Home Credit Default Risk : Flask API Deployment Pipeline

1. Importing the Necessary Libraries

In [2]:

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

import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import numpy as np
import gc
import xgboost as xgb
import lightgbm as lgb
import seaborn as sns
import math
import pickle
import os

from lightgbm import LGBMClassifier
from sklearn.metrics import roc_auc_score
from scipy.stats import randint as sp_randint
from sklearn.model_selection import KFold, StratifiedKFold
from prettytable import PrettyTable
from sklearn.metrics import roc_curve,auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.linear_model import SGDClassifier
from collections import Counter
from scipy.sparse import hstack
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from bayes_opt import BayesianOptimization
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from datetime import datetime
```

2. Class which has all the Necessary Functions Defined

In [1]:

```
class initial_function_definition:

    def reduce_memory_usage(df):

        start_mem = df.memory_usage().sum() / 1024**2
        print('Memory usage of dataframe is {:.2f} MB'.format(start_mem
))

        for col in df.columns:
            col_type = df[col].dtype

            if col_type != object:
                c_min = df[col].min()
                c_max = df[col].max()
                if str(col_type)[:3] == 'int':
                    if c_min > np.iinfo(np.int8).min and c_max < np.iin
fo(np.int8).max:
                        df[col] = df[col].astype(np.int8)
                    elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)
                    elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)
                    elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)
                else:
                    if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                        df[col] = df[col].astype(np.float16)
                    elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                        df[col] = df[col].astype(np.float32)
                    else:
                        df[col] = df[col].astype(np.float64)

        end_mem = df.memory_usage().sum() / 1024**2
        print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
        print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

        return df



    def fix_nulls_outliers(data):

        #Replace NA with the most frequently occuring class for Count of Client Family Members
        data['CNT_FAM_MEMBERS'].fillna(data['CNT_FAM_MEMBERS'].value_counts().idxmax(), \
                                     inplace=True)
        data.replace(max(data['DAYS_EMPLOYED'].values), np.nan, inplace=True)
        data['NAME_FAMILY_STATUS'].fillna('Data_Not_Available', inplace=True)
        data['NAME_HOUSING_TYPE'].fillna('Data_Not_Available', inplace=True)
        data['FLAG_MOBIL'].fillna('Data_Not_Available', inplace=True)
        data['FLAG_EMP_PHONE'].fillna('Data_Not_Available', inplace=True)
        data['FLAG_CONT_MOBILE'].fillna('Data_Not_Available', inplace=True)
        data['FLAG_EMAIL'].fillna('Data_Not_Available', inplace=True)
        data['OCCUPATION_TYPE'].fillna('Data_Not_Available', inplace=True)

        #Replace NA with the most frequently occuring class for Count of Client Family Members
        data['CNT_FAM_MEMBERS'].fillna(data['CNT_FAM_MEMBERS'].value_counts().idxmax(), \
                                             inplace=True)
        data.replace(max(data['DAYS_EMPLOYED'].values), np.nan, inplace=True)

        data['CODE_GENDER'].replace('XNA','M',inplace=True)
        data['AMT_ANNUITY'].fillna(0, inplace=True)
        data['AMT_GOODS_PRICE'].fillna(0, inplace=True)
        data['NAME_TYPE_SUITE'].fillna('Unaccompanied', inplace=True)
        data['NAME_FAMILY_STATUS'].replace('Unknown','Married', inplace=True)
        data['OCCUPATION_TYPE'].fillna('Data_Not_Available', inplace=True)

        data['EXT_SOURCE_1'].fillna(0, inplace=True)
        data['EXT_SOURCE_2'].fillna(0, inplace=True)
        data['EXT_SOURCE_3'].fillna(0, inplace=True)

        return data



    def FE_application_data(data):

        data['CREDIT_INCOME_PERCENT'] = data['AMT_CREDIT'] / data['AMT_INCOME_TOTAL']
        data['ANNUITY_INCOME_PERCENT'] = data['AMT_ANNUITY'] / data['AMT_INCOME_TOTAL']
        data['CREDIT_ANNUITY_PERCENT'] = data['AMT_CREDIT'] / data['AMT_ANNUITY']

        data['FAMILY_CNT_INCOME_PERCENT'] = data['AMT_INCOME_TOTAL'] / data['CNT_FAM_MEMBERS']
        data['CREDIT_TERM'] = data['AMT_ANNUITY'] / data['AMT_CREDIT']
        data['BIRTH_EMPLOYED_PERCENT'] = data['DAYS_EMPLOYED'] / data['DAYS_BIRTH']
        data['CHILDREN_CNT_INCOME_PERCENT'] = data['AMT_INCOME_TOTAL']/data['CNT_CHILDREN']

        data['CREDIT_GOODS_DIFF'] = data['AMT_CREDIT'] - data['AMT_GOODS_PRICE']
        data['EMPLOYED_REGISTRATION_PERCENT'] = data['DAYS_EMPLOYED'] / data['DAYS_REGISTRATION']
        data['BIRTH_REGISTRATION_PERCENT'] = data['DAYS_BIRTH'] / data['DAYS_REGISTRATION']
        data['ID_REGISTRATION_DIFF'] = data['DAYS_ID_PUBLISH'] - data['DAYS_REGISTRATION']

        data['ANNUITY_LENGTH_EMPLOYED_PERCENT'] = data['CREDIT_TERM']/data['DAYS_EMPLOYED']

        data['AGE_LOAN_FINISH'] = data['DAYS_BIRTH']*(-1.0/365) + \
                         (data['AMT_CREDIT']/data['AMT_ANNUITY']) *(1.0/12)
        #(This basically refers to the client's age when he/she finishes loan repayment)

        data['CAR_AGE_EMP_PERCENT'] = data['OWN_CAR_AGE']/data['DAYS_EMPLOYED']
        data['CAR_AGE_BIRTH_PERCENT'] = data['OWN_CAR_AGE']/data['DAYS_BIRTH']
        data['PHONE_CHANGE_EMP_PERCENT'] = data['DAYS_LAST_PHONE_CHANGE']/data['DAYS_EMPLOYED']
        data['PHONE_CHANGE_BIRTH_PERCENT'] = data['DAYS_LAST_PHONE_CHANGE']/data['DAYS_BIRTH']
        income_by_contract = data[['AMT_INCOME_TOTAL', 'NAME_CONTRACT_TYPE']].groupby('NAME_CONTRACT_TYPE').median()['AMT_INCOME_TOTAL']
        data['MEDIAN_INCOME_CONTRACT_TYPE'] = data['NAME_CONTRACT_TYPE'].map(income_by_contract)

        income_by_suite = data[['AMT_INCOME_TOTAL', 'NAME_TYPE_SUITE']].groupby('NAME_TYPE_SUITE').median()['AMT_INCOME_TOTAL']
        data['MEDIAN_INCOME_SUITE_TYPE'] = data['NAME_TYPE_SUITE'].map(income_by_suite)

        income_by_housing = data[['AMT_INCOME_TOTAL', 'NAME_HOUSING_TYPE']].groupby('NAME_HOUSING_TYPE').median()['AMT_INCOME_TOTAL']
        data['MEDIAN_INCOME_HOUSING_TYPE'] = data['NAME_HOUSING_TYPE'].map(income_by_housing)

        income_by_org = data[['AMT_INCOME_TOTAL', 'ORGANIZATION_TYPE']].groupby('ORGANIZATION_TYPE').median()['AMT_INCOME_TOTAL']
        data['MEDIAN_INCOME_ORG_TYPE'] = data['ORGANIZATION_TYPE'].map(income_by_org)

        income_by_occu = data[['AMT_INCOME_TOTAL', 'OCCUPATION_TYPE']].groupby('OCCUPATION_TYPE').median()['AMT_INCOME_TOTAL']
        data['MEDIAN_INCOME_OCCU_TYPE'] = data['OCCUPATION_TYPE'].map(income_by_occu)

        income_by_education = data[['AMT_INCOME_TOTAL', 'NAME_EDUCATION_TYPE']].groupby('NAME_EDUCATION_TYPE').median()['AMT_INCOME_TOTAL']
        data['MEDIAN_INCOME_EDU_TYPE'] = data['NAME_EDUCATION_TYPE'].map(income_by_education)

        data['ORG_TYPE_INCOME_PERCENT'] = data['MEDIAN_INCOME_ORG_TYPE']/data['AMT_INCOME_TOTAL']
        data['OCCU_TYPE_INCOME_PERCENT'] = data['MEDIAN_INCOME_OCCU_TYPE']/data['AMT_INCOME_TOTAL']
        data['EDU_TYPE_INCOME_PERCENT'] = data['MEDIAN_INCOME_EDU_TYPE']/data['AMT_INCOME_TOTAL']

        data= data.drop(['FLAG_DOCUMENT_2','FLAG_DOCUMENT_4','FLAG_DOCUMENT_5','FLAG_DOCUMENT_6','FLAG_DOCUMENT_7',
        'FLAG_DOCUMENT_8','FLAG_DOCUMENT_9','FLAG_DOCUMENT_10', 'FLAG_D
```

```
OCUMENT_11','FLAG_DOCUMENT_12','FLAG_DOCUMENT_13',
        'FLAG_DOCUMENT_14','FLAG_DOCUMENT_15','FLAG_DOCUMENT_16','FLAG_
DOCUMENT_17','FLAG_DOCUMENT_18','FLAG_DOCUMENT_19',
        'FLAG_DOCUMENT_20','FLAG_DOCUMENT_21'],axis=1)

        cat_col = [category for category in data.columns if data[catego
ry].dtype == 'object']
        data = pd.get_dummies(data, columns= cat_col)

        return data


    def one_hot_encode(df):

        original_columns = list(df.columns)
        categories = [cat for cat in df.columns if df[cat].dtype == 'ob
ject']
        df = pd.get_dummies(df, columns= categories, dummy_na= True) #o
ne_hot_encode the categorical features
        categorical_columns = [cat for cat in df.columns if cat not in
original_columns]
        return df, categorical_columns


    def generate_credit_type_code(x):

        if x == 'Closed':
            y = 0
        elif x=='Active':
            y = 1
        else:
            y = 2
        return y


    def FE_bureau_data_1(bureau_data):

        bureau_data['CREDIT_DURATION'] = -bureau_data['DAYS_CREDIT'] +
bureau_data['DAYS_CREDIT_ENDDATE']
        bureau_data['ENDDATE_DIFF'] = bureau_data['DAYS_CREDIT_ENDDATE'
```

```
] - bureau_data['DAYS_ENDDATE_FACT']
        bureau_data['UPDATE_DIFF'] = bureau_data['DAYS_CREDIT_ENDDATE']
- bureau_data['DAYS_CREDIT_UPDATE']
        bureau_data['DEBT_PERCENTAGE'] = bureau_data['AMT_CREDIT_SUM']
/ bureau_data['AMT_CREDIT_SUM_DEBT']
        bureau_data['DEBT_CREDIT_DIFF'] = bureau_data['AMT_CREDIT_SUM']
- bureau_data['AMT_CREDIT_SUM_DEBT']
        bureau_data['CREDIT_TO_ANNUITY_RATIO'] = bureau_data['AMT_CREDI
T_SUM'] / bureau_data['AMT_ANNUITY']
        bureau_data['DEBT_TO_ANNUITY_RATIO'] = bureau_data['AMT_CREDIT_
SUM_DEBT'] / bureau_data['AMT_ANNUITY']
        bureau_data['CREDIT_OVERDUE_DIFF'] = bureau_data['AMT_CREDIT_SU
M'] - bureau_data['AMT_CREDIT_SUM_OVERDUE']

        #Refer :- https://www.kaggle.com/c/home-credit-default-risk/dis
cussion/57750
        #Calculating the Number of Past Loans for each Customer
        no_loans_per_customer = bureau_data[['SK_ID_CURR', 'SK_ID_BUREA
U']].groupby(by = \
                                                                    ['S
K_ID_CURR'])['SK_ID_BUREAU'].count()
        no_loans_per_customer = no_loans_per_customer.reset_index().ren
ame(columns={'SK_ID_BUREAU': 'CUSTOMER_LOAN_COUNT'})
        bureau_data = bureau_data.merge(no_loans_per_customer, on='SK_I
D_CURR', how='left')

        #Calculating the Past Credit Types per Customer
        credit_types_per_customer = bureau_data[['SK_ID_CURR','CREDIT_T
YPE']].groupby(by=['SK_ID_CURR'])['CREDIT_TYPE'].nunique()
        credit_types_per_customer = credit_types_per_customer.reset_ind
ex().rename(columns={'CREDIT_TYPE':'CUSTOMER_CREDIT_TYPES'})
        bureau_data = bureau_data.merge(credit_types_per_customer, on=
'SK_ID_CURR',how='left')

        #Average Loan Type per Customer
        bureau_data['AVG_LOAN_TYPE'] = bureau_data['CUSTOMER_LOAN_COUN
T']/bureau_data['CUSTOMER_CREDIT_TYPES']

        bureau_data['CREDIT_TYPE_CODE'] = bureau_data.apply(lambda x:\
                                        initial_function_definition.gen
```

```
erate_credit_type_code(x.CREDIT_ACTIVE), axis=1)

        customer_credit_code_mean = bureau_data[['SK_ID_CURR','CREDIT_TYPE_CODE']].groupby(by=['SK_ID_CURR'])['CREDIT_TYPE_CODE'].mean()
        customer_credit_code_mean.reset_index().rename(columns={'CREDIT_TYPE_CODE':'CUSTOMER_CREDIT_CODE_MEAN'})
        bureau_data = bureau_data.merge(customer_credit_code_mean, on='SK_ID_CURR', how='left')

        #Computing the Ratio of Total Customer Credit and the Total Customer Debt
        bureau_data['AMT_CREDIT_SUM'] = bureau_data['AMT_CREDIT_SUM'].fillna(0)
        bureau_data['AMT_CREDIT_SUM_DEBT'] = bureau_data['AMT_CREDIT_SUM_DEBT'].fillna(0)
        bureau_data['AMT_ANNUITY'] = bureau_data['AMT_ANNUITY'].fillna(0)

        credit_sum_customer = bureau_data[['SK_ID_CURR','AMT_CREDIT_SUM']].groupby(by=['SK_ID_CURR'])['AMT_CREDIT_SUM'].sum()
        credit_sum_customer = credit_sum_customer.reset_index().rename(columns={'AMT_CREDIT_SUM':'TOTAL_CREDIT_SUM'})
        bureau_data = bureau_data.merge(credit_sum_customer, on='SK_ID_CURR', how='left')

        credit_debt_sum_customer = bureau_data[['SK_ID_CURR','AMT_CREDIT_SUM_DEBT']].groupby(by=['SK_ID_CURR'])['AMT_CREDIT_SUM_DEBT'].sum()
        credit_debt_sum_customer = credit_debt_sum_customer.reset_index().rename(columns={'AMT_CREDIT_SUM_DEBT':'TOTAL_DEBT_SUM'})
        bureau_data = bureau_data.merge(credit_debt_sum_customer, on='SK_ID_CURR', how='left')
        bureau_data['CREDIT_DEBT_RATIO'] = bureau_data['TOTAL_CREDIT_SUM']/bureau_data['TOTAL_DEBT_SUM']

        return bureau_data


    def FE_bureau_data_2(bureau_data,bureau_balance,bureau_data_columns,bureau_balance_columns):
        bureau_balance_agg = {'MONTHS_BALANCE': ['min','max','mean','si
ze']}

        for column in bureau_balance_columns:
            bureau_balance_agg[column] = ['min','max','mean','size']
            bureau_balance_final_agg = bureau_balance.groupby('SK_ID_BU
REAU').agg(bureau_balance_agg)

        col_list_1 =[]

        for col in bureau_balance_final_agg.columns.tolist():
            col_list_1.append(col[0] + "_" + col[1].upper())

        bureau_balance_final_agg.columns = pd.Index(col_list_1)
        bureau_data_balance = bureau_data.join(bureau_balance_final_agg
, how='left', on='SK_ID_BUREAU')
        bureau_data_balance.drop(['SK_ID_BUREAU'], axis=1, inplace= Tru
e)

        del bureau_balance_final_agg
        gc.collect()


        numerical_agg = {'AMT_CREDIT_SUM_DEBT': ['mean', 'sum'],'AMT_CR
EDIT_SUM_OVERDUE': ['mean','sum'],
        'DAYS_CREDIT': ['mean', 'var'],'DAYS_CREDIT_UPDATE': ['mean','m
in'],'CREDIT_DAY_OVERDUE': ['mean','min'],
        'DAYS_CREDIT_ENDDATE': ['mean'],'CNT_CREDIT_PROLONG': ['sum'],
'MONTHS_BALANCE_SIZE': ['mean', 'sum'],
        'AMT_CREDIT_SUM_LIMIT': ['mean', 'sum'],'AMT_CREDIT_MAX_OVERDU
E': ['mean','max'],
        'AMT_ANNUITY': ['max', 'mean','sum'],'AMT_CREDIT_SUM': ['mean',
'sum','max']}

        categorical_agg = {}

        for col in bureau_data_columns:
            categorical_agg[col] = ['mean']
            categorical_agg[col] = ['max']
        for col in bureau_balance_columns:
            categorical_agg[col + "_MEAN"] = ['mean']
            categorical_agg[col + "_MIN"] = ['min']
            categorical_agg[col + "_MAX"] = ['max']

        bureau_data_balance_2 = bureau_data_balance.groupby('SK_ID_CUR
R').agg({**numerical_agg,\

**categorical_agg})
        col_list_2=[]

        for col in bureau_data_balance_2.columns.tolist():
            col_list_2.append('BUREAU_'+col[0]+'_'+col[1])
        bureau_data_balance_2.columns = pd.Index(col_list_2)


        bureau_data_balance_3 = bureau_data_balance[bureau_data_balance
['CREDIT_ACTIVE_Active'] == 1]
        bureau_data_balance_3_agg = bureau_data_balance_3.groupby('SK_I
D_CURR').agg(numerical_agg)

        col_list_3=[]

        for col in bureau_data_balance_3_agg.columns.tolist():
            col_list_3.append('A_'+col[0]+'_'+col[1].upper())

        bureau_data_balance_3_agg.columns = pd.Index(col_list_3)
        b3_final = bureau_data_balance_2.join(bureau_data_balance_3_agg
, how='left', on='SK_ID_CURR')

        bureau_data_balance_4 = bureau_data_balance[bureau_data_balance
['CREDIT_ACTIVE_Closed'] == 1]
        bureau_data_balance_4_agg = bureau_data_balance_4.groupby('SK_I
D_CURR').agg(numerical_agg)

        col_list_4 =[]

        for col in bureau_data_balance_4_agg.columns.tolist():
            col_list_4.append('C_'+col[0]+'_'+col[1].upper())
        bureau_data_balance_4_agg.columns = pd.Index(col_list_4)
        bureau_data_balance_final = bureau_data_balance_2.join(bureau_data_balance_4_agg, how='left', on='SK_ID_CURR')

        del bureau_data_balance_3, bureau_data_balance_4_agg
        gc.collect()

        return bureau_data_balance_final



    def preprocess_previous_application(data):

        data['DAYS_FIRST_DRAWING'].replace(max(data['DAYS_FIRST_DRAWING'].values),np.nan, inplace=True)
        data['DAYS_FIRST_DUE'].replace(np.nan,0, inplace= True)
        data['DAYS_FIRST_DUE'].replace(0,np.nan, inplace= True)
        data['DAYS_FIRST_DUE'].replace(max(data['DAYS_FIRST_DUE'].values),np.nan, inplace=True)

        data['DAYS_LAST_DUE_1ST_VERSION'].replace(np.nan,0, inplace= True)
        data['DAYS_LAST_DUE_1ST_VERSION'].replace(0,np.nan, inplace= True)
        data['DAYS_LAST_DUE_1ST_VERSION'].replace(max(data['DAYS_LAST_DUE_1ST_VERSION'].values),np.nan, inplace=True)

        data['DAYS_LAST_DUE'].replace(np.nan,0, inplace= True)
        data['DAYS_LAST_DUE'].replace(0,np.nan, inplace= True)
        data['DAYS_LAST_DUE'].replace(max(data['DAYS_LAST_DUE'].values),np.nan, inplace=True)

        data['DAYS_TERMINATION'].replace(np.nan,0, inplace= True)
        data['DAYS_TERMINATION'].replace(0,np.nan, inplace= True)
        data['DAYS_TERMINATION'].replace(max(data['DAYS_TERMINATION'].values),np.nan, inplace=True)

        return data
    def FE_previous_application(previous_application):

        prev_app, previous_application_columns = initial_function_definition.one_hot_encode(previous_application)

        prev_app['APPLICATION_CREDIT_DIFF'] = prev_app['AMT_APPLICATION'] - prev_app['AMT_CREDIT']
        prev_app['APPLICATION_CREDIT_RATIO'] = prev_app['AMT_APPLICATION'] / prev_app['AMT_CREDIT']
        prev_app['CREDIT_TO_ANNUITY_RATIO'] = prev_app['AMT_CREDIT']/prev_app['AMT_ANNUITY']
        prev_app['DOWN_PAYMENT_TO_CREDIT'] = prev_app['AMT_DOWN_PAYMENT'] / prev_app['AMT_CREDIT']

        total_payment = prev_app['AMT_ANNUITY'] * prev_app['CNT_PAYMENT']
        prev_app['SIMPLE_INTERESTS'] = (total_payment/prev_app['AMT_CREDIT'] - 1)/prev_app['CNT_PAYMENT']

        prev_app['DAYS_LAST_DUE_DIFF'] = prev_app['DAYS_LAST_DUE_1ST_VERSION'] - prev_app['DAYS_LAST_DUE']

        numerical_agg_prev = {'AMT_ANNUITY': ['max', 'mean'], 'AMT_APPLICATION': ['max','mean'],\
                     'AMT_CREDIT':['max','mean'], 'AMT_DOWN_PAYMENT': ['max','mean'],\
                      'AMT_GOODS_PRICE':['mean','sum'], 'HOUR_APPR_PROCESS_START' :\
                      ['max','mean'], 'RATE_DOWN_PAYMENT':['max','mean'], 'RATE_INTEREST_PRIMARY':\
                      ['max','mean'],'RATE_INTEREST_PRIVILEGED':['max','mean'], \
                      'DAYS_DECISION': ['max','mean'], 'CNT_PAYMENT' :['mean','sum'], \
                      'DAYS_FIRST_DRAWING': ['max','mean'], 'DAYS_TERMINATION' : ['max','mean'],\
                      'APPLICATION_CREDIT_RATIO': ['max','mean'], 'DOWN_PAYMENT_TO_CREDIT' : \
                      ['max','mean'], 'DAYS_LAST_DUE_DIFF': ['max','mea
```

```
n']}

        categorical_agg_prev = {}

        for column in previous_application_columns:
            categorical_agg_prev[column] = ['mean']

        prev_app_agg1 = prev_app.groupby('SK_ID_CURR').agg({**numerical
_agg_prev, **categorical_agg_prev})

        col_list_5 =[]

        for col in prev_app_agg1.columns.tolist():
            col_list_5.append('PREV_'+col[0]+'_'+col[1].upper())

        prev_app_agg1.columns = pd.Index(col_list_5)

        prev_app_cs_approved = prev_app[prev_app['NAME_CONTRACT_STATUS_
Approved']==1]
        prev_app_agg2 = prev_app_cs_approved.groupby('SK_ID_CURR').agg(
numerical_agg_prev)

        col_list_6 = []

        for col in prev_app_agg2.columns.tolist():
            col_list_6.append('CS_APP_' + col[0] + '_' + col[1].upper
())

        prev_app_agg2.columns = pd.Index(col_list_6)
        prev_app_agg1_join = prev_app_agg1.join(prev_app_agg2, how='lef
t', on='SK_ID_CURR')

        prev_app_cs_refused = prev_app[prev_app['NAME_CONTRACT_STATUS_R
efused']==1]
        prev_app_agg3 = prev_app_cs_refused.groupby('SK_ID_CURR').agg(n
umerical_agg_prev)

        col_list_7 =[]

        for col in prev_app_agg3.columns.tolist():
            col_list_7.append('CS_REF_' + col[0] + '_' + col[1].upper())

        prev_app_agg3.columns = pd.Index(col_list_7)
        prev_app_agg_final = prev_app_agg1_join.join(prev_app_agg3,how='left', on='SK_ID_CURR')

        del prev_app_agg1_join, prev_app_agg3, prev_app_cs_refused, prev_app_agg1, prev_app_agg2,prev_app_cs_approved
        gc.collect()

        return prev_app_agg_final



    def FE_previous_application_days_decision(data,data_temp,previous_application):

        temp_1 = initial_function_definition.FE_previous_application(initial_function_definition.reduce_memory_usage(previous_application))
        data = data_temp.merge(temp_1, how='left', on='SK_ID_CURR')
        del temp_1
        gc.collect()

        temp_2 = initial_function_definition.reduce_memory_usage(previous_application[previous_application['DAYS_DECISION']>=-365].reset_index())
        temp_2.drop(['index'], axis=1, inplace=True)
        temp_2 = initial_function_definition.FE_previous_application(temp_2)
        data = data.join(temp_2, how='left', on='SK_ID_CURR',rsuffix='_year')
        del temp_2
        gc.collect()

        temp_3 = initial_function_definition.reduce_memory_usage(previous_application[previous_application['DAYS_DECISION']>=-182].reset_index())
        temp_3.drop(['index'], axis=1, inplace=True)
        temp_3 = initial_function_definition.FE_previous_application(te
mp_3)
        data = data.join(temp_3, how='left', on='SK_ID_CURR', rsuffix=
'_half_year')
        del temp_3
        gc.collect()

        temp_4 = initial_function_definition.reduce_memory_usage(previo
us_application[previous_application['DAYS_DECISION']>=-90].reset_index
())
        temp_4.drop(['index'], axis=1, inplace=True)
        temp_4 = initial_function_definition.FE_previous_application(te
mp_4)
        data = data.join(temp_4, how='left', on='SK_ID_CURR', rsuffix=
'_quarter')
        del temp_4
        gc.collect()

        temp_5 = initial_function_definition.reduce_memory_usage(previo
us_application[previous_application['DAYS_DECISION']>=-30].reset_index
())
        temp_5.drop(['index'], axis=1, inplace=True)
        temp_5 = initial_function_definition.FE_previous_application(te
mp_5)
        data = data.join(temp_5, how='left', on='SK_ID_CURR', rsuffix=
'_month')
        del temp_5
        gc.collect()

        temp_6 = initial_function_definition.reduce_memory_usage(previo
us_application[previous_application['DAYS_DECISION']>=-14].reset_index
())
        temp_6.drop(['index'], axis=1, inplace=True)
        temp_6 = initial_function_definition.FE_previous_application(te
mp_6)
        data = data.join(temp_6, how='left', on='SK_ID_CURR', rsuffix=
'_fortnight')
        del temp_6
        gc.collect()

        temp_7 = initial_function_definition.reduce_memory_usage(previo
us_application[previous_application['DAYS_DECISION']>=-7].reset_index
())
        temp_7.drop(['index'], axis=1, inplace=True)
        temp_7 = initial_function_definition.FE_previous_application(te
mp_7)
        data = data.join(temp_7, how='left', on='SK_ID_CURR', rsuffix=
'_week')
        del temp_7
        gc.collect()

        return data


    def FE_pos_cash_balance(pos_cash_balance):

        pos_balance_data, pos_balance_columns = initial_function_defini
tion.one_hot_encode(pos_cash_balance)

        pos_balance_data['LATE_PAYMENT'] = pos_balance_data['SK_DPD'].a
pply(lambda x:1 if x>0 else 0)

        numerical_agg_pos_balance = {'SK_DPD_DEF': ['max', 'mean','min'
],'SK_DPD': ['max', 'mean','min'],
        'MONTHS_BALANCE': ['max', 'mean', 'size'], 'CNT_INSTALMENT': [
'max','size'],
        'CNT_INSTALMENT_FUTURE': ['max','size','sum']}

        categorical_agg_pos_balance = {}

        for col in pos_balance_columns:
            categorical_agg_pos_balance[col] = ['mean']

        pos_balance_agg = pos_balance_data.groupby('SK_ID_CURR').agg({*
*numerical_agg_pos_balance, **categorical_agg_pos_balance})

        col_list_8=[]

        for col in pos_balance_agg.columns.tolist():
            col_list_8.append('POS_'+col[0] + '_' + col[1].upper())
        pos_balance_agg.columns = pd.Index(col_list_8)

        sort_pos_balance = pos_balance_data.sort_values(by=['SK_ID_PREV', 'MONTHS_BALANCE'])
        pos_group = sort_pos_balance.groupby('SK_ID_PREV')

        pos_final_df = pd.DataFrame()
        pos_final_df['SK_ID_CURR'] = pos_group['SK_ID_CURR'].first()
        pos_final_df['MONTHS_BALANCE_MAX'] = pos_group['MONTHS_BALANCE'].max()

        pos_final_df['POS_LOAN_COMPLETED_MEAN'] = pos_group['NAME_CONTRACT_STATUS_Completed'].mean()
        pos_final_df['POS_COMPLETED_BEFORE_MEAN'] = pos_group['CNT_INSTALMENT'].first() - pos_group['CNT_INSTALMENT'].last()

        pos_final_df['POS_COMPLETED_BEFORE_MEAN'] = pos_final_df.apply(lambda x: 1 if x['POS_COMPLETED_BEFORE_MEAN'] > 0
                                                and x['POS_LOAN_COMPLETED_MEAN'] > 0 else 0, axis=1)

        pos_final_df['POS_REMAINING_INSTALMENTS'] = pos_group['CNT_INSTALMENT_FUTURE'].last()
        pos_final_df['POS_REMAINING_INSTALMENTS_RATIO'] = pos_group['CNT_INSTALMENT_FUTURE'].last()/pos_group['CNT_INSTALMENT'].last()

        pos_final_df_groupby = pos_final_df.groupby('SK_ID_CURR').sum().reset_index()
        pos_final_df_groupby.drop(['MONTHS_BALANCE_MAX'], axis=1, inplace= True)
        pos_final_agg = pd.merge(pos_balance_agg, pos_final_df_groupby, on= 'SK_ID_CURR', how= 'left')

        del pos_balance_agg, pos_final_df_groupby, pos_group, sort_pos_balance
        gc.collect()
        return pos_final_agg


    def FE_pos_cash_balance_months_balance(data, data_temp, pos_cash_ba
lance):

        temp_8 = initial_function_definition.FE_pos_cash_balance(initia
l_function_definition.reduce_memory_usage(pos_cash_balance))
        data = data_temp.merge(temp_8, how='left', on='SK_ID_CURR')
        del temp_8
        gc.collect()

        temp_9 = initial_function_definition.reduce_memory_usage(pos_ca
sh_balance[pos_cash_balance['MONTHS_BALANCE']>=-12].reset_index())
        temp_9.drop(['index'], axis=1, inplace=True)
        temp_9 = initial_function_definition.FE_pos_cash_balance(temp_9
)
        data = data.join(temp_9, how='left', on='SK_ID_CURR',rsuffix='_
year')
        del temp_9
        gc.collect()

        temp_10 = initial_function_definition.reduce_memory_usage(pos_c
ash_balance[pos_cash_balance['MONTHS_BALANCE']>=-6].reset_index())
        temp_10.drop(['index'], axis=1, inplace=True)
        temp_10 = initial_function_definition.FE_pos_cash_balance(temp_
10)
        data = data.join(temp_10, how='left', on='SK_ID_CURR', rsuffix=
'_half_year')
        del temp_10
        gc.collect()

        temp_11 = initial_function_definition.reduce_memory_usage(pos_c
ash_balance[pos_cash_balance['MONTHS_BALANCE']>=-3].reset_index())
        temp_11.drop(['index'], axis=1, inplace=True)
        temp_11 = initial_function_definition.FE_pos_cash_balance(temp_
11)
        data = data.join(temp_11, how='left', on='SK_ID_CURR', rsuffix=
'_quarter')
        del temp_11
        gc.collect()

        temp_12 = initial_function_definition.reduce_memory_usage(pos_c
ash_balance[pos_cash_balance['MONTHS_BALANCE']>=-1].reset_index())
```

```
        temp_12.drop(['index'], axis=1, inplace=True)
        temp_12 = initial_function_definition.FE_pos_cash_balance(temp_
12)
        data = data.join(temp_12, how='left', on='SK_ID_CURR', rsuffix=
'_month')
        del temp_12
        gc.collect()

        return data



    def FE_installments_payments(installments_payments):

        pay1 = installments_payments[['SK_ID_PREV', 'NUM_INSTALMENT_NUM
BER']+ ['AMT_PAYMENT']]
        pay2 = pay1.groupby(['SK_ID_PREV', 'NUM_INSTALMENT_NUMBER'])['A
MT_PAYMENT'].sum().reset_index()
        pay_final = pay2.rename(columns={'AMT_PAYMENT': 'AMT_PAYMENT_GR
OUPED'})
        payments_final = installments_payments.merge(pay_final,\
                            on=['SK_ID_PREV','NUM_INSTALMENT_NUMBER'],
how='left')

        payments_final['PAYMENT_DIFFERENCE'] = payments_final['AMT_INST
ALMENT'] - payments_final['AMT_PAYMENT_GROUPED']
        payments_final['PAYMENT_RATIO'] = payments_final['AMT_INSTALMEN
T'] / payments_final['AMT_PAYMENT_GROUPED']

        payments_final['PAID_OVER_AMOUNT'] = payments_final['AMT_PAYMEN
T'] - payments_final['AMT_INSTALMENT']
        payments_final['PAID_OVER'] = (payments_final['PAID_OVER_AMOUN
T'] > 0).astype(int)

        payments_final['DPD'] = payments_final['DAYS_ENTRY_PAYMENT'] -
 \
                        payments_final['DAYS_INSTALMENT']
        payments_final['DPD'] = payments_final['DPD'].apply(lambda x: 0
if x <= 0 else x)
```

```
        payments_final['DBD'] = payments_final['DAYS_INSTALMENT'] - \
                        payments_final['DAYS_ENTRY_PAYMENT']
        payments_final['DBD'] = payments_final['DBD'].apply(lambda x: 0 if x <= 0 else x)
        payments_final['LATE_PAYMENT'] = payments_final['DBD'].apply(lambda x: 1 if x > 0 else 0)

        payments_final['INSTALMENT_PAYMENT_RATIO'] = payments_final['AMT_PAYMENT'] / payments_final['AMT_INSTALMENT']
        payments_final['LATE_PAYMENT_RATIO'] = payments_final.apply(lambda x: x['INSTALMENT_PAYMENT_RATIO'] if x['LATE_PAYMENT'] == 1 else 0, axis=1)

        payments_final['SIGNIFICANT_LATE_PAYMENT'] = payments_final['LATE_PAYMENT_RATIO'].apply(lambda x: 1 if x > 0.05 else 0)

        payments_final['DPD_7'] = payments_final['DPD'].apply(lambda x: 1 if x >= 7 else 0)
        payments_final['DPD_15'] = payments_final['DPD'].apply(lambda x: 1 if x >= 15 else 0)
        payments_final['DPD_30'] = payments_final['DPD'].apply(lambda x: 1 if x >= 30 else 0)
        payments_final['DPD_60'] = payments_final['DPD'].apply(lambda x: 1 if x >= 60 else 0)
        payments_final['DPD_90'] = payments_final['DPD'].apply(lambda x: 1 if x >= 90 else 0)
        payments_final['DPD_180'] = payments_final['DPD'].apply(lambda x: 1 if x >= 180 else 0)
        payments_final['DPD_WOF'] = payments_final['DPD'].apply(lambda x: 1 if x >= 720 else 0)

        payments_final, pay_final_columns = initial_function_definition.one_hot_encode(payments_final)

        numeric_agg_payments = {'LATE_PAYMENT': ['max','mean','min'],'AMT_PAYMENT': ['min', 'max',\
                      'mean', 'sum'], 'NUM_INSTALMENT_VERSION': ['nunique'], \
                      'NUM_INSTALMENT_NUMBER':['max'], 'AMT_INSTALMENT': ['max', 'mean', 'sum'],
        'PAYMENT_DIFFERENCE': ['max','mean','min','sum'],'DAYS_ENTRY_PAYMENT': ['max', \
        'mean', 'sum'],  'PAID_OVER_AMOUNT': ['max','mean','min']}

        for col in pay_final_columns:
            numeric_agg_payments[col] = ['mean']

        payments_final_agg = payments_final.groupby('SK_ID_CURR').agg(numeric_agg_payments)
        col_list_9=[]

        for col in payments_final_agg.columns.tolist():
            col_list_9.append('INS_'+col[0]+'_'+col[1].upper())

        payments_final_agg.columns = pd.Index(col_list_9)
        payments_final_agg['INSTALLATION_COUNT'] = payments_final.groupby('SK_ID_CURR').size()

        del payments_final
        gc.collect()

        return payments_final_agg


    def FE_installments_payments_days_instalment(data, data_temp, installments_payments):

        installments_payments['DAYS_ENTRY_PAYMENT'].fillna(0, inplace=True)
        installments_payments['AMT_PAYMENT'].fillna(0.0, inplace=True)

        temp_13 = initial_function_definition.FE_installments_payments(initial_function_definition.reduce_memory_usage(installments_payments))
        data = data_temp.join(temp_13, how='left', on='SK_ID_CURR')
        del temp_13
        gc.collect()

        temp_14 = initial_function_definition.reduce_memory_usage(installments_payments[installments_payments['DAYS_INSTALMENT']>=-365].reset_index())
        temp_14.drop(['index'], axis=1, inplace=True)
        temp_14 = initial_function_definition.FE_installments_payments(temp_14)
        data = data.join(temp_14, how='left', on='SK_ID_CURR', rsuffix='_year')
        del temp_14
        gc.collect()

        temp_15 = initial_function_definition.reduce_memory_usage(installments_payments[installments_payments['DAYS_INSTALMENT']>=-182].reset_index())
        temp_15.drop(['index'], axis=1, inplace=True)
        temp_15 = initial_function_definition.FE_installments_payments(temp_15)
        data = data.join(temp_15, how='left', on='SK_ID_CURR', rsuffix='_half_year')
        del temp_15
        gc.collect()

        temp_16 = initial_function_definition.reduce_memory_usage(installments_payments[installments_payments['DAYS_INSTALMENT']>=-90].reset_index())
        temp_16.drop(['index'], axis=1, inplace=True)
        temp_16 = initial_function_definition.FE_installments_payments(temp_16)
        data = data.join(temp_16, how='left', on='SK_ID_CURR', rsuffix='_quarter')
        del temp_16
        gc.collect()

        temp_17 = initial_function_definition.reduce_memory_usage(installments_payments[installments_payments['DAYS_INSTALMENT']>=-30].reset_index())
        temp_17.drop(['index'], axis=1, inplace=True)
        temp_17 = initial_function_definition.FE_installments_payments(temp_17)
        data = data.join(temp_17, how='left', on='SK_ID_CURR', rsuffix='_month')
        del temp_17
        gc.collect()
        temp_18 = initial_function_definition.reduce_memory_usage(installments_payments[installments_payments['DAYS_INSTALMENT']>=-14].reset_index())
        temp_18.drop(['index'], axis=1, inplace=True)
        temp_18 = initial_function_definition.FE_installments_payments(temp_18)
        data = data.join(temp_18, how='left', on='SK_ID_CURR', rsuffix='_fortnight')
        del temp_18
        gc.collect()

        temp_19 = initial_function_definition.reduce_memory_usage(installments_payments[installments_payments['DAYS_INSTALMENT']>=-7].reset_index())
        temp_19.drop(['index'], axis=1, inplace=True)
        temp_19 = initial_function_definition.FE_installments_payments(temp_19)
        data = data.join(temp_19, how='left', on='SK_ID_CURR', rsuffix='_week')
        del temp_19
        gc.collect()

        return data


    def FE_credit_card_balance(credit_card_balance):

        cc_balance_data, cc_balance_columns = initial_function_definition.one_hot_encode(credit_card_balance)
        cc_balance_data.rename(columns={'AMT_RECIVABLE': 'AMT_RECEIVABLE'}, inplace=True)

        cc_balance_data['LIMIT_USE'] = cc_balance_data['AMT_BALANCE'] / cc_balance_data['AMT_CREDIT_LIMIT_ACTUAL']
        cc_balance_data['PAYMENT_DIV_MIN'] = cc_balance_data['AMT_PAYMENT_CURRENT'] / cc_balance_data['AMT_INST_MIN_REGULARITY']
        cc_balance_data['LATE_PAYMENT'] = cc_balance_data['SK_DPD'].apply(lambda x: 1 if x > 0 else 0)
        cc_balance_data['DRAWING_LIMIT_RATIO'] = cc_balance_data['AMT_D
RAWINGS_ATM_CURRENT'] / cc_balance_data['AMT_CREDIT_LIMIT_ACTUAL']

        cc_balance_data.drop(['SK_ID_PREV'], axis= 1, inplace = True)
        cc_balance_data_agg = cc_balance_data.groupby('SK_ID_CURR').agg
(['max', 'mean', 'sum', 'var'])

        col_list_9=[]

        for col in cc_balance_data_agg.columns.tolist():
            col_list_9.append('CR_'+col[0]+'_'+col[1].upper())

        cc_balance_data_agg.columns = pd.Index(col_list_9)

        cc_balance_data_agg['CREDIT_COUNT'] = cc_balance_data.groupby(
'SK_ID_CURR').size()

        del cc_balance_data, cc_balance_columns
        gc.collect()

        return cc_balance_data_agg



    def FE_credit_card_balance_months_balance(data,data_temp,credit_car
d_balance):

        temp_20 = initial_function_definition.FE_credit_card_balance(in
itial_function_definition.reduce_memory_usage(credit_card_balance))
        data = data_temp.join(temp_20, how='left', on='SK_ID_CURR')
        del temp_20
        gc.collect()

        temp_21 = initial_function_definition.reduce_memory_usage(credi
t_card_balance[credit_card_balance['MONTHS_BALANCE']>=-12].reset_index
())
        temp_21.drop(['index'], axis=1, inplace=True)
        temp_21 = initial_function_definition.FE_credit_card_balance(te
mp_21)
        data = data.join(temp_21, how='left', on='SK_ID_CURR', rsuffix=
'_year')
        del temp_21
        gc.collect()

        temp_22 = initial_function_definition.reduce_memory_usage(credit_card_balance[credit_card_balance['MONTHS_BALANCE']>=-6].reset_index())
        temp_22.drop(['index'], axis=1, inplace=True)
        temp_22 = initial_function_definition.FE_credit_card_balance(temp_22)
        data = data.join(temp_22, how='left', on='SK_ID_CURR', rsuffix='_half_year')
        del temp_22
        gc.collect()

        temp_23 = initial_function_definition.reduce_memory_usage(credit_card_balance[credit_card_balance['MONTHS_BALANCE']>=-3].reset_index())
        temp_23.drop(['index'], axis=1, inplace=True)
        temp_23 = initial_function_definition.FE_credit_card_balance(temp_23)
        data = data.join(temp_23, how='left', on='SK_ID_CURR', rsuffix='_quarter')
        del temp_23
        gc.collect()

        temp_24 = initial_function_definition.reduce_memory_usage(credit_card_balance[credit_card_balance['MONTHS_BALANCE']>=-1].reset_index())
        temp_24.drop(['index'], axis=1, inplace=True)
        temp_24 = initial_function_definition.FE_credit_card_balance(temp_24)
        data = data.join(temp_24, how='left', on='SK_ID_CURR', rsuffix='_month')
        del temp_24
        gc.collect()

        return data
```

3. Computing the Probabilities on the Test Dataset

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")
import os
import os.path
import sqlite3
import flask

from flask import Flask, jsonify, request
from lightgbm import LGBMClassifier
from sqlalchemy import create_engine
from hcdr_model import initial_function_definition

if os.path.isfile('pickles/test_data')==False:

    train_data = initial_function_definition.reduce_memory_usage(pd.read_csv('home-credit-default-risk/application_train.csv'))
    test_data = initial_function_definition.reduce_memory_usage(pd.read_csv('home-credit-default-risk/application_test.csv'))
    bureau_data = initial_function_definition.reduce_memory_usage(pd.read_csv('home-credit-default-risk/bureau.csv'))
    bureau_balance = initial_function_definition.reduce_memory_usage(pd.read_csv('home-credit-default-risk/bureau_balance.csv'))

    bureau_data_fe = initial_function_definition.FE_bureau_data_1(bureau_data)

    #One Hot Encoding the Bureau Datasets
    bureau_data, bureau_data_columns = initial_function_definition.one_hot_encode(bureau_data_fe)
    bureau_balance, bureau_balance_columns = initial_function_definition.one_hot_encode(bureau_balance)

    bureau_data_balance_final = initial_function_definition.FE_bureau_data_2(bureau_data, bureau_balance,bureau_data_columns,bureau_balance_columns)

    previous_application = initial_function_definition.reduce_memory_us
```

```
age(pd.read_csv('home-credit-default-risk/previous_application.csv'))
    previous_application = initial_function_definition.preprocess_previ
ous_application(previous_application)

    pos_cash_balance = initial_function_definition.reduce_memory_usage(
pd.read_csv('home-credit-default-risk/POS_CASH_balance.csv'))
    installments_payments = initial_function_definition.reduce_memory_u
sage(pd.read_csv('home-credit-default-risk/installments_payments.csv'))
    credit_card_balance = initial_function_definition.reduce_memory_usa
ge(pd.read_csv('home-credit-default-risk/credit_card_balance.csv'))


    start = datetime.now()

    test_data = initial_function_definition.fix_nulls_outliers(test_dat
a)
    test_data_temp_1 = initial_function_definition.FE_application_data(
test_data)
    bureau_data_balance_final = initial_function_definition.FE_bureau_d
ata_2(bureau_data, bureau_balance,bureau_data_columns,bureau_balance_co
lumns)
    test_data_temp_2 = test_data_temp_1.join(bureau_data_balance_final,
how='left', on='SK_ID_CURR')


    test_data_temp_2 = initial_function_definition.FE_previous_applicat
ion_days_decision(test_data,test_data_temp_2,previous_application)
    test_data_temp_2 = initial_function_definition.FE_pos_cash_balance_
months_balance(test_data,test_data_temp_2, pos_cash_balance)
    test_data_temp_2 = initial_function_definition.FE_installments_paym
ents_days_instalment(test_data,test_data_temp_2,installments_payments)
    test_data_temp_2 = initial_function_definition.FE_credit_card_balan
ce_months_balance(test_data,test_data_mod_temp_2,credit_card_balance)

    #Removing any duplicate features, if any are present in the final d
ataset
    test_data = test_data_temp_2.loc[:,~test_data_temp_2.columns.duplic
ated()]
```

```
    print("Time taken to run this cell :", datetime.now() - start)

else:

    test_data = pd.read_pickle('pickles/test_data')


features_top_df_train = pd.read_pickle('pickles/features_top_df_train.pkl')
features_top_df_test = test_data[features_top_df_train.columns]
features_top_df_test['SK_ID_CURR'] = test_data['SK_ID_CURR']
features_top_df_test['TARGET'] = np.nan


app = Flask(__name__)

#home page
@app.route('/', methods = [])
def hello_world():
    return 'Hello World!'


#prediction page
@app.route('/index')
def index():
    return flask.render_template('index.html')


#results page
@app.route('/predict', methods = ['POST'])
def predict():

    conn = sqlite3.connect('Home_Credit_DB_Connection.db')
    sk_id_curr = request.form.to_dict()['SK_ID_CURR']
    sk_id_curr = int(sk_id_curr)

    test_datapoint = pd.read_sql_query(f'SELECT * FROM test_data_feats WHERE SK_ID_CURR == {sk_id_curr}', conn)
    test_datapoint = test_datapoint.replace([None], np.nan)
    with open('lgbm/lgbm_model_500f_3.pickle','rb') as f:
        lgbm_model = pickle.load(f)

    if os.path.isfile('lgbm/lgbm_best_threshold_500f_api.pkl')==False:

        feats = [f for f in features_top_df_train.columns if f not in ['TARGET','SK_ID_CURR','SK_ID_BUREAU','SK_ID_PREV','index']]
        test_predict = np.zeros(features_top_df_test.shape[0])
        test_predict += lgbm_model.predict_proba(features_top_df_test[feats], num_iteration=lgbm_model.best_iteration_)[:, 1] / 5
    else:

        with open('lgbm/lgbm_test_predict_500f.pkl','rb') as f:
            test_predict = pickle.load(f)

    threshold = 0.3741018248484985

    test_predict_rounded = np.round(test_predict,4)
    predicted_class_label = np.where(test_predict_rounded > threshold, 1, 0)

    select_index = list(np.where(test_data["SK_ID_CURR"] == sk_id_curr)[0])
    final_class_label = predicted_class_label[select_index[0]]
    final_test_predict_rounded = test_predict_rounded[select_index[0]]

    if final_class_label == 1:
        prediction = 'The customer with this ID is a Potential Defaulter with a probability of {}'.format(final_test_predict_rounded)
    else:
        prediction = 'The customer with this ID is not a Potential Defaulter with a probability of {}'.format(final_test_predict_rounded)
        predicted_proba = 1 - final_test_predict_rounded

    return jsonify({'prediction': prediction})


if __name__ == '__main__':
    app.debug=True
    app.run(host='0.0.0.0', port=8080)
```