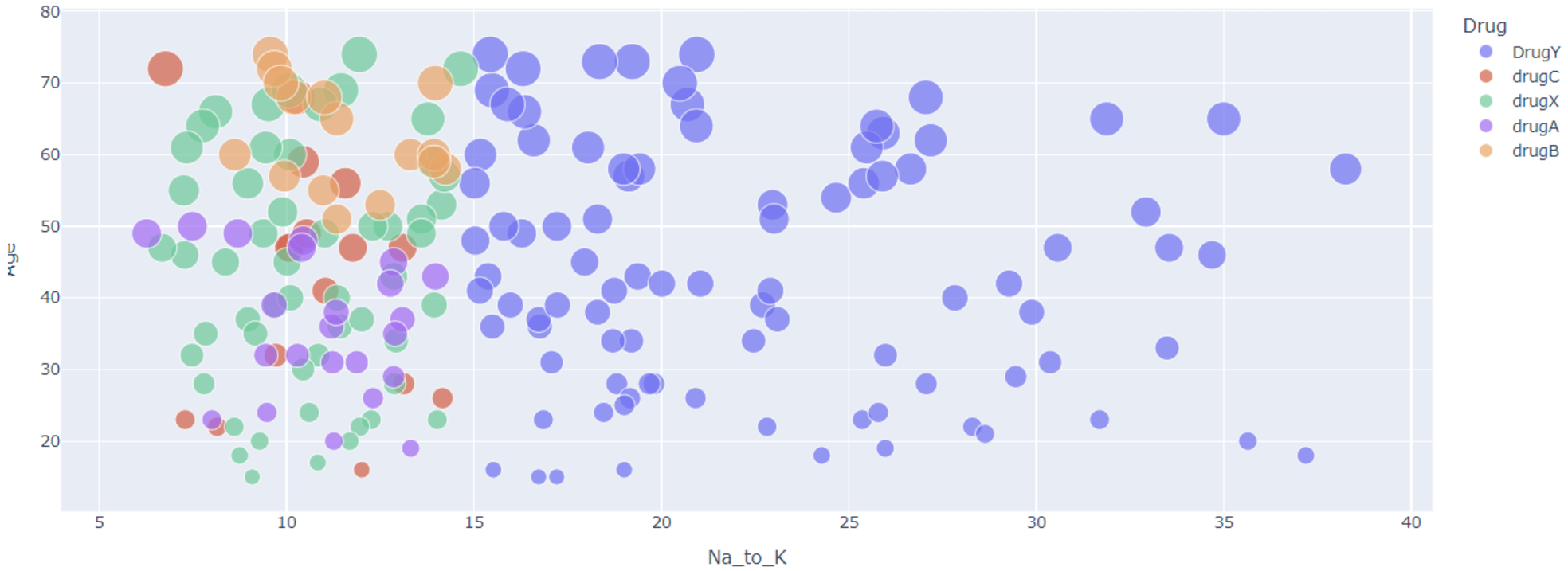
```
fig = go.Figure(data =
[go.Pie(labels=["DrugY","DrugX","DrugA","DrugC","DrugB"],
values=[91,54,23,16,16])])

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()
```



#2-2 Checking the data inside & Visualization

#7 relationship between age and Na_To_K(Na to Potassium Ration)

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

fig.show()
```



#2-3-1 Models – Data Preparing

#1 Age

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

#2 Blood Pressure

```
# LOW = 2
# NORMAL = 1
# HIGH = 0

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
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
dataclass['BP'] = le.fit_transform(dataclass['BP'])
```

#2-3-1 Models – Data Preparing

#3 Cholesterol

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

#4 Drug Type

```
# DrugY = 4
# drugX = 3
# drugA = 2
# drugC = 1
# drugB = 0

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
```

#2-3-1 Models – Data Preparing

#5 자료형 바꾸기

```
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    int64  
 3   Cholesterol  200 non-null    int64  
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    int64  
dtypes: float64(1), int64(5)
memory usage: 9.5 KB
```

#2-3-1 Models – Data Preparing

#6 train-test data split

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

#y_data
y_data = dataclass.Drug.values
```

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

```
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)
```

#2-3-2 Models – Decision Tree

#1 일반 DT

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


dtc = DecisionTreeClassifier()


# Fit the model
dtc.fit(x_train, y_train)


# Predict the x_test
predict = dtc.predict(x_test)


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

결과 : The accuracy of the Decision Tree is 0.9666666666666667

96.67 %

#2-3-2 Models – Decision Tree

#2 DT with gini

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

   # Fit the model
   DTC_gini.fit(x_train, y_train)
2  y_pred_gini = DTC_gini.predict(x_test)

4  y_pred_train_gini = DTC_gini.predict(x_train)

3  print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_score(y_test, y_pred_gini)))

5  print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train_gini)))
```

```
# Print the scores on training and test set

print('Training set score: {:.4f}'.format(DTC_gini.score(x_train, y_train)))


print('Test set score: {:.4f}'.format(DTC_gini.score(x_test, y_test)))
```

91.43%

90.00%

EWHA EURON

#2-3-2 Models – Decison Tree

#3 DT with entropy

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

   # Fit the model
   DTC_en.fit(x_train, y_train)
2  y_pred_en = DTC_en.predict(x_test)

3  print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy_score(y_test, y_pred_en))

4   y_pred_train_en = DTC_en.predict(x_train)

5  print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train_en)))
```

```
# Print the scores on training and test set

print('Training set score: {:.4f}'.format(DTC_en.score(x_train, y_train)))


print('Test set score: {:.4f}'.format(DTC_en.score(x_test, y_test)))
```

91.43%

90.00%

EWHA EURON

#2-3-3 Models – Random Forest

#1 일반 RF

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state = 0)

# Fit the model
rfc.fit(x_train, y_train)

# Predict the model
predict = rfc.predict(x_test)

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

95.00 %

#2 RF with n_estimators=100

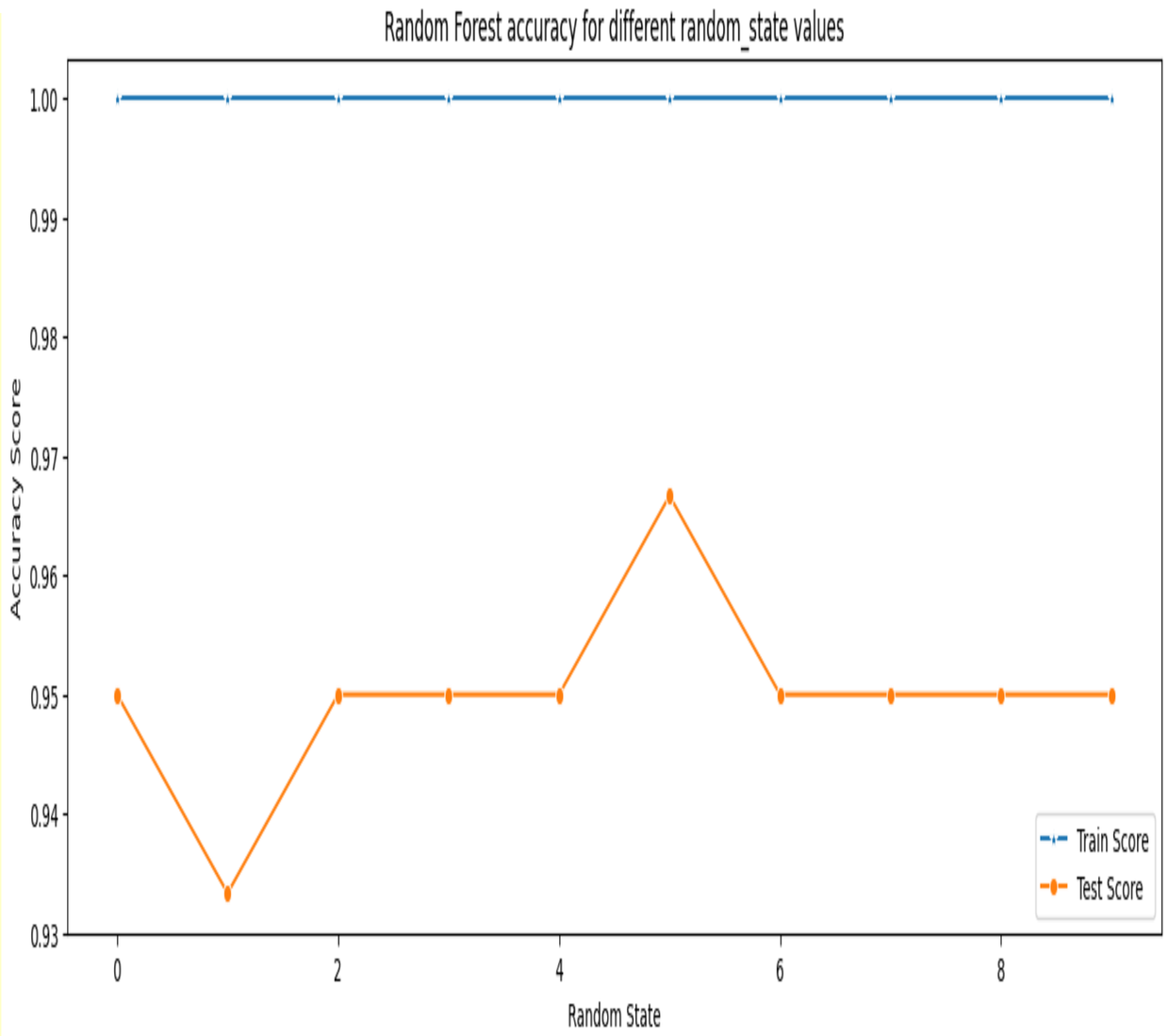
```
from sklearn.ensemble import RandomForestClassifier

rfc_100 = RandomForestClassifier(n_estimators=100,
random_state=0)

# Fit the model
rfc_100.fit(x_train, y_train)

# Predict the model
predict = rfc_100.predict(x_test)

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

95.00 %

#2-3-3 Models – Random Forest

#3-1 Finding best random state

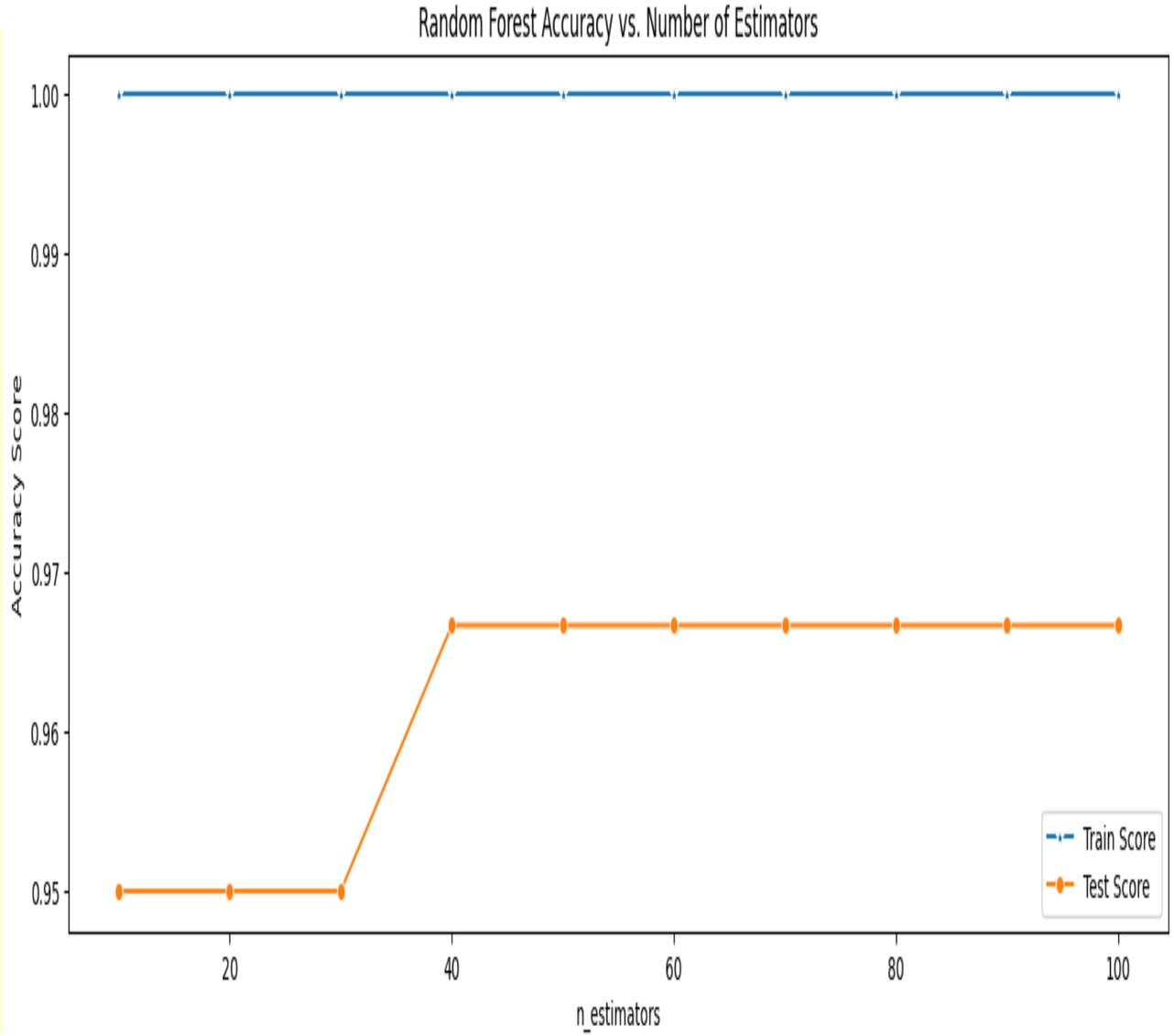
```
test_score_list = []
train_score_list = []


for i in range(0,10):
    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')
plt.xlabel("Random State")
plt.ylabel("Accuracy Score")
plt.title("Random Forest accuracy for different random_state
values")
plt.show()
```



#2-3-3 Models – Random Forest

#3-2 Finding best n_estimators

```
test_score_list = []
train_score_list = []
list_n_estimators = [10,20,30,40,50,60,70,80,90,100]
for n in list_n_estimators:
    rfc3 = RandomForestClassifier(n_estimators=n,
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=list_n_estimators, y=train_score_list,
marker='*', label='Train Score')
p = sns.lineplot(x=list_n_estimators, y=test_score_list,
marker='o', label='Test Score')
plt.xlabel("n_estimators")
plt.ylabel("Accuracy Score")
plt.title("Random Forest Accuracy vs. Number of Estimators")
plt.legend()
plt.show()
```



#2-3-3 Models – Random Forest

#3-3 RF With The Best Parameters

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

# fit the model
last_rfc.fit(x_train,y_train)

predict = last_rfc.predict(x_test)

print('The accuracy of the Random Forest is',metrics.accuracy_score(predict,y_test)) #실제/예측 순서여야 함
```

96.67 %

```
y_pred_en = last_rfc.predict(x_test)
print('Model accuracy score with best parameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred_en)))

y_pred_train_en = last_rfc.predict(x_train)
```

96.67 %

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train_en)))
```

91.43 %

```
print('Training set score: {:.4f}'.format(last_rfc.score(x_train, y_train)))
```

100 %

```
print('Test set score: {:.4f}'.format(last_rfc.score(x_test, y_test)))
```

96.67 %

EWHA EURON

#2-4 Evaluation

#1 Confusion Matrix with 일반 DT

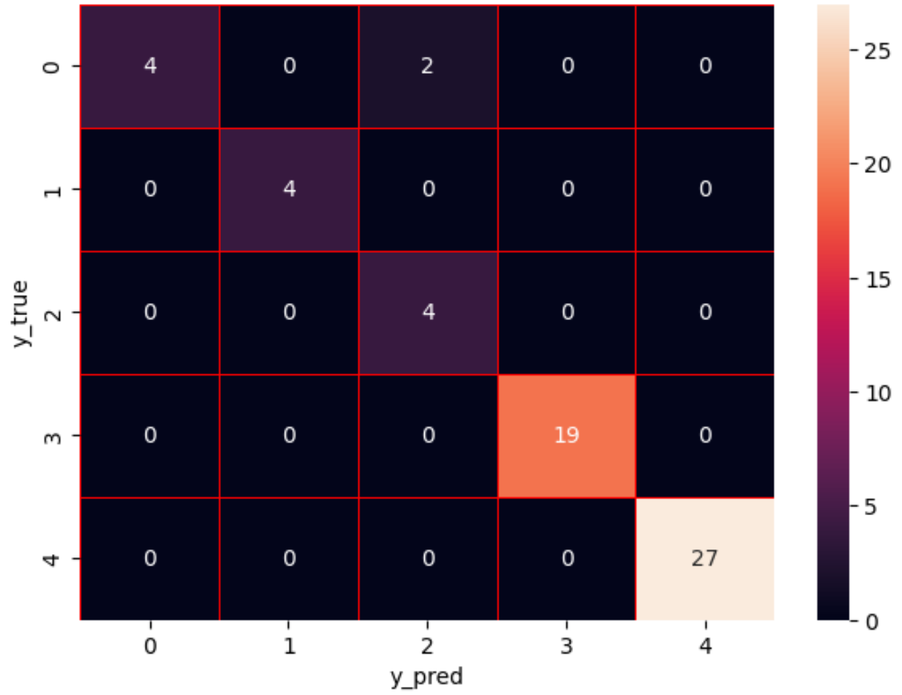
```
from sklearn.metrics import confusion_matrix


cm_des = DecisionTreeClassifier()

# fit the model
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)
cm_des1
```

```
array([[ 4,  0,  2,  0,  0],
       [ 0,  4,  0,  0,  0],
       [ 0,  0,  4,  0,  0],
       [ 0,  0,  0, 19,  0],
       [ 0,  0,  0,  0, 27]])
```

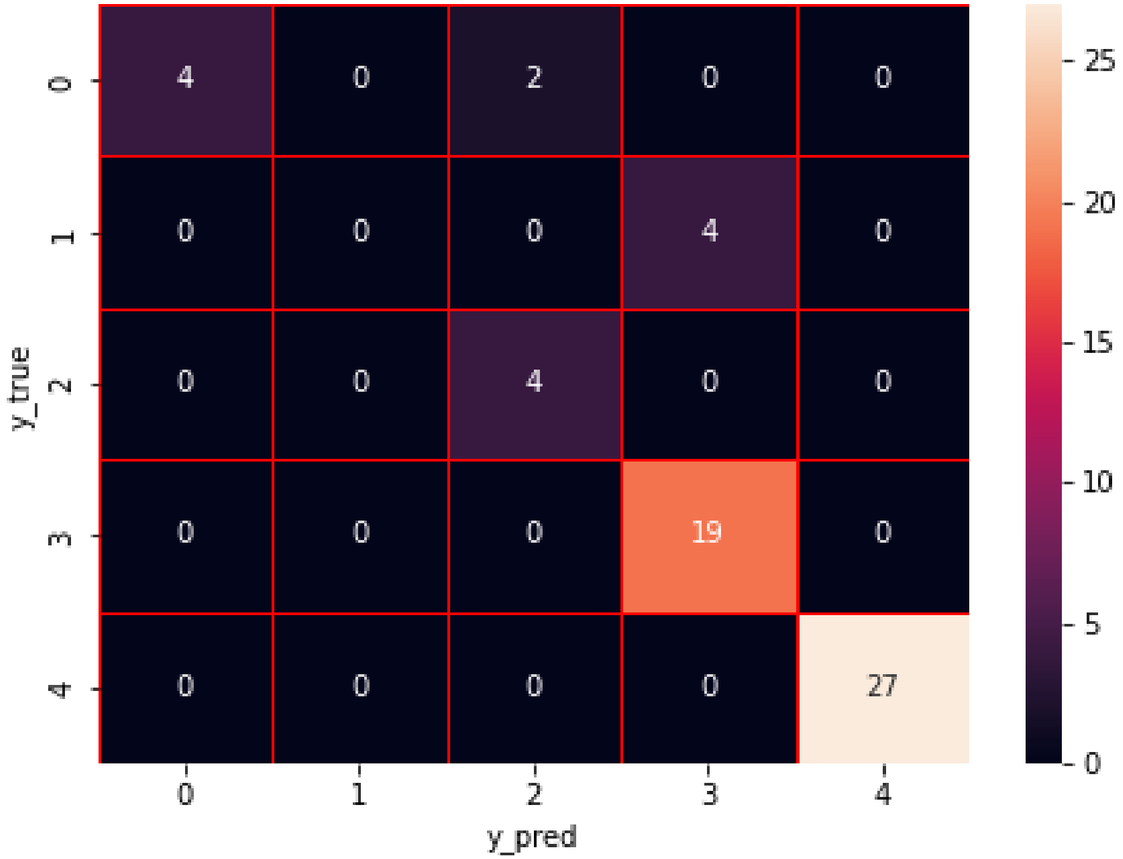
#2-4 Evaluation

#1 Confusion Matrix with 일반 DT

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

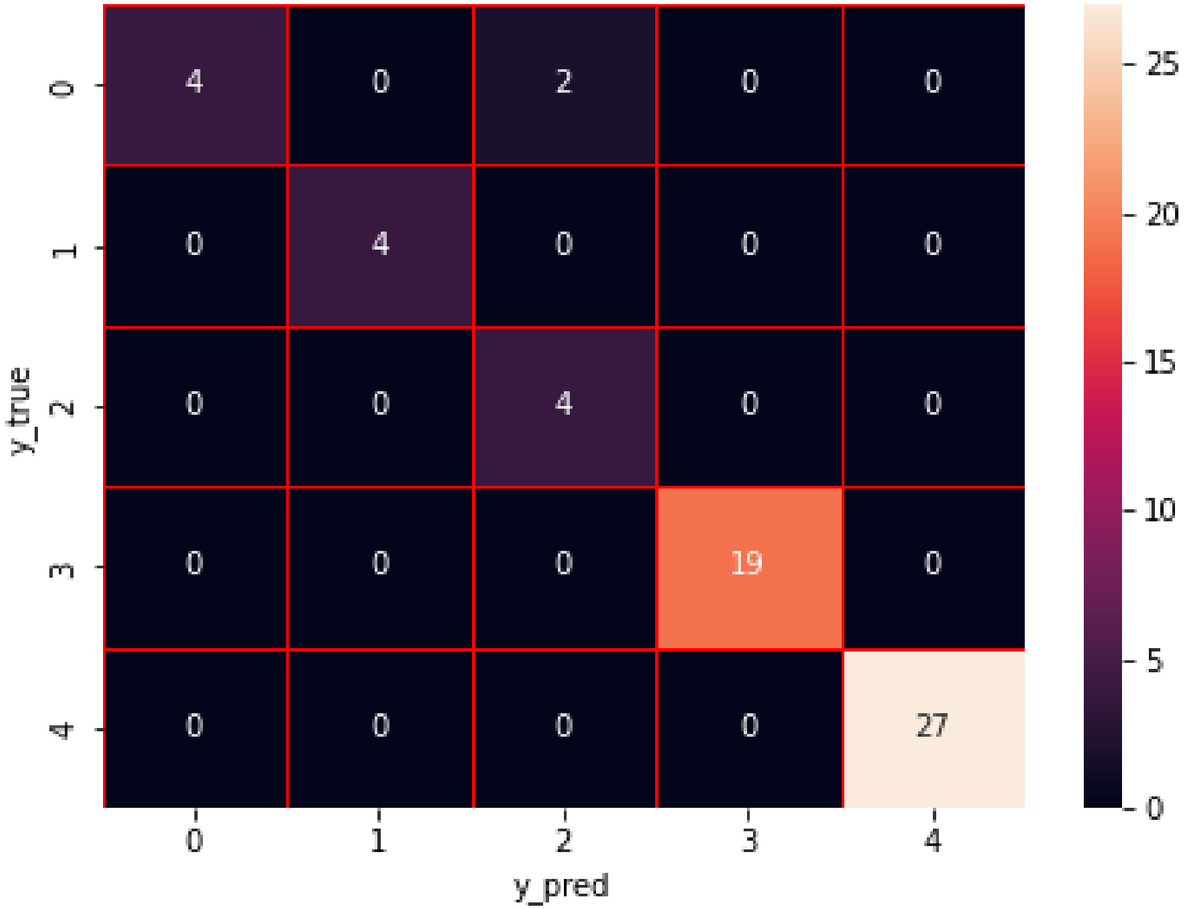


#2 Confusion Matrix with gini&DT



#2-4 Evaluation

#3 Confusion Matrix with RF



03. Beginner Friendly CATBOOST with OPTUNA

EWHA EURON

#3-1Data Review

#1 Optuna와 catboost 설치 및 라이브러리 불러오기

```
data = pd.read_csv("drug200.csv")
data
```

#2 Data 불러오고 읽기

```
df = pd.read_csv('/content/drive/MyDrive/heart.csv')
  display(df.head())
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
0	40	M	ATA	140	289	0	Normal	172
1	49	F	NAP	160	180	0	Normal	156
2	37	M	ATA	130	283	0	ST	98
3	48	F	ASY	138	214	0	Normal	108
4	54	M	NAP	150	195	0	Normal	122

```
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
```

#3-1Data Review

#1 복사된 데이터 확인

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

#2 missing data 확인

```
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
    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

#3 이상 데이터 확인

```
df[categorical].nunique()
```

	0
Sex	2
ChestPainType	4
RestingECG	3
ExerciseAngina	2
ST_Slope	3

dtype: int64

#3-2 Exploratory Data Analysis& Visualization

#1 Target Variable

.value_counts() 활용해서 심장병 환자 비율과 명수 확인

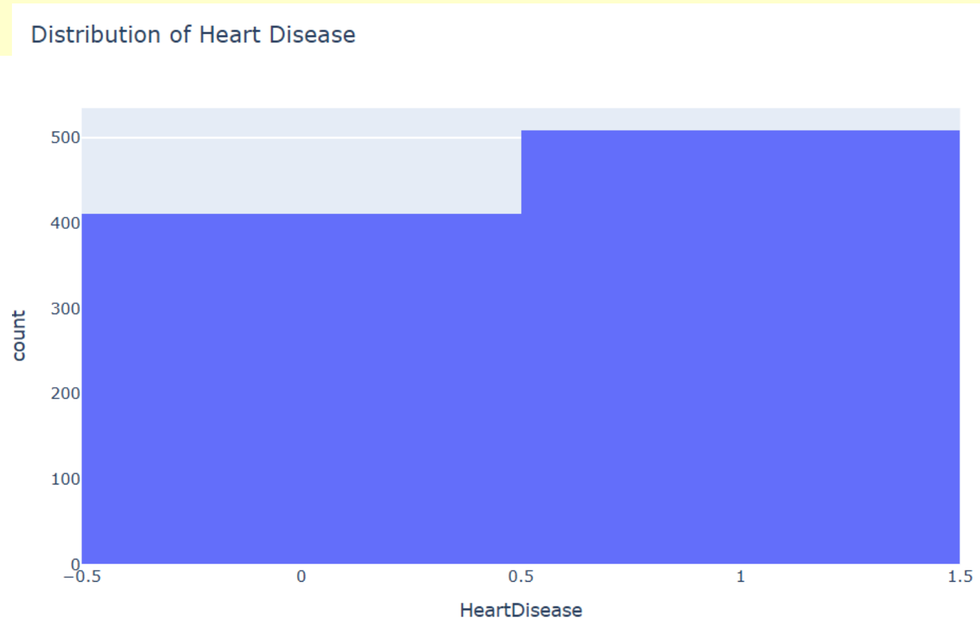
결과 :

```
e: {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)
```

*Kaggle 오류로 아래의 plotly.express를 활용해서 시각화

```
import plotly.express as px
fig = px.histogram(df, x='HeartDisease', title='Distribution of Heart Disease')
fig.show()
```



#3-2 Exploratory Data Analysis& Visualization

#2 Numerical Features

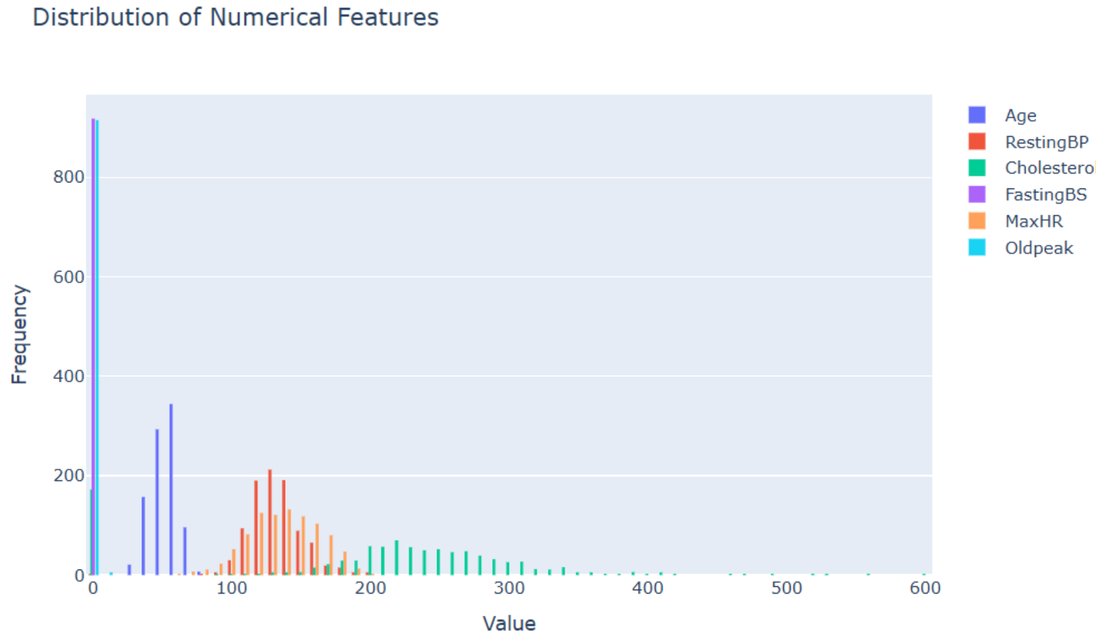
df[numerical].describe() 활용해서 심장병에 대한 수치형 데이터의 통계 확인

결과 :

	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

*Kaggle 오류로 아래의 plotly.graph_objects를 활용해서 시각화

```
import plotly.graph_objects as go

fig = go.Figure()

for col in numerical:
    fig.add_trace(go.Histogram(x=df[col], name=col))

fig.update_layout(title='Distribution of Numerical Features',
                  xaxis_title='Value',
                  yaxis_title='Frequency')

fig.show()
```



#3-2 Exploratory Data Analysis& Visualization

#2 Numerical Features

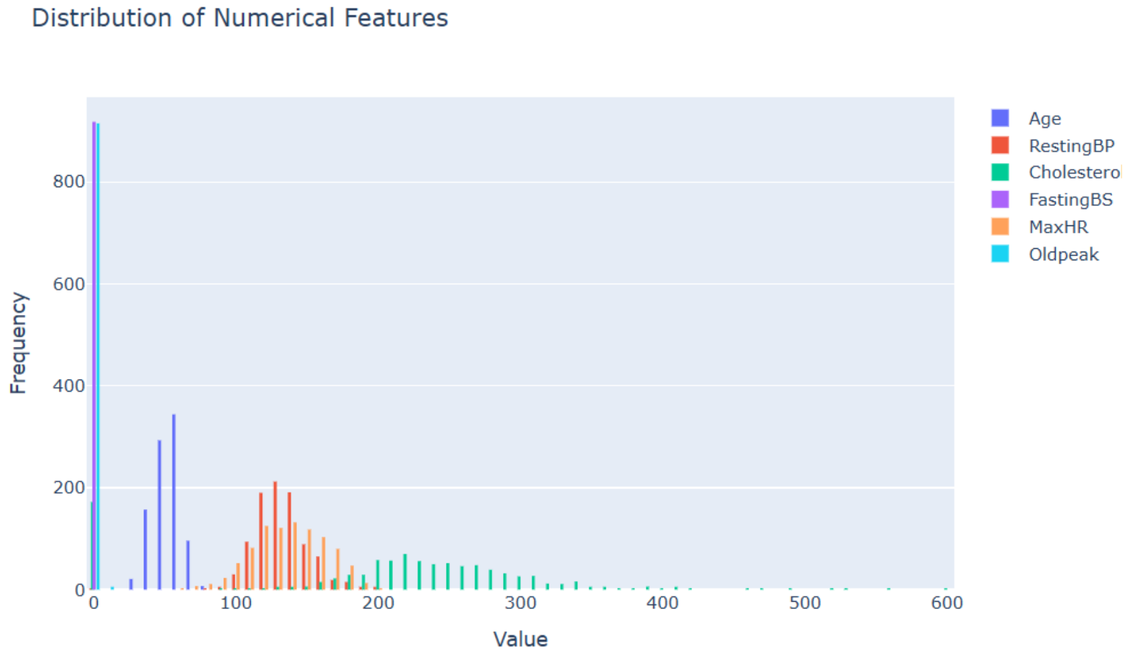
df[numerical].describe() 활용해서 심장병에 대한 수 데이터의 통계 확인

결과 :

	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

*Kaggle 오류로 아래의 plotly.graph_objects를 활용해서 시각화

```
import plotly.graph_objects as go

fig = go.Figure()

for col in numerical:
    fig.add_trace(go.Histogram(x=df[col], name=col))

fig.update_layout(title='Distribution of Numerical Features',
                  xaxis_title='Value',
                  yaxis_title='Frequency')

fig.show()
```

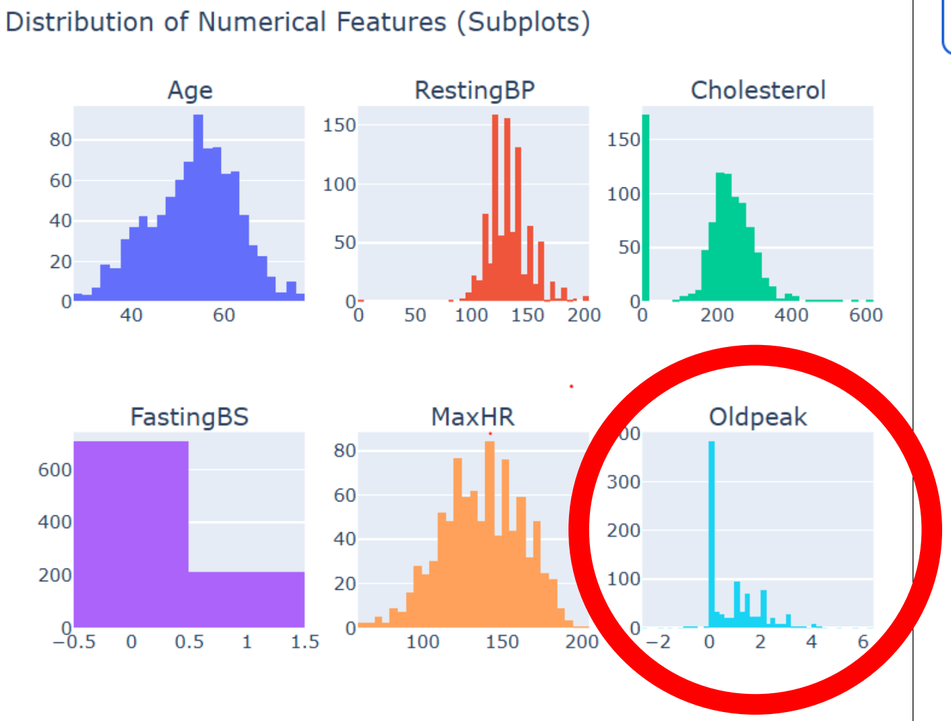


#3-2 Exploratory Data Analysis& Visualization

#2 Numerical Features

개별 그래프

결과 :




```
skew_limit = 0.75 # This is our threshold-limit to evaluate skewness. Overall below abs(1) seems acceptable for tl
skew_vals = df[numerical].drop('FastingBS', axis=1).skew()
skew_cols= skew_vals[abs(skew_vals)> skew_limit].sort_values(ascending=False)
skew_cols
```

```
Oldpeak    1.022872
dtype:
```

*코드 설명:

Skew_limit: 왜도(데이터 분포의 비대칭성(asymmetry)을 나타내는 통계적 지표)를 평가하기 위한 임계값을 0.75로 설정. 절댓값이 이 값보다 큰 왜도를 가진 변수를 '치우침이 있는' 변수로 간주.

skew_vals: 왜도를 계산. 이때, 'FastingBS' 열을 제외. 이 열은 이진 변수(0 또는 1)이기 때문에 왜도를 계산하는 것이 통계적으로 유의미하지 않을 수 있기 때문.

Skew_cols: 최종적으로 왜도의 절댓값이 0.75를 초과하는 수치형 열들의 이름과 해당 왜도 값이 출력

#3-2 Exploratory Data Analysis& Visualization

#2 Numerical Features

```
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='coolwarm',mask=matrix, ax=ax);
```

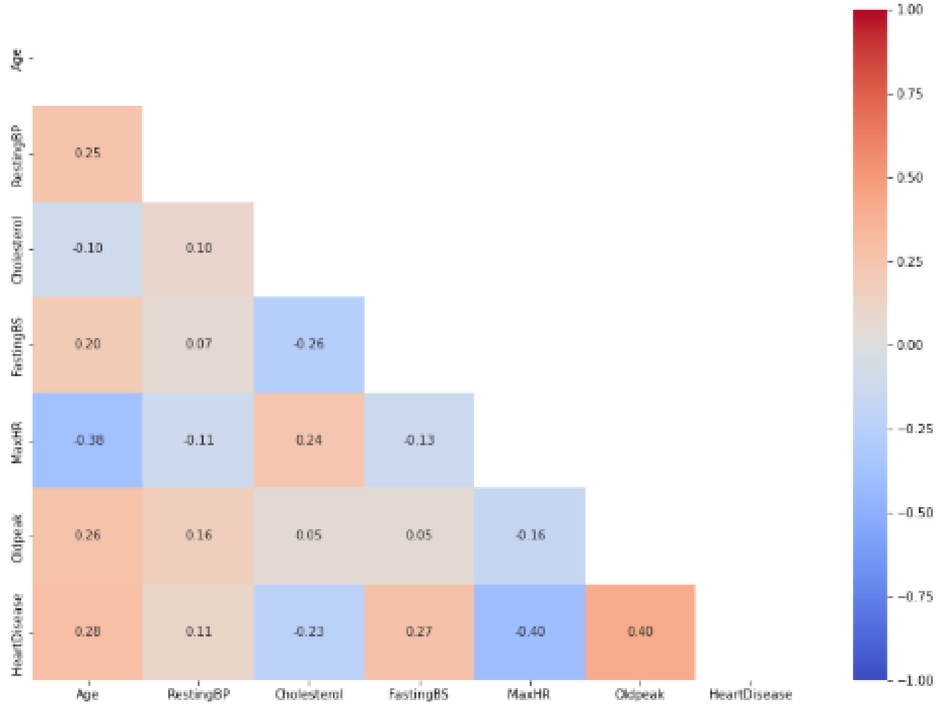
*코드 설명:

코드 쓰임-데이터프레임의 수치형 변수들 간의 상관관계(correlation)를 계산하고 이를 히트맵(heatmap)으로 시각화가 목적

df[numerical1].corr() :

numerical1에 저장된 수치형 열들 간의 피어슨 상관계수(Pearson correlation coefficient)를 계산하여 상관 행렬(correlation matrix)을 생성함. 상관계수는 두 변수 간의 선형적 관계의 강도와 방향을 나타냄. (-1에서 1 사이의 값을 가짐)

np.triu(...):
NumPy 함수로, 주어진 행렬의 상삼각 행렬(upper triangle)을 반환. 이 경우 상관 행렬의 대각선 위쪽 부분만 남기고 나머지는 0으로 채워진 행렬이 생성

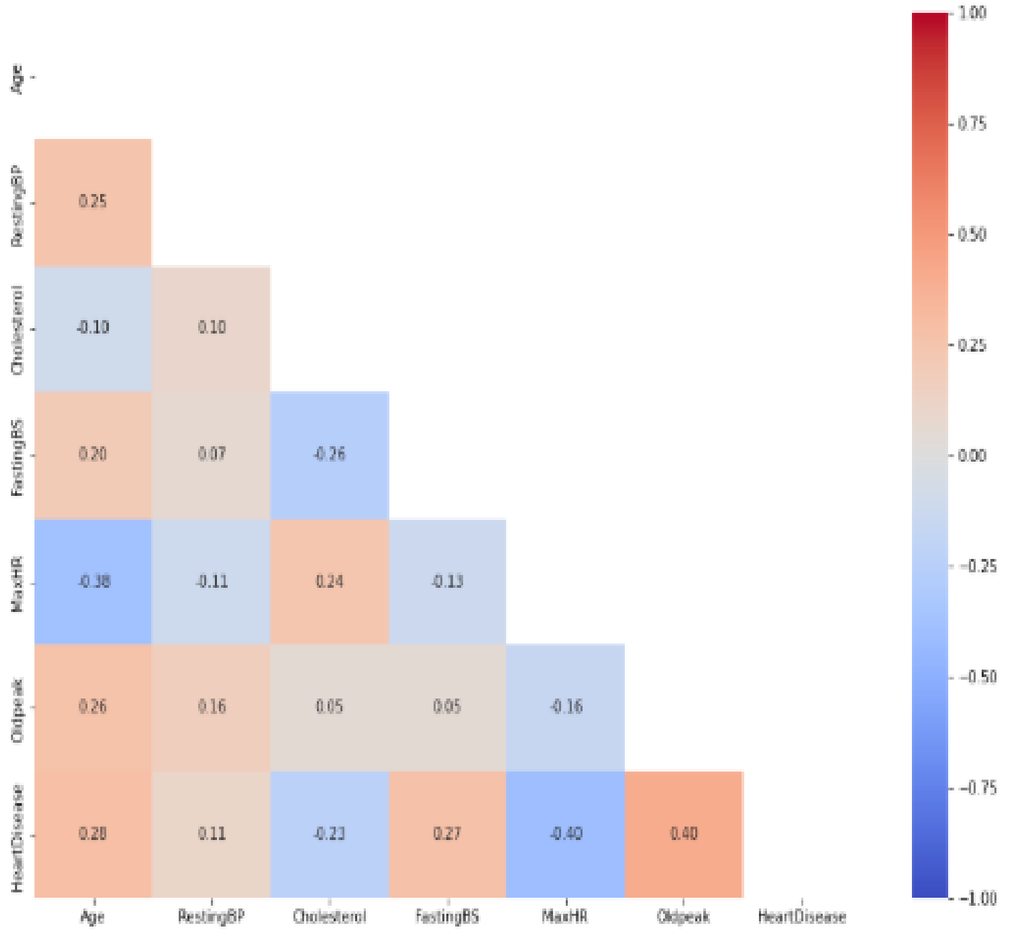


#3-2 Exploratory Data Analysis& Visualization

#2 Numerical Features

-Numerical Features로 얻을 수 있는 데이터 분석 인사이트

1. 수치형 데이터와 target 변수들 사이의 상관관계는 낮다.
2. Oldpeak(우울증 관련 수치)는 심장병과 양의 상관관계를 갖는다.
3. 최대 심박수는 심장병과 음의 상관관계를 갖는다.
4. 콜레스테롤은 심장병과 음의 상관관계를 갖는다.



#3-2 Exploratory Data Analysis& Visualization

#3 Categorical Features

```
df[categorical].head()
```

	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_Slope
0	M	ATA	Normal	N	Up
1	F	NAP	Normal	N	Flat
2	M	ATA	ST	N	Up
3	F	ASY	Normal	Y	Flat
4	M	NAP	Normal	N	Up

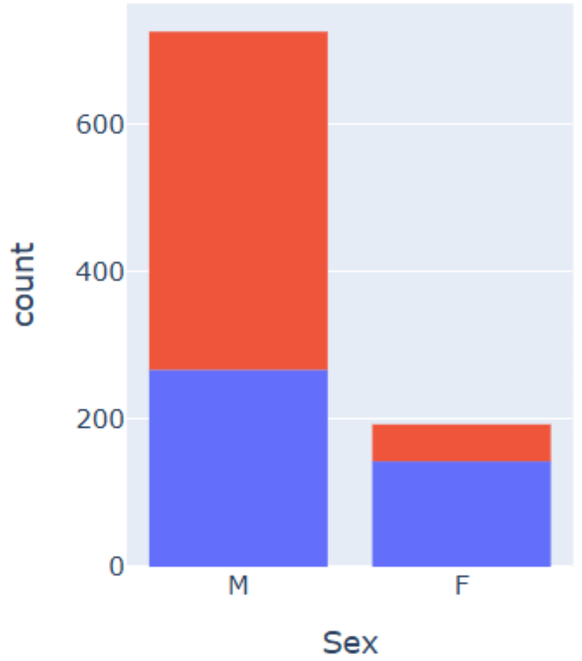
* 성별에 따른 심장별 발병률 계산

```
print (f'A female person has a probability of {round(df[df["Sex"]=="F"][

print()

print (f'A male person has a probability of {round(df[df["Sex"]=="M"]["H

print()
```

```
A female person has a probability of 25.91 % have a HeartDisease

A male person has a probability o 63.17 % have a HeartDisease
```

*그래프로 시각화

```
fig = px.histogram(df, x="Sex", color="HeartDisease",width=400,
height=400)
fig.show()
```



0은 심장병, 1은 심장병 X

#3-2 Exploratory Data Analysis& Visualization

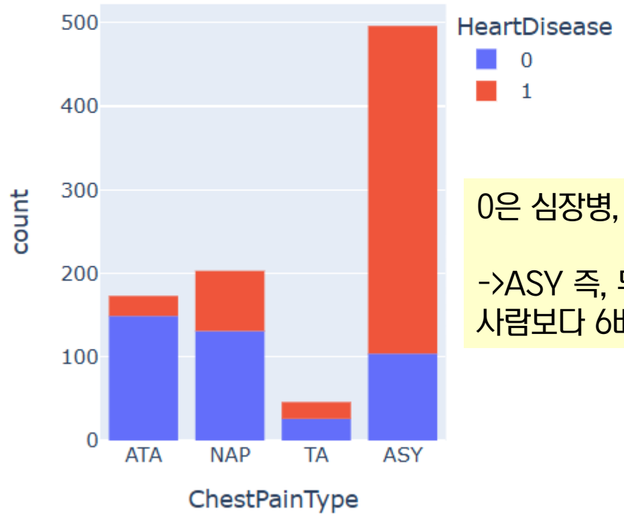
#3 Categorical Features

- ChestPainType에 따른 심장별 발병률 계산

df.groupby('ChestPainType')['HeartDisease'].mean().sort_values(ascending=False))

[25]
✓ 0초

```
df.groupby('ChestPainType')['HeartDisease'].mean().sort_values(ascending=False)
fig = px.histogram(df, x="ChestPainType", color="HeartDisease",width=400, height=400)
fig.show()
```



0은 심장병, 1은 심장병 X

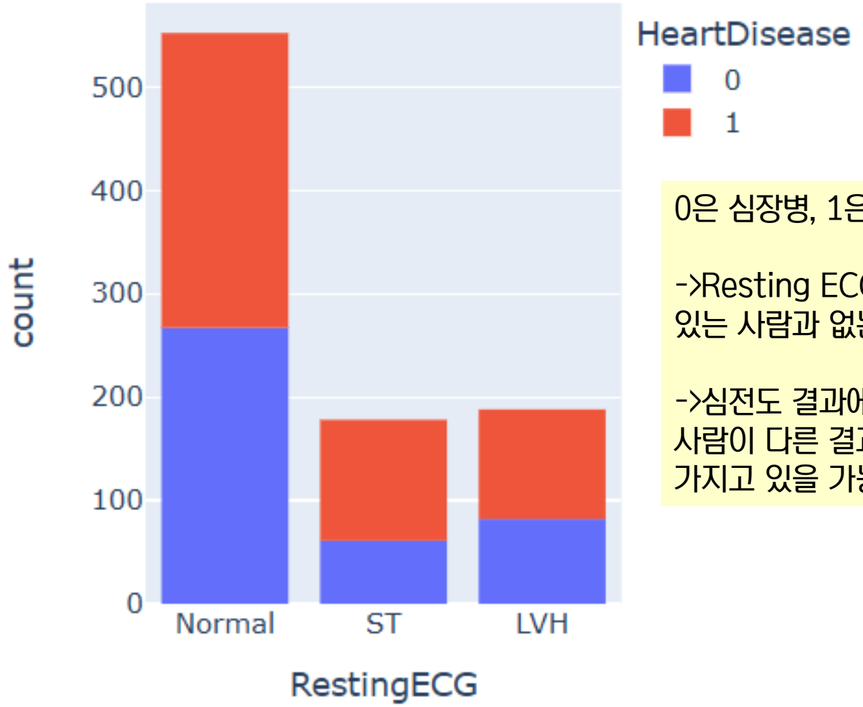
->ASY 즉, 무증상 흉통을 가진 사람임 ATA (비정형 협심증)을 가진 사람보다 6배 더 심장병 발생률이 높음.

#3-2 Exploratory Data Analysis& Visualization

#3 Categorical Features

- RestingECG에 따른 심장별 발병률 계산

```
df.groupby('RestingECG')['HeartDisease'].mean().sort_values(ascending=False)
```



0은 심장병, 1은 심장병 X

->Resting ECG (안정 시 심전도 검사) 결과만으로는 심장 질환이 있는 사람과 없는 사람 사이에 뚜렷한 차이가 잘 나타나지 않는다

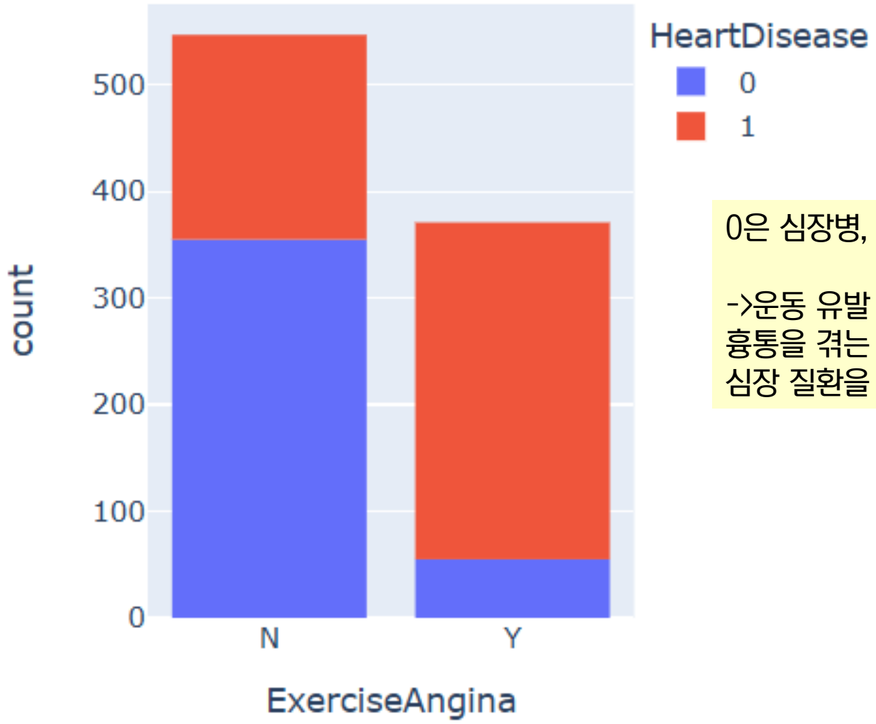
->심전도 결과에서 ST-T파 이상(abnormality) 소견을 보이는 사람이 다른 결과를 보이는 사람(정상, 또는 LVH 등)보다 심장 질환을 가지고 있을 가능성이 더 높다

#3-2 Exploratory Data Analysis& Visualization

#3 Categorical Features

- ExerciseAngina 에 따른 심장별 발병률 계산

```
df.groupby('ExerciseAngina')['HeartDisease'].mean().sort_values(ascending=False)
```



0은 심장병, 1은 심장병 X

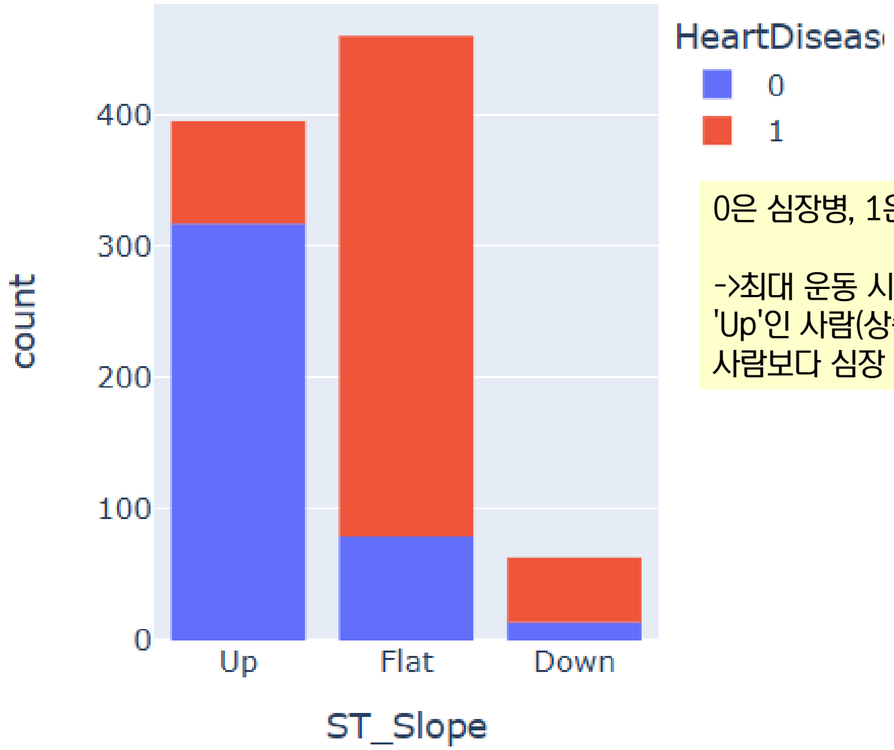
->운동 유발 협심증(ExerciseAngina)이 'Yes'인 사람(운동 시 흉통을 겪는 사람)은 'No'인 사람(운동 시 흉통을 겪지 않는 사람)보다 심장 질환을 가질 가능성이 거의 2.4배 더 높습니다.

#3-2 Exploratory Data Analysis& Visualization

#3 Categorical Features

- ST_Slope 에 따른 심장별 발병률 계산

```
df.groupby('ST_Slope')['HeartDisease'].mean().sort_values(ascending=False)
```



0은 심장병, 1은 심장병 X

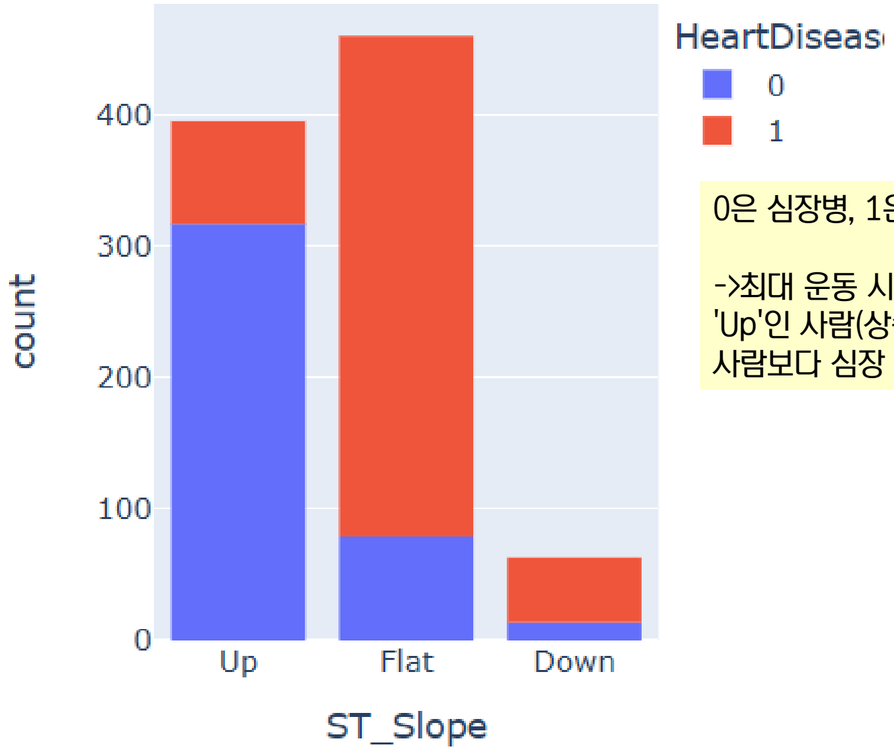
->최대 운동 시 ST_Slope는 차이가 있음.특히, ST 분절 기울기가 'Up'인 사람(상승하는 기울기)은 나머지 두 가지 기울기를 가진 사람보다 심장 질환을 가질 가능성이 현저히 낮음.

#3-2 Exploratory Data Analysis& Visualization

#3 Categorical Features

- ST_Slope 에 따른 심장별 발병률 계산

```
df.groupby('ST_Slope')['HeartDisease'].mean().sort_values(ascending=False)
```



0은 심장병, 1은 심장병 X

->최대 운동 시 ST_Slope는 차이가 있음.특히, ST 분절 기울기가 'Up'인 사람(상승하는 기울기)은 나머지 두 가지 기울기를 가진 사람보다 심장 질환을 가질 가능성이 현저히 낮음.

#3-2 Exploratory Data Analysis& Visualization

#4 최종 정리

범주	지표	상관관계/위험도	상세 내용
성별	Gender	**남성 위험도 ⬆**	**남성**은 여성보다 심장 질환을 가질 가능성이 **약 2.44배 높음**.
심전도/운동	Exercise Angina	**'Yes' 위험도 ⬆**	**운동 유발 협심증이 있는 경우**('Yes'), 없는 경우('No')보다 심장 질환 위험이 **약 2.4배 높음**.
심전도/운동	ST_Slope	**'Up' 위험도 ⬇**	운동 시 ST 분절 기울기가 상승(Up)하는 경우, 다른 기울기(Flat, Down) 대비 심장 질환 가능성이 **현저히 낮음**.
흉통 유형	ASY (무증상)	**ATA 대비 위험도 ⬆**	**무증상 흉통(ASY)** 환자는 비정형 협심증(ATA) 환자보다 심장 질환 위험이 **약 6배 높음** (무증상에 대한 주의 필요).
수치형 지표	Oldpeak	**정적(+) 상관관계**	**Oldpeak (ST 하강/우울증 관련 수치)** 수치가 높을수록 심장 질환 위험이 증가함.
수치형 지표	Max Heart Rate	**부적(-) 상관관계**	**최대 심박수**가 높을수록 심장 질환 위험이 감소하는 경향을 보임.
특이사항	Cholesterol	**부적(-) 상관관계**	**콜레스테롤 수치**는 흥미롭게도 심장 질환과 부적 상관관계(콜레스테롤이 높을수록 위험 낮음)를 보임 (추가 분석 필요).

#3-3 Models – Base model

#1 use dummy classifier model as a base model

[16]
✓ 0초

```
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
```

	Accuracy
DummyClassifier	0.5942

#3-4 Models

#1 dummy classifier model 정확도

[16]
✓ 0초

```
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
```

	Accuracy
DummyClassifier	0.5942

#3-4 Models

#2 Logistic & Linear Discriminant & SVC & KNN 정확도 비교

```
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
```

	Accuracy
Logistic	0.8841
LinearDiscriminant	0.8696
SVM	0.7246
KNeighbors	0.7174

#3-4 Models

#3 Logistic & Linear Discriminant & SVC & KNN with Scaler(전처리)

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

```
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
```

	Accuracy
Logistic_scl	0.8804
LinearDiscriminant_scl	0.8696
SVM_scl	0.8841
KNeighbors_scl	0.8841

스케일러를 사용했을 때, KNN과 SVM의 성능이 좋아짐. 왜냐하면 이 둘은 거리 기반 알고리짐이기 때문임. 스케일링을 통해 모든 특성이 공평하게 반영되어 모델의 예측정확도가 높아짐.

#3-4 Models

#4 Ensemble Models (AdaBoost & Gradient Boosting & Random Forest & Extra Trees)

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

```
model : AdaBoostClassifier(random_state=0) and  accuracy score is : 0.8732
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
```

	Accuracy
Ada	0.8732
Gradient	0.8768
Random	0.8877
ExtraTree	0.8804

-정확도 점수가 서로 비슷함.
-랜덤 포레스트랑 엑스트라 트리 모두 비슷한 정확도 점수를 가지고 있음.
이 두 모델 모두 하이퍼파라미터 튜닝을 통해 정확도가 더 개선될 수도

#3-4 Models

#5 Ensemble Models (XGBoost & LightGBM & Catboost)

- XGBoost & LightGBM

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

	Accuracy
XGBoost	0.8478
LightGBM	0.8732

- Catboost

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

	Accuracy
Catboost_default	0.8804

- Catboost란?

Categorical (범주형)과 Boosting (부스팅)의 합성어로, 특히 범주형 데이터를 효과적으로 처리하도록 최적화된 결정 트리 기반의 경사 부스팅(Gradient Boosting) 알고리즘

-사이킷런 도구와의 호환성 제공

-목적함수:
Logloss: 타겟이 두 가지 값만 가질 때, 또는 target_border 매개변수가 ┬one이 아닐 때
MultiClass: 타겟이 두 가지 이상, border_count 매개변수가 None일때.

#3-4 Models

#6 Catboost HyperParameter Tuning with Optuna

▼파라미터 설명

Parameter	설명
Objective	**목적 함수:** 과적합 감지 및 최적 모델 선택에 사용되는 지원되는 지표(metric).
colsample_bylevel	**수준별 열 샘플링:** 훈련 속도를 높여주며, 일반적으로 모델 품질에 영향을 미치지 않는다.
depth	**트리 깊이:** 생성되는 결정 트리의 최대 깊이.
boosting_type	**부스팅 유형:** 기본적으로 작은 데이터셋에는 [A]로 설정된다. 이는 과적합을 방지하지만 계산 비용이 많이 든다. 훈련 속도를 높이려면 이 매개변수 값을 [B]로 설정해 볼 수 있다.
bootstrap_type	**부트스트랩 유형:** 기본적으로 객체의 가중치를 샘플링하는 방법은 [C]로 설정된다. 이 방법이 [D]로 설정되고 배깅(bagging)의 샘플링 비율 값이 1보다 작으면 훈련이 더 빠르게 수행된다.

▼Optuna로 최적의 하이퍼 파라미터 조합 찾기

```
study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50, timeout=600)
```

▼ 최적 파라미터

```
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
```

#3-4 Models

#7 Catboost HyperParameter Tuning with Optuna

▼ 최적 파라미터로 실행

```
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
```

	Accuracy
Catboost_tuned	0.9094

최적 파라미터로 정확도가 증가함.

Precision:
모델이 클래스 1이라고 예측한 것 중 실제로 클래스 1이 맞은 비율이 93%로, 클래스 0 예측(88%)보다 더 정확했습니다.

Recall:
실제 클래스 1인 데이터 중 모델이 클래스 1이라고 정확하게 찾아낸 비율이 91%였습니다.

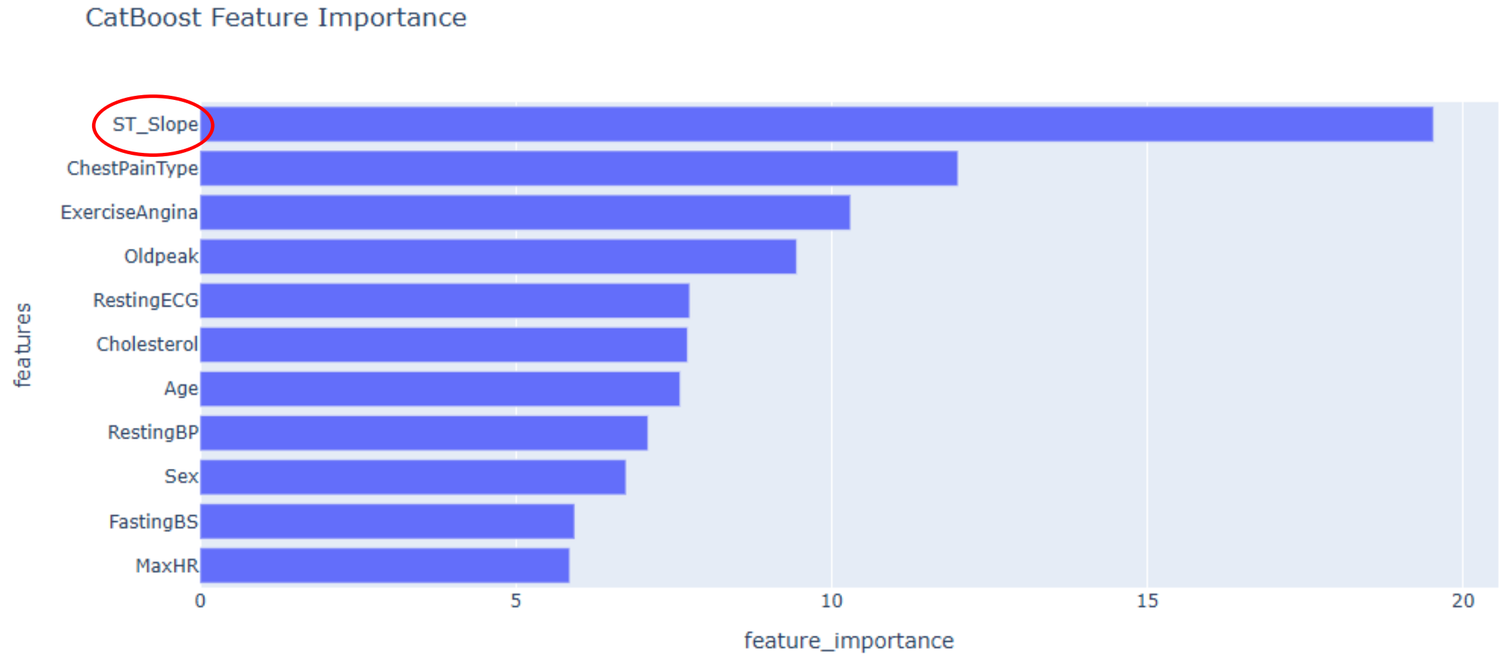
F1:
정밀도와 재현율의 조화 평균을 나타내는 점수로, 클래스 1에 대한 성능이 클래스 0보다 약간 더 높게 나타났습니다.

Support:
테스트 데이터 중 실제 클래스 0은 112개, 실제 클래스 1은 164개였습니다. 데이터가 클래스 1에 약간 치우쳐 있습니다.

#3-5 feature_importance

#1 feature_importance

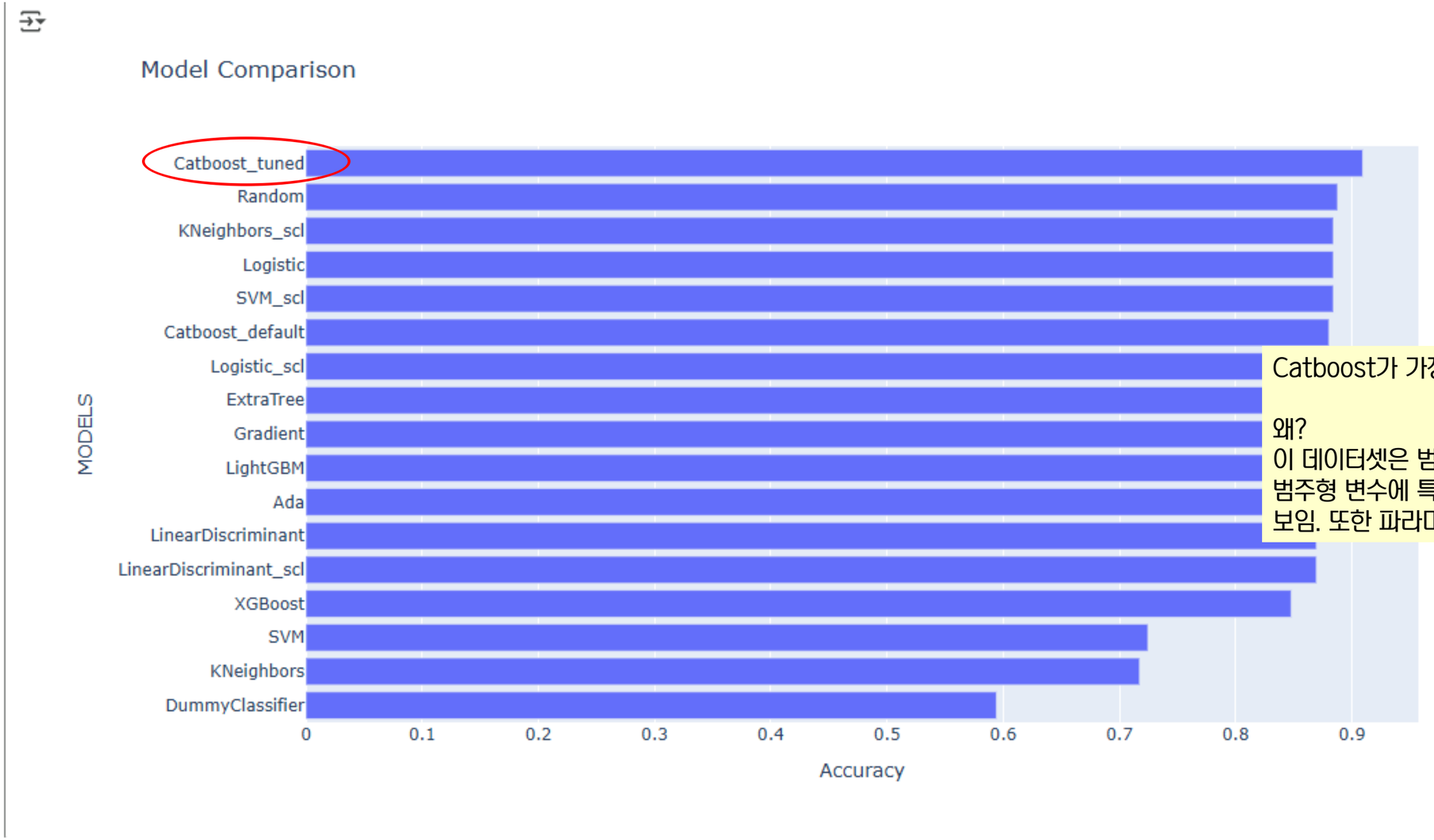
```
feature_importance = np.array(model.get_feature_importance())
features = np.array(X_train.columns)
fi={'features':features,'feature_importance':feature_importance}
df_fi = pd.DataFrame(fi)
df_fi.sort_values(by=['feature_importance'], ascending=True,inplace=True)
fig = px.bar(df_fi, x='feature_importance', y='features',title="CatBoost Feature Importance",height=500)
fig.show()
```



ST_slope 가 가장 큰 영향을 끼친
즉,중요한 피쳐임을 알 수 있음.

#3-6 Model Comparison

#1 Model Comparison



Catboost가 가장 높은 정확도를 보임.

왜?
이 데이터셋은 범주형 변수의 영향력이 크기 때문에, 범주형 변수에 특화된 Catboost가 가장 좋은 효과를 보임. 또한 파라미터 튜닝으로 강점 강화.