Loading libraries and Data Using Kaggle Api

In [ ]:

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

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

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import datetime
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
! pip install -q kaggle

from google.colab import files

files.upload()

! mkdir ~/.kaggle

! cp kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json

! kaggle competitions download -c elo-merchant-category-recommendation
```

Browse... No files selected.

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle (3).json
mkdir: cannot create directory '/root/.kaggle': File exists
Warning: Looks like you're using an outdated API Version, please cons
ider updating (server 1.5.10 / client 1.5.4)
merchants.csv.zip: Skipping, found more recently modified local copy
(use --force to force download)
sample_submission.csv.zip: Skipping, found more recently modified loc
al copy (use --force to force download)
train.csv.zip: Skipping, found more recently modified local copy (use
--force to force download)
Data%20Dictionary.xlsx: Skipping, found more recently modified local
copy (use --force to force download)
Data_Dictionary.xlsx: Skipping, found more recently modified local co
py (use --force to force download)
historical_transactions.csv.zip: Skipping, found more recently modifi
ed local copy (use --force to force download)
test.csv.zip: Skipping, found more recently modified local copy (use
--force to force download)
new_merchant_transactions.csv.zip: Skipping, found more recently modi
fied local copy (use --force to force download)
```

In [ ]:
```
# unzipping the dataset
! unzip '/content/historical_transactions.csv.zip'

! unzip '/content/merchants.csv.zip'

! unzip '/content/new_merchant_transactions.csv.zip'

! unzip '/content/train.csv.zip'

! unzip '/content/test.csv.zip'
```

```
Archive:  /content/historical_transactions.csv.zip
replace historical_transactions.csv? [y]es, [n]o, [A]ll, [N]one, [r]e
name: Archive:  /content/merchants.csv.zip
replace merchants.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: Archive:
/content/new_merchant_transactions.csv.zip
replace new_merchant_transactions.csv? [y]es, [n]o, [A]ll, [N]one,
[r]ename: Archive:  /content/train.csv.zip
replace train.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: Archive:  /c
ontent/test.csv.zip
replace test.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

In [ ]:
```
# https://www.kaggle.com/c/champs-scalar-coupling/discussion/96655
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float
64']
    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(n
p.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinf
o(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinf
o(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinf
o(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finf
o(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.fi
nfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)

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

    return df
```

In [ ]:
```
test = reduce_mem_usage(pd.read_csv('/content/test.csv'))
```

```
Memory usage after optimization is: 2.24 MB
```

In [ ]:
```
def train_features(train,test):

  # imputing missing values with mode
  # https://stackoverflow.com/questions/42789324/pandas-fillna-mode
  test['first_active_month'].fillna(test['first_active_month'].mode
()[0], inplace=True)

  # converting date features to datetime
  train['first_active_month']=pd.to_datetime(train['first_active_month
'])
  test['first_active_month']=pd.to_datetime(test['first_active_month'])

  # making a new columns with outliers as seen in eda notebook
  train['outliers'] = 0
  train.loc[train['target'] < -30, 'outliers'] = 1

  # https://www.geeksforgeeks.org/mean-encoding-machine-learning/
  # mean encoding categorical features by grouping them with outliers.
  for feature in ['feature_1', 'feature_2', 'feature_3']:
    mapping = train.groupby([feature])['outliers'].mean()
    train[feature] = train[feature].map(mapping)
    test[feature] = test[feature].map(mapping)

  # https://www.kaggle.com/mks2192/feature-engineering
  train['quarter']=train['first_active_month'].dt.quarter
  train['total_time'] = (datetime.datetime.today() - train['first_activ
e_month']).dt.days
  train['start_month'] = train['first_active_month'].dt.month
  train['start_year'] = train['first_active_month'].dt.year
  train['dayofweek'] = train['first_active_month'].dt.dayofweek
  train['quarter']=train['first_active_month'].dt.quarter

  train['total_time_feature1']=train['total_time']*train['feature_1']
  train['total_time_feature2']=train['total_time']*train['feature_2']
  train['total_time_feature3']=train['total_time']*train['feature_3']

  train['total_time_feature1_ratio']=(train['feature_1']/train['total_t
ime'])
  train['total_time_feature2_ratio']=(train['feature_2']/train['total_t
ime'])
  train['total_time_feature3_ratio']=(train['feature_3']/train['total_t
ime'])

  # getting aggregated features from categorical variables
  train['feature_sum'] = train['feature_1'] + train['feature_2'] + trai
n['feature_3']
  train['feature_mean'] = train['feature_sum']/3
  train['feature_max'] = train[['feature_1', 'feature_2', 'feature_3
']].max(axis=1)
  train['feature_min'] = train[['feature_1', 'feature_2', 'feature_3
']].min(axis=1)
  train['feature_var'] = train[['feature_1', 'feature_2', 'feature_3
']].std(axis=1)

  # https://www.kaggle.com/mks2192/feature-engineering
```

```
  test['quarter']=test['first_active_month'].dt.quarter
  test['total_time'] = (datetime.datetime.today() - test['first_active_
month']).dt.days
  test['start_month'] = test['first_active_month'].dt.month
  test['start_year'] = test['first_active_month'].dt.year
  test['dayofweek'] = test['first_active_month'].dt.dayofweek
  test['quarter']=test['first_active_month'].dt.quarter

  test['total_time_feature1']=test['total_time']*test['feature_1']
  test['total_time_feature2']=test['total_time']*test['feature_2']
  test['total_time_feature3']=test['total_time']*test['feature_3']

  test['total_time_feature1_ratio']=(test['feature_1']/test['total_time
'])
  test['total_time_feature2_ratio']=(test['feature_2']/test['total_time
'])
  test['total_time_feature3_ratio']=(test['feature_3']/test['total_time
'])

  # getting aggregated features from categorical variables
  test['feature_sum'] = test['feature_1'] + test['feature_2'] + test['f
eature_3']
  test['feature_mean'] = test['feature_sum']/3
  test['feature_max'] = test[['feature_1', 'feature_2', 'feature_3']].m
ax(axis=1)
  test['feature_min'] = test[['feature_1', 'feature_2', 'feature_3']].m
in(axis=1)
  test['feature_var'] = test[['feature_1', 'feature_2', 'feature_3']].s
td(axis=1)

  gc.collect()

  return train,test
```

In [ ]:
```
def hist_features(hist_trans):

  # preprocessing the csv file
  # imputing the missing values
  hist_trans['category_3'].fillna(hist_trans['category_3'].mode()[0], i
nplace=True)
  hist_trans['merchant_id'].fillna(hist_trans['merchant_id'].mode()[0],
inplace=True)
  hist_trans['category_2'].fillna(hist_trans['category_2'].mode()[0], i
nplace=True)

  # mapping catrgorical variables
  hist_trans['authorized_flag'] = hist_trans['authorized_flag'].map({'Y
':1, 'N':0})
  hist_trans['category_1'] = hist_trans['category_1'].map({'Y':1, 'N':
0})
  hist_trans['category_3'] = hist_trans['category_3'].map({'A':0, 'B':
1, 'C':2})
  hist_trans['installments'] = hist_trans['installments'].map({-1:13,
0:0.1,1:1,2:2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,10:10,11:11,12:12,999:13})

  # taking 99 percrntile value a max to remove the outliers
  hist_trans['purchase_amount'] = hist_trans['purchase_amount'].apply(l
ambda x: min(x, 1.22))

  # feature engineering based on dates
  hist_trans['purchase_date']=pd.to_datetime(hist_trans['purchase_date
'])
  hist_trans['year'] = hist_trans['purchase_date'].dt.year
  hist_trans['day'] = hist_trans['purchase_date'].dt.day
  hist_trans['month'] = hist_trans['purchase_date'].dt.month
  hist_trans['dayofweek'] = hist_trans['purchase_date'].dt.dayofweek
  hist_trans['weekofyear'] = hist_trans['purchase_date'].dt.weekofyear
  hist_trans['hour_of_purchase'] = hist_trans['purchase_date'].dt.hour
  hist_trans['Minute_of_purchase'] = hist_trans['purchase_date'].dt.min
ute
  hist_trans['Second_of_purchase'] = hist_trans['purchase_date'].dt.sec
ond
  hist_trans['purchased_on_weekend'] = (hist_trans.dayofweek >=5).astyp
e(int)
  hist_trans['purchased_on_weekday'] = (hist_trans.dayofweek <5).astype
(int)

  hist_trans['month_diff'] = ((datetime.datetime.today() - hist_trans['
purchase_date']).dt.days)//30
  hist_trans['month_diff'] += hist_trans['month_lag']

  # feature engineering based on installments and purchase amount
  # purchase amount is highly normalized so we denormalizing it
  # inspired from https://chandureddyvari.com/posts/elo-merchant-featur
e/
  hist_trans['EMI'] = hist_trans['purchase_amount'] / hist_trans['insta
llments']
  hist_trans['purchase_amount_quantiles'] = pd.qcut(hist_trans['purchas
e_amount'], 5, labels=False)
```

```
  hist_trans['duration'] = hist_trans['purchase_amount']*hist_trans['mo
nth_diff']
  hist_trans['amount_month_ratio'] = hist_trans['purchase_amount']/hist
_trans['month_diff']

  hist_trans = reduce_mem_usage(hist_trans)

  # aggregating by grouping them by card_id.
  aggregations = {
    'purchase_date' : ['max','min'],
    'purchased_on_weekend': ['sum', 'mean'],
    'purchased_on_weekday': ['sum', 'mean'],
    'dayofweek' : ['nunique', 'sum', 'mean','max'],
    'hour_of_purchase': ['nunique', 'mean', 'min', 'max'],
    'Minute_of_purchase': ['nunique', 'mean', 'min', 'max'],
    'Second_of_purchase': ['nunique', 'mean', 'min', 'max'],
    'weekofyear': ['nunique', 'mean', 'min', 'max'],
    'month_diff': ['max','min','mean','var','skew'],
    'day': ['nunique', 'sum', 'min'],
    'month' : ['sum', 'mean', 'nunique','max'],
    'purchase_amount_quantiles' : ['var', 'mean', 'skew'],
    'duration' : ['mean','min','max','var','skew'],
    'amount_month_ratio' : ['mean','min','max','var','skew'],
    'authorized_flag' : ['sum','mean'],
    'subsector_id': ['nunique'],
    'card_id': ['size'],
    'city_id' : ['nunique'],
    'state_id' : ['nunique'],
    'merchant_id': ['nunique'],
    'installments': ['sum','max','mean','var','skew'],
    'merchant_category_id': ['nunique'],
    'purchase_amount': ['sum', 'mean', 'min', 'max', 'var','skew'],
    'EMI' : ['sum','mean','max','min','var'],
    'category_1' : ['sum','mean', 'max','min'],
    'category_2' : ['sum','mean'],
    'category_3' : ['sum','mean'],
    'month_lag' : ['sum','max','min','mean','var','skew']
  }
  aggregated_trans = hist_trans.groupby('card_id').agg(aggregations)
  aggregated_trans.columns = ['transactions_'+'_'.join(col).strip()
                             for col in aggregated_trans.columns.values]
  aggregated_trans.reset_index(inplace=True)

  # extracting some more features based on aggregated features.
  aggregated_trans['transactions_purchase_date_diff'] = (aggregated_tra
ns['transactions_purchase_date_max']-aggregated_trans['transactions_pur
chase_date_min']).dt.days
  aggregated_trans['transactions_purchase_date_average'] = aggregated_t
rans['transactions_purchase_date_diff']/aggregated_trans['transactions_
card_id_size']
  aggregated_trans['transactions_purchase_date_uptonow'] = (datetime.da
tetime.today()-aggregated_trans['transactions_purchase_date_max']).dt.d
ays
  aggregated_trans['transactions_purchase_date_uptomin'] = (datetime.da
tetime.today()-aggregated_trans['transactions_purchase_date_min']).dt.d
ays
```

```
    gc.collect()
    return aggregated_trans
```

In [ ]:
```
def get_new_trans_features(new_hist_trans):

  # preprocessing the csv file
  # imputing the missing values
  new_hist_trans['category_3'].fillna(new_hist_trans['category_3'].mode
()[0], inplace=True)
  new_hist_trans['merchant_id'].fillna(new_hist_trans['merchant_id'].mo
de()[0], inplace=True)
  new_hist_trans['category_2'].fillna(new_hist_trans['category_2'].mode
()[0], inplace=True)

  # mapping catrgorical variables
  new_hist_trans['authorized_flag'] = new_hist_trans['authorized_flag
'].map({'Y':1, 'N':0})
  new_hist_trans['category_1'] = new_hist_trans['category_1'].map({'Y':
1, 'N':0})
  new_hist_trans['category_3'] = new_hist_trans['category_3'].map({'A':
0, 'B':1, 'C':2})
  new_hist_trans['installments'] = new_hist_trans['installments'].map
({-1:13, 0:0.1,1:1,2:2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,10:10,11:11,12:12,99
9:13})

  # taking 99 percrntile value a max to remove the outliers
  new_hist_trans['purchase_amount'] = new_hist_trans['purchase_amount
'].apply(lambda x: min(x, 1.22))

  # feature engineering based on dates
  new_hist_trans['purchase_date']=pd.to_datetime(new_hist_trans['purcha
se_date'])
  new_hist_trans['year'] = new_hist_trans['purchase_date'].dt.year
  new_hist_trans['day'] = new_hist_trans['purchase_date'].dt.day
  new_hist_trans['month'] = new_hist_trans['purchase_date'].dt.month
  new_hist_trans['dayofweek'] = new_hist_trans['purchase_date'].dt.dayo
fweek
  new_hist_trans['weekofyear'] = new_hist_trans['purchase_date'].dt.wee
kofyear
  new_hist_trans['hour_of_purchase'] = new_hist_trans['purchase_date'].
dt.hour
  new_hist_trans['Minute_of_purchase'] = new_hist_trans['purchase_date
'].dt.minute
  new_hist_trans['Second_of_purchase'] = new_hist_trans['purchase_date
'].dt.second
  new_hist_trans['purchased_on_weekend'] = (new_hist_trans.dayofweek >=
5).astype(int)
  new_hist_trans['purchased_on_weekday'] = (new_hist_trans.dayofweek
<5).astype(int)

  new_hist_trans['month_diff'] = ((datetime.datetime.today() - new_hist
_trans['purchase_date']).dt.days)//30
  new_hist_trans['month_diff'] += new_hist_trans['month_lag']

  # feature engineering based on installments and purchase amount
  # purchase amount is highly normalized so we denormalizing it
  # inspired from https://chandureddyvari.com/posts/elo-merchant-featur
e/
```

```
  new_hist_trans['EMI'] = new_hist_trans['purchase_amount'] / new_hist_
trans['installments']
  new_hist_trans['purchase_amount_quantiles'] = pd.qcut(new_hist_trans
['purchase_amount'], 5, labels=False)
  new_hist_trans['duration'] = new_hist_trans['purchase_amount']*new_hi
st_trans['month_diff']
  new_hist_trans['amount_month_ratio'] = new_hist_trans['purchase_amoun
t']/new_hist_trans['month_diff']

  new_hist_trans = reduce_mem_usage(new_hist_trans)

  # aggregating by grouping them by card_id.
  aggregations = {
    'purchase_date' : ['max','min'],
    'purchased_on_weekend': ['sum', 'mean'],
    'purchased_on_weekday': ['sum', 'mean'],
    'dayofweek' : ['nunique', 'sum', 'mean','max'],
    'hour_of_purchase': ['nunique', 'mean', 'min', 'max'],
    'Minute_of_purchase': ['nunique', 'mean', 'min', 'max'],
    'Second_of_purchase': ['nunique', 'mean', 'min', 'max'],
    'weekofyear': ['nunique', 'mean', 'min', 'max'],
    'month_diff': ['max','min','mean','var','skew'],
    'day': ['nunique', 'sum', 'min'],
    'month' : ['sum', 'mean', 'nunique','max'],
    'purchase_amount_quantiles' : ['var', 'mean', 'skew'],
    'duration' : ['mean','min','max','var','skew'],
    'amount_month_ratio' : ['mean','min','max','var','skew'],
    'subsector_id': ['nunique'],
    'card_id': ['size'],
    'city_id' : ['nunique'],
    'state_id' : ['nunique'],
    'merchant_id': ['nunique'],
    'installments': ['sum','max','mean','var','skew'],
    'merchant_category_id': ['nunique'],
    'purchase_amount': ['sum', 'mean', 'min', 'max', 'var','skew'],
    'EMI' : ['sum','mean','max','min','var'],
    'category_1' : ['sum','mean', 'max','min'],
    'category_2' : ['sum','mean'],
    'category_3' : ['sum','mean'],
    'month_lag' : ['sum','max','min','mean','var','skew']
  }

  aggregated_trans_1 = new_hist_trans.groupby('card_id').agg(aggregatio
ns)
  aggregated_trans_1.columns = ['new_transactions_'+'_'.join(col).strip
()
                              for col in aggregated_trans_1.columns.value
s]
  aggregated_trans_1.reset_index(inplace=True)

  # extracting some more features based on aggregated features.
  aggregated_trans_1['new_transactions_purchase_date_diff'] = (aggregat
ed_trans_1['new_transactions_purchase_date_max']-aggregated_trans_1['ne
w_transactions_purchase_date_min']).dt.days
  aggregated_trans_1['new_transactions_purchase_date_average'] = aggreg
ated_trans_1['new_transactions_purchase_date_diff']/aggregated_trans_1
```

```
['new_transactions_card_id_size']
  aggregated_trans_1['new_transactions_purchase_date_uptonow'] = (datet
ime.datetime.today()-aggregated_trans_1['new_transactions_purchase_date
_max']).dt.days
  aggregated_trans_1['new_transactions_purchase_date_uptomin'] = (datet
ime.datetime.today()-aggregated_trans_1['new_transactions_purchase_date
_min']).dt.days

  return aggregated_trans_1
```

In [ ]:
```
def get_train_features(train):

  print('loading data ........')
  test = reduce_mem_usage(pd.read_csv('/content/test.csv'))
  hist_trans = reduce_mem_usage(pd.read_csv('/content/historical_transa
ctions.csv'))
  new_hist_trans = reduce_mem_usage(pd.read_csv('/content/new_merchant_
transactions.csv'))
  print('processing train......')

  train1,test1 = train_features(train,test)

  print('processing historical transactions.....')
  aggregated_trans = hist_features(hist_trans)
  print('processing new merchant transactions.....')
  aggregated_trans_1 = get_new_trans_features(new_hist_trans)
  print('merging all files together......')
  train1=pd.merge(train1, aggregated_trans, on='card_id', how='left')
  train1=pd.merge(train1, aggregated_trans_1, on='card_id', how='left')

  print('creating some more new features from merged file......')
  # converting engineered date features to datetime so that we can use
them afterwards.
  train1['transactions_purchase_date_max'] = pd.to_datetime(train1['tra
nsactions_purchase_date_max'])
  train1['transactions_purchase_date_min'] = pd.to_datetime(train1['tra
nsactions_purchase_date_min'])
  train1['new_transactions_purchase_date_max'] = pd.to_datetime(train1
['new_transactions_purchase_date_max'])
  train1['new_transactions_purchase_date_min'] = pd.to_datetime(train1
['new_transactions_purchase_date_min'])

  # extracting some more features from train by performing some simple
caluculations.
  # inspired by https://www.kaggle.com/mfjwr1/simple-lightgbm-without-b
lending
  train1['transactions_purchase_date_difference']=train1['transactions_
purchase_date_max'] - train1['transactions_purchase_date_min']
  train1['new_transactions_purchase_date_difference'] = train1['new_tra
nsactions_purchase_date_max'] - train1['new_transactions_purchase_date_
min']
  train1['Avg_purchase'] = train1['transactions_purchase_date_differenc
e'] / train1['transactions_card_id_size']
  train1['new_Avg_purchase'] = train1['new_transactions_purchase_date_d
ifference'] / train1['new_transactions_card_id_size']
  train1['last_purchase_from_now'] = (datetime.datetime.today() - train
1['transactions_purchase_date_max']).dt.days
  train1['new_last_purchase_from_now'] = (datetime.datetime.today() - t
rain1['new_transactions_purchase_date_max']).dt.days
  train1['first_purchase_from_now'] = (datetime.datetime.today() - trai
n1['transactions_purchase_date_min']).dt.days
  train1['new_first_purchase_from_now'] = (datetime.datetime.today() -
train1['new_transactions_purchase_date_min']).dt.days

  train1['card_id_total'] = train1['new_transactions_card_id_size']+tra
```

```
in1['transactions_card_id_size']
  train1['card_id_ratio'] =  train1['new_transactions_card_id_size']/tr
ain1['transactions_card_id_size']

  train1['total_purchase_amount_max'] = train1['new_transactions_purcha
se_amount_max']+train1['transactions_purchase_amount_max']
  train1['total_purchase_amount_min'] = train1['new_transactions_purcha
se_amount_min']+train1['transactions_purchase_amount_min']
  train1['total_purchase_amount_mean'] = train1['new_transactions_purch
ase_amount_mean']+train1['transactions_purchase_amount_mean']
  train1['total_purchase_amount_sum'] = train1['new_transactions_purcha
se_amount_sum']+train1['transactions_purchase_amount_sum']
  train1['total_purchase_amount_ratio'] = train1['new_transactions_purc
hase_amount_sum']/train1['transactions_purchase_amount_sum']

  train1['total_installments_max'] = train1['new_transactions_installme
nts_max'] + train1['transactions_installments_max']
  train1['total_installments_mean'] = train1['new_transactions_installm
ents_mean'] + train1['transactions_installments_mean']
  train1['total_installments_sum'] = train1['new_transactions_installme
nts_sum'] + train1['transactions_installments_sum']
  train1['total_installments_ratio'] = train1['new_transactions_install
ments_sum'] / train1['transactions_installments_sum']

  train1['total_month_lag_max'] = train1['new_transactions_month_lag_ma
x'] + train1['transactions_month_lag_max']
  train1['total_month_lag_min'] = train1['new_transactions_month_lag_mi
n'] + train1['transactions_month_lag_min']
  train1['total_month_lag_mean'] = train1['new_transactions_month_lag_m
ean'] + train1['transactions_month_lag_mean']
  train1['total_month_lag_sum'] = train1['new_transactions_month_lag_su
m'] + train1['transactions_month_lag_sum']
  train1['total_month_lag_ratio'] = train1['new_transactions_month_lag_
sum'] / train1['transactions_month_lag_sum']

  train1['total_duration_max'] = train1['new_transactions_duration_max
'] + train1['transactions_duration_max']
  train1['total_duration_min'] = train1['new_transactions_duration_min
'] + train1['transactions_duration_min']
  train1['total_duration_mean'] = train1['new_transactions_duration_mea
n'] + train1['transactions_duration_mean']


  train1['total_month_diff_max'] = train1['new_transactions_month_diff_
max'] + train1['transactions_month_diff_max']
  train1['total_month_diff_mean'] = train1['new_transactions_month_diff
_mean'] + train1['transactions_month_diff_mean']
  train1['total_month_diff_min'] = train1['new_transactions_month_diff_
min'] + train1['transactions_month_diff_min']

  train1['total_amount_month_ratio_max'] = train1['new_transactions_amo
unt_month_ratio_max'] + train1['transactions_amount_month_ratio_max']
  train1['total_amount_month_ratio_min'] = train1['new_transactions_amo
unt_month_ratio_min'] + train1['transactions_amount_month_ratio_min']
  train1['total_amount_month_ratio_mean'] = train1['new_transactions_am
ount_month_ratio_mean'] + train1['transactions_amount_month_ratio_mean
```

```
']

  train1['customer_rating'] = train1['transactions_card_id_size'] * tra
in1['transactions_purchase_amount_sum'] / train1['transactions_month_di
ff_mean']
  train1['new_customer_rating'] = train1['new_transactions_card_id_size
'] * train1['new_transactions_purchase_amount_sum'] / train1['new_trans
actions_month_diff_mean']
  train1['customer_rating_ratio'] = train1['customer_rating'] / train1
['new_customer_rating']

  print('preprocessing final data........')
  # replacing inf values with nan.
  train1.replace([-np.inf,np.inf], np.nan, inplace=True)

  # checking for nan values.
  k= train1.columns[train1.isna().any()]

  # imputing using mode
  for i in range(len(k)):
    train1[k[i]].fillna(train1[k[i]].mode()[0], inplace=True)

  # getting columns having datetime64[ns] datatypes
  types=train1.select_dtypes(include=['datetime64[ns]']).columns

  # removing columns having datetime64[ns] datatypes
  train1=train1.drop(types,axis=1)

  # getting columns having timedelta64[ns] datatypes
  types1=train1.select_dtypes(include=['timedelta64[ns]']).columns

  # changing timedelta64[ns] to int64 datatype so that we can perform m
odelling
  for i in types1:
    train1[i] = train1[i].astype(np.int64) * 1e-9

  k = [i for i in range(201917)]
  train1.insert(loc=0, column='Unnamed', value=k)

  train1_cols = [c for c in train1.columns if c not in ['card_id','targ
et','outliers']]

  return train1[train1_cols]
```

Final Function1

In [ ]:

```
def fun1(train):
  train_features = get_train_features(train)
  filename = '/content/drive/MyDrive/Colab Notebooks/CASE_STUDY_1/123'
  lgbm = pd.read_pickle(filename)
  predictions = lgbm.predict(train_features)
  return predictions
```

Final Function2

In [ ]:
```
def fun2(train,target):
  predictions = fun1(train)
  print('predicted score is : {}'.format(predictions))
  actual = target
  rmse = mean_squared_error(predictions, actual)**0.5
  print('rmse value for is : {}'.format(rmse))
```

In [ ]:
```
train = reduce_mem_usage(pd.read_csv('/content/train.csv'))
```

```
Memory usage after optimization is: 4.04 MB
```

In [ ]:
```
%%time
fun2(train,train['target'])
```

```
loading data ........
Memory usage after optimization is: 2.24 MB
Memory usage after optimization is: 1749.11 MB
Memory usage after optimization is: 114.20 MB
processing train......
processing historical transactions.....
Memory usage after optimization is: 1638.06 MB
processing new merchant transactions.....
Memory usage after optimization is: 110.45 MB
merging all files together......
creating some more new features from merged file......
preprocessing final data........
rmse value for is : 3.415138445872954
CPU times: user 16min 3s, sys: 20.5 s, total: 16min 24s
Wall time: 15min 58s
```