Elo Merchant Category Recommendation

“ Help understand customer loyalty “

AML FALL 2023 PROJECT GROUP NO. 14
MEMBER NAMES AND UNI

Harsh Benahalkar	hb2776
Quinn Booth	qab2004
Shan Hui	sh4477
Steven Chase	sc4859

Project Deliverable #2: 07th November 2023

Dataset Handing

The problem comprises of 4 different datasets:
- historical_transactions.csv
- merchants.csv,
- new_merchants_transactions.csv
- train.csv

Datasets are so large that at least 16GB of system RAM is required to load and perform just basic processing on all 4 datasets.

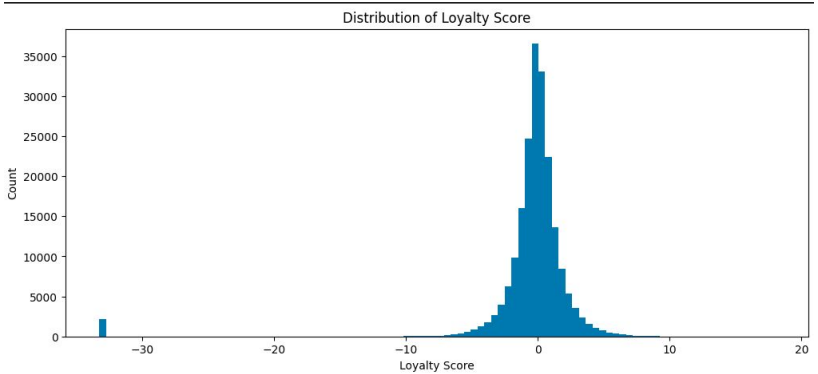
One solution to this problem is to implement a memory reduction algorithm.

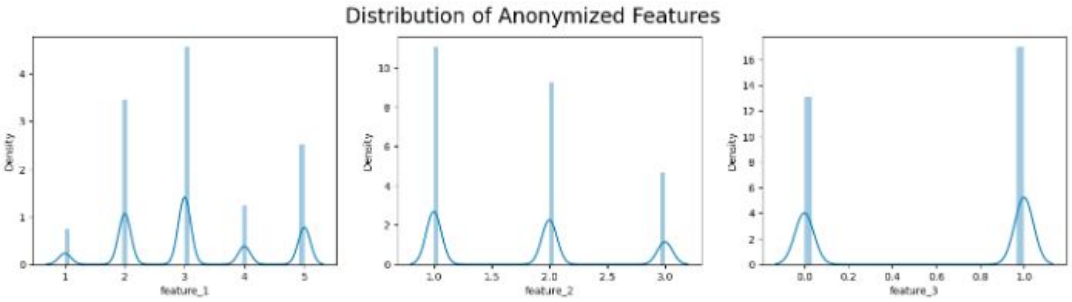
On implementing the algorithm on the pandas dataframes, the memory consumption of the datasets decreased by approximately 52%.

ALGORITHM IMPLEMENTED FOR MEMORY REDUCTION

```
load the dataset in chunks of a pre-fixed size

iterate and perform memory reduction on a single chunk

determine all columns present

for each column,

  check datatype of all data present in the column

  downcast to a lower "int" if datatype is "int"

  downcast to a lower "float" if datatype is "float"

  let "object" variables be as it is

append this chunk to a final dataframe
```

Data Exploration - Train dataset



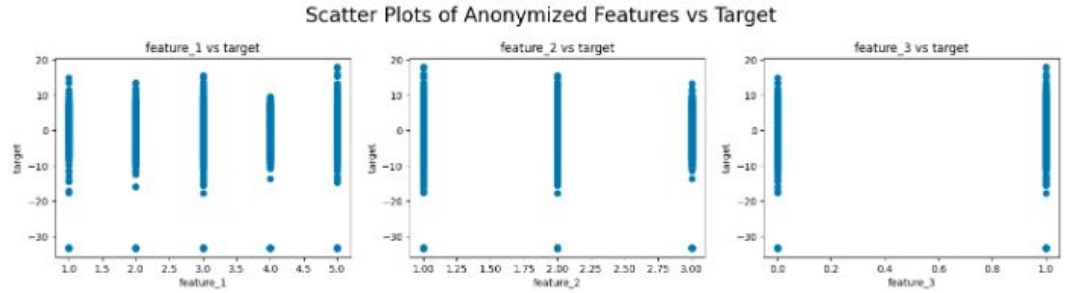


Exploring the distribution of loyalty score, there may be some outliers we need to handle

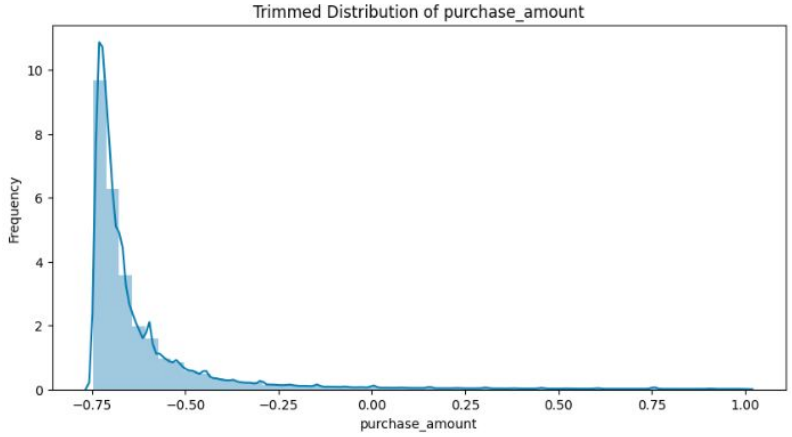
Other than that, the data is centered around 0 with a range of (-10,10)

Visualising the distribution of the anonymized features doesn't offer use much insight into what they could be

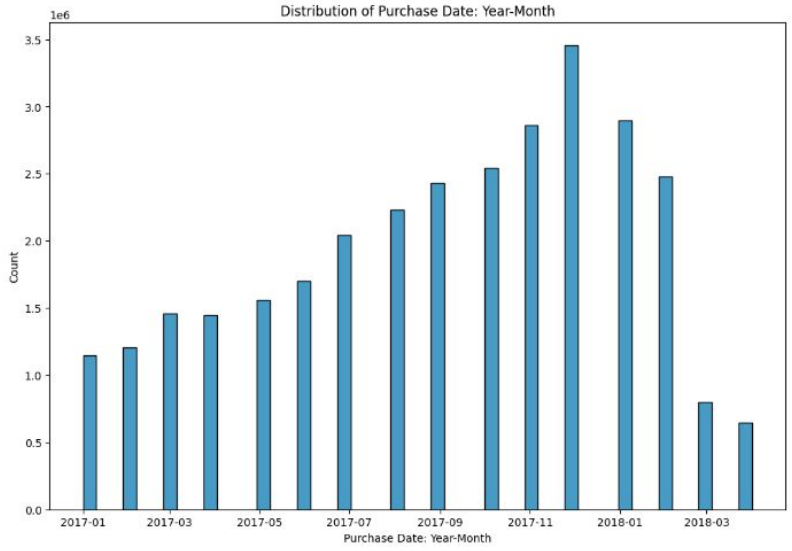
Additionally, plotting them against the target doesn't show us much



Data Exploration



- Plotting the distribution of the purchase amount we can see most transactions are close to 0
- Only visualizing values less than 1 allows us to zoom in and get a better understanding of the distribution of the data closer to 0
- Clearly the data has been normalized in some manner
- Viewing the distribution of the purchase date allows us to understand the timeframe the data was collected



Insights from Data Exploration

- Feature engineering will be key to building a successful model. The transactions dataset can be used to aggregate transaction metrics for each card_id
- For the time being, we have decided the merchant dataset doesn't hold valuable information for predicting the loyalty of a customer
- There are 6 anonymized categories of which we don't know what the values refer to. They are: feature_1, feature_2, feature_3, category_1, category_2, category_3
- The purchase amount has been normalized already
- There may be some outliers with below a -30 loyalty score

Cleaning Missing values

After merging historical data and new merchant transactions we needed to handle missing values

Missing values:

```
shape: (31075392, 15)

authorized_flag              0
card_id                      0
city_id                      0
category_1                   0
installments                 0
category_3              234081
merchant_category_id         0
merchant_id             164697
month_lag                    0
purchase_amount              0
purchase_date                0
category_2             2764609
state_id                     0
subsector_id                 0
new_merchant_flag            0
dtype: int64
```

1) We choose to drop rows with missing merchant data as it was less than 0.5% and we couldn't tie these purchases back to a merchant
2) We then chose to not impute the missing values for 'category_3' and 'category_2' instead recording the fact that these values were missing as it could be informative
3) We then hot encoded these two categories as the cardinality was low and we did not know if there was an order to the values

This gave us a clean transactions dataset to work with

Cleaned dataset

```
card_id                      0
city_id                      0
installments                 0
merchant_category_id         0
merchant_id                  0
month_lag                    0
purchase_amount              0
purchase_date                0
state_id                     0
subsector_id                 0
new_merchant_flag            0
purchase_month               0
purchase_year                0
purchase_hour_section        0
purchase_day                 0
authorized_flag_Y            0
category_1_Y                 0
category_2_2.0               0
category_2_3.0               0
category_2_4.0               0
category_2_5.0               0
category_2_NA                0
category_3_B                 0
category_3_C                 0
category_3_NA                0
```

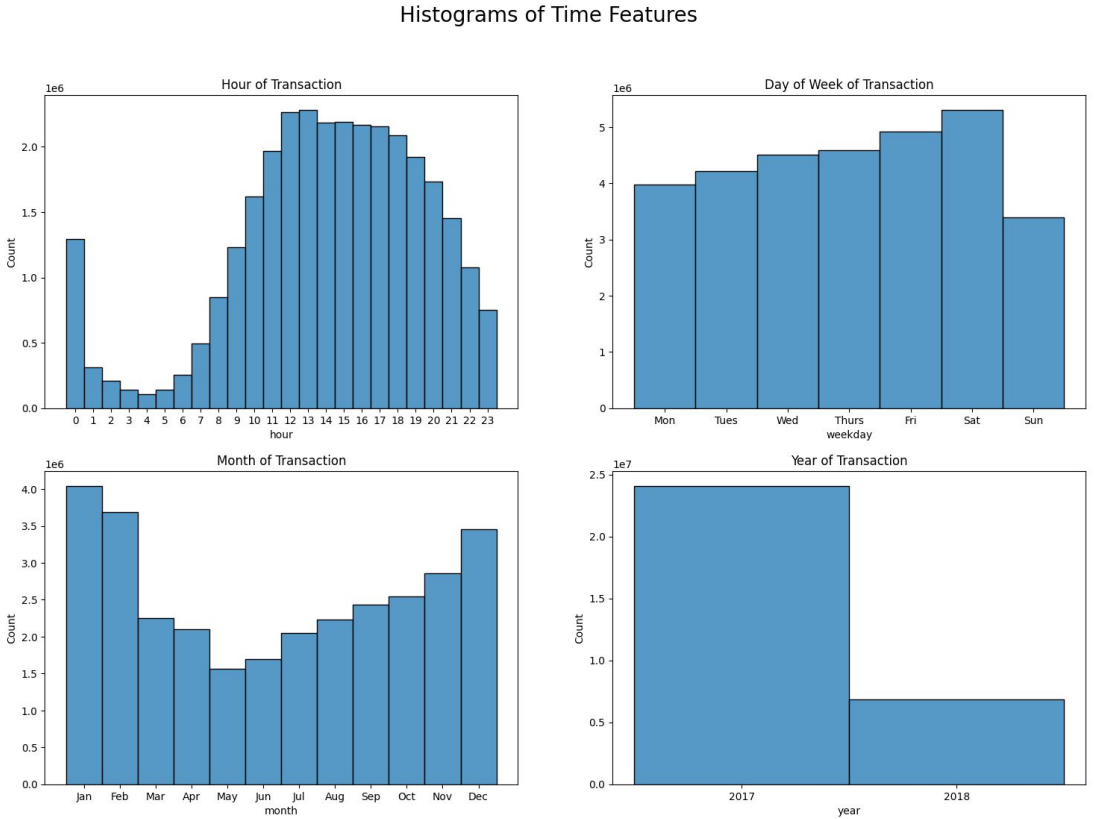
Cleaning

On further cleaning,

The purchase_date column gave us time series information which could be useful for feature engineering.

We transformed the purchase_date to datetime format and split out the following columns

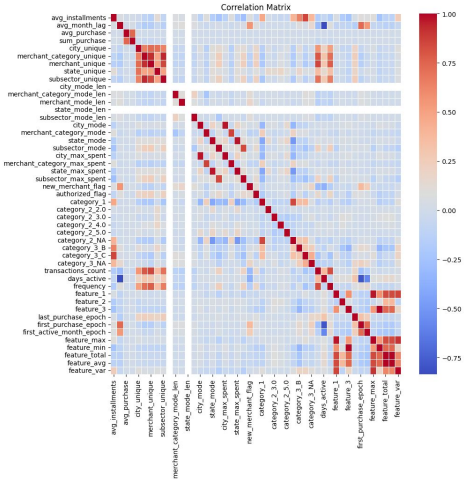
- purchase_month
- purchase_year
- purchase_day
- purchase_hour_section



Feature Engineering

Most of our engineered features came from aggregating the transaction dataset and creating metrics for each card_id

The table to the right describes how we aggregated various columns in the transaction dataset to create one or more features



On plotting the correlation matrix, we can see “light” and “moderate” correlation between the majority of features, with some features showing “heavy” correlation.

Feature	Aggregation Features
card_id	index
city_id	1) Number of unique
	2) Mode (if tied, take first value)
	3) Number of tied mode values
	4) value with max spent
merchant_category_id	1) Number of unique
	2) Mode (if tied, take first value)
	3) Number of tied mode values
	4) value with max spent
merchant_id	1) Number of unique
	2) Mode (if tied, take first value)
	3) Number of tied mode values
	4) value with max spent
state_id	1) Number of unique
	2) Mode (if tied, take first value)
	3) Number of tied mode values
	4) value with max spent
subsector_id	1) Number of unique
	2) Mode (if tied, take first value)
	3) Number of tied mode values
	4) value with max spent
category_1	percentage
category_2	percentage
category_3	percentage
new_merchant_flag	percentage of transactions that come from the new_merchant_period dataset
authorized_flag	percentage of authorized transactions
month_lag	Average purchase month lag
purchase_date	1) Transaction count
	2) first purchase date
	3) most recent purchase date
	4) days as an active user
	5) frequency of transactions (trans count / days active)
installments	Average installments
purchase_amount	1) Average purchase amount
	2) total of all purchases

Baseline Machine Learning Models - Linear Model

```
# Parameter grid for Ridge Regression
param_grid = {
    'alpha': [ 0.005, 0.007, 0.01, 0.02],
    'solver': ['auto', 'svd',  'lsqr', 'sag', 'saga']
}

# Initialize Ridge Regression model
ridge = Ridge(random_state=1)

# Initialize GridSearch with 5-fold cross-validation
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Best estimator
best_ridge = grid_search.best_estimator_

# Predict on the test set
y_pred = best_ridge.predict(X_test_scaled)

# Evaluate the model
rmse = sqrt(mean_squared_error(y_test, y_pred))

print("Best parameters: {}".format(grid_search.best_params_))
print("Root Mean Squared Error: {:.4f}".format(rmse))
```

```
Best parameters: {'alpha': 0.005, 'solver': 'saga'}
Root Mean Squared Error: 3.8348
```

Trained a simple ridge model to compare the performance with the boosting model.

The RMSE of ridge regression model on test set is 3.8348, which is a little high.

XGBoost model apparently performs better than ridge model. We should consider boosting models and NN models in the following steps, such as Lightgbm, LSTM and RNN.

Baseline Machine Learning Models - XGBoost Regressor

We selected an XGBoost Regressor to get an estimate of our evaluation.

Training RMSE of XGBR trained on modest feature selection - 3.8021

Training RMSE of XGBR with “added” features and KFold sampling - 1.6286

On performing hyper-parameter tuning on XGBR, we can achieve an improvement in the model performance.

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
XGB_train_RMSE, XGB_test_RMSE = [], []

XGBR_start_time = time.time()
XGBmodel = XGBRegressor(objective='reg:squarederror')
for train_index, test_index in kf.split(df_X):
    X_train, X_test = df_X.iloc[train_index], df_X.iloc[test_index]
    y_train, y_test = df_Y[train_index], df_Y[test_index]

    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    XGBmodel.fit(X_train, y_train)
    train_y_pred = XGBmodel.predict(X_train)
    test_y_pred = XGBmodel.predict(X_test)

    XGBR_train_accuracy = metrics.mean_squared_error(y_train, train_y_pred, squared=False)
    XGBR_test_accuracy = metrics.mean_squared_error(y_test, test_y_pred, squared=False)

    XGB_train_RMSE.append(XGBR_train_accuracy)
    XGB_test_RMSE.append(XGBR_test_accuracy)
XGBR_model_time = time.time() - XGBR_start_time
print("Time to train model:", XGBR_model_time)

print("Training RMSE:", XGB_train_RMSE)
print("Mean Training RMSE:", np.mean(XGB_train_RMSE))
print("Testing RMSE:", XGB_test_RMSE)
print("Mean Testing RMSE:", np.mean(XGB_test_RMSE))
```

```
Time to train model: 24.83251404762268
Training RMSE: [1.4986286065500707, 1.4941580183647727, 1.491350908108202, 1.4917318428799835, 1.4933320601004894]
Mean Training RMSE: 1.4938402872007035
Testing RMSE: [1.6094066881803721, 1.6306172233765954, 1.638694236326484, 1.6313334920080145, 1.6333709839008086]
Mean Testing RMSE: 1.6286845247584547
```