In [52]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
gc.collect()
from tqdm import tqdm
import datetime
import joblib
from sklearn.metrics import mean_squared_error
```

In [53]:
```
#https://www.kaggle.com/artgor/elo-eda-and-models
def reduce_mem_usage(df, verbose=True):
    '''
    This is done as there are lot of historical data, which requires lot of RAM.
    This method  tries to reduce the size of data, it works on only numeric data
    in which it can be represented.
    '''
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            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.iinfo(np.int8).ma
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.flo
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.fo
    return df
```

In [54]:
```
def std(x):
    return np.std(x)
```

In [55]:
```
#https://www.geeksforgeeks.org/python-pandas-series-dt-date/
def getFeaturesFromTrainAndTest(data):

    max_dte = data['first_active_month'].dt.date.max()

    #Time elapsed since first purchase
    data['time_elapsed'] = (max_dte - data['first_active_month'].dt.date).dt.days

    #Breaking first_active_month in year and month
    data['month'] = data['first_active_month'].dt.month
    data['year'] = data['first_active_month'].dt.year
    data['day'] = data['first_active_month'].dt.day

    return data
```

In [121]:
```
#https://www.kaggle.com/artgor/elo-eda-and-models
def getFeaturesFromTransactionData(data, prefix):

    #Breaking purchase date into year,month, day
    data['purchase_year'] = data['purchase_date'].dt.year
    data['purchase_month'] = data['purchase_date'].dt.month
    data['purchase_day'] = data['purchase_date'].dt.day

    data['month_diff'] = ((datetime.datetime.today() - data['purchase_date']).dt.
    data['month_diff'] += data['month_lag']

    data['weekend'] = (data.purchase_date.dt.weekday >=5).astype(int)
    data['hour'] = data['purchase_date'].dt.hour

    category2Unique = ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0']
    category3Unique = ['1', '2', '3']

    #Converting category_2 and category_3 into indicator variables
    #https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummie
    data = pd.get_dummies(data, columns=['category_2', 'category_3'])

    #this is done to handle missing categorical values incase of test data
    for i in range(len(category2Unique)):
        name = "category_2_" + str(category2Unique[i])
        if name not in data.columns:
            data[name] = 0

    for i in range(len(category3Unique)):
        name = "category_3_" + str(category3Unique[i])
        if name not in data.columns:
            data[name] = 0

    agg_func = {
        'authorized_flag': ['sum', 'mean'],
        'category_1': ['sum', 'mean'],
        'category_2_1.0': ['mean', 'sum'],
        'category_2_2.0': ['mean', 'sum'],
        'category_2_3.0': ['mean', 'sum'],
        'category_2_4.0': ['mean', 'sum'],
        'category_2_5.0': ['mean', 'sum'],
        'category_3_1': ['sum', 'mean'],
        'category_3_2': ['sum', 'mean'],
        'category_3_3': ['sum', 'mean'],
        'merchant_id': ['nunique'],
        'purchase_amount': ['sum', 'mean', 'max', 'min', std],
        'installments': ['sum', 'mean', 'max', 'min', std],
        'purchase_month': ['mean', 'max', 'min', std],
        'purchase_year': ['mean', 'max', 'min', std],
        'purchase_day': ['mean', 'max', 'min', std],
        'month_lag': ['min', 'max'],
        'merchant_category_id': ['nunique'],
        'state_id': ['nunique'],
        'subsector_id': ['nunique'],
        'city_id': ['nunique'],
        'month_diff': ['min', 'max', 'mean']
    }
```

```
    agg_trans = data.groupby(['card_id']).agg(agg_func)
    agg_trans.columns = [prefix + '_'.join(col).strip() for col in agg_trans.colu
    agg_trans.reset_index(inplace=True)

    df = (data.groupby('card_id')
          .size()
          .reset_index(name='{}transactions_count'.format(prefix)))

    agg_trans = pd.merge(df, agg_trans, on='card_id', how='left')

    return agg_trans
```

In [122]:

```
def aggregate_per_month(history):
    grouped = history.groupby(['card_id', 'month_lag'])
    history['installments'] = history['installments'].astype(int)
    agg_func = {
            'purchase_amount': ['count', 'sum', 'mean', 'min', 'max', std],
            'installments': ['count', 'sum', 'mean', 'min', 'max', std],
            }

    intermediate_group = grouped.agg(agg_func)
    intermediate_group.columns = ['_'.join(col).strip() for col in intermediate_g
    intermediate_group.reset_index(inplace=True)

    final_group = intermediate_group.groupby('card_id').agg(['mean', std])
    final_group.columns = ['_'.join(col).strip() for col in final_group.columns.v
    final_group.reset_index(inplace=True)

    return final_group
```

In [123]:
```
def getFeaturesFromMerchantsData(data, prefix):

    salesUnique = ['1', '2', '3', '4', '5']
    purchasesUnique = ['1', '2', '3', '4', '5']

    data = pd.get_dummies(data, columns=['most_recent_sales_range', 'most_recent_

     #this is done to handle missing categorical values incase of test data
    for i in range(len(salesUnique)):
        name = "most_recent_sales_range_" + str(salesUnique[i])
        if name not in data.columns:
            data[name] = 0

    for i in range(len(purchasesUnique)):
        name = "most_recent_purchases_range_" + str(purchasesUnique[i])
        if name not in data.columns:
            data[name] = 0

    agg_func = {
        'merchant_group_id' : ['nunique'],
        'numerical_1' :['sum', 'mean', std],
        'numerical_2' :['sum', 'mean', std],
        'category_4' :['sum', 'mean', std],
        'most_recent_sales_range_1' :['sum', 'mean', std],
        'most_recent_sales_range_2' :['sum', 'mean', std],
        'most_recent_sales_range_3' :['sum', 'mean', std],
        'most_recent_sales_range_4' :['sum', 'mean', std],
        'most_recent_sales_range_5' :['sum', 'mean', std],
        'most_recent_purchases_range_1' :['sum', 'mean', std],
        'most_recent_purchases_range_2' :['sum', 'mean', std],
        'most_recent_purchases_range_3' :['sum', 'mean', std],
        'most_recent_purchases_range_4' :['sum', 'mean', std],
        'most_recent_purchases_range_5' :['sum', 'mean', std],
        'avg_sales_lag3' :['sum', 'mean', std],
        'avg_purchases_lag3' :['sum', 'mean', std],
        'active_months_lag3' :['sum', 'mean', std],
        'avg_sales_lag6' :['sum', 'mean', std],
        'avg_purchases_lag6' :['sum', 'mean', std],
        'active_months_lag6' :['sum', 'mean', std],
        'avg_sales_lag12' :['sum', 'mean', std],
        'avg_purchases_lag12' :['sum', 'mean', std],
        'active_months_lag12' :['sum', 'mean', std],
    }

    agg_trans = data.groupby(['card_id']).agg(agg_func)
    agg_trans.columns = [prefix + '_'.join(col).strip() for col in agg_trans.colu
    agg_trans.reset_index(inplace=True)

    df = (data.groupby('card_id')
          .size()
          .reset_index(name='{}transactions_count'.format(prefix)))

    agg_trans = pd.merge(df, agg_trans, on='card_id', how='left')

    return agg_trans
```

In [124]:
```
def getAllTheFeatures(data, df_train, df_hist, df_newTrans, df_merchants):

    card_id = data['card_id'].values[0]
    #Train Features
    trainFeatures = getFeaturesFromTrainAndTest(data)
    df_train['is_rare'] = 0
    df_train.loc[df_train['target'] < -30, 'is_rare'] = 1
    for f in ['feature_1','feature_2','feature_3']:
        mean_encoding = df_train.groupby([f])['is_rare'].mean()
        trainFeatures[f] = trainFeatures[f].map(mean_encoding)
    columns_to_drop = ['first_active_month']
    trainFeatures = trainFeatures.drop(columns_to_drop, axis = 1)

    #historical Transaction Features
    df_hist = df_hist[df_hist['card_id'] == card_id] #selecting only relevant car
    historicalTransactionFeatures = getFeaturesFromTransactionData(df_hist, prefi
    historicalTransactionFeaturesMonth = aggregate_per_month(df_hist)

    #New Transaction Features
    df_newTrans = df_newTrans[df_newTrans['card_id'] == card_id] #selecting only
    newTransactionFeatures = getFeaturesFromTransactionData(df_newTrans, prefix =
    newTransactionFeaturesMonth = aggregate_per_month(df_newTrans)

    #merchants features
    allTransactions = pd.concat([df_hist, df_newTrans], axis = 0)
    columns_to_drop = ['merchant_category_id', 'subsector_id', 'city_id', 'state_
    allTransactions = allTransactions.drop(columns_to_drop, axis = 1)
    df_merchants = df_merchants.drop(columns_to_drop, axis = 1)
    del df_hist, df_newTrans
    gc.collect()
    df_merchants_trans = pd.merge(allTransactions, df_merchants, on='merchant_id'
    del allTransactions
    gc.collect()
    merchantsFeatures = getFeaturesFromMerchantsData(df_merchants_trans, prefix =
    del df_merchants_trans

    #merging all the data
    gc.collect()
    train = pd.merge(trainFeatures, historicalTransactionFeatures, on='card_id',h
    train = pd.merge(train, newTransactionFeatures, on='card_id', how = 'left')
    train = pd.merge(train, historicalTransactionFeaturesMonth, on = 'card_id', h
    train = pd.merge(train, newTransactionFeaturesMonth, on = 'card_id', how = 'l
    train = pd.merge(train, merchantsFeatures, on = 'card_id', how='left')
    del trainFeatures, newTransactionFeatures, newTransactionFeaturesMonth, histo

    #Handling inf values
    train.replace([-np.inf,np.inf], np.nan, inplace=True)
    try:
        train['new_Trans_transactions_count'].fillna(train['new_Trans_transactior
    except :
        pass
    try:
        train['new_Trans_authorized_flag_sum'].fillna(train['new_Trans_authorized
    except:
        pass
    try:
```

```
    train['new_Trans_authorized_flag_mean'].fillna(train['new_Trans_authorize
except :
    pass
try:
    train['new_Trans_category_1_sum'].fillna(train['new_Trans_category_1_sum'
except :
    pass
try:
    train['new_Trans_category_1_mean'].fillna(train['new_Trans_category_1_mea
except :
    pass
try:
    train['new_Trans_category_2_1.0_mean'].fillna(train['new_Trans_category_2
except :
    pass
try:
    train['new_Trans_category_2_1.0_sum'].fillna(train['new_Trans_category_2_
except :
    pass
try:
    train['new_Trans_category_2_2.0_mean'].fillna(train['new_Trans_category_2
except :
    pass
try:
    train['new_Trans_category_2_2.0_sum'].fillna(train['new_Trans_category_2_
except:
    pass
try:
    train['new_Trans_category_2_3.0_mean'].fillna(train['new_Trans_category_2
except:
    pass
try:
    train['new_Trans_category_2_3.0_sum'].fillna(train['new_Trans_category_2_
except:
    pass
try:
    train['new_Trans_category_2_4.0_mean'].fillna(train['new_Trans_category_2
except:
    pass
try:
    train['new_Trans_category_2_4.0_sum'].fillna(train['new_Trans_category_2_
except:
    pass
try:
    train['new_Trans_category_2_5.0_sum'].fillna(train['new_Trans_category_2_
except:
    pass
try:
    train['new_Trans_category_2_5.0_mean'].fillna(train['new_Trans_category_2
except:
    pass
try:
    train['new_Trans_category_3_1_sum'].fillna(train['new_Trans_category_3_1_
except:
    pass
try:
    train['new_Trans_category_3_1_mean'].fillna(train['new_Trans_category_3_1
```

```
except:
    pass
try:
    train['new_Trans_category_3_2_sum'].fillna(train['new_Trans_category_3_2_
except:
    pass
try:
    train['new_Trans_category_3_2_mean'].fillna(train['new_Trans_category_3_2
except:
    pass
try:
    train['new_Trans_category_3_3_sum'].fillna(train['new_Trans_category_3_3_
except:
    pass
try:
    train['new_Trans_category_3_3_mean'].fillna(train['new_Trans_category_3_3
except:
    pass
try:
    train['new_Trans_merchant_id_nunique'].fillna(train['new_Trans_merchant_i
except:
    pass
try:
    train['new_Trans_purchase_amount_sum'].fillna(train['new_Trans_purchase_a
except:
    pass
try:
    train['new_Trans_purchase_amount_mean'].fillna(train['new_Trans_purchase_
except:
    pass
try:
    train['new_Trans_purchase_amount_max'].fillna(train['new_Trans_purchase_a
except:
    pass
try:
    train['new_Trans_purchase_amount_min'].fillna(train['new_Trans_purchase_a
except:
    pass
try:
    train['new_Trans_purchase_amount_std'].fillna(train['new_Trans_purchase_a
except:
    pass
try:
    train['new_Trans_installments_sum'].fillna(train['new_Trans_installments_
except:
    pass
try:
    train['new_Trans_installments_mean'].fillna(train['new_Trans_installments
except:
    pass
try:
    train['new_Trans_installments_max'].fillna(train['new_Trans_installments_
except:
    pass
try:
    train['new_Trans_installments_min'].fillna(train['new_Trans_installments_
except:
```

```
        pass
    try:
        train['new_Trans_installments_std'].fillna(train['new_Trans_installments_
    except:
        pass
    try:
        train['new_Trans_purchase_month_mean'].fillna(train['new_Trans_purchase_m
    except:
        pass
    try:
        train['new_Trans_purchase_month_max'].fillna(train['new_Trans_purchase_mo
    except:
        pass
    try:
        train['new_Trans_purchase_month_min'].fillna(train['new_Trans_purchase_mo
    except:
        pass
    try:
        train['new_Trans_purchase_month_std'].fillna(train['new_Trans_purchase_mo
    except:
        pass
    try:
        train['new_Trans_purchase_year_mean'].fillna(train['new_Trans_purchase_ye
    except:
        pass
    try:
        train['new_Trans_purchase_year_max'].fillna(train['new_Trans_purchase_yea
    except:
        pass
    try:
        train['new_Trans_purchase_year_min'].fillna(train['new_Trans_purchase_yea
    except:
        pass
    try:
        train['new_Trans_purchase_year_std'].fillna(train['new_Trans_purchase_yea
    except:
        pass
    try:
        train['new_Trans_purchase_day_mean'].fillna(train['new_Trans_purchase_day
    except:
        pass
    try:
        train['new_Trans_purchase_day_max'].fillna(train['new_Trans_purchase_day_
    except:
        pass
    try:
        train['new_Trans_purchase_day_min'].fillna(train['new_Trans_purchase_day_
    except:
        pass
    try:
        train['new_Trans_purchase_day_std'].fillna(train['new_Trans_purchase_day_
    except:
        pass
    try:
        train['new_Trans_month_lag_min'].fillna(train['new_Trans_month_lag_min'].
    except:
        pass
```

```
try:
    train['new_Trans_month_lag_max'].fillna(train['new_Trans_month_lag_max'].
except:
    pass
try:
    train['new_Trans_merchant_category_id_nunique'].fillna(train['new_Trans_m
except:
    pass
try:
    train['new_Trans_state_id_nunique'].fillna(train['new_Trans_state_id_nuni
except:
    pass
try:
    train['new_Trans_subsector_id_nunique'].fillna(train['new_Trans_subsector
except:
    pass
try:
    train['new_Trans_city_id_nunique'].fillna(train['new_Trans_city_id_nuniqu
except:
    pass
try:
    train['merchant_avg_purchases_lag3_sum'].fillna(train['merchant_avg_purch
except:
    pass
try:
    train['merchant_avg_purchases_lag3_std'].fillna(train['merchant_avg_purch
except:
    pass
try:
    train['merchant_avg_purchases_lag6_sum'].fillna(train['merchant_avg_purch
except:
    pass
try:
    train['merchant_avg_purchases_lag6_std'].fillna(train['merchant_avg_purch
except:
    pass
try:
    train['merchant_avg_purchases_lag12_sum'].fillna(train['merchant_avg_purc
except:
    pass
try:
    train['merchant_avg_purchases_lag12_std'].fillna(train['merchant_avg_purc
except:
    pass
try:
    train['new_Trans_month_diff_min'].fillna(train['new_Trans_month_diff_min'
except:
    pass
try:
    train['new_Trans_month_diff_max'].fillna(train['new_Trans_month_diff_max'
except:
    pass
try:
    train['new_Trans_month_diff_mean'].fillna(train['new_Trans_month_diff_mea
except:
    pass
try:
```

```
    train['month_lag_mean_y'].fillna(train['month_lag_mean_y'].mode()[0], inp
except:
    pass
try:
    train['month_lag_std_y'].fillna(train['month_lag_std_y'].mode()[0], inpla
except:
    pass
try:
    train['purchase_amount_count_mean_y'].fillna(train['purchase_amount_count
except:
    pass
try:
    train['purchase_amount_count_std_y'].fillna(train['purchase_amount_count_
except:
    pass
try:
    train['purchase_amount_sum_mean_y'].fillna(train['purchase_amount_sum_mea
except:
    pass
try:
    train['purchase_amount_sum_std_y'].fillna(train['purchase_amount_sum_std_
except:
    pass
try:
    train['purchase_amount_mean_mean_y'].fillna(train['purchase_amount_mean_m
except:
    pass
try:
    train['purchase_amount_mean_std_y'].fillna(train['purchase_amount_mean_st
except:
    pass
try:
    train['purchase_amount_min_mean_y'].fillna(train['purchase_amount_min_mea
except:
    pass
try:
    train['purchase_amount_min_std_y'].fillna(train['purchase_amount_min_std_
except:
    pass
try:
    train['purchase_amount_max_mean_y'].fillna(train['purchase_amount_max_mea
except:
    pass
try:
    train['purchase_amount_max_std_y'].fillna(train['purchase_amount_max_std_
except:
    pass
try:
    train['purchase_amount_std_mean_y'].fillna(train['purchase_amount_std_mea
except:
    pass
try:
    train['purchase_amount_sum_mean_y'].fillna(train['purchase_amount_sum_mea
except:
    pass
try:
    train['purchase_amount_std_std_y'].fillna(train['purchase_amount_std_std_
```

```
    except:
        pass
    try:
        train['installments_count_mean_y'].fillna(train['installments_count_mean_
    except:
        pass
    try:
        train['installments_count_std_y'].fillna(train['installments_count_std_y'
    except:
        pass
    try:
        train['installments_sum_mean_y'].fillna(train['installments_sum_mean_y'].
    except:
        pass
    try:
        train['installments_sum_std_y'].fillna(train['installments_sum_std_y'].mo
    except:
        pass
    try:
        train['installments_mean_mean_y'].fillna(train['installments_mean_mean_y'
    except:
        pass
    try:
        train['installments_mean_std_y'].fillna(train['installments_mean_std_y'].
    except:
        pass
    try:
        train['installments_min_mean_y'].fillna(train['installments_min_mean_y'].
    except:
        pass
    try:
        train['installments_min_std_y'].fillna(train['installments_min_std_y'].mo
    except:
        pass
    try:
        train['installments_max_mean_y'].fillna(train['installments_max_mean_y'].
    except:
        pass
    try:
        train['installments_max_std_y'].fillna(train['installments_max_std_y'].mo
    except:
        pass
    try:
        train['installments_std_mean_y'].fillna(train['installments_std_mean_y'].
    except:
        pass
    try:
        train['installments_std_std_y'].fillna(train['installments_std_std_y'].mo
    except:
        pass

    train = train.drop(['target', 'card_id'], axis = 1)

    return train
```

In [125]:
```
def final_fun_1(data):
    df_train = reduce_mem_usage(pd.read_csv("train_EDA.csv", parse_dates=['first_
    df_hist = reduce_mem_usage(pd.read_csv("histTrans_EDA.csv", parse_dates=['pur
    df_newTrans = reduce_mem_usage(pd.read_csv("newTrans_EDA.csv", parse_dates=['
    df_merchants = reduce_mem_usage(pd.read_csv("merchants_EDA.csv"))

    allFeatures = getAllTheFeatures(data, df_train, df_hist, df_newTrans, df_merc
    print(allFeatures.shape)
    clf = joblib.load('finalModel.pkl')
    prediction = clf.predict(allFeatures)
    return prediction
```

In [126]:
```
def final_fun_2(X,Y):
    y_pred = final_fun_1(X)
    print("Actual Loyalty Score:", Y[0])
    print("Predicted Loyalty Score:", y_pred)
    print("Root mean squared error: {}".format(np.sqrt(mean_squared_error(Y, y_pr
```

In [127]:
```
df_train = reduce_mem_usage(pd.read_csv("train_EDA.csv", parse_dates=['first_acti
```

```
Mem. usage decreased to  4.04 Mb (56.2% reduction)
```

In [128]:
```
%%time
gc.collect()
final_fun_2(df_train[0:1], df_train['target'][0:1].values)
```

```
Mem. usage decreased to  4.04 Mb (56.2% reduction)
Mem. usage decreased to 1166.08 Mb (62.5% reduction)
Mem. usage decreased to 74.88 Mb (64.3% reduction)
Mem. usage decreased to 15.64 Mb (72.2% reduction)
(1, 233)
Actual Loyalty Score: 0.646
Predicted Loyalty Score: [-0.52284553]
Root mean squared error: 1.1688416275640034
CPU times: user 57 s, sys: 6.21 s, total: 1min 3s
Wall time: 1min 2s
```

In [ ]:
```
```