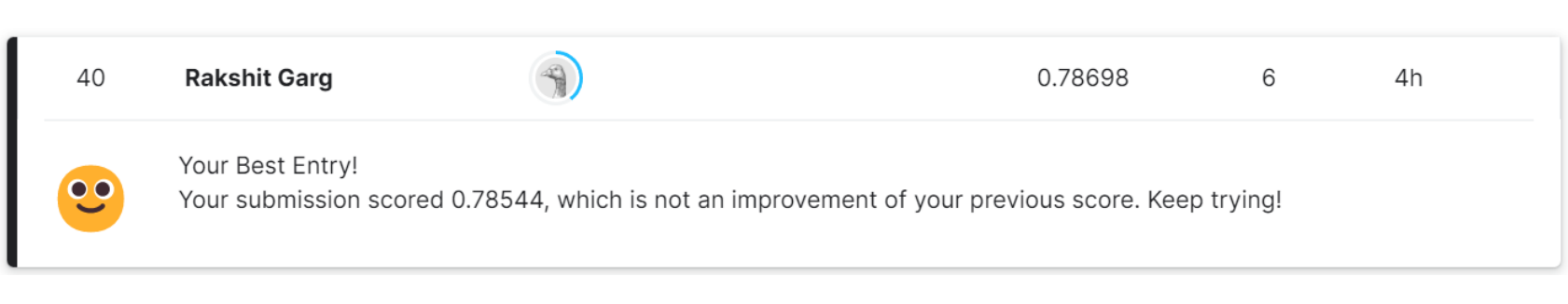
Rakshit_102083022_CO28

Lab Evaluation

- Problem Name- Netflix Appetency
- Problem Link- https://www.kaggle.com/c/netflix-appetency/overview
- Problem Type- Classification
- GitHub Link- https://github.com/rakshitgarg99/Rapids_data_science/tree/main/Lab_Evaluation
- Kaggle Link- https://www.kaggle.com/rakshitgarg99/netflix-classification/edit
- Libraries used- cuPy, cuDF, cuML, sklearn, pandas, matplotlib, seaborn
- Models Implemented- Logistic Regression, Random Forest, Xgboost, LGBM
- Evaluation Metrics Used- AUC
- Kaggle Rank Achieved with total number of teams- 40th Rank out of 147 (On-going contest)
- Tasks done in code-

1. Data Preprocessing (EDA, Handling null values, High cardinality features, Numerical and categorical features, Temporal or datetime features and coorelation)
2. Model building and evaluation(AUC)
3. Detailed Coding documentationm

- Proof of Rank-



Netflix Appetency - Classify consumers according to their appetite to subscribe to Netflix. More information about this contest:-

1. The metric used is AUC
2. The competition will last 2 months until 2022-03-31
3. The test file is split into 2 parts, 20% for the public classification and 80% for the private classification.

Important - As it's an ongoing contest and only 20% of data is used in the evaluation. We cannot implement a more optimized model because of the fear of overfitting. I am attaching some threads of the discussion section; please go through them. https://www.kaggle.com/c/netflix-appetency/discussion/303029 https://www.kaggle.com/c/netflix-appetency/discussion/304284

With the above information from the host and discussion section, I have done the assignment considering the best score will be 0.7878 with a variance of +/-0.0024

In [94]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input direc

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

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you c
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Importing libraries

In [95]:
```
import cupy as np
import cudf as pd
import matplotlib.pylab as plt
import seaborn as sns
```

Getting Input Files

In [96]:
```
train_df = pd.read_csv("../input/netflix-appetency/train.csv")
test_df = pd.read_csv("../input/netflix-appetency/test.csv")
ss_df = pd.read_csv("../input/netflix-appetency/sample_submission.csv")

train_df.shape, test_df.shape, ss_df.shape
```

EDA - Expolratory Data Analysis

In [97]:
```
sns.countplot(train_df.target.to_array())
plt.show()
```

In [98]:
```
train_df.head()
```

We can observe that lots for featues have `NaN` values

In [100…
```
train_df.isna().sum(axis=0).sort_values(ascending=False).to_frame().T
```

we can see that some features have all null values, we can drop those

In [101…
```
null_columns = ['feature_193', 'feature_196', 'feature_197', 'feature_198', 'feature_83']
train_df = train_df.drop(columns=null_columns)
test_df = test_df.drop(columns=null_columns)

train_df.shape, test_df.shape
```

In [102…
```
train_df.info()
```

Categorical Colummns

In [104…
```
train_df.select_dtypes('object').head()
```

- we can observe that those are catgorical features encoded with some labels
- we can also observe that some date features, we can drop those as of now

In [107…
```
## getting date columns
cat_columns = [col for col in train_df.select_dtypes('object').columns]
mask = train_df[cat_columns][:10].to_pandas().astype(str).apply(lambda x : x.str.match('(\d{2,4}(-|\/|\\|\.| )\
```

In [134…
```
mask_idx = list(mask.index)
for i in range(len(mask)):
    if mask[i]==True:
        print(mask_idx[i])
```

In [74]:
```
date_columns = ['feature_191', 'feature_192', 'feature_194', 'feature_195', 'feature_199', 'feature_200', 'feat
display(train_df[date_columns].head())

train_df = train_df.drop(columns=date_columns)
test_df = test_df.drop(columns=date_columns)

train_df.shape, test_df.shape
```

In [75]:
```
# Number of unique categories
train_df.to_pandas().select_dtypes('object').nunique().sort_values().to_frame().T
```

- we can see that some featues have only single categorie, so we can drop those featues

In [77]:
```
single_feature_columns = ['feature_190', 'feature_55', 'feature_187', 'feature_188', 'feature_249', 'feature_24

train_df = train_df.drop(columns=single_feature_columns)
test_df = test_df.drop(columns=single_feature_columns)

train_df.shape, test_df.shape
```

In [105…
```
cat_columns = [col for col in train_df.select_dtypes('object').columns]
train_df[cat_columns].head()
```

Numerical Columns

In [80]:
```
train_df.select_dtypes(['int', 'float']).head()
```

- we can observe that nan values also present in numarical columns

In [81]:
```
target_column = 'target'
drop_columns = ['id', target_column]

num_columns = [col for col in train_df.select_dtypes(['int', 'float']).columns if col not in drop_columns]
train_df[num_columns].head()
```

In [82]:
```
# Number of unique values
train_df[num_columns].to_pandas().nunique().sort_values().to_frame().T #@.columns[30:45]
```

In [24]:
```
single_feature_columns = [
       'feature_461', 'feature_474', 'feature_472', 'feature_409',
       'feature_469', 'feature_467', 'feature_411', 'feature_465',
       'feature_312', 'feature_463', 'feature_313', 'feature_375',
       'feature_427', 'feature_412', 'feature_413', 'feature_455',
       'feature_453', 'feature_419', 'feature_451', 'feature_100',
       'feature_449', 'feature_421', 'feature_445', 'feature_433',
       'feature_432', 'feature_459', 'feature_476', 'feature_405',
       'feature_425', 'feature_498', 'feature_227', 'feature_228',
       'feature_149', 'feature_495', 'feature_385', 'feature_493',
       'feature_478', 'feature_491', 'feature_389', 'feature_489',
       'feature_252', 'feature_487', 'feature_502', 'feature_505',
       'feature_484', 'feature_395', 'feature_482', 'feature_377',
       'feature_358', 'feature_480', 'feature_397', 'feature_357',
       'feature_500']

train_df = train_df.drop(columns=single_feature_columns)
test_df = test_df.drop(columns=single_feature_columns)

train_df.shape, test_df.shape
```

In [84]:
```
num_columns = [col for col in train_df.select_dtypes(['int', 'float']).columns if col not in drop_columns]
```

Handling NAN values

- using median for numerical features
- whereas mode for categorical features

In [85]:
```
for col in num_columns:
    train_df[col].fillna(train_df[col].median(),inplace=True)
    test_df[col].fillna(test_df[col].median(),inplace=True)

def get_mode(feature,df):
    return df[feature].mode()[0]

for col in cat_columns:
    train_df[col].fillna(get_mode(col,train_df),inplace=True)
    test_df[col].fillna(get_mode(col,test_df),inplace=True)
```

- checking correlation of numerical features and target

In [86]:
```
corr_df = train_df[['target'] + num_columns].corr()
corr_df.head()
```

- Label Encoding - encode the categorical features

In [88]:
```
from cuml.preprocessing.LabelEncoder import LabelEncoder
```

In [89]:
```
# label encoding those cat columns
def label_encoding(train_df, test_df, columns):
    for col in columns:
        le = LabelEncoder()
        values = train_df[col].append(test_df[col])
        le.fit(values)
        train_df[col] = le.transform(train_df[col])
        test_df[col] = le.transform(test_df[col])
    return train_df, test_df

train_df, test_df = label_encoding(train_df, test_df, cat_columns)
train_df[cat_columns].head()
```

In [90]:
```
target_col = 'target'
train_cols = cat_columns + num_columns
```

Modeling

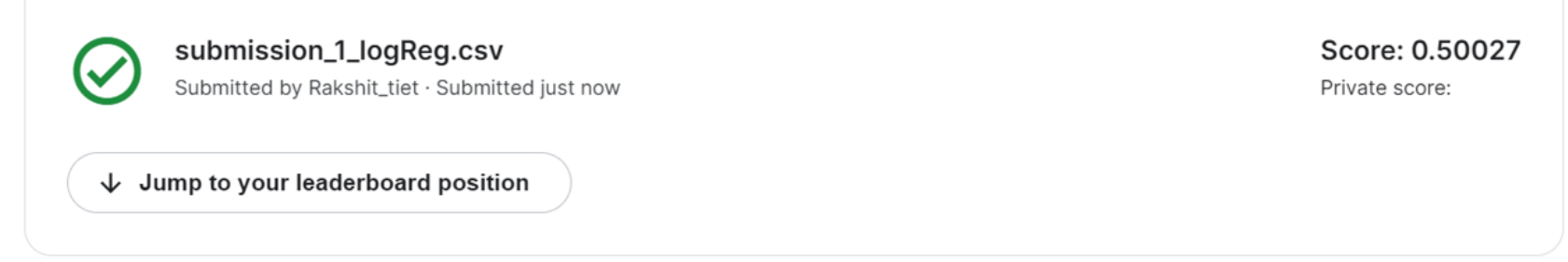
- Logistic Regression

In [91]:
```
from cuml.linear_model import LogisticRegression
```

In [92]:
```
reg = LogisticRegression()
reg.fit(train_df[train_cols].astype('float32'),train_df[target_col].astype('float32'))
y_pred = reg.predict_proba(test_df[train_cols].astype('float32'))
```

In [59]:
```
ss_df['target'] = y_pred[1]
ss_df.to_csv("submission_1_logReg.csv", index=False)
ss_df.head()
```

Score of Logistic Regression



- Random Forest Classifier

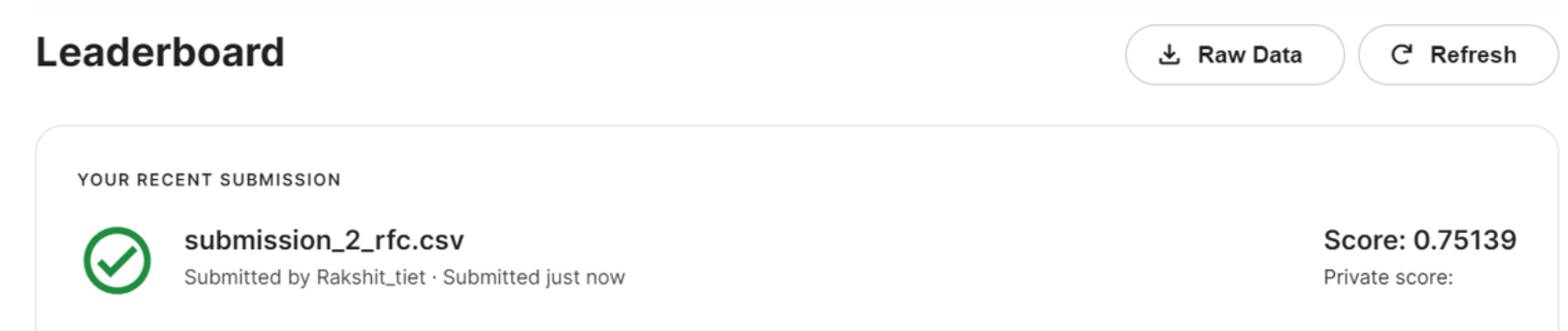
In [43]:
```
from cuml.ensemble import RandomForestClassifier as cuRFC
```

In [52]:
```
cuml_model = cuRFC(n_bins=156,
                   n_estimators=1450,random_state=42, max_depth=5, split_criterion='gini')
cuml_model.fit(train_df[train_cols].astype('float32'),train_df[target_col].astype('float32'))
y_predict = cuml_model.predict_proba(test_df[train_cols].astype('float32'))
```

In [55]:
```
y_predict[1]
```

In [56]:
```
ss_df['target'] = y_predict[1]
ss_df.to_csv("submission_2_rfc.csv", index=False)
ss_df.head()
```

Score of Random Forest Classifier

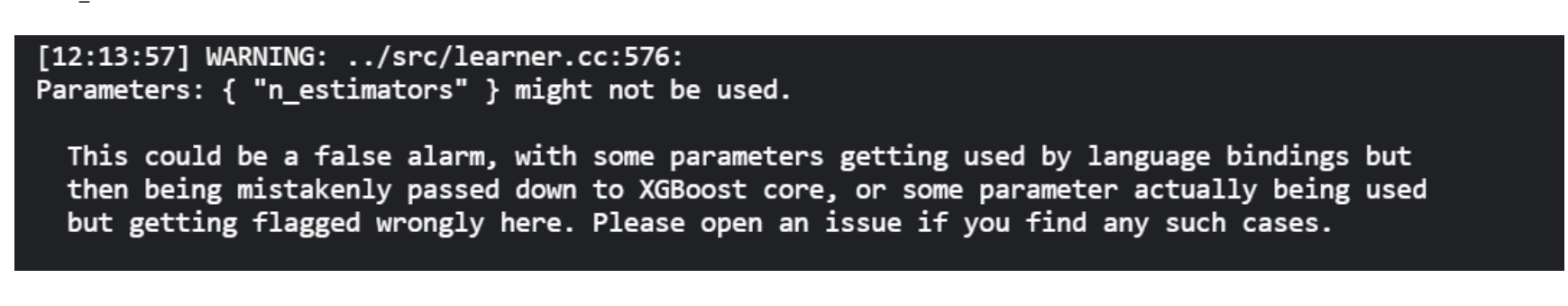


- XGB

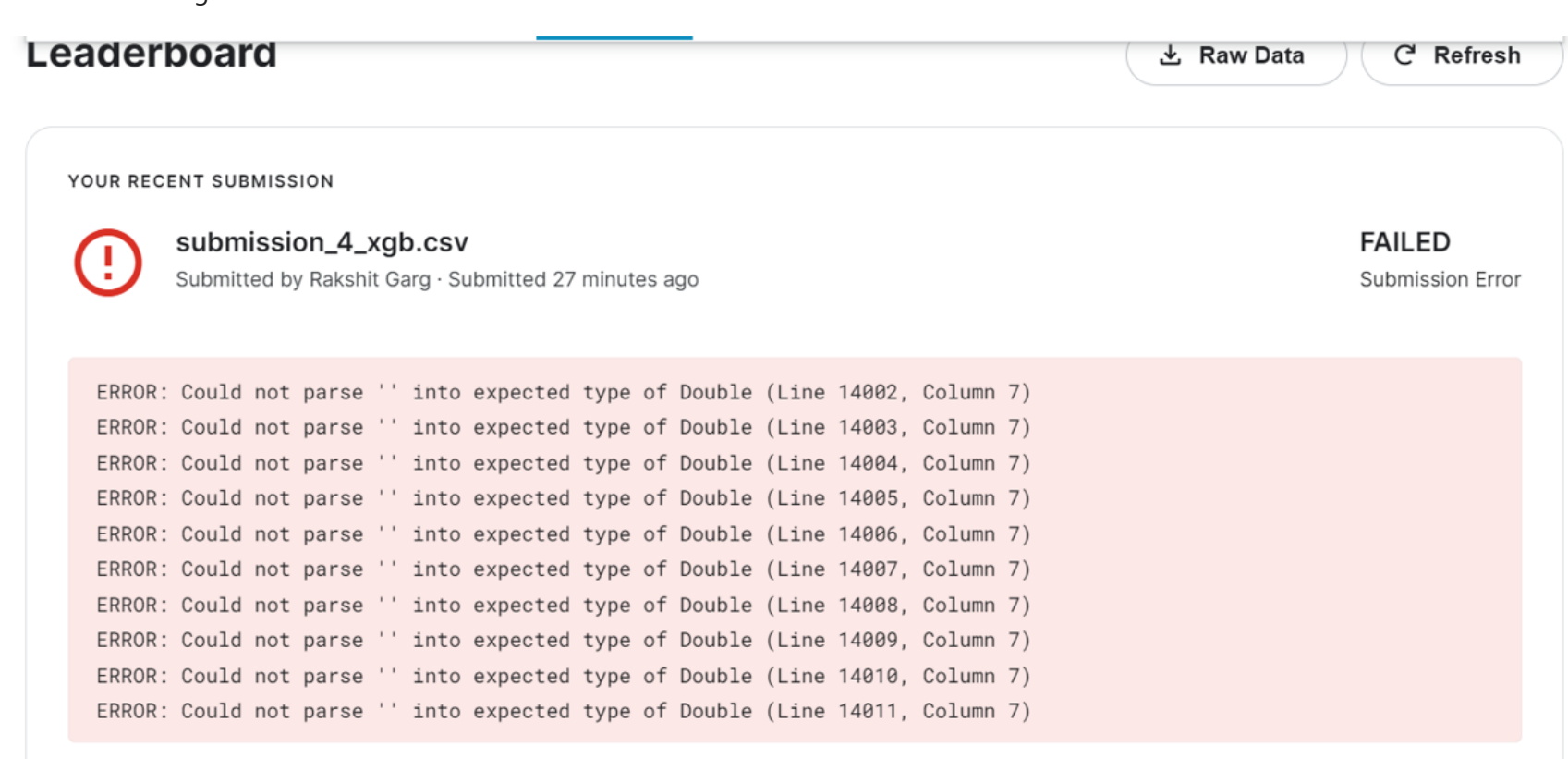
Below is the implementation for cuML based XGBClassifer. But I am getting a error while submitting, Moreover the cuML based XGB didn't make us used the "n_esimators" parameters. I tried to find a solution on https://docs.rapids.ai/api/cuml/stable/api.html#forest-inferencing . But didn't get any suitable way to solve problem.

Attaching screenshots regarding the same.

- n_estimators not used



- Failure during submission



In [28]:
```
# # from xgboost import XGBClassifier
# import xgboost as xgb
# from cuml import ForestInference
# from cuml.test.utils import array_equal
# from cuml.model_selection import train_test_split


# def train_xgboost_model(X_train, y_train,
#                           num_rounds, model_path):
#     # set the xgboost model parameters
#     params = { 'n_estimators': 14140,
#     'learning_rate': 0.011,
#     'max_depth': 4,
#     'reg_alpha': 2.97,
#     'reg_lambda': 2.214,
#     'scale_pos_weight': 0.9,
#     'subsample': 0.776,
#     'gamma': 0.5}
#     dtrain = xgb.DMatrix(X_train, label=y_train)
#     # train the xgboost model
#     bst = xgb.train(params, dtrain, num_rounds)

#     # save the trained xgboost model
#     bst.save_model(model_path)

#     return bst

# def predict_xgboost_model(X_validation, y_validation, xgb_model):

#     # predict using the xgboost model
#     dvalidation = xgb.DMatrix(X_validation, label=y_validation)
#     xgb_preds = xgb_model.predict(dvalidation)

#     # convert the predicted values from xgboost into class labels
#     xgb_preds = np.around(xgb_preds)
#     return xgb_preds

# model_path = 'xgb.model'

# # num of iterations for which the model is trained
# num_rounds = 15

# # xgboost_model = train_xgboost_model(X_train, y_train,
# #                                      num_rounds, model_path)
# # fil_predsxgb = predict_xgboost_model(X_valid,y_valid,xgboost_model)


# trained_model_preds = predict_xgboost_model(X_valid,
#                                             y_valid,
#                                             xgboost_model)

# fm = ForestInference.load(filename=model_path,
#                           algo='BATCH_TREE_REORG',
#                           output_class=True,
#                           threshold=0.50,
#                           model_type='xgboost')

# test_predictions = fm.predict(X_valid)
```

Because of the above problems I decided to implement the xgb and lgbm in sklearn to score a more.

- XGB SKLearn

In [51]:
```
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn import metrics
import lightgbm as lgb
```

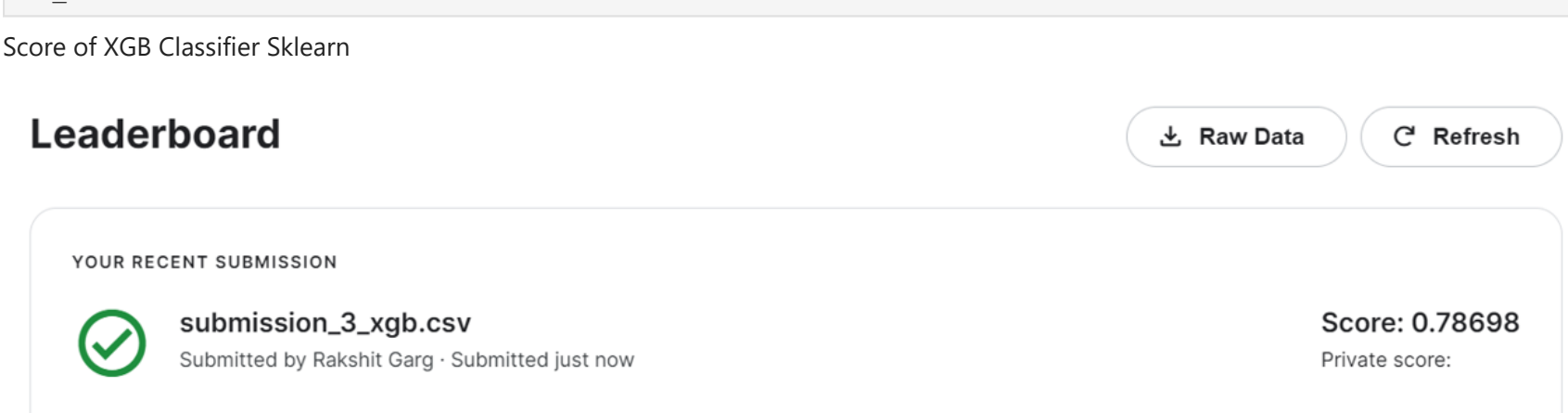
In [30]:
```
train_df = train_df.to_pandas()
test_df = test_df.to_pandas()
```

- using KFold to get best model split

In [31]:
```
skf = model_selection.StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
test_predictions = []

oof_predictions_proba = np.zeros(len(train_df))

X_test = test_df[train_cols]

for idx, (train_idx, valid_idx) in enumerate(skf.split(train_df, train_df[target_col])):

    print("="*100)
    print("FOLD : ", idx)
    print("_"*100)

    X_train = train_df.iloc[train_idx][train_cols]
    y_train = train_df.iloc[train_idx][target_col]

    X_valid = train_df.iloc[valid_idx][train_cols]
    y_valid = train_df.iloc[valid_idx][target_col]

    print("Trian :", X_train.shape, y_train.shape)
    print("Valid :", X_valid.shape, y_valid.shape)

    XGB_params = {
   'n_estimators': 14140,
    'learning_rate': 0.0111538499996725174,
    'max_depth': 4,
    'reg_alpha': 2.9735993984217104,
    'reg_lambda': 2.214625977393439,
    'scale_pos_weight': 0.9,
    'subsample': 0.7768428006197691,
    'gamma': 0.5
    }
#XGBoost
    model = XGBClassifier(**XGB_params)
    model.fit(X_train, y_train, eval_set=[(X_valid, y_valid)],
            early_stopping_rounds=200,
            eval_metric='auc',
            verbose=100)

    y_valid_pred = model.predict_proba(X_valid)[:, 1]
    predictions = model.predict_proba(X_test)[:, 1]

    test_predictions.append(predictions)
    oof_predictions_proba[valid_idx] = y_valid_pred

score = metrics.roc_auc_score(train_df[target_col], np.asnumpy(oof_predictions_proba))
test_predictions = np.mean(np.asnumpy(test_predictions), axis=0)

print("OOF MEAN AUC :", score)
```

In [ ]:
```
ss_df['target'] = test_predictions
ss_df.to_csv("submission_3_xgb.csv", index=False)
ss_df.head()
```

Score of XGB Classifier Sklearn

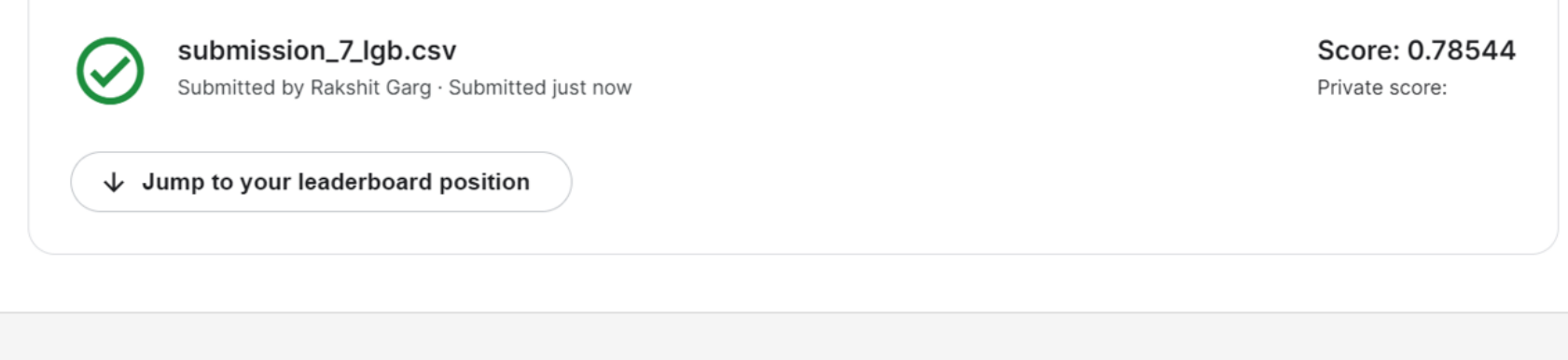


- LGBM Classifier
- using KFold to get best model split

In [52]:
```
skf = model_selection.StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
test_predictions = []

oof_predictions_proba = np.zeros(len(train_df))

X_test = test_df[train_cols]

for idx, (train_idx, valid_idx) in enumerate(skf.split(train_df, train_df[target_col])):

    print("="*100)
    print("FOLD : ", idx)
    print("_"*100)

    X_train = train_df.iloc[train_idx][train_cols]
    y_train = train_df.iloc[train_idx][target_col]

    X_valid = train_df.iloc[valid_idx][train_cols]
    y_valid = train_df.iloc[valid_idx][target_col]

    print("Train :", X_train.shape, y_train.shape)
    print("Valid :", X_valid.shape, y_valid.shape)

    params = {

        'boosting_type': 'gbdt',
        'tree_learner': 'feature', #''serial' or feature' or 'data' or 'voting'
        'num_leaves': 31,
        'max_depth': -1,
        'learning_rate': 5e-2,
        'n_estimators': 10000,
        'importance_type': 'gain',
        'subsample_for_bin': 200000,
        'objective': 'binary',
        'min_split_gain': 0.0,
        'min_child_weight': 1e-3,
        'min_child_samples': 20,
        #'bagging_freq': 0,
        #'bagging_fraction': 1.0,
        #'feature_fraction': 1.0,
        'reg_alpha': 0.2,
        'reg_lambda': 0.2,
        'random_state': 42,
        'n_jobs': -1,
    }

    model = lgb.LGBMClassifier(**params)
    model.fit(X_train, y_train, eval_set=[(X_valid, y_valid)],
            early_stopping_rounds=200,
            eval_metric='auc',
            verbose=100)

    y_valid_pred = model.predict_proba(X_valid)[:, 1]
    predictions = model.predict_proba(X_test)[:, 1]

    test_predictions.append(predictions)
    oof_predictions_proba[valid_idx] = y_valid_pred

score = metrics.roc_auc_score(train_df[target_col], np.asnumpy(oof_predictions_proba))
test_predictions = np.mean(np.asnumpy(test_predictions), axis=0)

print("OOF MEAN AUC :", score)
```

In [55]:
```
ss_df['target'] = test_predictions
ss_df.to_csv("submission_4_lgb.csv", index=False)
ss_df.head()
```

Score of LGBM Classifier Sklearn



In [ ]: