Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

This Case Study is about solving a business problem of Elo using the methods of machine learning which is described briefly along with the various steps of building a machine learning model.

Intro:

Elo, one of the largest payment brands in Brazil, has built partnerships with merchants in order to offer promotions or discounts to its cardholders. Which is a part of marketing campaign, in this campaign their main objective is to keep their customers happy, by this sort of customized and specific perks. So, In order to know whom to target, they want to assign a loyalty score to each of the customers which is the main objective of this case study also. So, in this case study we will be building one machine learning model to predict loyalty score for a customer given its card-id and purchase history.

Business problem:

By doing this Elo will make sure that only its loyal customers get the best experiences and its partner merchants get repeated business. This will also save Elo from doing unwanted campaigns.

The reason of focusing loyalty is that, it has been seen that loyal customers convert friends and family into customers just through word of mouth, and they contribute higher revenues for brands they're loyal to. This isn't merely speculation. Research by Bain and Company, cites that in almost no instance can a retailer break even on one-time shoppers due to the high cost of customer acquisition.

Evangelistic customers are much more effective in boosting the top line of a brand more than marketing any campaign can. This has led to the rise and popularity of loyalty programs.

ML formulation of business problem:

The ML problem that we want solve here is to predict the loyalty scores based on the available purchase data of cardholders and merchants. Loyalty score is a real number hence we are to use regression models here.

Performance metric:

Also, the metric provided is RMSE to evaluate this model.

RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

where $\hat{y}$ is the predicted loyalty score for each `card_id`, and y is the actual loyalty score assigned to a `card_id`.

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

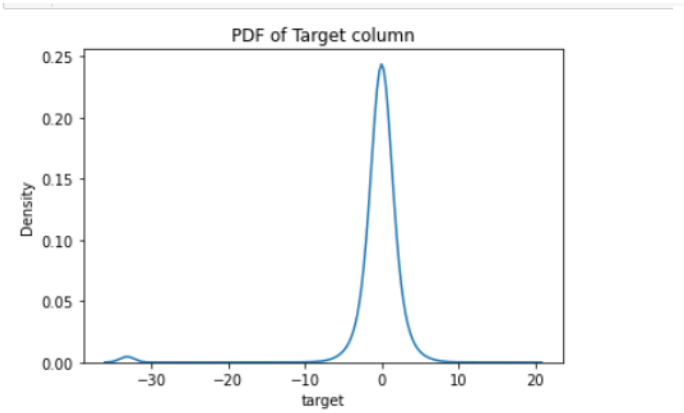
EDA ON THE DATA SETS –

a. **Train.csv and test.csv** - This file consists of card id , various categorical features whose meaning are not explicit and also the loyalty score which we have to predict.

Train dataset columns detail

Columns	Description
card_id	Unique card identifier
first_active_month	'YYYY-MM', month of first purchase
feature_1	Anonymized card categorical feature
feature_2	Anonymized card categorical feature
feature_3	Anonymized card categorical feature
target	Loyalty numerical score calculated 2 months after historical and evaluation period

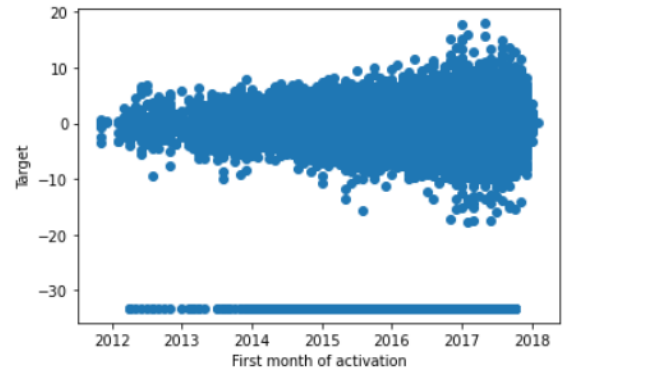
***Test data set has all the similar features except the target column which is to be predicted**



We can see most of the data is distributed between -10 to +10 though some distribution can be noticed below -30 as well. Apart from this we can see most of the values are tend have 0 loyalty scores.

This could mean data has been standardized at first and then this values below -30 has been added

TARGET VS FIRST MONTH OF ACTIVATION – SCATTER PLOT



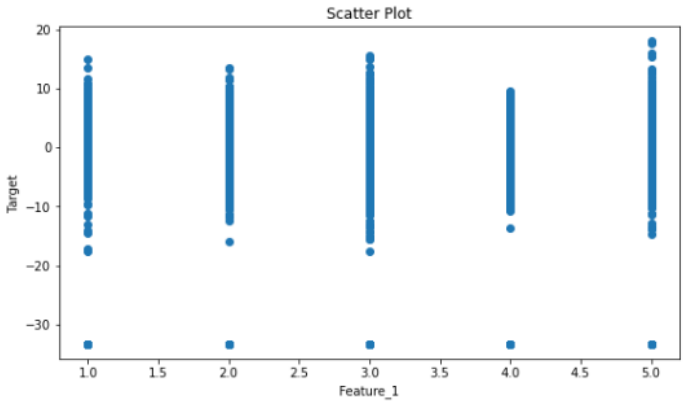
It's clear from above plot that with time number of users have increased. Even though with time the range of loyalty score have increased but the polarity is almost equal on both sides.

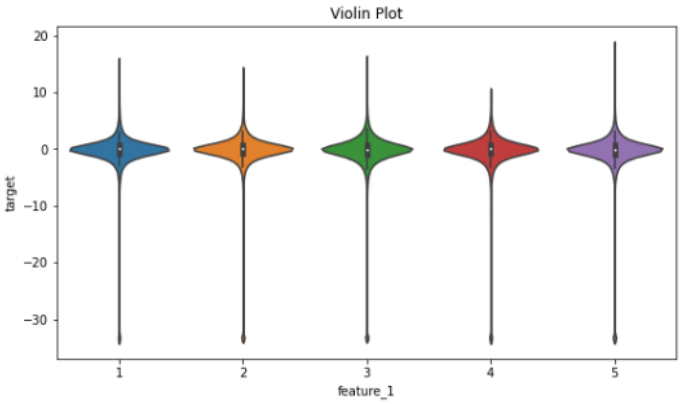
`

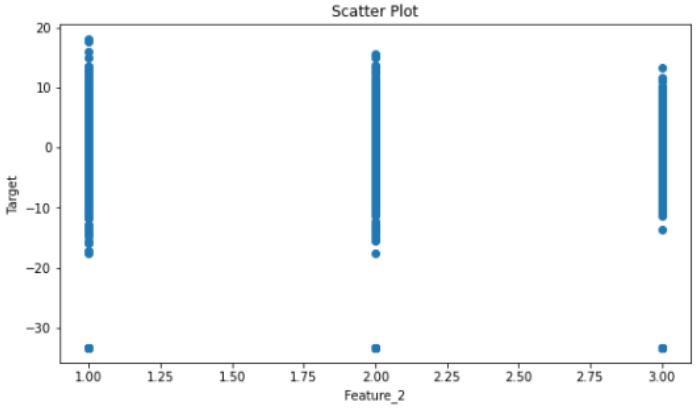
Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

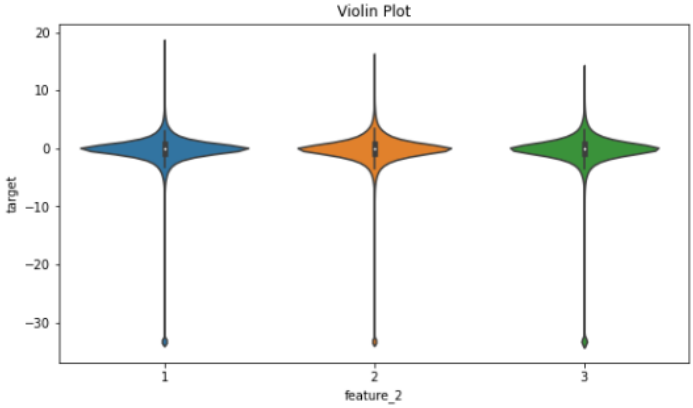
Though from this line plot it seems like after 2016 loyalty scores are getting more stable and tend to have positive values.

Distribution of Feature 1,2 & 3



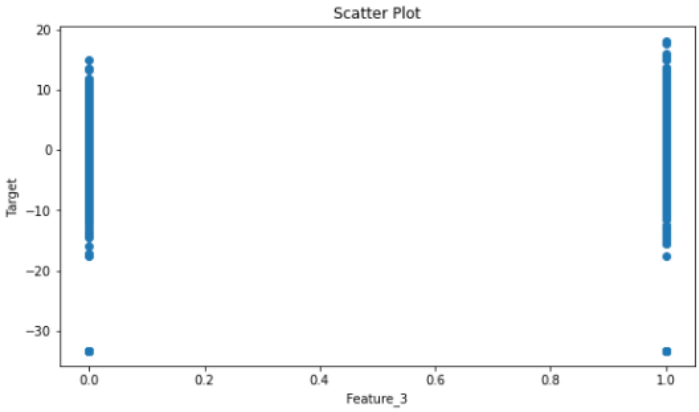


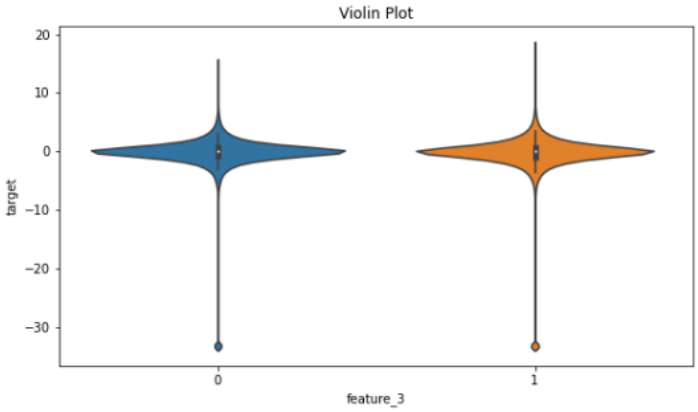




`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)





from above observation of feature 1,2 & 3 I can conclude that

#1. All these features are categorical

#2. There's no such distinguishable relationship that can be found between loyalty scores and these features

#3. Some data imbalanced can be noticed of these features

#5. most of the data of these features are contributing to the loyalty scores between the range of -10,10

#6. Data overlap is too much here

b. **Transactions Files-**

i. **historical_transactions.csv** - up to 3 months' worth of historical transactions for each card_id

ii. **new_merchant_transactions.csv** - two months' worth of data for each card_id containing ALL purchases that card_id made at merchant_ids that were not visited in the historical data

Columns Detail

`

Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)

Columns	Description
card_id	Card identifier
month_lag	month lag to reference date
purchase_date	Purchase date
authorized_flag	Y' if approved, 'N' if denied
category_3	anonymized category
installments	number of installments of purchase
category_1	anonymized category
merchant_category_id	Merchant category identifier (anonymized)
subsector_id	Merchant category group identifier (anonymized)
merchant_id	Merchant identifier (anonymized)
purchase_amount	Normalized purchase amount
city_id	City identifier (anonymized)
state_id	State identifier (anonymized)
category_2	anonymized category

Some basic stats for numerical features of Transaction tables

Historical Transaction

	installments	month_lag	purchase_amount
count	2.911236e+07	2.911236e+07	2.911236e+07
mean	6.484954e-01	-4.487294e+00	3.640090e-02
std	2.795577e+00	3.588800e+00	1.123522e+03
min	-1.000000e+00	-1.300000e+01	-7.469078e-01
25%	0.000000e+00	-7.000000e+00	-7.203559e-01
50%	0.000000e+00	-4.000000e+00	-6.883495e-01
75%	1.000000e+00	-2.000000e+00	-6.032543e-01
max	9.990000e+02	0.000000e+00	6.010604e+06

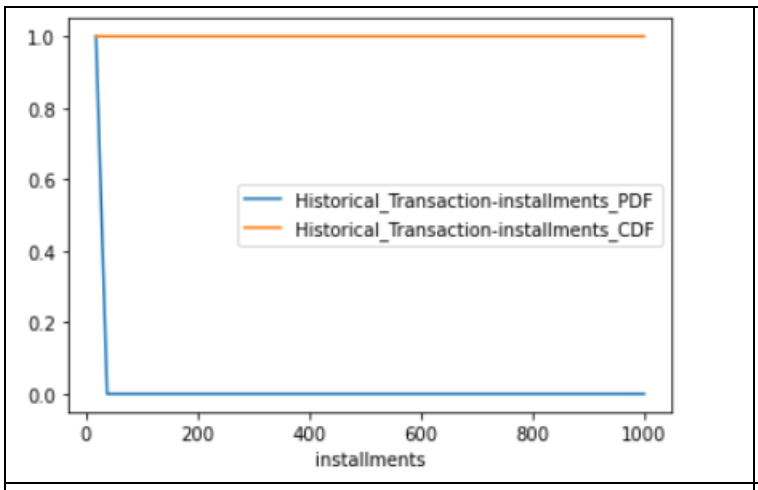
new_merchant_transactions

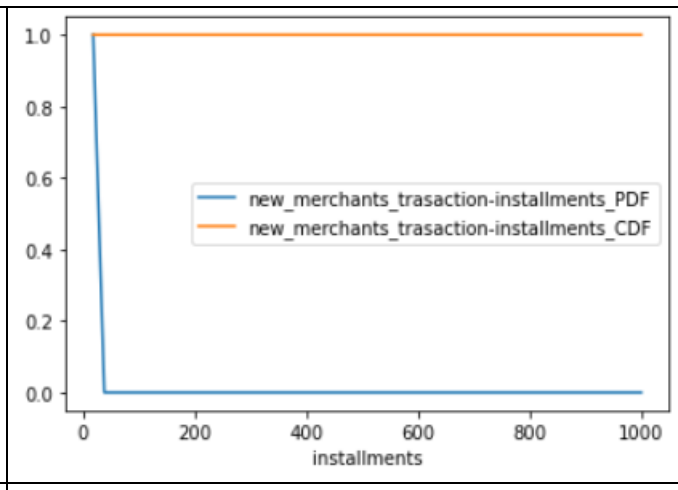
	installments	month_lag	purchase_amount
count	1.963031e+06	1.963031e+06	1.963031e+06
mean	6.829643e-01	1.476515e+00	-5.509690e-01
std	1.584069e+00	4.994483e-01	6.940043e-01
min	-1.000000e+00	1.000000e+00	-7.468928e-01
25%	0.000000e+00	1.000000e+00	-7.166294e-01
50%	1.000000e+00	1.000000e+00	-6.748406e-01
75%	1.000000e+00	2.000000e+00	-5.816162e-01
max	9.990000e+02	2.000000e+00	2.631575e+02

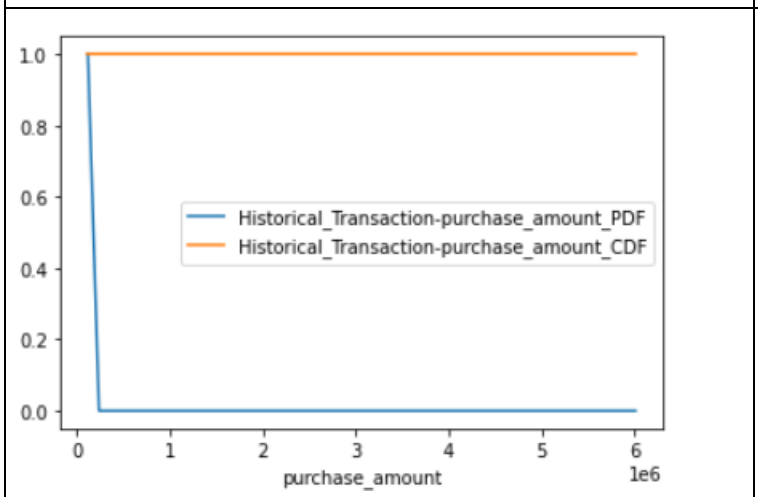
`

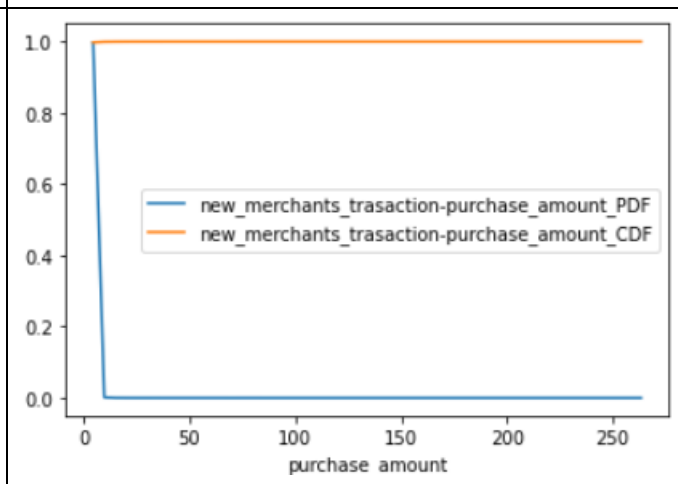
Elo Merchant Category Recommendation

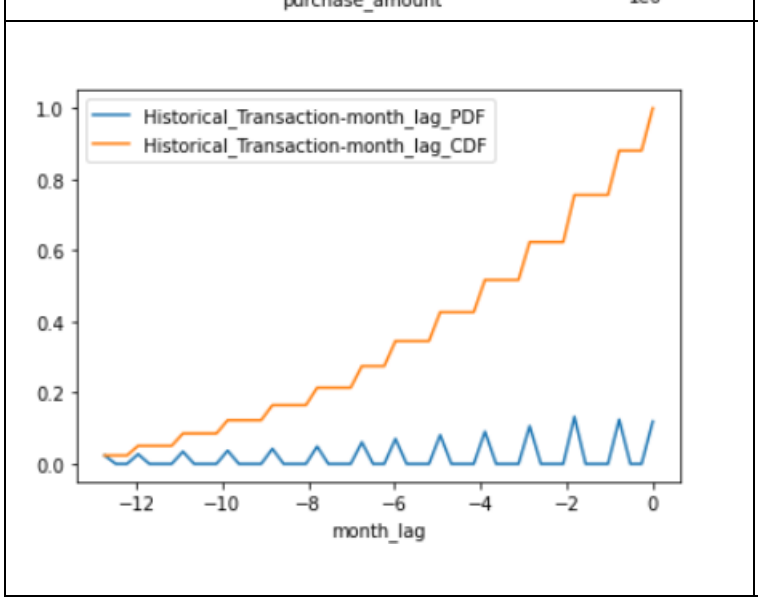
(MACHINE LEARNIG CASE STUDY)

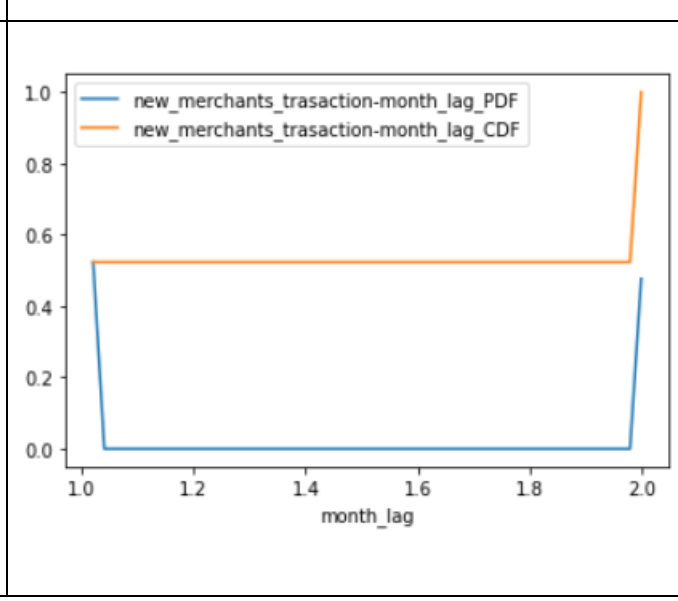












From the above I would like to say that only a few card holders have high installment and purchase amount its so rare that we can treat these high values as outliers.

`

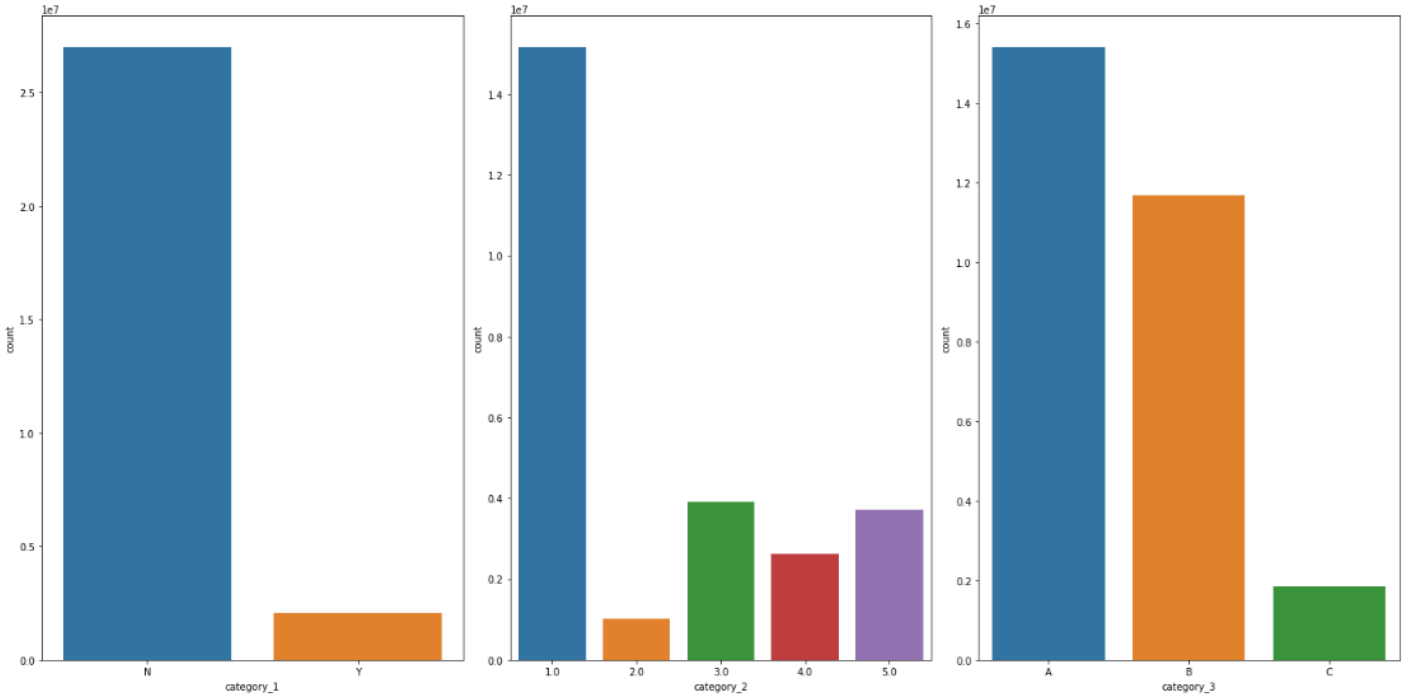
Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)

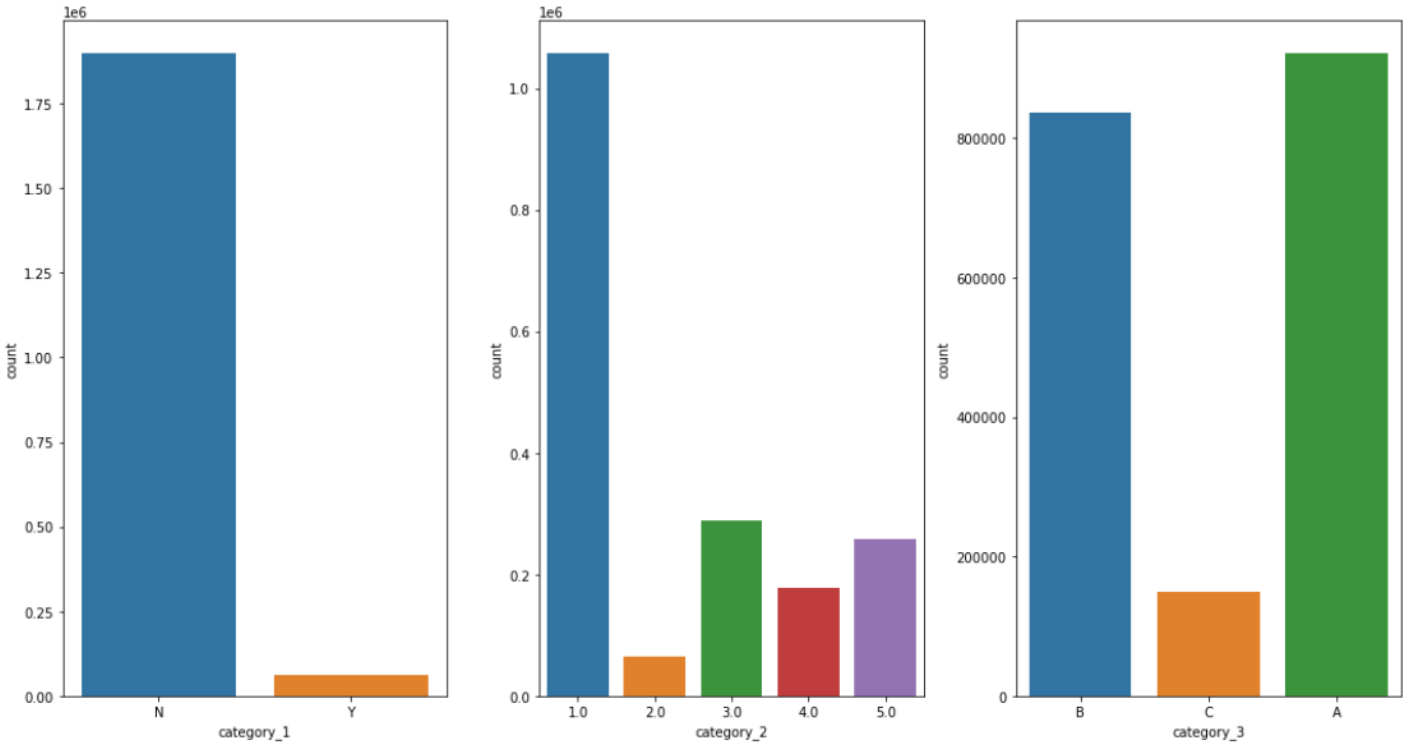
Also purchase_amount and installment doesn't seems to be proportional

Plots for categorical features

1.1 Bar PLots for category 1, 2, 3 of historical transaction table



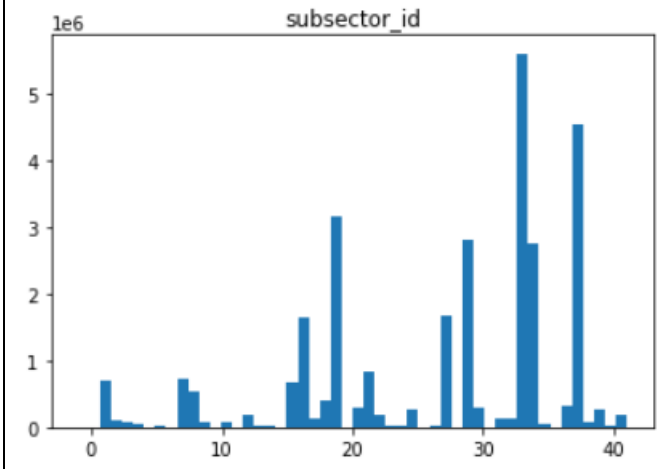
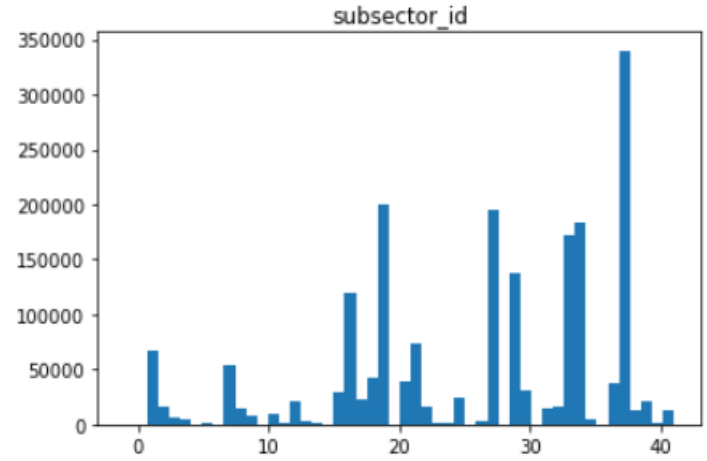
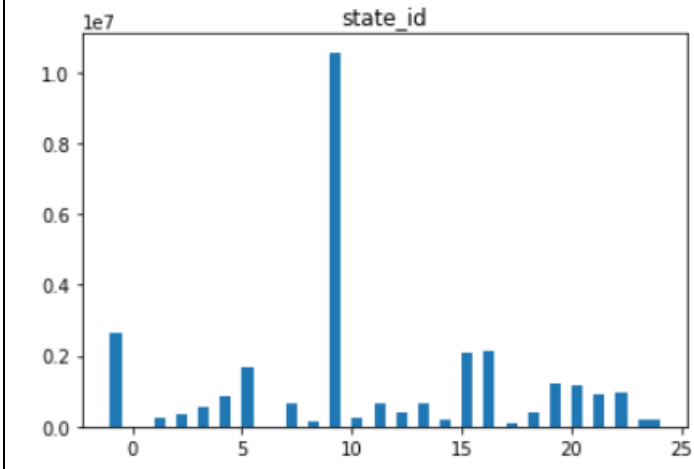
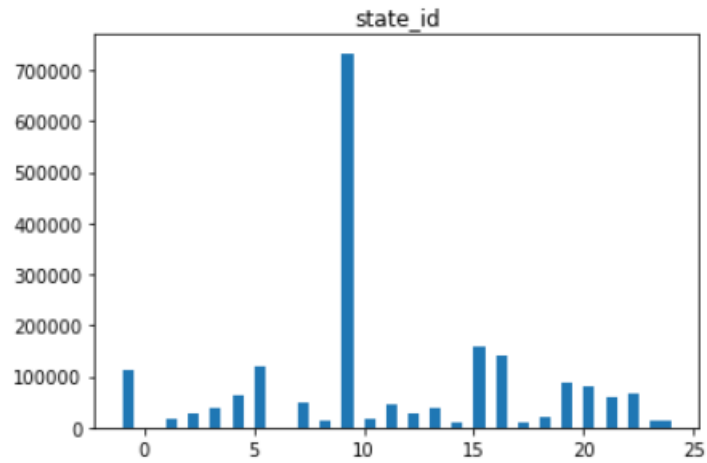
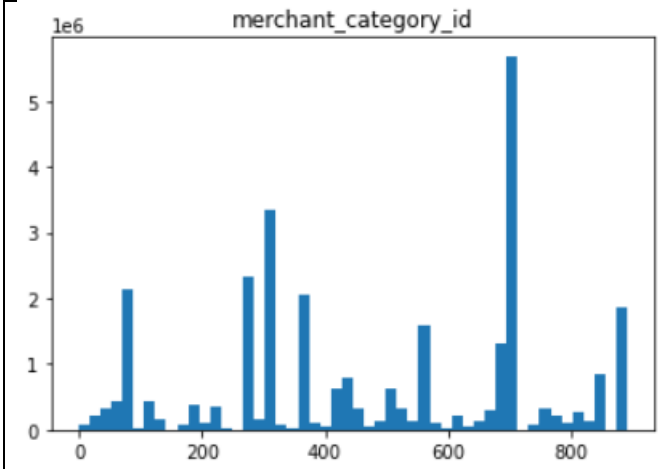
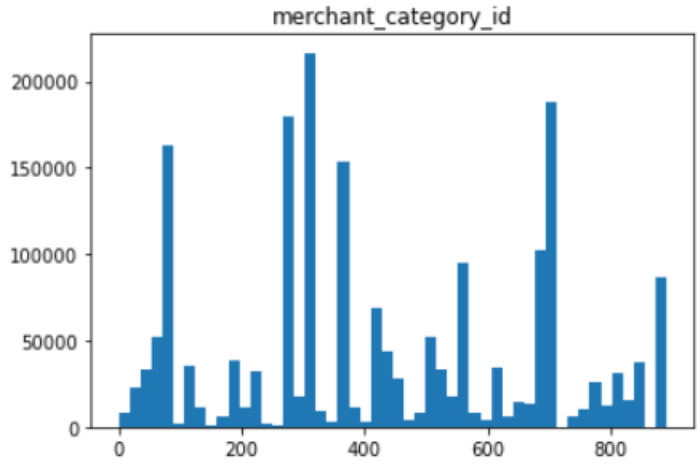
1.2 Bar PLots for category 1, 2, 3 of new_merchant_transactions



`

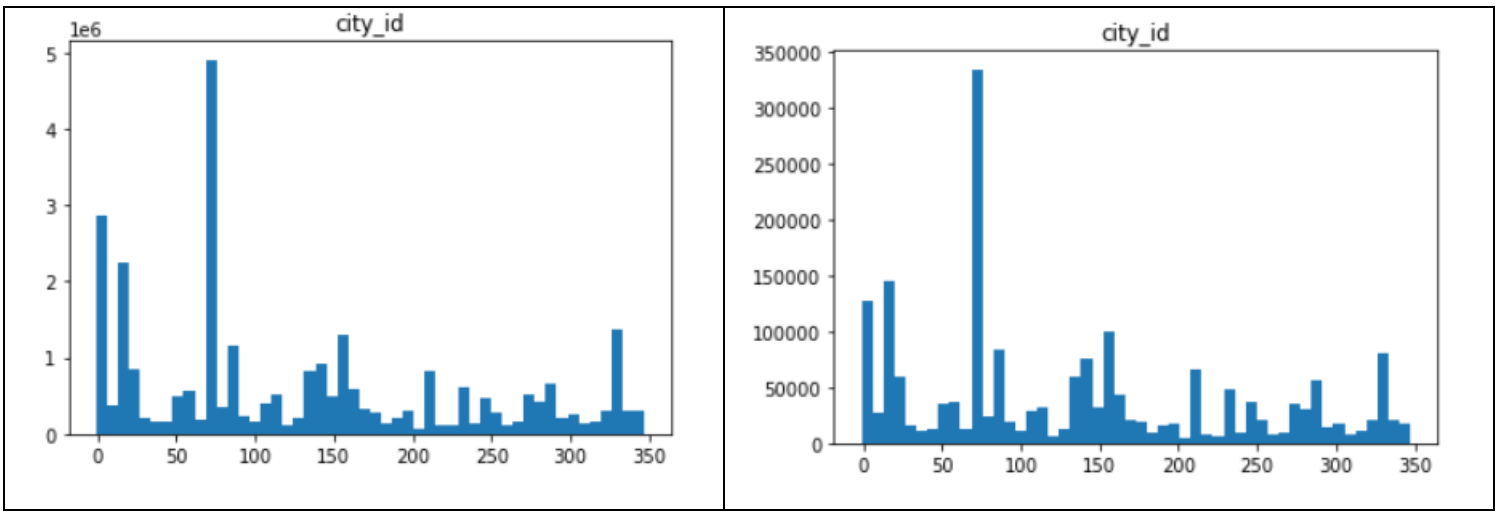
Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)

Historical Transaction	new_merchant_transactions
	
	
	

Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)



c. Merchants.csv - This file consists of merchants details.

Columns Detail

Columns	Description
merchant_id	Unique merchant identifier
merchant_group_id	Merchant group (anonymized)
merchant_category_id	Unique identifier for merchant category (anonymized)
subsector_id	Merchant category group (anonymized)
numerical_1	anonymized measure
numerical_2	anonymized measure
category_1	anonymized category
most_recent_sales_range	Range of revenue (monetary units) in last active month --> A > B > C > D > E
most_recent_purchases_range	Range of quantity of transactions in last active month --> A > B > C > D > E
avg_sales_lag3	Monthly average of revenue in last 3 months divided by revenue in last active month
avg_purchases_lag3	Monthly average of transactions in last 3 months divided by transactions in last active month
active_months_lag3	Quantity of active months within last 3 months
avg_sales_lag6	Monthly average of revenue in last 6 months divided by revenue in last active month
avg_purchases_lag6	Monthly average of transactions in last 6 months divided by transactions in last active month

`

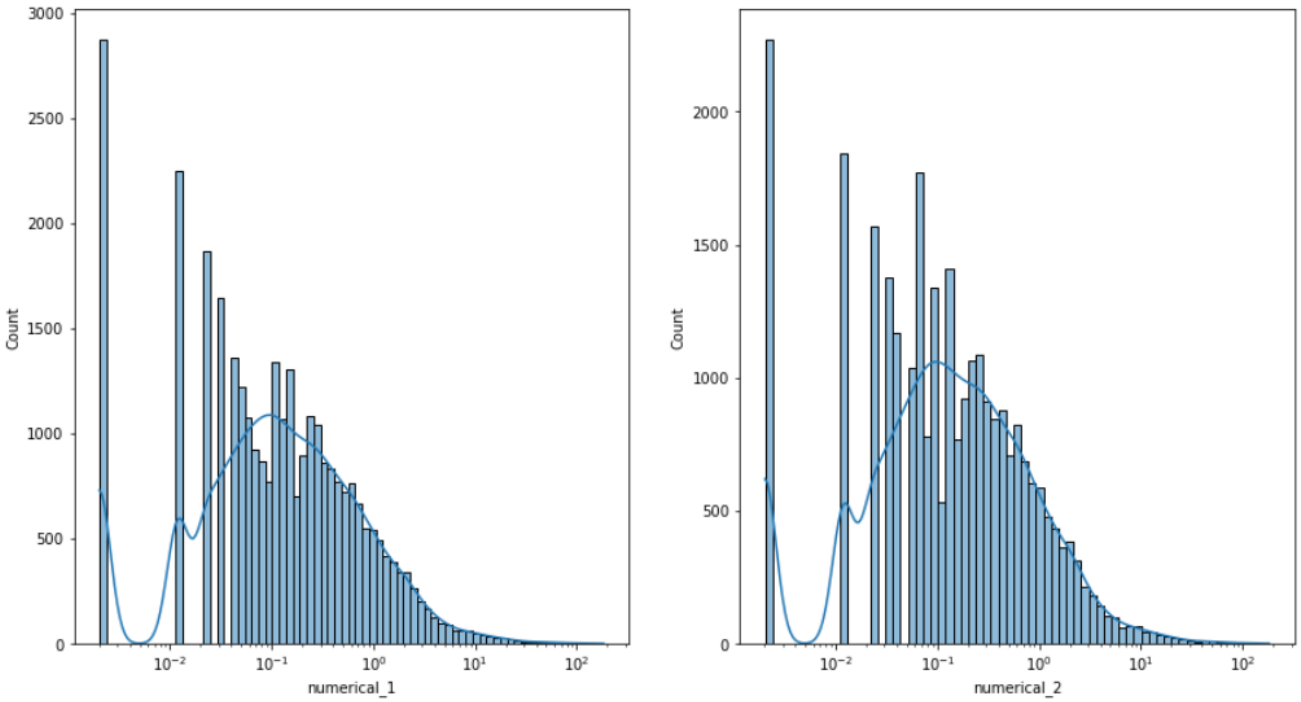
Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)

active_months_lag6	Quantity of active months within last 6 months
avg_sales_lag12	Monthly average of revenue in last 12 months divided by revenue in last active month
avg_purchases_lag12	Monthly average of transactions in last 12 months divided by transactions in last active month
active_months_lag12	Quantity of active months within last 12 months
category_4	anonymized category
city_id	City identifier (anonymized)
state_id	State identifier (anonymized)
category_2	anonymized category

Distribution plots of numerical feature columns

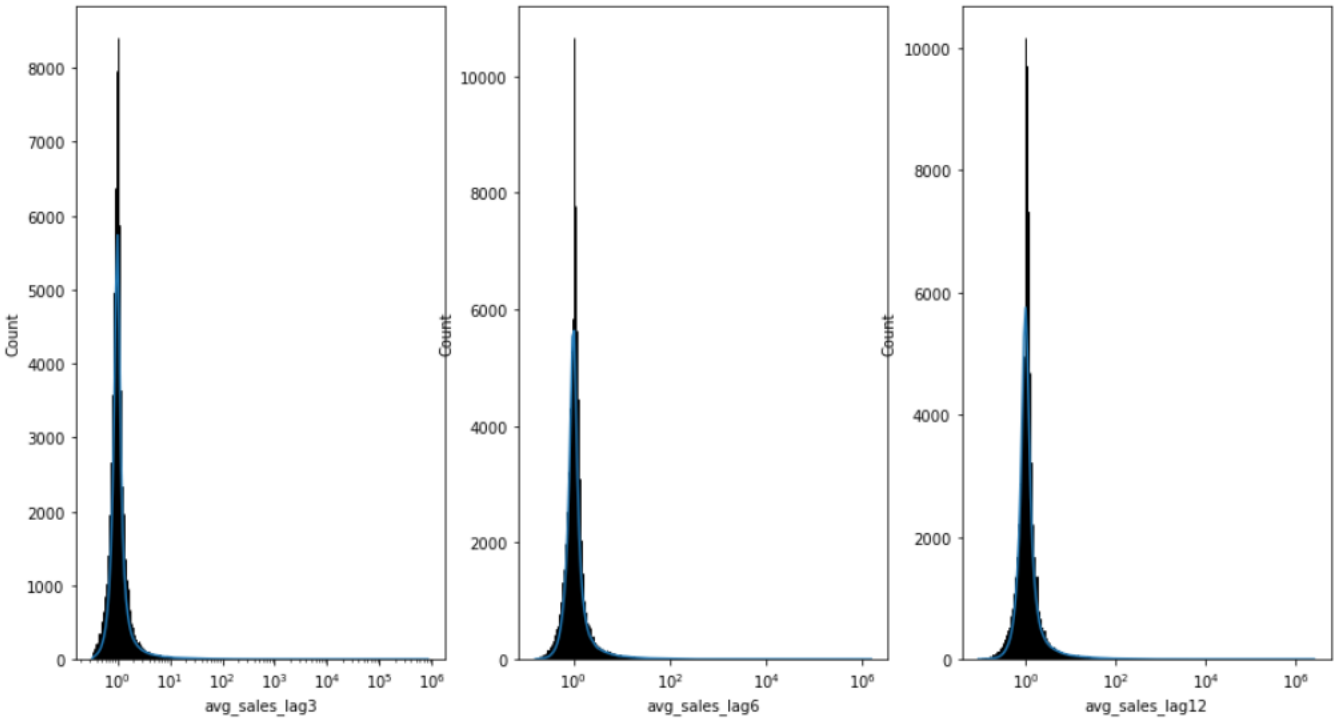
- For numerical 1& 2



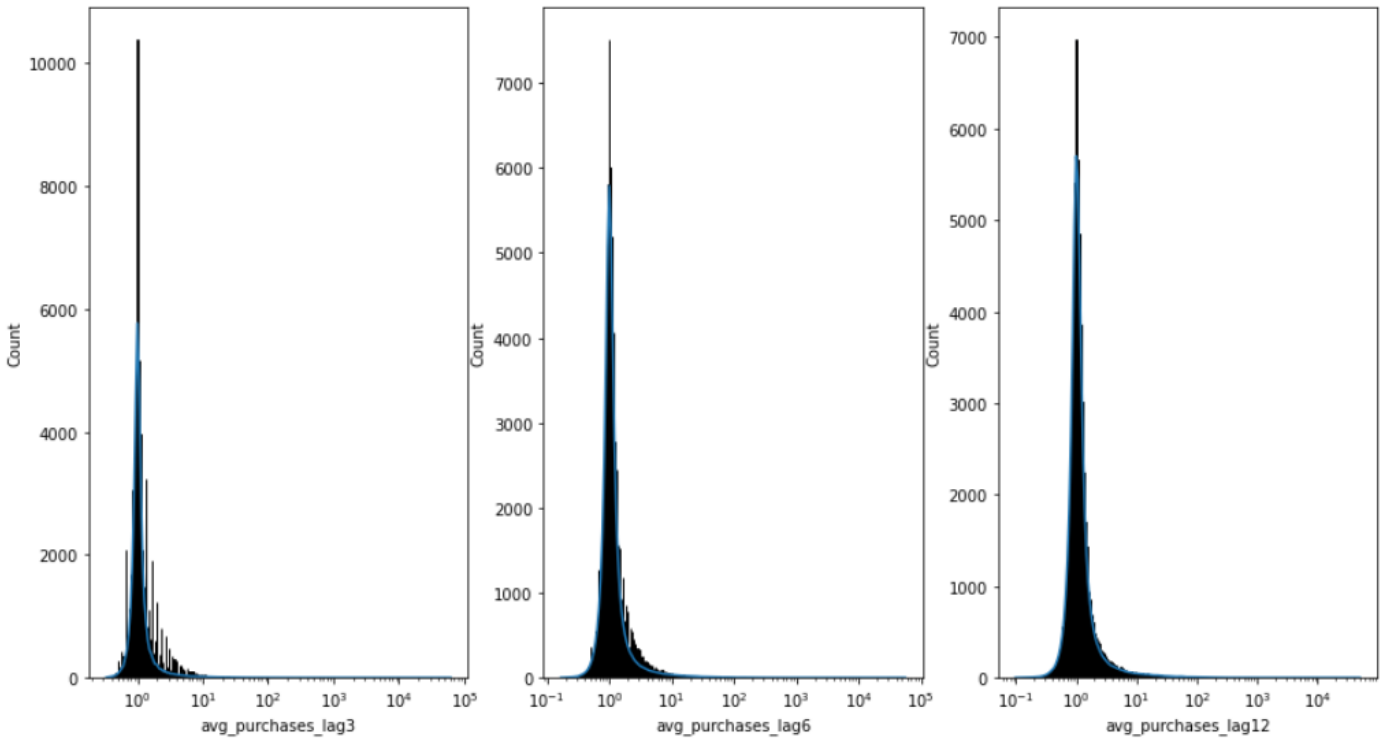
`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

- For average sales



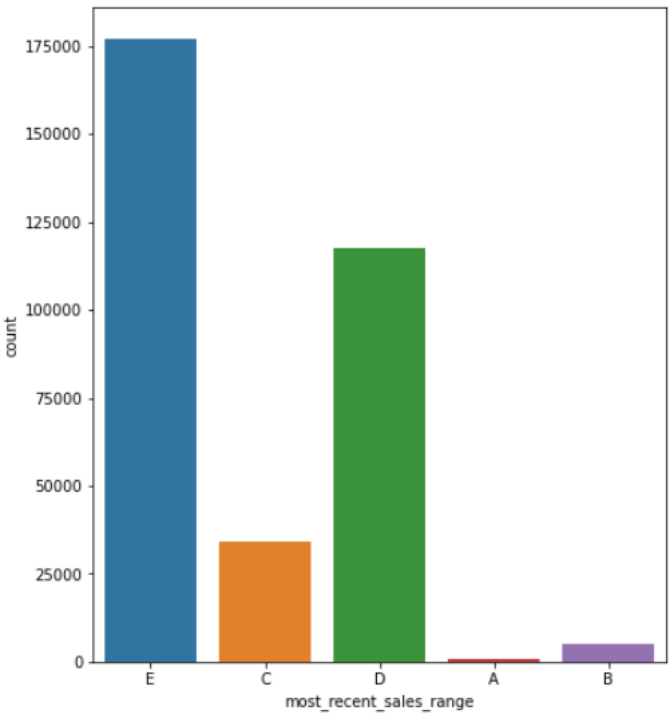
- For average purchase:

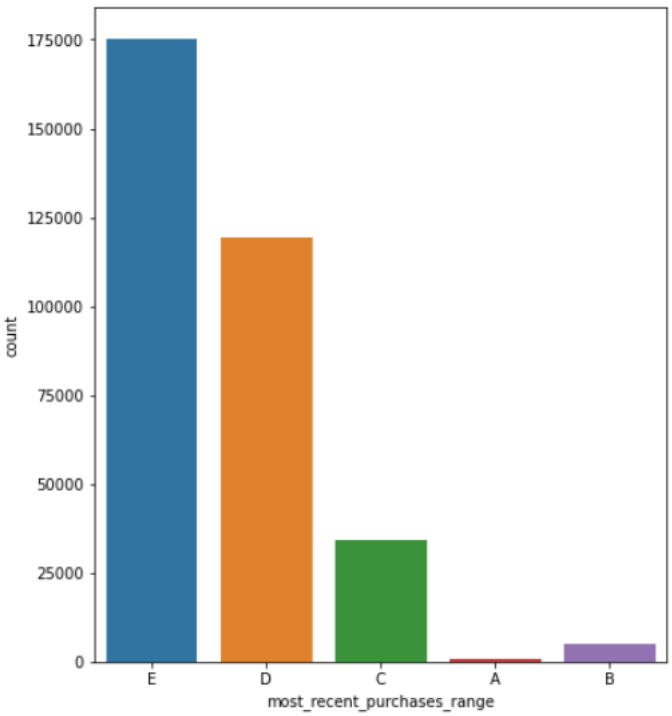


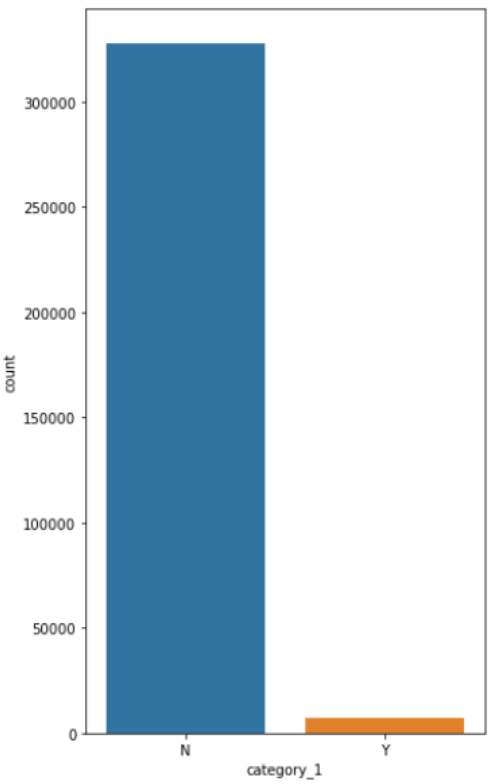
`

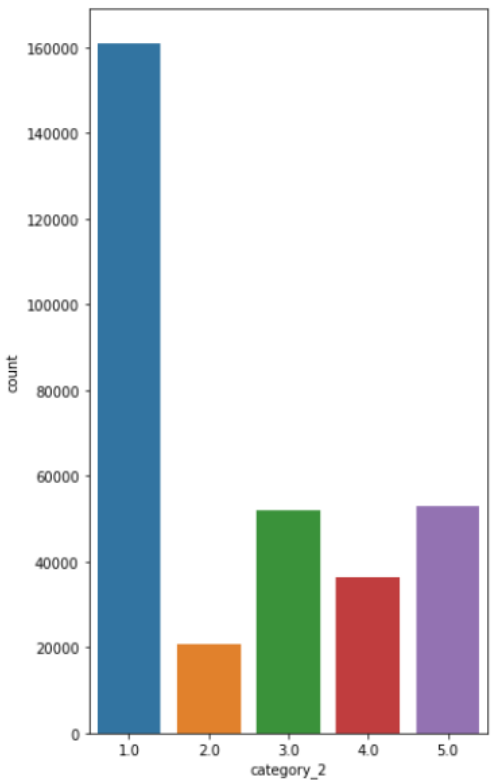
Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

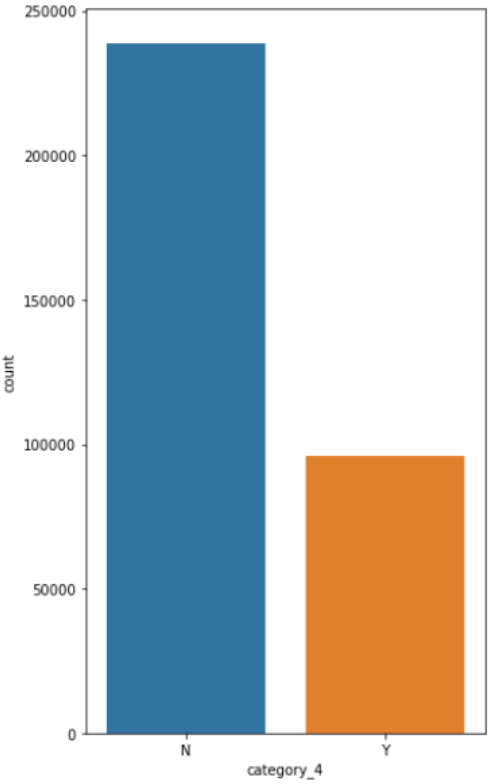
Bar Plots for categorical features:





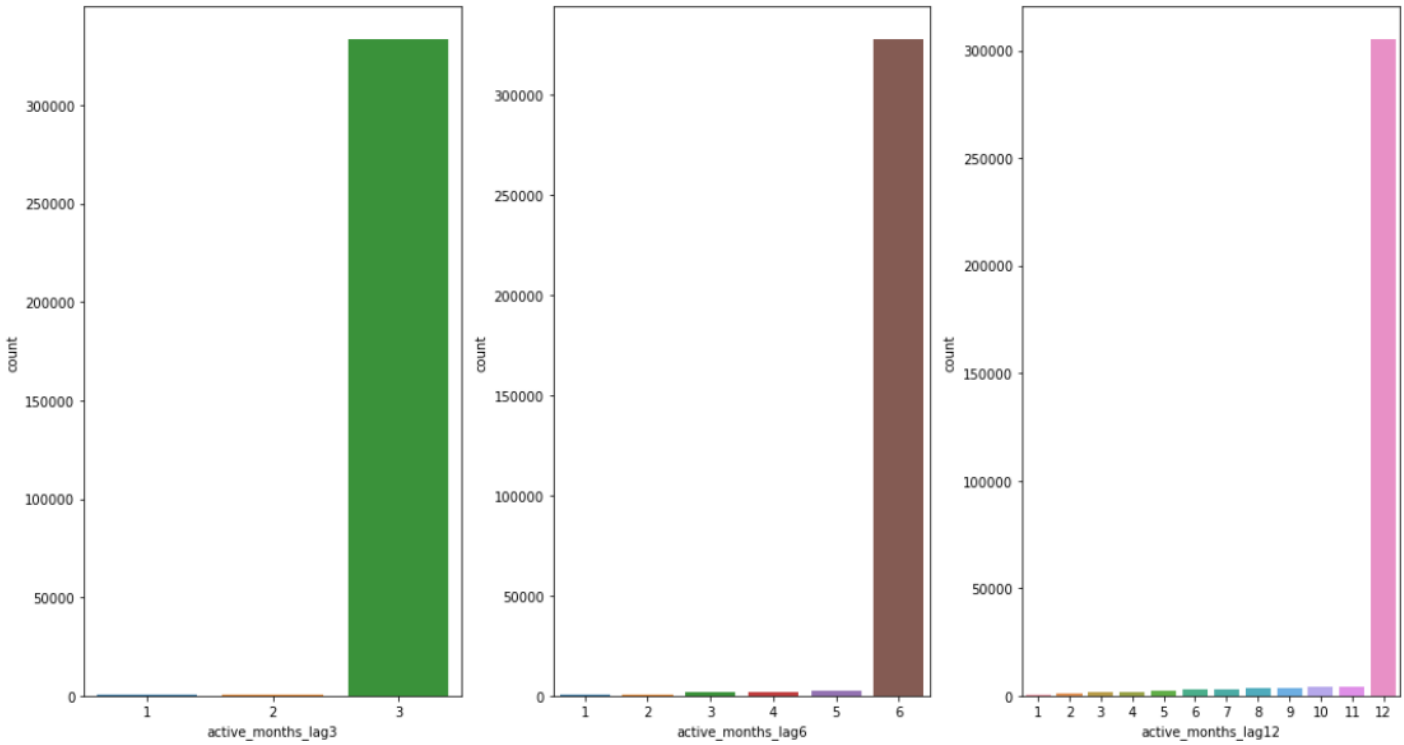






`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)



Techniques adopted for missing value imputation and data cleaning:

1. Train table didn't have any missing value so it left as it is.

2. Test.csv had one missing card id which has been imputed with most frequent card_id.

3. Now, transactions files had few categorical columns which had large number of missing values as shown below, that we simply can't impute with most frequent values else it'd create a biased dataset,

```
#Missing Values
new_merchants_trasaction.isna().sum()
```

```
authorized_flag              0
card_id                      0
city_id                      0
category_1                   0
installments                 0
category_3                55922
merchant_category_id         0
merchant_id               26216
month_lag                    0
purchase_amount              0
purchase_date                0
category_2               111745
state_id                     0
subsector_id                 0
dtype: int64
```

```
Historical_Transaction.isna().sum()
```

```
authorized_flag              0
card_id                      0
city_id                      0
category_1                   0
installments                 0
category_3              178159
merchant_category_id         0
merchant_id             138481
month_lag                    0
purchase_amount              0
purchase_date                0
category_2              2652864
state_id                     0
subsector_id                 0
dtype: int64
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

So, at first to impute merchant ids, rather taking most frequent merchant id in the entire dataset I took most frequent merchant ids belonging to each of the merchant categorie ids.

```
missing_id = historical_transaction.merchant_category_id.loc[historical_transaction.merchant_id.isnull()].unique()
```

```
historical_transaction_imp = historical_transaction.copy()


for i in tqdm(missing_id):

    mask = historical_transaction_imp.merchant_category_id == i

    value = historical_transaction_imp.loc[mask,"merchant_id"].value_counts().index[0]

    historical_transaction_imp.loc[mask,"merchant_id"] = historical_transaction_imp.loc[mask,"merchant_id"].fillna(value)
```

For other 2 columns I used iterative imputation technique

```
clean_nm = new_merchants_trasaction.drop(columns = ['card_id','purchase_date','merchant_id'])
clm = list(new_merchants_trasaction.drop(columns = ['card_id','purchase_date','merchant_id']).columns)
print(" Index numbers for category2 , 3 in new merchant",clm.index('category_2'),clm.index('category_3'))

clean_ht = historical_transaction.drop(columns = ['card_id','purchase_date','merchant_id'])
clm = list(historical_transaction.drop(columns = ['card_id','purchase_date','merchant_id']).columns)
print(" Index numbers for category2 , 3 in historical data",clm.index('category_2'),clm.index('category_3'))
```

```
 Index numbers for category2 , 3 in new merchant 8 4
 Index numbers for category2 , 3 in historical data 8 4
```

```
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
imp = IterativeImputer(max_iter= 15, random_state=0)

clean_nm = np.round(imp.fit_transform(clean_nm))

clean_ht = np.round(imp.fit_transform(clean_ht))
```

```
new_merchants_trasaction['category_3'] = clean_nm[:,4]
new_merchants_trasaction['category_2'] = clean_nm[:,8]

historical_transaction_imp['category_3'] = clean_ht[:,4]
historical_transaction_imp['category_2'] = clean_ht[:,8]
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

4. Merchant table has few missing values and inf values that has been taken care as followed

```
#Loading the merchants data set
merchants = reduce_mem_usage(pd.read_csv("/content/merchants.csv"))
merchants.isnull().sum()
```

```
Mem. usage decreased to 30.32 Mb (46.0% reduction)
merchant_id                      0
merchant_group_id                0
merchant_category_id             0
subsector_id                     0
numerical_1                      0
numerical_2                      0
category_1                       0
most_recent_sales_range          0
most_recent_purchases_range      0
avg_sales_lag3                  13
avg_purchases_lag3               0
active_months_lag3               0
avg_sales_lag6                  13
avg_purchases_lag6               0
active_months_lag6               0
avg_sales_lag12                 13
avg_purchases_lag12              0
active_months_lag12              0
category_4                       0
city_id                          0
state_id                         0
category_2                   11887
dtype: int64
```

```
na_imputer = lambda x: x.fillna(x.mean())
inf_imputer = lambda x: x.replace([np.inf, -np.inf], x.value_counts().index[0])

merchants_clean = merchants.copy()

merchants_clean[['avg_purchases_lag3']] = merchants_clean[['avg_purchases_lag3']].apply(inf_imputer)
merchants_clean[['avg_purchases_lag6']] = merchants_clean[['avg_purchases_lag6']].apply(inf_imputer)
merchants_clean[['avg_purchases_lag12']] = merchants_clean[['avg_purchases_lag12']].apply(inf_imputer)

merchants_clean[['avg_sales_lag3']] = merchants_clean[['avg_sales_lag3']].apply(na_imputer)
merchants_clean[['avg_sales_lag6']] = merchants_clean[['avg_sales_lag6']].apply(na_imputer)
merchants_clean[['avg_sales_lag12']] = merchants_clean[['avg_sales_lag12']].apply(na_imputer)
```

Feature engineering

In order to extract features from all the data sets techniques like **one hot encoding, mean hot encoding, cross feature extraction and aggregation methods** have been used along with time series data.

I am discussing in brief each of the kernels used for each of the tables.

1. For train and test table at the below kernel is used to get date time features, then feature 1, 2 & 3 has been mean hot encoded with the target values.

`

Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)

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

    min_dte = data['first_active_month'].dt.date.min()

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

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

    return data
```

```
# Mean hot encoding of categorical values(f1, f2, f3)
train_table['rare_value'] = train_table['target'].apply(lambda x: 1 if x<= -30 else 0)

inf_imputer = lambda x: x.replace([np.inf, -np.inf], x.value_counts().index[0])


for i in ['feature_1', 'feature_2', 'feature_3']:

  #train_table['days_' + i] = train_table['first_active_month'] * train_table[i]

  #train_table['days_' + i + '_ratio'] = np.log(train_table[i] / train_table['first_active_month'])
  #train_table['days_' + i + '_ratio'] = train_table[['days_' + i + '_ratio']].apply(inf_imputer)

  #test_table['days_' + i] = test_table['first_active_month'] * test_table[i]

  #test_table['days_' + i + '_ratio'] = np.log(test_table[i] / test_table['first_active_month'])
  #test_table['days_' + i + '_ratio'] = test_table[['days_' + i + '_ratio']].apply(inf_imputer)

  rare_data_mean = train_table.groupby([i])['rare_value'].mean()

  train_table[i] = train_table[i].map(rare_data_mean)

  test_table[i] = test_table[i].map(rare_data_mean)
```

2. To get features from merchant data set below kernel is used which mainly gets aggregates over merchant ids to get various feature's mean value and unique value counts

```
] def agg_merchant(df):


    dic_agg = {i:['mean'] for i in df.columns if (i not in ['merchant_id','category_4'])}
    dic_agg['most_recent_sales_range'] = ['nunique']
    dic_agg['most_recent_purchases_range'] = ['nunique']
    dic_agg['merchant_group_id'] = ['nunique']
    dic_agg['category_4'] = ['nunique','mean']


    agg = df.groupby(['merchant_id']).agg(dic_agg)

    agg.columns = ['_'.join(col).strip() for col in agg.columns.values]
    agg.reset_index(inplace=True)

    return agg
```

`

Elo Merchant Category Recommendation

(MACHINE LEARNIG CASE STUDY)

3. For transaction files at first I have merged two transaction files then at first I used two kernels, as mentioned below.

   One is get_date_feature kernel which is used to get the detail about how frequently one user is purchasing each year, each month, weekends and week days.

```
def get_date_feature(df_m):

  df = df_m.copy()

  df['purchase_date'] = pd.to_datetime(df['purchase_date'])

  df['m_op'] = df['purchase_date'].dt.month
  df['y_op'] = df['purchase_date'].dt.year

  df['week_day'] = df['purchase_date'].dt.day.apply(lambda x : 0 if x > 4 else 1)
  df['weekend'] = df['purchase_date'].dt.day.apply(lambda x : 0 if x < 4 else 1)

  df['days'] = ((df['purchase_date'] - datetime.datetime(2011,11,1))).dt.days

  df['month_diff'] = ((df['purchase_date'] - datetime.datetime(2011,11,1))).dt.days//30 #Month diff from the date of first card activation

  df['month_diff'] += df['month_lag']

  df = pd.get_dummies(df, columns =['m_op','y_op'])

  agg_func_1 = {
      'week_day' : ['sum','mean'],
      'weekend' : ['sum','mean'],
      'month_diff' : ['mean','min','max','std'],
      'month_lag' : ['mean','min','max','std']
  }

  agg_func_2 = { i : ['sum', 'mean'] for i in df.columns if ('m_op' in i) or ('y_op' in i)}

  df_1 = df.groupby('card_id').agg(agg_func_1)
  df_1.columns = [ "_".join(col).strip() for col in df_1.columns]
  df_1 = df_1.reset_index()

  df_2 = df.groupby('card_id').agg(agg_func_2)
  df_2.columns = ["_".join(col).strip() for col in  df_2.columns]
  df_2 = df_2.reset_index()

  df = df_1.merge(df_2, how = 'left', on = 'card_id')
  return df
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

And another kernel is get_purchase_feature, which I used to get an idea about how much each card user have spent on each month, each year etc.

```
def get_purchase_features(df_main):
  df = df_main.copy()
  agg_d = {'purchase_amount' : ['sum']}

  df['purchase_date'] = pd.to_datetime(df['purchase_date'])

  df['m_op'] = df['purchase_date'].dt.month
  df['y_op'] = df['purchase_date'].dt.year

  df_m = df.groupby(['card_id', 'm_op']).agg(agg_d)

  #fm = pd.get_dummies(nm, columns = ['m_op'])
  df_m.columns = ["_".join(col).strip() for col in df_m.columns]
  df_m = df_m.reset_index()
  #print('Pass')
  df_m = pd.get_dummies(df_m, columns = ['m_op'])

  ar = np.array(df_m.drop(columns = ['card_id','purchase_amount_sum']))

  ar1 = np.array(df_m['purchase_amount_sum'])
  ar1 = ar1.reshape(-1,1)

  l1 = ['m_op_'+ str(i) for i in range(1,13)]

  df_m[l1] = ar1*ar
  df_m = df_m.drop(columns = ['purchase_amount_sum'])
  df_m = df_m.groupby('card_id').agg(['sum','mean'])

  #print('Pass')
  ##########################################################

  df_y = df.groupby(['card_id', 'y_op']).agg(agg_d)

  #fm = pd.get_dummies(nm, columns = ['m_op'])
  df_y.columns = ["_".join(col).strip() for col in df_y.columns]
  df_y = df_y.reset_index()

  df_y = pd.get_dummies(df_y, columns = ['y_op'])

  ar = np.array(df_y.drop(columns = ['card_id','purchase_amount_sum']))

  ar1 = np.array(df_y['purchase_amount_sum'])
  ar1 = ar1.reshape(-1,1)

  l1 = ['y_op_2017','y_op_2018']

  df_y[l1] = ar1*ar
  df_y = df_y.drop(columns = ['purchase_amount_sum'])
  df_y = df_y.groupby('card_id').agg(['sum','mean'])

  df = df_y.merge(df_m, on = 'card_id', how = 'left')

  df.columns = ["_".join(col).strip() for col in df.columns]
  df.reset_index(inplace = True)
  return df
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

I also used two more kernals that is used after merging transaction dataframe with featurized merchant dateframe and these kernals are mention bleow:
aggregate_transactions in this kernel I'm aggregating all the columns after grouping them card ids

```
def aggregate_transactions(df):

    agg_func = {
    'category_1': ['sum', 'mean',mode],

    'category_3_0.0': ['mean','sum'],
    'category_3_1.0': ['mean','sum'],
    'category_3_2.0': ['mean','sum'],

    'category_2_1.0': ['mean','sum'],
    'category_2_2.0': ['mean','sum'],
    'category_2_3.0': ['mean','sum'],
    'category_2_4.0': ['mean','sum'],
    'category_2_5.0': ['mean','sum'],

    'authorized_flag_0' :['mean','sum'],
    'authorized_flag_1' :['mean','sum'],

    'merchant_id': ['nunique'],
    'merchant_category_id': ['nunique',mode],

    'state_id': ['nunique',mode],
    'city_id': ['nunique',mode],
    'subsector_id': ['nunique',mode],

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

    'numerical_1_mean' : ['sum', 'mean', 'max', 'min',std],
    'numerical_2_mean' : ['sum', 'mean', 'max', 'min',std],

    'most_recent_sales_range_mode_c_A' : ['mean','sum'],
    'most_recent_sales_range_mode_c_B' : ['mean','sum'],
    'most_recent_sales_range_mode_c_C' : ['mean','sum'],
    'most_recent_sales_range_mode_c_D' : ['mean','sum'],
    'most_recent_sales_range_mode_c_E' : ['mean','sum'],

    'most_recent_purchases_range_mode_c_A' : ['mean','sum'],
    'most_recent_purchases_range_mode_c_B' : ['mean','sum'],
    'most_recent_purchases_range_mode_c_C' : ['mean','sum'],
    'most_recent_purchases_range_mode_c_D' : ['mean','sum'],
    'most_recent_purchases_range_mode_c_E' : ['mean','sum'],
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

```
    'avg_sales_lag3_mean' : ['sum', 'mean', 'max', 'min',std],
    'avg_purchases_lag3_mean':  ['sum', 'mean', 'max', 'min',std],
    'active_months_lag3_mean':['sum', 'mean', 'max', 'min',std],

    'avg_sales_lag6_mean':['sum', 'mean', 'max', 'min',std],
    'avg_purchases_lag6_mean':['sum', 'mean', 'max', 'min',std],
    'active_months_lag6_mean': ['sum', 'mean', 'max', 'min',std],

    'avg_sales_lag12_mean':['sum', 'mean', 'max', 'min',std],
    'avg_purchases_lag12_mean': ['sum', 'mean', 'max', 'min',std],
    'active_months_lag12_mean': ['sum', 'mean', 'max', 'min',std],

    'category_4_mode_c_N' : ['mean','sum'],
    'category_4_mode_c_Y' : ['mean','sum']

     }

    df_n = df.groupby('card_id').agg(agg_func)

    df_tr = (df.groupby('card_id')\
            .size()\
            .reset_index(name='transactions_count'))

    df_n.columns = ['_'.join(col).strip() for col in df_n.columns.values]

    df_n = df_n.merge(df_tr, how = 'left', on = 'card_id')



    return df_n
```

And another feature kernel on this merged transaction data frame **aggregate_per_month,** which is basically for grouped on card id and month lag then aggerated on installment and purchase amount.

```
def aggregate_per_month(df):
    grouped = df.groupby(['card_id', 'month_lag'])

    agg_func = {
            'purchase_amount': ['count', 'sum', 'mean', 'min', 'max',std],
            'installments': ['count', 'sum', 'mean', 'min', 'max',std],
            }

    intermediate_group = grouped.agg(agg_func)
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

```
    intermediate_group.columns = ['_'.join(col).strip() for col in intermed
iate_group.columns.values]
    intermediate_group.reset_index(inplace=True)

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

    return final_group
```

Modeling:

From the research done with the already available solutions and kernels, the simple model like SVM regression, KNN doesn't work better for the prediction of the target value. So, I jumped into the complicated model like GBDT.

At first I used models like random forest and LGBM model among these two I got better result with LGBM model so I went with this base model for my further model building process.

In later stages I used k-fold and stratified k fold to get oof1 and oof2 features then using these two features I built one staked model .

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

k-fold model

```
folds = KFold(n_splits=5, shuffle=True, random_state=15)
oof1 = np.zeros(len(train))
predictions_kf = np.zeros(len(test))
#start = time.time()
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, train.target.values )):
    print("fold n°{}".format(fold_))
    trn_data = lgb.Dataset(train.iloc[trn_idx][features],
                           label= train.target.iloc[trn_idx]
                           #categorical_feature=categorical_feats
                          )
    val_data = lgb.Dataset(train.iloc[val_idx][features],
                           label= train.target.iloc[val_idx]
                           #categorical_feature=categorical_feats
                          )
    num_round = 10000
    clf = lgb.train(param,
                    trn_data,
                    num_round,
                    valid_sets = [trn_data, val_data],
                    verbose_eval=100,
                    early_stopping_rounds = 200)

    oof1[val_idx] = clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_iteration)

    fold_importance_df = pd.DataFrame()
    fold_importance_df["feature"] = features
    fold_importance_df["importance"] = clf.feature_importance()
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    predictions_kf += clf.predict(test[features], num_iteration=clf.best_iteration) / folds.n_splits

print("CV score: {:<8.5f}".format(mean_squared_error(oof1, train.target)**0.5))
```

Stratified k-fold model

```
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=4590)
oof2 = np.zeros(len(train))
predictions_skf = np.zeros(len(test))
#start = time.time()
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train,train['rare_value'].values)):
    print("fold {}".format(fold_))
    trn_data = lgb.Dataset(train.iloc[trn_idx][features], label=target.iloc[trn_idx])
    val_data = lgb.Dataset(train.iloc[val_idx][features], label=target.iloc[val_idx])

    num_round = 10000
    clf = lgb.train(param, trn_data, num_round, valid_sets = [trn_data, val_data], verbose_eval=100, early_stopping_rounds = 300)
    oof2[val_idx] = clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_iteration)


    predictions_skf += clf.predict(test[features], num_iteration=clf.best_iteration) / folds.n_splits

np.sqrt(mean_squared_error(oof2, target))
```

`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

Stacking with stratifed kfold with oof1 & oof2

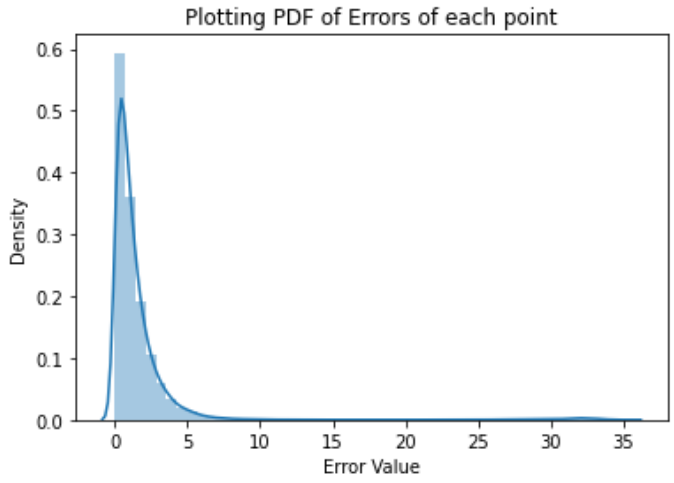
```
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import Ridge

train_stack = np.vstack([oof1, oof2]).transpose()
test_stack = np.vstack([predictions_kf, predictions_skf]).transpose()

folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=15)
oof_stack = np.zeros(train_stack.shape[0])
predictions_stack = np.zeros(test_stack.shape[0])

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train_stack, train['rare_value'].values)):
    print("fold n°{}".format(fold_))
    trn_data, trn_y = train_stack[trn_idx], target.iloc[trn_idx].values
    val_data, val_y = train_stack[val_idx], target.iloc[val_idx].values

    clf = Ridge(alpha=1)
    clf.fit(trn_data, trn_y)

    oof_stack[val_idx] = clf.predict(val_data)
    predictions_stack += clf.predict(test_stack) / folds.n_splits

np.sqrt(mean_squared_error(target.values, oof_stack))
```

```
fold n°0
fold n°1
fold n°2
fold n°3
fold n°4
3.700448110974923
```

MODEL	PRIVATE SCORE	PUBLIC SCORE
RANDOM FOREST	3.69694	3.798669
LGBM	3.66822	3.7625
LGBM with k-fold	3.65965	3.75837
LGBM with stratified k-fold	3.65922	3.75827
STACKED model with Ridge regression	3.66479	3.75600

POST TRAINING ERROR ANALYSIS:

Below we can see a pdf of error values for each points.



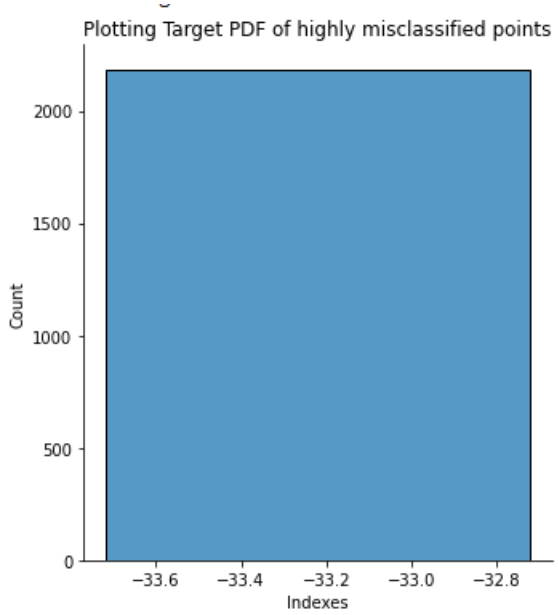
`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

Though from above pdf we can say most of points have error values which is between 0-5 , and I did some analysis to get number points that falls into different range.

```
# Miss classified points with error value greater than 20
index_20 = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 20]
len([i for i in range(len(target)) if abs(target[i] - oof_stack[i]) > 20])
```

```
2185
```




```
[ ] index = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 10 and abs(target[i] - oof_stack[i])<20 ]
    print(" No of points with error value in 10 to 20 range: ", len(index))
```

```
 No of points with error value in 10 to 20 range:  357
```

```
    index = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 5 and abs(target[i] - oof_stack[i])<10 ]
    print(" No of points with error value in 5 to 10 range: ", len(index))
```

```
 No of points with error value in 5 to 10 range:  4356
```

```
[25] index = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 3.5 and abs(target[i] - oof_stack[i])<5 ]
     print(" No of points with error value in 3.5 to 5 range: ", len(index))
```

```
 No of points with error value in 3.5 to 5 range:  8361
```

```
[23] index = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 3.5]
     len(index)
```

```
15259
```

```
    index = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 3.5 and abs(target[i] - oof_stack[i])<10 ]
    len(index)
```

```
12717
```

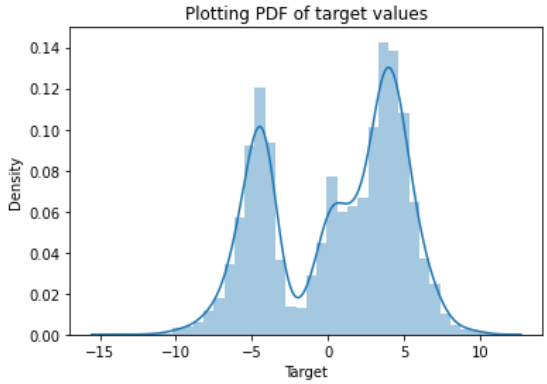
`

Elo Merchant Category Recommendation (MACHINE LEARNIG CASE STUDY)

```
index = [i for i in range(len(target)) if abs(target[i] - oof_stack[i])> 3.5 and abs(target[i] - oof_stack[i])<10 ]
len(index)
```

```
12717
```

```
sns.distplot(target.iloc[index].values).set(title = "Plotting PDF of target values", xlabel = "Target")
plt.show()
```



Conclusion:
So far the best model is the stacked model with LGBM models trained on stratified and non-stratified data with ridge meta learner, gave the Kaggle score 3.66479.

The reason model scores are poor its due to the fact that data over lap is too much and it's very much difficult to separate rare data points from normal data points. Which is affecting rmse scores very much.

Also, from this assignment we can understand how much feature engineering is important in machine learning.

From the above analysis we can see all the highly miss-classified points are in the range below -30.

Future Work:
Since, we can see my model isn't able distinguish rare points from the normal points very well, thus I would like to train one classification model before using any regression model. That will classify the rare points first then train 2 different regression models, One for normal points and another for rare points.

`