import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score,roc\_auc\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.preprocessing import StandardScaler  
import warnings  
warnings.simplefilter(action="ignore")  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import GridSearchCV, cross\_validate  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from catboost import CatBoostClassifier  
from lightgbm import LGBMClassifier  
from xgboost import XGBClassifier  
  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.width', None)  
pd.set\_option('display.max\_rows', 20)  
pd.set\_option('display.float\_format', lambda x: '%.3f' % x)  
  
df= pd.read\_csv(r'C:\Users\PC\PycharmProjects\pythonProject\heart.csv')  
df.head()  
  
  
def check\_df(dataframe, head=5):  
 print("##################### Shape #####################")  
 print(dataframe.shape)  
 print("##################### Types #####################")  
 print(dataframe.dtypes)  
 print("##################### Head #####################")  
 print(dataframe.head(head))  
 print("##################### Tail #####################")  
 print(dataframe.tail(head))  
 print("##################### NA #####################")  
 print(dataframe.isnull().sum())  
  
  
check\_df(df, head=2)  
  
*# NUMERİK VE KATEGORİK DEĞİŞKENLERİN YAKALANMASI*def grab\_col\_names(dataframe, cat\_th=10, car\_th=20):  
 *"""  
  
 Veri setindeki kategorik, numerik ve kategorik fakat kardinal değişkenlerin isimlerini verir.  
 Not: Kategorik değişkenlerin içerisine numerik görünümlü kategorik değişkenler de dahildir.  
  
 Parameters  
 ------  
 dataframe: dataframe  
 Değişken isimleri alınmak istenilen dataframe  
 cat\_th: int, optional  
 numerik fakat kategorik olan değişkenler için sınıf eşik değeri  
 car\_th: int, optional  
 kategorik fakat kardinal değişkenler için sınıf eşik değeri  
  
 Returns  
 ------  
 cat\_cols: list  
 Kategorik değişken listesi  
 num\_cols: list  
 Numerik değişken listesi  
 cat\_but\_car: list  
 Kategorik görünümlü kardinal değişken listesi  
  
 Examples  
 ------  
 import seaborn as sns  
 df = sns.load\_dataset("iris")  
 print(grab\_col\_names(df))  
  
  
 Notes  
 ------  
 cat\_cols + num\_cols + cat\_but\_car = toplam değişken sayısı  
 num\_but\_cat cat\_cols'un içerisinde.  
  
 """  
 # cat\_cols, cat\_but\_car* cat\_cols = [col for col in dataframe.columns if dataframe[col].dtypes == "O"]  
 num\_but\_cat = [col for col in dataframe.columns if dataframe[col].nunique() < cat\_th and dataframe[col].dtypes != "O"]  
 cat\_but\_car = [col for col in dataframe.columns if dataframe[col].nunique() > car\_th and dataframe[col].dtypes == "O"]  
 cat\_cols = cat\_cols + num\_but\_cat  
 cat\_cols = [col for col in cat\_cols if col not in cat\_but\_car]  
  
 *# num\_cols* num\_cols = [col for col in dataframe.columns if dataframe[col].dtypes != "O"]  
 num\_cols = [col for col in num\_cols if col not in num\_but\_cat]  
  
 print(f"Observations: {dataframe.shape[0]}")  
 print(f"Variables: {dataframe.shape[1]}")  
 print(f'cat\_cols: {len(cat\_cols)}')  
 print(f'num\_cols: {len(num\_cols)}')  
 print(f'cat\_but\_car: {len(cat\_but\_car)}')  
 print(f'num\_but\_cat: {len(num\_but\_cat)}')  
  
 return cat\_cols, num\_cols, cat\_but\_car  
cat\_cols, num\_cols, cat\_but\_car = grab\_col\_names(df)  
  
cat\_cols  
num\_cols  
cat\_but\_car  
  
def cat\_summary(dataframe, col\_name, plot=True):  
 print(pd.DataFrame({col\_name: dataframe[col\_name].value\_counts(),  
 "Ratio": 100 \* dataframe[col\_name].value\_counts() / len(dataframe)}))  
 print("##########################################")  
 if plot:  
 sns.countplot(x=dataframe[col\_name], data=dataframe)  
 plt.show(block=True)  
  
for col in cat\_cols:  
 cat\_summary(df, col)  
  
def num\_summary(dataframe, numerical\_col, plot=False):  
 quantiles = [0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.95, 0.99]  
 print(dataframe[numerical\_col].describe(quantiles).T)  
  
 if plot:  
 dataframe[numerical\_col].hist(bins=20)  
 plt.xlabel(numerical\_col)  
 plt.title(numerical\_col)  
 plt.show(block=True)  
  
for col in num\_cols:  
 num\_summary(df, col, plot=True)  
  
def target\_summary\_with\_num(dataframe, target, numerical\_col):  
 print(dataframe.groupby(target).agg({numerical\_col: "mean"}), end="\n\n\n")  
  
for col in num\_cols:  
 target\_summary\_with\_num(df, "HeartDisease", col)  
  
  
for col in num\_cols:  
 num\_summary(df, col, plot=True)  
  
zero\_columns= [col for col in df.columns if (df[col].min() == 0 and col not in ['Oldpeak','FastingBS','HeartDisease'])]  
  
for col in zero\_columns :  
 df[col] = df[col].replace(0, np.nan)  
  
df.isnull().sum()  
  
def missing\_values\_table(dataframe, na\_name=False):  
 na\_columns = [col for col in dataframe.columns if dataframe[col].isnull().sum() > 0]  
 n\_miss = dataframe[na\_columns].isnull().sum().sort\_values(ascending=False)  
 ratio = (dataframe[na\_columns].isnull().sum() / dataframe.shape[0] \* 100).sort\_values(ascending=False)  
 missing\_df = pd.concat([n\_miss, np.round(ratio, 2)], axis=1, keys=['n\_miss', 'ratio'])  
 print(missing\_df, end="\n")  
 if na\_name:  
 return na\_columns  
  
na\_columns = missing\_values\_table(df, na\_name=True)  
  
for col in zero\_columns:  
 df.loc[df[col].isnull(), col] = df[col].median()  
  
df.isnull().sum()  
  
def outlier\_thresholds(dataframe, col\_name, q1=0.05, q3=0.95):  
 quartile1 = dataframe[col\_name].quantile(q1)  
 quartile3 = dataframe[col\_name].quantile(q3)  
 interquantile\_range = quartile3 - quartile1  
 up\_limit = quartile3 + 1.5 \* interquantile\_range  
 low\_limit = quartile1 - 1.5 \* interquantile\_range  
 return low\_limit, up\_limit  
  
def check\_outlier(dataframe, col\_name):  
 low\_limit, up\_limit = outlier\_thresholds(dataframe, col\_name)  
 if dataframe[(dataframe[col\_name] > up\_limit) | (dataframe[col\_name] < low\_limit)].any(axis=None):  
 return True  
 else:  
 return False  
  
def replace\_with\_thresholds(dataframe, variable, q1=0.05, q3=0.95):  
 low\_limit, up\_limit = outlier\_thresholds(dataframe, variable, q1=0.05, q3=0.95)  
 dataframe.loc[(dataframe[variable] < low\_limit), variable] = low\_limit  
 dataframe.loc[(dataframe[variable] > up\_limit), variable] = up\_limit  
  
for col in num\_cols:  
 print(col, check\_outlier(df, col))  
 if check\_outlier(df, col):  
 replace\_with\_thresholds(df, col)  
  
for col in num\_cols:  
 print(col, check\_outlier(df, col))  
  
cat\_cols, num\_cols, cat\_but\_car = grab\_col\_names(df)  
  
def label\_encoder(dataframe, binary\_col):  
 labelencoder = LabelEncoder()  
 dataframe[binary\_col] = labelencoder.fit\_transform(dataframe[binary\_col])  
 return dataframe  
  
binary\_cols = [col for col in df.columns if df[col].dtypes == "O" and df[col].nunique() == 2]  
binary\_cols  
  
for col in binary\_cols:  
 df = label\_encoder(df, col)  
  
df.head()  
  
cat\_cols = [col for col in cat\_cols if col not in binary\_cols and col not in ["HeartDisease"]]  
cat\_cols  
  
def one\_hot\_encoder(dataframe, categorical\_cols, drop\_first=False):  
 dataframe = pd.get\_dummies(dataframe, columns=categorical\_cols, drop\_first=drop\_first)  
 return dataframe  
  
df = one\_hot\_encoder(df, cat\_cols, drop\_first=True)  
  
df.head()  
  
df.shape  
  
*#BASE MODEL*y= df['HeartDisease']  
X=df.drop('HeartDisease', axis= 1)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=17)  
  
models = [('LR', LogisticRegression(random\_state=12345)),  
 ('KNN', KNeighborsClassifier()),  
 ('CART', DecisionTreeClassifier(random\_state=12345)),  
 ('RF', RandomForestClassifier(random\_state=12345)),  
 ('XGB', XGBClassifier(random\_state=12345)),  
 ("LightGBM", LGBMClassifier(random\_state=12345)),  
 ("CatBoost", CatBoostClassifier(verbose=False, random\_state=12345))]  
  
for name, model in models:  
 cv\_results = cross\_validate(model, X, y, cv=10, scoring=["accuracy", "f1", "roc\_auc", "precision", "recall"])  
 print(f"########## {name} ##########")  
 print(f"Accuracy: {round(cv\_results['test\_accuracy'].mean(), 4)}")  
 print(f"Auc: {round(cv\_results['test\_roc\_auc'].mean(), 4)}")  
 print(f"Recall: {round(cv\_results['test\_recall'].mean(), 4)}")  
 print(f"Precision: {round(cv\_results['test\_precision'].mean(), 4)}")  
 print(f"F1: {round(cv\_results['test\_f1'].mean(), 4)}")  
  
*#Özellik Çıkarımı*df.loc[(df['Age'] >= 28) & (df['Age'] < 47), 'NEW\_AGE\_CAT']= 'young'  
df.loc[(df['Age']>= 47) & (df['Age'] < 54), 'NEW\_AGE\_CAT']= 'senior'  
df.loc[(df['Age']>= 54), 'NEW\_AGE\_CAT']= 'old'  
  
df.head()  
df['NEW\_RestingBP']= pd.cut(x=df['RestingBP'], bins=[0,80, 120,130,140, 200], labels = ['Under','Normal','Careful','Severe','AlmostDie'])  
df['NEW\_Cholesterol']= pd.cut(x=df['Cholesterol'], bins=[0,85,214,237,267,576], labels = ['Under\_Cho','Normal\_Cho','Careful\_Cho','Severe\_Cho','AlmostDie\_Cho'])  
  
df.head()  
  
df['NEW\_MaxHR']= pd.cut(x=df['MaxHR'], bins=[0,80,120,202], labels = ['Normal','Careful','High'])  
df["NEW\_AGE\*CHOLESTEROL"] = df["Age"] \* df["Cholesterol"]  
df["NEW\_AGE\*MAXHR"] = df["Age"] \* df["MaxHR"]  
df['NEW\_AGE\*OLdPeak']= df['Age']\*df['Oldpeak']  
df['NEW\_AGE\*RestingBP']= df['Age']\* df['RestingBP']  
df['NEWCHOLOESTEROL\*RESTİNGBP']= df['Cholesterol']\*df['RestingBP']  
  
df['NEW\_CHOLESTOROL\*Olppeak']= df['Cholesterol']\*df['Oldpeak']  
df['NEW\_RestingBP\*MaxHR']= df['RestingBP']\* df['MaxHR']  
df['NEW\_Cholesterol\*MaxHR']= df['Cholesterol']\*df['MaxHR']  
df['NEW\_Oldpeak\*MaxHR']= df['MaxHR']\* df['Oldpeak']  
df['NEW\_RestingBP']= df['RestingBP']\*df['Oldpeak']  
  
  
df.shape  
  
cat\_cols, num\_cols, cat\_but\_car = grab\_col\_names(df)  
  
cat\_cols  
num\_cols  
  
binary\_cols = [col for col in df.columns if df[col].dtypes == "O" and df[col].nunique() == 2]  
binary\_cols  
  
cat\_cols = [col for col in cat\_cols if col not in binary\_cols and col not in ["HeartDisease"]]  
cat\_cols  
  
def one\_hot\_encoder(dataframe, categorical\_cols, drop\_first=False):  
 dataframe = pd.get\_dummies(dataframe, columns=categorical\_cols, drop\_first=drop\_first)  
 return dataframe  
  
df = one\_hot\_encoder(df, cat\_cols, drop\_first=True)  
  
df.head()  
  
num\_cols  
  
scaler = StandardScaler()  
df[num\_cols] = scaler.fit\_transform(df[num\_cols])  
  
df.head()  
  
*#fınal model*from sklearn.model\_selection import GridSearchCV  
  
y= df['HeartDisease']  
X=df.drop('HeartDisease', axis= 1)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=17)  
  
models = [('LR', LogisticRegression(random\_state=12345)),  
 ('KNN', KNeighborsClassifier()),  
 ('CART', DecisionTreeClassifier(random\_state=12345)),  
 ('RF', RandomForestClassifier(random\_state=12345)),  
 ('XGB', XGBClassifier(random\_state=12345)),  
 ("LightGBM", LGBMClassifier(random\_state=12345)),  
 ("CatBoost", CatBoostClassifier(verbose=False, random\_state=12345))]  
  
for name, model in models:  
 cv\_results = cross\_validate(model, X, y, cv=10, scoring=["accuracy", "f1", "roc\_auc", "precision", "recall"])  
 print(f"########## {name} ##########")  
 print(f"Accuracy: {round(cv\_results['test\_accuracy'].mean(), 4)}")  
 print(f"Auc: {round(cv\_results['test\_roc\_auc'].mean(), 4)}")  
 print(f"Recall: {round(cv\_results['test\_recall'].mean(), 4)}")  
 print(f"Precision: {round(cv\_results['test\_precision'].mean(), 4)}")  
 print(f"F1: {round(cv\_results['test\_f1'].mean(), 4)}")  
  
*########## LightGBM ##########  
#Accuracy: 0.8462  
#Auc: 0.9114  
#Recall: 0.8656  
#Precision: 0.8612  
#F1: 0.86  
########## CatBoost ##########  
#Accuracy: 0.8527  
#Auc: 0.9189  
#Recall: 0.8775  
#Precision: 0.8626  
#F1: 0.8663  
  
  
################################################  
# LightGBM  
################################################*lgbm\_model = LGBMClassifier(random\_state=17)  
  
lgbm\_params = {"learning\_rate": [0.01, 0.1, 0.001],  
 "n\_estimators": [100, 300, 500, 1000],  
 "colsample\_bytree": [0.5, 0.7, 1]}  
  
lgbm\_best\_grid = GridSearchCV(lgbm\_model, lgbm\_params, cv=5, n\_jobs=-1, verbose=True).fit(X, y)  
  
lgbm\_final = lgbm\_model.set\_params(\*\*lgbm\_best\_grid.best\_params\_, random\_state=17).fit(X, y)  
  
cv\_results = cross\_validate(lgbm\_final, X, y, cv=10, scoring=["accuracy", "f1", "roc\_auc"])  
cv\_results['test\_accuracy'].mean()  
cv\_results['test\_f1'].mean()  
cv\_results['test\_roc\_auc'].mean()  
*#cv\_results['test\_accuracy'].mean()  
 #0.8495341614906833  
#cv\_results['test\_f1'].mean()  
 #0.8637177529379612  
#cv\_results['test\_roc\_auc'].mean()  
#0.9193553323768533*