



[캐글 노트북]

[통계](#) [수정](#) [삭제](#)

진규빈 · 방금 전

0

파이썬 머신러닝 완벽 가이드

[▼ 목록 보기](#)

5/5



Decision Tree And Random Forest Classifier Models

[링크 클릭](#)

Data Review

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
data = pd.read_csv(r'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

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
```

- 데이터 파악

v kyubinwrld.log



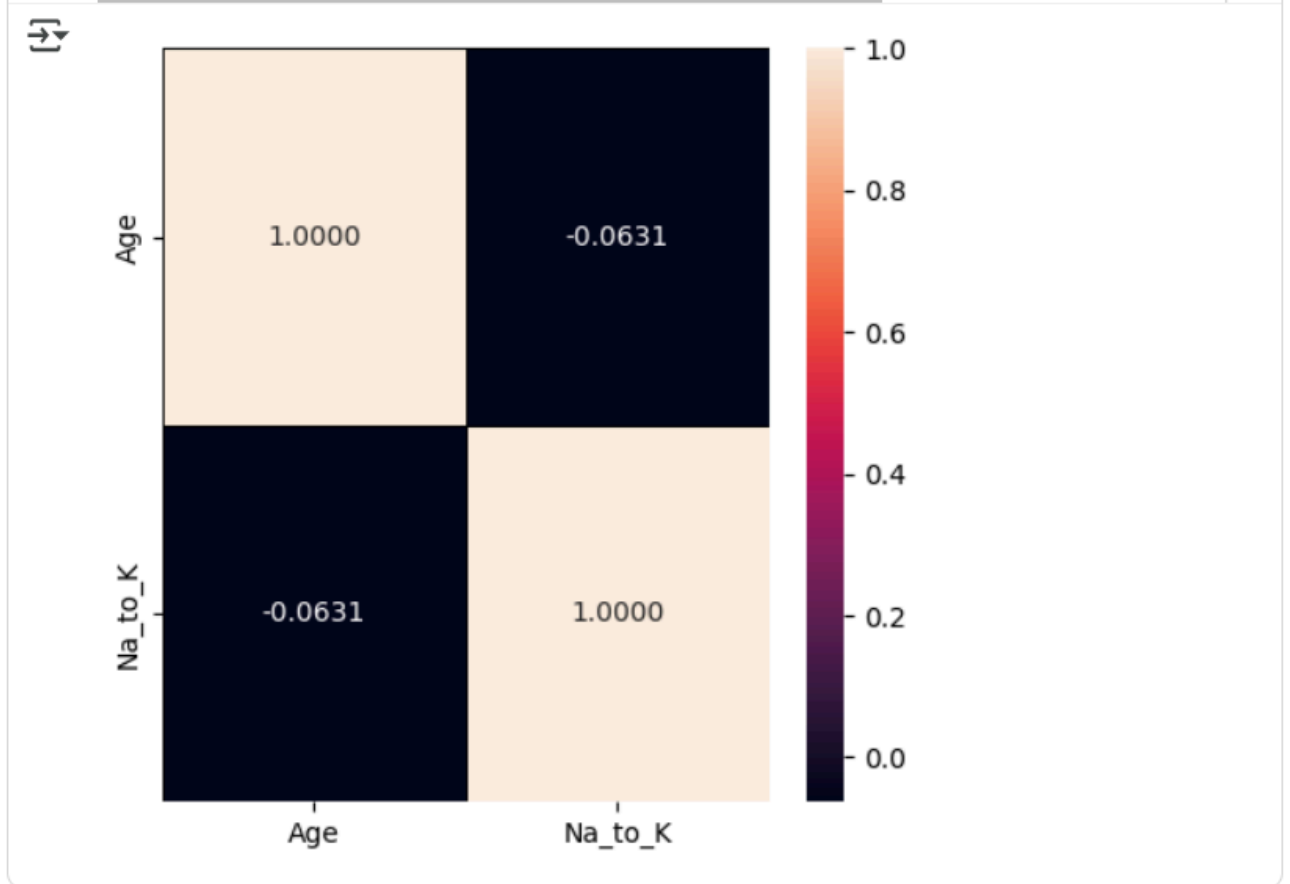
	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

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

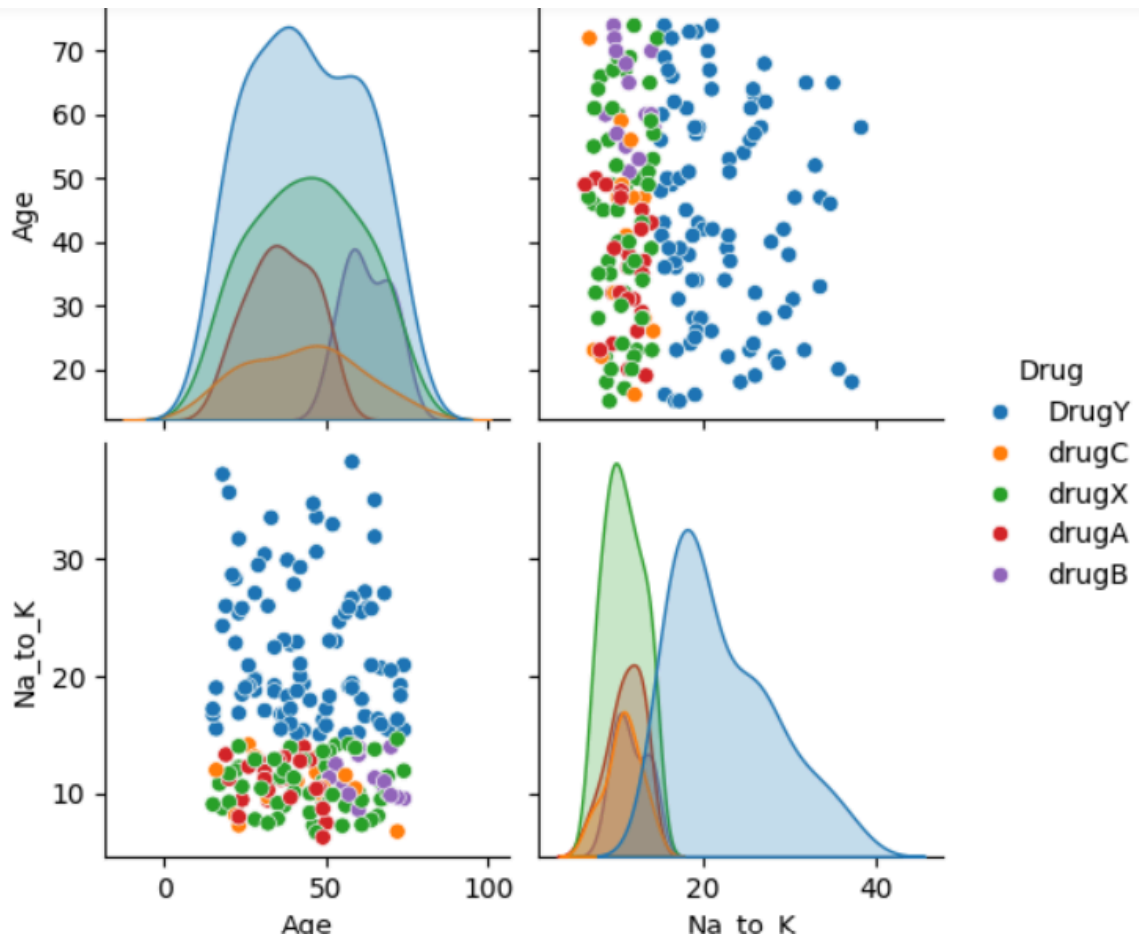


	Age	Na_to_K
Age	1.000000	-0.063119
Na_to_K	-0.063119	1.000000

- 데이터셋 values 파악 (max, min, std 등등)
- 데이터 correlation, 즉 컬럼 간 관계 파악
(괄호 안에 `numeric_only=True` 넣어서 수치형만!)



- seaborn library heatmap 사용해서 correlation 시각화



- seaborn library pairplot 사용해서 correlation 시각화

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



count	
Age	
47	8
23	7
28	7
49	7
32	6
39	6
50	5
60	5
22	5
37	5
58	5

- 데이터셋 칼럼명 파악
- Age 칼럼의 데이터 파악

v kyubinwrld.log



count

Sex

M	104
---	-----

F	96
---	----

dtype: int64

data["BP"].value_counts()



count

BP

HIGH	77
------	----

LOW	64
-----	----

NORMAL	59
--------	----

dtype: int64

- Sex 칼럼의 데이터 파악
- BP(Blood Pressure Levels) 칼럼의 데이터 파악

Cholesterol

HIGH 103

NORMAL 97

dtype: int64

data["Drug"].value_counts()



count

Drug

DrugY 91

drugX 54

drugA 23

drugC 16

drugB 16

dtype: int64

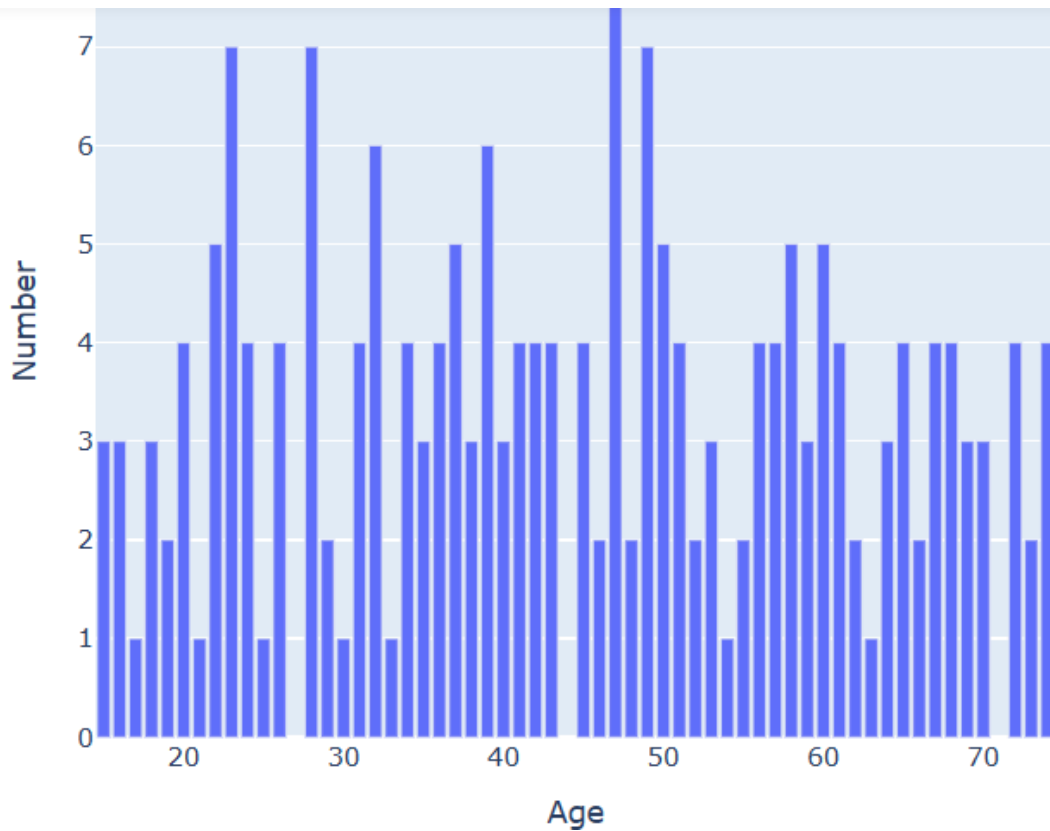
- Cholesterol 칼럼의 데이터 파악
- Drug 칼럼의 데이터 파악

Data Visualization

```
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()
```



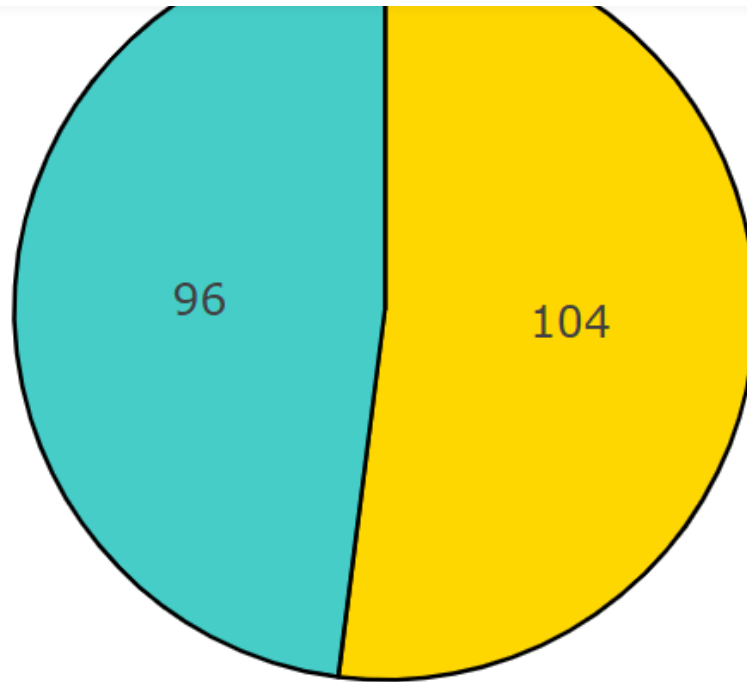
- Age 값 빈도를 막대그래프로 시각화

```
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)))
```

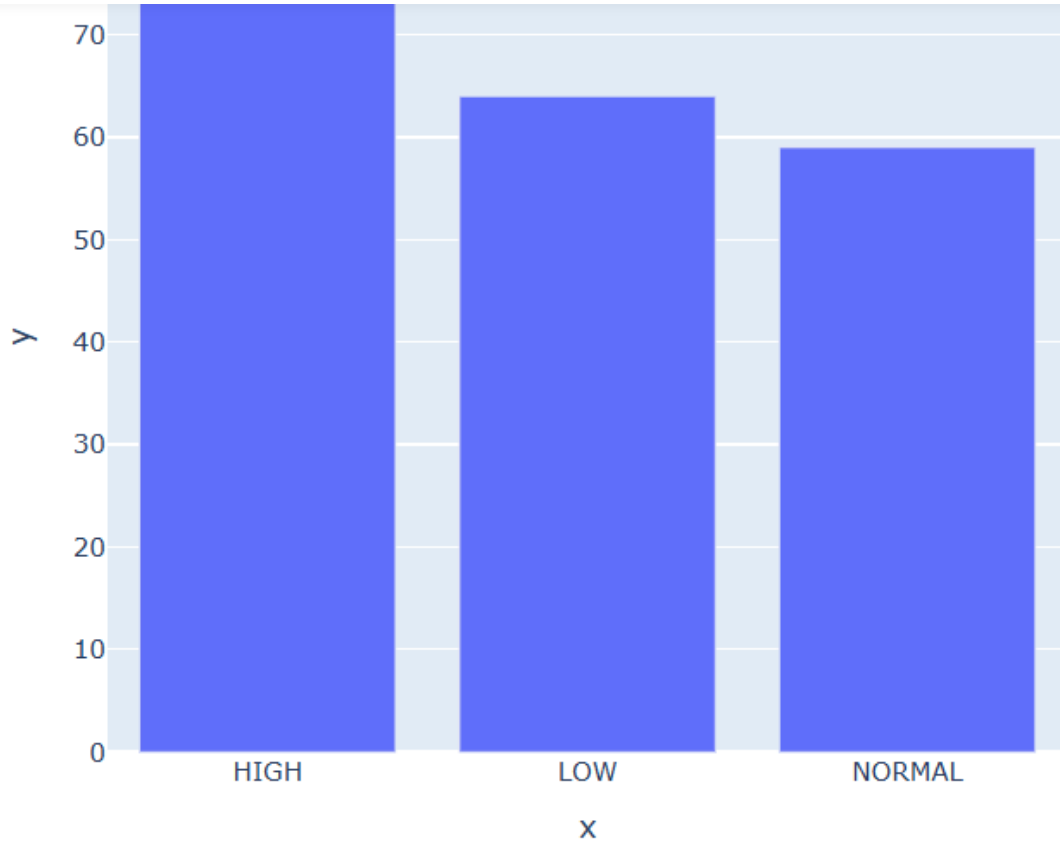
```
fig.show()
```



- Sex 분포를 파이 차트로 시각화

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

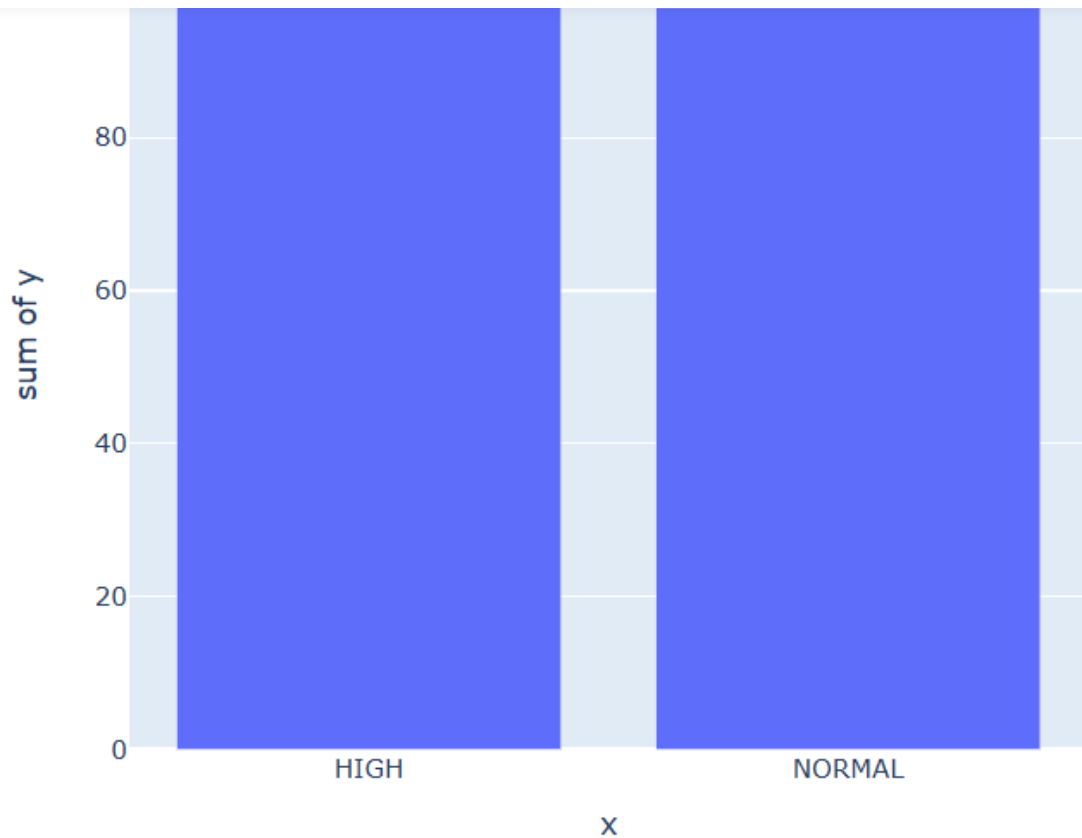
```
fig.show()
```



- BP 범주별 개수를 막대그래프로 시각화

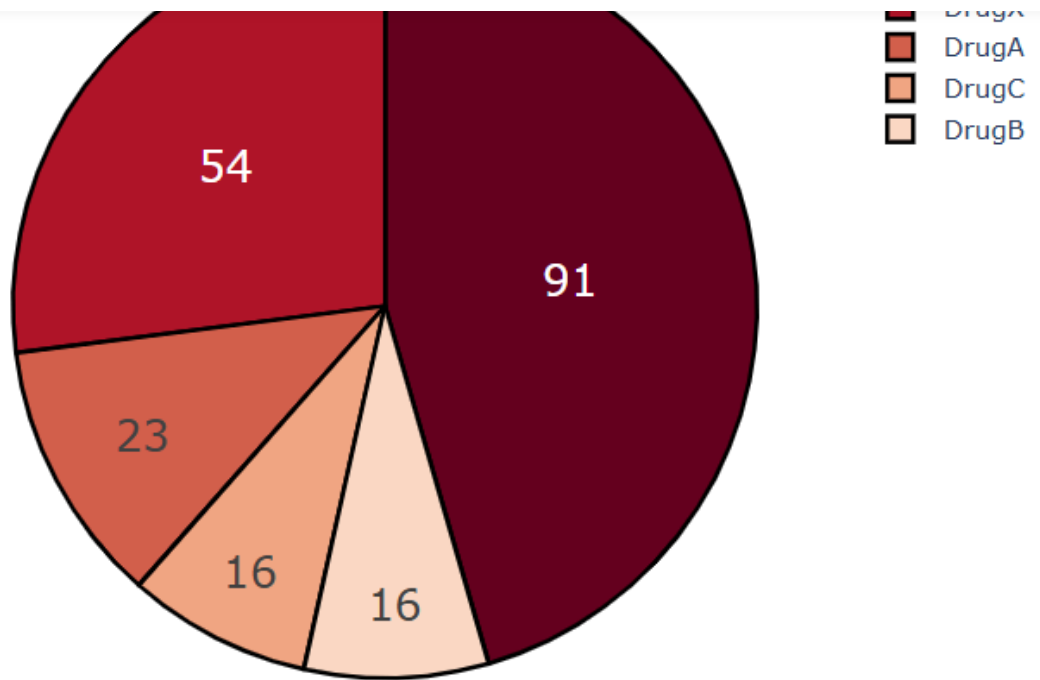
```
fig = px.histogram(x = ["HIGH", "NORMAL"], y = [103, 97])
```

```
fig.show()
```



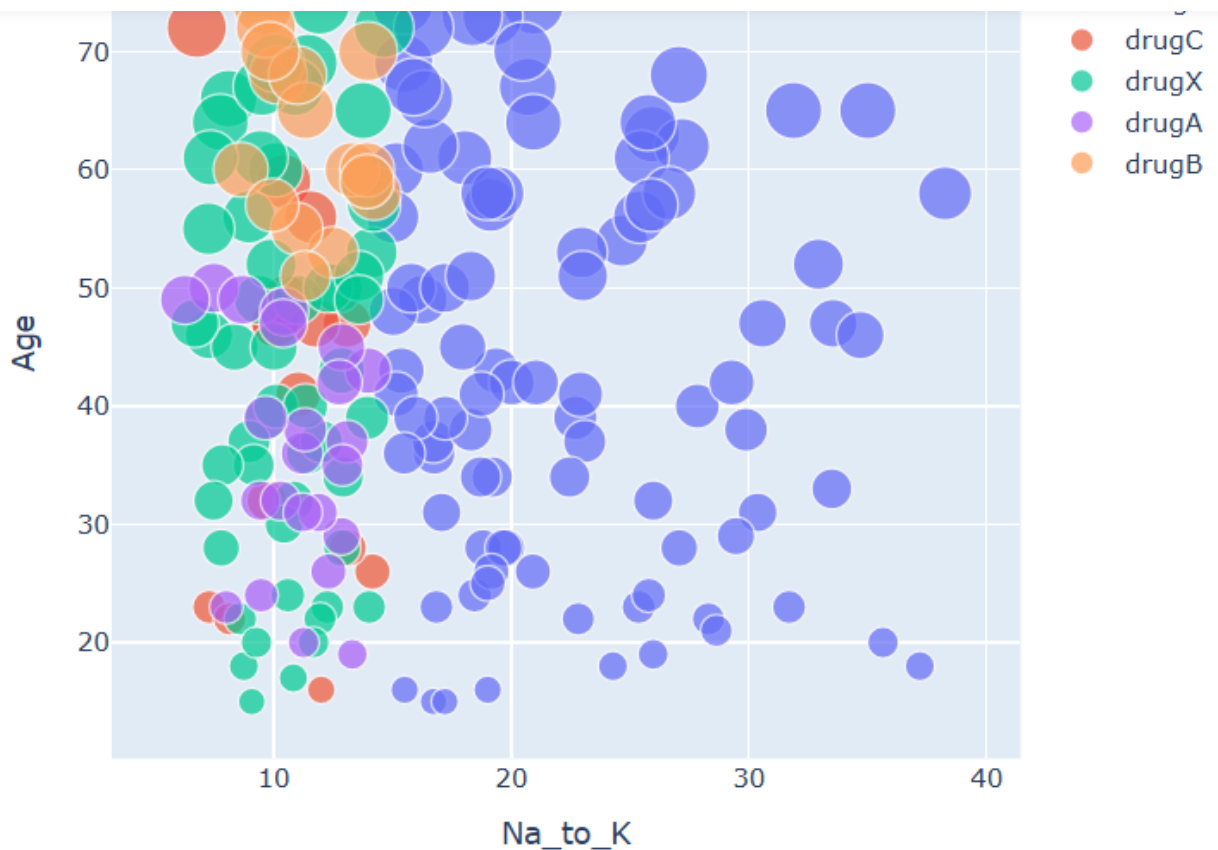
- Cholesterol 범주형 개수를 막대그래프로 시각화

```
fig = go.Figure(data = [go.Pie(labels=["DrugY","DrugX","DrugA","DrugC","DrugB"], values=[91,54
fig.update_traces(hoverinfo = 'label + percent', textinfo = 'value', textfont_size = 20,
                    marker = dict(colors = px.colors.sequential.RdBu, line = dict( color = '#0000
fig.show()
```



- Drug의 클래스 분포(각 약물 종류의 개수)를 파이 차트로 시각화

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



- Age와 Na_to_K(나트륨/칼륨 비율)의 관계를 산점도로 시각화

Classifications Models

```
dataclass = pd.read_csv(r'drug200.csv')
```

v kyubinwrld.log



	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

데이터 로드 및 파악

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
```

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

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



- Female=1, Male=0으로 바뀐 것 확인

```
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
```

- BP 컬럼 역시 문자열 값을 숫자로 변환
- `warnings.filterwarnings('ignore')` => 이 과정에서 발생할 수 있는 경고 메시지를 숨기는 설정

	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

- LOW=2, NORMAL=1, HIGH=0으로 바뀐 것 확인

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

- Cholesterol 컬럼 역시 문자열 값을 숫자로 변환

dataclass



	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

```
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
```




- Drug 컬럼 역시 문자열 값을 숫자로 변환

dataclass



	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

- DrugY=4, DrugX=3, DrugA=2, DrugC=1, DrugB=0으로 바뀐 것 확인

```

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

```

- 아직 문자열로 남아있는 자료형 있는지 다시 확인

```

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

```

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

y_data = dataclass.Drug.values

```

- BP와 Drug 컬럼을 정수형으로 형변환 후 마지막 확인
- 학습용 입력(X)과 정답(y)을 분리
 - 데이터프레임에서 타깃 컬럼 Drug만 제거하고, 나머지 컬럼(Age, Sex, BP, Cholesterol, Na_to_K)을 입력 피쳐 X로 사용
 - 타깃 컬럼 Drug를 넘파이 배열 형태의 y 레이블로 추출



	Age	Sex	BMI	Cholesterol	Na_t0_1
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

```
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])
```

• 결과 확인

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3, random_stat
```

• 데이터를 학습용(train) 과 평가용(test)으로 분리

- test_size=0.3 : 전체의 30%를 테스트 세트, 70%를 학습 세트로 분리
- x_train, y_train : 모델 학습에 쓸 입력/정답
- x_test, y_test : 최종 성능 평가에 쓸 입력/정답

```

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))

```

⇒ The accuracy of the Decision Tree is 0.9666666666666667

- DecisionTreeClassifier 학습/예측/정확도 출력
 - 테스트 샘플의 약 96.67%를 맞춤

```

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

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

```



DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=0)

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



```

from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}'.format(accuracy_s

```



Model accuracy score with criterion gini index: 0.9000



```
array([3, 3, 2, 4, 2, 3, 3, 3, 4, 2, 0, 3, 4, 3, 4, 3, 4, 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_pre
```

```
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
```

- 깊이 제한(max_depth=3) 둔 DecisionTreeClassifier(gini 기준)를 학습·평가
 - DTC_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0) => gini 불순도 사용, 트리 깊이를 3으로 제한한 모델 생성
 - DTC_gini.fit(x_train, y_train) => 학습
 - y_pred_gini = DTC_gini.predict(x_test) => 테스트셋 예측
 - accuracy_score(y_test, y_pred_gini) => 테스트 정확도 0.9000 출력
 - y_pred_train_gini = DTC_gini.predict(x_train) => 훈련셋 예측 라벨 배열
 - accuracy_score(y_train, y_pred_train_gini) 또는 DTC_gini.score(...) => 훈련 정확도 0.9143, 테스트 정확도 0.9000 재확인
- ∴ 깊이 3의 얇은 트리로 학습했더니 훈련(0.9143) 대비 테스트(0.9000)가 비슷하게 나와, 깊이 제한 없는 기본 트리보다 덜 과적합된 상태 보여줌



```
DecisionTreeClassifier
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(accuracy_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, 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
```

- 깊이 제한(max_depth=3) 둔 DecisionTreeClassifier(entropy 기준)를 학습·평가
 - DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0) => 불순도 측정에 entropy 사용, 트리 깊이 3으로 제한
 - 학습(x_train, y_train) 후, 테스트셋 예측(y_pred_en).
 - accuracy_score(y_test, y_pred_en) => 테스트 정확도 0.9000
 - accuracy_score(y_train, y_pred_train_en) 또는 .score(...) => 훈련 정확도 0.9143, 테스트 0.9000

Random Forest Classifier

```
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

```
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))
```



The accuracy of the Random Forest is 0.95

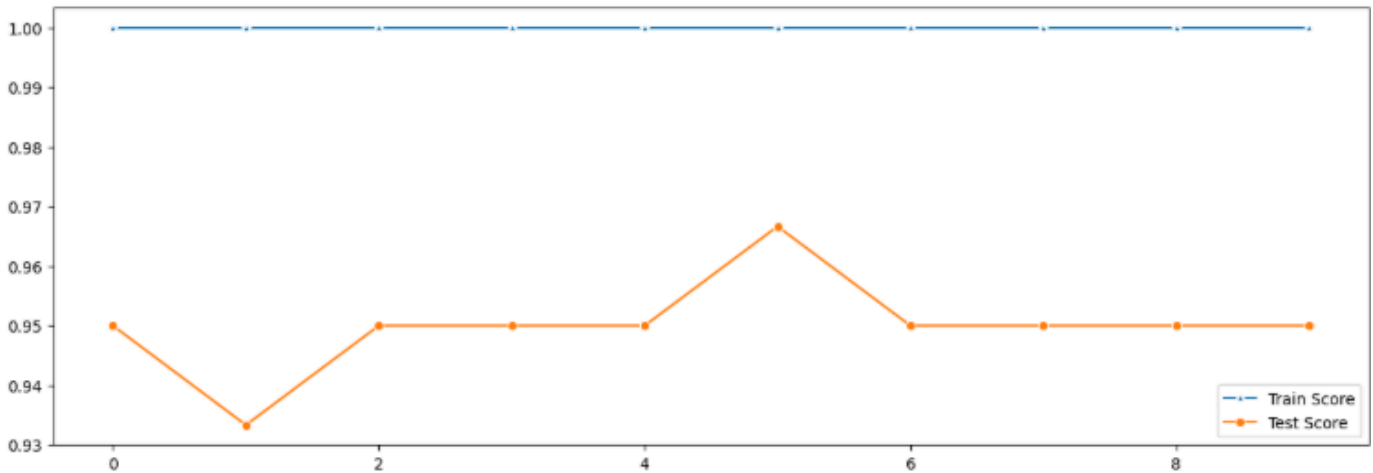
- Random Forest Classifier 학습/예측/정확도 출력
 - 기본 하이퍼파라미터로 학습했을 때 테스트 정확도 0.95
 - n_estimators=100 설정해도 정확도 0.95로 동일

```
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))
```

<Axes: >



- 시드 변화에 대한 모델 성능의 민감도와 일반화 성능 점검하는 시각화
 - random_state 를 0~9로 바꿔가며 RandomForestClassifier 학습하고 train_score_list 에는 훈련 정확도, test_score_list 에는 테스트 정확도 저장
 - sns.lineplot 활용해 시드에 따른 성능 변동(안정성), 과적합 여부(Train-Test 차이)를 선 그래프로 시각화
 - Seaborn 0.12+에서 lineplot은 x, y를 위치 인자로 못 주고, 키워드 인자로만 받기 때문에 에러 발생 => x=와 y=를 명시하도록 코드 수정

```
test_score_list = []
train_score_list = []
```

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

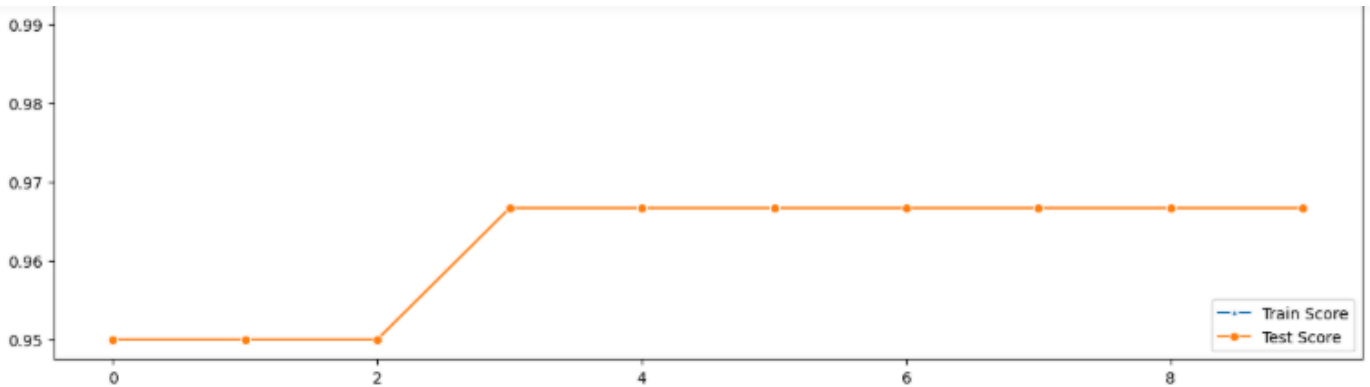
```
for i in range(0,len(list_n_estimators)):
    rfc3 = RandomForestClassifier(n_estimators=list_n_estimators[i], random_state=5)
    rfc3.fit(x_train, y_train)
    test_score_list.append(rfc3.score(x_test, y_test))
    train_score_list.append(rfc3.score(x_train, y_train))
```

```
plt.figure(figsize=(15,5))
```

```
plt.figure(figsize=(15,5))
```

```
p = sns.lineplot(x=range(0, len(list_n_estimators)), y=train_score_list, marker='*', label='Tr
```

```
p = sns.lineplot(x=range(0, len(list_n_estimators)), y=test_score_list, marker='o', label='Te
```



- 트리 개수 변화에 대한 모델 성능의 민감도와 일반화 성능 점검하는 시각화
 - n_estimators 를 10~100으로 바꿔가며 RandomForestClassifier 학습하고 train_score_list 에는 훈련 정확도, test_score_list 에는 테스트 정확도 저장
 - sns.lineplot 활용해 트리 개수 변화에 따른 성능 변동, 과적합 여부(Train-Test 차이)를 선 그래프로 시각화
 - Seaborn 0.12+에서 lineplot은 x, y를 위치 인자로 못 주고, 키워드 인자로만 받기 때문에 에러 발생 => x=와 y=를 명시하도록 코드 수정

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

last_rfc.fit(x_train,y_train)

predict = last_rfc.predict(x_test)

print('The accuracy of the 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(
```

⇒ Model accuracy score with best parameters: 0.9667



```

⇒ array([[3, 3, 2, 4, 2, 3, 3, 3, 4, 2, 0, 1, 4, 1, 4, 3, 4, 3, 4, 4, 4, 4,
          3, 4, 2, 4, 1, 3, 2, 3, 4, 2, 0, 0, 3, 3, 1, 3, 3, 4, 4, 4, 4, 3,
          1, 0, 0, 2, 4, 3, 4, 1, 4, 4, 1, 3, 4, 4, 2, 4, 4, 4, 2, 3, 4, 4,
          4, 3, 4, 1, 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, 1, 4, 4, 3, 4, 4, 4, 4, 4, 4, 1, 4,
          4, 2, 3, 1, 1, 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_pre
```

```

⇒ 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

```

- `n_estimators=100`, `random_state=5` 튜닝으로 랜덤포레스트 최종 학습·평가
 - 모델 생성/학습/테스트셋 예측 => 테스트 정확도 0.9667
 - 훈련셋 예측 라벨을 배열로 출력(중간 점검용) => 훈련 정확도 1.0000
- ∴ 선택한 하이퍼파라미터로 테스트 정확도 0.9667의 최상 성능을 얻었고, 훈련 정확도는 1.0이라 모델이 훈련 데이터에는 완벽히 적합했음을 보여줌

Evaluation Classification Models

```
cm_des = DecisionTreeClassifier()

cm_des.fit(x_train,y_train)

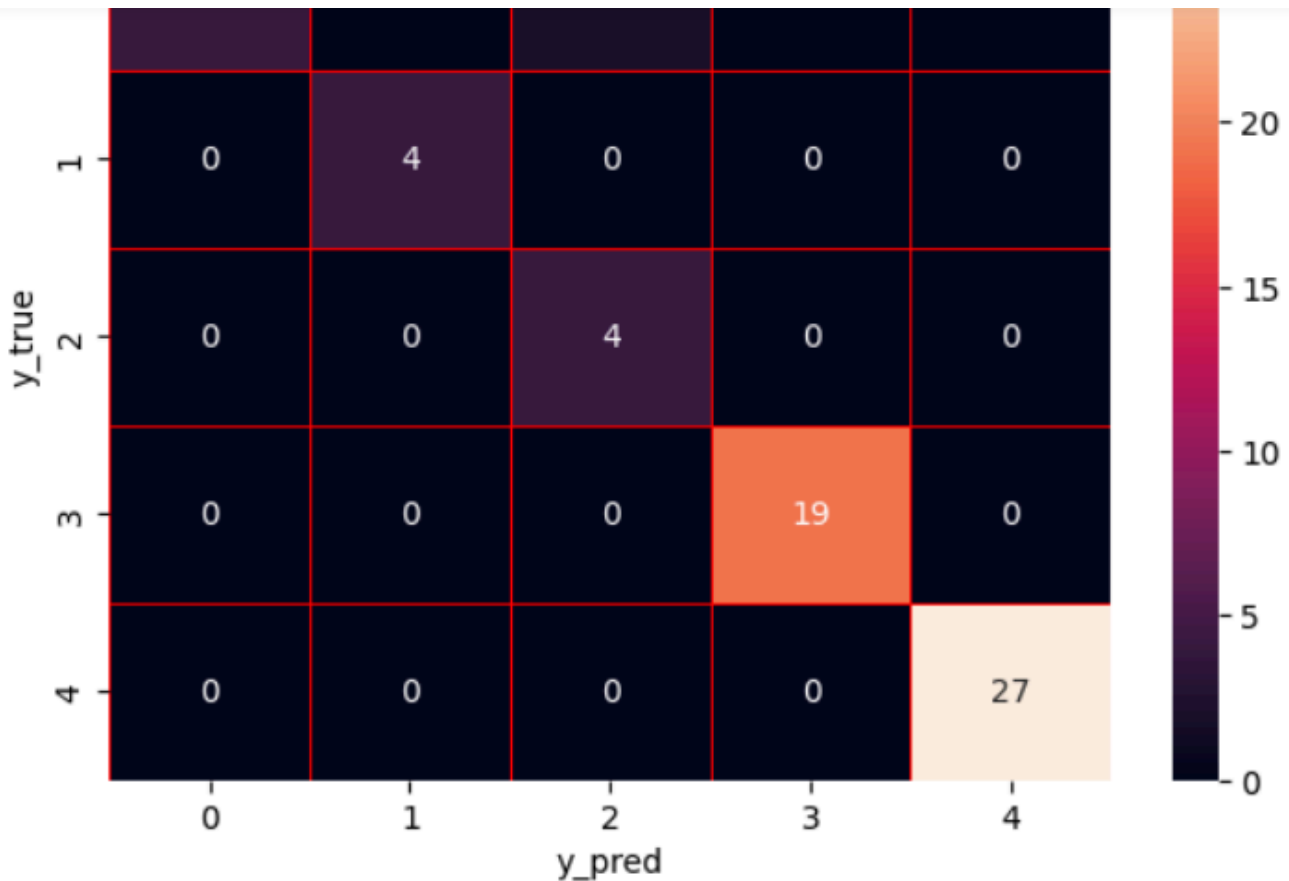
y_pred_cm = cm_des.predict(x_test)
y_true = y_test

cm_des1 = confusion_matrix( y_true, y_pred_cm)
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]])
```

- Confusion Matrix 계산/확인

```
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 Matrix를 히트맵 시각화

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

cm_des_gini.fit(x_train,y_train)

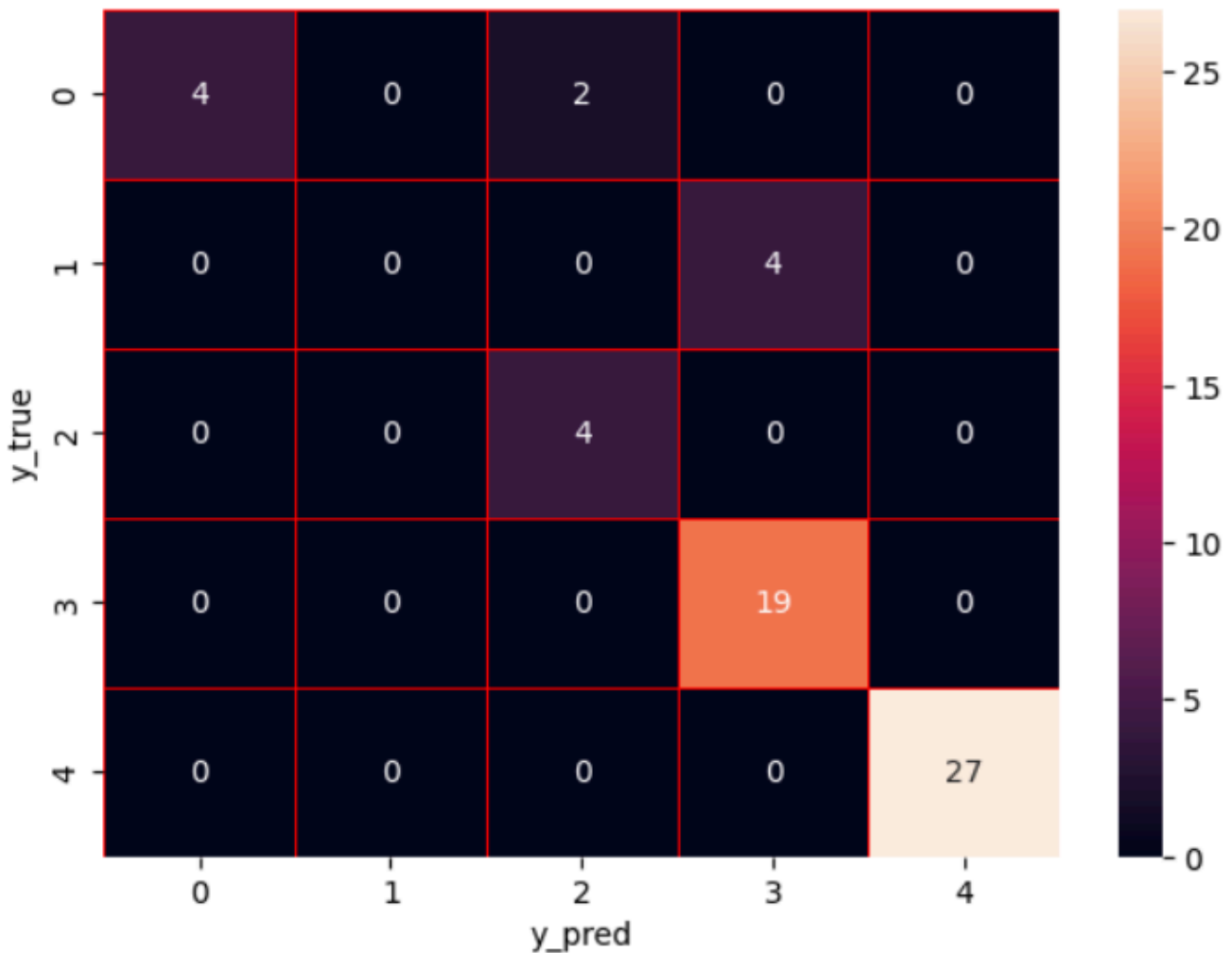
y_pred_cm = cm_des_gini.predict(x_test)
y_true = y_test

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

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

- 깊이 3인 DecisionTreeClassifier(gini 기준) 학습
- 테스트셋 예측과 실제값으로 Confusion Matrix 계산

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



- Confusion Matrix를 히트맵 시각화

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

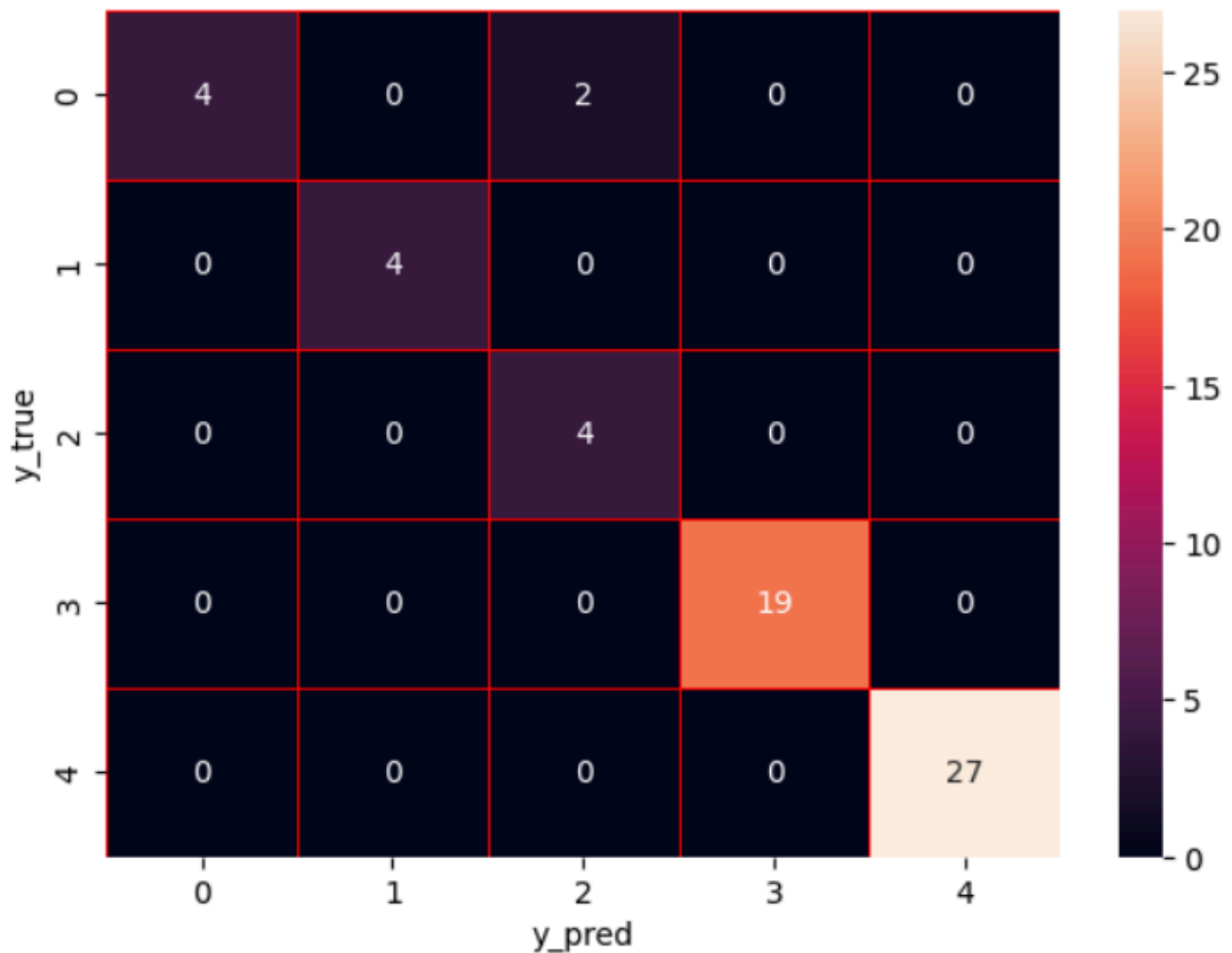
cm_last_rfc.fit(x_train, y_train)

y_pred_cm = cm_last_rfc.predict(x_test)
y_true = y_test

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

```
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_rfc, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f", ax = ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



- Confusion Matrix를 히트맵 시각화
- 아주 우수한 성능 시사

Beginner Friendly CATBOOST with OPTUNA

[링크 클릭](#)

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

from sklearn.model_selection import KFold, cross_val_score, RepeatedStratifiedKFold, StratifiedK
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PowerTransformer
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

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

import optuna
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

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

from sklearn.model_selection import KFold, cross_val_predict, train_test_split, GridSearchCV, cr
from sklearn.metrics import accuracy_score, classification_report

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

import missingno as msno
```

- 필요한 거 import 및 준비

```
# 에러 발생해서 display. 접두사 붙이는 방식으로 코드 수정
pd.set_option('display.max_columns',100)
pd.set_option('display.max_rows',900)
pd.set_option('display.max_colwidth',200)

df = pd.read_csv(r'heart.csv')
df.head()
```



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

- 데이터 로드

```
df.info()
```



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

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



```
0
```

```
def missing(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_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
MaxHR	0	0.0
ExerciseAngina	0	0.0
Oldpeak	0	0.0
ST_Slope	0	0.0
HeartDisease	0	0.0



- 결측치 요약표 생성 => 값 전부 0이면 결측치 없다는 뜻

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

⇒ Numerical Columns: Index(['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR',
Categorical Columns: Index(['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',

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

⇒

	0
Sex	2
ChestPainType	4
RestingECG	3
ExerciseAngina	2
ST_Slope	3

dtype: int64

- 수치형 컬럼과 범주형 컬럼을 자동으로 분리해서 확인
- 각 변수에 서로 다른 라벨이 몇 개 있는지 확인
 - df[categorical].nunique() => 범주형 컬럼마다 고유한 값의 개수(카디널리티) 계산

Target Variable



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

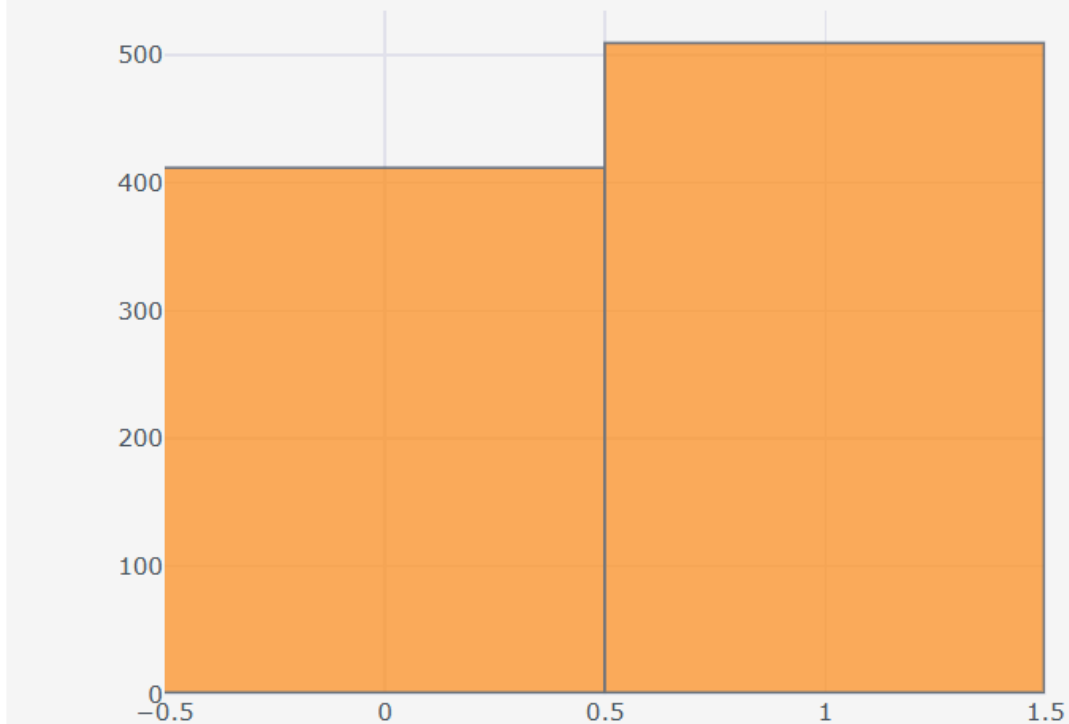
```
import cufflinks as cf
import plotly.offline as py
import plotly.io as pio

# 노트북 렌더러/오프라인 모드 초기화
py.init_notebook_mode(connected=True)
cf.go_offline()
cf.set_config_file(offline=True, world_readable=False)

# Colab이면 렌더러를 명시적으로
pio.renderers.default = 'colab'
```



- Target Variable(HeartDisease)의 클래스 분포를 퍼센트와 건수로 출력
 - HeartDisease=1(질병 있음): 55.34% (508명)
 - HeartDisease=0(정상): 44.66% (410명)
 - 거의 균형에 가까운 이진 분포 => 기본 평가지표로 Accuracy를 사용해도 크게 문제가 없음



- Cufflinks+Plotly로 Target Variable(HeartDisease) 히스토그램 시각화

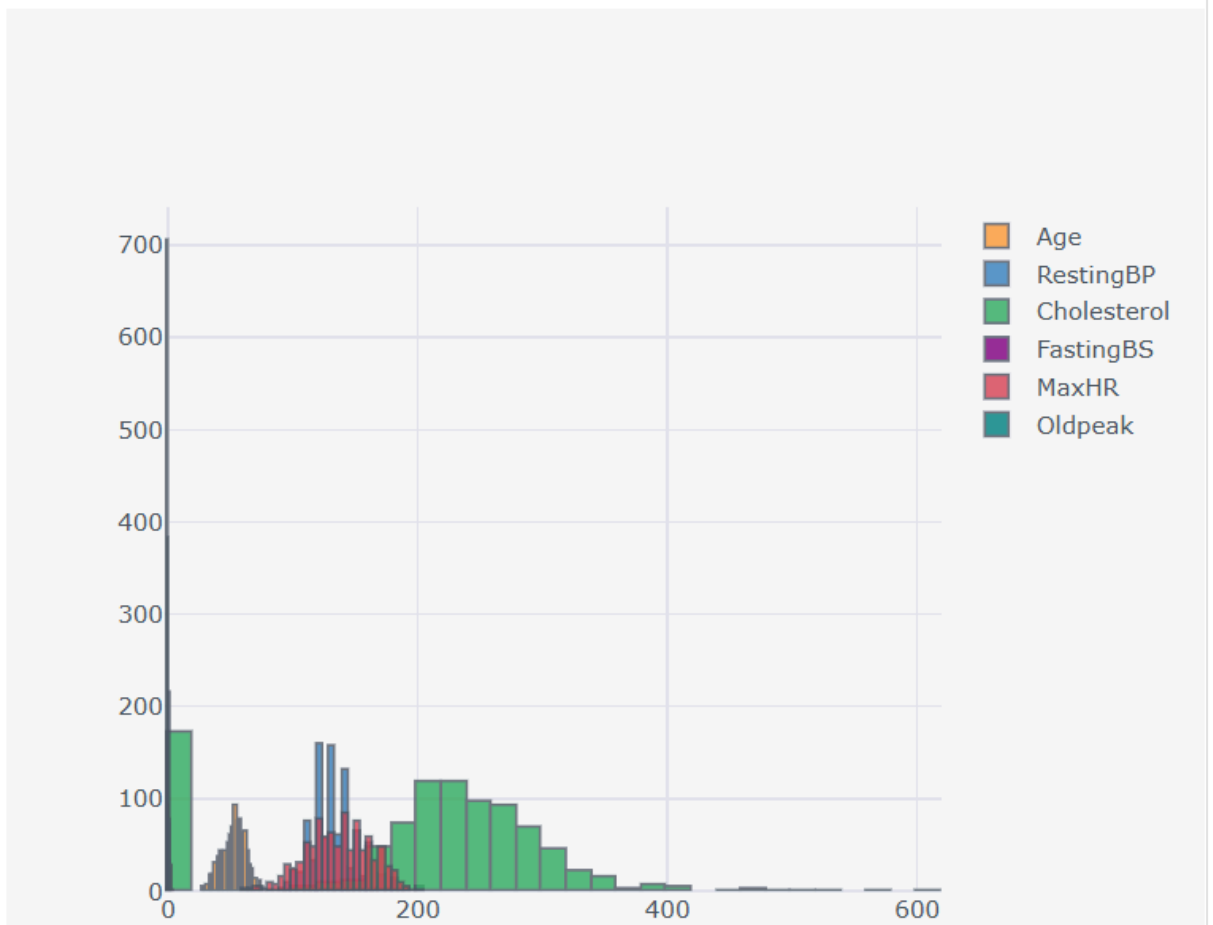
Numerical Features

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

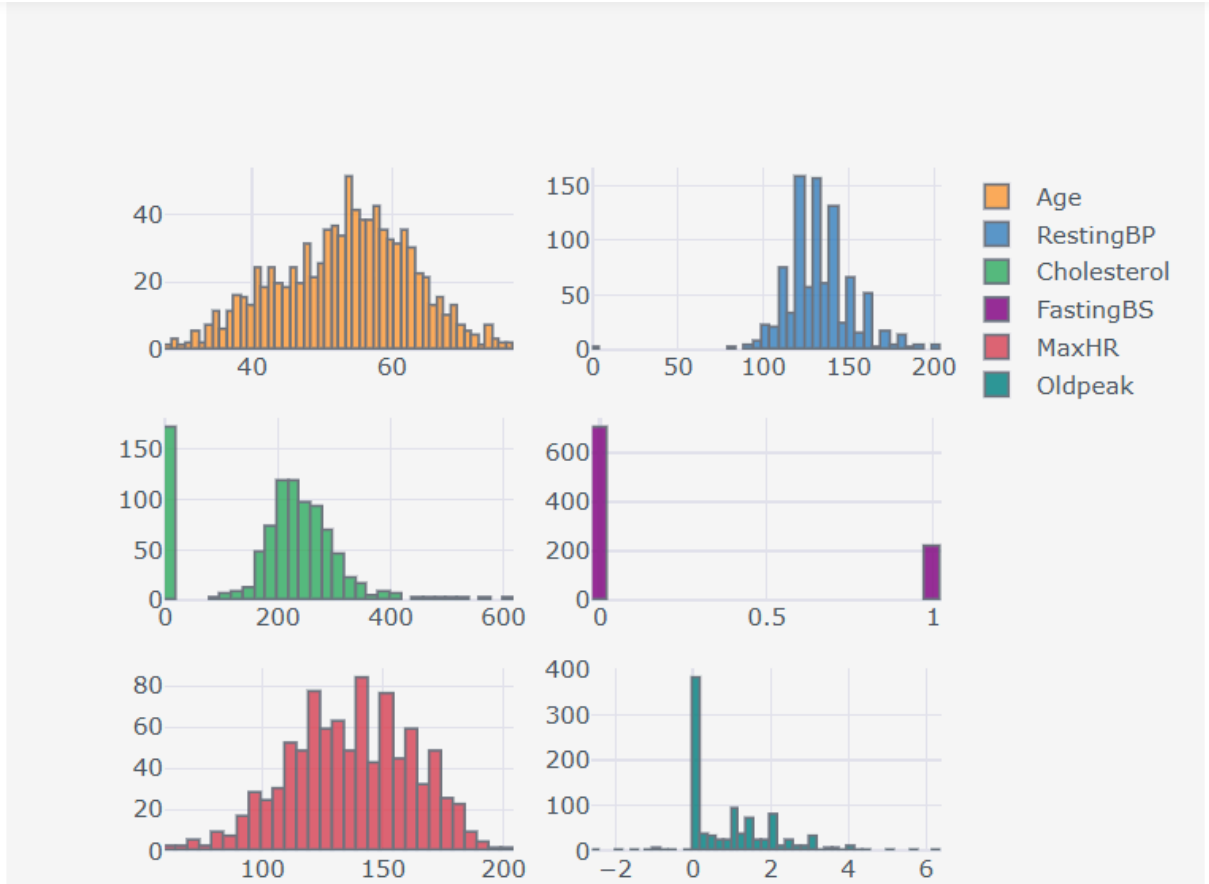


	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570
min	28.000000	0.000000	0.000000	0.000000	60.000000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

```
df[numerical].iplot(kind='hist')
```



- Cufflinks+Plotly로 수치형 변수들 히스토그램 시각화



- 수치형 컬럼 각각에 대해 50개의 bin을 가진 히스토그램을 개별 서브플롯으로 시각화

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



0

Oldpeak 1.022872

dtype: float64

수치형 변수들의 skewness 계산해, skewness가 큰 변수만 골라보기

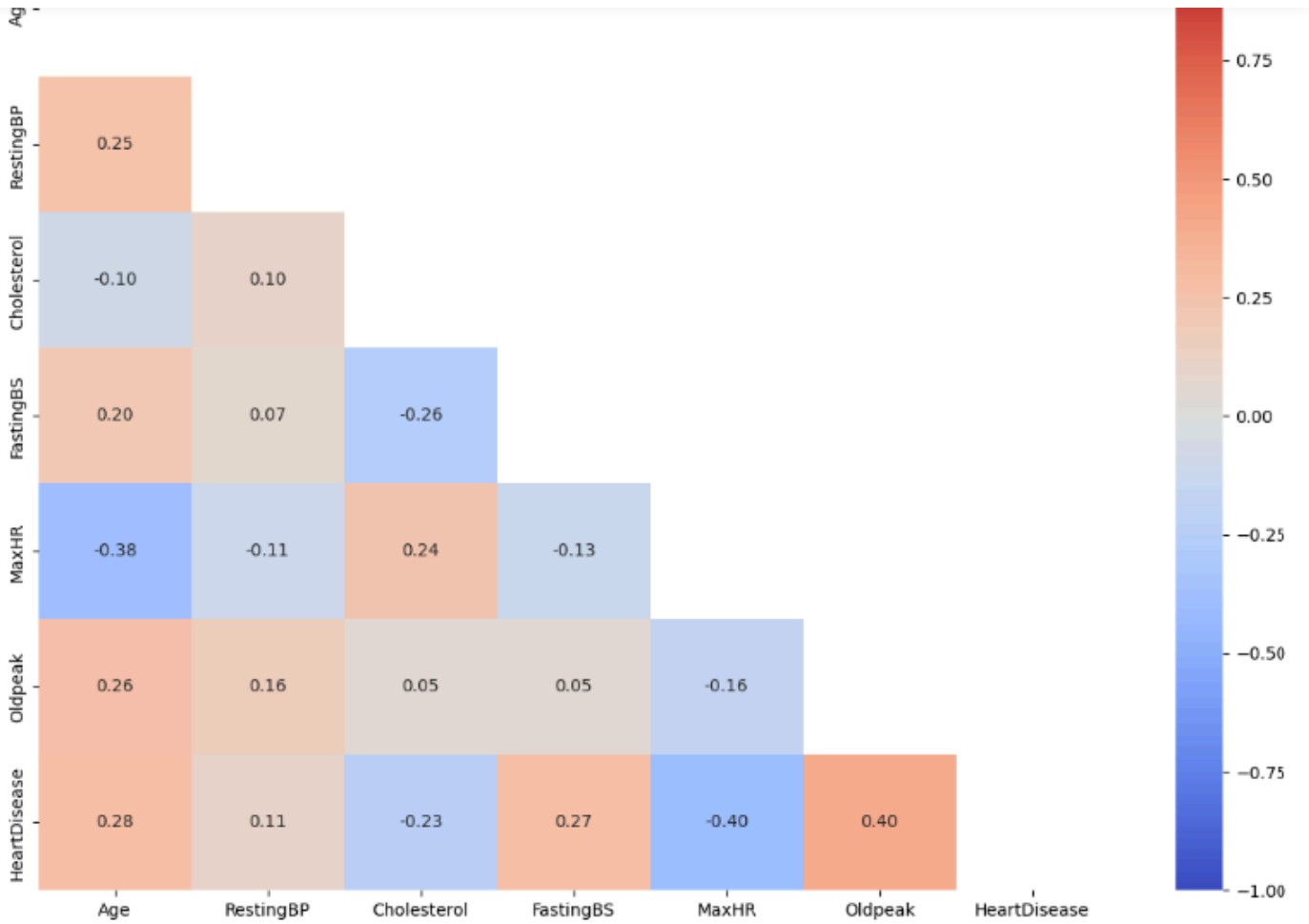
- skew_limit = 0.75 => skewness 절대값이 0.75 초과면 치우침이 크다는 기준값

```
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='c
```



- 수치형 변수들 간 상관관계(피어슨 r) 히트맵 시각화

Categorical Features

v kyubinwrld.log



	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

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

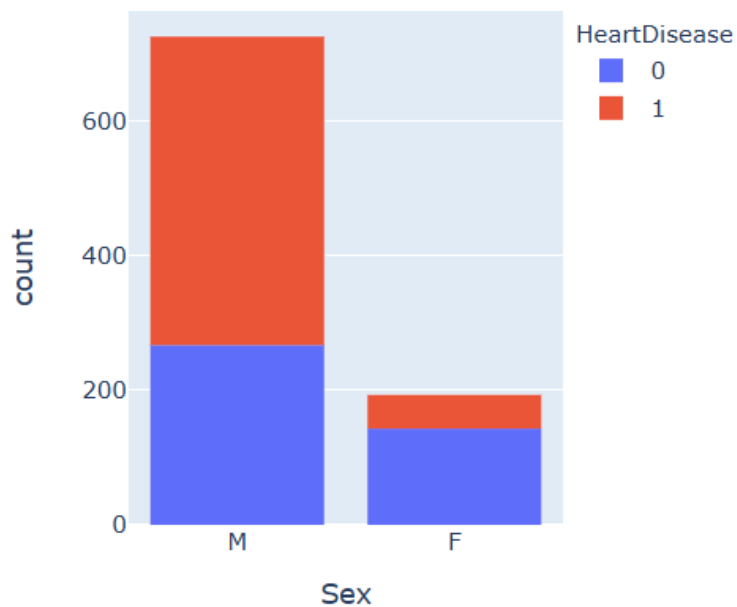
print (f'A male person has a probability of {round(df[df["Sex"]=="M"]["HeartDisease"].mean()*100, 2)} % have a HeartDisease')
print()
```



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

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

- 범주형 컬럼들만 추려서 상위 5행을 미리보기
- Sex별 Heart Disease 비율 계산해 출력
 - 여성 약 25.91%, 남성 약 63.17%가 HeartDisease=1(질환 있음)으로 나타남



- Plotly 히스토그램으로 Sex별 HeartDisease 분포 시각화

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



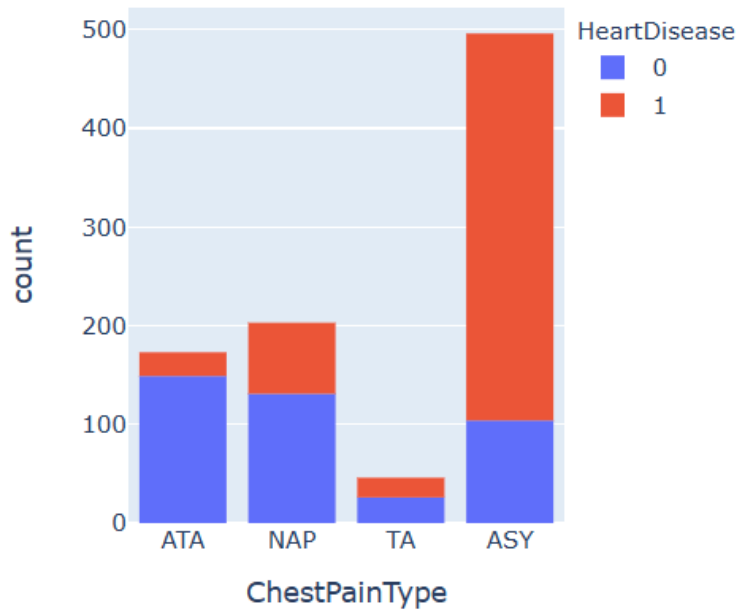
HeartDisease

ChestPainType

ASY	0.790323
TA	0.434783
NAP	0.354680
ATA	0.138728

dtype: float64

- Chest Pain Type별 Heart Disease 비율 계산해 출력



- Plotly 히스토그램으로 Chest Pain Type별 HeartDisease 분포 시각화

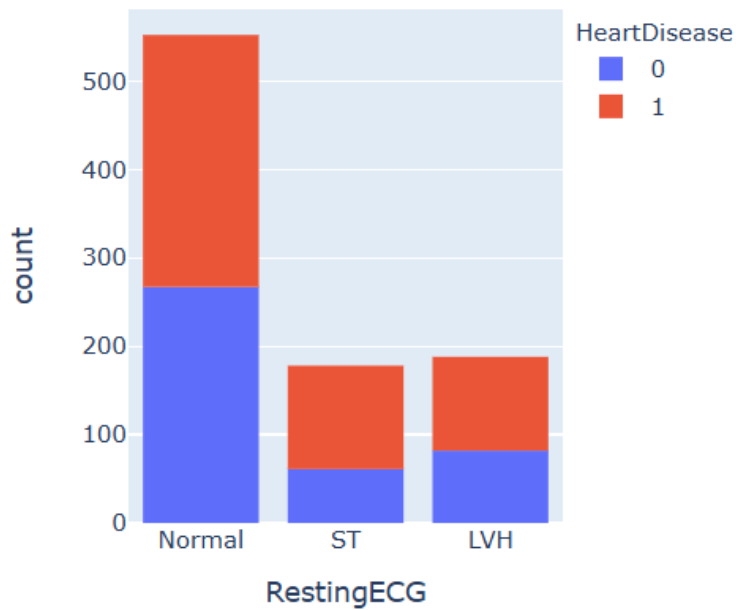
```
df.groupby('RestingECG')['HeartDisease'].mean().sort_values(ascending=False)
```



HeartDisease	
RestingECG	
ST	0.657303
LVH	0.563830
Normal	0.516304

dtype: float64

- RestingECG별 Heart Disease 비율 계산해 출력



- Plotly 히스토그램으로 RestingECG별 HeartDisease 분포 시각화

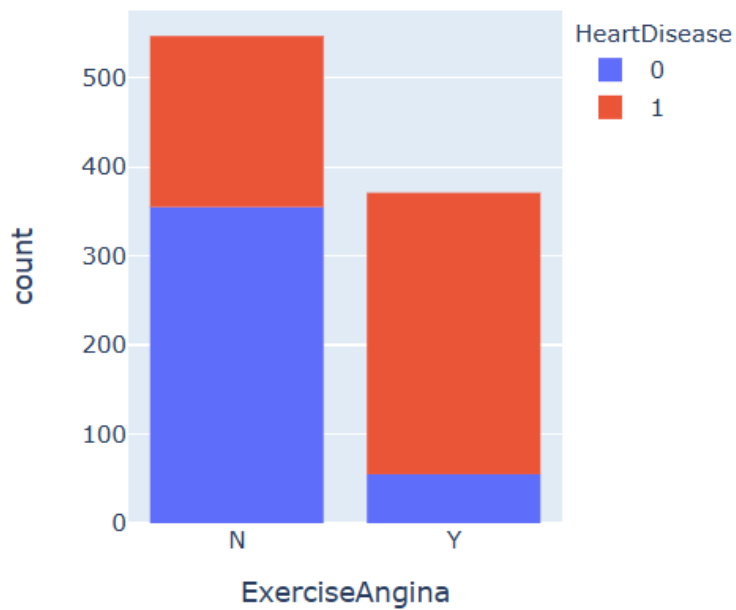
```
df.groupby('ExerciseAngina')['HeartDisease'].mean().sort_values(ascending=False)
```



HeartDisease	
ExerciseAngina	
Y	0.851752
N	0.351005

dtype: float64

- ExerciseAngina별 Heart Disease 비율 계산해 출력



- Plotly 히스토그램으로 ExerciseAngina별 HeartDisease 분포 시각화

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

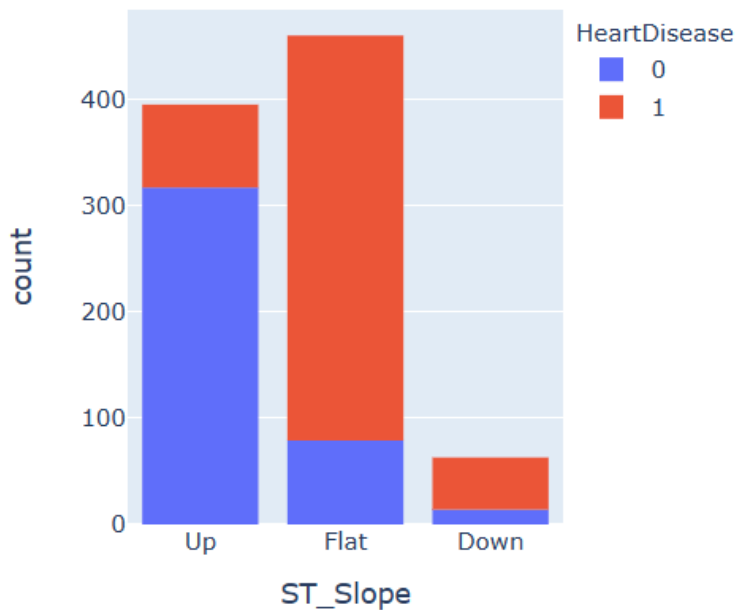


HeartDisease

ST_Slope	HeartDisease
Flat	0.828261
Down	0.777778
Up	0.197468

dtype: float64

- ST_Slope별 Heart Disease 비율 계산해 출력



- Plotly 히스토그램으로 ST_Slope별 HeartDisease 분포 시각화

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_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
```

DummyClassifier

0.5942



- Baseline Model 성능 얻기 위한 준비/학습/평가
 - 항상 1을 예측하는 아주 단순한 기준선 정확도를 만들어, 이후 모델들의 성능이 이 기준보다 의미 있게 높은지 비교하기 위한 단계

Logistic & Linear Discriminant & SVC & KNN

```

accuracy = []
model_names = []

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

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

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




models = [lr,lda,svm,knn]

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

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

```

model : KNeighborsClassifier() and accuracy score is : 0.7174

	Accuracy	
Logistic	0.8841	
LinearDiscriminant	0.8696	
SVM	0.7246	
KNeighbors	0.7174	

- 네 가지 기본 분류기(Logistic & Linear Discriminant & SVC & KNN)를 같은 전처리로 학습/평가해 정확도 비교 표 생성

Logistic & Linear Discriminant & SVC & KNN with Scaler

```
accuracy = []
model_names = []

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

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

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

models = [lr,lda,svm,knn]

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

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

v kyubinwrld.log



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

- 스케일링까지 포함한 전처리 + 4개 모델의 정확도 비교 수행

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

```

accuracy = []
model_names = []

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

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

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

models = [ada,gb,rf,et]

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

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

```

v kyubinwrld.log



model : ExtraTreesClassifier(random_state=0) and accuracy score is : 0.8804

	Accuracy	
Ada	0.8732	
Gradient	0.8768	
Random	0.8877	
ExtraTree	0.8804	

- Ensemble Models 4종(AdaBoost & Gradient Boosting & Random Forest & Extra Trees)을 같은 전처리로 학습/평가해 정확도 비교

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_state=42)

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

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

models = [xgbc,lgbmc]

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

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

v kyubinwrld.log



```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Accuracy



XGBoost 0.8478



LightGBM 0.8732



- Famous Trio (XGBoost & LightGBM & Catboost)를 같은 전처리로 학습/평가해 정확도 비교

CATBOOST

```
accuracy = []
model_names = []

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
# 오류 발생으로 범주형 컬럼만 뽑는 방식으로 코드 수정
cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()

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

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

model.fit(X_train, y_train, cat_features=cat_cols, 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
```



Accuracy



Catboost_default 0.8913



- CatBoost를 전처리 없이 바로 학습/평가해 정확도를 얻기

```

X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()

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

param = {
    "objective": trial.suggest_categorical("objective", ["Logloss", "CrossEntropy"]),
    "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.1),
    "depth": trial.suggest_int("depth", 1, 12),
    "boosting_type": trial.suggest_categorical("boosting_type", ["Ordered", "Plain"]),
    "bootstrap_type": trial.suggest_categorical(
        "bootstrap_type", ["Bayesian", "Bernoulli", "MVS"]
    ),
    "used_ram_limit": "3gb",
}

if param["bootstrap_type"] == "Bayesian":
    param["bagging_temperature"] = trial.suggest_float("bagging_temperature", 0, 10)
elif param["bootstrap_type"] == "Bernoulli":
    param["subsample"] = trial.suggest_float("subsample", 0.1, 1)

cat_cls = CatBoostClassifier(**param)

cat_cls.fit(X_train, y_train, eval_set=[(X_test, y_test)], cat_features=cat_cols, verbose=0)

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 2025-10-05 15:55:19,741] Trial 48 finished with value: 0.894927536231884 and pa
[I 2025-10-05 15:55:29,687] Trial 49 finished with value: 0.894927536231884 and pa
Number of finished trials: 50
Best trial:
  Value: 0.9094202898550725
  Params:
    objective: CrossEntropy
    colsample_bylevel: 0.0858922491618681
    depth: 12
    boosting_type: Plain
    bootstrap_type: Bernoulli
    subsample: 0.9614400584870564
```

- Optuna로 Catboost 하이퍼 파라미터 튜닝
 - Optuna가 정한 하이퍼파라미터 조합으로 CatBoost를 학습/검증하고, 정확도를 점수로 돌려주며, 이를 반복해 가장 높은 정확도를 주는 설정을 찾음

<파라미터>

- Objective: 과적합 감지 및 최적 모델 선택을 위해 지원되는 평가 지표
- colsample_bylevel: 훈련 속도를 높여주며, 보통 모델의 품질에는 영향을 미치지 않음
depth : 트리의 깊이
- boosting_type : 기본적으로 부스팅 유형은 작은 데이터셋에 맞게 설정되어 있으며, 이는 과적합을 방지하지만 계산 비용이 많이 듭니다
- bootstrap_type : 기본적으로 객체의 가중치를 샘플링하는 방법 설정이며, 배깅을 위한 샘플 비율 값이 1보다 작으면 훈련이 더 빠르게 수행됨

```
accuracy =[]
model_names =[]
```

```
X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()

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

model = CatBoostClassifier(verbose=False,random_state=0,
                           objective= 'CrossEntropy',
                           colsample_bylevel= 0.04292240490294766,
                           depth= 10,
```

```

model.fit(X_train, y_train, cat_features=cat_cols, 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	0.85	0.88	0.87	112
1	0.92	0.90	0.91	164
accuracy			0.89	276
macro avg	0.89	0.89	0.89	276
weighted avg	0.89	0.89	0.89	276

Accuracy



Catboost_tuned 0.8913



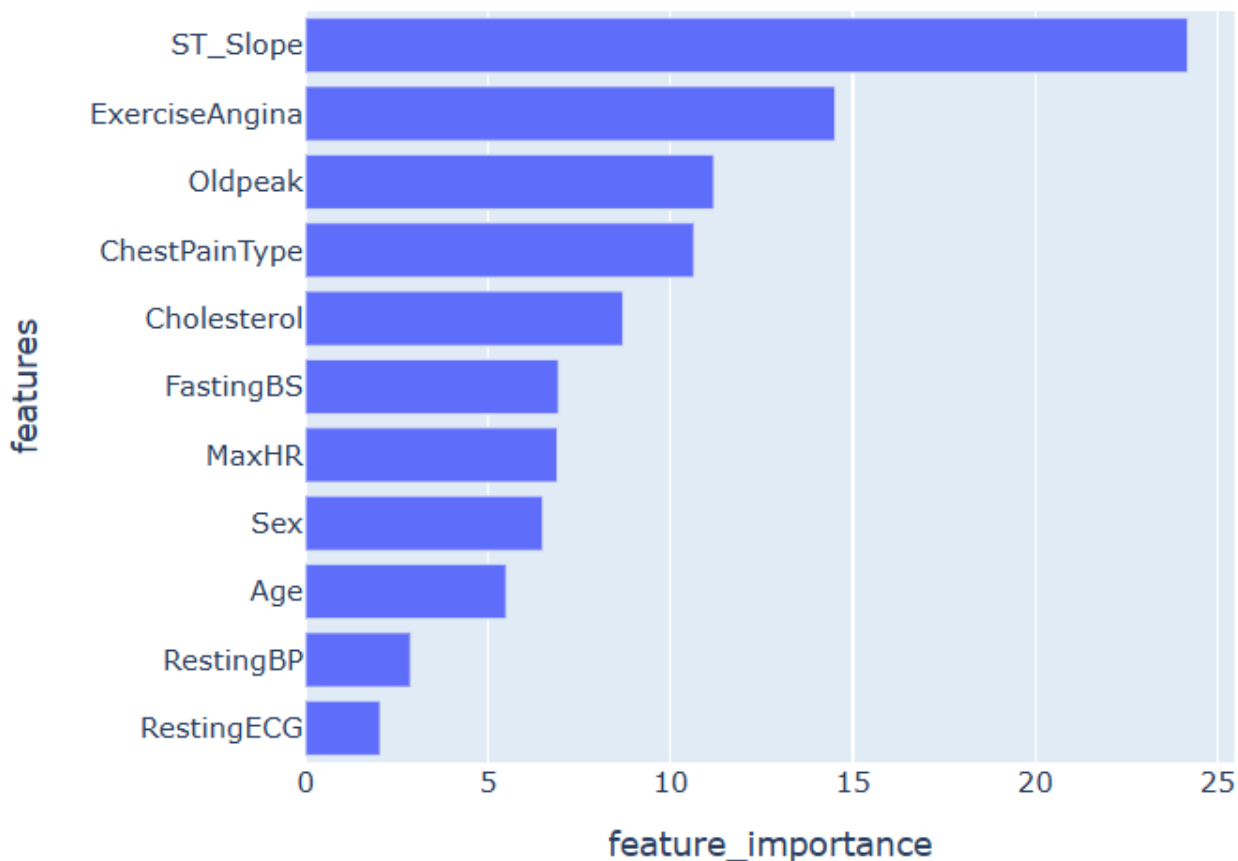
- CatBoostClassifier 모델 사용해 Heart Disease 예측하는 분류 모델을 학습/평가하고
- 하이퍼파라미터 튜닝을 통해 얻은 특정 설정 값들을 CatBoost 모델에 적용하여 학습
진행하고 성능 확인

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",h
fig.show()

```



- CatBoost 분류 모델이 Heart Disease 예측을 위해 어떤 특징(Feature)들을 가장 중요하게 사용했는지 중요도(Feature Importance) 계산하고 시각화

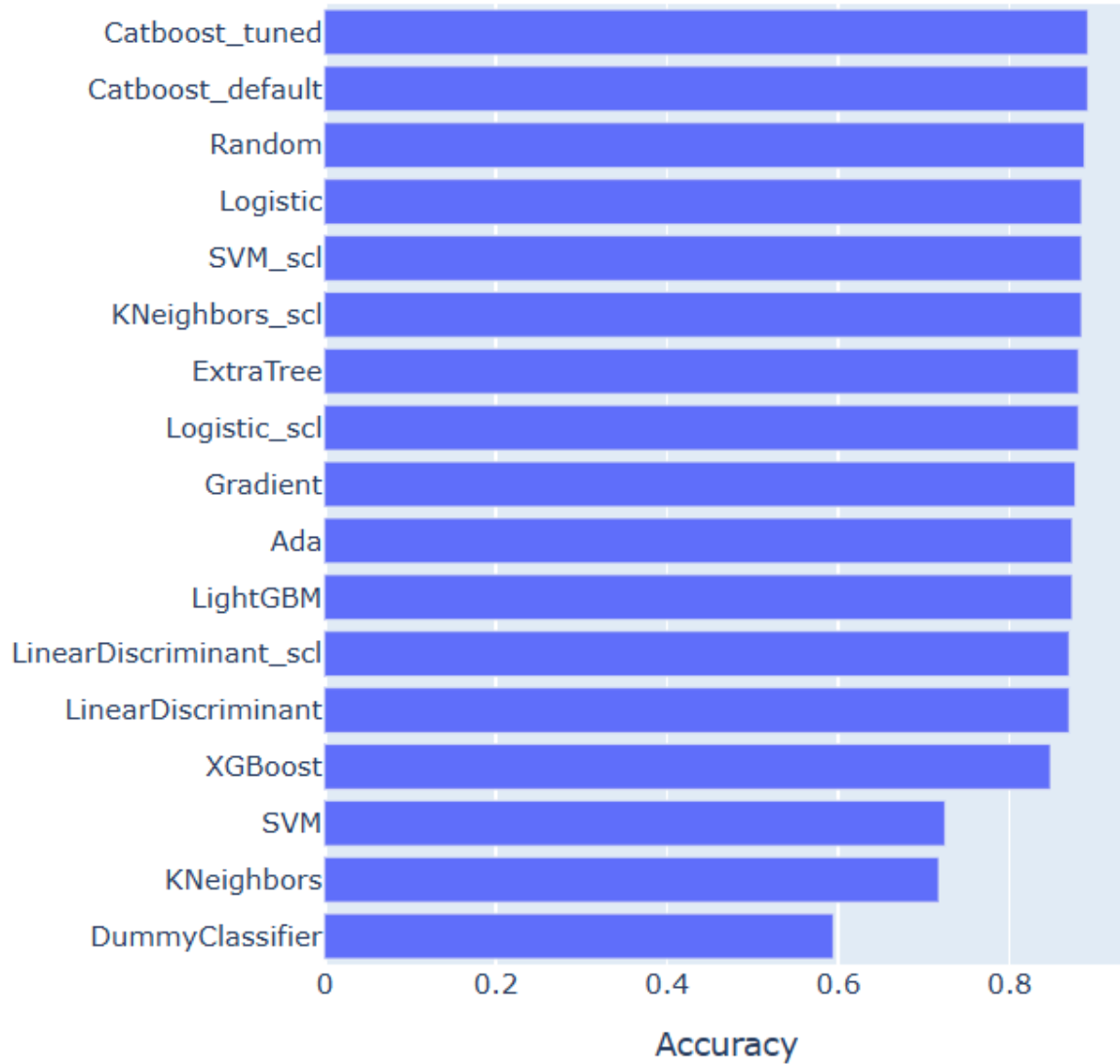
Model Comparison

```
result_final = pd.concat([dummy_result_df,result_df1,result_df2,result_df3,result_df4,result_d
```

```
result_final.sort_values(by=['Accuracy'], ascending=True,inplace=True)
fig = px.bar(result_final, x='Accuracy', y=result_final.index,title='Model Comparison',height=
fig.show()
```



MODELS



- 이전에 학습하고 평가했던 여러 모델의 성능 결과를 하나로 통합
- 가장 성능이 좋은 모델을 한눈에 비교할 수 있도록 시각화



진규빈



이전 포스트

[파머완] 04 분류