Exploring Credit Card Transaction Fraud Through Machine Learning Algorithms

Ziyi Gao, Bingxin Li, David Shin,
Yamato Tadokoro, Lizhu Wang, Zhiyu Zhang

Team 106

DSO 562 Fraud Analytics

Professor Stephen Coggeshall

May 5, 2022

Table of Contents

Executive Summary

The damages from Credit Card Transaction Fraud to merchants, banks, and private cardholders has been increasing for years both in the United States and globally. Among several countries, the U.S. has the greatest losses from fraud transactions. According to the Nilson Report 2021, in 2020, the U.S. lost 10.24 billion dollars on transaction frauds, increasing from 9.62 billion in 2019. In 2020, the U.S. transaction fraud amount represented 35.83% of the global transaction fraud amount, increasing from 33.58% in 2019. In contrast, the U.S. accounted for only 22% of the total global card volume. Card transaction fraud is also harming countries outside of the U.S. According to the projection made by the Nilson Report, the total volume of global fraud transactions was about $28.5 billion and is going to increase to $49.3 billion by 2030. Although the fraud cent per $100 will decrease from 6.81 in 2021 to 6.23 in 2030, the total volume will still bring a tremendous amount of losses. These high fraud losses were due to the broader usage of cards not present payment methods, such as transactions made online, through phones, or by mail. Therefore, to reduce the fraud damages brought by these transactions, our group built a real-time supervised model based on historical labeled credit card transactions data from a US government organization to help detect future credit transaction frauds.

We broke up model building into three steps, which were data exploration, data preparation, and model building. In the data exploration section, we examined the dataset by identifying each variable, calculating the minimum, maximum, the percentage of null values for the numeric variables, and calculating the number of unique values, the most common field values for the categorical variables.

Then, we did data preparation with data cleaning and variable creation. For data cleaning, we filtered transactions by only including transactions with type “P”, and filled most missing values by mapping with other features. With the cleaned data, we created 1012 variables through target encodings, applying Benford’s Law, and feature combinations based on domain experts’ insights.

Lastly, we selected 20 variables from the 1012 created variables to build the final models. For the selection process, we first used the KS method to select 80 variables and used wrapper–forward selection to select the top 20 best variables for model buildings. After comparing 5 models with different hyperparameters, we decided to use LightGBM as our final model for its high fraud detection rate (FDR) at 3% and relatively low overfitting problem. Using this model, we can eliminate 56.48% of fraud transactions, by declining 3% of the transactions. With this model and the 3% threshold, we will be able to save the company around $190K per year.

1. Description of Data

The data used in this project is actual credit card purchases from a US government organization. The dataset covers the time from January 1st, 2006, to December 31st, 2006. There are 10 fields and 96,753 records. The dataset has the following 10 fields:

1. Recnum – The record number or unique index of the transactions
2. Cardnum – The cardnum of the transaction
3. Date – The date of the transaction was placed
4. Merchnum – The merchant number of the transaction
5. Merch Description – The merch description of the transaction
6. Merch state – The merchant state of the transaction
7. Merch zip – The merchant zip code of the transaction
8. Transtype – The type of the transaction
9. Amount – The dollar amount of the transaction
10. Fraud – Indicates whether the application was detected as fraud | 0 = not, 1 = yes

1.1 Categorical Fields Summary

Field Name	% Populated	# Unique Values	Mode
Recnum	100%	96,753	NA
Cardnum	100%	1,654	5142148452
Merchnum	96.51%	13,091	930090121224
Merch description	100%	13,126	GSA-FSS-ADV
Merch state	98.76%	227	TN
Merch zip	95.19%	4,567	38118.0
Transtype	100%	4	P
Fraud	100%	2	0

1.2 Numerical Fields Summary

Field Name	% Populated	Min	Max	Mode	Mean	Stdev	% Zero
date	100%	2006-01-01	2006-12-31	2006-02-28	N/A	N/A	0
Amount	100%	0.01	3102045.53	3.62	427.89	10006.14	0

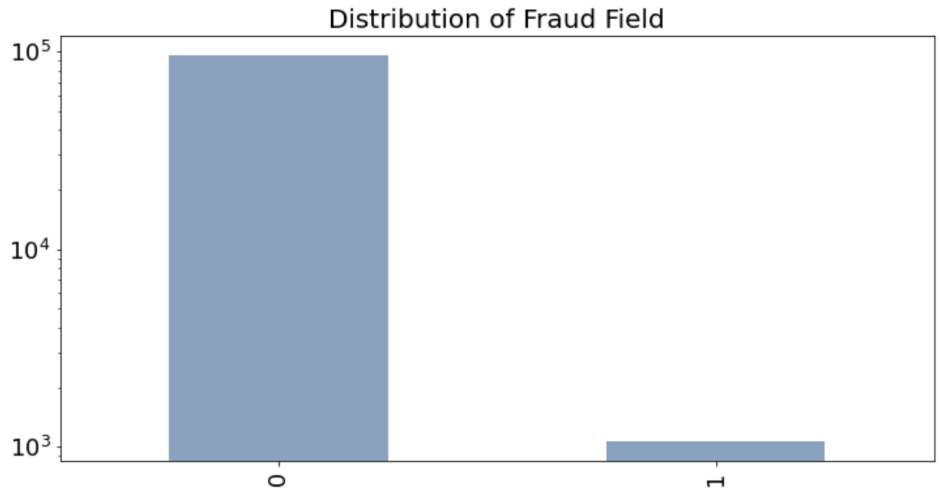


Figure 1. Count-plot of Fraud Field (scale: log)

We first need to understand the background behind credit card transaction fraud. Looking at individual transactions on paper, it is extremely difficult to tell if a transaction is fraudulent from an outside perspective. Our fundamental strength is being able to compare these individual transactions against many others to see if a transaction is relatively different than normal. In this case, understanding "normal" means answering questions such as does this cardholder normally use their card this often? Do they normally spend this amount? Isn't this store very far from their normal locations? On the other side, there are dishonest employees and merchants that create fraudulent transactions and even whole companies. To identify these cases, it's pertinent to ask how we can measure the fictitious nature of a transaction's card number or merchant ID. Answering these questions begins with framing them in terms of our variables. The distributions of the Cardnum, Merchnum, Merch State, and Amount variables are shown in the following pages.

1.3 Field Distributions

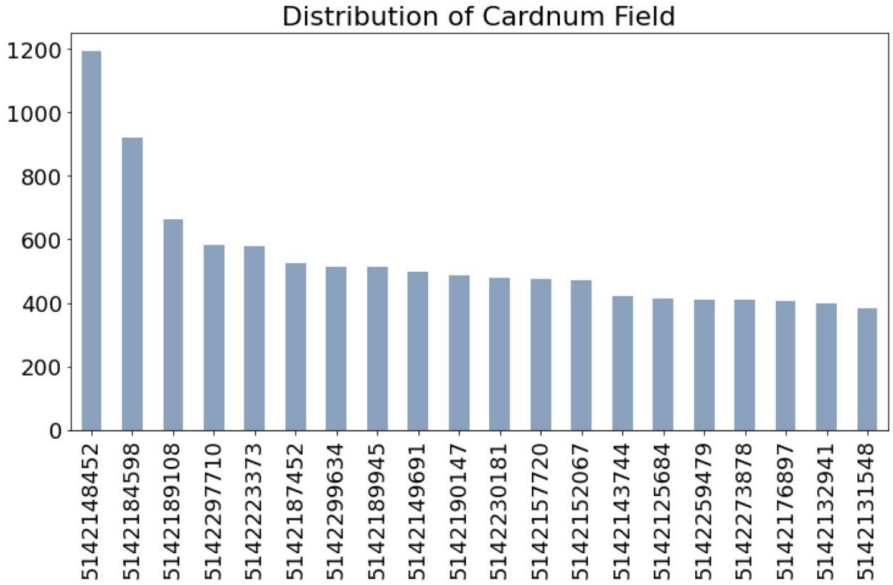


Figure 2. Count-plot of Cardnum Field

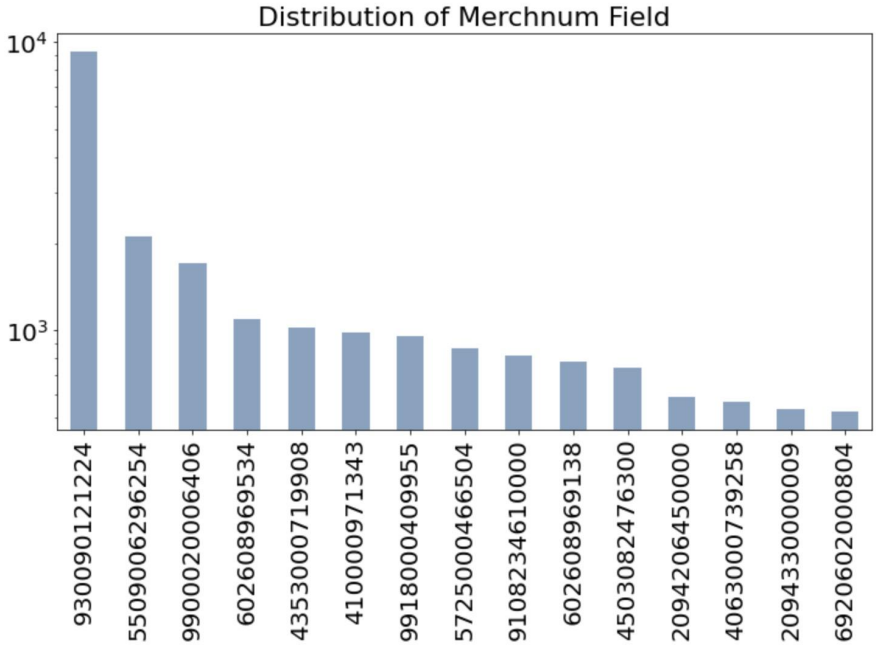


Figure 3. Count-plot of Merchnum Field (scale:log)

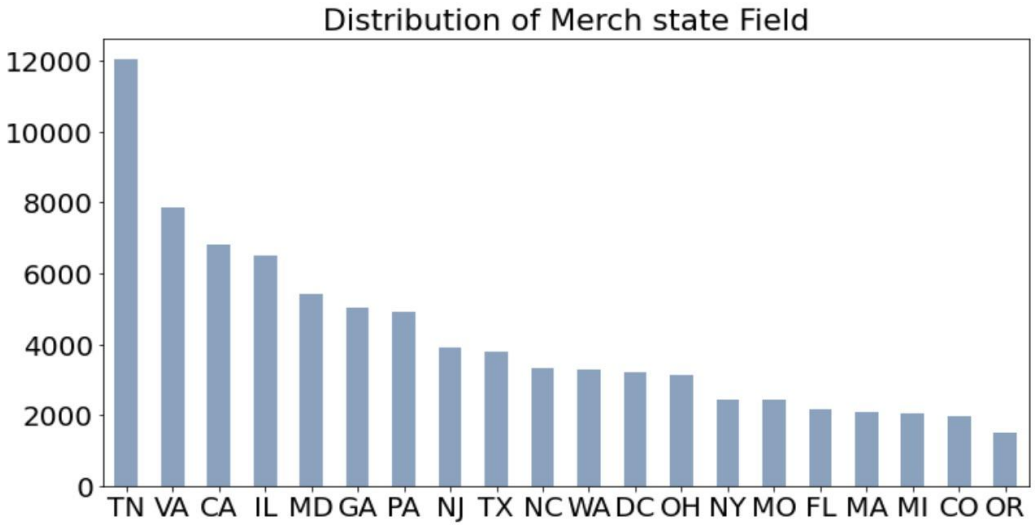


Figure 4. Count-plot of Merch state Field

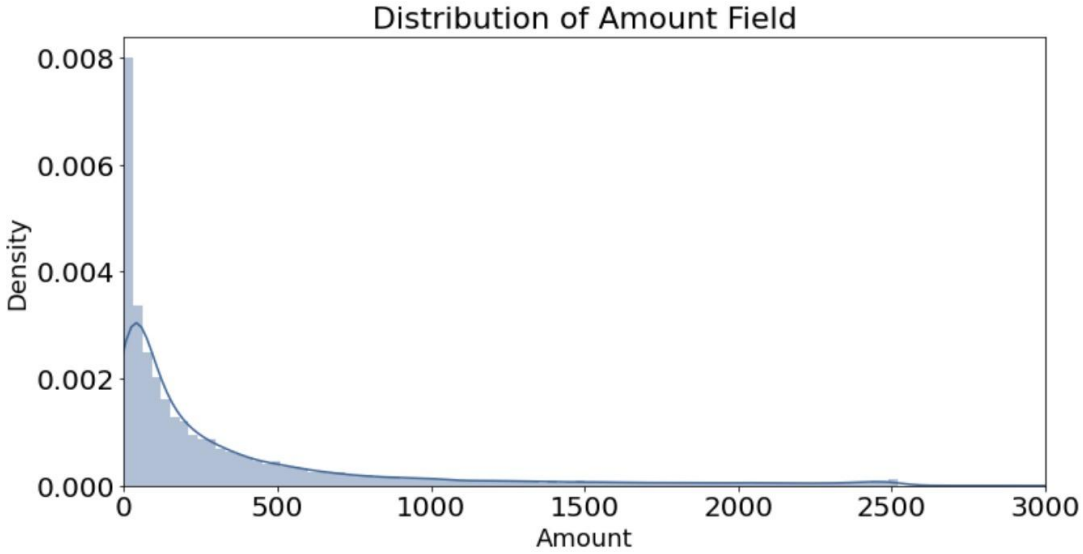


Figure 5. Density-plot of Amount Field (x-max to 3000)

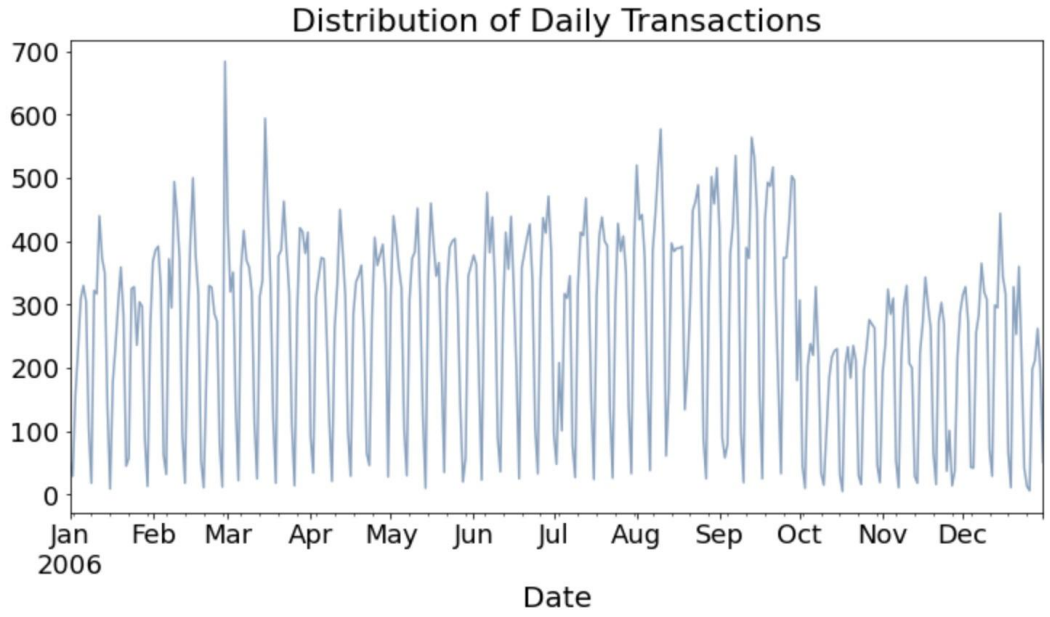


Figure 6. Count-plot of Transactions on Daily Basis

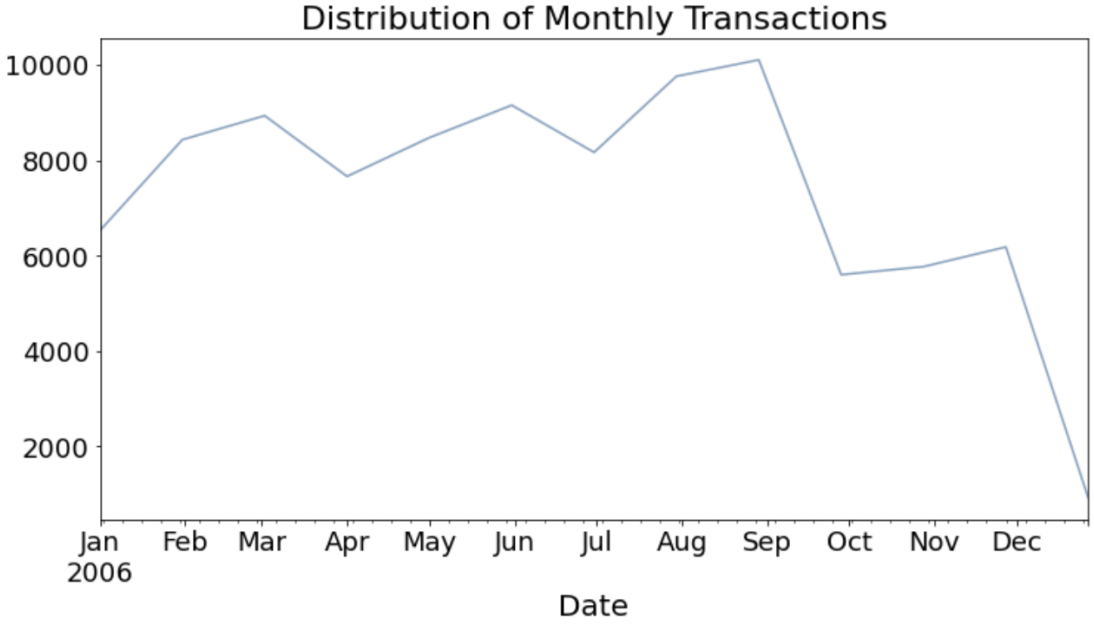


Figure 7. Count-plot of Transactions on Monthly Basis

The above 2 graphs plot the number of transactions on a daily and monthly basis. For the first plot, there is a regular fluctuation in the number, but the trends do not differ a lot. It is mainly because government related organizations are always open on weekdays and closed on weekends and transactions are more likely to happen during the week. In contrast, for the plot on the monthly basis, we can see an obvious drop in September. This phenomenon can be explained by the limitation of the budget at the end of the US Government budget year (always starting from October). Therefore, these trends of the changing number of transactions are reasonable according to the real world situations.

2. Data Cleaning

After we completed the exploratory analysis on the credit card transaction dataset, we needed to perform data manipulation to handle any outliers and missing fields. The first step we did for our data cleaning process was to identify any erroneous records. During our exploration, we noticed that there was one transaction where the dollar amount was in Mexican pesos. As it had the wrong currency and it was five times bigger than the usual records, we treated this single transaction amount as an extreme outlier and excluded this record from the entire process. Since we only focused on the purchased transaction, we kept the transaction type of “P”. Then, we examined all the fields and found that all the fields were 100% populated except the field of Merchnum, Merch state and Merch zip, so we needed to fill in the missing value for these three fields.

2.1 Date Field Reformatting

The original values in the date field were object type and converting this column to the datetime type was pertinent in being able to efficiently organize the data. This was done by converting the date field to datetime type by the pandas to_datetime function.

2.2 Merchnum

To fill in the missing Merchnum, if the value was zero, we replaced it with nan. Then we filled in by mapping with the corresponding Merch description and replaced values where Merch description was “RETAIL DEBIT ADJUSTMENT” with “unknown”. For any remaining null values, we filled it in with “unknown”.

2.3 Merch zip

For the merch zip field, we filled in missing values by mapping to corresponding Merch Zip values for the Merchnum, and Merch description fields. Then we replaced any values where the Merch description was "RETAIL DEBIT ADJUSTMENT" with "unknown. For any remaining null values, we filled in with “unknown”.

2.4 Merch state

For the missing Merch state values, we filled in null values by finding the relative state value for Merch zip, Merchnum, and Merch description columns. We then mapped these values over the

Merch state. For example, if the Merch state was null but had Merch zip, Merch number or Merch description, we filled the missing value with the Merch state associated with one of the listed fields. We also assigned unknown for adjustments transactions where Merch description was "RETAIL DEBIT ADJUSTMENT". If Merch state could not be mapped using the three columns that mentioned above, we filled in the value with “unknown”.

3. Create Candidate Variables

Feature creation is an essential step in balancing the inconsistencies of our data. Before we thought about creating candidate variables, we needed to reference domain experts to learn about the different types of credit card transaction fraud. Identifying the different types is crucial in understanding what types of signals to be looking for when building candidate variables. For example, one type of fraud is when a victim loses their card or has it stolen. The fraudster would try to use the card as much as possible before it's turned off, so we would likely see bursts of activity at different and non-common merchants, merchants in different geographical areas, and larger than average purchase amounts. On the other side, there are also fraudulent employees and merchants that invent fictitious transactions and companies. This type of fraud is difficult to recognize using only fraud signals, so we needed to create variables that assessed whether the card and merchant numbers were made up or real.

As we were working with a smaller dataset with about 100,000 records, we needed to create as many variables as possible using the entity fields (cardnum, merchnum, merch_description, merch_state, merch_zip) from the original dataset. To hone in on the previously mentioned fraud signals, we created candidate variables that looked at the number and amount of transactions over different time periods. We knew that the amount fraudulers spent each time was highly correlated to the psychological factors (not too much to incur police supervision and not too little to incur high risk cost), so we created a new variable called amount type, where we sectioned the amount field into 6 different sections and made categorical labels for each bin.

3.1 Linking Entities

From our preliminary exploration, it was apparent that the individual fields for each application would not be a dependable indicator for identifying people, as many applications could have the same name or zip code. In order to create candidate variables, we created combination groups of the identity fields to create linking variables. On top of using the cardnum and merchnum fields, we created the cardnum_merchnum, cardnum_merstate and cardnum_merzip by concatenating the cardnum field with these others. We also combined our new amount type field with the other groupings to have the cardnum_amount_type, merchant_amount_type and state_amount_type entities. Then for each entity and combination group, we created amount, velocity, days since last seen, and relative velocity variables to detect possible frauds. Using the linking entities, we created the following variables:

3.2 Amount

The amount variables looked at various statistics of the amount field for each entity over the past N (1, 7, 14, 30) days. The amount variables used the following statistics:

Maximum	**Median**
Total	**Actual - Average**
Actual - Median	**Actual / Average**
Actual / Max	**Actual / Total**
Actual / Median	

3.3 Velocity

Velocity refers to how many times a combo group of entities has appeared over the past n days where n can be 0, 1, 3, 7, 14 or any other numbers. Velocity is a measure of how frequently the combined group appears in the application.

3.4 Days since last seen

Among the combination groups of entities, the days since last seen variable indicates how many days have passed between consecutive appearances of that entity in the data.

3.5 Relative velocity

A relative velocity variable was computed by dividing the number of applications with a specific combination group in the recent past (0 or 1 day) by the number of applications with that same combination group in the past n days where n can be 3, 7, 14 or 30 days. It is a ratio of the short-term velocity to a long-term averaged velocity.

3.6 Target Encoding

In order to better utilize our categorical variables, we utilized target encoding. Target encoding is a process that replaces a categorical value with an average for the target variable. For two of our fields, we created a risk table, which is a reference that contains the average fraud proportion for each categorical value. In order to ensure sufficient sample sizes and avoid overfitting, we put the values through a logistic smoothing formula, and then assigned the relative value to each transaction.

3.6.1 Day of Week Field

To look further into the date field, we created a Day of Week (dow) column by extracting the day from the date field. Since we could only use numerical variables, we utilized target encoding to convert each day into an average. The risk table variable was built by calculating the fraud proportion averages, processing the values through a statistical smoothing formula and then mapping it onto the Day of Week column. The new variable was called dow_risk.

3.6.2 Amount Type Field

As previously mentioned, we created a new field called amount type that divided the amount field into 6 categorical bins. Furthermore, we utilized target encoding for the amount type to get the average fraud proportion for each bin to see if a certain price range had a higher chance of fraud. The new variable was called atype_risk. The categories for the amount type field are:

- <= 5: Extremely Small Amount
- <= 20: Small Amount
- <= 50: Normal Amount
- <= 150: Relatively Large Amount
- <= 500: Large Amount
- > 500: Extremely Large Amount

3.7 Benford's Law Variables

In order to find fraudulent merchants who are making false transactions, we utilized Benford's Law, a theory that states that the leading digit of real-life numbers is likely to be small. This is useful for identifying made-up transactions because we can compare the distribution of the leading digits of the financial information in these statements to the Benford's Law Distribution and see how different they are. Due to the number of records, we combined the nine bins into two: 1 through 2, and 3 through 9, where 1 through 2 made up 47.7% of the distribution and 3 through 9 made up 52.3%. Then, using Dr. Steve's Pretty Good Benford's Law Measure, we counted the number of first digits beginning with 1 or 2, calculated the R value, and set the measure of unusualness as the Max of R and 1/R. Finally, to reduce randomness, we put the value through a smoothing formula.

3.7.1 Cardnum U*

Smoothed measure of unusualness that quantifies how different the first digit distribution of the individual card numbers was compared to the Benford's Law Distribution. A high U* value indicates that maybe the card is generating fraudulent transactions.

3.7.2 Merchnum U*

Smoothed measure of unusualness that quantifies how different the first digit distribution of the individual merchant numbers was compared to the Benford's Law Distribution. A high U* value indicates that maybe the merchant is generating fraudulent transactions.

4. Feature Selection Process

After creating 1012 potential features based on the 10 given features, we continued for the feature selection. We filtered the features by using Kolmogorov-Smirnov (KS) first to select 80 relatively important features and then used a forward selection wrapper to select top 20 variables for further model building. This is the general workflow of the feature selection process.

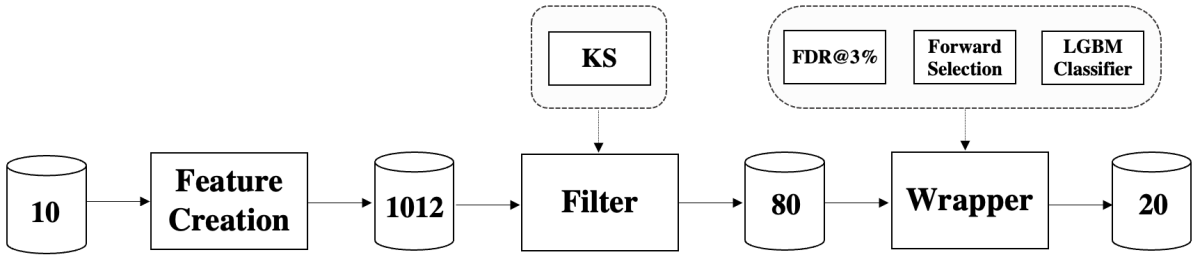


Figure 8. Workflow of feature selection.

4.1 Kolmogorov-Smirnov filter

Kolmogorov-Smirnov (KS) is a robust measure of how well two distributions are separated. For choosing the best variables to build models for fraud application detection, variables that can separate the good applications most from the bad applications are the best variables.

To calculate the separation or the KS value manually, we first removed the last two months of the records as the out-of-time data. For each possible value of the variable, we subtracted the proportion of bad applications from the proportion of good applications to get the difference. Next, we selected the max differences among all values' differences as the KS score of this variable. Lastly, after we calculated all KS scores for the variables, we ranked them numerically and selected 80 variables with the highest KS scores.

$$KS = \max_{x} \left| \int_{x_{min}}^{x} P_{\text{good}}\, dx - \int_{x_{mi}}^{x} P_{\text{bad}}\, dx \right|$$

4.2 Wrapper – Forward Selection

With the selected 80 variables with the highest KS scores, we selected the top 20 variables by using a wrapper of forward selection. For the forward selecting wrapper, we started with an empty model and used LightGBM to add one more variable each time. The model kept the best variables it chose previously, and added one more variable that performs the best. The performance was measured by the fraud detection rate (FDR) at 3%, which was also used for measuring the performance of the models we built. The wrapper not only helps find the most valuable variables to differentiate the good and bad applications, but also helps remove correlated variables by only choosing one of the correlated variables in the model.

For the FDR score at 3%, it is a measure of the performance of a classifier. Higher the FDR score, the better the classifier is. To calculate this score, we first ranked the fraud scores calculated by the classifier of all records in descending order, and we cut the records into 100 bins. We selected the top 3 bins (3%) and compared the total number of fraud labels in these bins with the total number of frauds in the dataset. The higher the ratio is, the better the model is performing. In the wrapper, each time we added a new variable, we calculated the FDR score after adding one of the variables into the dataset and selected the variable that generated the highest FDR score of the classifier.

Lastly, after building the classifier with top 20 performing variables, we ranked these variables according to their rank being added into the model. With these ranked variables, we could more easily select the best performing number of variables for further model building and testing.

4.3 Final 20 Variables with Their Descriptions

After calculating the KS scores and running a wrapper, we reduced our variables to top 20 ones. Here is the table of the variables listed in the order of being added into the wrapper with their individual KS score.

Variable	KS Score	Variable	KS Score
Cardnum_Merchnum_total_7	0.6813	Cardnum_Merzip_total_30	0.6034
Cardnum_Merstate_max_30	0.5982	Merchnum_Atype_total_7	0.5825
Cardnum_Merchnum_total_14	0.6754	Zip_Atype_total_14	0.5816

Cardnum_Merstate_total_14	0.6690	Cardnum_Merzip_avg_1	0.5721
Merchnum_avg_1	0.5743	State_Atype_med_14	0.5767
Cardnum_Merchnum_total_30	0.6592	Cardnum_Merchnum_avg_1	0.5725
Zip_Atype_max_3	0.5707	Cardnum_Merstate_avg_0	0.5898
Cardnum_Merzip_total_14	0.6719	Cardnum_Merchnum_avg_3	0.5735
Cardnum_Merstate_total_30	0.6356	Cardnum_Merchnum_avg_0	0.5732
Cardnum_Merchnum_total_7	0.6565	Cardnum_Merzip_avg_0	0.5904

Table 1. Final Top 20 Variables with KS Score for Each

- Cardnum_Merzip_total_7: amount of transactions made with the same combination of card and merchant zip over the past 7 days
- Cardnum_Merstate_max_30: max transaction amount made by the same combination of card and merchant zip over the past 30 days
- Cardnum_Merchnum_total_14: amount of transactions made with the same combination of card and merchant over the past 14 days
- Cardnum_Merstate_total_14: amount of transactions made with the same combination of card and merchant state over the past 14 days
- Merchnum_avg_1: average amount of transactions made at the merchant over the past 1 day
- Cardnum_Merchnum_total_30: amount of transactions made with the same combination of card and merchant over the past 30 days
- Zip_Atype_max_3: max amount of transactions made in the same combination of zip code and amount type over the past 3 days
- Cardnum_Merzip_total_14: amount of transactions made with the same combination of card and merchant zip over the past 14 days
- Cardnum_Merstate_total_30: amount of transactions made with the same combination of card and merchant state over the past 30 days
- Cardnum_Merzip_total_30: amount of transactions made with the same combination of card and merchant zip over the past 30 days
- Merchnum_Atype_total_7: amount of transactions made with the same combination of merchant and amount type over the past 7 days

- Zip_Atype_total_14: total amount of transactions made with the same combination of zip code and amount type over the past 14 days
- Cardnum_Merzip_avg_1: average amount of transactions made with the same combination of card and merchant zip over the past 1 days
- State_Atype_med_14: median amount of transactions made with the same combination of state and amount type over the past 14 days
- Cardnum_Merchnum_avg_1: average amount of transactions made with the same combination of card and merchant over the past 1 days
- Cardnum_Merstate_avg_0: average amount of transactions made with the same combination of card and merchant state over the past 0 days
- Cardnum_Merchnum_avg_3: average amount of transactions made with the same combination of card and merchant over the past 3 days
- Cardnum_Merchnum_avg_0: average amount of transactions made with the same combination of card and merchant zip over the past 0 days
- Cardnum_Merzip_avg_0: average amount of transactions made with the same combination of card and merchant zip over the past 0 days
- Cardnum_Merzip_avg_3: average amount of transactions made with the same combination of card and merchant zip over the past 3 days

The top 5 variables are: Cardnum_Merchnum_total_7, Cardnum_Merstate_max_30, Cardnum_Merchnum_total_14, Cardnum_Merstate_total_14, and Merchnum_avg_1. All these variables are frequency-related variables. This result is in line with expectations because when a fraudster gets a credit card that does not belong to himself, he needs to make transactions using this card as frequently as possible in a short period of time before the actual owner realizes and freezes this card. So, intuitively, frequency of fraud transactions in a specific period of time for a specific card may represent a bunch of important features for this project.

5. Model Algorithms

This project implemented five different models – Logistic Regression, Decision Tree, K-Nearest Neighbors, LightGBM, and Neural Network. For each model, we have tuned various parameters to achieve the best performance.

Before training the models, data after November 1st, 2006 was reserved as the out-of-time (OOT) data for the evaluation purpose which is about 13% of the whole data. In addition, data before January 14th was also left out, because many of the features would not be effective. This is equivalent to 2,994 rows of data. We then randomly split the rest of the data into the training and test data sets, where the training set contains 70% of the data and the testing set contains 30%. Since this project is a classification problem, we used the stratifying method when splitting to ensure that each set had a similar proportion of fraud label records.

After training the models, we obtained the probability of each observation being a fraud (class 1) and sorted them in descending order. Then, we calculated the fraud detection rate in the top 3% of the sorted lists for the training, testing, and OOT dataset respectively to evaluate the model performance.

Since we had 20 variables to build the model, we wanted to see if there is a sufficient number of variables to develop a predictive model. Therefore, while tuning various parameters, we tried combinations of using the top 5, 10, 15, and 20 variables for each of the model algorithms. The following five models demonstrate the performance changes as we increased the size of variables.

5.1 Five Models

5.1.1 Logistic Regression

We chose to use the Logistic Regression model as our baseline model. Due to its simplicity and high efficiency, the Logistic Regression algorithm is widely used in most binary classification problems. It estimates the probability of y=1 using a regression function of x, in other words, P(Y=1| X=x). By giving a user-defined decision boundary threshold, the model can automatically classify the observations based on the output probability. Specifically, in logistic regression, the probability of y=1 regress on x using a function in the inverse logit form. Below is the inverse logit function.

$$P(Y=1|\ X=x) = \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}}$$

The table below shows the various parameters we used to test the Logistic Regression model.

- penalty: Any modification on the learning algorithm that is intended to reduce its generalization error but not its training error
- C: The inverse of regularization strength
- solver: Solves optimization problems by successively performing approximate minimization along with coordinate directions or coordinate hyperplanes

Model		Parameters				Average FDR at 3%		
	Iteration	#Variables	penalty	solver	C	Train	Test	OOT
Logistic Regression	1	10	l2	lbfgs	0.6	59.59	59.19	27.99
	2	10	l2	saga	0.6	60.17	58.75	27.77
	3	20	l2	lbfgs	2	60.26	56.38	27.71
	4	10	l1	saga	2	59.71	59.54	27.65
	5	15	l2	lbfgs	0.6	60.07	57.99	27.43
	6	15	l1	saga	2	59.73	58.61	27.32
	7	10	l2	saga	1	59.33	59.54	26.87
	8	15	l1	saga	1	59.66	57.39	26.65

Table 2. Logistic Regression high-level results

The table above is sorted by how well the model performed based on the OOT dataset. We can conclude that the Logistic Regression model reached the best performance when we use the top 10 variables, the 'lbfgs' solver, and apply L2 regularization with inverted strength of 0.6. Looking across all of the top models, the highest score was just under 28% in the OOT score and we could not get the models to become more overfitting and score higher.

5.1.2 Decision Tree

A decision tree algorithm is one of the most used and intuitive machine learning algorithms. From its name, a Decision Tree consists of a tree of decisions to partition the data space into smaller subspaces, where each subspace is given a label. During the training process, the tree is built by splitting one node at a time, and the algorithm scans all possible splits along all axes to optimize the split. Entropy, information gain, and Gini are examples of metrics to measure how good a split is. These all measure the impurity of the resulting two partitions, and the best split point is found to result from the least impurity among partitions. After the tree is built, we then can traverse the tree according to the rules and get the predicted label for new records.

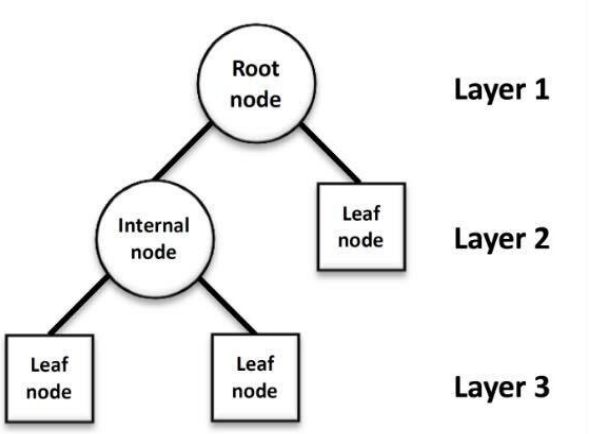


Figure 9. Decision Tree Diagram

A decision tree is commonly used due to its fast training, reliability, and robustness to dirty data; however, it is prone to overfit, where the algorithm can generate over-complex trees that fit training data so perfectly that they do not generalize to unseen data.

There are 4 related hyper-parameters tuned for our decision tree algorithm:

- criterion: The criterion to measure impurity.
- splitter: Cost coefficient for cost-complex pruning.
- max_depth: Maximum depth of the tree.
- min_samples_leaf: Minimal number of samples in each leaf node. If the number is lower, two leaf nodes will be combined.

After tuning, the final setting for parameters is shown in Table 3.

Model		**Parameters**					**Average FDR at 3%**		
	Iteration	#Variables	criterion	splitter	max_depth	min_samples_leaf	Train	Test	OOT
Decision Tree	1	20	gini	best	10	30	78.41	72.37	50.34
	2	15	gini	best	10	30	80.29	73.45	49.89
	3	15	gini	best	10	15	79.59	70.83	49.05
	4	10	gini	best	15	20	89.78	74.1	47.43
	5	10	gini	best	15	30	87.26	73.75	46.59
	6	20	gini	best	10	25	79.54	71.74	46.54
	7	10	gini	best	15	15	90.56	72.31	46.48
	8	10	gini	best	20	20	92.7	72.67	46.26

Table 3. Decision Tree high-level results

The table above is sorted by how well the model performed based on the OOT dataset and listed the top 4 best models and worst 4 models as a comparison. Therefore, we can conclude that the Decision Tree model reached the best performance when we use all of the top 20 variables, the 'gini' criterion, best splitter, max depth of trees to be 10, and minimum samples on a leaf to be 30. Surprisingly, the top 8 models had the Gini criterion and the best splitter.

5.1.3 K-Nearest Neighbor (KNN)

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. The distance calculation can be chosen from Euclidean distance, Hamming distance, Minkowski distance, or even Kullback-Leiber (KL) divergence. Then select the K number of points that are closest to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and the class that holds the highest probability will be selected.

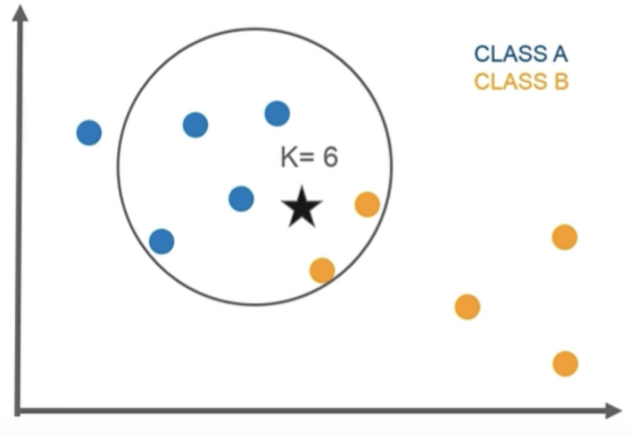


Figure 10. K-Nearest Neighbor Diagram

There are 2 related hyper-parameters tuned for our KNN algorithm:

- n_neighbors: The number of closest neighbors the algorithm chooses from the data point.
- weights: A kernel function that calculates the nearest k points.

Model		Parameters			Average FDR at 3%		
	Iteration	#Variables	n_neighbors	weights	Train	Test	OOT
KNN	1	5	15	distance	100	75.54	39.55
	2	20	15	uniform	88.51	77.16	38.27
	3	5	8	distance	100	73.65	37.26
	4	5	15	uniform	84.8	73.64	36.26
	5	10	8	distance	100	74.43	36.03
	6	15	15	uniform	89.02	76.68	35.92
	7	10	15	distance	100	76.71	35.87
	8	5	10	distance	100	74.77	35.7

Table 4. K-Nearest Neighbor high-level results

The table above is sorted by how well the model performed based on the OOT dataset. We can conclude that the K-Nearest Neighbor model reached the best performance when we used 15 for the number of neighbors and 'distance' as the weights. Overall, the KNN model had an extreme overfitting problem that even the best model did not have an OOT above 40%. Most of the models fit the training data set so well that they could not perform well on unseen data.

5.1.4 Light Gradient Boosting Machine (LightGBM)

Gradient boosting is a machine learning technique that uses an ensemble of weak learners to improve the model performance, which is slightly better than random guesses, sequentially to reduce the bias. One of the most popular Gradient Boosting models is Boosted Decision Trees. The weak learners for the GBDT are the individual trees. All the trees are connected in sequences and each tree tries to minimize the error of the previous tree so that the model improves. The final model aggregates the result of each step and thus creates a strong learner. A loss function is used to detect the residuals.

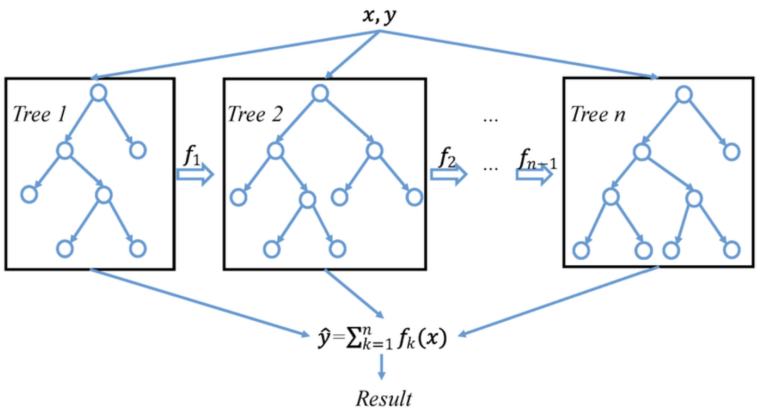


Figure 11. Gradient Boosting Diagram

LightGBM is an improved implementation under the GBDT algorithm framework. The major difference between conventional GBDT and LightGBM lies in the construction of trees. Instead of growing level-wise (horizontal), LightGBM carries out leaf-wise (vertical) growth. Leaf-wise strategy grows the tree by splitting the data at the leaves with the largest loss decrease. Therefore, it results in higher accuracy while being faster. However, leaf-wise can lead to overfitting when used in a small dataset as it produces much more complex trees.

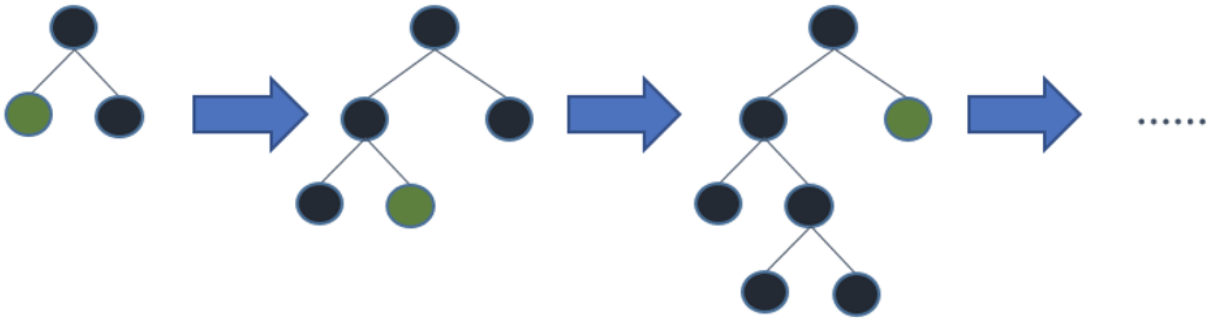


Figure 12. Leaf-wise tree growth

Other than that, LightGBM uses histogram-based algorithms. By binning values into a fixed number of buckets, the algorithm doesn't need to evaluate every single value of the features to compute the split but only the bins of the histogram. As such, this approach dramatically accelerates the construction of the decision tree and requires lesser memory.

There are 4 related hyper-parameters tuned for our LightGBM algorithm:

- learning_rate: a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.
- n_estimators: Number of boosted trees to fit.
- max_depth: Maximum tree depth for base learners.
- subsample: Subsample ratio of the training instance.

Model		Parameters					Average FDR at 3%		
	Iteration	#Variables	n_estimators	learning_rate	max_depth	subsample	Train	Test	OOT
LightGBM	1	20	1000	0.001	4	1	78.2	73.81	56.93
	2	5	1000	0.01	3	1	82.32	77.26	56.87
	3	5	500	0.02	3	0.7	81.74	79.9	56.76
	4	5	500	0.02	3	1	81.65	79.2	56.7
	5	20	1000	0.001	4	0.7	77.33	74.17	56.59
	6	20	100	0.01	4	1	77.71	73.57	56.54
	7	5	100	0.01	4	0.8	76.33	73.86	56.48
	8	15	200	0.02	4	1	87.51	80.33	56.48

Table 5. Light Gradient Boosting Machine high-level results

The table above is sorted by how well the model performed based on the OOT dataset. We can conclude that the Light Gradient Boosting Machine model reached the best performance when we used the learning rate at 0.001, 1000 estimators, max depth of trees to be 4, and subsample to be 1. Overall, the LightGBM model performed very well despite changing different parameters since the top 8 models have an OOT of 56% or higher.

5.1.5 Neural Network

The neural network model is an imitation of the multiple neurons in the human brain. This model is often used in supervised learning to train the model to recognize and distinguish one thing from another through large data sets.

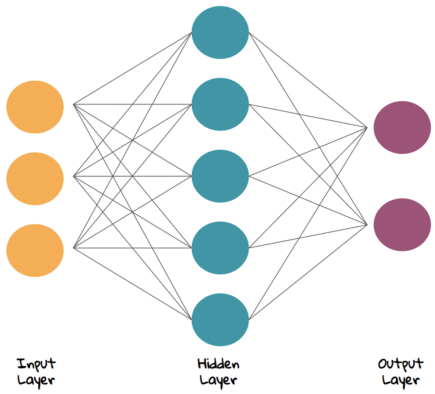


Figure 13. Neural Network Diagram

MLP, which represents Multi-layer Perceptron, is a supervised learning algorithm, and it is the classic neural network model above. In other words, it is a class of feedforward artificial neural networks that use backpropagation for training. It can have multiple layers, and non-linear activation distinguishes multi-layer perceptrons.

There are 3 related hyper-parameters tuned for our Neural Network algorithm:

- Hidden layer size: The number of neurons in the ith hidden layer.
- Max iteration: The maximum number of times a batch of data passed through the algorithm.
- Learning rate: It schedules weight updates.

Model		Parameters				Average FDR at 3%		
	Iteration	#Variables	hidden_layer_size	max_iteration	learning_rate	Train	Test	OOT
Neural Network	1	5	(20, 10)	40	invscaling	70.98	70.94	54.47
	2	5	(20, 10)	60	invscaling	71.71	69.57	54.47
	3	5	(20, 10)	40	constant	71.22	70.57	54.36
	4	5	(10, 5)	100	constant	70.28	69.83	54.3
	5	5	(20, 10)	60	constant	71.79	70.14	54.08
	6	5	(10, 5)	60	constant	70.73	69.27	53.52
	7	5	(10, 5)	40	invscaling	68.82	69.51	52.96
	8	5	(20, 10)	100	constant	72	70.18	52.91

Table 6. Neural Network high-level results

The table above is sorted by how well the model performed based on the OOT dataset. We can conclude that the Neural Network model reached the best performance when we use criterion with 2 layers and 20 & 10 nodes in each layer respectively, the max iteration to be 40, and the learning rate of 'invscaling'. Surprisingly, the top 8 models for the neural network were just using the top 5 variables.

5.2 Model Comparison with 5, 10, 15, and 20 variables

Models	Parameters	Average FDR at 3%		
	#Variables	Train	Test	OOT
Logistic Regression	10	59.59	59.19	27.99
Decision Tree	20	78.41	72.37	50.34
KNN	5	100	75.54	39.55
LightGBM	20	78.2	73.81	56.93
Neural Network	5	70.98	70.94	54.47

Table 7. Best Model Comparison for Each Algorithm

This table shows the best OOT performance for each model. We can see that depending on the algorithm, the size of the variables to achieve the best model performance differs. Looking at the KNN and Neural Network model, even though they both use the top 5 variables, the train, test,

and OOT results are completely different since KNN shows a significant sign of overfitting, but Neural Network does not. Also, Decision Tree and LightGBM use the maximum 20 variables and have a high-performance result.

Next, we will look into each of the model algorithms that use only 5 variables.

Models	Parameters	Average FDR at 3%		
	#Variables	Train	Test	OOT
Logistic Regression	5	59.56	59.85	23.13
Decision Tree	5	80.13	71.97	44.47
KNN	5	100	75.54	39.55
LightGBM	5	76.33	73.86	56.48
Neural Network	5	70.98	70.94	54.47

Table 8. Best Model Comparison for using 5 Variables

From Table 8 results, we can see that starting from a basic linear model, the Logistic Regression model, we will see the OOT increase as we use more advanced and complex non-linear models. Even though using 5 variables seems quite small since will had a total of 1012 variables, the results are acceptable. There isn't a significant difference comparing it to models with more variables. Therefore, we can conclude that by making the model complexity low but at the same time, keeping the model performance high, LightGBM with 5 variables will be the best model for this dataset.

6. Result

6.1 Final Model Results

From table 5, the LightGBM algorithm performed the overall best out of all models. As a matter of fact, this model performed very well in the training and the test dataset as well. For this dataset, it seems non-linear models performed much better than the linear model.
The table below shows the best hyperparameters for this model.

Model	Parameters					Average FDR at 3%		
	#Variables	n_estimators	learning_rate	max_depth	subsample	Train	Test	OOT
LightGBM	5	100	0.01	4	0.8	76.33	73.86	56.48

Table 9. The Final Best Model

The following tables are training, testing, and the OOT dataset's final performance tables:

Training	# Records	# Goods	# Bads	Fraud Rate
	59010	58374	636	1.09%

Bin Statisitcs						Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total Records	Cumulative Goods	Cumulative Bads	% Goods	FDR	KS	FPR
1	591	224	367	37.9	62.1	591	224	367	0.38	57.7	57.32	0.61
2	591	510	81	86.29	13.71	1182	734	448	1.26	70.44	69.18	1.64
3	591	555	36	93.91	6.09	1773	1289	484	2.21	76.33	73.89	2.66
4	591	577	14	97.63	2.37	2364	1866	498	3.2	78.3	75.1	3.75
5	591	581	10	98.31	1.69	2955	2447	508	4.19	79.87	75.68	4.82
6	591	587	4	99.32	0.68	3546	3034	512	5.2	80.5	75.3	5.93
7	591	586	5	99.15	0.85	4137	3620	517	6.2	81.29	75.09	7
8	591	586	5	99.15	0.85	4728	4206	522	7.21	82.08	74.87	8.06
9	591	583	8	98.65	1.35	5319	4789	530	8.2	83.33	75.13	9.04
10	591	589	2	99.66	0.34	5910	5378	532	9.21	83.65	74.44	10.11
11	591	589	2	99.66	0.34	6501	5967	534	10.22	83.96	73.74	11.17
12	591	584	7	98.82	1.18	7092	6551	541	11.22	85.06	73.84	12.11
13	591	576	15	97.46	2.54	7683	7127	556	12.21	87.42	75.21	12.82
14	591	588	3	99.49	0.51	8274	7715	559	13.22	87.89	74.67	13.8
15	591	589	2	99.66	0.34	8865	8304	561	14.23	88.21	73.98	14.8
16	591	587	4	99.32	0.68	9456	8891	565	15.23	88.84	73.61	15.74
17	591	589	2	99.66	0.34	10047	9480	567	16.24	89.15	72.91	16.72
18	591	586	5	99.15	0.85	10638	10066	572	17.24	89.94	72.7	17.6
19	591	589	2	99.66	0.34	11229	10655	574	18.25	90.25	72	18.56
20	591	589	2	99.66	0.34	11820	11244	576	19.26	90.57	71.31	19.52

Table 10. Performance Table for Training Dataset

Test	# Records	# Goods	# Bads	Fraud Rate			
	25290	25046	244	0.97%			

	Bin Statisitcs					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total Records	Cumulative Goods	Cumulative Bads	% Goods	FDR	KS	FPR
1	253	121	132	47.83	52.17	253	121	132	0.48	54.1	53.62	0.92
2	253	217	36	85.77	14.23	506	338	168	1.35	68.85	67.5	2.01
3	253	235	18	92.89	7.11	759	573	186	2.29	76.86	73.94	3.08
4	253	251	2	99.21	0.79	1012	824	188	3.29	77.05	73.76	4.38
5	253	243	10	96.05	3.95	1265	1067	198	4.26	81.15	76.89	5.39
6	253	244	9	96.44	3.56	1518	1311	207	5.23	84.84	79.61	6.33
7	253	252	1	99.6	0.4	1771	1563	208	6.24	85.25	79.01	7.51
8	253	250	3	98.81	1.19	2024	1813	211	7.24	86.48	79.24	8.59
9	253	251	2	99.21	0.79	2277	2064	213	8.24	87.3	79.06	9.69
10	253	253	0	100	0	2530	2317	213	9.25	87.3	78.05	10.88
11	253	252	1	99.6	0.4	2783	2569	214	10.26	87.7	77.44	12
12	253	250	3	98.81	1.19	3036	2819	217	11.26	88.93	77.67	12.99
13	253	252	1	99.6	0.4	3289	3071	218	12.26	89.34	77.08	14.09
14	253	250	3	98.81	1.19	3542	3321	221	13.26	90.57	77.31	15.03
15	253	252	1	99.6	0.4	3795	3573	222	14.27	90.98	76.71	16.09
16	253	252	1	99.6	0.4	4048	3825	223	15.27	91.39	76.12	17.15
17	253	252	1	99.6	0.4	4301	4077	224	16.28	91.8	75.52	18.2
18	253	253	0	100	0	4554	4330	224	17.29	91.8	74.51	19.33
19	253	251	2	99.21	0.79	4807	4581	226	18.29	92.62	74.33	20.27
20	253	252	1	99.6	0.4	5060	4833	227	19.3	93.03	73.73	21.29

Table 11. Performance Table for Test Dataset

OOT	# Records	# Goods	# Bads	Fraud Rate			
	12097	11918	179	1.50%			

	Bin Statisitcs					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total Records	Cumulative Goods	Cumulative Bads	% Goods	FDR	KS	FPR
1	121	47	74	38.84	61.16	121	47	74	0.39	41.34	40.95	0.64
2	121	101	20	83.47	16.53	242	148	94	1.24	52.51	51.27	1.57
3	121	114	7	94.21	5.79	363	262	101	2.2	56.48	54.22	2.59
4	121	120	1	99.17	0.83	484	382	102	3.21	56.98	53.77	3.75
5	121	116	5	95.87	4.13	605	498	107	4.18	59.78	55.6	4.65
6	121	118	3	97.52	2.48	726	616	110	5.17	61.45	56.28	5.6
7	121	120	1	99.17	0.83	847	736	111	6.18	62.01	55.83	6.63
8	121	121	0	100	0	968	857	111	7.19	62.01	54.82	7.72
9	121	120	1	99.17	0.83	1089	977	112	8.2	62.57	54.37	8.72
10	121	121	0	100	0	1210	1098	112	9.21	62.57	53.36	9.8
11	121	120	1	99.17	0.83	1331	1218	113	10.22	63.13	52.91	10.78
12	121	117	4	96.69	3.31	1452	1335	117	11.2	65.36	54.16	11.41
13	121	117	4	96.69	3.31	1573	1452	121	12.18	67.6	55.42	12
14	121	119	2	98.35	1.65	1694	1571	123	13.18	68.72	55.54	12.77
15	121	119	2	98.35	1.65	1815	1690	125	14.18	69.83	55.65	13.52
16	121	120	1	99.17	0.83	1936	1810	126	15.19	70.39	55.2	14.37
17	121	119	2	98.35	1.65	2057	1929	128	16.19	71.51	55.32	15.07
18	121	121	0	100	0	2178	2050	128	17.2	71.51	54.31	16.02
19	121	117	4	96.69	3.31	2299	2167	132	18.18	73.74	55.56	16.42
20	121	120	1	99.17	0.83	2420	2287	133	19.19	74.3	55.11	17.2

Table 12. Performance Table for OOT Dataset

With our best performing LightGBM model, the three tables from above show FDR and other statistics for each population bin. Records are ranked in descent order based on their probability of being a fraud. From tables 8 to 10, we can see that more than 50% of frauds are caught within the very first percentile bin. Fraud detection rates at 3%, or the cumulative percentage of bad, are 76.33%, 73.86%, and 56.48% for training, testing, and OOT respectively.

6.2 Fraud Score Activity Explanation

Fraud scores generally rise as the model sees more activity at the entity level. To understand the dynamics of how the fraud score changes, here are two examples that show the transactions for a specific card number and merchant number. In our card number example, we had 46 transactions over about two months. 40 of the 46 transactions occurred after 12-05, which is when we see the fraud score rise. After about a month of inactivity in November, the card was used much more frequently in December and we can see how the fraud score rises accordingly with the number of transactions. After the break in usage, the spike and consistency in usage caused our model to suspect these transactions as fraudulent, and we can see that this trend continued to the end of our data.

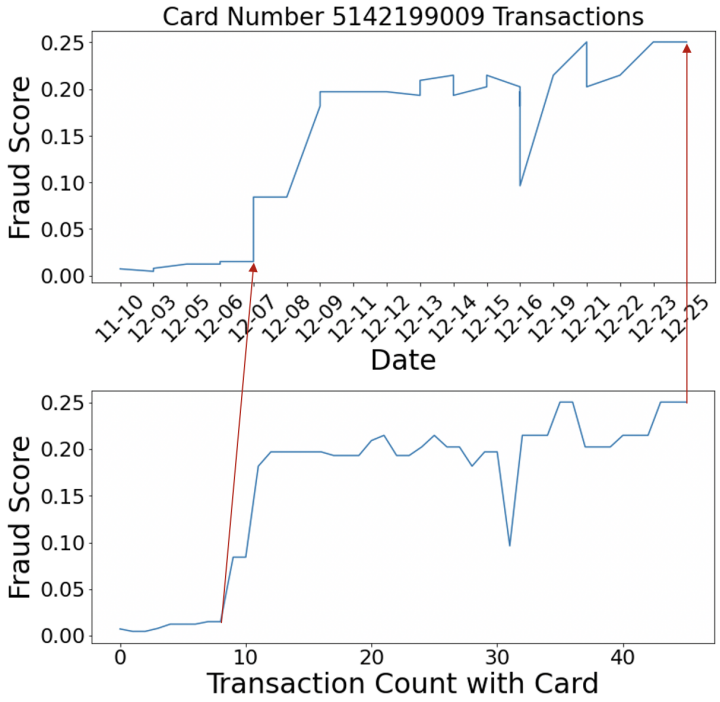


Figure 14. Cardnum Change in Fraud Score Based on Date and Number of Transactions

With our merchant example, we had 87 transactions over the course of about two months. Looking at the transactions over time, we can see that there are multiple small bunches of transactions that cause a spike in the fraud score. The increase from 12-03 to 12-05 contains 12 transactions, which is a relatively significant rise in transactions compared to this merchant's historical performance. Although these increases in transactions were small, we can see that every spike in transactions led to a spike in the fraud score.

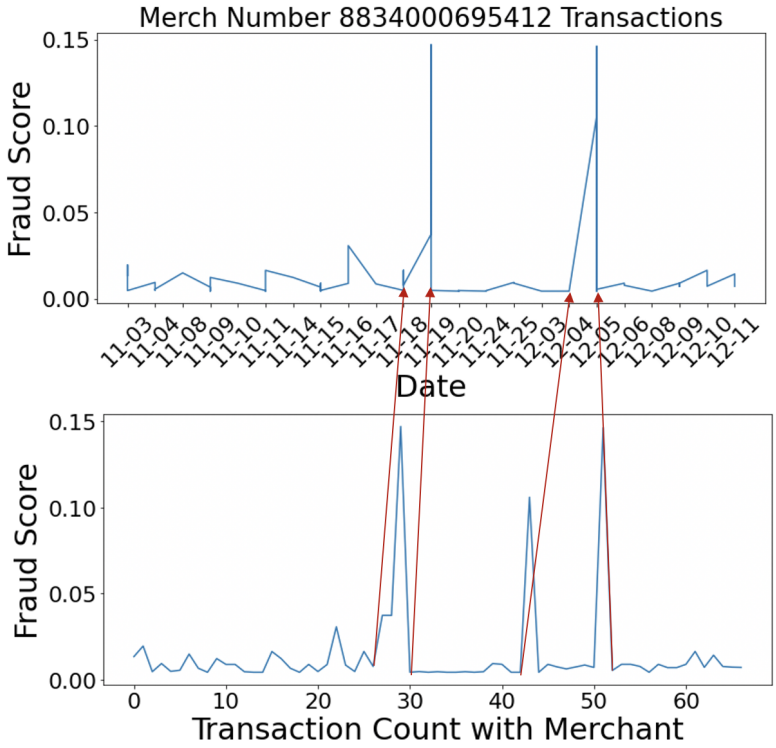


Figure 15. Merch Number Change in Fraud Score Based on Date and Number of Transactions

6.3 Fraud Saving Cutoff

Throughout this report, we have fixed the fraud detection rate at 3% and found that the LightGBM model performed the best. Even though this is our best model, it does not catch all the frauds and has no false positives. Therefore, there will be some gain and pain by choosing this model at a cutoff line of 3%. Now we will see what will happen to the overall cost and

savings as we change the fraud detection rate. The figure below shows the cost and savings the company will come across as the fraud detection rate goes from 0% to 50%.

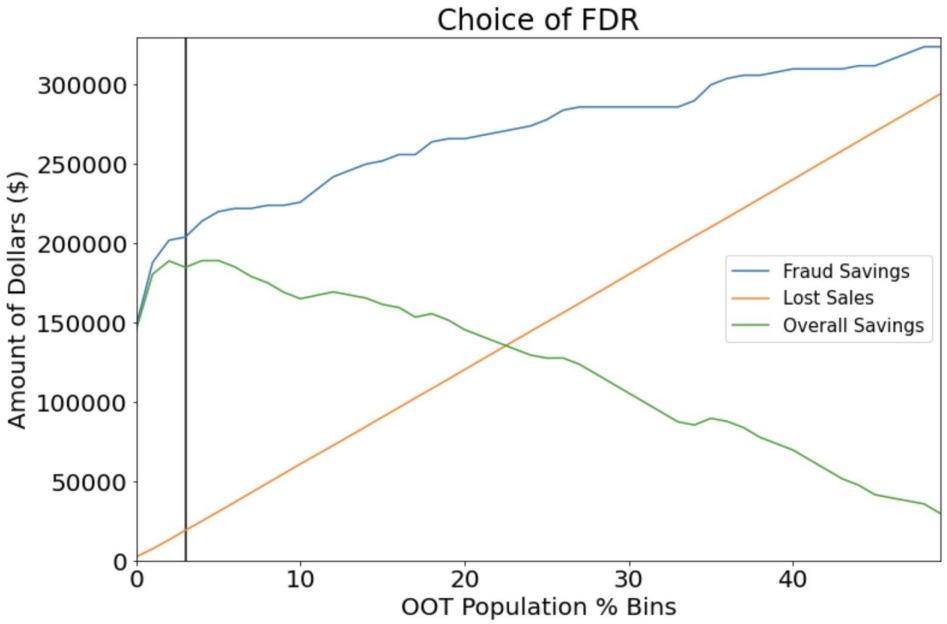


Figure 16. Change in Cost/Savings by Fraud Detection Rate

The blue curve represents the gain for every fraud this model catches and assumes that it will be $2,000 per fraud. The orange line represents the loss for every false positive assuming the cost will be $50. Finally, the green curve represents the overall savings, which is subtracting the orange line from the blue curve.

We can see that as the percentage increases, the green curve decreases in the amount of dollars. Around 3%, the green curve hits its peak where we want to set our cutoff. Here, the total fraud savings is $202,000 and at a cost of $13,100, which results in an overall savings of $188,900 annually. By using our LightGBM model, we will be able to save the company around $190K per year.

7. Conclusion

Fraud transactions are creating increasing losses to merchants, banks, and private cardholders globally. By 2030, the total volume of global fraud transactions is going to be $49.3 billion. What's more, the United States is enduring the greatest loss from these fraud credit transactions. To decrease the amount of fraud transactions, we built a fraud detection model based on a credit card purchase record dataset from a US government organization in 2006. We first explored the dataset by calculating the summary statistics and plotting the distribution of the 10 fields. A Data Quality Report was written. Then we cleaned the data by substituting the frivolous values. After the data was cleaned, we created several new fields by combining different identity fields. Also, we used target encoding to represent categorical fields such as day of the week and amount type Then, we measured the unusualness of a card or a merchant by applying Benford's Law and generated 1012 candidate variables in total. Since the dimensionality of the data was too high to fit machine learning models, we used KS as a filter to reduce the number of variables to 80, and then we ran a wrapper to select the top 20 best variables for modeling.

Before training the models, we first removed the last two months of the records as the OOT data. Also, we left out the first two weeks since they are not effective for modeling. For models, we chose Logistic Regression, Decision Tree, K-Nearest Neighbors, LightGBM, and Neural Network. The Logistic Regression reached the best performance of 27.99% FDR at 3% when we included 10 variables, using the 'lbfgs' solver, applying L2 regularization, and 0.6 for the inverse of regularization strength. Then we built the Decision Tree model, KNN, LightGBM, and Neural Network model subsequently, applying various hyperparameters. Among these, the best performance was from the LightGBM model, which reached 56.48% FDR at 3% when we used 5 variables, 100 estimators, the learning rate at 0.01, the max depth of trees to be 4, and the subsample as 0.8.

By assuming the gain for every fraud detected is $2,000 and the loss for every false detection is $30, we drew a fraud saving, a lost sales and an overall saving curve for our model's OOT result with different choices of fraud detection rate. To reach the peak of overall savings which is about $190K, our group chose to use 3% as our fraud detection rate. The model can eliminate 56.48% of total fraud transactions by declining 3% of transactions. In conclusion, with our choice of LightGBM model, we can save around $190K for the company's loss on fraud transactions.

There are also aspects of our models that can be improved. By exploring different hyperparameters on the other algorithms, we might be able to find close results to the best model.

Appendix

A.1 File Description

The data used in this project is actual credit card purchases from a US government organization. The dataset covers the time from January 1st, 2006, to December 31st, 2006. There are 10 fields and 96,753 records.

A.2 Numerical Fields Summary Statistics

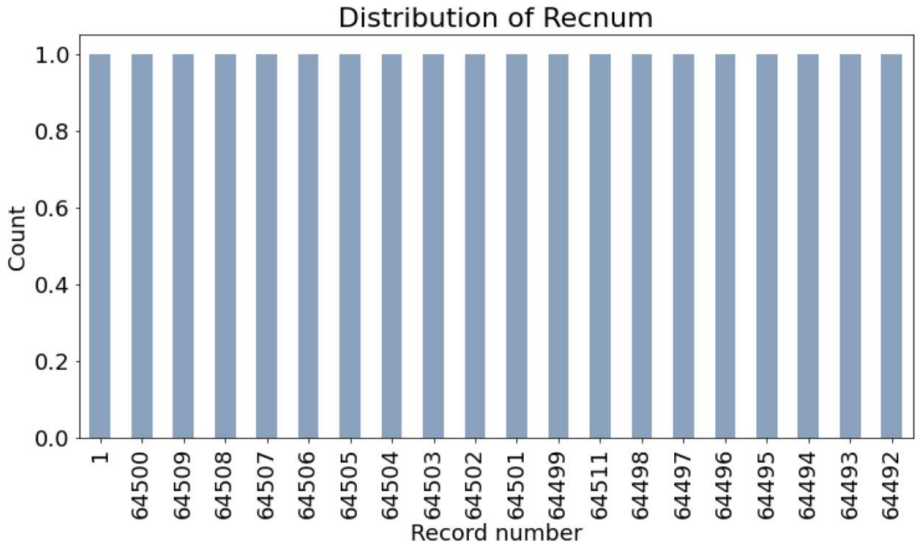
Field Name	% Populated	Min	Max	Mode	Mean	Stdev	% Zero
date	100%	2006-01-01	2006-12-31	2006-02-28	N/A	N/A	0
Amount	100%	0.01	3102045.53	3.62	427.89	10006.14	0

A.3 Categorical Fields Summary Statistics

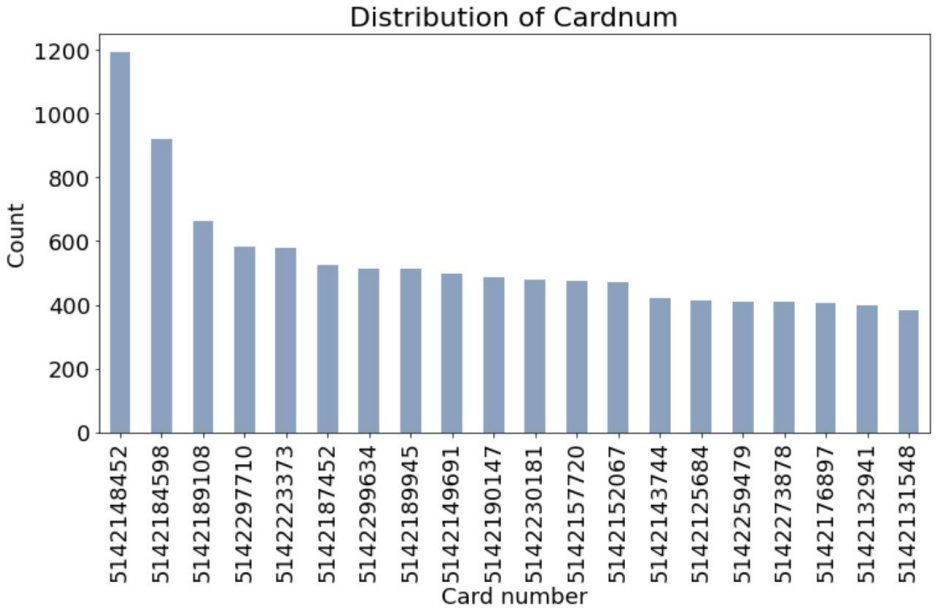
Field Name	% Populated	# Unique Values	Mode
Recnum	100%	96,753	NA
Cardnum	100%	1,654	5142148452
Merchnum	96.51%	13,091	930090121224
Merch description	100%	13,126	GSA-FSS-ADV
Merch state	98.76%	227	TN
Merch zip	95.19%	4,567	38118.0
Transtype	100%	4	P
Fraud	100%	2	0

A.4 Distributions of Fields

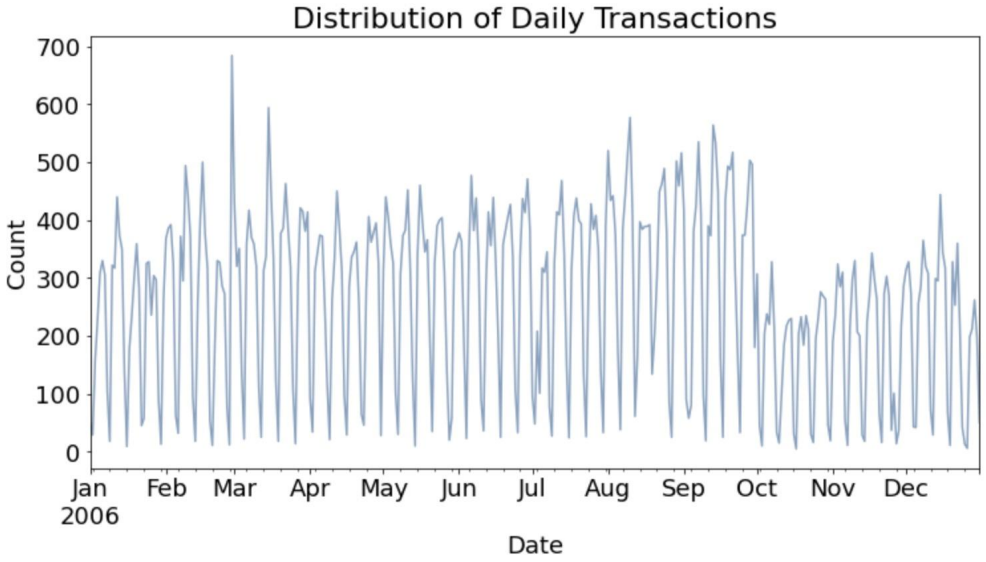
Field 1: Recnum
Description: The unique index of the dataset. Each credit card transaction has a unique record number.

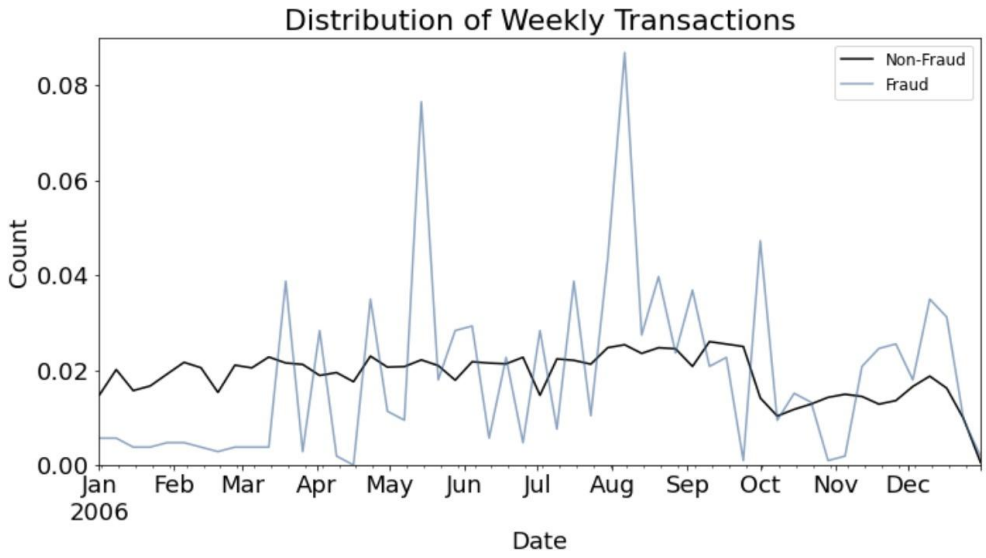


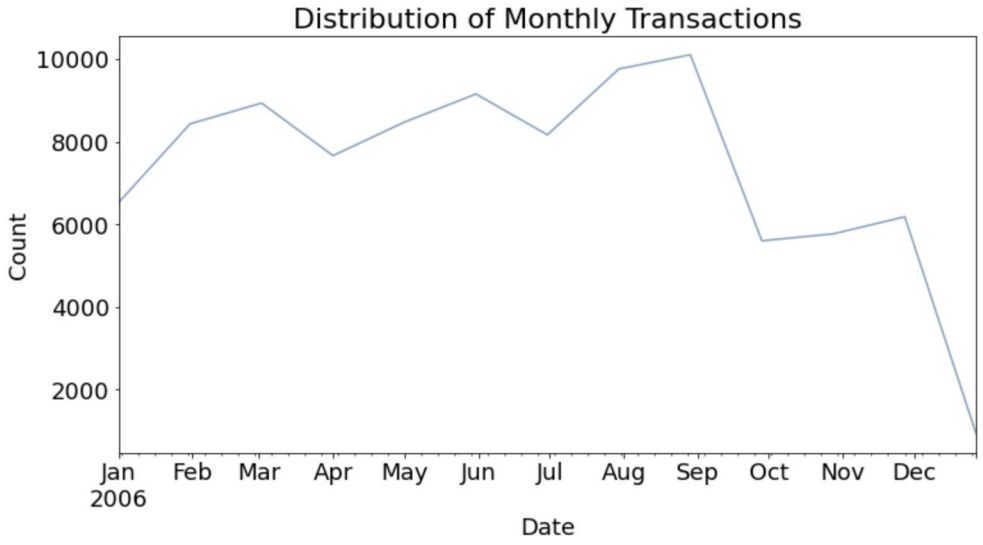
Field 2: Cardnum
Description: The card number of the transaction



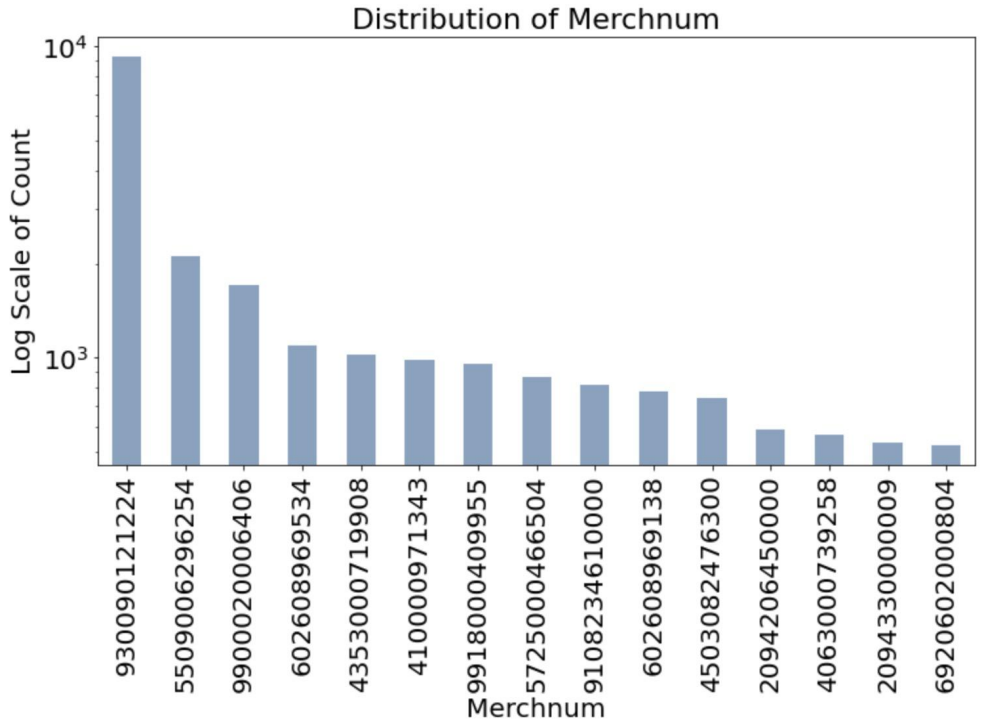
Field 3: Date
Description: The date when the transaction was placed



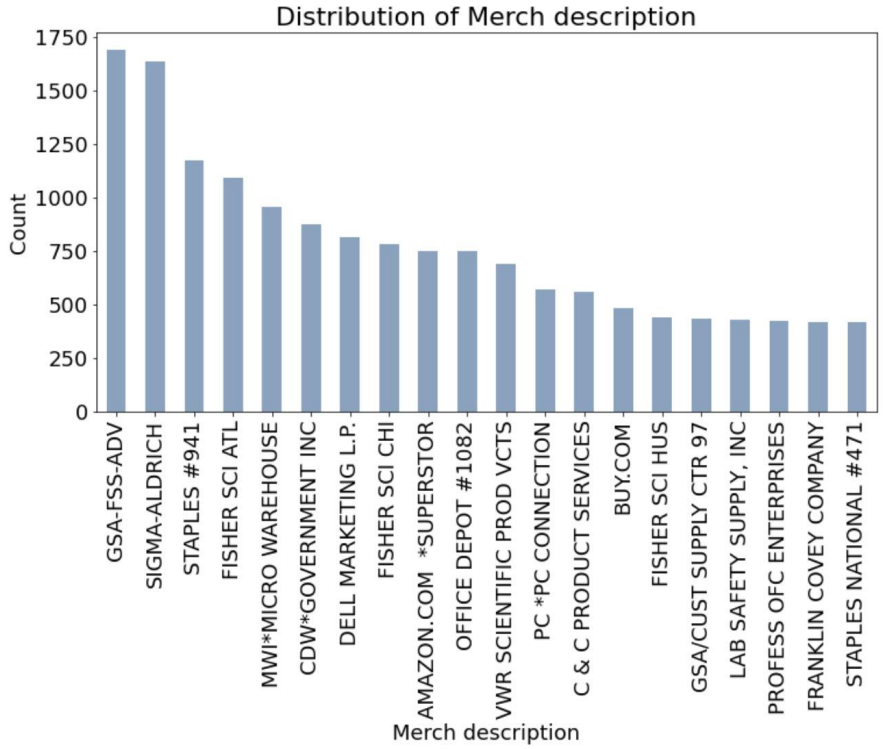




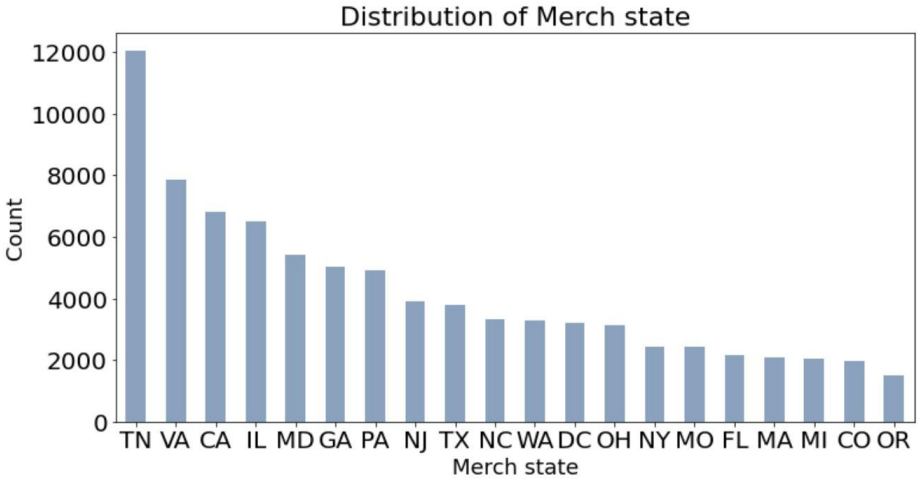
Field 4: Merchnum
Description: The Merchant number of the transaction



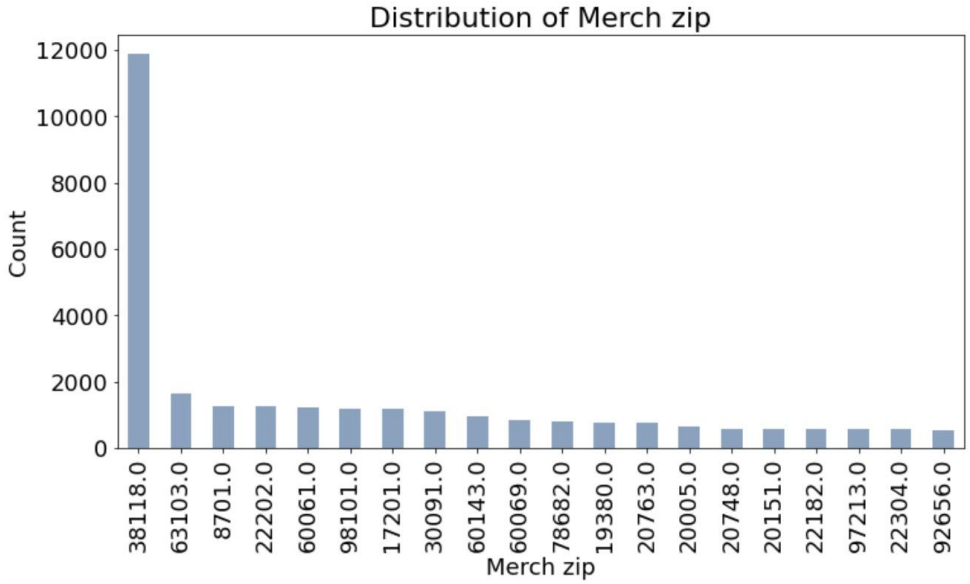
Field 5: Merch description
Description: The merchant description of the transaction



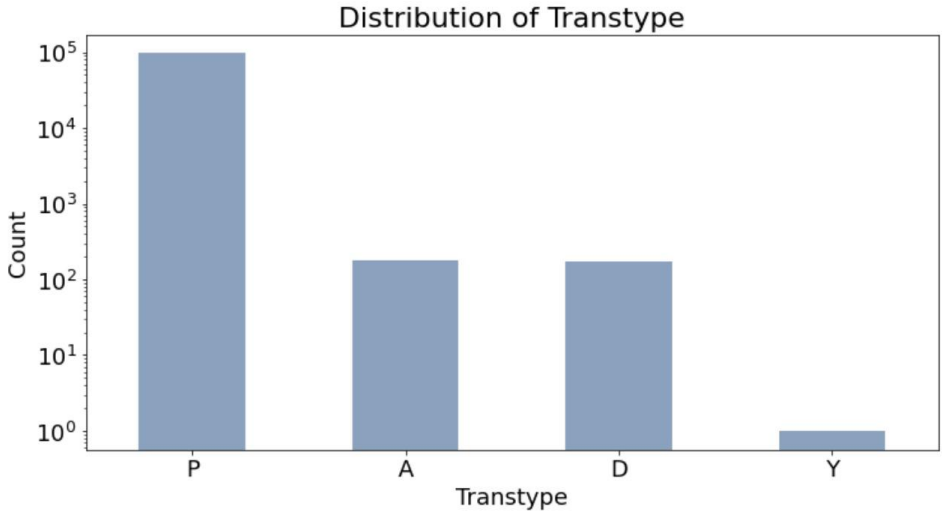
Field 6: Merch state
Description: The merchant state of the transaction



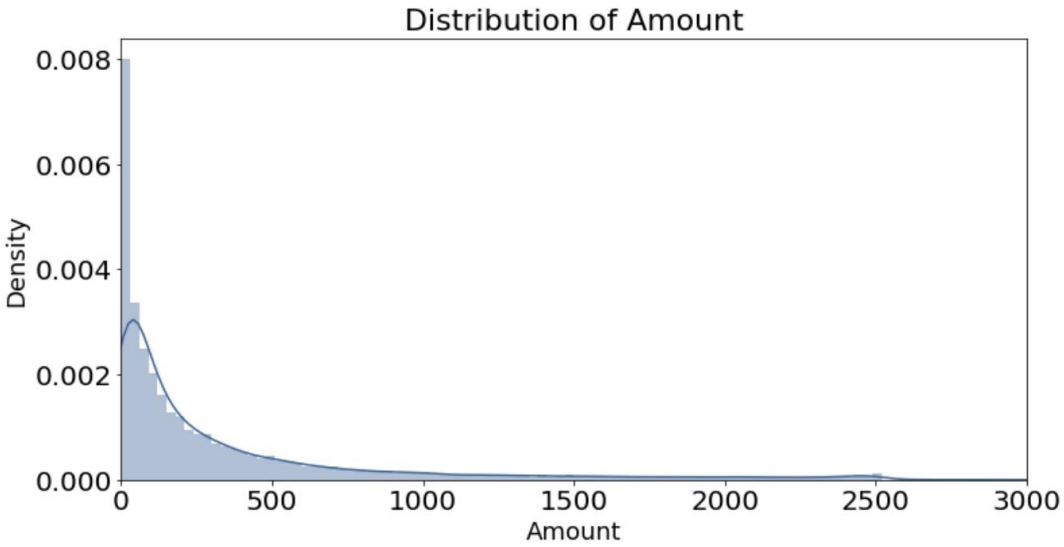
Field 7: Merch zip
Description: The merchant zipcode of the transaction



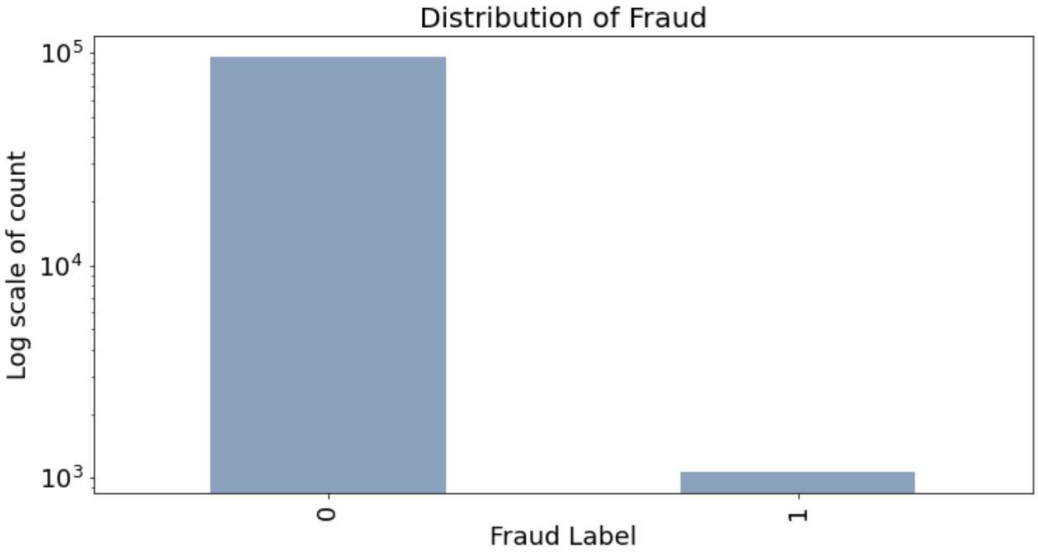
Field 8: Transtype
Description: The type of the transaction



Field 9: Amount
Description: The amount of the transaction



Field 10: Fraud
Description: Whether or not the model detected the credit card transaction as fraudulent. Values are 1 for fraudulent, 0 for not.



A.5 Created Variables

New Variables for DSO562 Project 3 (Credit Cards Transactions Data)			
Description of Variables	**# Variables Created**	**Approximate CPU Time (Minute)**	**Names of New Columns**
Day of Week of Each Application (including Monday to Sunday, 7 unique values)	1	<0.1 min	dow
Target Encoding for dow column (transferring categorical variables to numerical variables)	1	<0.1min	Dow_risk
The categories of amount spent each transaction (6 categories, according to the daily experience)	1	<0.1m	Amount_type
Target Encoding for amount type (transferring categorical variables to numerical variables)	1	<0.1m	atype_risk
Combination of card number and merchant number, merchant state, merchant zipcode and amount type of each transaction	4	<0.1m	Cardnum_Merchnum Cardnum_Merstate Cardnum_Merzip Cardnum_Atype
Combination of amount type and merchant number, merchant state and merchant zipcode for each transaction	3	<0.1m	Merchant_Atype State_Atype Zip_Atype
Number of days since transaction with that entity has been seen. Entities are {Card number, merchant	9	About 20 s	{Cardnum, Merchnum, Cardnum_Merchnum Cardnum_Merstate Cardnum_Merzip

number, Cardnum_Merchnum, Cardnum_Merstate, Cardnum_Merzip, Cardnum_Atype, Merchant_Atype, State_Atype, Zip_Atype}			Cardnum_Atype, Merchant_Atype State_Atype Zip_Atype }_day_since
Number of transactions at that entity over the past n days. Entities are {Card number, merchant number, Cardnum_Merchnum, Cardnum_Merstate, Cardnum_Merzip, Cardnum_Atype, Merchant_Atype, State_Atype, Zip_Atype}, n is {0,1,3,7,14,30}	54	About 1 min	{Cardnum, Merchnum, Cardnum_Merchnum Cardnum_Merstate Cardnum_Merzip Cardnum_Atype, Merchant_Atype State_Atype Zip_Atype}_count_{0,1,3,7,14,30 }
{Average, max, median, total, actual/average, actual/max, actual/median, actual/total} transaction amount at that entity over the past n days. Entities are {Card number, merchant number, Cardnum_Merchnum, Cardnum_Merstate, Cardnum_Merzip, Cardnum_Atype, Merchant_Atype, State_Atype, Zip_Atype}, n is {0,1,3,7,14,30}	432	About 3 min	{Cardnum, Merchnum, Cardnum_Merchnum Cardnum_Merstate Cardnum_Merzip Cardnum_Atype, Merchant_Atype State_Atype Zip_Atype}_{avg, max, med, total, actual/avg, actual/max, axtual/med, actual/toal}_{0,1,3,7,14,30}
Number of transactions at that entity seen in a short time window {today or past day} compared to number of transactions at that entity over a longer time window {past 3,	72	About 34 s	{Cardnum, Merchnum, Cardnum_Merchnum Cardnum_Merstate Cardnum_Merzip Cardnum_Atype, Merchant_Atype State_Atype

7,14,30 days}. Entities are {Card number, merchant number, Cardnum_Merchnum, Cardnum_Merstate, Cardnum_Merzip, Cardnum_Atype, Merchant_Atype, State_Atype, Zip_Atype}			Zip_Atype}_count_{0,1}_by_{3,7,14,30}
Number of unique identy in the specific field during the past {1,3,7,14,30,60} days. Both entity and field are {Card number, merchant number, Cardnum_Merchnum, Cardnum_Merstate, Cardnum_Merzip, Cardnum_Atype, Merchant_Atype, State_Atype, Zip_Atype}, but with different values.	432	About 32 mins	{ Cardnum, Merchnum, Cardnum_Merchnum Cardnum_Merstate Cardnum_Merzip Cardnum_Atype, Merchant_Atype State_Atype Zip_Atype}_for_{1,3,7,14,30,60}
Benford Law's variables for each card number and each merchant number except for Fedex transactions	2	About 40 s	Cardnum_b, Merchnum_b
Total	1012	About 40 mins	