4.9 분류 실습 - 캐글 산타데르 고객 만족 예측(267-278)

- 370개의 피처로 주어진 데이터 세트 기반에서 고객 만족 여부를 예측
- 클래스 레이블 명은 TARGET이며, 이 값이 1이면 불만을 가진 고객, 0이면 만족한 고객
- 모델의 성능 평가는 ROC-AUC(ROC 곡선 영역)로 평가
  - 대부분 만족이고 불만족은 일부이기에 정확도 수치보다 더 적합
  - https://www. kaggle.com/c/santander—customer—satisfaction/data

1단계 데이터 전처리

(1) DataFrame으로 로딩 → 피처의 개수 확인

```
from google.colab import files
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

cust_df = pd.read_csv("/content/drive/MyDrive/kaggle/train.csv.zip", enco
ding = 'latin-1')
#latin-1, 판다스가 파일을 서유럽 문자 코드 체계로 저장된 파일로 읽어라
#UTF-8이 아니라 다른 인코딩을 하면 오류날 때가 있음
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	imp_op_var40_e
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

(2) Null값, 피처의 타입 알아보기

- Null 없음
- 111개 피처 float
- 260개 int

```
cust_df.info()
```

```
#결과
<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
```

(3) 만족과 불만족 비율

- 대부분 만족
- 0 = 만족
- 1 = 불만족

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

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

(4) 각 피처의 값 분포

- NaN이나 특정 예외 값을 -99999로 변환
- print(cust_df.var3.value_counts( )[:10])로 var3의 값을 조사해 보면 -999999 값이 116개

```
cust_df.describe()
```

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

8 rows × 371 columns

(5) -999999 → 2 & ID 피처 드롭 & 클래스 데이터 세트와 피처 데이터 세트 분리

```
cust_df['var3'].replace(-999999, 2, inplace=True)
cust_df.drop('ID', axis=1, inplace=True)

X_features = cust_df.iloc[:, :-1]
y_labels = cust_df.iloc[:, -1]
```

(6) 학습 데이터/검증 데이터

- 학습/검증 데이터 분리

- 피처 개수 확인
- 비율 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},\n테스트 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)
```

```
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
```

```
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.3, random_state=0)
```

2. XGBoost 모델 학습과 하이퍼 파라미터 튜닝

(1) XGBoost 모델 생성 후 학습 수행 후 테스트 데이터로 진행된 ROC-AUC 값 확인

- n_estimators = 500
- early_stopping_ rounds = 100
- eval_metric = auc'(logloss로 해도 큰 차이는 없음)
- eval_set=[(X_tr, y_tr), (X_val, y_val)

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

xgb_clf = XGBClassifier(n_estimators=500, early_stopping_rounds=100, eva
l_metric='auc', random_state = 156)

xgb_clf.fit(X_tr, y_tr, eval_set=[(X_val, y_val)], verbose = 0)
xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:,1])
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

```
ROC AUC: 0.8321
```

(2) HyperOpt를 이용해 베이지안 최적화 → 파라미터 튜닝

- max_depth는 5에서 15까지 1 간격으로, min_child_weight는 1에서 6까지 1 간격
- colsample_bytree는 0.5에서 0.95 사이, learning_rate는 0.01 에서 0.2 사이 정규 분포된 값

```
from hyperopt import hp

xgb_search_space = {
    'max_depth': hp.quinform('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)
                }
```

- 목적 함수: 3 Fold 교차 검증

- 평균 ROC- AUC 값을 반환하되 1을 곱해주어 최대 ROC-AUC 값이 최소 반환값
- KFold 클래스를 이용하여 직접 학습과 검증 데이터 세트를 추출하고 이를 교차 검증 횟수만큼 학습과 성능 평가를 수행
- estimators는 100으로 줄이고, early_stopping_rounds도 30으로 줄여서 테스트한 뒤 나중에 하이퍼 파라미터 튜닝이 완료되면 다시 증가

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

def objective_func(search_space):
    xgb_clf = XGBClassifier(n_estimators = 100, early_stopping_rounds=30, e
val_metric='auc',
                            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'])
    kf_list = []
    kf = KFold(n_splits = 3)
    for tr_idx, val_idx in kf.split(X_train):
        X_tr, y_tr = X_train.iloc[tr_idx], y_train.iloc[tr_idx]
        X_val, y_val = X_train.iloc[val_idx], y_train.iloc[val_idx]
        xgb_clf.fit(X_tr, y_tr, eval_set=[(X_val, y_val)], verbose=0)
        score = roc_auc_score(y_val, xgb_clf.predict_proba(X_val)[:,1])
        kf_list.append(score)
    return -1.0 * np.mean(kf_list)
```

⚠️ .iloc[숫자] 헷갈리지 않기

(3) fmin() 함수를 호출: max_eval=50회만큼 반복하면서 최적의 하이퍼 파라미터를 도출

```
from hyperopt import fmin, tpe, Trials

trials = Trials()
best = fmin(fn = objective_func,
            space = xgb_search_space,
            algo = tpe.suggest,
```

```
        max_evals = 50,
        trials = trials,
        rstate = np.random.default_rng(seed = 30))
print('best:', best)
```

(4) 피처 중요도 그래프로 나타냄

- xgboost 모듈의 plot_ importance () 메서드

```
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)
```

(5) XGBClassifier 재학습, 성능 평가

- n_estimators는 500으로 증가

```
xgb_clf = XGBClassifier(n_estimators =500, early_stopping_rounds=100, ev
al_metric='auc',
                        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))

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

3. LightGbM 모델 학습과 하이퍼 파라미터 튜닝

(1) LightGbM 모델 생성 후 학습 수행 후 테스트 데이터로 진행된 ROC-AUC 값 확인

- n_estimators = 500
- early_stopping_ rounds = 100
- eval_metric = auc'(logloss로 해도 큰 차이는 없음)
- eval_set=[(X_tr, y_tr), (X_val, y_val)

```
from lightgbm import LGBMClassifier
from sklearn.metrics import roc_auc_score

lgbm_clf = LGBMClassifier(n_estimators=500, early_stopping_rounds=100,
eval_metric='auc')

eval_set = [(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))
```

(2) HyperOpt를 이용해 베이지안 최적화 → 파라미터 튜닝

- numjeaves, max_depth, min_child_samples, subsample, learning_rate

```
from hyperopt import hp

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)
                }
```

- 목적 함수: 3 Fold 교차 검증
- 평균 ROC- AUC 값을 반환하되 1을 곱해주어 최대 ROC-AUC 값이 최소 반환값

- KFold 클래스를 이용하여 직접 학습과 검증 데이터 세트를 추출하고 이를 교차 검증 횟수만큼 학습과 성능 평가를 수행
- estimators는 100으로 줄이고, early_stopping_rounds도 30으로 줄여서 테스트한 뒤 나중에 하이퍼 파라미터 튜닝이 완료되면 다시 증가

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

def objective_func(search_space):
    lgbm_clf = LGBMClassifier(n_estimators = 100,
                              early_stopping_rounds=30, eval_metric='auc',
                              num_leaves = int(search_space['num_leaves']),
                              max_depth = int(search_space['max_depth']),
                              min_child_samples = int(search_space['min_child_sample
s']),
                              subsample = search_space['subsample'],
                              learning_rate = search_space['learning_rate'])
    kf_list = []
    kf = KFold(n_splits = 3)
    for tr_idx, val_idx in kf.split(X_train):
        X_tr, y_tr = X_train.iloc[tr_idx], y_train.iloc[tr_idx]
        X_val, y_val = X_train.iloc[val_idx], y_train.iloc[val_idx]
        lgbm_clf.fit(X_tr, y_tr, eval_set=[(X_val, y_val)], verbose=0)
        score = roc_auc_score(y_val, lgbm_clf.predict_proba(X_val)[:,1])
        kf_list.append(score)
    return -1.0 * np.mean(kf_list)
```

(3) fmin() 함수를 호출: max_eval=50회만큼 반복하면서 최적의 하이퍼 파라미터를 도출

```
from hyperopt import fmin, tpe, Trials

trials = Trials()
best = fmin(fn = objective_func,
            space = lgbm_search_space,
            algo = tpe.suggest,
            max_evals = 50,
```

```
        trials = trials,
        rstate = np.random.default_rng(seed = 30))
print('best:', best)
```

(4) LightGbM 재학습, 성능 평가

- n_estimators는 500으로 증가

```
lgbm_clf = LGBMClassifier(n_estimators =500, early_stopping_rounds=100,
eval_metric='auc',
                          learning_rate = round(best['learning_rate'], 5),
                          max_depth = int(best['max_depth']),
                          num_leaves = int(best['num_leaves']),
                          min_child_samples = int(best['min_child_samples']),
                          subsample = round(best['subsample'], 5))

lgbm_clf.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=0)
xgb_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:,1])
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

Beginner Friendly CATBOOST with OPTUNA

https://www.kaggle.com/code/kaanboke/beginner-friendly-catboost-with-optuna

- **Heart Failure Prediction Dataset**

문제

1. **Heart Disease =** target
2. **Build a model to get best calssification**
3. balance of the target variable → **Accuracy score**

모듈 설명

CatBoost: 머신러닝 모델 (부스팅 알고리즘)
Optuna: 하이퍼파라미터 자동 최적화 라이브러리

Plotly는 파이썬에서 대화형(interactive) 그래프를 그릴 수 있는 시각화 라이브러리

- 정적인 그래프(matplotlib)와 다르게
- **줌(zoom), 마우스 오버, 클릭, 툴팁, 드래그 등 인터랙티브 시각화 라이브러리**
- 마우스로 움직이고, 확대하고, 항목을 클릭해서 숨기거나 볼 수 있는 **웹 기반 그래프**

❤️ `from sklearn.discriminant_analysis import LinearDiscriminantAnalysis`

- 고차원 데이터를 저차원으로 줄이면서 분류 경계 명확히 할 때 사용

`from sklearn.compose import ColumnTransformer`

- 범주형 변수엔 `OneHotEncoder`, 수치형 변수엔 `StandardScaler` 적용

`from sklearn.pipeline import Pipeline`

- 전처리–학습–예측 과정을 일관되게 수행

`from sklearn.linear_model import LogisticRegression`

- 입력 변수의 선형 조합에 로지스틱 함수를 적용해 확률값을 예측

`from sklearn.svm import SVC`

- 클래스 간의 **최대 margin**을 갖는 초평면

`from sklearn.impute import SimpleImputer`

- 결측치(NaN)를 평균, 중앙값, 최빈값 등으로 채우는 전처리 도구

`from sklearn.dummy import DummyClassifier`

- 진짜 모델의 성능 비교용 — 무작위 예측이나 가장 빈도가 높은 클래스로만 예측

`from imblearn.over_sampling import SMOTE`

- **불균형 데이터(imbalanced dataset)** 를 보정하기 위한 오버샘플링 기법

`import cufflinks as cf`

- Plotly와 Pandas를 연결
- `.iplot()` 메서드를 이용해 바로 대화형 그래프를 그릴 수 있게 해줌

`cf.go_offline()`

- 인터넷 연결 없이도 Jupyter Notebook/Colab에서 그래프를 렌더링

`import plotly`, `import plotly.express as px`

- Plotly의 고수준(high-level) API → 한 줄 코드로 빠르게 시각화 가능

`import plotly.graph_objs as go`

- Plotly의 **저수준(low-level)** 그래픽 객체 인터페이스

`from plotly.subplots import make_subplots`

- 여러 개의 그래프(서브플롯)를 한 Figure에 배치

`import plotly.figure_factory as ff`

- 히스토그램, 덴드로그램, 히트맵, 분포 그래프

`import missingno as msno`

- 결측치(누락된 데이터) 시각화 전용 라이브러리

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

#데이터 생성, 분리, 학습, 검증
#데이터 전처리 sklearn.preprocessing
from sklearn.model_selection import KFold,cross_val_score, RepeatedStrati
fiedKFold,StratifiedKFold
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import OneHotEncoder,StandardScaler,PowerT
ransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
from sklearn.dummy import DummyClassifier
from imblearn.over_sampling import SMOTE

#앙상블
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
#분류 모델
import optuna
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

#여러 단계 한번에 처리
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_transformer

#교차검증 및 성능
from sklearn.model_selection import KFold, cross_val_predict, train_test_spl
it,GridSearchCV,cross_val_score
from sklearn.metrics import accuracy_score,classification_report

------------------------------------------------------------
#데이터 시각화
#importing plotly and cufflinks in offline mode
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)


import plotly
import plotly.express as px
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

#결측치 시각화
import missingno as msno

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

데이터 로드

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

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

Data 결측치 확인

`info()` , `duplicated().sum()` , `isnull().sum()`

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

```
np.int64(0)
```

- 중복 행 없음

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

missing(df)
```

	Missing_Number	Missing_Percent
Age	0	0.0
Sex	0	0.0
ChestPainType	0	0.0
RestingBP	0	0.0
Cholesterol	0	0.0
FastingBS	0	0.0
RestingECG	0	0.0

- 결측치 없음

데이터 수치/카테고리 확인

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

- **DataFrame에서** `HeartDisease` **를 제외한 나머지 수치형 컬럼 이름들만** 저장
- `.columns` 는 단순히 **컬럼 이름 리스트**를 반환

`.select_dtypes('object').columns`

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

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

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

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


Categorical Columns: Index(['Sex', 'ChestPainType', 'RestingECG', 'Exercise
Angina', 'ST_Slope'], dtype='object')
```

`df[categorical].nunique()`

- 각 컬럼(column)별로 **중복을 제외한 고유값(unique value)** 의 개수를 계산

데이터 라벨의 비율

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

```
Percentage of patient had a HeartDisease: 55.34 % → (508 patient)
Percentage of patient did not have a HeartDisease: 44.66 % → (410 patient)
```

- 심장병 걸림과 안걸림 유사함

데이터 시각화

구분	`iplot`	`px` (`plotly.express`)
시대	**구버전 (2017~)**	**현대 버전 (2019~)**
라이브러리	`cufflinks` 기반	순수 `plotly` 기반
사용 목적	**Pandas용 간편 wrapper**	**범용적 시각화 API**
대표 호출법	`df.iplot(...)`	`px.scatter(...)`
지원 범위	한정적 (bar, scatter, hist 등)	매우 넓음 (facet, mapbox, animation 등)
유지보수	❌ 중단됨	✅ 활발히 유지보수됨

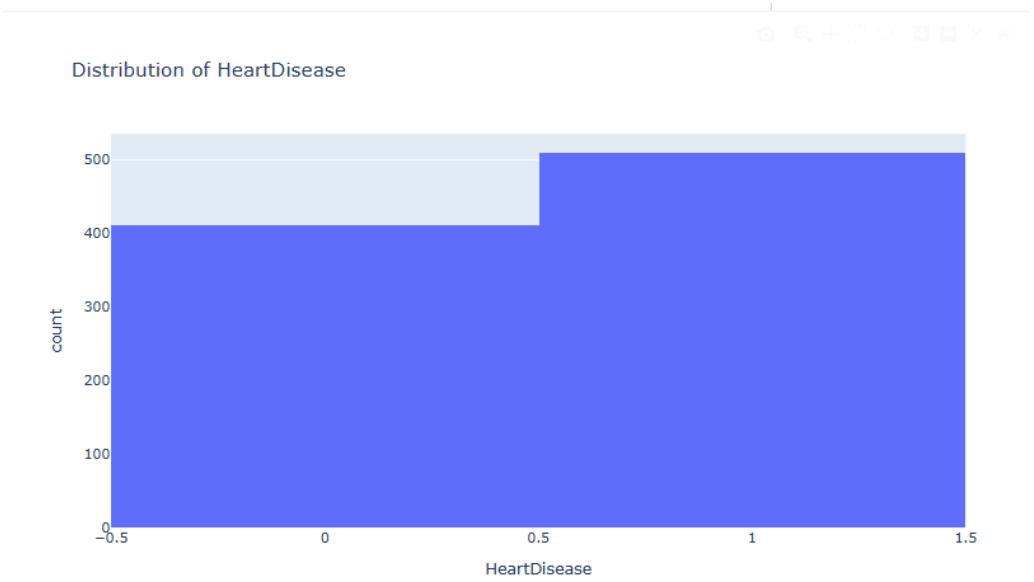
→ colab에서 작동이 안되는 문제로 px로 바꿔서 학습함

Plotly는 환경별로 그래프를 표시하는 "렌더러(renderer)"가 다름

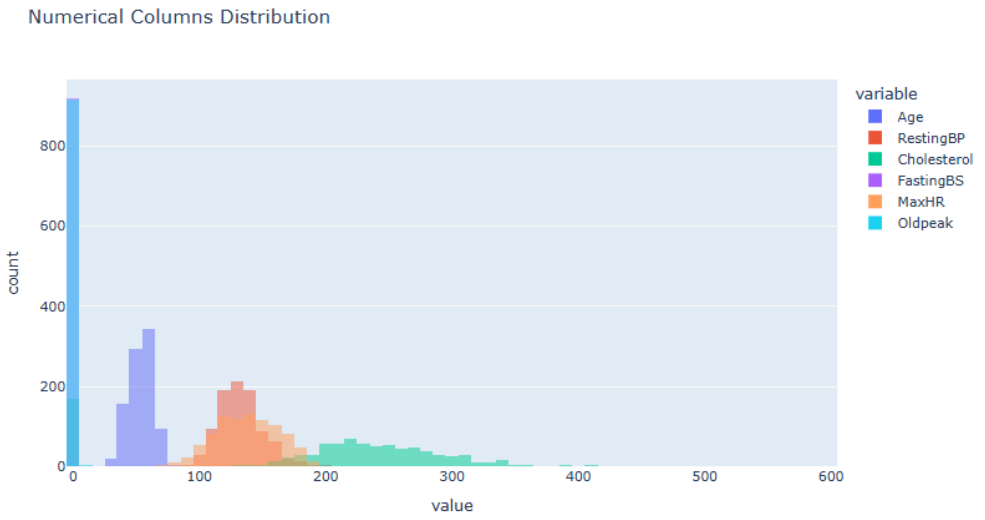
- Jupyter / Colab → `'notebook'` 또는 `'colab'`
- VSCode / PyCharm → `'browser'` 또는 `'iframe_connected'`

```
import plotly.express as px
import plotly.io as pio

# Colab 렌더러 설정
pio.renderers.default = 'colab'

# 히스토그램 그리기
```

```
fig = px.histogram(df, x="HeartDisease", title="Distribution of HeartDiseas
e")
fig.show()
```




```
pio.renderers.default = 'colab'
fig = px.histogram(df.melt(value_vars=numerical),
                   x="value",
                   color="variable",
                   barmode="overlay",
                   title="Numerical Columns Distribution")
fig.show()
```




```
pio.renderers.default = 'colab'
fig = px.histogram(df.melt(value_vars=numerical),
                   x="value",
                   color="variable",
                   facet_col_wrap=2,
                   facet_col="variable",
                   nbins=50
                  )
fig.show(height=600,
    width=700,)
```

❤️ 데이터의 왜도(skewness)를 평가하기 위한 임계 기준(threshold)

- 일반적으로 **절대값 기준으로 1 미만(-1 ~ +1)**의 왜도
- 이 값 이상이면 '왜곡이 심하다'고 판단
- **선형 모델에 적합한 수준으로 허용 가능**

왜곡이 심한 수치형 칼럼 삭제

- 수치형 칼럼에서 왜도가 심한 칼럼 삭제
- `.skew()` 각 컬럼의 **왜도(skewness)** 값을 계산

```
skew_limit = 0.75
skew_vals = df[numerical].drop('FastingBS', axis=1).skew()
skew_cols= skew_vals[abs(skew_vals)> skew_limit].sort_values(ascending=
False)
skew_cols
```

```
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);
```

수치형 칼럼 간 상관계수

- `df[numerical1].corr()` 수치형 칼럼들끼리 상관계수 계산
- np.triu(): 위쪽 삼각형만

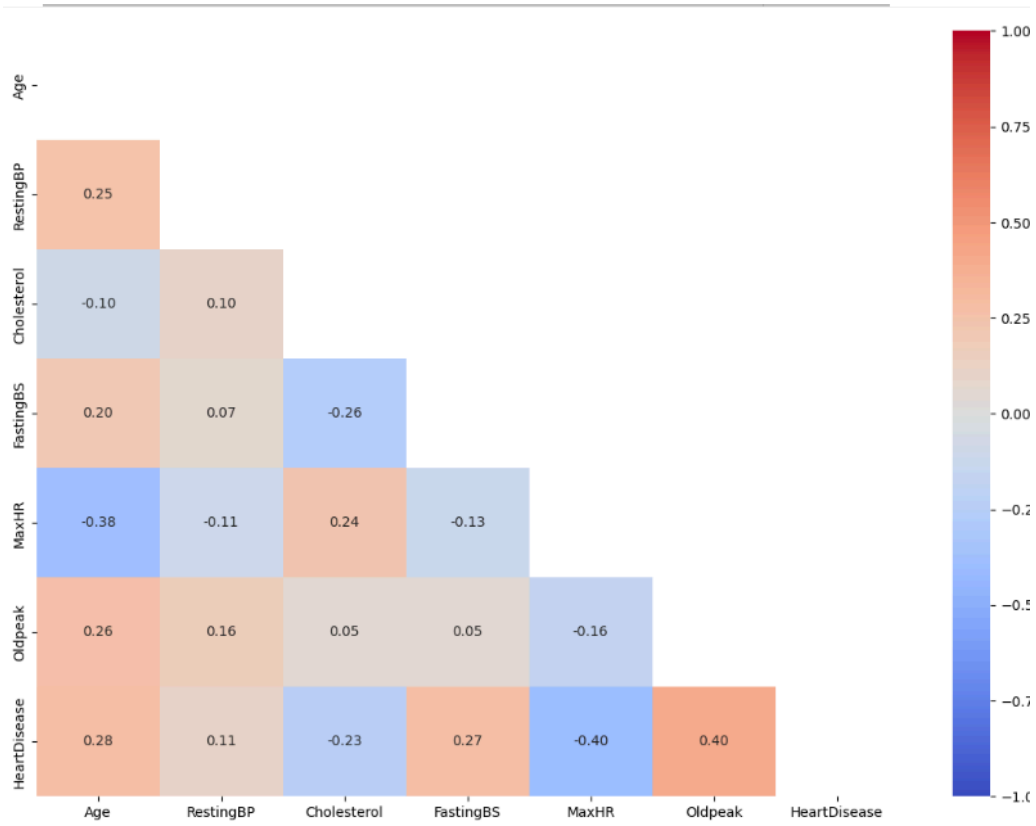
하 삼각행렬 : L_n

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

상 삼각행렬 : U_n

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

- 시본을 통해 예쁘게 시각화



피처 변수에서 특정 라벨 %

```
print (f'A female person has a probability of {round(df[df["Sex"]=="F"]["HeartDisease"].mean()*100,2)} % have a HeartDisease')

print()

print (f'A male person has a probability of {round(df[df["Sex"]=="M"]["HeartDisease"].mean()*100,2)} % have a HeartDisease')

print()
```

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

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

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

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

RestingECG	HeartDisease
ST	0.657303
LVH	0.563830
Normal	0.516304

dtype: float64

- `groupby( )[ ].mean( )` : Resting EEG 상태에 따른 심장병 발병 값들의 평균을 볼 수 있음
- `x="ChestPainType"` 은 **x축에 어떤 데이터 컬럼을 쓸지 지정**
  - x 설정 안 하면 Plotly가 **DataFrame의 모든 수치형 컬럼(numeric columns)**

  을 자동으로 찾아서 각각에 대한 히스토그램을 **겹쳐서(overlaid)** 그리려 시도

Model Selection

❤️

- Base model: Dummy classifier model
  - 모델이 아무것도 배우지 않았을 때의 예측 성능
- Logistic & Linear Discriminant & KNeighbors and Support Vector Machine models with and without scaler.
- Ensemble models
  - Bagging: Randomforest, Extra Trees
  - Boosting: Adaboost, , Gradient Boosting
    - XGBoost,LightGBM & Catboost
- Finally we will look in detail to hyperparameter tuning for Catboost

Baseline model

```
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_s
tate=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_t
```

```
est, y_pred),4)}')

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

→ 정확도: 0.5942

Logistic & Linear Discriminant & SVC & KNN

```
accuracy =[]
model_names =[]


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

	Accuracy
Logistic	0.8841
LinearDiscriminant	0.8696
SVM	0.7246
KNeighbors	0.7174

Logistic & Linear Discriminant & SVC & KNN with Scaler

```
accuracy =[]
model_names =[]


X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
tate=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','KNeighbo
rs_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
```

	Accuracy
Logistic_scl	0.8804
LinearDiscriminant_scl	0.8696
SVM_scl	0.8841
KNeighbors_scl	0.8841

다음 단계: result_df2 변수로 코드 생성 New interactive sheet

Ensemble Models

```
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_s
tate=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 : 0.8732
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

Famous Trio (XGBoost & LightGBM & Catboost)

```
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_s
tate=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
```

***CATBOOST**

범주형 데이터를 잘 다루는 부스팅 기반 분류 모델

```
accuracy =[]
model_names =[]


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

#넘파이12 이후 np.float 라는 코드 자체가 사라짐


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
tate=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
```

- 넘파이12 이후 np.float 라는 코드 자체가 사라짐

	Accuracy
Catboost_default	0.8804

파라미터 튜닝

Catboost HyperParameter Tuning with Optuna

- Optuna: 하이퍼파라미터를 자동으로 탐색(튜닝)하는 라이브러리 (베이지안 최적화 기반)
- GridSearch / RandomSearch: 직접 모든 조합 돌려봄

항목	Hyperopt	Optuna
출시 시기	2013년 (매우 오래됨)	2018년 (최신 세대)
탐색 알고리즘	TPE (Tree-structured Parzen Estimator)	TPE + 여러 샘플링/프루닝 지원
코드 스타일	함수 + `fmin()`	객체지향적 (`study.optimize()`)
학습 중단 (Pruning)	지원 X (외부 코드 필요)	✅ 자동 지원 (validation 중 early stop 가능)
시각화	별도 없음	✅ 내장 그래프 (`optuna.visualization`)
병렬 실행	가능 (Spark, MongoDB 등)	✅ 더 간단 (`n_jobs` 등으로 바로 병렬화)
통합성	scikit-learn, Keras, XGBoost 등	✅ 더 다양한 연동 기능 (lightgbm, catboost, sklearn 등 자동 wrapper)
설치 난이도	가벼움	약간 더 큼 (기능 많음)
직관성	코드 짧음	코드 구조 명확

항목	Hyperopt	Optuna
UI	없음	✅ dashboard (Optuna Dashboard, Streamlit 연동 가능)

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

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

    param = {
        "objective": trial.suggest_categorical("objective", ["Logloss", "CrossEn
tropy"]),
        "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", ["Ordere
d", "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_tempera
ture", 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=categ
orical_features_indices,verbose=0, early_stopping_rounds=100)

    preds = cat_cls.predict(X_test)
    pred_labels = np.rint(preds)
```

```
    accuracy = accuracy_score(y_test, pred_labels)
    return accuracy


if __name__ == "__main__":
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50, timeout=600)

    print("Number of finished trials: {}".format(len(study.trials)))

    print("Best trial:")
    trial = study.best_trial

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

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

```
[I 2025-10-06 06:04:50,997] Trial 35 finished with value: 0.894927536231884 and parameters: {'objective': 'Log
[I 2025-10-06 06:04:52,683] Trial 36 finished with value: 0.8913043478260869 and parameters: {'objective': 'Lo
[I 2025-10-06 06:04:54,358] Trial 37 finished with value: 0.8876811594202898 and parameters: {'objective': 'Lo
[I 2025-10-06 06:04:55,845] Trial 38 finished with value: 0.8876811594202898 and parameters: {'objective': 'Cr
[I 2025-10-06 06:04:59,356] Trial 39 finished with value: 0.8913043478260869 and parameters: {'objective': 'Lo
[I 2025-10-06 06:05:01,583] Trial 40 finished with value: 0.8985507246376812 and parameters: {'objective': 'Lo
[I 2025-10-06 06:05:03,662] Trial 41 finished with value: 0.8840579710144928 and parameters: {'objective': 'Lo
[I 2025-10-06 06:05:05,590] Trial 42 finished with value: 0.894927536231884 and parameters: {'objective': 'Log
[I 2025-10-06 06:05:07,219] Trial 43 finished with value: 0.894927536231884 and parameters: {'objective': 'Log
[I 2025-10-06 06:05:08,847] Trial 44 finished with value: 0.894927536231884 and parameters: {'objective': 'Log
[I 2025-10-06 06:05:11,203] Trial 45 finished with value: 0.8913043478260869 and parameters: {'objective': 'Lo
[I 2025-10-06 06:05:14,364] Trial 46 finished with value: 0.8695652173913043 and parameters: {'objective': 'Lo
[I 2025-10-06 06:05:15,886] Trial 47 finished with value: 0.8876811594202898 and parameters: {'objective': 'Lo
[I 2025-10-06 06:05:17,538] Trial 48 finished with value: 0.8840579710144928 and parameters: {'objective': 'Cr
[I 2025-10-06 06:05:19,497] Trial 49 finished with value: 0.8913043478260869 and parameters: {'objective': 'Lo
Number of finished trials: 50
Best trial:
  Value: 0.8985507246376812
  Params:
    objective: CrossEntropy
    colsample_bylevel: 0.02683225697907876
    depth: 3
    boosting_type: Plain
    bootstrap_type: MVS
```

❤️ **Parameters:**
Objective: Supported metrics for overfitting detection and best model selection
colsample_bylevel: this parameter speeds up the training and usually does not affect the quality.
depht : Depth of the tree.
boosting_type : By default, the boosting type is set to for small datasets. This prevents overfitting but it is expensive in terms of computation. Try to set the value of this parameter to to speed up the training.
bootstrap_type : By default, the method for sampling the weights of objects is set to . The training is performed faster if the method is set and the value for the sample rate for bagging is smaller than 1.

CatBoost 모델의 하이퍼파라미터 튜닝 후 성능 평가 파이프라인

```
accuracy =[]
model_names =[]

###피처 데이터, 타겟 레이블 분리
X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
###카테고리형 피처 인덱스
categorical_features_indices = np.where(X.dtypes != float)[0]

#학습/테스트 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
tate=42)
#캣부스트분류
model = CatBoostClassifier(verbose=False,random_state=0,
                           objective= 'CrossEntropy',
    colsample_bylevel= 0.04292240490294766,
    depth= 10,
    boosting_type= 'Plain',
    bootstrap_type= 'MVS')
#학습/예측/성능평가
model.fit(X_train, y_train,cat_features=categorical_features_indices,eval_se
```

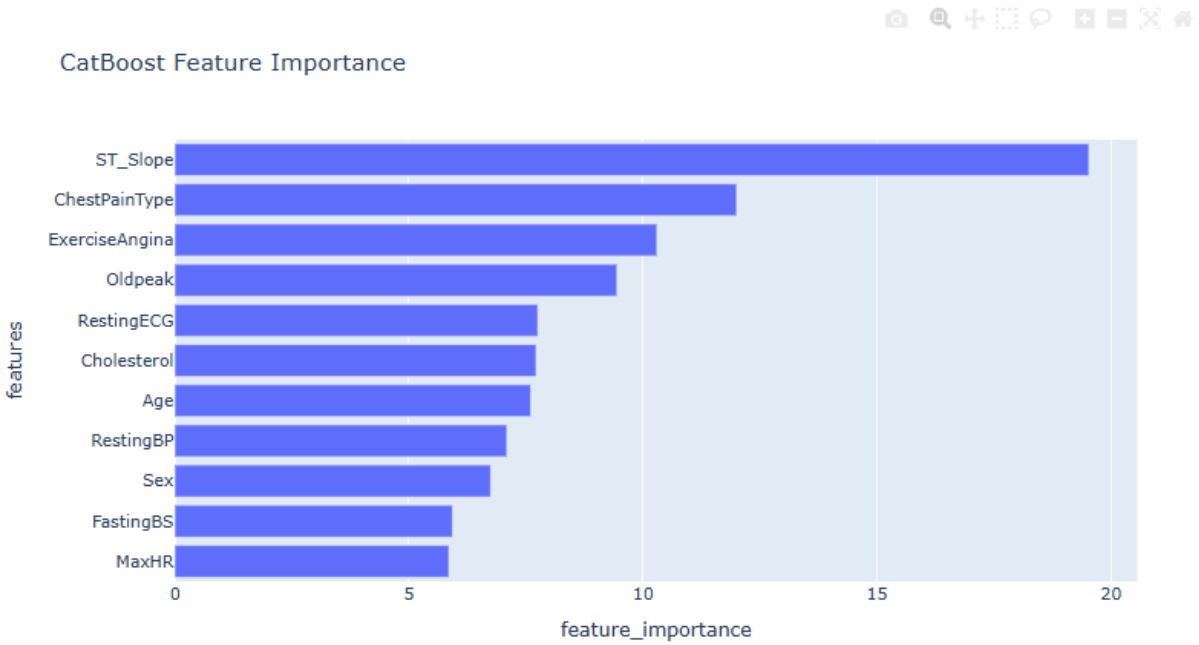
```
t=(X_test, y_test))
y_pred = model.predict(X_test)
accuracy.append(round(accuracy_score(y_test, y_pred),4))
#classification_report() → precision, recall, F1-score까지 함께 출력
print(classification_report(y_test, y_pred))

##결과를 DataFrame으로 정리해서 모델명 vs 정확도 형태로 저장
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

피처 중요도

```
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 Feat
ure Importance",height=500)
fig.show()
```

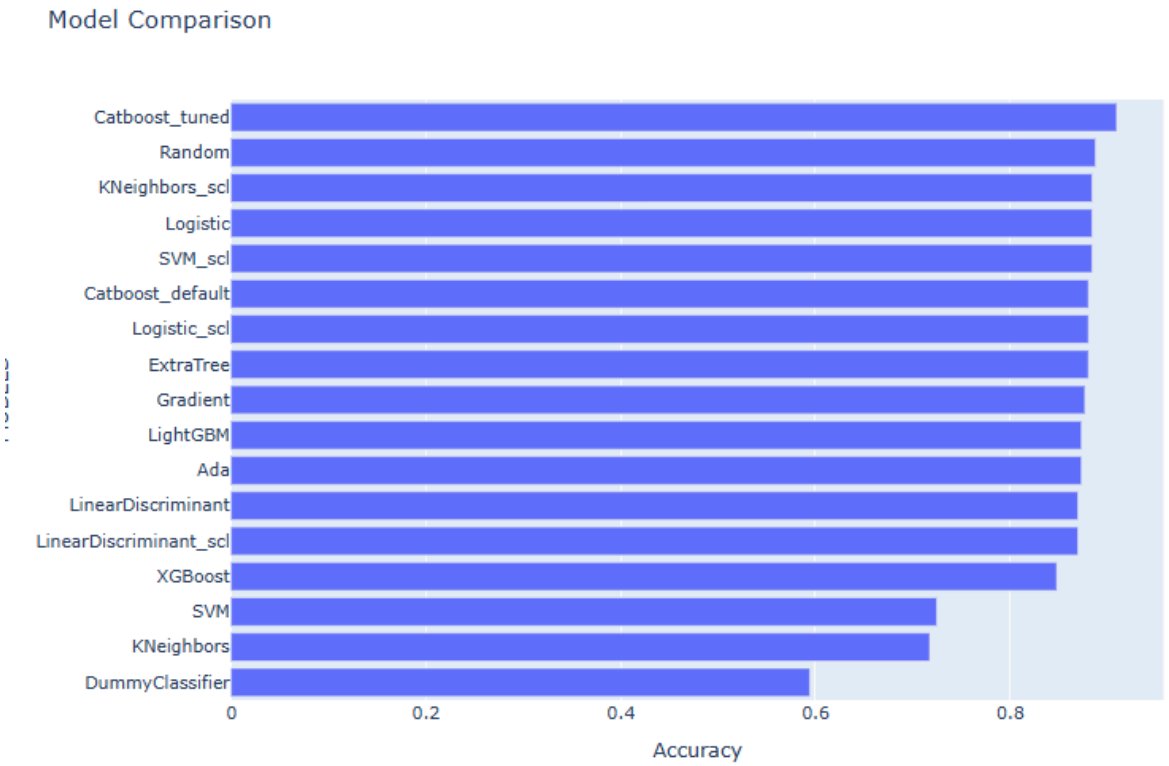


모델 비교

- `pd.concat():` pandas의 **concat(=concatenate)** 함수는 여러 `DataFrame`을 **합치는 함수**

```
result_final = pd.concat([dummy_result_df,result_df1,result_df2,result_df3,r
esult_df4,result_df5,result_df6],axis=0)
```

```
result_final.sort_values(by=['Accuracy'], ascending=True,inplace=True)
fig = px.bar(result_final, x='Accuracy', y=result_final.index,title='Model Com
parison',height=600,labels={'index':'MODELS'})
fig.show()
```


Model Comparison
Catboost_tuned
Random
KNeighbors_scl
Logistic
SVM_scl
Catboost_default
Logistic_scl
ExtraTree
Gradient
LightGBM
Ada
LinearDiscriminant
LinearDiscriminant_scl
XGBoost
SVM
KNeighbors
DummyClassifier
0
0.2
0.4
0.6
0.8
Accuracy

Decision Tree And Random Forest Classifier Models

https://www.kaggle.com/code/ihsncnkz/decision-tree-and-random-forest-classifier-models#Classifications-Models-

Data Review

약물 데이터.csv

- 나이, 성별, Blood Pressure, Cholesterol levels, Na 등 → 사용하는 약물 예측

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt ##정적인 그래프
import seaborn as sns #matplotlib 고급 버전

import plotly.graph_objects as go ##저수준 수동 제어(축, 색, 레이아웃 등)
import plotly.express as px ##자동화 됨 -> 줌, 클릭 등 가능한 그래프
```

`data.info()` , `describe()` , `corr()`

```
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
```

data.describe()

```
1 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

data_corr = data.select_dtypes(include=['float', 'int']).corr()
data_corr

```
1 data.select_dtypes(include=['float', 'int']).corr()
```

	Age	Na_to_K
Age	1.000000	-0.063119
Na_to_K	-0.063119	1.000000

- `describe()` : 자동으로 숫자형 타입만

- `corr()` : 문자열 칼럼은 처리 X

sns.heatmap(data.corr())

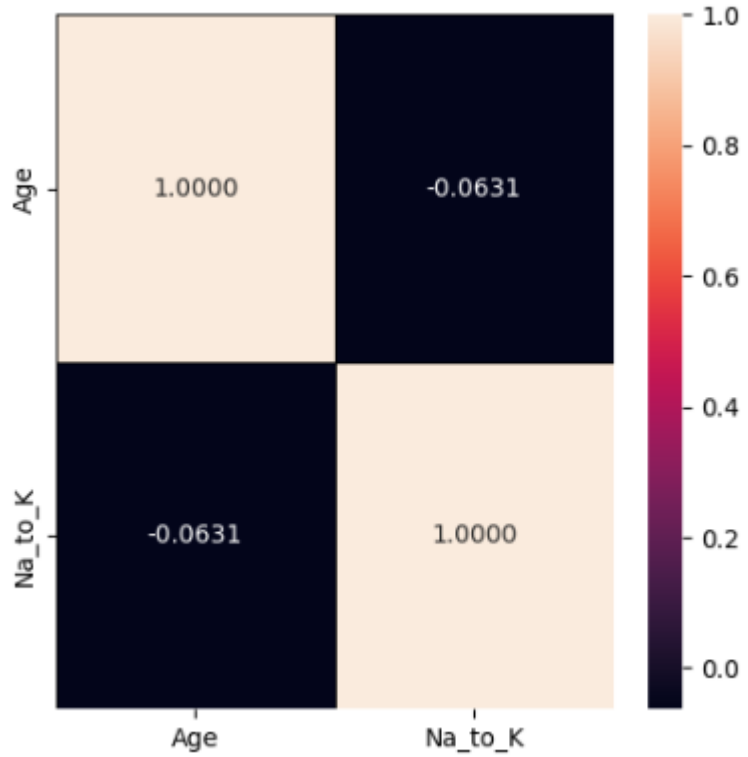
: 시본으로 상관관계 보여주는 히트맵

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

`annot=True` : 각 셀 안에 숫자(상관계수)를 표시

`ax=ax` 특정 matplotlib Axes 객체에 그림을 그리기

`linewidths=0.5` 셀 사이의 경계선 두께

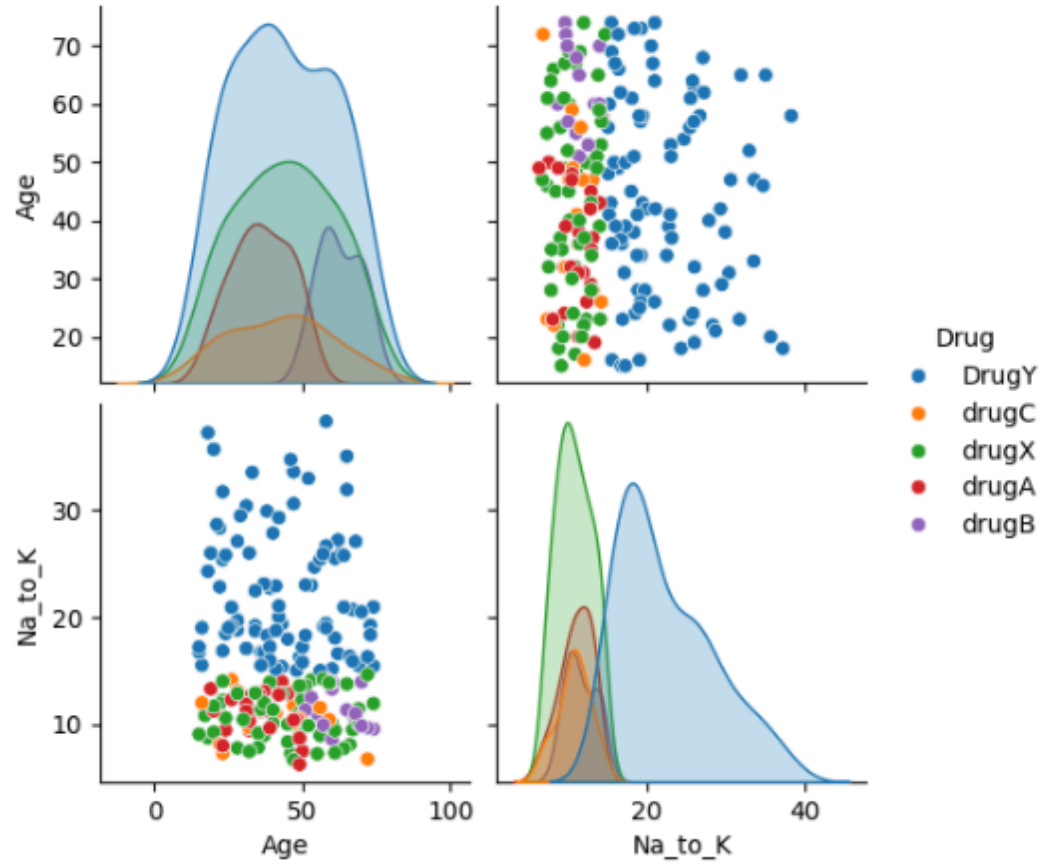


sns.pairplot(df)

모든 수치형 변수들의 쌍(pair)별 관계를 한꺼번에 시각화

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

```
<seaborn.axisgrid.PairGrid at 0x784749424380>
```



```
data.columns
```

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

⭐ 모델 학습에는 문자열형 데이터를 사용할 수 없기 때문에 열(column) 안의 데이터를 확인해야 한다.

data[''].value_counts()

칼럼별 특정 값이 얼마나 많은지 세줌

```
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

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

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

Sex	count
M	104
F	96

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

BP	count
HIGH	77
LOW	64
NORMAL	59

dtype: int64

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

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

Cholesterol	count
HIGH	103
NORMAL	97

dtype: int64

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

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

Drug	count
DrugY	91
drugX	54
drugA	23
drugC	16
drugB	16

dtype: int64

Data Visualization

❤️ Bar Chart 막대 그래프 시각화

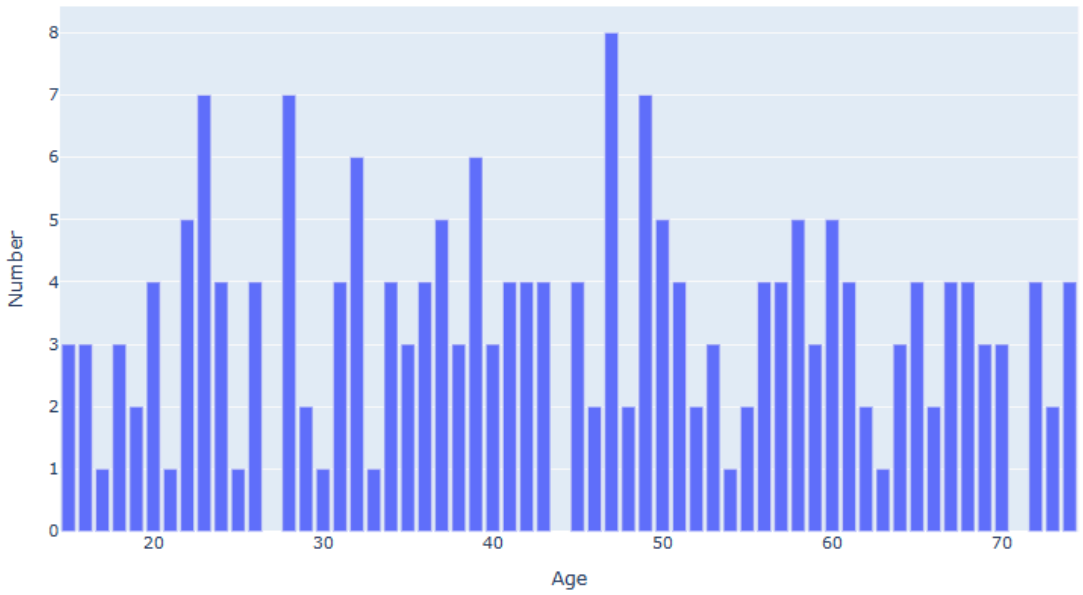
- the number of age in the dataset.
- **막대그래프(bar chart)** 를 만드는 코드
- `value_counts()` 결과(Series)를 `numpy.ndarray`로 바꾸면 👉**값(value) 부분만 남고 인덱스(index)는 사라짐**

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

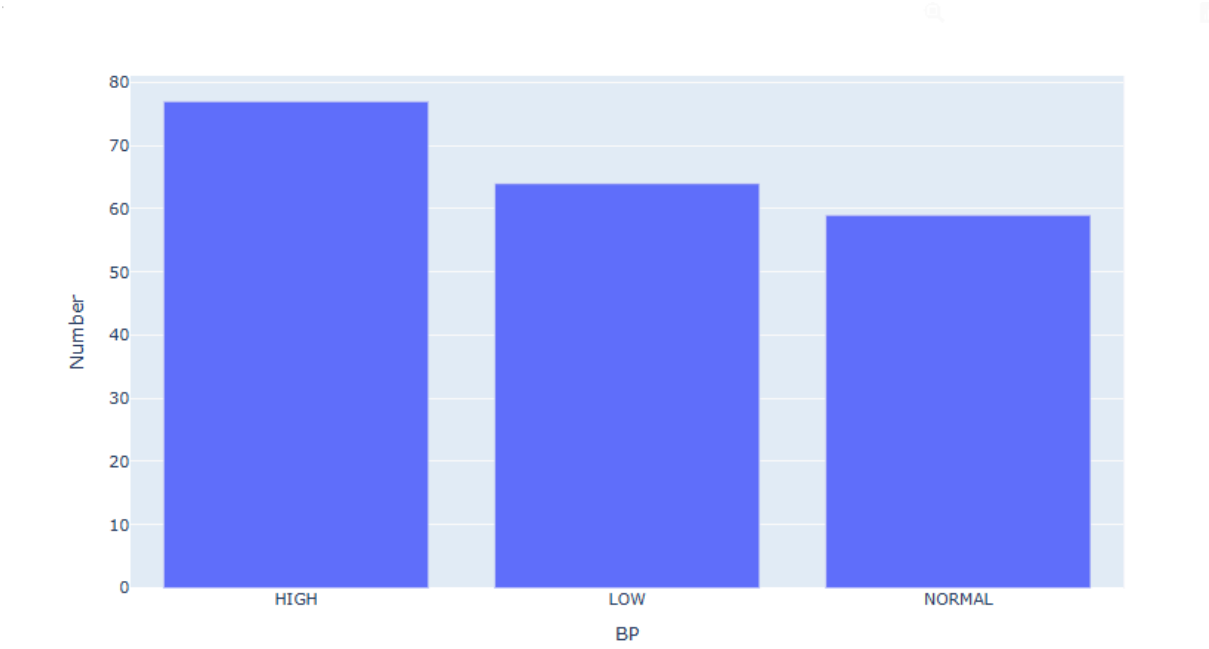
- **꼭 DataFrame만 넣을 필요는 없다.**
- **하지만** `Plotly Express(px)` **에서는 DataFrame 형태를 기본 전제로 설계**되어 있어서, DataFrame을 쓰는 게 가장 **안전하고 편리**

px.histogram(x = [], y = [])

```
##y축: Age 범주별 수

dataAge = data["Age"].value_counts(dropna = False)
npar_dataAge = np.array(dataAge)
##x축에 표시할 숫자들로 array 형태로 바꿈(인덱스 사라짐. 값만 남음)
x = list(npar_dataAge)
##x축에 표시할 숫자들로 list로 바꿈

##x축: Age 범주들
y = data.Age.value_counts().index
#나이 값들(25, 30, 20)반환

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

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



```
# the number of BP(Blood Pressure Levels)
fig = px.bar(x = ["HIGH", "LOW", "NORMAL"], y = [77, 64, 59])

fig.show()
```



❤️원형 그래프(Pie Chart)

`go.Figure()`

- 여러 trace(조각)들을 하나의 그래프 객체로 묶고, 레이아웃(layout), 제목, 축, 크기 등 전체 설정을 담당하는 **컨테이너(container)**

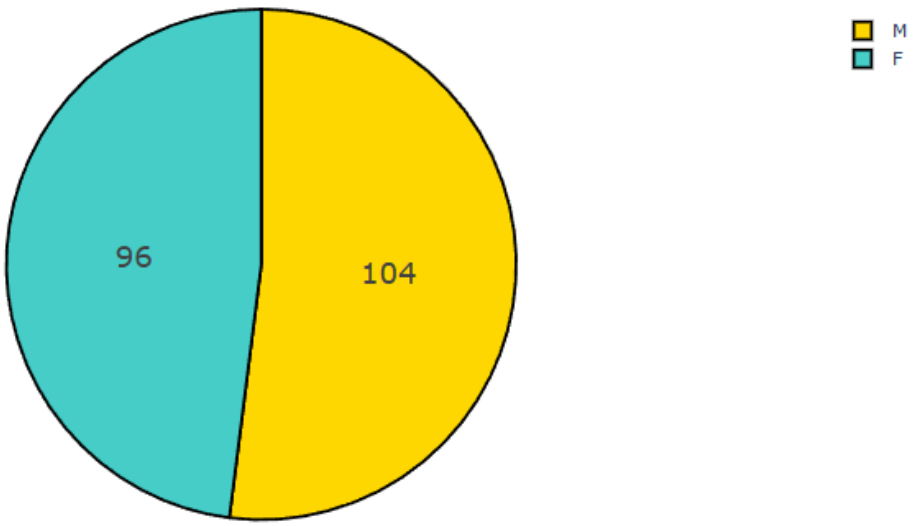
`go.Pie()` : 원형 그래프 생성

- `labels` → 각 조각 이름 (“M”, “F”)
- `values` → 각 조각의 크기 (104명, 96명)

`fig.update_traces` : 그래프 디자인

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

#원형 그래프 생성
fig = go.Figure(data = [go.Pie(labels= ['M', 'F'], values=[104, 96])])
```

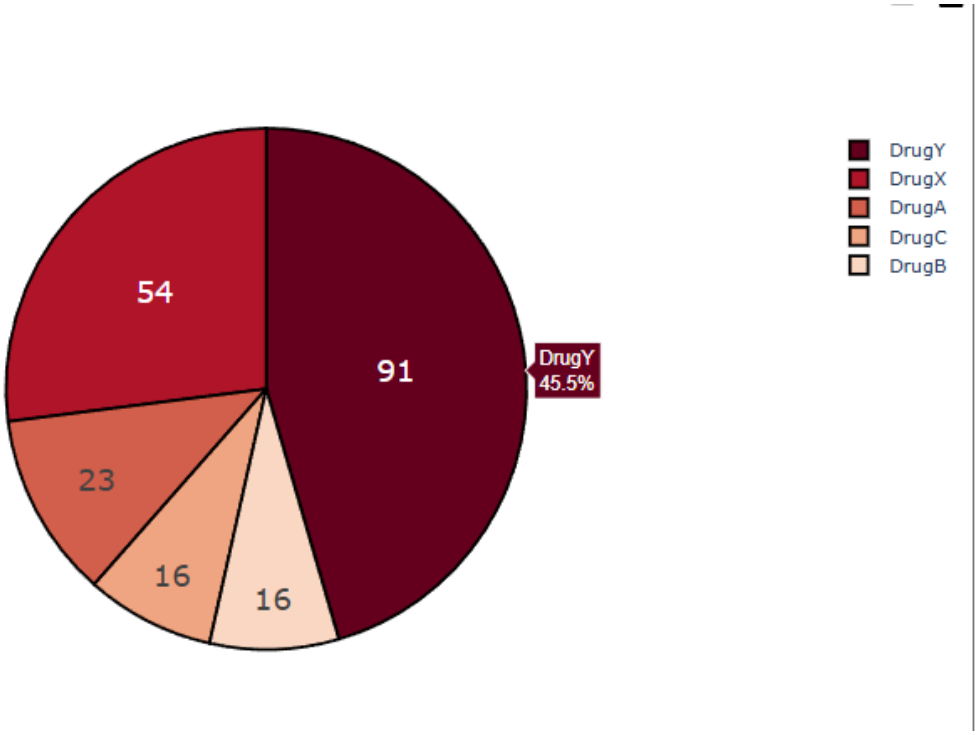
```
#그래프 디자인 설정
fig.update_traces(hoverinfo = 'label + percent', # 마우스 올리면 라벨과 퍼센트 표시
                  textinfo = 'value', # 조각 안쪽 text→ 실제 숫자 표시 (104, 96)
                  textfont_size = 20,  # 글씨 크기
                marker = dict(colors = colors,
                line = dict( color = '#000000', width = 2)))
                # 조각 테두리: 검정색, 두께 2

fig.show()
```




```
#the number of Cholesterol in the datase

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

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

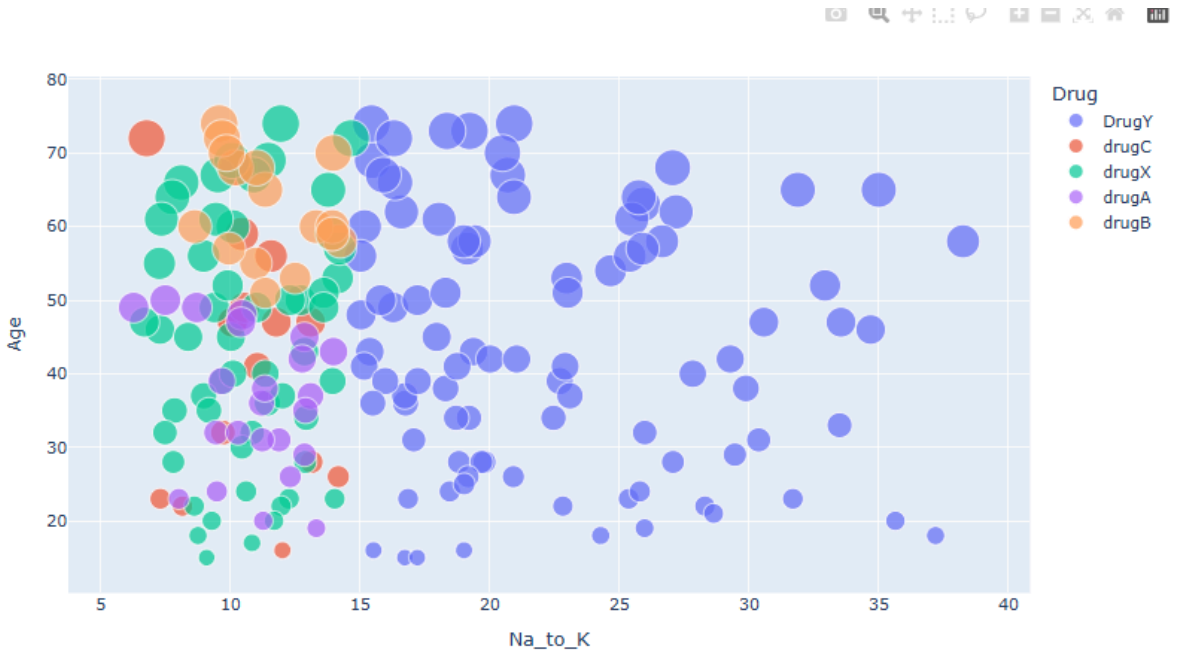
```
fig.update_traces(hoverinfo = 'label + percent', textinfo = 'value', textfont_size = 20,
                  marker = dict(colors = px.colors.sequential.RdBu, line = dict( color = '#000000', width = 2)))

fig.show()
```



❤️Plotly Express의 산점도(scatter plot)

- `px.scatter()`

```
fig = px.scatter(data, # 데이터프레임
                 x = "Na_to_K", y="Age", color="Drug",
                 size='Age', hover_data=['Na_to_K']) # 마우스를 올렸을 때 표시할 추가 정보
fig.show()
```



Classifications Models

Data Preparing

```
dataclass = pd.read_csv("/content/drive/MyDrive/kaggle/drug200.csv")
```

```
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    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
```

```
#나이 변수 -> 남 0 여 1
```

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

```
dataclass
```

1 dataclass

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

200 rows × 6 columns

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

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

200 rows × 6 columns

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

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

200 rows × 6 columns

```
# 약물이름 변경
# DrugY = 4
# DrugX = 3
# DrugA = 2
# DrugC = 1
# DrugB = 0

import warnings
warnings.filterwarnings('ignore')

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
```

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

200 rows × 6 columns

```
#재확인
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
```

→ BP, Drug에 안 바뀐 게 있음

- `.astype(dict)`

```
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
```

```
#레이블과 피처 데이터 분리

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

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

```
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

200 rows × 5 columns

y_data

1 y_data

	Drug
0	4
1	1
2	1
3	3
4	4
...	...
195	1
196	1
197	3
198	3
199	3

200 rows × 1 columns

```
from sklearn.model_selection import train_test_split
#학습/검증 데이터
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3,
random_state=1)
```

Decision Tree Classifier

```
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(predic
t,y_test))
```

`The accuracy of the Decision Tree is 0.9666666666666667`

Decision Tree Classifier with "gini"

- `DecisionTreeClassifier(criterion='gini')`
  - **노드를 분할(split)하는 기준**을 `'gini'`로 지정
  - 'entropy'라는 다른 옵션도 있음

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


# 학습
```

```
DTC_gini.fit(x_train, y_train)
y_pred_gini = DTC_gini.predict(x_test)
```

```
from sklearn.metrics import accuracy_score

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

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

```
y_pred_train_gini = DTC_gini.predict(x_train)

y_pred_train_gini
```

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

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

⇒ `Training-set accuracy score: 0.9143`

```
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)))
```

Training set score: 0.9143
Test set score: 0.9000

Decision Tree Classifier with "entropy"

```
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: {0:0.4f}'. format(accura
cy_score(y_test, y_pred_en)))
```

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

```
y_pred_train_en = DTC_en.predict(x_train)

y_pred_train_en
```

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

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

```
Training-set accuracy score: 0.9143
```

```
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)))
```

```
Training set score: 0.9143
Test set score: 0.9000
```

Random Forest Classifier

- 최고의 random_state

```
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))
```

```
The accuracy of the Random Forest is 0.95
```

Random Forest Classifier with "n_estimators=100"¶

```
from sklearn.ensemble import RandomForestClassifier

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

# 학습
rfc_100.fit(x_train, y_train)

# 예측
predict = rfc_100.predict(x_test)

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

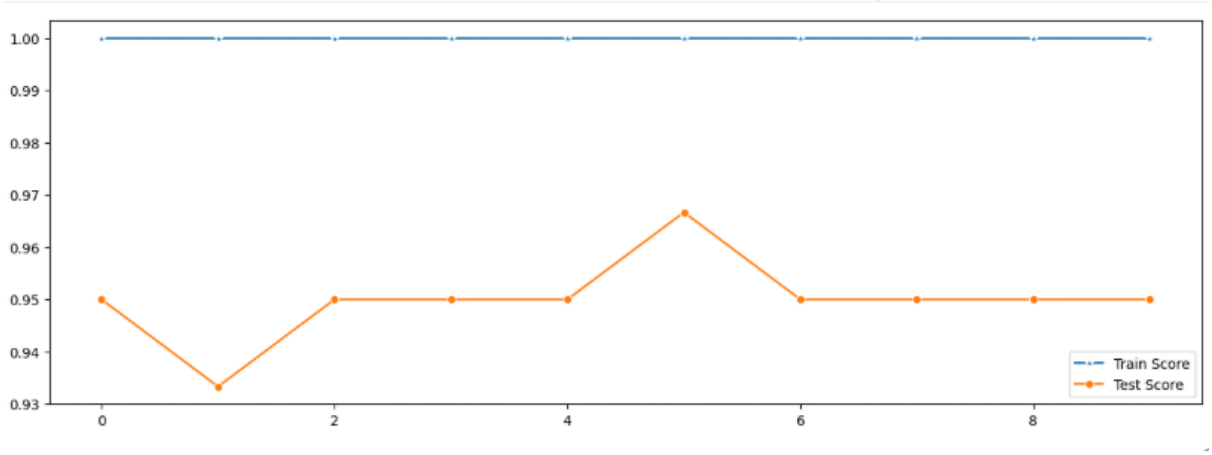
⇒ The accuracy of the Random Forest is 0.95

```
# 최고의 random_state value
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(range(0,10),train_score_list,marker='*',label='Train Score')
p = sns.lineplot(range(0,10),test_score_list,marker='o',label='Test Score')
```

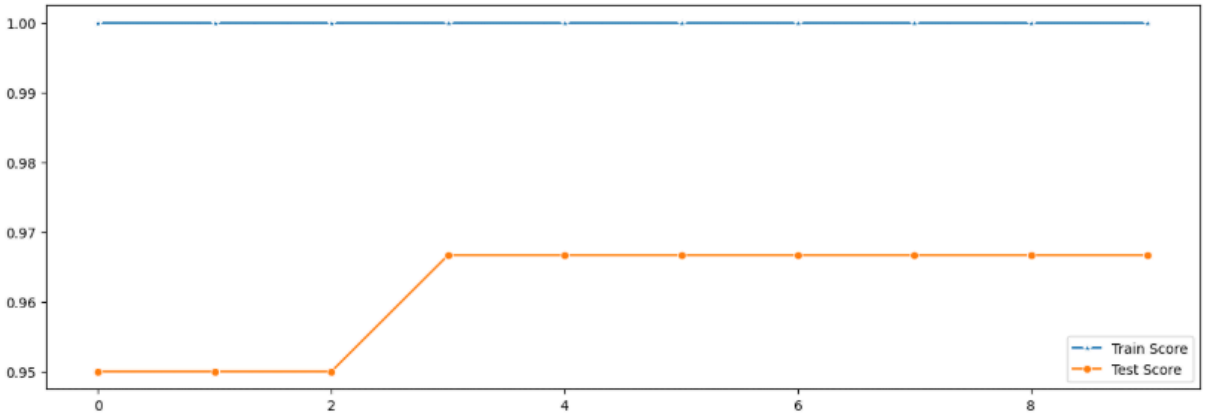
- `sns.lineplot(data=df, x='x열이름', y='y열이름')` : **Seaborn** 라이브러리에서 “선 그래프(line chart)”를 그리는 함수
    - **x, y 인자를 "위치 인자(positional arguments)로 받지X**

```
# 최고의 random_state value
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))

#x, y 인자를 "위치 인자(positional arguments)로 받지X

p = sns.lineplot(x=range(0,10),y=train_score_list,marker='*',label='Train Sc
ore')
```

```
p = sns.lineplot(x=range(0,10),y=test_score_list,marker='o',label='Test Scor
e')
```




```
#the best n_estimators parameter.
test_score_list = []
train_score_list = []

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


for i in range(0,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,len(list_n_estimators)),y=train_score_list,marker
='*',label='Train Score')
p = sns.lineplot(x=range(0,len(list_n_estimators)),y=test_score_list,marker
='o',label='Test Score')
```



Random Forest Classifier → 최고의 파라미터로 재학습 → train, test 예측 정확도

```
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))
```

```
The accuracy of the Random Forest is 0.9666666666666667
```

```
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)))
```

```
Model accuracy score with best parameters: 0.9667
```

```
y_pred_train_en = last_rfc.predict(x_train)

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

```
Training-set accuracy score: 1.0000
```

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

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

```
Training set score: 1.0000
Test set score: 0.9667
```

Evaluation Classification Models

- 여러개의 metrics가 사용
  - performance metrics or evaluation metrics.
- **Confusion Matrix**
  - `from sklearn.metrics import confusion_matrix`
  - `confusion_matrix( y_true, y_pred_cm)`

Confusion Matrix

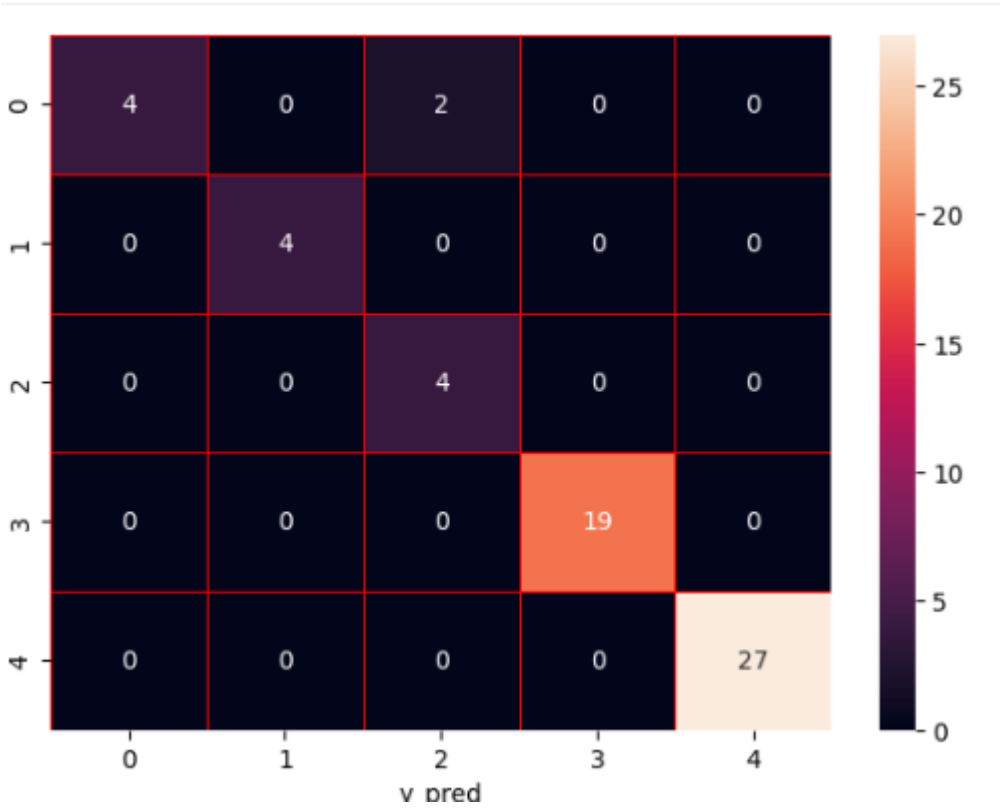
	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

- prediction outcomes of any binary classifier, which is used to describe the performance of the classification model on a set of test data when true values are known

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

cm_des = DecisionTreeClassifier()

# 학습
cm_des.fit(x_train,y_train)
#예측
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]])
```

```
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()
```

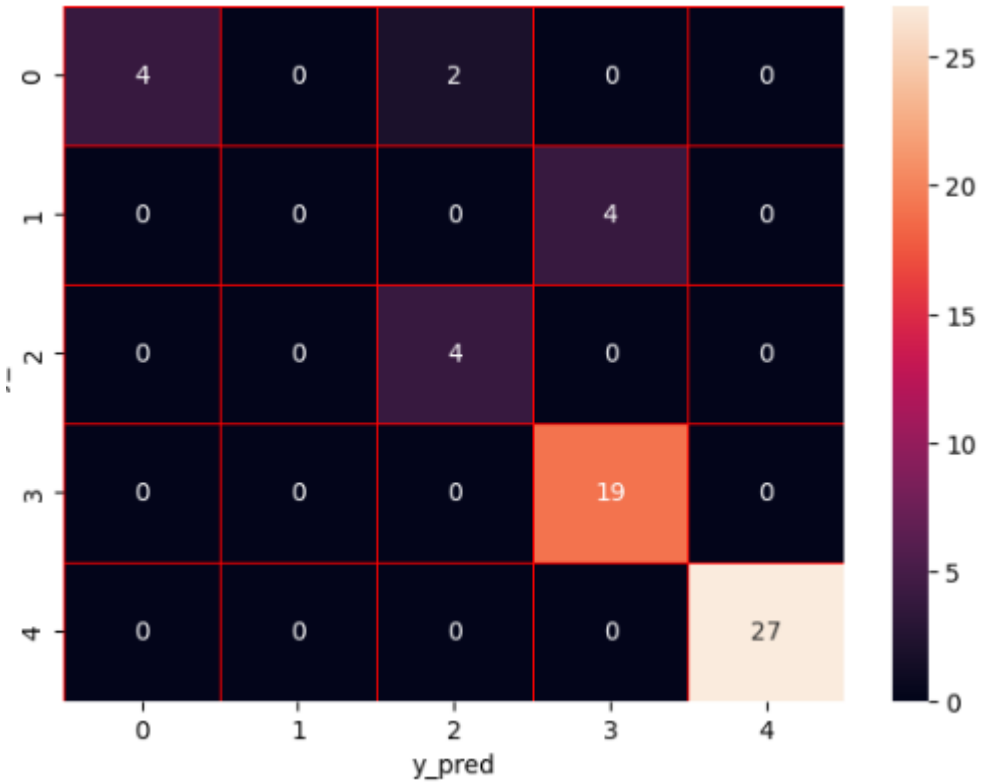


Confusion Matrix For Decision Tree Classifier With "gini"¶

```
cm_des_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, rando
m_state=0)

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

y_pred_cm = cm_des_gini.predict(x_test)
y_true = y_test
```

```
cm_des2 = confusion_matrix( y_true, y_pred_cm)
cm_des2
```

```
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()
```



- **대각선 값들 (`4, 4, 4, 19, 27`)** → 각 클래스에서 **정확히 맞춘 개수**
- **비대각선 값 (`2` in first row)**→ 실제 클래스 0의 데이터 중 **2개를 다른 클래스로 잘못 예측**

Confusion Matrix For Random Forest Classifier With The Best Parameters

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

```
# Fit The Model
cm_last_rfc.fit(x_train, y_train)

y_pred_cm = cm_last_rfc.predict(x_test)
y_true = y_test

cm_rfc = confusion_matrix(y_true, y_pred_cm)
cm_rfc
```

```
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()
```