In [45]:
```
import datetime
import numpy as np
import pandas as pd
import os
import time
import warnings
import gc
gc.collect()
import os
from six.moves import urllib
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
#from datetime import datetime
warnings.filterwarnings('ignore')
from scipy.stats import norm, skew
from sklearn.preprocessing import StandardScaler
import sys
import joblib

from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_log_error,mean_squared_error, r2_scor
e,mean_absolute_error


from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
from scipy.stats import reciprocal, uniform

from sklearn.model_selection import StratifiedKFold, RepeatedKFold
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
from scipy import sparse
import pickle
import re
from sklearn.model_selection import StratifiedKFold
import lightgbm as lgbm

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import RepeatedKFold

from sklearn.linear_model import BayesianRidge
```

In [46]:
```
"""
This function returns all the features for a given customer which already exis
ts

"""
def existing_cust_feature(param):
    ids=param['card_id']
    data = [{'first_active_month': param['first_active_month'],'card_id':param
['card_id'],'feature_1':param['feature_1'],'feature_2':param['feature_2'],'fea
ture_3':param['feature_3']}]
    df = pd.DataFrame(data)
    details=pd.read_pickle('../input/todaydata/train.pkl')
    exists = details.isin([ids]).any().any()
    if (exists):
        print("The customer is an existing customer")
    else:
        print("The customer is new, please go for new customer option")
        sys.exit()

    cust_data=details.loc[details['card_id'] == param['card_id']]

    order_label_1 = pd.read_pickle('../input/data-prod/feature_1')
    order_label_2 = pd.read_pickle('../input/data-prod/feature_2')
    order_label_3 = pd.read_pickle('../input/data-prod/feature_3')

    for features in ['feature_1','feature_2','feature_3']:
        if(features=='feature_1'):
            df[features] = df[features].map(order_label_1)
        if(features=='feature_2'):
            df[features] = df[features].map(order_label_2)
        if(features=='feature_3'):
            df[features] = df[features].map(order_label_3)

    df['first_active_month']=pd.to_datetime(df['first_active_month'],format='%
Y-%m')
    df['day'] = (datetime.date(2018, 2, 1) - df['first_active_month'].dt.date)
.dt.days
    df['quarter'] = df['first_active_month'].dt.quarter

    for feature in ['feature_1','feature_2','feature_3']:
        column=feature+'_day'
        df[column] = df['day'] * df[feature]
        column=feature+'_day_ratio'
        df[column] = df[feature] / df['day']

    for feature in ['first_active_month', 'card_id', 'feature_1', 'feature_2',
'feature_3', 'day', 'quarter', 'feature_1_day','feature_1_day_ratio', 'feature
_2_day', 'feature_2_day_ratio','feature_3_day', 'feature_3_day_ratio']:
        cust_data[feature]=df[feature]

    with open("../input/feature/feature.txt", "rb") as fp:
        features = pickle.load(fp)
    feature = [c for c in cust_data.columns if c not in ['first_active_month',
'target', 'card_id', 'outliers',
                  'hist_purchase_date_max', 'hist_purchase_date_min', 'hist_ca
rd_id_size',
```

```
                    'new_purchase_date_max', 'new_purchase_date_min', 'new_card_
id_size']]

    return cust_data[feature]
```

In [47]:
```
"""

This function predicts the loyalty score for an existing customer

"""

def loyalty_score_prediction_1(param):
    all_features=existing_cust_feature(param)
    predictions_1 = np.zeros(len(all_features))

    lgbm_1 = joblib.load('../input/finalmodel/lgb_model-1.pkl')
    predictions_1 += lgbm_1.predict(all_features) / 5

    lgbm_2 = joblib.load('../input/finalmodel/lgb_model-2.pkl')
    predictions_2 = np.zeros(len(all_features))
    predictions_2 += lgbm_2.predict(all_features) / (5*2)

    final_model = joblib.load('../input/finalmodel/lgb_model-3.pkl')
    predictions_3 = np.zeros(len(all_features))
    test_stack = np.vstack([predictions_1, predictions_2]).transpose()
    predictions_3 += final_model.predict(test_stack) / 5

    return predictions_3
```

In [48]:
```
"""

This function returns the loyalty score as well as RMSE for an existing custom
er

"""

def loyalty_score_prediction_2(param,target):
    all_features=existing_cust_feature(param)
    predictions_1 = np.zeros(len(all_features))

    lgbm_1 = joblib.load('../input/finalmodel/lgb_model-1.pkl')
    predictions_1 += lgbm_1.predict(all_features) / 5

    lgbm_2 = joblib.load('../input/finalmodel/lgb_model-2.pkl')
    predictions_2 = np.zeros(len(all_features))
    predictions_2 += lgbm_2.predict(all_features) / (5*2)

    final_model = joblib.load('../input/finalmodel/lgb_model-3.pkl')
    predictions_3 = np.zeros(len(all_features))
    test_stack = np.vstack([predictions_1, predictions_2]).transpose()
    predictions_3 += final_model.predict(test_stack) / 5

    rmse=root_mean_squared_error(target,predictions_3)

    return predictions_3,rmse
```

In [49]:
```
def root_mean_squared_error(y_true, y_pred):
    """Root mean squared error regression loss"""
    return np.sqrt(np.mean(np.square(y_true-y_pred)))
```

In [50]:
```
def date_features(data):
  import datetime
  current_time = datetime.datetime.now()
  data['months_diff']= (current_time.year - data.purchase_date.dt.year) * 12 +
(current_time.month - data.purchase_date.dt.month)
  data['months_diff'] = data['months_diff'] + data['month_lag']
  data['purchase_month']=data.purchase_date.dt.month
  data['purchase_day']=data['purchase_date'].dt.day
  data['weekday']=data['purchase_date'].dt.weekday
  data['purchase_year'] = data['purchase_date'].dt.year
  data['weekofyear'] = data['purchase_date'].dt.weekofyear
  data['dayofweek'] = data['purchase_date'].dt.dayofweek
  data['weekend'] = (data.purchase_date.dt.weekday >=5).astype(int)
  data['hour'] = data['purchase_date'].dt.hour

  return data
```

In [51]:
```
def aggregate_func(data,str_data):
    agg_func= {
      'authorized_flag':['sum','mean'],
      'card_id':['size','count'],
      'category_1':['mean','sum','max','min'],
      'installments':['max','var','mean','skew','sum'],
      'merchant_category_id':['nunique'],
      'month_lag':['max','mean','min','var','skew'],
      'purchase_amount':['max','mean','min','var','sum','skew'],
      'subsector_id':['nunique'],
      'months_diff':['mean','max','min','var','skew'],
      'purchase_month':['max','min','mean','nunique'],
      'weekofyear': ['mean','max','min','nunique'],
      'weekend': ['sum', 'mean'],
      'weekday':['sum','mean'],
      'hour': ['mean','max','min','nunique'],
      'purchase_day':['nunique','max','min','mean'],
      'pur_date':['max','min'],
      'price' :['sum','mean','max','min','var'],
      'duration' : ['mean','min','max','var','skew'],
      'amount_month_ratio':['mean','min','max','var','skew'],

    }
    featured_data=data.groupby(['card_id']).agg(agg_func)
    col_list=[]
    for col in featured_data.columns:
        col_str='_'.join(col)
        col_str=str_data + col_str
        ren_name=col_str.split(",")
        col_list.extend(ren_name)

    col_list.insert(0,'card_id')
    featured_data.reset_index(inplace=True)
    return featured_data,col_list
```

In [52]:
```
def additional_feature(data,str):
  data[str +'_purchase_date_max'] = pd.to_datetime(data[str + '_pur_date_max'
])
  data[str + '_purchase_date_min'] = pd.to_datetime(data[str + '_pur_date_min'
])
  data[str + '_purchase_date_diff'] = (data[str + '_purchase_date_max'] - data
[str + '_purchase_date_min']).dt.days
  data[str + '_purchase_date_average'] = data[str + '_purchase_date_diff']/dat
a[str + '_card_id_size']
  data[str + '_purchase_date_uptonow'] = (datetime.datetime.today() - data[str
+ '_purchase_date_max']).dt.days
  data[str + '_purchase_date_uptomin'] = (datetime.datetime.today() - data[str
+ '_purchase_date_min']).dt.days
  data[str + '_first_buy'] = (data[str + '_purchase_date_min'] - data['first_a
ctive_month']).dt.days
  data[str + '_last_buy'] = (data[str + '_purchase_date_max'] - data['first_ac
tive_month']).dt.days

  if (str=='hist'):
    for feature in [str + '_purchase_date_max', str + '_purchase_date_min']:
      data[feature] = data[feature].astype(np.int64) * 1e-9

  if (str=='new'):
    for feature in ['new_purchase_date_max','new_purchase_date_min']:
      data[feature] = pd.DatetimeIndex(data[feature]).astype(np.int64) * 1e-9

  return data
```

In [53]:
```
def combined_feature(data):
  data['card_id_total'] = data['new_card_id_size'] + data['hist_card_id_size']
  data['card_id_cnt_total'] = data['new_card_id_count'] + data['hist_card_id_c
ount']
  data['card_id_cnt_ratio'] = data['new_card_id_count'] / data['hist_card_id_c
ount']
  data['purchase_amount_total'] = data['new_purchase_amount_sum'] + data['hist
_purchase_amount_sum']
  data['purchase_amount_mean'] = data['new_purchase_amount_mean'] + data['hist
_purchase_amount_mean']
  data['purchase_amount_max'] = data['new_purchase_amount_max'] + data['hist_p
urchase_amount_max']
  data['purchase_amount_min'] = data['new_purchase_amount_min']+ data['hist_pu
rchase_amount_min']
  data['purchase_amount_ratio'] = data['new_purchase_amount_sum'] / data['hist
_purchase_amount_sum']
  data['month_diff_mean'] = data['new_months_diff_mean'] + data['hist_months_d
iff_mean']
  data['month_diff_ratio'] = data['new_months_diff_mean'] / data['hist_months_
diff_mean']
  data['month_lag_mean'] = data['new_month_lag_mean'] + data['hist_month_lag_m
ean']
  data['month_lag_max'] = data['new_month_lag_max'] + data['hist_month_lag_ma
x']
  data['month_lag_min'] = data['new_month_lag_min']+ data['hist_month_lag_min'
]
  data['category_1_mean'] = data['new_category_1_mean'] + data['hist_category_
1_mean']
  data['installments_total'] = data['new_installments_sum'] + data['hist_insta
llments_sum']
  data['installments_mean'] = data['new_installments_mean'] + data['hist_insta
llments_mean']
  data['installments_max'] = data['new_installments_max'] + data['hist_install
ments_max']
  data['installments_ratio'] = data['new_installments_sum'] / data['hist_insta
llments_sum']
  data['price_total'] = data['purchase_amount_total'] / data['installments_tot
al']
  data['price_mean'] = data['purchase_amount_mean'] / data['installments_mean'
]
  data['price_max'] = data['purchase_amount_max'] / data['installments_max']
  data['duration_mean'] = data['new_duration_mean'] + data['hist_duration_mea
n']
  data['duration_min'] = data['new_duration_min'] + data['hist_duration_min']
  data['duration_max'] = data['new_duration_max'] + data['hist_duration_max']
  data['amount_month_ratio_mean']= data['new_amount_month_ratio_mean'] + data[
'hist_amount_month_ratio_mean']
  data['amount_month_ratio_min'] = data['new_amount_month_ratio_min'] + data[
'hist_amount_month_ratio_min']
  data['amount_month_ratio_max']= data['new_amount_month_ratio_max'] + data['h
ist_amount_month_ratio_max']
  data['new_CLV'] = data['new_card_id_count'] * data['new_purchase_amount_sum'
] / data['new_months_diff_mean']
  data['hist_CLV'] = data['hist_card_id_count'] * data['hist_purchase_amount_s
um'] / data['hist_months_diff_mean']
  data['CLV_ratio'] = data['new_CLV'] / data['hist_CLV']
```

```
    return data
```

In [54]:
```
"""
predicts the loyalty score for a new customer
"""
def loyalty_score_prediction_new_1(param):
    all_features=new_cust_feature(param)
    predictions_1 = np.zeros(len(all_features))

    lgbm_1 = joblib.load('../input/finalmodel/lgb_model-1.pkl')
    predictions_1 += lgbm_1.predict(all_features) / 5

    lgbm_2 = joblib.load('../input/finalmodel/lgb_model-2.pkl')
    predictions_2 = np.zeros(len(all_features))
    predictions_2 += lgbm_2.predict(all_features) / (5*2)

    final_model = joblib.load('../input/finalmodel/lgb_model-3.pkl')
    predictions_3 = np.zeros(len(all_features))
    test_stack = np.vstack([predictions_1, predictions_2]).transpose()
    predictions_3 += final_model.predict(test_stack) / 5

    return predictions_3
```

In [55]:
```
"""
predicts the loyalty scoreas well as RMSE  for a new customer
"""
def loyalty_score_prediction_new_2(param,target):
    all_features=new_cust_feature(param)
    #oof = np.zeros(len(train))
    predictions_1 = np.zeros(len(all_features))

    lgbm_1 = joblib.load('../input/finalmodel/lgb_model-1.pkl')
    predictions_1 += lgbm_1.predict(all_features) / 5

    lgbm_2 = joblib.load('../input/finalmodel/lgb_model-2.pkl')
    predictions_2 = np.zeros(len(all_features))
    predictions_2 += lgbm_2.predict(all_features) / (5*2)

    final_model = joblib.load('../input/finalmodel/lgb_model-3.pkl')
    predictions_3 = np.zeros(len(all_features))
    test_stack = np.vstack([predictions_1, predictions_2]).transpose()
    predictions_3 += final_model.predict(test_stack) / 5
    rmse=root_mean_squared_error(target,predictions_3)
    return predictions_3,rmse
```

In [56]:
```
"""
calculates the features for a new customer
"""

def new_cust_feature(param):
    train = [{'first_active_month': param['first_active_month'],'card_id':para
m['card_id'], 'feature_1':param['feature_1'],'feature_2':param['feature_2'],'f
eature_3':param['feature_3']}]
    hist=[{'authorized_flag':param['hist_authorized_flag'],'card_id':param['ca
rd_id'],'city_id':param['hist_city_id'],'category_1':param['hist_category_1'],
'installments':param['hist_installments'],'category_3':param['hist_category_3'
],'merchant_category_id':param['hist_merchant_category_id'],'merchant_id':para
m['hist_merchant_id'],'month_lag':param['hist_month_lag'],'purchase_amount':pa
ram['hist_purchase_amount'],'purchase_date':param['hist_purchase_date'],'categ
ory_2':param['hist_category_2'],'state_id':param['hist_state_id'],'subsector_i
d':param['hist_subsector_id']}]
    new=[{'authorized_flag':param['new_authorized_flag'],'card_id':param['card
_id'],'city_id':param['new_city_id'],'category_1':param['new_category_1'],'ins
tallments':param['new_installments'],'category_3':param['new_category_3'],'mer
chant_category_id':param['new_merchant_category_id'],'merchant_id':param['new_
merchant_id'],'month_lag':param['new_month_lag'],'purchase_amount':param['new_
purchase_amount'],'purchase_date':param['new_purchase_date'],'category_2':para
m['new_category_2'],'state_id':param['new_state_id'],'subsector_id':param['new
_subsector_id']}]
    train_csv = pd.DataFrame(train)
    historical_transactions=pd.DataFrame(hist)
    new_merchant_transactions=pd.DataFrame(new)


    order_label_1 = pd.read_pickle('../input/data-prod/feature_1')
    order_label_2 = pd.read_pickle('../input/data-prod/feature_2')
    order_label_3 = pd.read_pickle('../input/data-prod/feature_3')

    for features in ['feature_1','feature_2','feature_3']:
        if(features=='feature_1'):
            train_csv[features] = train_csv[features].map(order_label_1)
        if(features=='feature_2'):
            train_csv[features] = train_csv[features].map(order_label_2)
        if(features=='feature_3'):
            train_csv[features] = train_csv[features].map(order_label_3)

    train_csv['first_active_month']=pd.to_datetime(train_csv['first_active_mon
th'],format='%Y-%m')
    #print(df['first_active_month'])
    train_csv['day'] = (datetime.date(2018, 2, 1) - train_csv['first_active_mo
nth'].dt.date).dt.days
    train_csv['quarter'] = train_csv['first_active_month'].dt.quarter

    for feature in ['feature_1','feature_2','feature_3']:
        column=feature+'_day'
        train_csv[column] = train_csv['day'] * train_csv[feature]
        column=feature+'_day_ratio'
        train_csv[column] = train_csv[feature] / train_csv['day']

    #history
    historical_transactions['purchase_amount'] = historical_transactions['purc
```

```
hase_amount'].apply(lambda x: min(x, 0.8))

    #new
    new_merchant_transactions['purchase_amount'] = new_merchant_transactions[
'purchase_amount'].apply(lambda x: min(x, 0.8))

    #history
    historical_transactions['authorized_flag'] = historical_transactions['auth
orized_flag'].map({'Y': 1, 'N': 0})
    historical_transactions['category_1'] = historical_transactions['category_
1'].map({'Y': 1, 'N': 0})
    historical_transactions['category_3'] = historical_transactions['category_
3'].map({'A': 1, 'B': 2, 'C': 3})

    #new
    new_merchant_transactions['authorized_flag'] = new_merchant_transactions[
'authorized_flag'].map({'Y': 1, 'N': 0})
    new_merchant_transactions['category_1'] = new_merchant_transactions['categ
ory_1'].map({'Y': 1, 'N': 0})
    new_merchant_transactions['category_3'] = new_merchant_transactions['categ
ory_3'].map({'A':0, 'B':1, 'C':2})

    #history
    historical_transactions['pur_date'] = pd.DatetimeIndex(historical_transact
ions['purchase_date']).date
    historical_transactions['pur_date'] = pd.DatetimeIndex(historical_transact
ions['pur_date']).astype(np.int64) * 1e-9

    #new
    new_merchant_transactions['pur_date'] = pd.DatetimeIndex(new_merchant_tran
sactions['purchase_date']).date
    new_merchant_transactions['pur_date'] = pd.DatetimeIndex(new_merchant_tran
sactions['pur_date']).astype(np.int64) * 1e-9

    #history
    historical_transactions['purchase_date']=pd.to_datetime(historical_transac
tions['purchase_date'],format='%Y-%m')

    #new
    new_merchant_transactions['purchase_date']=pd.to_datetime(new_merchant_tra
nsactions['purchase_date'],format='%Y-%m')


    historical_transactions = date_features(historical_transactions)
    new_merchant_transactions = date_features(new_merchant_transactions)

    #other features:
    historical_transactions['duration'] = historical_transactions['purchase_am
ount'] * historical_transactions['months_diff']
    historical_transactions['amount_month_ratio'] = historical_transactions['p
urchase_amount'] / historical_transactions['months_diff']
    historical_transactions['price'] = historical_transactions['purchase_amoun
t'] / historical_transactions['installments']

    new_merchant_transactions['duration'] = new_merchant_transactions['purchas
e_amount'] * new_merchant_transactions['months_diff']
    new_merchant_transactions['amount_month_ratio'] = new_merchant_transaction
```

```
s['purchase_amount'] / new_merchant_transactions['months_diff']
    new_merchant_transactions['price'] = new_merchant_transactions['purchase_a
mount'] / new_merchant_transactions['installments']

    for i in ['category_2','category_3']:
        historical_transactions[i + '_mean']=historical_transactions['purchase
_amount'].groupby(historical_transactions[i]).agg('mean')
        historical_transactions[i + '_min']=historical_transactions['purchase_
amount'].groupby(historical_transactions[i]).agg('min')
        historical_transactions[i + '_max']=historical_transactions['purchase_
amount'].groupby(historical_transactions[i]).agg('max')
        historical_transactions[i + '_sum']=historical_transactions['purchase_
amount'].groupby(historical_transactions[i]).agg('sum')
        historical_transactions[i + '_var']=historical_transactions['purchase_
amount'].groupby(historical_transactions[i]).agg('var')

        new_merchant_transactions[i + '_mean']=new_merchant_transactions['purc
hase_amount'].groupby(new_merchant_transactions[i]).agg('mean')
        new_merchant_transactions[i + '_min']=new_merchant_transactions['purch
ase_amount'].groupby(new_merchant_transactions[i]).agg('min')
        new_merchant_transactions[i + '_max']=new_merchant_transactions['purch
ase_amount'].groupby(new_merchant_transactions[i]).agg('max')
        new_merchant_transactions[i + '_sum']=new_merchant_transactions['purch
ase_amount'].groupby(new_merchant_transactions[i]).agg('sum')
        new_merchant_transactions[i + '_var']=new_merchant_transactions['purch
ase_amount'].groupby(new_merchant_transactions[i]).agg('var')

    hist_trans,col_list=aggregate_func(historical_transactions,'hist_')
    hist_trans.columns=col_list

    new_trans,col_list=aggregate_func(new_merchant_transactions,'new_')
    new_trans.columns=col_list

    train_data=pd.merge(train_csv,hist_trans,on='card_id',how='left')
    train_data=pd.merge(train_data,new_trans,on='card_id',how='left')

    train_data=additional_feature(train_data,'hist')
    train_data=additional_feature(train_data,'new')

    train=combined_feature(train_data)

    with open("../input/feature/feature.txt", "rb") as fp:
        features = pickle.load(fp)

    feature = [c for c in train.columns if c not in ['first_active_month', 'ta
rget', 'card_id', 'outliers',
                     'hist_purchase_date_max', 'hist_purchase_date_min', 'hist_ca
rd_id_size',
                     'new_purchase_date_max', 'new_purchase_date_min', 'new_card_
id_size']]
    return train[feature]
```

In [57]:
```
existing_cust=False

if (existing_cust):
    param={
    'first_active_month':'2017-08-01',
    'card_id':'C_ID_186d6a6901',
    'feature_1':4,
    'feature_2':3,
    'feature_3':0
    }
    start_time = datetime.datetime.now()
    score=loyalty_score_prediction_1(param)
    end_time = datetime.datetime.now()
    print("------------------------------------------------------------------
------------------")
    print('loyalty score:',score)
    print("Total execution time:",(end_time-start_time))
    print("------------------------------------------------------------------
------------------")

    start_time = datetime.datetime.now()
    score,rmse=loyalty_score_prediction_2(param,-0.06540639)
    end_time = datetime.datetime.now()
    print("------------------------------------------------------------------
------------------")
    print('loyalty score:',score)
    print("RMSE:",rmse)
    print("Total execution time:",(end_time-start_time))
    print("------------------------------------------------------------------
------------------")

else:
    param={
        'first_active_month':'2017-09',
        'card_id':'C_ID_186d6a6901',
        'feature_1':4,
        'feature_2':3,
        'feature_3':0,
        'hist_authorized_flag':'Y',
        'hist_city_id':17,
        'hist_category_1':'N',
        'hist_installments':1,
        'hist_category_3':'B',
        'hist_merchant_category_id':195,
        'hist_merchant_id':'M_ID_309752ddea',
        'hist_month_lag':-1,
        'hist_purchase_amount':-0.716855,
        'hist_purchase_date':'2018-01-31 16:23:49',
        'hist_category_2':4.0,
        'hist_state_id':22,
        'hist_subsector_id':34,
        'new_authorized_flag':'Y',
        'new_city_id':17,
        'new_category_1':'N',
        'new_installments':1,
        'new_category_3':'B',
```

```
        'new_merchant_category_id':195,
        'new_merchant_id':'M_ID_309752ddea',
        'new_month_lag':-1,
        'new_purchase_amount':-0.716855,
        'new_purchase_date':'2018-01-31 16:23:49',
        'new_category_2':4.0,
        'new_state_id':22,
        'new_subsector_id':34
    }

    start_time = datetime.datetime.now()
    score=loyalty_score_prediction_new_1(param)
    end_time = datetime.datetime.now()
    print("--------------------------------------------------------------------
------------------")
    print('loyalty score:',score)
    print("Total execution time:",(end_time-start_time))
    print("--------------------------------------------------------------------
------------------")
    start_time = datetime.datetime.now()
    score,rmse=loyalty_score_prediction_new_2(param,0.142456)
    end_time = datetime.datetime.now()
    print("--------------------------------------------------------------------
------------------")
    print("loyalty score is:",score)
    print("RMSE is:",rmse)
    print("Total execution time:",(end_time-start_time))
    print("--------------------------------------------------------------------
------------------")
```

```
--------------------------------------------------------------------------------
---------
loyalty score: [0.02744678]
Total execution time: 0:00:00.433255
--------------------------------------------------------------------------------
---------
--------------------------------------------------------------------------------
---------
loyalty score is: [0.02744678]
RMSE is: 0.11500922256225433
Total execution time: 0:00:00.453006
--------------------------------------------------------------------------------
---------
```

In [ ]:

In [ ]: