Elo Data Processing and Feature Engineering

This notebook focuses on processing the Elo merchant category recommendation dataset and engineer features for the model.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from tqdm import tqdm
import pickle
import datetime
import os
import warnings
warnings.simplefilter("ignore")
```

Function to reduce the memory usage by any pandas dataframe variable.

In [2]:
```
#https://www.kaggle.com/c/champs-scalar-coupling/discussion/96655
def reduce_mem_usage(df, verbose=True):

    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) / s
    return df
```

Function for one hot encoding categorical columns of a dataframe.

In [3]:
```
def onehotencoder(df, columns):
    """This function performs one hot encoding on categorical columns in a dataset and concat
    those encoded columns to the dataset and drops the original categorical columns. It takes
    dataset as dataframe object and categorical column names as list for input."""

    for col in columns:
        dummy = pd.get_dummies(df[col], prefix = col)
        df = pd.concat([df, dummy], axis = 1)
        df.drop(col, axis = 1, inplace = True)

    return df
```

Function to perform aggregate featurization.

In [4]:
```
def agg_featurization(df, groupby, agg_dict, prefix = ""):
    """This function performs aggregation on a dataframe and returns the aggregate features
    dataframe. It takes dataset as dataframe, groupby columns on which aggregate has to be performed
    as list, aggregate functions to be performed on columns as dictionary and prefix to be added to
    aggregated feature column name."""

    agg_df = df.groupby(groupby).agg(agg_dict)
    if prefix != "":
        agg_df.columns = [prefix + '_' + '_'.join(col) for col in agg_df.columns.values]
    else:
        agg_df.columns = ['_'.join(col) for col in agg_df.columns.values]
    agg_df.reset_index(inplace = True)

    return agg_df
```

Function to perform successive aggregate featurization.

In [5]:
```
def successive_agg_featurization(df, groupby1, groupby2, columns, agg_dict, prefix = ""):
    """This function performs successive aggregation on a dataframe and returns the successive aggregate
    features dataframe. It takes dataset as dataframe, groupby1 and groupby2 on which aggregate
    has to be performed as strings, columns on which the aggregate function is to be performed and
    aggregate functions to be performed on columns as dictionary."""

    intermediate_agg_df = df.groupby([groupby1, groupby2])[columns].mean()
    successive_agg_df = agg_featurization(intermediate_agg_df, groupby1, agg_dict, prefix = prefix + "_" + grou

    return successive_agg_df
```

Loading the Datasets

In [6]:
```
train = pd.read_csv("data/train.csv", parse_dates = ['first_active_month'])
train = reduce_mem_usage(train)
test = pd.read_csv("data/test.csv", parse_dates = ['first_active_month'])
test = reduce_mem_usage(test)
```

```
Mem. usage decreased to  4.04 Mb (56.2% reduction)
Mem. usage decreased to  2.24 Mb (52.5% reduction)
```

In [7]:
```
historical_transactions = pd.read_csv("data/historical_transactions.csv", parse_dates = ['purchase_date'],
                                      dtype = {"card_id" : "category"})
historical_transactions = reduce_mem_usage(historical_transactions)
new_transactions = pd.read_csv("data/new_merchant_transactions.csv", parse_dates = ['purchase_date'],
                                      dtype = {"card_id" : "category"})
new_transactions = reduce_mem_usage(new_transactions)
```

```
Mem. usage decreased to 1648.61 Mb (45.2% reduction)
Mem. usage decreased to 116.98 Mb (44.9% reduction)
```

Processing Dataset

Imputing test first_active_month missing value.

We will find all the test observations having similar feature_1, feature_2 and feature_3 values as the missing first_active_month observation and impute it with mode of the first_active_month of similar observations.

In [8]:
```
test_null = test[test['first_active_month'].isnull()]
test_null
```

Out[8]:

	first_active_month	card_id	feature_1	feature_2	feature_3
11578	NaT	C_ID_c27b4f80f7	5	2	1

In [9]:
```
test_similar = test[(test.feature_1 == test_null.feature_1.values[0]) & (test.feature_2 == test_null.feature_2.
                    & (test.feature_3 == test_null.feature_3.values[0])]
test.first_active_month[test['first_active_month'].isnull()] = test_similar['first_active_month'].mode()[0]
test.isnull().sum()
```

Out[9]:
```
first_active_month    0
card_id               0
feature_1             0
feature_2             0
feature_3             0
dtype: int64
```

In [10]:
```
del test_null
del test_similar
```

In [11]:
```
train.to_csv("data/train_processed.csv", index = False)
test.to_csv("data/test_processed.csv", index = False)
```

Label encoding the authorized_flag, category_1 and category_3 columns of historical transactions and new transactions.

In [12]:
```
historical_transactions['authorized_flag'] = historical_transactions['authorized_flag'].map({'Y':1, 'N':0}).ast
historical_transactions['category_1'] = historical_transactions['category_1'].map({'Y':1, 'N':0}).astype(np.int
historical_transactions['category_3'] = historical_transactions['category_3'].map({'A':0, 'B':1, 'C':2})
```

In [13]:
```
new_transactions['authorized_flag'] = new_transactions['authorized_flag'].map({'Y':1, 'N':0}).astype(np.int8)
new_transactions['category_1'] = new_transactions['category_1'].map({'Y':1, 'N':0}).astype(np.int8)
new_transactions['category_3'] = new_transactions['category_3'].map({'A':0, 'B':1, 'C':2})
```

Imputing merchant_id missing values of historical transactions and new transactions.

We find 'merchant_ids' in merchants having similar 'merchant_category_id', 'subsector_id' and 'city_id 'as null observations and impute null observations with the mode of similar 'merchant_ids'. We are not considering 'state_id' as any two similar 'city_id' will have same 'state_id'. After imputing 'merchant_ids' from merchant dataset, we still have null values in 'merchant_ids', so for the remaining null values we will simply impute 'NAN'.

In [14]:
```
def impute_merchant_id(df):
    """This function imputes the null merchant ids in historical and new transactions. It takes takes
    transaction dataset as dataframe and returns the transaction dataframe after imputing."""

    Merchants_Categorical_Columns = ["merchant_category_id", "subsector_id", "city_id", "state_id"]
    Merchants_Categorical_Dtypes = {col: "category" for col in Merchants_Categorical_Columns}
    merchants = pd.read_csv("data/merchants.csv", dtype = Merchants_Categorical_Dtypes)

    df_null = df[df['merchant_id'].isnull()]
    df_null_index = df_null.index
    for idx in tqdm(df_null_index):
        df_similar = merchants[(merchants.merchant_category_id == df_null.merchant_category_id.loc[idx]) &
                               (merchants.subsector_id == df_null.subsector_id.loc[idx]) &
                               (merchants.city_id == df_null.city_id.loc[idx])]
        if df_similar.shape[0] != 0:
            df.merchant_id.loc[idx] = df_similar['merchant_id'].mode()[0]
        del df_similar
    del df_null
    del merchants
    df['merchant_id'].fillna('NAN', inplace = True)
    df['merchant_id'] = df['merchant_id'].astype('category')

    return df
```

In [15]:
```
historical_transactions = impute_merchant_id(historical_transactions)
```

```
100%|████████████████████████████████████████████████████████████| 138481/138481 [02:25<00:00, 95
3.66it/s]
```

In [16]:
```
new_transactions = impute_merchant_id(new_transactions)
```

```
100%|████████████████████████████████████████████████████████████| 26216/26216 [00:28<00:00, 91
1.37it/s]
```

Imputing category_2 and category_3 missing values of historical transactions and new transactions.

We are imputing null values in category_2 and category_3 columns of historical and new transactions by training classifier models from non null columns of these transactions. We will use these classifier models to predict the null values in category_2 and category_3 of historical and new transactions.

In [17]:
```
def impute_category(df, null_columns, train_columns, model_prefix):
    """This function imputes the null category columns of historical and new transactions
    by training classifier model from non null columns. It takes transaction as dataframe,
    categorical columns with null values as list, non null columns as list and prefix for
    the saved model as string for input."""

    for col in null_columns:
        test_df = df.loc[df[col].isna()][train_columns]
        train_df = df.loc[df[col].notna()][train_columns]
        train_y = df.loc[df[col].notna()][col]

        path = 'data/' + model_prefix + '_' + str(col) + '_model'
        if os.path.exists(path):
            clf = pickle.load(open(path, 'rb'))
        else:
            print("Training model to impute ", col)
            clf = LogisticRegression()
            clf.fit(train_df, train_y)
            pickle.dump(clf, open(path, 'wb'))
        print("Imputing predicted category from model to null values in ", col)
        df.loc[df[col].isna(), col] = clf.predict(test_df)
        df[col] = df[col].astype(np.int8)
        del train_df
        del test_df
        del train_y

    return df
```

In [18]:
```
null_columns = ['category_2', 'category_3']
train_columns = ['authorized_flag', 'category_1', 'installments', 'month_lag', 'purchase_amount',
                 'merchant_category_id', 'subsector_id', 'city_id', 'state_id']
```

In [19]:
```
historical_transactions = impute_category(historical_transactions, null_columns, train_columns, 'historical')
```

```
Imputing predicted category from model to null values in  category_2
Imputing predicted category from model to null values in  category_3
```

In [20]:
```
historical_transactions.isnull().sum()
```

Out[20]:
```
authorized_flag         0
card_id                 0
city_id                 0
category_1              0
installments            0
category_3              0
merchant_category_id    0
merchant_id             0
month_lag               0
purchase_amount         0
purchase_date           0
category_2              0
state_id                0
subsector_id            0
dtype: int64
```

In [21]:
```
new_transactions = impute_category(new_transactions, null_columns, train_columns, 'new')
```

```
Imputing predicted category from model to null values in  category_2
Imputing predicted category from model to null values in  category_3
```

In [22]:
```
new_transactions.isnull().sum()
```

Out[22]:
```
authorized_flag         0
card_id                 0
city_id                 0
category_1              0
installments            0
category_3              0
merchant_category_id    0
merchant_id             0
month_lag               0
purchase_amount         0
purchase_date           0
category_2              0
state_id                0
subsector_id            0
dtype: int64
```

De-anonymizing purchase amount in historical transactions and new transactions.

It should be natural to expect the values of purchase amount to be positive which is obviously not the case here. We are using insights provided by Raddar in his notebook in Kaggle for de-anonymizing the purchase amount and transforming the purchase amount into it's observed value.

In [23]:
```
#https://www.kaggle.com/code/raddar/towards-de-anonymizing-the-data-some-insights/notebook
historical_transactions['purchase_amount'] = ((historical_transactions['purchase_amount'].astype(np.float64) /
                                              + 497.06)
new_transactions['purchase_amount'] = ((new_transactions['purchase_amount'].astype(np.float64) / 0.00150265118)
```

Encoding categorical columns of historical transactions and new transactions.

We will one hot encode the categorical columns in historical transactions and new transactions.

In [24]:
```
categorical_columns = ['category_1', 'category_2', 'category_3']
historical_transactions = onehotencoder(historical_transactions, categorical_columns)
new_transactions = onehotencoder(new_transactions, categorical_columns)
```

In [25]:
```
historical_transactions.to_csv("data/historical_transactions_processed.csv", index = False)
new_transactions.to_csv("data/new_transactions_processed.csv", index = False)
```

In [26]:
```
historical_transactions.head()
```

Out[26]:

	authorized_flag	card_id	city_id	installments	merchant_category_id	merchant_id	month_lag	purchase_amount	purchase_date
0	1	C_ID_4e6213e9bc	88	0	80	M_ID_e020e9b302	-8	28.999985	2017-06-25 15:33:07
1	1	C_ID_4e6213e9bc	88	0	367	M_ID_86ec983688	-7	9.169997	2017-07-15 12:10:45
2	1	C_ID_4e6213e9bc	88	0	80	M_ID_979ed661fc	-6	17.649983	2017-08-09 22:04:29
3	1	C_ID_4e6213e9bc	88	0	560	M_ID_e6d5ae8ea6	-5	7.690007	2017-09-02 10:06:26
4	1	C_ID_4e6213e9bc	88	0	80	M_ID_e020e9b302	-11	15.999983	2017-03-10 01:14:19

5 rows × 21 columns

In [27]:
```
new_transactions.head()
```

Out[27]:

	authorized_flag	card_id	city_id	installments	merchant_category_id	merchant_id	month_lag	purchase_amount	purchase_date
0	1	C_ID_415bb3a509	107	1	307	M_ID_b0c793002c	1	125.971091	2018-03-11 14:57:36
1	1	C_ID_415bb3a509	140	1	307	M_ID_88920c89e8	1	118.172375	2018-03-19 18:53:37
2	1	C_ID_415bb3a509	330	1	507	M_ID_ad5237ef6b	2	130.195395	2018-04-26 14:08:44
3	1	C_ID_415bb3a509	-1	1	661	M_ID_9e84cda3b1	1	49.933608	2018-03-07 09:43:21
4	1	C_ID_ef55cf8d4b	-1	1	166	M_ID_3c86fa3831	1	58.057271	2018-03-22 21:07:53

5 rows × 21 columns

Feature Engineering

Performing featurization on train and test.

We will use the reference date of 1/2/2018 to calculate the elapsed time for each card id. The elapsed time feature will indicate the number of days, the cardholder has been using the card. We will also divide first active month column into first active year and first active month categorical columns. First active year will denote the year and first active month will denote the month, the cardholder started using the card.

In [28]:
```
train['elapsed_time'] = (datetime.date(2018, 2, 1) - train['first_active_month'].dt.date).dt.days
train['first_active_year'] = train['first_active_month'].dt.year
train['first_active_month'] = train['first_active_month'].dt.month

test['elapsed_time'] = (datetime.date(2018, 2, 1) - test['first_active_month'].dt.date).dt.days
test['first_active_year'] = test['first_active_month'].dt.year
test['first_active_month'] = test['first_active_month'].dt.month
```

Adding outlier feature column to train set.

We will be adding outlier identification column to train set. The outlier columns will be 1 for card_ids having outlier value target and 0 for remaining card_ids. This outlier column will be used for stratified splitting of train set during model training.

In [29]:
```
train['outlier'] = 0
train['outlier'][train['target'] > 30] = 1
```

In [30]:
```
train.to_csv("data/train_featurized.csv", index = False)
test.to_csv("data/test_featurized.csv", index = False)
```

In [31]:
```
train.head()
```

Out[31]:

	first_active_month	card_id	feature_1	feature_2	feature_3	target	elapsed_time	first_active_year	outlier
0	6	C_ID_92a2005557	5	2	1	-0.820312	245	2017	0
1	1	C_ID_3d0044924f	4	1	0	0.392822	396	2017	0
2	8	C_ID_d639edf6cd	2	2	0	0.687988	549	2016	0
3	9	C_ID_186d6a6901	4	3	0	0.142456	153	2017	0
4	11	C_ID_cdbd2c0db2	1	3	0	-0.159790	92	2017	0

In [32]:
```
test.head()
```

Out[32]:

	first_active_month	card_id	feature_1	feature_2	feature_3	elapsed_time	first_active_year
0	4	C_ID_0ab67a22ab	3	3	1	306	2017
1	1	C_ID_130fd0cbdd	2	3	0	396	2017
2	8	C_ID_b709037bc5	5	1	1	184	2017
3	12	C_ID_d27d835a9f	2	1	0	62	2017
4	12	C_ID_2b5e3df5c2	5	1	1	793	2015

Performing featurization on historical and new transactions.

Engineering new features from purchase date.

We will engineer new columns from purchase date such as purchase year, month, weekday, date etc. These columns will be used to generate time related features.

In [33]:
```
def date_featurization(df, column):
    """This function featurize the date column of a dataframe by engineering
    new features such as year, month, day, hour etc. It takes the dataset as
    dataframe and date column as string for input and returns the dataframe
    with added features."""

    df['year'] = df[column].dt.year
    df['month'] = df[column].dt.month
    df['dayofweek'] = df[column].dt.dayofweek
    df['date'] = df[column].dt.day
    df['hour'] = df[column].dt.hour
    df['weekend'] = 0
    df['weekend'][df['dayofweek'] >= 5] = 1

    return df
```

In [34]:
```
historical_transactions = date_featurization(historical_transactions, 'purchase_date')
new_transactions = date_featurization(new_transactions, 'purchase_date')
```

Engineering count of historical and new transactions features.

This feature will indicate the number of historical and new transactions done by each card id.

In [35]:
```
hist_transactions_features = historical_transactions.groupby(['card_id']).size().reset_index()
hist_transactions_features.columns = ['card_id', 'hist_transc_count']

new_transactions_features = new_transactions.groupby(['card_id']).size().reset_index()
new_transactions_features.columns = ['card_id', 'new_transc_count']
```

Perfoming aggregate featurization on historical and new transactions.

First we will perform aggregation of card_id's to find different features for all historical and new transactions columns.

1. authorized_flag features will indicate total authorized transactions and percentage of authorized transactions done by each card_id.
2. category_1, category_2 and category_3 features will indicate total and percentage of transactions for that particular category value done by each card_id.
3. merchant_id, merchant_category_id, subsector_id, city_id and state_id features will indicate number of unique merchants, merchant categories, subsectors, cities and states, each card_id did transaction at.
4. month_lag features will indicate minimum and maximum transaction lag from reference date and and recency of transaction for each card_id.
5. purchase date features will indicate the oldest and the newest date of transactions done by each card_id.
6. year, month, dayofweek, date, hour features will indicate number of unique, mean, maximum and minimum of years, months, day of weeks, dates and hours, during which each card_id did transactions.
7. weekend feature will indicate total and percentage of transactions done by each card_id on weekend.
8. purchase_amount features will indicate total, average, maximum, minimum and variance of the amount spent by each card_id.
9. installments features will indicate number of unique, total, average, maximum and minimum number of installments for transactions done by each card_id.

In [36]:
```
groupby = ['card_id']
agg_dict = {
    'authorized_flag' : ['sum', 'mean'],
    'category_1_0'    : ['sum', 'mean'],
    'category_1_1'    : ['sum', 'mean'],
    'category_2_1'    : ['sum', 'mean'],
    'category_2_2'    : ['sum', 'mean'],
    'category_2_3'    : ['sum', 'mean'],
    'category_2_4'    : ['sum', 'mean'],
    'category_2_5'    : ['sum', 'mean'],
    'category_3_0'    : ['sum', 'mean'],
    'category_3_1'    : ['sum', 'mean'],
    'category_3_2'    : ['sum', 'mean'],

    'merchant_id'         : ['nunique'],
    'merchant_category_id': ['nunique'],
    'subsector_id'        : ['nunique'],
    'city_id'             : ['nunique'],
    'state_id'            : ['nunique'],

    'month_lag'    : ['min', 'max', 'mean'],
    'purchase_date': ['min', 'max'],
    'year'         : ['nunique', 'mean', 'min', 'max'],
    'month'        : ['nunique', 'mean', 'min', 'max'],
    'dayofweek'    : ['nunique', 'mean', 'min', 'max'],
    'date'         : ['nunique', 'mean', 'min', 'max'],
    'hour'         : ['nunique', 'mean', 'min', 'max'],
    'weekend'      : ['sum', 'mean'],

    'purchase_amount': ['sum', 'mean', 'max', 'min', 'std'],
    'installments'   : ['nunique', 'sum', 'mean', 'max', 'min']
    }
```

In [37]:
```
hist_transactions_features = pd.merge(hist_transactions_features,
                                      agg_featurization(historical_transactions, groupby, agg_dict, prefix = 'h
                                      on = 'card_id', how = 'left')
```

In [38]:
```
del agg_dict['authorized_flag']
```

In [39]:
```
new_transactions_features = pd.merge(new_transactions_features,
                                     agg_featurization(new_transactions, groupby, agg_dict, prefix = 'new'),
                                     on = 'card_id', how = 'left')
```

Engineering some additional features from generated aggregate features.

1. The difference in historical transaction count and historical authorized flag sum date will indicate the number of declined historical transactions for by cardholders.
2. The difference in maximum purchase date and minimum purchase date will indicate the duration in days when the transactions were done by each cardholders.
3. The ratio of total purchase amount and duration of transactions will indicate the purchase amount spent per day by each cardholders.
4. The difference in maximum purchase amount and minimum purchase amount will indicate the range of amount spent by each cardholders.
5. The ratio of transaction count and duration of transactions will indicate the transactions done per day by each cardholders.
6. The ratio of transaction count and unique number of merchant ids will indicate the transactions done per merchant by each cardholders.
7. The ratio of transaction count and unique number of city ids will indicate the transactions done per city by each cardholders.
8. The ratio of transaction count and unique number of state ids will indicate the transactions done per state by each cardholders.
9. The ratio of transaction count and unique merchant category ids will indicate the transactions done per merchant category by each cardholders.

In [40]:
```
hist_transactions_features['hist_denied_count'] = (hist_transactions_features['hist_transc_count'] -
                                                   hist_transactions_features['hist_authorized_flag_sum'])

hist_transactions_features['hist_transaction_days'] = (hist_transactions_features['hist_purchase_date_max'] -
                                                       hist_transactions_features['hist_purchase_date_min']).dt

hist_transactions_features['hist_purchase_amount_per_day'] = (hist_transactions_features['hist_purchase_amount_
                                                              (1 + hist_transactions_features['hist_transaction

hist_transactions_features['hist_purchase_amount_diff'] = (hist_transactions_features['hist_purchase_amount_max
                                                           hist_transactions_features['hist_purchase_amount_min

hist_transactions_features['hist_transactions_per_day'] = (hist_transactions_features['hist_transc_count'] /
                                                           (1 + hist_transactions_features['hist_transaction_da

hist_transactions_features['hist_transactions_per_merchant_id'] = (hist_transactions_features['hist_transc_coun
                                                                   (1 + hist_transactions_features['hist_merchant_

hist_transactions_features['hist_transactions_per_city_id'] = (hist_transactions_features['hist_transc_count']
                                                               (1 + hist_transactions_features['hist_city_id_nu

hist_transactions_features['hist_transactions_per_state_id'] = (hist_transactions_features['hist_transc_count']
                                                                (1 + hist_transactions_features['hist_state_id_

hist_transactions_features['hist_transactions_per_merchant_category_id'] = (hist_transactions_features['hist_tr
                                                          (1 + hist_transactions_features['hist_merchant_category_id_
```

In [41]:
```
hist_transactions_features = hist_transactions_features.drop(columns = ['hist_purchase_date_max', 'hist_purchas
```

In [42]:
```
hist_transactions_features = reduce_mem_usage(hist_transactions_features)
hist_transactions_features.head()
```

```
Mem. usage decreased to 56.18 Mb (69.3% reduction)
```

Out[42]:

	card_id	hist_transc_count	hist_authorized_flag_sum	hist_authorized_flag_mean	hist_category_1_0_sum	hist_category_1_0_mean	hist_
0	C_ID_0046db9f8a	150	142.0	0.946777	147.0	0.979980	
1	C_ID_011b0d9794	27	27.0	1.000000	0.0	0.000000	
2	C_ID_01904d743d	71	50.0	0.704102	71.0	1.000000	
3	C_ID_01b098ff01	182	128.0	0.703125	156.0	0.856934	
4	C_ID_0382b662f4	170	162.0	0.953125	163.0	0.958984	

5 rows × 73 columns

In [43]:
```
new_transactions_features['new_transaction_days'] = (new_transactions_features['new_purchase_date_max'] -
                                                     new_transactions_features['new_purchase_date_min']).dt.day

new_transactions_features['new_purchase_amount_per_day'] = (new_transactions_features['new_purchase_amount_sum'
                                                            (1 + new_transactions_features['new_transaction_day

new_transactions_features['new_purchase_amount_diff'] = (new_transactions_features['new_purchase_amount_max'] -
                                                         new_transactions_features['new_purchase_amount_min'])

new_transactions_features['new_transaction_per_day'] = (new_transactions_features['new_transc_count'] /
                                                        (1 + new_transactions_features['new_transaction_days'])

new_transactions_features['new_transactions_per_merchant_id'] = (new_transactions_features['new_transc_count']
                                                                 (1 + new_transactions_features['new_merchant_i

new_transactions_features['new_transactions_per_city_id'] = (new_transactions_features['new_transc_count'] /
                                                             (1 + new_transactions_features['new_city_id_nuniqu

new_transactions_features['new_transactions_per_state_id'] = (new_transactions_features['new_transc_count'] /
                                                              (1 + new_transactions_features['new_state_id_nuni

new_transactions_features['new_transactions_per_merchant_category_id'] = (new_transactions_features['new_transc
                                                        (1 + new_transactions_features['new_merchant_category_i
```

In [44]:
```
new_transactions_features = new_transactions_features.drop(columns = ['new_purchase_date_max', 'new_purchase_da
```

In [45]:
```
new_transactions_features = reduce_mem_usage(new_transactions_features)
new_transactions_features.head()
```

```
Mem. usage decreased to 45.40 Mb (67.1% reduction)
```

Out[45]:

	card_id	new_transc_count	new_category_1_0_sum	new_category_1_0_mean	new_category_1_1_sum	new_category_1_1_mean	new_ca
0	C_ID_0001238066	26	24	0.922852	2	0.076904	
1	C_ID_0001793786	31	31	1.000000	0	0.000000	
2	C_ID_0009150c11	9	9	1.000000	0	0.000000	
3	C_ID_003e56029c	12	12	1.000000	0	0.000000	
4	C_ID_006005dd0a	4	4	1.000000	0	0.000000	

5 rows × 70 columns

Perfoming aggregate featurization on historical and new transactions based on category 1, category 2 and category 3 values.

During EDA, we found that purchase amount had different distributions for different values of category 1, category 2 and category 3. We will perform aggregation of card id with different values of category 1, category 2 and category 3 columns and find features for purchase amount.

In [46]:
```
def category_aggregate_featurization(df, columns, groupby, agg_dict, prefix = ""):
    """This function performs aggregation on a dataframe based on groupby and each categorical columns
    and returns the aggregate features dataframe. It takes dataset as dataframe, groupby columns on
    which aggregate has to be performed as list, categorical columns which have to be aggregated with
    groupby columns as list and aggregate functions to be performed on columns as dictionary."""

    df_features = pd.DataFrame(df['card_id'].unique(), columns = ['card_id'])
    for col in columns:
        agg_df = agg_featurization(df[df[col] == 1], groupby, agg_dict, prefix = prefix + "_" + col)
        df_features = pd.merge(df_features, agg_df, on = 'card_id', how = 'left')
        del agg_df

    return df_features
```

In [47]:
```
agg_dict = {
    'purchase_amount': ['sum', 'mean', 'min', 'max', 'std']
           }
category_col = ['category_1_0', 'category_1_1', 'category_2_1', 'category_2_2', 'category_2_3', 'category_2_4',
                'category_2_5', 'category_3_0', 'category_3_1', 'category_3_2']
```

In [48]:
```
hist_category_features = category_aggregate_featurization(historical_transactions, category_col, groupby, agg_d
                                                          prefix = 'hist')
```

In [49]:
```
hist_category_features = reduce_mem_usage(hist_category_features)
hist_category_features.head()
```

```
Mem. usage decreased to 72.02 Mb (48.0% reduction)
```

Out[49]:

	card_id	hist_category_1_0_purchase_amount_sum	hist_category_1_0_purchase_amount_mean	hist_category_1_0_purchase_amount_min	l
0	C_ID_4e6213e9bc	5340.200195	13.350500	1.00000	
1	C_ID_5037ff576e	29740.710938	252.039917	20.28125	
2	C_ID_0e171c1b48	7345.680176	26.328604	1.00000	
3	C_ID_48fb13e70f	7949.169922	105.988937	6.00000	
4	C_ID_fc8e41b9cf	45930.800781	190.584229	10.00000	

5 rows × 51 columns

In [50]:
```
new_category_features = category_aggregate_featurization(new_transactions, category_col, groupby, agg_dict, pre
```

In [51]:
```
new_category_features = reduce_mem_usage(new_category_features)
new_category_features.head()
```

```
Mem. usage decreased to 51.76 Mb (58.3% reduction)
```

Out[51]:

	card_id	new_category_1_0_purchase_amount_sum	new_category_1_0_purchase_amount_mean	new_category_1_0_purchase_amount_min
0	C_ID_415bb3a509	374.338867	124.779617	118.172371
1	C_ID_ef55cf8d4b	10175.540039	565.307800	28.162193
2	C_ID_241a01e9d9	490.777863	245.388931	189.823074
3	C_ID_a97720321f	2959.939697	246.661636	19.063690
4	C_ID_fb0875cd28	405.751007	57.964428	22.963049

5 rows × 51 columns

Perfoming aggregate featurization on historical and new transactions based on month lag values.

We will perform aggregation of card id for each month lag column values to find features for purchase amount. These features will indicate month wise features of purchase amount.

In [52]:
```
def month_lag_aggregate_featurization(df, groupby, agg_dict, prefix = ""):
    """This function performs aggregation on a dataframe based on groupby and each value of month lag
    columns and returns the aggregate features dataframe. It takes dataset as dataframe, groupby columns on
    which aggregate has to be performed as list and aggregate functions to be performed on columns as
    dictionary."""

    df_features = pd.DataFrame(df['card_id'].unique(), columns = ['card_id'])
    for value in df['month_lag'].unique():
        agg_df = agg_featurization(df[df['month_lag'] == value], groupby, agg_dict,
                                   prefix = prefix + '_month_lag_' + str(value))
        df_features = pd.merge(df_features, agg_df, on = 'card_id', how = 'left')
        del agg_df

    return df_features
```

In [53]:
```
agg_dict = {
    'purchase_amount': ['sum', 'mean', 'min', 'max', 'std']
           }
```

In [54]:
```
hist_month_lag_features = month_lag_aggregate_featurization(historical_transactions, groupby, agg_dict, prefix
```

In [55]:
```
hist_month_lag_features = reduce_mem_usage(hist_month_lag_features)
hist_month_lag_features.head()
```

```
Mem. usage decreased to 95.61 Mb (49.2% reduction)
```

Out[55]:

	card_id	hist_month_lag_-8_purchase_amount_sum	hist_month_lag_-8_purchase_amount_mean	hist_month_lag_-8_purchase_amount_min
0	C_ID_4e6213e9bc	304.970032	9.837743	3.000020
1	C_ID_5037ff576e	351.809998	50.258572	28.399992
2	C_ID_0e171c1b48	683.950073	22.798336	1.789999
3	C_ID_48fb13e70f	4140.770020	147.884644	13.000019
4	C_ID_fc8e41b9cf	0.000000	NaN	NaN

5 rows × 71 columns

In [56]:
```
new_month_lag_features = month_lag_aggregate_featurization(new_transactions, groupby, agg_dict)
```

In [57]:
```
new_month_lag_features = reduce_mem_usage(new_month_lag_features)
new_month_lag_features.head()
```

```
Mem. usage decreased to 22.44 Mb (37.2% reduction)
```

Out[57]:

	card_id	_month_lag_1_purchase_amount_sum	_month_lag_1_purchase_amount_mean	_month_lag_1_purchase_amount_min	_month_lag_
0	C_ID_415bb3a509	294.077087	98.00	49.93750	
1	C_ID_ef55cf8d4b	13912.424805	927.50	28.15625	
2	C_ID_241a01e9d9	0.000000	NaN	NaN	
3	C_ID_a97720321f	2623.511475	262.25	19.06250	
4	C_ID_fb0875cd28	320.939484	80.25	49.93750	

Perfoming successive aggregate featurization on historical and new transactions with installments.

We will perform aggregation of card id installments columns to find features for authorized flag and purchase amount. These features will indicate installment wise features of authorized flag and purchase amount.

In [58]:
```
groupby1 = 'card_id'
groupby2 = 'installments'
columns = ['purchase_amount', 'authorized_flag']
agg_dict = {
    'authorized_flag': ['sum', 'mean'],
    'purchase_amount': ['sum', 'mean', 'min', 'max', 'std']
           }
```

In [59]:
```
hist_installments_features = successive_agg_featurization(historical_transactions, groupby1, groupby2, columns,
                                                          prefix = 'hist')
```

In [60]:
```
hist_installments_features = reduce_mem_usage(hist_installments_features)
hist_installments_features.head()
```

```
Mem. usage decreased to 19.24 Mb (34.1% reduction)
```

Out[60]:

	card_id	hist_installments_authorized_flag_sum	hist_installments_authorized_flag_mean	hist_installments_purchase_amount_sum	hist_ins
0	C_ID_0046db9f8a	8.390625	0.932129	6845.940918	
1	C_ID_011b0d9794	9.000000	1.000000	4191.762207	
2	C_ID_01904d743d	0.714355	0.357178	54.598984	
3	C_ID_01b098ff01	2.500000	0.416748	1525.291870	
4	C_ID_0382b662f4	1.969727	0.656738	238.732391	

In [61]:
```
new_installments_features = successive_agg_featurization(new_transactions, groupby1, groupby2, columns, agg_dic
                                                         prefix = 'new')
```

In [62]:
```
new_installments_features = reduce_mem_usage(new_installments_features)
new_installments_features.head()
```

```
Mem. usage decreased to 17.47 Mb (35.0% reduction)
```

Out[62]:

	card_id	new_installments_authorized_flag_sum	new_installments_authorized_flag_mean	new_installments_purchase_amount_sum	new_i
0	C_ID_0001238066	5.0	1.0	1074.932495	
1	C_ID_0001793786	1.0	1.0	492.143921	
2	C_ID_0009150c11	1.0	1.0	11.301080	
3	C_ID_003e56029c	1.0	1.0	62.565903	
4	C_ID_006005dd0a	1.0	1.0	127.067787	

RFM featurization on trasactions.

RFM is a market research tool for customer segmentation based on customer value to the firm. R stands for Recency, F for Frequency and M for Monetary value. Recency is the number of days since last purchase, Frequency is the total number of purchases and Monetary Value is the total money, the customer spent. An RFM analysis evaluates customers by scoring them in three categories: how recently they've made a purchase, how often they buy, and the size of their purchases.

Based on target values, we will find quantiles which will be used to calculate scores for each card id. The card ids will be scored based on which quantile their recency, frequency and monetary values fall into. Also, recency will be scored opposite of frequency and monetary value i.e., smaller the recency value higher the score whereas larger the frequency and monetary values higher the score.

The RFM score is the sum of the recency score, frequency score and monetary score while the RFM index is obtained by combining the recency score, frequency score and monetary score.

In [63]:
```
def RFM_Score(x, col, rfm_quantiles):
    """Function to calculate Recency, Frequency and Monetary value score based on quantiles.
    It takes respective value, column name and quantiles dataframe as input."""
    score_1 = 1
    score_2 = rfm_quantiles.shape[0]
    for i in range(rfm_quantiles.shape[0]):
        if x <= rfm_quantiles[col].values[i]:
            return score_2 if col is 'recency' else score_1
        score_1 += 1
        score_2 -= 1
```

In [64]:
```python
#https://www.kaggle.com/code/rajeshcv/customer-loyalty-based-on-rfm-analysis/notebook
def rfm_feature(df, quantiles):
    """This function performs the RFM featurization on dataset by generating the RFM score
    and RFM index. It takes dataset as dataframe, and quantile values for scoring as list and
    returns the RFM features as dataframe."""
    agg_dict = {
        'card_id'         : ['count'],
        'purchase_date'   : ['max'],
        'purchase_amount' : ['sum']
    }
    rfm_feature = agg_featurization(historical_transactions, groupby, agg_dict)
    rfm_feature['recency'] = (datetime.date(2018, 3, 1) - rfm_feature['purchase_date_max'].dt.date).dt.days
    rfm_feature.rename(columns = {'card_id_count' : 'frequency', 'purchase_amount_sum' : 'monetary_value'}, inp
    rfm_feature = rfm_feature.drop(columns = ['purchase_date_max'])

    rfm_quantiles = rfm_feature.quantile(q = quantiles)
    rfm_feature['R_score'] = rfm_feature['recency'].apply(RFM_Score, args = ('recency', rfm_quantiles))
    rfm_feature['F_score'] = rfm_feature['frequency'].apply(RFM_Score, args = ('frequency', rfm_quantiles))
    rfm_feature['M_score'] = rfm_feature['monetary_value'].apply(RFM_Score, args = ('monetary_value', rfm_quant
    rfm_feature['RFM_Score'] = rfm_feature['R_score'] + rfm_feature['F_score'] + rfm_feature['M_score']
    rfm_feature['RFM_index'] = rfm_feature['R_score'].map(str) + rfm_feature['F_score'].map(str) + rfm_feature[
    rfm_feature['RFM_index'] = rfm_feature['RFM_index'].astype(int)
    rfm_feature = rfm_feature.drop(columns = ['recency', 'frequency', 'monetary_value'])

    return rfm_feature
```

In [65]:
```python
quantiles = [0.012, 0.02, 0.05, 0.2, 0.5, 0.8, 0.96, 0.992, 1.0]
hist_rfm_feature = rfm_feature(historical_transactions, quantiles)
```

In [66]:
```python
hist_rfm_feature = reduce_mem_usage(hist_rfm_feature)
hist_rfm_feature.head()
```

```
Mem. usage decreased to 13.65 Mb (40.6% reduction)
```

Out[66]:

	card_id	R_score	F_score	M_score	RFM_Score	RFM_index
0	C_ID_0046db9f8a	9	7	7	23	977
1	C_ID_011b0d9794	4	5	6	15	456
2	C_ID_01904d743d	4	6	5	15	465
3	C_ID_01b098ff01	4	7	6	17	476
4	C_ID_0382b662f4	3	7	6	16	376

Merging all transactions features.

In [67]:
```python
all_transaction_features = pd.merge(hist_transactions_features, new_transactions_features, on = 'card_id', how 
all_transaction_features = pd.merge(all_transaction_features, hist_category_features, on = 'card_id', how = 'le
all_transaction_features = pd.merge(all_transaction_features, new_category_features, on = 'card_id', how = 'lef
all_transaction_features = pd.merge(all_transaction_features, hist_month_lag_features, on = 'card_id', how = 'l
all_transaction_features = pd.merge(all_transaction_features, new_month_lag_features, on = 'card_id', how = 'le
all_transaction_features = pd.merge(all_transaction_features, hist_installments_features, on = 'card_id', how =
all_transaction_features = pd.merge(all_transaction_features, new_installments_features, on = 'card_id', how =
all_transaction_features = pd.merge(all_transaction_features, hist_rfm_feature, on = 'card_id', how = 'left')
```

In [68]:
```python
all_transaction_features = reduce_mem_usage(all_transaction_features)
all_transaction_features.head()
```

```
Mem. usage decreased to 301.46 Mb (19.4% reduction)
```

Out[68]:

	card_id	hist_transc_count	hist_authorized_flag_sum	hist_authorized_flag_mean	hist_category_1_0_sum	hist_category_1_0_mean	hist_(
0	C_ID_0046db9f8a	150	142.0	0.946777	147.0	0.979980	
1	C_ID_011b0d9794	27	27.0	1.000000	0.0	0.000000	
2	C_ID_01904d743d	71	50.0	0.704102	71.0	1.000000	
3	C_ID_01b098ff01	182	128.0	0.703125	156.0	0.856934	
4	C_ID_0382b662f4	170	162.0	0.953125	163.0	0.958984	

5 rows × 341 columns

In [69]:
```python
all_transaction_features.to_csv('data/all_transaction_features.csv')
```

In [70]:
```python
del hist_transactions_features
del new_transactions_features
del hist_category_features
del new_category_features
del hist_month_lag_features
del new_month_lag_features
del hist_installments_features
del new_installments_features
del hist_rfm_feature
```

Merging transactions features to train and test set on card_id.

Finally we will merge all features engineered fom transactions to train and test set based on card ids and features having null values are imputed with 0.

In [71]:
```python
train = pd.merge(train, all_transaction_features, on = 'card_id', how = 'left')
test = pd.merge(test, all_transaction_features, on = 'card_id', how = 'left')
```

In [72]:
```python
train.fillna(value = 0, inplace = True)
train.head()
```

Out[72]:

	first_active_month	card_id	feature_1	feature_2	feature_3	target	elapsed_time	first_active_year	outlier	hist_transc_count	...
0	6	C_ID_92a2005557	5	2	1	-0.820312	245	2017	0	260	...
1	1	C_ID_3d0044924f	4	1	0	0.392822	396	2017	0	350	...
2	8	C_ID_d639edf6cd	2	2	0	0.687988	549	2016	0	43	...
3	9	C_ID_186d6a6901	4	3	0	0.142456	153	2017	0	77	...
4	11	C_ID_cdbd2c0db2	1	3	0	-0.159790	92	2017	0	133	...

5 rows × 349 columns

In [73]:
```python
test.fillna(value = 0, inplace = True)
test.head()
```

Out[73]:

	first_active_month	card_id	feature_1	feature_2	feature_3	elapsed_time	first_active_year	hist_transc_count	hist_authorized_flag_sur
0	4	C_ID_0ab67a22ab	3	3	1	306	2017	68	44
1	1	C_ID_130fd0cbdd	2	3	0	396	2017	78	77
2	8	C_ID_b709037bc5	5	1	1	184	2017	13	9
3	12	C_ID_d27d835a9f	2	1	0	62	2017	26	26
4	12	C_ID_2b5e3df5c2	5	1	1	793	2015	110	87

5 rows × 347 columns

In [74]:
```python
train.to_csv('data/featurized_train.csv', index = False)
test.to_csv('data/featurized_test.csv', index = False)
```

Summary

1. Null values in merchant id columns of historical and new trasactions were imputed with mode while null values in category 2 and category 3 columns were imputed by training classifiers.
2. The purchase amount in historical and new transactions was de-anonymized and the categorical columns were one hot encoded.
3. Elapsed time feature for each card id was engineered from first active month and first active month was segregated into year and month.
4. Historical and New transactions were aggregated on card id and diffrent features were engineered from transactions columns.
5. Historical and New transactions were aggregated on card id with different category 1, category 2 and category 3 values, and different features were engineered from purchase amount columns.
6. Historical and New transactions were aggregated on card id with different month lag values, and different features were engineered from purchase amount columns.
7. Historical and New transactions were aggregated successively aggregated on card id with installments columns, and different features were engineered from purchase amount and authorized flag columns.
8. RFM analysis was used to engineer RFM score and RFM index features for each card id.
9. All the features were merged with train and test set on card ids and the null values were filled with 0.