In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

Imports and functions

In [2]:

```
import numpy as np
import pandas as pd
import os
import gc
import matplotlib.pylab as plt
import seaborn as sns
import warnings
import datetime
import pickle
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

warnings.filterwarnings('ignore')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: p
andas.util.testing is deprecated. Use the functions in the public API at pandas.testing i
nstead.
  import pandas.util.testing as tm
```

In [3]:

```
def reduce_mem_usage(df, verbose=True):
  #paste the kaggle kernel link
  '''
  The data size is too big to get rid of memory error this method will reduce memory
  usage by changing types. It does the following
  Load objects as categories
  Binary values are switched to int8
  Binary values with missing values are switched to float16
  64 bits encoding are all switched to 32 or 16bits if possible.
  Parameters
  ---------------
  df - DataFrame whose size to be reduced
  ---------------
  '''

  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).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
  if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end_
mem, 100 * (start_mem - end_mem) / start_mem))
  return df
```

In [4]:

```
def get_basic_time_feat(df, grpby, col, s):
  '''
  create basic time feats like differece in minute, days etcetera
  and return the dataframe.

  Parameters
  ---------------------
  df      - Features will be created
  grpby   - group the DF based on this value
  col     - column where the operations will be performed
  s       - shift value
  ---------------------
  '''

  df = df.sort_values(col)
  for i in range(s):
    df['prev_{}_'.format(i+1)+col] = df.groupby([grpby])[col].shift(i+1)
    df['purchase_date_diff_{}_days'.format(i+1)] = (df[col] - df['prev_{}_'.format(i+1)+
col]).dt.days.values
    df['purchase_date_diff_{}_seconds'.format(i+1)] = df['purchase_date_diff_{}_days'.fo
rmat(i+1)].values * 24 * 3600
    df['purchase_date_diff_{}_seconds'.format(i+1)] += (df[col] - df['prev_{}_'.format(i
+1)+col]).dt.seconds.values
    df['purchase_date_diff_{}_hours'.format(i+1)] = df.iloc[:, -1].values // 3600

  return df
```

In [5]:

```
def s_agg(new_df, df, op, prefix, grpby, col):
  '''
  takes the data frame as input and return the dataframe with the aggregate operations pe
rformed.

  Parameters
  ----------------------------
  new_df  - DF with new features added
  df      - original DF
  op      - statistical operations like min, max, mean etc.
  prefix  - prefix for the feature name
  grpby   - based on which column to group by
  col     - operations will be performed on this column
  ----------------------------
  '''

  for o in op:
    new_df[prefix+col+'_{}'.format(o)] = df.groupby([grpby])[col].agg([o]).values
  return new_df
```

In [6]:

```
def find_single_val(new_df, df, col, grpby, op, name='',  prefix='', use_col=False):
  '''
  find a value like min, max, mean in the specified column and return the DF

  Parameters
  ------------------
  new_df   - features will be added to this DF
  df       - original DF from which the features will be created
  col      - operations will be performed on this column
  grpby    - based on this column we'll to group by
  name     - name for the new features created
  op       - statistical operations to be performed
  prefix   - added to the name of the feature -- default value empty
  use_col  - if set True then the original column name will be uesd to name the new featu
re -- default value False
  ------------------
  '''

  if use_col:
    for c in col:
      for o in op:
        if o is 'min':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].min().values
        elif o is 'max':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].max().values
        elif o is 'mean':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].mean().values
        elif o is 'sum':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].sum().values
        elif o is 'nunique':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].nunique().val
ues
        elif o is 'std':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].std().values
        elif o is 'count':
          new_df[prefix+'_'+c+'_'+'{}'.format(o)] = df.groupby([grpby])[c].count().value
s

  else:
    for c in col:
      for o in op:
        if o is 'min':
          new_df[name] = df.groupby([grpby])[c].min().values
        elif o is 'max':
          new_df[name] = df.groupby([grpby])[c].max().values
        elif o is 'mean':
          new_df[name] = df.groupby([grpby])[c].mean().values
        elif o is 'sum':
          new_df[name] = df.groupby([grpby])[c].sum().values
        elif o is 'nunique':
          new_df[name] = df.groupby([grpby])[c].nunique().values
        elif o is 'std':
          new_df[name] = df.groupby([grpby])[c].std().values
        elif o is 'count':
          new_df[name] = df.groupby([grpby])[c].count().values

  return new_df
```

In [7]:

```
def get_monthlag_stat(new_df, df, grpby, op, col, name, prefix=''):

  '''
  group by the the specified column and find the count or sum depending on the input.
  Then perform basic operations like std, min, max etcetera
  parameters
  ----------------------------
  new_df - new features will be added to this DF
  df     - original DF
  grpby  - column using which we will group the data by
  col    - operations will be performed on this column
  name   - name for this columnn
  prefix - prefix to the column name
  ----------------------------
  '''
  if op == 'sum':
    tmp = df.groupby(grpby)[col].sum().unstack()
    new_df[prefix+grpby[1]+'_'+name[0]] = tmp.reset_index().iloc[:, -1].values
    new_df[prefix+grpby[1]+'_'+name[1]] = tmp.reset_index().iloc[:, -2].values

  if op == 'count':
    tmp = df.groupby(grpby)[col].count().unstack()
    # check if there is any null value and fill it with 0
    # for the sum we are not performing any null value imputation
    # as we are directly using the value. However, here we are performing operations like
    # min, max, std etcetera so we are imputing the null values.
    if tmp.isna().sum().any() > 0:
      tmp = tmp.fillna(0.0)
    new_df[prefix+grpby[1]+'_'+name[0]] = tmp.reset_index().iloc[:, 1:].std(axis=1).valu
es
    new_df[prefix+grpby[1]+'_'+name[1]] = tmp.reset_index().iloc[:, 1:].max(axis=1).valu
es
  return new_df
```

In [8]:

```
#https://www.kaggle.com/fabiendaniel/elo-world?scriptVersionId=8335387
def successive_aggregates(df, field1, field2):
    '''
    what this function does is that it group the data twice and find
    basic aggregate values.
    First it will goup by card_id and all the specified column one by one.
    Then it will find the agg values like mean, min, max and std
    for the purchase amount for each group.
    Parameters
    -------------------
    df      - original DataFrame
    field1  - first groupby along with card_id
    field2  - second grouby along with card_id
    -------------------
    '''

    t = df.groupby(['card_id', field1])[field2].mean()
    u = pd.DataFrame(t).reset_index().groupby('card_id')[field2].agg(['mean', 'min', 'ma
x', 'std'])
    u.columns = ['new_transac_' + field1 + '_' + field2 + '_' + col for col in u.columns
.values]
    u.reset_index(inplace=True)
    return u
```

In [9]:

```
def get_influential(df, col_name, date):
  '''
  This function return whether a purchase is influential or not.
  A purchase is considered influential if it is made 100 days before a festival.
  If it is not influential it will give a value 0 else the actual value.
  Parameters
  --------------------------------
  df       - Dataframe where the operations will be performed
  col_name - name of the new feature
  date     - on which date the holiday is occuring
  --------------------------------
  '''

  df[col_name] = (pd.to_datetime(date) - pd.to_datetime(df['purchase_date'])).dt.days
  df[col_name] = df[col_name].apply(lambda x: x if x > 0 and x < 100 else 0)
  return df
```

In [10]:

```
def lab_enc_load(df, col, file):
  '''
  This function will laod the saved label encoder and
  transform the data.

  Parameter
  ------------------------
  df   - Return the dataframe after encoded
  col  - Column in which the encoding will be done
  file - location of the label encoder
  ------------------------
  '''
  lbl = LabelEncoder()
  lbl.classes_ = np.load(file, allow_pickle=True)
  df[col] = lbl.transform(df[col].astype(str))
  return df
```

In [11]:

```
def preprocess():
  print('Preprocessing New Merchant Dataset...')
  new_merchant = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merchan
t_transactions.csv',parse_dates=["purchase_date"])
  new_merchant = reduce_mem_usage(new_merchant)

  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/new_merch_fill_na.cs
v'):
        print('Filled Missing values for new_merchant...')
        del new_merchant;gc.collect()
  else:

    a = pd.DataFrame()
    a['card_id'] = new_merchant['card_id']
    a['merchant_id'] = new_merchant['merchant_id']
    a['purchase_date'] = new_merchant['purchase_date']

    new_merchant.drop(['card_id', 'merchant_id', 'purchase_date'], axis=1, inplace=True)
    gc.collect()

    feat = new_merchant.columns
    cols = ['category_2', 'category_3']
    #laabel encode the variables

    file = '/content/drive/My Drive/case study/upload 15mis/label/new_merchant_authorized
_flag_enc.npy'
    new_merchant['authorized_flag'] = lab_enc_load(new_merchant, 'authorized_flag', file
)

    file = '/content/drive/My Drive/case study/upload 15mis/label/new_merchant_category_1
_enc.npy'
    new_merchant['category_1'] = lab_enc_load(new_merchant, 'category_1', file)

    #list to hold the null values
    no_nan = []
    #select only columns which doesn't have any null values
    for c in feat:
      if c not in cols:
        no_nan.append(c)

    #label encode the category 3 variables before predicting
    d = {'A':1, 'B':2, 'C':3}
    test['category_3'] = test['category_3'].map(d)

    #Loading the model and pedicing the category_2 misssing
    with open('/content/drive/My Drive/case study/upload 15mis/clf_cat2.sav', 'rb') as pi
ckle_file:
      mod = pickle.load(pickle_file)
    #create a test set by selecting only rows which are having null values
    test = new_merchant[new_merchant['category_2'].isna()]
    #make prediction only for the rows with null value
    new_merchant.loc[new_merchant['category_2'].isna(), 'category_2'] = mod.predict(test
[no_nan])

    #Loading the model and pedicing the category_3 misssing
    with open('/content/drive/My Drive/case study/upload 15mis/clf_cat3.sav', 'rb') as pi
ckle_file:
      mod = pickle.load(pickle_file)
    test = new_merchant[new_merchant['category_3'].isna()]
    new_merchant.loc[new_merchant['category_3'].isna(), 'category_3'] = mod.predict(test
[no_nan])

    new_merchant['card_id'] = a['card_id']
    new_merchant['merchant_id'] = a['merchant_id']
    new_merchant['purchase_date'] = a['purchase_date']
    del a, new_merchant, new_merch_fill_na ;gc.collect()
    print('Saving the file...')
    new_merchant.to_csv('tnew_merch_fill_na.csv')
    !cp tnew_merch_fill_na.csv "/content/drive/My Drive/case study/upload 15mis/"
    print('Filled missing values for new_merchant...')

  print('Working on Historical Transaction Dataset...')

  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/ht_fill_na.csv'):
    print('Filled missing values for historical transaction dataset...')
    #del ht_fill_na;gc.collect()
  else:
    a = pd.DataFrame()
    a['card_id'] = ht['card_id']
    a['merchant_id'] = ht['merchant_id']
    a['purchase_date'] = ht['purchase_date']

    ht.drop(['card_id', 'merchant_id', 'purchase_date'], axis=1, inplace=True)
    gc.collect()

    feat = ht.columns
    cols = ['category_2', 'category_3']
    #laabel encode the variables
    file = '/content/drive/My Drive/case study/upload 15mis/label/ht_authorized_flag_enc.
npy'
    ht['authoirzed_flag'] = lab_enc_load(ht, 'authorized_flag', file)

    file = '/content/drive/My Drive/case study/upload 15mis/label/ht_category_1_enc.npy'
    ht['category_1'] =lab_enc_load(ht, 'category_1', file)

    #list to hold the null values
    no_nan = []
    #select only columns which doesn't have any null values
    for c in feat:
      if c not in cols:
        no_nan.append(c)

    #label encode the category 3 variables before feeding it to the model
    d = {'A':1, 'B':2, 'C':3}
    test['category_3'] = test['category_3'].map(d)

    #Loading the model and pedicing the category_2 misssing
    with open('/content/drive/My Drive/case study/upload 15mis/ht_clf_cat2.sav', 'rb') as
pickle_file:
      mod = pickle.load(pickle_file)
    test = ht[ht['category_2'].isna()]
    #make prediction only for the rows with null value
    ht.loc[ht['category_2'].isna(), 'category_2'] = mod.predict(test[no_nan])

    #Loading the model and pedicing the category_3 misssing
    with open('/content/drive/My Drive/case study/upload 15mis/ht_clf_cat3.sav', 'rb') as
pickle_file:
      mod = pickle.load(pickle_file)
    test = ht[ht['category_3'].isna()]
    ht.loc[ht['category_3'].isna(), 'category_3'] = mod.predict(test[no_nan])

    ht['card_id'] = a['card_id']
    ht['merchant_id'] = a['merchant_id']
    ht['purchase_date'] = a['purchase_date']

    print('Saving the file...')
    ht.to_csv('tht_fill_na.csv')
    !cp tht_fill_na.csv "/content/drive/My Drive/case study/upload 15mis/"
    print('Filled missing values for historical transactions...')
    del ht,a;gc.collect()

  print('Loading Merchant Dataset for preprocessing...')
  merchant = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/merchants.csv')
  merchant = reduce_mem_usage(merchant)

  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/merchants.csv'):
    #processed = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/merchants.csv')
    print('Filled missing values for merchant dataset...')
  else:
    merchant = merchant[merchant['avg_purchases_lag3']!=np.inf]
    tmp = pd.DataFrame()
    tmp['merchant_id'] = merchant['merchant_id']
    tmp['category_2'] = merchant['category_2']

    merchant.drop(['merchant_id', 'category_2'], axis=1, inplace=True)

    file = '/content/drive/My Drive/case study/upload 15mis/label/merchant_category_4_enc.npy'
    merchant['category_4'] = lab_enc_load(merchant, 'category_4', file)

    file = '/content/drive/My Drive/case study/upload 15mis/label/merchant_category_1_enc.npy'
    merchant['category_1'] = lab_enc_load(merchant, 'category_1', file)

    file = '/content/drive/My Drive/case study/upload 15mis/label/merchant_most_recent_sales_range_enc.npy'
    merchant['most_recent_sales_range'] = lab_enc_load(merchant, 'most_recent_sales_range', file)

    file = '/content/drive/My Drive/case study/upload 15mis/label/merchant_most_recent_purchases_range_enc.npy'
    merchant['most_recent_purchases_range'] = lab_enc_load(merchant, 'most_recent_purchases_range', file)

    feat = merchant.columns
    cols = ['avg_sales_lag3','avg_sales_lag6','avg_sales_lag12']
    no_nan = []

    for c in feat:
      if c not in cols:
        no_nan.append(c)

    #Loading the model and predict the missing values in avg_sales_lag3
    with open('/content/drive/My Drive/case study/upload 15mis/merch_clf_knn.sav', 'rb') as pickle_file:
      mod = pickle.load(pickle_file)
    test = merchant[merchant['avg_sales_lag3'].isna()]
    merchant.loc[merchant['avg_sales_lag3'].isna(), 'avg_sales_lag3'] = mod.predict(test[no_nan])

    #Loading the model and predict the missing values in avg_sales_lag6
    with open('/content/drive/My Drive/case study/upload 15mis/merch_clf2_knn.sav', 'rb') as pickle_file:
      mod = pickle.load(pickle_file)
    test = merchant[merchant['avg_sales_lag6'].isna()]
    merchant.loc[merchant['avg_sales_lag6'].isna(), 'avg_sales_lag6'] = mod.predict(test[no_nan])

    #Loading the model and predict the missing values in avg_sales_lag12
    with open('/content/drive/My Drive/case study/upload 15mis/merch_clf3_knn.sav', 'rb') as pickle_file:
      mod = pickle.load(pickle_file)
    test = merchant[merchant['avg_sales_lag12'].isna()]
    merchant.loc[merchant['avg_sales_lag12'].isna(), 'avg_sales_lag12'] = mod.predict(test[no_nan])

    #predicting the missing value for category_2
    merchant['category_2'] = tmp['category_2']

    feat = merchant.columns
    cols = ['category_2']
    no_nan = []
    for c in feat:
      if c not in cols:
        no_nan.append(c)

    test = merchant[merchant['category_2'].isna()]
    with open('/content/drive/My Drive/case study/upload 15mis/merch_clf_cat2.sav', 'rb')
as pickle_file:
      mod = pickle.load(pickle_file)
    merchant.loc[merchant['category_2'].isna(), 'category_2'] = mod.predict(test[no_nan]
)

    merchant['merchant_id'] = tmp['merchant_id']
    merchant.to_csv('tmerch_fill_na.csv')
    !cp tmerch_fill_na.csv "/content/drive/My Drive/case study/upload 15mis/"
    del merchant;gc.collect()

  #print('Loading the dataset filled NaN...')

  print('One Hot Encoding the variables...')
  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/new_merchant_process
ed_fill_na.csv'):
    print('Completed...')
    #del new_merch_fill_na;gc.collect()

  else:
    new_merchant = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merch
_fill_na.csv')
    new_merchant = reduce_mem_usage(new_merchant)
    ht = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/ht_fill_na.csv')
    ht = reduce_mem_usage(ht)

    file = '/content/drive/My Drive/case study/upload 15mis/label/ht_category_3_enc.npy'
    ht['category_3'] = lab_enc_load(ht, 'category_3', file)

    file = '/content/drive/My Drive/case study/upload 15mis/label/new_merchant_category_3
_enc.npy'
    new_merchant['category_3'] = lab_enc_load(new_merchant, 'category_3', file)

    gc.collect()

    mont = [0,-1,-2,-3,-4,-5,-6]
    cat_2 = [1.,2.,3.,4.,5.]
    cat_3 = [0,1,2,3]

    for val in mont:
      ht['month_lag={}'.format(val)] = (ht['month_lag'] == val).astype(int)

    for val in cat_2:
      ht['category_2={}'.format(int(val))] = (ht['category_2'] == val).astype(int)

    for val in cat_3:
      ht['category_3={}'.format(int(val))] = (ht['category_3'] == val).astype(int)
    gc.collect()

    cat_2 = [1.,2.,3.,4.,5.]
    cat_3 = [0,1,2,3]
    mont = [1,2]

    for val in mont:
      new_merchant['month_lag={}'.format(val)] = (new_merchant['month_lag'] == val).asty
pe(int)
    for val in cat_2:
      new_merchant['category_2={}'.format(int(val))] = (new_merchant['category_2'] == va
l).astype(int)
    for val in cat_3:
      new_merchant['category_3={}'.format(int(val))] = (new_merchant['category_3'] == va
l).astype(int)
    gc.collect()

    ht['purchase_month'] = ht['purchase_date'].astype(str)
    ht['reference_month'] = pd.to_datetime(ht['purchase_month'].apply(lambda x: x[:7] +
'-28')) - \
                                          ht['month_lag'].apply(lambda x: np.timedelta6
4(x, 'M'))
    gc.collect()

    ht['reference_month'] = [x[:7] for x in ht['reference_month'].astype(str)]
    del ht['purchase_month'];gc.collect()

    new_merchant['reference_month'] = (pd.to_datetime(pd.DatetimeIndex(new_merchant['pur
chase_date']).date) - \
                                   new_merchant['month_lag'].apply(lambda x: np.timedelt
a64(x, 'M')))
    new_merchant['reference_month'] = [x[:7] for x in new_merchant['reference_month'].as
type(str)]

    new_merchant.to_csv('tnew_merchant_processed_fill_na.csv', index=False)
    !cp tnew_merchant_processed.csv "/content/drive/My Drive/Colab Notebooks/ELO"
    ht.to_csv('tht_processed_fill_na.csv', index=False)
    !cp tht_processed.csv "/content/drive/My Drive/Colab Notebooks/ELO"

    #del new_merch_fill_na;gc.collect()

    print('Completed One Hot Encoding...')
```

In [12]:

```
def fe_inf():
  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/new_merch_info_filln
a.csv'):
    print('Completed FE of transaction info...')

  else:
    new = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merch_fill_na_
processed.csv')
    new_merchant_feats = pd.DataFrame(new.groupby(['card_id']).size()).reset_index()
    new_merchant_feats.columns = ['card_id', 'new_transac_count']
    new['purchase_amount'] = np.round(new['purchase_amount'] / 0.00150265118 + 497.06, 2
)
    ht = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/ht_processed_fill_n
a.csv')
    historical_trans_features = pd.DataFrame(ht.groupby(['card_id']).size()).reset_index
()
    historical_trans_features.columns = ['card_id', 'hist_transac_count']
    ht['purchase_amount'] = np.round(ht['purchase_amount'] / 0.00150265118 + 497.06, 2)

    cols = ['city_id', 'state_id', 'merchant_category_id', 'subsector_id', 'merchant_id'
]
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=cols, grpby='card_i
d',\
                                    op=['nunique'], prefix='new_transac', use_col=True)

    cols = ['city_id', 'state_id', 'merchant_category_id', 'subsector_id', 'merchant_id'
]
    historical_trans_features = find_single_val(historical_trans_features, new, col=cols,
grpby='card_id',\
                                  op=['nunique'], prefix='hist_transac', use_col=True)

    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['category_1'], grp
by='card_id',\
                                    op=['sum'], prefix='new_transac', use_col=True)
    new_merchant_feats['new_transac_category_0_sum'] = new_merchant_feats['new_transac_co
unt'].values - new_merchant_feats.iloc[:, -1].values

    historical_trans_features = find_single_val(historical_trans_features, new, col=['cat
egory_1'], grpby='card_id',\
                                    op=['sum'], prefix='hist_transac', use_col=True)
    historical_trans_features['hist_transac_category_0_sum'] = historical_trans_features[
'hist_transac_count'].values - \
                                                            historical_trans_features.i
loc[:, -1].values

    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['category_1'], grp
by='card_id',\
                                op=['mean','std'], prefix='new_transac', use_col=True)

    new_merchant_feats = s_agg(new_merchant_feats, new, col='installments', grpby='card_i
d', \
                              op=['mean', 'sum', 'max', 'min', 'std', 'skew'], prefix='
new_transac_')

    historical_trans_features = find_single_val(historical_trans_features, new, col=['cat
egory_1'], grpby='card_id',\
                                                op=['mean','std'], prefix='hist_transac
', use_col=True)

    historical_trans_features = s_agg(historical_trans_features, new, col='installments',
grpby='card_id', \
                                      op=['mean', 'sum', 'max', 'min', 'std', 'skew'],
\
                      prefix='hist_transac')

    cols = ['category_2=1', 'category_2=2', 'category_2=3', 'category_2=4', 'category_2=5
',
            'category_3=0', 'category_3=1', 'category_3=2', 'category_3=3']
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=cols, grpby='card_i
d',\
                                        op=['mean','sum'], prefix='new_transac', use_co
l=True)

    cols = ['category_2=1', 'category_2=2', 'category_2=3', 'category_2=4', 'category_2=5
',
            'category_3=0', 'category_3=1', 'category_3=2', 'category_3=3']
    historical_trans_features = find_single_val(historical_trans_features, new, col=cols,
grpby='card_id',\
                                                op=['mean','sum'], prefix='new_transac'
, use_col=True)

    historical_trans_features = get_monthlag_stat(historical_trans_features, new, grpby=
['card_id','month_lag'], op='count', \
                                              col='purchase_amount', prefix='hist_trans
ac', name=['count_std','count_max'])

    historical_trans_features = find_single_val(historical_trans_features, new, col=['aut
horized_flag'], grpby='card_id',\
                                                op=['sum', 'mean'], prefix='hist_transa
c', use_col=True)
    historical_trans_features['hist_transac_denied_count'] = historical_trans_features['
hist_transac_count'].values - \
                                                            historical_trans_features.i
loc[:, -1].values

    historical_trans_features['hist_transac_merchant_id_count_mean'] = historical_trans_f
eatures['hist_transac_count'].values \
                                                                    / historical_trans_
features['hist_transac_merchant_id_nunique'].values

    historical_trans_features['hist_transac_merchant_count_max'] = ht.groupby(['card_id'
, 'merchant_id']).size().reset_index().\
                                                                  groupby(['card_id'])[
0].max().values

    new_merchant_feats = get_monthlag_stat(new_merchant_feats, new, grpby=['card_id','mon
th_lag'], op='count', \
                                          col='purchase_amount', prefix='new_transac_',
name=['count_std','count_max'])

    historical_trans_features['hist_transac_merchant_ratio'] = historical_trans_features.
iloc[:, -1].values \
                                                                          / historical_
trans_features['hist_transac_count'].values
    historical_trans_features['hist_transac_merchant_id_ratio'] = historical_trans_featur
es.iloc[:, -2].values \
                                                                              / histori
cal_trans_features['hist_transac_merchant_id_count_mean'].values
```

```
    historical_trans_features['hist_transac_merchant_count_std'] = ht.groupby(['card_id'
, 'merchant_id']).size().reset_index().\
                                                                  groupby(['card_id'])[
0].std().values
    historical_trans_features.to_csv('thist_transac_info_fill_na.csv')
    new_merchant_feats.to_csv('tnew_merch_info_fillna.csv')
    print('Completed FE of transaction info...')
```

In [13]:

```
def fe_am():
  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/new_merch_amount_fil
lna.csv'):
    print('Completed FE of purchase amount...')

  else:
    new = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merchant_proce
ssed_fill_na.csv')
    new_merchant_feats = pd.DataFrame(new.groupby(['card_id']).size()).reset_index()
    new_merchant_feats.columns = ['card_id', 'new_transac_count']
    #the purchase amount given to us is normalized. It does not make any sense if we look
 at it.
    #Credits to the user radar he somehow deanonymize the data and give the below formula
 to transform the purchase
    #amount which will make much sense
    # kaggle.com/raddar/towards-de-anonymizing-the-data-some-insights
    new['purchase_amount'] = np.round(new['purchase_amount'] / 0.00150265118 + 497.06, 2
)

    ht = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/ht_processed_fill_n
a.csv')
    historical_trans_features = pd.DataFrame(ht.groupby(['card_id']).size()).reset_index
()
    historical_trans_features.columns = ['card_id', 'hist_transac_count']
    ht['purchase_amount'] = np.round(ht['purchase_amount'] / 0.00150265118 + 497.06, 2)

    #crete agg features based on the purchase amount
    op = ['sum', 'mean', 'max', 'min', 'median', 'std', 'skew']
    new_merchant_feats = s_agg(new_merchant_feats, new, op=op, prefix='new_transac_', co
l='purchase_amount', grpby='card_id')
    #finding the difference between the maximum and minmum purchase amount
    new_merchant_feats['new_transac_amount_diff'] = new_merchant_feats['new_transac_purch
ase_amount_max'].values - \
                                                    new_merchant_feats['new_transac_pur
chase_amount_min'].values

    #create basic agg features from the purchase amount column grouped by card id
    op = ['sum', 'mean', 'max', 'min', 'median', 'std', 'skew']
    historical_trans_features = s_agg(historical_trans_features, ht, op=op, prefix='hist
_transac_', \
                      col='purchase_amount', grpby='card_id')
    #finding the difference between the purchase amount max and min
    historical_trans_features['hist_transac_amount_diff'] = historical_trans_features['hi
st_transac_purchase_amount_max'].values - \
                                            historical_trans_features['hist_transac_purc
hase_amount_min'].values

    #basic month features
    new_merchant_feats = get_monthlag_stat(new_merchant_feats, new, grpby=['card_id','mon
th_lag'], op='sum',\
                                            col='purchase_amount', prefix='new_transac_'
, \
                                            name=['1_amount','2_amount'])
    # dividing monthlag2 by 1 to find the ratio
    new_merchant_feats['new_transac_monthlag_ratio'] = (new_merchant_feats.iloc[:, -1] /
new_merchant_feats.iloc[:, -2])\
                                                              .replace([np.inf, -np.inf
], np.nan)
    #create another feature by taking the log of the ratio
    new_merchant_feats['new_transac_monthlag_log_ratio'] = np.log2(new_merchant_feats.ilo
c[:, -1])
```

```
    #successive agg features
    #create a temp DF ADD to hold the new features
    add = successive_aggregates(ht, field1='category_1', field2='purchase_amount')
    col = ['installments', 'city_id', 'merchant_category_id', 'merchant_id',\
          'subsector_id','category_2','category_3']

    #for each column commpute the agg and merge with the temp DF
    for c in col:
      add = add.merge(successive_aggregates(ht, c, 'purchase_amount'), \
                on=['card_id'], how='left')
    #merge the temp DF with our feature set
    new_merchant_feats = new_merchant_feats.merge(add, on=['card_id'], how='left')

    #successive agg features
    #create a temp DF ADD to hold the new features
    add = successive_aggregates(new, 'category_1', 'purchase_amount')
    col = ['installments', 'city_id', 'merchant_category_id', 'merchant_id',\
          'subsector_id','category_2','category_3']

    #for each column commpute the agg and merge with the temp DF
    for c in col:
      add = add.merge(successive_aggregates(new, c, 'purchase_amount'), \
                on=['card_id'], how='left')
    #merge the temp DF with our feature set
    historical_trans_features = historical_trans_features.merge(add, on=['card_id'], how
='left')

    #save the created features
    new_merchant_feats.to_csv('tnew_merch_amount_fillna.csv', index=False)
    historical_trans_features.to_csv('thist_transac_amount_fill_na.csv', index=False)
    print('Completed FE of purchase amount...')
```

In [14]:

```
def fe_tm():
  if os.path.isfile('/content/drive/My Drive/case study/upload 15mis/new_merch_time_fill_
na.csv'):
    print('Completed FE of purchase amount...')

  else:
    new = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merchant_proce
ssed_fill_na.csv')
    new_merchant_feats = pd.DataFrame(new.groupby(['card_id']).size()).reset_index();gc.
collect()
    new_merchant_feats.columns = ['card_id', 'new_transac_count']

    ht = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/ht_processed_fill_n
a.csv')
    historical_trans_features = pd.DataFrame(ht.groupby(['card_id']).size()).reset_index
();gc.collect()
    historical_trans_features.columns = ['card_id', 'hist_transac_count']
    ht['purchase_amount'] = np.round(ht['purchase_amount'] / 0.00150265118 + 497.06, 2)

    #agg feat like mean, std, max for the column monthlag grouped by card_id
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max'], prefi
x='new_transac_', grpby='card_id', col='month_lag')

    #get agg feats like min, mean, std for the col specified
    historical_trans_features = s_agg(historical_trans_features, ht, ['nunique', 'mean',
'std', 'min', 'skew'], 'hist_transac_', 'card_id', 'month_lag')

    #get values like min and max values from the col purchase_date
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['purchase_date'],
grpby='card_id', op=['max','min'], prefix='new_transac', use_col=True)
    #based on the min and max find difference and ratio
    new_merchant_feats['purchase_date_diff'] = (pd.to_datetime(new_merchant_feats.iloc[:,
-2]) -
                                                pd.to_datetime(new_merchant_feats.iloc[
:, -1])).dt.days.values
    new_merchant_feats['purchase_count_ratio'] = new_merchant_feats['new_transac_count'].
```

```
values / (1. + new_merchant_feats.iloc[:, -1].values)

    #get values like min and max values from the col purchase_date
    historical_trans_features = find_single_val(historical_trans_features, ht, col=['pur
chase_date'], grpby='card_id',\
                                op=['max','min'], prefix='hist_transac', use_col=True)
    #create feats like difference and ratio between the first and last purchases made for
a card_id
    historical_trans_features['hist_purchase_date_diff'] = (pd.to_datetime(historical_tr
ans_features.iloc[:, -2]) - \
                                                            pd.to_datetime(historical_t
rans_features.iloc[:, -1])).dt.days.values
    historical_trans_features['hist_purchase_count_ratio'] = historical_trans_features['
hist_transac_count'].values / (1. + historical_trans_features.iloc[:, -1].values)

    reference_date = '2018-12-31'
    #features based on if the particular day is a weekend
    new['is_weekend'] = (pd.DatetimeIndex(new['purchase_date']).dayofweek)
    #>5 to check whether the day is sat or sunday if it is then assign a val 1 else 0
    new['is_weekend'] = new['is_weekend'].apply(lambda x: 1 if x >= 5 else 0).values
    #get the values of mean and sum grouped by card_id for the weekend feature
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['is_weekend'], grp
by='card_id', name='purchase_weekend_count',\
                                op=['sum'], prefix='new_transac')
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['is_weekend'], grp
by='card_id', name='purchase_weekend_mean',\
                                op=['mean'], prefix='new_transac')

    #features based on if the particular day is a weekend
    #day is termed as weekend if it is either sat or sunday
    ht['is_weekend'] = (pd.DatetimeIndex(ht['purchase_date']).dayofweek)
    #>5 to check whether the day is sat or sunday if it is then assign a val 1 else 0
    ht['is_weekend'] = ht['is_weekend'].apply(lambda x: 1 if x >= 5 else 0).values
    #get the values of mean and sum grouped by card_id for the weekend feature
    # find purchases made in weekend sum
    historical_trans_features = find_single_val(historical_trans_features, ht, col=['is_
weekend'], grpby='card_id', \
                                                name='purchase_weekend_count', op=['sum
'], prefix='hist_transac')
    #find purchases made in weekend mean
    historical_trans_features = find_single_val(historical_trans_features, ht, col=['is_
weekend'], grpby='card_id', \
                                                name='purchase_weekend_mean', op=['mean
'], prefix='hist_transac')
    historical_trans_features = historical_trans_features.merge(ht[['card_id', 'referenc
e_month']]\
    .drop_duplicates(), on='card_id', how='left')
    historical_trans_features['reference_month'] = pd.to_datetime(historical_trans_featu
res['reference_month'])

    purchase_date = pd.to_datetime(new['purchase_date'])
    reference_date = pd.to_datetime(reference_date)
    # We need to find the difference in days then we can divide by 30 to convert it into
months.
    # as timedelta doesn't have attribute to directly get months.
    new['month_diff'] = (reference_date - purchase_date).dt.days
    new['month_diff'] = new['month_diff'] // 30 + new['month_lag']
    new['month_diff'].head()
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['month_diff'], grp
by='card_id', \
                                        name='new_month_diff_mean', op=['mean'])

    purchase_date = pd.to_datetime(ht['purchase_date'])
    reference_date = pd.to_datetime(reference_date)
    # We need to find the difference in days then we can divide by 30 to convert it into
months.
    # as timedelta doesn't have attribute to directly get months.
    ht['month_diff'] = (reference_date - purchase_date).dt.days
    ht['month_diff'] = ht['month_diff'] // 30 + ht['month_lag']
    ht['month_diff'].head()
    historical_trans_features = s_agg(historical_trans_features, ht, op=['mean', 'std',
'min', 'max'], \
                      col='month_diff', grpby='card_id', prefix='hist_')

    new['amount_month_ratio'] = new['purchase_amount'].values / (1. + new['month_diff'].
values)
    #agg feat based on the cols created in the last part
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'min', 'max',
'skew'], \
                              prefix='new_transac_', grpby='card_id', col='duration')
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'min', 'max',
'skew'], \
                              prefix='new_transac_', grpby='card_id', col='amount_month_
ratio')
    #find sum and mean of the col monthlag col grouped by card_id
    new_merchant_feats = find_single_val(new_merchant_feats, new, col=['month_lag=1', 'mo
nth_lag=2'], grpby='card_id',\
                                op=['sum','mean'], prefix='new_transac', use_col=True)

    ht['amount_month_ratio'] = ht['purchase_amount'].values / (1. + ht['month_diff'].val
ues)
    #agg feat based on the cols created in the last part
    historical_trans_features = s_agg(historical_trans_features, ht, ['mean', 'std', 'mi
n', 'max', 'skew'], \
                      prefix='hist_transac_', grpby='card_id', col='duration')
    historical_trans_features = s_agg(historical_trans_features, ht, ['mean', 'std', 'mi
n', 'max', 'skew'], \
                      prefix='hist_transac_', grpby='card_id', col='amount_month_ratio')
    #find sum and mean of the col monthlag col grouped by card_id
    historical_trans_features = find_single_val(historical_trans_features, ht, col=['mon
th_lag=0', 'month_lag=-1', 'month_lag=-2'],\
                                                grpby='card_id', op=['sum','mean'], pre
fix='hist_transac', use_col=True)

    #extract week, day, and hour from the date column then
    #create agg features like mean, min, max for each of the
    #features separately
    ht['week'] = pd.DatetimeIndex(ht['purchase_date']).week.values
    ht['day'] = pd.DatetimeIndex(ht['purchase_date']).dayofweek.values
    ht['hour'] = pd.DatetimeIndex(ht['purchase_date']).hour.values
    #get aggregate values from the cols week, day and hour
    gc.collect()
    historical_trans_features = s_agg(historical_trans_features, ht, op=['nunique', 'mea
n', 'min', 'max'], \
                      col='week', grpby='card_id', prefix='hist_transac')
    historical_trans_features = s_agg(historical_trans_features, ht, op=['nunique', 'mea
n', 'min', 'max'], \
                      col='day', grpby='card_id', prefix='hist_transac')
    historical_trans_features = s_agg(historical_trans_features, ht, op=['nunique', 'mea
n', 'min', 'max'], \
                      col='hour', grpby='card_id', prefix='hist_transac')

    #calculating the ratio between the two monthlags
    new_merchant_feats['new_transac_month_lag=1_2_ratio'] = new_merchant_feats['new_trans
ac_month_lag=1_sum'].values \
                                                            / (1. + new_merchant_feats[
'new_transac_month_lag=2_sum'].values)
    #get basic time feat and create agg features based on the created cols
    new = get_basic_time_feat(new, 'card_id', 'purchase_date', 2)
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max', 'min']
, prefix='new_transac_', \
                              grpby='card_id', col='purchase_date_diff_1_seconds')
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max', 'min']
, prefix='new_transac_', \
                              grpby='card_id', col='purchase_date_diff_1_days')
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max', 'min']
, prefix='new_transac_', \
                              grpby='card_id', col='purchase_date_diff_1_hours')
    #get basic time feat and create agg features based on the created cols
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max', 'min']
, prefix='new_transac_', \
                              grpby='card_id', col='purchase_date_diff_2_seconds')
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max', 'min']
, prefix='new_transac_', \
                              grpby='card_id', col='purchase_date_diff_2_days')
    new_merchant_feats = s_agg(new_merchant_feats, new, op=['mean', 'std', 'max', 'min']
, prefix='new_transac_', \
                              grpby='card_id', col='purchase_date_diff_2_hours')

    #find the ratio between the monthlag cols
    historical_trans_features['hist_transac_monthlag_0_-1_ratio'] = historical_trans_feat
ures.iloc[:, -6].values \
                                                                  / (1. + historical_tr
ans_features.iloc[:, -4].values)
    historical_trans_features['hist_transac_monthlag_0_-2_ratio'] = historical_trans_feat
ures.iloc[:, -7].values \
                                                                / (1. + historical_tran
s_features.iloc[:, -3].values)
    #create a feature of the sum of all the three monthlag sum
    #crete a temp dataframe which holds the three cols
    col = ['hist_transac_month_lag=0_sum', 'hist_transac_month_lag=-1_sum', 'hist_transac
_month_lag=-2_sum']
    tmp = historical_trans_features[col]
    #perform sum operation over the cols
    historical_trans_features['hist_transac_3mon_sum'] = tmp.sum(axis=1)
    del tmp;gc.collect()
    historical_trans_features['hist_transac_3mon_ratio'] = historical_trans_features.iloc
[:, -1].values \
                                                          / (1. + historical_trans_feat
ures['hist_transac_count'].values)

    #if it gives an error use ht['purchase_date'] = pd.to_datetime(ht['purchase_date'])
    ht = ht.sort_values('purchase_date')
    #get basic time feat and create agg features based on the created cols
    ht = get_basic_time_feat(ht, 'card_id', 'purchase_date', 1)
    #get basic time feat and create agg features based on the created cols
    historical_trans_features = s_agg(historical_trans_features, ht, op=['mean', 'std',
'max', 'min'], prefix='hist_transac_', \
                                      grpby='card_id', col='purchase_date_diff_1_seconds
')
    historical_trans_features = s_agg(historical_trans_features, ht, op=['mean', 'std',
'max', 'min'], prefix='hist_transac_', \
                                      grpby='card_id', col='purchase_date_diff_1_days')
    historical_trans_features = s_agg(historical_trans_features, ht, op=['mean', 'std',
'max', 'min'], prefix='hist_transac_', \
                                      grpby='card_id', col='purchase_date_diff_1_hours')

    #create influential day features. If a purchase is made withing 100 days
    #before or after a festival then it is called as influential days.
    holiday = ['ChristmasDay_2017', 'FathersDay_2017', 'ChildrenDay_2017', 'BlackFriday_
2017', 'ValentineDay_2017', 'MothersDay_2018']
    date = ['2017-12-25', '2017-08-13', '2017-10-12', '2017-11-24', '2017-06-12', '2018-0
5-13']

    for idx, day in enumerate(holiday):
      new = get_influential(new, day, date[idx])

    #loop through all the created features and add it to the DataFrame
    for c in holiday:
        gc.collect()
        new_merchant_feats['new_transac_{}_mean'.format(c)] = new.groupby(['card_id'])[c
]\
                                                              .mean().values
    new_merchant_feats.drop(['new_transac_count'], axis=1, inplace=True)

    #create influential day features. If a purchase is made withing 100 days
    #before or after a festival then it is called as influential days.
    ht['purchase_date'] = pd.to_datetime(ht['purchase_date'])
    for idx, day in enumerate(holiday):
      ht = get_influential(ht, day, date[idx])

    #loop through all the created features and add it to the DataFrame
    for c in holiday:
      gc.collect()
      historical_trans_features['hist_transac_{}_mean'.format(c)] = ht.groupby(['card_id
'])[c]\
                                                                  .mean().values
    historical_trans_features.drop(['hist_transac_count'],axis=1,inplace=True)

    #save the created features
    new_merchant_feats.to_csv('tnew_merch_time_fillna.csv', index=False)
    historical_trans_features.to_csv('thist_transac_time_fill_na.csv', index=False)
```

In [15]:

```
def get_train(train):
  print('Started Preprocessing...')
  preprocess()
  print('Started Feature Engineering...')
  fe_inf()
  fe_am()
  fe_tm()
  print('Feature Engineering Completed')

  print('Preparing train and test set...')
  hist_transac_amount = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/hist
_transac_amount_fill_na.csv')
  hist_transac_info = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/hist_t
ransac_info_fill_na.csv')
  hist_transac_time = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/hist_t
ransac_time_fill_na.csv')
  new_merch_amount = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_mer
ch_amount_fillna.csv')
  new_merch_info = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merch
_info_fillna.csv')
  new_merch_time = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/new_merch
_time_fill_na.csv')

  hist_transac_info.drop('Unnamed: 0', axis=1, inplace=True)
  new_merch_info.drop('Unnamed: 0', axis=1, inplace=True)
  new_merch_time.drop('Unnamed: 0', axis=1, inplace=True)
  hist_feats = hist_transac_info.merge(hist_transac_amount, on='card_id', how='left')
  hist_feats = hist_feats.merge(hist_transac_time, on='card_id', how='left')
  del hist_transac_info, hist_transac_amount, hist_transac_time;gc.collect()

  new_feats = new_merch_info.merge(new_merch_amount, on='card_id', how='left')
  new_feats = new_feats.merge(new_merch_time, on='card_id', how='left')
  del new_merch_info, new_merch_amount, new_merch_time;gc.collect()
  print('Loading train and test')

  train_df = train
  print('merge train and new features...')
  train_df = train_df.merge(hist_feats, on=['card_id'], how='left')
  train_df = train_df.merge(new_feats, on=['card_id'], how='left')
  train_df['outliers'] = 0
  train_df.loc[train_df['target'] < -30, 'outliers'] = 1
  act_date = pd.to_datetime('2018-12-31')

  for df in [train_df]:
      #converting the col ref_month an first_act_month to datetime type
      reference_month = pd.to_datetime(df['reference_month'])
      first_act_month = pd.to_datetime(df['first_active_month'])
      #extracting the year and month from the first_act_month
      df['year'] = pd.DatetimeIndex(df['first_active_month']).year.values
      df['month'] = pd.DatetimeIndex(df['first_active_month']).month.values
      df['month_diff'] = (reference_month - \
                          first_act_month).dt.days.values
      df['elapsed_days'] = (act_date - reference_month).dt.days.values
      df['hist_purchase_active_diff'] = (pd.to_datetime(df['hist_transac_purchase_date_m
in'].astype(str)\
                                          .apply(lambda x: x[:7])) - first_act_month).d
t.days.values
      df['hist_purchase_recency'] = (act_date - pd.to_datetime(df['hist_transac_purchase
_date_max'])).dt.days.values
      df['new_purchase_recency'] = (act_date - pd.to_datetime(df['new_transac_purchase_d
ate_max'])).dt.days.values
```

```
  train_cols = [c for c in train_df.columns if c not in ['hist_transac_purchase_date_max', 'hist_transac_purchase_date_min', 'new_transac_purchase_date_max', 'new_transac_purchase_date_min',\
  'hist_purchase_date_last', 'hist_purchase_date_first', 'reference_month', 'hist_purchase_a_date_last', 'hist_purchase_a_date_first', 'new_purchase_date_last', 'new_purchase_date_first','card_id', 'first_active_month','first_active_month', 'target','outliers','feature_1','feature_2','feature_3','refernce_month','ref_first_month_diff_days']]
  target = train_df['target']
  outliers = train_df['outliers']
  card_id = train_df['card_id']
  del train_df['target']
  print('Completed')
  return train_df, train_cols
```

In [16]:

```
def fun_1(train):
  train = train
  train_df, train_cols = get_train(train)

  print('Loading pickle file...')
  path = '/content/drive/My Drive/case study/upload 15mis/lgb_final_323_tune.sav'
  import pickle
  with open(path, 'rb') as pickle_file:
      mod = pickle.load(pickle_file)
  predictions = mod.predict(train_df[train_cols])
  return predictions
```

In [25]:

```
train_df = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/train.csv')
```

In [23]:

```
def final_fun_2(train, target):
  train = train
  target = target
  predictions = fun_1(train=train)
  score = np.sqrt(mean_squared_error(predictions, target))
  print('Actual Value:', target)
  print('Predicted Value:', predictions)
  print('RMSE Score:', score)
```

In [26]:

```
from sklearn.metrics import mean_squared_error
final_fun_2(train=train_df, target=train_df['target'])
```

```
Started Preprocessing...
Preprocessing New Merchant Dataset...
Filled Missing values for new_merchant...
Working on Historical Transaction Dataset...
Filled missing values for historical transaction dataset...
Loading Merchant Dataset for preprocessing...
Mem. usage decreased to 30.32 Mb (46.0% reduction)
Filled missing values for merchant dataset...
One Hot Encoding the variables...
Completed...
Started Feature Engineering...
Completed FE of transaction info...
Completed FE of purchase amount...
Completed FE of purchase amount...
Feature Engineering Completed
Preparing train and test set...
Loading train and test
merge train and new features...
Completed
Loading pickle file...
RMSE Score: 3.6535296284973575
```

Model on a random single point from the train set

In [22]:

```
data = pd.read_csv('/content/drive/My Drive/case study/upload 15mis/train.csv')
train_df = data.sample(1)

from sklearn.metrics import mean_squared_error
final_fun_2(train=train_df, target=train_df['target'])
```

```
Started Preprocessing...
Preprocessing New Merchant Dataset...
Filled Missing values for new_merchant...
Working on Historical Transaction Dataset...
Filled missing values for historical transaction dataset...
Loading Merchant Dataset for preprocessing...
Mem. usage decreased to 30.32 Mb (46.0% reduction)
Filled missing values for merchant dataset...
One Hot Encoding the variables...
Completed...
Started Feature Engineering...
Completed FE of transaction info...
Completed FE of purchase amount...
Completed FE of purchase amount...
Feature Engineering Completed
Preparing train and test set...
Loading train and test
merge train and new features...
Completed
Loading pickle file...
Actual Value: 140372   -0.0333
Name: target, dtype: float64
Predicted Value: [-0.33539999]
RMSE Score: 0.3021004233466764
```