In [1]:
```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from category_encoders import OrdinalEncoder
import os
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler
```

In [ ]:
```
df = pd.read_csv('./data/tmp777.csv', encoding='utf-8')
df.drop(columns=['loanapply_insert_time','insert_time'],axis=1, inplace =True)
categorical_feats = ['bank_id','product_id','income_type', 'employment_type', 'houseown_type', 'purpose']

cat_df = df[categorical_feats]

for c in categorical_feats:
    df[c] = df[c].astype('category')
```

In [ ]:
```
enc1 = OrdinalEncoder(cols = cat_df.columns)
cat_df = enc1.fit_transform(cat_df)

df_tmp1 = df.drop(categorical_feats, axis = 1)
df_tmp1.reset_index(drop=True, inplace=True)
cat_df.reset_index(drop=True, inplace=True)
df = pd.concat([df_tmp1,cat_df], axis = 1)


scaler_df = df.drop(['is_applied','bank_id','product_id'], axis = 1)

scale_df_col = scaler_df.columns

scaler = RobustScaler()
df_robust = scaler.fit_transform(scaler_df)
df_robust = pd.DataFrame(df_robust, columns = scale_df_col)
target = df[['is_applied','bank_id','product_id']]
target.reset_index(drop=True, inplace=True)
afterscale_df = pd.concat([df_robust,target], axis = 1)
afterscale_df['is_applied'] = afterscale_df['is_applied'].astype('int')


user_data_col = ['is_applied', 'birth_year', 'gender',
       'yearly_income', 'desired_amount',
       'existing_loan_cnt', 'existing_loan_amt', 'kospi', 'log_length',
       'OpenApp', 'Login', 'CompleteIDCertification', 'UseDSRCalc',
       'UsePrepayCalc', 'SignUp', 'StartLoanApply', 'EndLoanApply',
       'GetCreditInfo', 'UseLoanManage', 'ViewLoanApplyIntro',
       'income_type', 'employment_type', 'houseown_type', 'purpose','credit_score','company_enter_month'] #
loan_data_col = ['loan_limit', 'loan_rate', 'loan_limit_rank', 'loan_rate_rank', 'bank_id', 'product_id']

user_data=df[user_data_col]
loan_data=df[loan_data_col]


user_X = user_data.drop(['is_applied'], axis=1)
user_y = user_data['is_applied']
del user_data

loan_X2 = loan_data.drop(['product_id'], axis=1)
loan_y2 = loan_data['product_id']


loan_X3 = loan_data.drop(['bank_id'], axis=1)
loan_y3 = loan_data['bank_id']
del loan_data

X_train3, X_test3, Y_train3, Y_test3 = train_test_split(loan_X3, loan_y3, test_size = 0.2, random_state = 42)
ix_test = X_test3.index
ix_train = [i for i in df.index if i not in ix_test]
X_train1 = user_X.iloc[ix_train]
X_test1 =  user_X.iloc[ix_test]
Y_train1 = user_y.iloc[ix_train]
Y_test1 = user_y.iloc[ix_test]
X_train2 = loan_X2.iloc[ix_train]
X_test2 =  loan_X2.iloc[ix_test]
Y_train2 = loan_y2.iloc[ix_train]
Y_test2 = loan_y2.iloc[ix_test]
```

In [ ]:
```
from lightgbm import LGBMClassifier

best_score = 0
best_idx = -1
```

```
idx = 1
for n_estim0 in [200,20,500,1000]:
    for n_estim1 in [200,20,500,1000]:
        for md1 in [16, 10,22]:
            for md2 in [16,10,22]:
                for nl1 in [80,100,60]:
                    for nl2 in [80,100,60]:
                        print(f'Param n_estim0 : {n_estim0}, n_estim1 : {n_estim1}, md1 : {md1}, md2 : {md2}, n
                        rfc3 = LGBMClassifier(n_estimators = n_estim0,
                            learning_rate=0.05,
                                # n_estimators 랑 같은 것 같음
                            max_depth = md1,
                            num_leaves = nl1,
                            n_jobs=-1,
                            scale_pos_weight=5,
                            boosting_type='goss',
                            boost_from_average=False,
                            application  = 'binary',
                            force_col_wise=True,
                            verbose=-1,
                            silent=True)

                        rfc3.fit(X_train3,Y_train3)
                        loan_prob3 = rfc3.predict_proba(X_test3)

                        rfc1 = LGBMClassifier(n_estimators = n_estim1,
                            learning_rate=0.05,
                            max_depth = md2,
                            num_leaves = nl2,
                            n_jobs=-1,
                            scale_pos_weight=5,
                            boosting_type='goss',
                            boost_from_average=False,
                            application  = 'binary',
                            force_col_wise=True,
                            verbose=-1,
                            silent=True)
                        rfc1.fit(X_train1,Y_train1)
                        user_prob = rfc1.predict_proba(X_test1)


                        result_col = []
                        for i in range(len(Y_test3.values)):
                            if Y_test3.values[i] <= 61:
                                result_col.append(loan_prob3[i][Y_test3.values[i]-1] * user_prob[:,1][i])

                        real_pred2 = []
                        min_th = 10000000
                        for criteria in range(1,9000):
                            result_bool2 = [0 if i <= criteria * 0.0001 else 1 for i in result_col]
                            if abs( (sum(result_bool2)/len(result_bool2)) - 0.058) < 0.0005:
                                print(f'Current Criteria {criteria}')
                                real_pred2 = result_bool2
                                break

                            if abs( (sum(result_bool2)/len(result_bool2)) - 0.058) < min_th :
                                min_th = abs( (sum(result_bool2)/len(result_bool2)) - 0.058)
                                real_pred2 = result_bool2


                        y_true = Y_test1
                        y_pred2 = real_pred2

                        cur_score = f1_score(y_true, y_pred2, average='macro')
                        if best_score < cur_score:
                            best_idx= idx
                            best_score = cur_score
                        print(cur_score)

                        idx += 1

print(f'{best_score}, {best_idx}')
```