```
import numpy as np
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

#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

import missingno as msno
import missingno as msno

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

```
pd.set_option('max_columns',100)
pd.set_option('max_rows',900)

pd.set_option('max_colwidth',200)

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

```
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_Numbe
    return missing_values

missing(df)
```

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

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

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

```
Numerical Columns:  Index(['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak'


Categorical Columns: Index(['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope
```

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

```
Sex               2
ChestPainType     4
RestingECG        3
ExerciseAngina    2
ST_Slope          3
dtype: int64
```

```
y = df['HeartDisease']
print(f'Percentage of patient had a HeartDisease:  {round(y.value_counts(normalize=True)[1]*10
Percentage of patient had a HeartDisease:  55.34 %  --> (508 patient)
Percentage of patient did not have a HeartDisease: 44.66  %  --> (410 patient)
```

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

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

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

```
df[numerical].iplot(kind='histogram',subplots=True,bins=50)
```

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

```
Oldpeak    1.022872
dtype: float64
```

```
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
```

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

Gender and Heart Disease

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

print()

print (f'A male person has a probability of {round(df[df["Sex"]=="M"]["HeartDisease"].mean()*1

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()
```

Chest Pain Type and Heart Disease

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

```
ChestPainType
ASY    0.790323
TA     0.434783
NAP    0.354680
ATA    0.138728
Name: HeartDisease, dtype: float64
```

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

RestingECG and Heart Disease

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

```
RestingECG
ST        0.657303
LVH       0.563830
Normal    0.516304
Name: HeartDisease, dtype: float64
```

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

ExerciseAngina and Heart Disease

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

```
ExerciseAngina
Y    0.851752
N    0.351005
Name: HeartDisease, dtype: float64
```

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

ST_Slope and Heart Disease

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

```
ST_Slope
Flat    0.828261
Down    0.777778
Up      0.197468
Name: HeartDisease, dtype: float64
```

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

Overall Insights from the Exploratory Data Analysis

```
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
```

```
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
```

```
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
```

```
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
```

```
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
```

```
accuracy =[]
model_names =[]


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

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

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

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

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


```

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

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

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

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

    cat_cls = CatBoostClassifier(**param)

    cat_cls.fit(X_train, y_train, eval_set=[(X_test, y_test)], cat_features=categorical_featur

    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 2022-04-04 11:09:46,497] Trial 3 finished with value: 0.8659420289855072 and parameter
[I 2022-04-04 11:09:49,333] Trial 4 finished with value: 0.8804347826086957 and parameter
[I 2022-04-04 11:09:55,124] Trial 5 finished with value: 0.8913043478260869 and parameter
[I 2022-04-04 11:09:56,384] Trial 6 finished with value: 0.8876811594202898 and parameter
[I 2022-04-04 11:09:59,400] Trial 7 finished with value: 0.8876811594202898 and parameter
[I 2022-04-04 11:10:01,216] Trial 8 finished with value: 0.8876811594202898 and parameter
[I 2022-04-04 11:10:02,627] Trial 9 finished with value: 0.8913043478260869 and parameter
[I 2022-04-04 11:10:06,275] Trial 10 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:07,608] Trial 11 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:08,988] Trial 12 finished with value: 0.8840579710144928 and paramete
[I 2022-04-04 11:10:10,432] Trial 13 finished with value: 0.8876811594202898 and paramete
[I 2022-04-04 11:10:11,768] Trial 14 finished with value: 0.8876811594202898 and paramete

[I 2022-04-04 11:10:13,133] Trial 15 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:14,787] Trial 16 finished with value: 0.8876811594202898 and paramete
[I 2022-04-04 11:10:16,118] Trial 17 finished with value: 0.8840579710144928 and paramete
[I 2022-04-04 11:10:17,620] Trial 18 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:20,256] Trial 19 finished with value: 0.8768115942028986 and paramete
```

```
[I 2022-04-04 11:10:21,998] Trial 20 finished with value: 0.8804347826086957 and paramete
[I 2022-04-04 11:10:23,460] Trial 21 finished with value: 0.8840579710144928 and paramete
[I 2022-04-04 11:10:24,807] Trial 22 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:26,146] Trial 23 finished with value: 0.8876811594202898 and paramete
[I 2022-04-04 11:10:27,444] Trial 24 finished with value: 0.9021739130434783 and paramete
[I 2022-04-04 11:10:28,674] Trial 25 finished with value: 0.8876811594202898 and paramete
[I 2022-04-04 11:10:29,414] Trial 26 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:34,060] Trial 27 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:35,374] Trial 28 finished with value: 0.8876811594202898 and paramete
[I 2022-04-04 11:10:36,093] Trial 29 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:36,786] Trial 30 finished with value: 0.8840579710144928 and paramete
[I 2022-04-04 11:10:37,530] Trial 31 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:38,303] Trial 32 finished with value: 0.8985507246376812 and paramete
[I 2022-04-04 11:10:39,083] Trial 33 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:39,760] Trial 34 finished with value: 0.8768115942028986 and paramete
[I 2022-04-04 11:10:41,120] Trial 35 finished with value: 0.8804347826086957 and paramete
[I 2022-04-04 11:10:41,925] Trial 36 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:43,250] Trial 37 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:44,048] Trial 38 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:44,767] Trial 39 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:45,545] Trial 40 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:46,383] Trial 41 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:47,223] Trial 42 finished with value: 0.8985507246376812 and paramete
[I 2022-04-04 11:10:48,170] Trial 43 finished with value: 0.8876811594202898 and paramete
[I 2022-04-04 11:10:48,920] Trial 44 finished with value: 0.8804347826086957 and paramete
[I 2022-04-04 11:10:49,806] Trial 45 finished with value: 0.8913043478260869 and paramete
[I 2022-04-04 11:10:50,563] Trial 46 finished with value: 0.8985507246376812 and paramete
[I 2022-04-04 11:10:51,341] Trial 47 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:52,220] Trial 48 finished with value: 0.894927536231884 and parameter
[I 2022-04-04 11:10:53,107] Trial 49 finished with value: 0.8876811594202898 and paramete
Number of finished trials: 50
Best trial:
  Value: 0.9021739130434783
  Params:
    objective: Logloss
    colsample_bylevel: 0.011349438888867012
    depth: 9
    boosting_type: Ordered
    bootstrap_type: MVS
```

```
accuracy =[]
model_names =[]


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

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

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

```

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

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


```

```
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()
```

```
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()
```

✓ 0초 오후 8:10에 완료됨 ● ✕