MGTA 463:

Identifying Fraud in Credit Card Transactions

Anh Vo, Ines Ye, Mouli Palacharla, Paige Dubelko,

Perla Anaya, Srikar Gunisetty, Xuefei Tan

10th June, 2022

Table of Contents

Executive Summary

Credit card fraud is a common type of identity theft which involves a fraudster stealing an individual's credit card or credit card information to make fraudulent purchases. Financial institutions must be able to accurately identify and stop fraudulent transactions in order to instill trust in their customer base and mitigate costs associated with these fraudulent transactions. Especially, the latest evolving fraud dynamics of counterfeit applications urge the banks to put a premium on credit card fraud detection. On average, correctly identifying fraudulent transactions can save a bank around $2000 per fraudulent transaction therefore leading to large savings and increased customer satisfaction.

In this report, we have analyzed credit card transaction data filed by the US government to build several supervised models to successfully detect fraudulent transactions. The report covers our data preparation process and machine learning model tuning methods to allow insight for stakeholders. Our main steps are as follows:

- Data Preparation: Investigate data, analyze important field distributions, and fill in missing records.
- Feature Engineering: Transform current data fields into new candidate variables that can improve model performance.
- Feature Selection: Perform feature selection to find a subset of variables that are the strongest predictors of fraud.
- Modeling: Train and test multiple ML models by using cross-validation and parameter tuning with GridSearchCV to identify the optimal model.
- Analyzing Results: Determine ML model with best performance and make a recommendation for fraud detection threshold.

Our team built over 2,000 variables and narrowed it down to 20 strong indicator variables to feed into our models. We built out 5 different models, including a logistic regression, neural network, single decision tree, random forest, and boosted tree. Our analysis found the random forest model to have the best performance, with the ability to detect out of time (OOT) fraudulent transactions 59% of the time.

Given this information, we are confident in suggesting a fraud detection cut off rate of 5% which in this dataset would lead to savings of more than $1.56 million/year.

Description of Data

The dataset, originally filed by the US government, includes transaction information of credit cards in 2006. It contains 96,753 records and 10 fields which cover the details of each transaction, i.e. time, location, merchant's information and amount. The dataset is composed of 2 numeric fields which are Amount and Date. The 8 remaining categorical fields give identifying and detailed information on the records and including a fraud indicator field.

Summary Statistics Tables

Numeric Fields

Field Name	% Populated	Min	Max	Mean	Std Dev	%Zero
Date	100	2006-01-01	2006-12-31	NA	NA	0
Amount	100	0.01	3,102,045.53	427.89	10,006.14	0

Categorical Variables

Field Name	% Populated	# Unique Values	Most Common Value
Recnum	100	96,753	Unique for all records
Cardnum	100	1,645	5142148452
Merchnum	96.51	13092	930090121224
Merch description	100	13126	GSA-FSS-ADV
Merch state	98.76	228	TN
Merch zip	95.19	4568	38118
Transtype	100	4	P
Fraud	100	2	0

Important Distribution of Fields

Below we have identified fields and distributions of interest that may be of importance when detecting fraud. To view full information on all fields, a data quality report (DQR) can be found in the Appendix A.

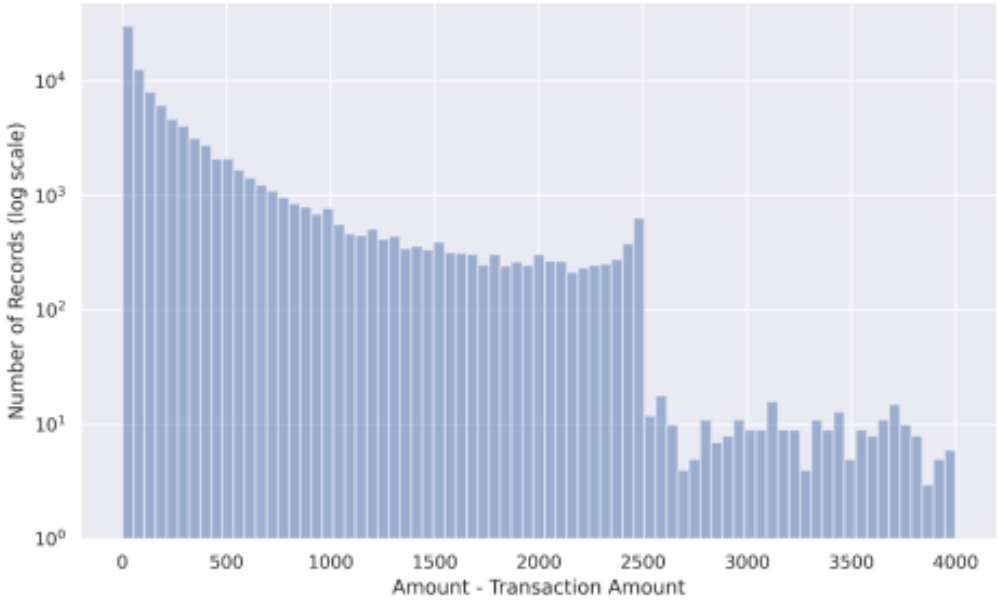
Date

The graph below shows weekly transactions throughout the year 2006. We have noticed a sharp drop off in transactions starting in October, which then begins to continuously grow for the rest of the year. This may indicate that transactions are being recorded based on activity related to government fiscal years which starts in October of the previous year and ends in September of its current year. Thus, the 2006 government fiscal year started in October 2005 and ended at the end

of September 2006. The 2007 government fiscal year started in October 2006 and ended in September 2007. The dataset therefore contains records for both the 2006 and 2007 government fiscal years.

Amount

The distribution plot below shows dollar amounts for credit card transactions in the dataset up to a value of $4,000 with the notable drop off depicted at/near the single purchase limit of $2,500. It is important to note that the bulk of the purchases are below approximately $2,500 due to government purchase cards having a single purchase limit of $2,500 in 2010.

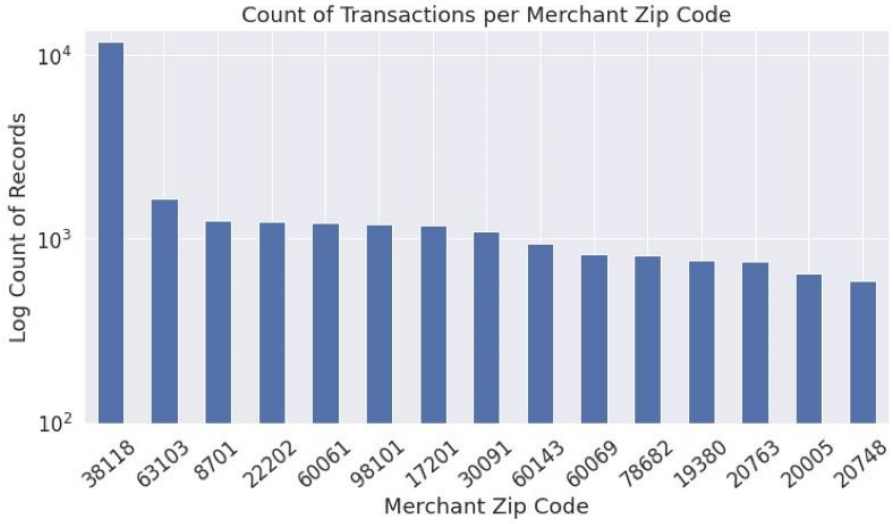


Fraud

Fraud is a binary label of 0 and 1 indicating whether each of the records is fraud or not. A 0 means the transactions are benign and a 1 refers to a fraudulent transaction. From the histogram, we can observe the significant amount difference between those two labels. Among the 96,753 records, there are 95,694 classified as normal while the other 1,059 observations are categorized as fraud. The percentage of frauds equates to 1.09%.

Zip Code

The graph below shows zip codes with the highest numbers of transactions. We notice that the highest number of merchant transactions per zip code is 38118 which belongs to the state of Tennessee. Tennessee also has the highest transactions per state.



Data Cleaning

Data cleaning is the process of identifying individual fields within a dataset with missing, incomplete, incorrect, or irrelevant data and making the decision of whether to remove or update the information.

After gaining an better understanding of the data and fields, we applied an initial filtering to remove the following rows from our dataset:

- Rows with transaction type (*transtype*) other than 'P'
- Single large transaction outlier

The majority of transactions in our dataset were of *transtype* 'P', other transaction types were removed so we could focus on more typical transactions. A single large transaction outlier was also removed as it was not representative of the fraudulent transactions we are investigating and could skew our results.

We then dug deeper to identify if any fields contained missing values that needed to be addressed. We determined *merchnum, merch state,* and *merch zip* to have a number of missing values that needed to be filled in. The following steps were applied to fill in missing values:

Merchnum:

- Total missing values: 3374
- Fill in missing values by mapping with known *merch description*.
- All remaining missing values are assigned as 'unknown'.

Merch zip:

- Total missing values: 4656
- Fill in the missing *zip code* using the *state* from that *zip code*, if known.
- Fill in by mapping with known *merchnum* and *merch description*.
- All remaining missing values are assigned as 'unknown'.

Merch state:

- Total missing values: 1195
- If the record has a *zip code*, assign the *state* based on that *zip code.*
- Fill in by mapping with known *zip code*, *merchnum*, and *merch description*.
- All remaining missing values are assigned as 'unknown'.

Feature Engineering

Feature engineering is a critical component in predictive analytics and often where the majority of time is spent in data preparation before modeling. Simply put, feature engineering is the process of transforming data into input for our machine learning model in an effort to improve model performance.

Considering a typical two-dimensional dataset, each *feature* is a column. There are two main types of features - *raw features* and *derived features. Raw features* are features original to the dataset while *derived features* are created during feature engineering and are extracted from existing attributes. Our goal is to engineer candidate variables that will allow us to better represent the underlying problem to our predictive model.

Understanding Credit Card Fraud Indicators

In order to create variables that will improve our models performance, we must first understand the ways in which accounts are compromised and signals that could indicate fraud. Common ways accounts are compromised include:

- Lost or Stolen Cards
- A Common Point of Compromise - Someone at a merchant stealing cards, making counterfeit cards, card skimmers at common places.
- Online Account Hack - Username and password gets compromised and card information is stolen.

Once an account has been compromised, typical signals associated with fraud are:

- Bursts of activity at different merchants
- Larger than normal purchase amounts, at the same or different merchants
- Purchases at merchants not used before for the card
- Purchases in a very different geographic location
- Purchases at high-risk merchants
- Increased usage when card is not present
- Charges at a fictitious merchant

Variable Summary

Given our knowledge of credit card fraud, we created 2,485 variables that we believe could be useful in predicting fraud. Not all variables will be strong predictors and used in the final model, but this will be determined during feature selection. Below is the logic used to create each type of variable.

Description of Variable	# of Variables Created	Additional Notes
Original data fields - excluding *Recnum* and *Fraud*	8	*Recnum* and *Fraud* were not included in the original variables as *Recnum* was a unique identifier and *Fraud* was our dependent variable
Day of the Week & DOW Risk	2	Day of the week was target encoded to replace our numeric measure of date with a categorical variable. DOW Risk calculates fraud proportion associated with a day of the week.
New entities from original field combinations - see below for entities crea	25	See below for information on entity creation.
Days Since: Number of days since the entity has been seen	25	Days since can be used to gauge average usage for credit cards.
Frequency: Number of transactions with the same entity over the last {0,1,3,7,14,30,60} days	175	A high frequency could be an indicator of fraud as it would indicate frequent purchases of the same entity in a rapid succession.
Aggregated Values: Average, max, median, sum, actual / average, actual / max, actual / median, actual / total for transaction amount per entity over {0,1,3,7,14,30,60} days	1400	Aggregated values can be useful in understanding data distribution across entities.
Velocity Change: Number of transactions for an entity seen recently, today or yesterday, divided by number of transactions for an entity seen in the past {7,14,30,60} days	200	This ratio captures recent spending in comparison to historical spending. This ratio could capture a sudden change in spending habits. A high ratio can be an indicator of fraud.
Velocity Days Since Ratio: Velocity change for {7,14,30,60} days / days since entity	200	This ratio captures the change velocity over days since this entity was last seen. A high ratio can be an indicator of fraud.
Variability: Aggregated variability of transaction amounts over {0,1,3,7,14,30} days	450	This entity captures the variability of transaction amounts across entities and aggregates by average, max, and median.

Entities Created: *card_merchnum, card_zip, card_state, merchnum_state, state_des, state_zip, zip3, card_zip3, merchnum_zip, merchnum_zip3, card_merchdesc, card_dow, merchnum_desc, merchnum_dow, merchdesc_state, merchdesc_zip, merchdesc_dow, card_merchnum_desc,*

card_merchnum_state, card_merchnum_zip, card_merchdesc_state, card_merchdesc_zip, merchnum_merchdesc_state, merchnum_merchdesc_zip

Zip3 = first 3 digits of zip code
_ indicates concatenation between entities. Example card_merch = card + merchnum

Feature Selection

Feature selection is used to reduce dimensionality without losing much information. High dimensionality can cause the data to sparse quickly and more points become outliers. In addition, models with high dimensionality tend to have a higher risk of overfitting. The purpose of the feature selection process is to only include variables with substantial information, and minimize the number of variables for the model. Ways to categorize feature selection include Filter, Wrapper and Embedded methods. These methods can be used for either linear or nonlinear, and are for supervised models only. In this project, the team implemented Filter and Wrapper for feature selections.

Filter

Filtering is performed to identify the importance of each independent variable by itself to predict y (the dependent variable). Pearson correlations, mutual information, univariate Kolmogorov-Smirnov (KS) scoring, and fraud detection rate (FDR) are common filtering approaches for binary classification issues. For this project, univariate KS was calculated to rank the 2,485 variables.

The KS score is the maximum difference of the cumulative distributions. It is calculated through the formula below.

$$KS = \max_{x} \int_{x_{min}}^{x} [P_{\text{good}} - P_{\text{bad}}]\, dx$$

For each candidate variable, the KS method plots the good (non-fraud) and bad (fraud) transaction scores separately. The greater the difference between the two curves, the better the variable is for separating, and thus the more important the variable is. An example distribution can be seen from figure 5.

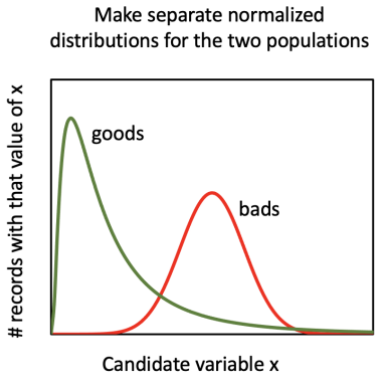


Figure 5

After the KS scores are calculated for all variables, the filter picks the variables with KS higher scores and drops the ones with lower scores. In this project, we filtered the top 80 variables with the highest KS scores for the following steps.

Wrapper

The goal of a wrapper is to find the best subsets of predictors and remove correlations between these variables. Common wrapper methods include backward selection, forward selection, and general stepwise selection. For this project, the team implemented the forward selection method. This method keeps adding features to the wrapper until the best result is reached. The wrapper was run on the remaining 80 variables determined after filtering.

We used fraud detection rate (FDR) to measure wrapper results. It represents the percentage of bad transactions (fraud) that have been caught at a particular examination cutoff point. For instance, FDR 50% at 3% means the model was able to catch 50% of fraud activities among 3% of the total population.

Our wrapper results are shown in the figure below. It can be seen there was no significant improvement after the wrapper reached 20 variables. Because of the drop off in improvement, after several wrapper runs, our team decided to pick 20 variables for modeling. Final features with their filter KS scores are listed in the table below.

Variable Name	Average KS Score
card_zip3_total_7	0.696
merchnum_max_7	0.608
card_zip_total_14	0.672
card_zip_total_60	0.646
merch_zip_max_7	0.603
card_merchnum_desc_total_60	0.646
zip3_total_0	0.615
card_merch_total_30	0.659
card_zip_total_30	0.656
card_merch_total_60	0.643

merchnum_desc_total_7	0.666
merchnum_desc_avg_7	0.663
amount_cat	0.545
merchnum_desc_total_14	0.663
card_merchnum_desc_total_30	0.657
card_merchdesc_total_60	0.645
merchnum_desc_max_7	0.651
merch_zip_total_0	0.609
zip3_actual/avg_60	0.555
card_merchdesc_total_7	0.671

Model Algorithms

In this section, we will introduce various model algorithms and the results that the models yield. We used the 20 variables selected in the previous steps to train distinct models. For each model, a few hyperparameters are tuned in order to find the optimal one which can best predict the fraud label. We also added weight to some of the models to better deal with the unbalanced data sets.

We used the last two months' data as out of time (OOT) to verify the predictability of the model facing the unknown data, and applied a train-test ratio of 70:30 for the rest of the data. The model performance part contains 10 FDR results of each model with different combinations of hyperparameters. For each result, we ran 10 iterations with randomly generated training and testing splits on the dataset and then took an average. Models that we tested include logistic regression, single decision tree, neural network, random forest and boosted tree.

Logistic Regression

Logistic regression is a statistical model that estimates the probability of an event occurring. In this case, we use binary logistic regression considering that the outcome y has only two possible values, e.g. 0 ("good - normal transaction") or 1 ("bad – fraud transaction"). The model output is a continuous number between 0 and 1, representing the probability that the record's label should be 1. The logistic regression process is represented in Figure 6.

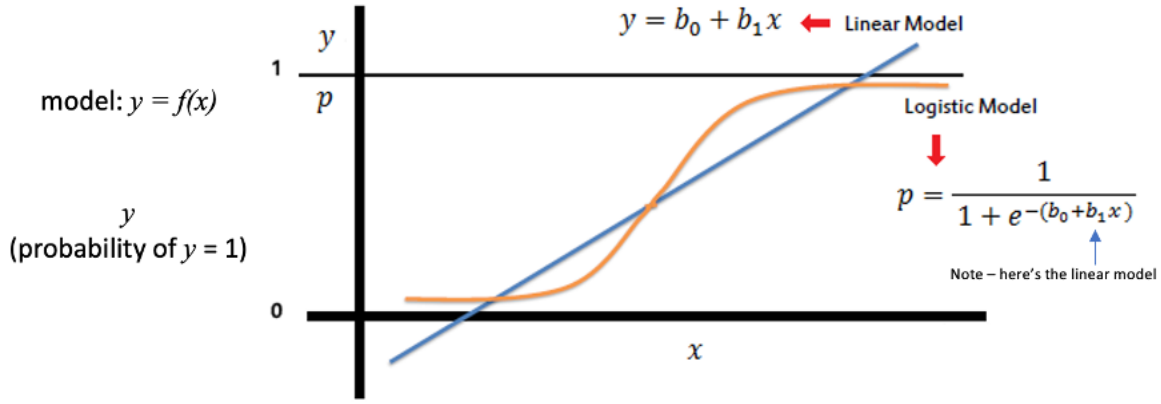


Figure 6: Logistic regression model

A logistic regression does a logistic transformation to a linear regression surface.
It smashes the high region to a maximum of 1 and squashes the lower region to a minimum of zero.

The linear regression model is: $$y = b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n = \sum b_ix_i$$

The logistic regression model is: $$y = \frac{1}{1 + e^{-\sum b_ix_i}}$$

Hyperparameters tuned:

- *penalty*: Norm of the penalty applied.

- *C* : Inverse of regularization strength. Smaller values specify stronger regularization.
- *solver*: Algorithm to use in the optimization problem.
- *l1_ratio*: The Elastic-Net mixing parameter. Used when penalty='elasticnet'.

Model performance:

Model	Iteration #	Variables	Hyperparameters					Average FDR at 3%		
Model Name	Iteration	Total Variables	Penalty	C	Class Weights	Solver	l1_ratio	Train	Test	OOT
Logistic Regression	1	20	l2	1.0	(1,91)	lbfgs	-	0.673	0.655	0.355
	2	20	l2	0.5	(1,91)	lbfgs	-	0.674	0.651	0.353
	3	20	l2	0.001	(1,91)	lbfgs	-	0.649	0.636	0.245
	4	20	none	1.0	(1,91)	lbfgs	-	0.671	0.661	0.370
	5	20	elasticnet	0.1	(1,91)	saga	0.1	0.663	0.671	0.355
	6	20	elasticnet	1.0	(1,91)	saga	0.1	0.662	0.676	0.347
	7	20	l1	1.0	(1,91)	saga	-	0.672	0.669	0.350
	8	20	l1	0.001	(1,91)	saga	-	0.638	0.623	0.435

Single Decision Tree

A decision tree is a tree-structured supervised learning algorithm that classifies data records by setting a series of rules. As shown in the flowchart below, each decision node represents one classification rule. Every node points to one child node for another rule or comes to leaf node(s) which is the outcome. The decision nodes thereby form a hierarchy, encoded as a tree.

A good classification rule can split the records with heterogeneous labels into subsets with nearly homogeneous labels. To measure the effectiveness of the split, Gini and entropy are introduced as the two most popular criteria. The decision tree process is represented in Figure 7.

In the credit card transactions project, all the transaction records are fed into the decision tree through the root node. Decision nodes then conduct evaluation to form homogenous subsets.

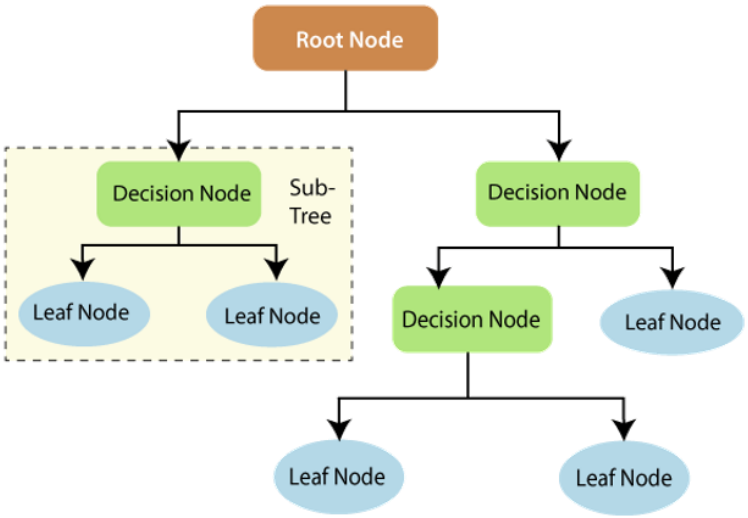


Figure 7: Decision tree model

Hyperparameters tuned:

- *criterion*: The function to measure the quality of a split.

- *splitter*: The strategy used to choose the split at each node.
- *max_depth*: The maximum depth of the tree.
- *min_samples_split*: The minimum number of samples required to split an internal node.
- *min_samples_leaf*: The minimum number of samples required to be at a leaf node.

Model performance:

Model	Iteration #	Variables	Hyperparameters				Average FDR at 3%		
Model Name	Iteration	Total Variables	criterion	Max_depth	Min_samples_split	Min_samples_leaf	Train	Test	OOT
Decision Trees	1	20	gini	None	2	1	1.000	0.561	0.303
	2	20	gini	2	3	2	0.616	0.624	0.302
	3	20	gini	3	50	10	0.642	0.618	0.340
	4	15	gini	8	100	2	0.834	0.727	0.456
	5	20	entropy	5	50	10	0.628	0.625	0.426
	6	20	gini	8	100	60	0.802	0.754	0.394
	7	20	gini	20	1000	10	0.701	0.668	0.405
	8	20	gini	10	100	2	0.834	0.727	0.456

Neural Networks

Neural networks are a subset of machine learning. Named after the biological neural networks, their layered structure mimics the way that human neurons signal to one another.

A neural network encompasses an input layer, one or more hidden layers, and an output layer. The input layer consists of all the independent variables, while the output layer comprises the dependent variable. Each hidden layer has several nodes. Each node connects to another and receives weighted signals from all the nodes in the previous layer and does a transform on the linear combination of signals. The transform or activation function we choose is logistic function. The neural network process is represented in Figure 8.

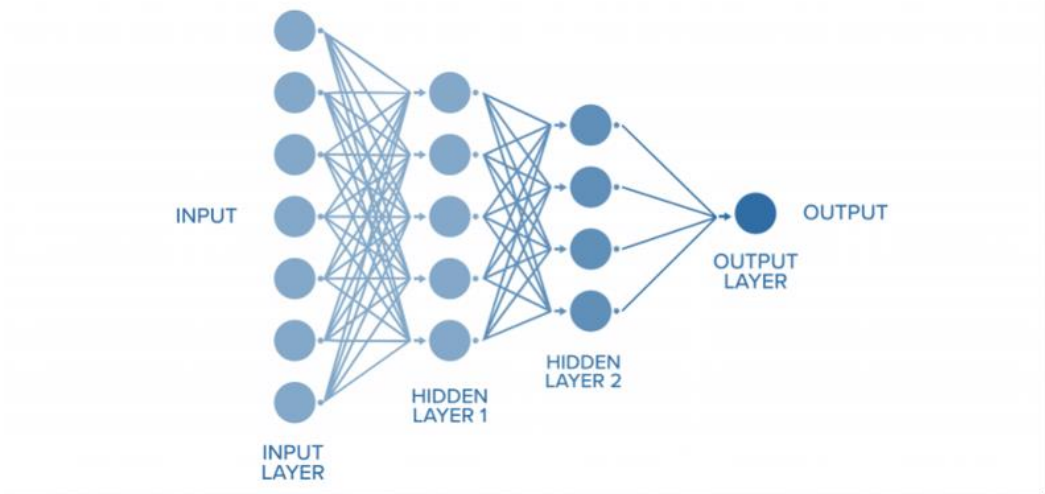


Figure 8: Neural network with two hidden layers

Hyperparameters tuned:

- *hidden_layer_sizes*: Represent the number of neurons in each hidden layer.
- *activation*: Activation function for the hidden layer.
- *solver*: The solver for weight optimization.
- *learning_rate_init*: Control the step-size in updating the weights.

Model performance:

Model	Iteration #	Variables	Hyperparameters				Average FDR at 3%		
Model Name	Iteration	Total Variables	Hidden_layer_sizes	Alpha	activation	Learning_rate_init	Train	Test	OOT
	1	20	(3,)	0.0001	relu	0.01	0.736	0.727	0.430
	2	20	(3,3)	0.0001	relu	0.1	0.686	0.653	0.409
	3	20	(5,)	0.001	relu	0.01	0.760	0.748	0.478
Neural	4	20	(10,10)	0.005	relu	0.001	0.811	0.796	0.480
Networks	5	20	(10,10,10)	0.001	relu	0.01	0.813	0.788	0.491
	6	20	(5,3)	0.001	relu	0.001	0.711	0.697	0.461
	7	20	(30,)	0.001	relu	0.01	0.820	0.784	0.487
	8	20	(30,)	0.0001	tanh	0.001	0.843	0.813	0.518

Random Forests

Random Forest is a statistical algorithm which is a slight improvement from a bagging algorithm. In a bagging algorithm, a number of decision trees are made by taking bootstrapped samples of rows from the dataset involving all the possible numbers of predictors. However, this algorithm still contains the problem of higher variance from the decision trees. This is because, all the decision trees contain the most powerful predictor among all which induces high correlation between the predictions of the individual predictions of decision trees. As a result, the problem of high variance in the result persists. To overcome this problem, random forest does not involve all the predictors in making the decision trees. It randomly selects the number of predictors and subsequently makes the decision trees. This helps in making a sequence of uncorrelated trees where the most powerful predictor is not always present in every decision tree. This helps in reducing the variance and helps in getting rid of overfitting. Traditionally, we choose the number of predictors, $m = \sqrt{p}$, where p is the total number of predictors in the data. Figure XX illustrates the differences between decision tree and random forest. All features are included in decision trees, whereas a random subset of the features is included in each tree of the random forest.

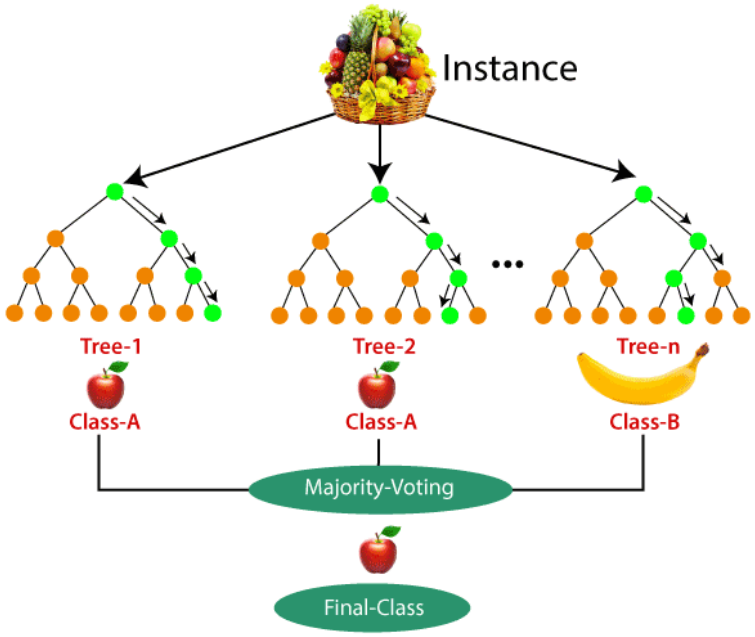


Figure 9. Random forest algorithm

Random forest includes various hyperparameters which we can tune in order to improve our accuracy.

Hyperparameters tuned:

- ***Number of Estimators:*** The number of trees that the algorithm will create, where the most common class predicted by each individual tree is the final prediction of the algorithm. Too many estimators in the model tends to cause overfitting. The number should be chosen such that there is little or less improvement in the testing accuracy.
- ***Max_features:*** The maximum number of features that the algorithm can choose to make each decision tree. The most common number of features to be selected are given by $\sqrt{p}$, where p is the total number of predictors.
- ***Max_depth:*** The depth of each tree in the forest. Higher depth results in higher splits in each tree which may lead to overfitting.
- ***Min_samples_leaf:*** The minimum number of samples required to be at a leaf node.
- ***Min_samples_split:*** The minimum number of samples required to split an internal node.
- ***Class_weight*** – "balanced_subsample": Used to balance the majority and minority classes of every bootstrapped sample for every tree grown.

Model performance:

	Iteration	# of variables	# of trees	# of features	depth	class weight	TRAIN	TEST	OOT
RANDOM FORESTS	1	30	500	6	20	(1,91)	1.000	0.920	0.570
	2	30	500	6	20	"balanced_subsample"	1.000	0.897	0.565
	3	30	50	5	8	(1,91)	0.964	0.806	0.448
	4	30	50	5	8	"balanced_subsample"	0.969	0.816	0.358
	5	30	50	5	8	None	0.863	0.832	0.579
	6	30	100	5	8	None	0.866	0.815	0.579
	7	30	150	5	8	None	0.865	0.829	0.577
	8	30	200	5	8	None	0.864	0.833	0.582
	9	30	200	5	8	(1,91)	0.965	0.802	0.486
	10	30	200	5	9	None	0.887	0.829	0.584
	11	30	200	5	10	None	0.914	0.854	0.580

Boosted Trees

In general, boosting is a method of improving prediction results by iteratively training a series of weak learners so that they may produce a strong learner. In applying the concept of boosting to decision trees, we get the boosted tree algorithm. The boosted tree algorithm is a supervised machine learning method that consists of using a series of simple or constrained decision trees as weak learners. The learners are grown sequentially such that each subsequent tree in the series is grown using information about the misclassified results from the previous tree. In a very simplistic form, if we let $h(x)$ be a weak learner's output and we let w be the weight relative to the weak learner's accuracy, then the predicted output ($\hat{y}(x)$) for the t-th iteration is:

$$\hat{y}(x) = \sum_{t} w_t h_t(x)$$

As the boosted tree algorithm progresses, it applies a weight (w_t) to each data record in the dataset relative to the accuracy of the output. The weights (w_t) for a record are therefore dependent on whether a record was misclassified with weights (w_t) increasing for misclassified records as it is subsequently misclassified throughout the algorithm so that each iterative decision tree in the series

can place a heavier importance on that record to increase the likelihood of properly classifying the record. The overall idea is that by fitting a series of weak learners to the residuals (i.e. misclassified results), we slowly improve the model in areas where it does not perform well. The ultimate goal is to minimize the losses or the residual error in the objective function to improve prediction results and accuracy. Refer to figure 19 below for a visualization of boosted trees.

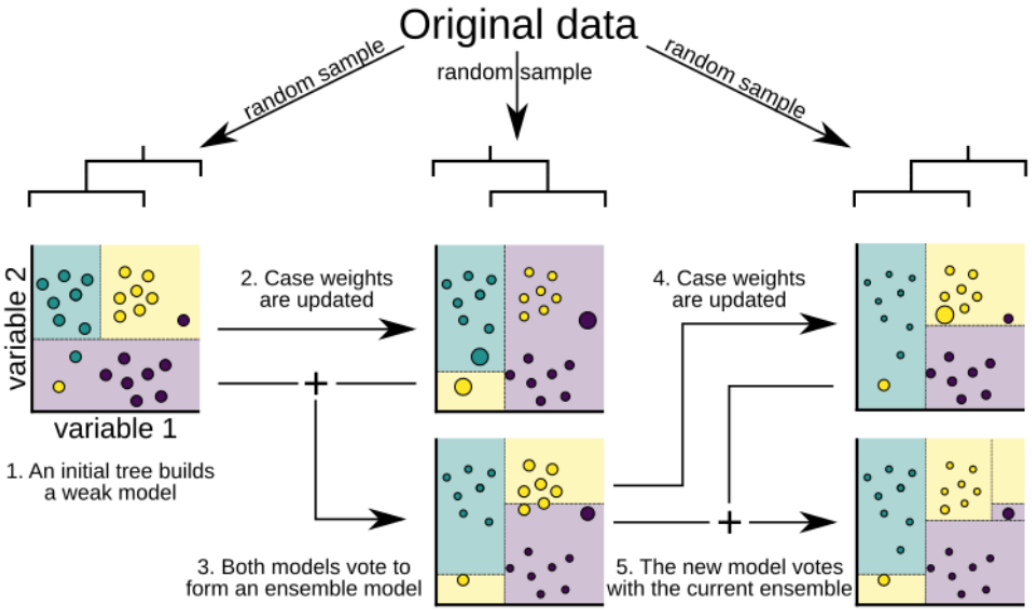


Figure 10. Boosted Trees Algorithm

There are a variety of boosted tree algorithms readily available for use within many of the popular machine learning libraries. For instance, Scikit-Learn and XGBoost are two popular machine learning libraries that contain boosted tree algorithms. For this dataset, we opted to use the XGBClassifier package from XGBoost's machine learning library for our boosted tree algorithm. For any boosted tree algorithm, there are a few key hyperparameters that help with tuning the algorithm for a particular problem. Namely, the number of decision trees, the size of the decision tree, and the algorithm's learning rate.

For the XGBClassifier, the number of decision trees corresponds to the "n_estimators" parameter, the size of the decision tree corresponds to the "max_depth" parameter, and the learning rate corresponds to the "eta" parameter. Additionally, since this particular dataset is unbalanced, we also found it helpful to scale the weights associated with the unbalanced classes by making use of the "scale_pos_weight" parameter. In selecting values for all of these hyperparameters, the information below provides some general guidelines when attempting to find optimal values:

- ***Number of decision trees***: Too many trees can cause a model to overfit. However, it is generally good practice to increase the number of trees until there is little to no improvement in the model.
- ***Decision tree depth:*** A higher tree depth correlates to increased complexity. In a boosted tree algorithm, we are attempting to use a series of weak learners. Thus, shorter or less complex decision trees are generally used in a boosted tree algorithm.
- ***Learning rate:*** The speed at which the algorithm learns from the updated weights at each iteration in the series can be controlled to help make the algorithm more robust to overfitting. Lowering the learning rate can shrink the weights at each step and therefore

slow down the speed at which the algorithm learns. This inherently means that more trees are often needed to tune the model and more time will be needed for the model to finish training. Common values for the learning rate are 0.01 and 0.001. However, when tuning this parameter, higher values may prove to be more optimal if the lower values show to result in little to no improvement.

- ***Scaling Weights for Unbalanced Classes***: When a dataset has unbalanced classes, it may be prudent to take steps towards applying weights to one of the classes to either place a heavier emphasis on the minority class, or a lesser emphasis on the majority class. A typical value to use is the $sum(negative\ instances)$ / $sum(positive\ instances)$.

In applying the XGBClassifier to the dataset, we used a short list of values for each of the parameters above to conduct an initial analysis of the performance of the algorithm on the dataset. We also wanted to conduct an initial attempt at dealing with the unbalanced classes by downscaling the majority class to 75% and using the "scale_pos_weight" parameter with a value obtained by using the $sum(negative\ instances)$ / $sum(positive\ instances)$. Lastly, we ran 10 iterations with different random states used for the training and testing splits on the dataset for each set of parameters and averaged the FDR results over those 10 iterations due to the inherent variances that can occur with random initializations. The results are shown in figure 20 below.
In addition to the results above, we attempted to use cross-validation and parameter tuning with ScikitLearn's GridSearchCV package to evaluate the XGBClassifier with additional values for each of the three main parameters seen in figure 20. Unfortunately, we found this method caused overfitting and did not enable our model to improve. Given the lack of improvement and the results from our initial analysis on the other algorithms, we opted to forego any further detailed hyperparameter tuning for boosted trees and the XGBClassifier model.

Model performance:

	Iteration	# of variables	CV	Max. Depth	Learning Rate	TRAIN	TEST	OOT
BOOSTED TREES	1	30	600	3	0.01	0.884	0.823	0.570
	2	30	600	4	0.01	0.938	0.866	0.527
	3	30	600	5	0.01	0.975	0.884	0.470
	4	30	800	3	0.01	0.918	0.849	0.554
	5	30	800	4	0.01	0.956	0.895	0.508
	6	30	800	5	0.01	0.993	0.893	0.424
	7	30	1000	3	0.01	0.934	0.874	0.543
	8	30	1000	4	0.01	0.979	0.893	0.515
	9	30	1000	5	0.01	1.000	0.910	0.444
	10	30	600	3	0.001	0.701	0.674	0.498
	11	30	600	4	0.001	0.770	0.718	0.460
	12	30	600	5	0.001	0.840	0.724	0.415

Results

After our initial results from training models with logistic regression, boosted trees, random forest, and neural network, we determined that our random forest model performed the best. The random forest model consistently outperformed the other models given its fraud detection rates for the testing and out-of-time (OOT) validation datasets. In addition to higher FDR, the variance in the average of the 10 results for every iteration was also very low as compared to the other models

Hyperparameter Selection:
In order to determine the optimal selection of hyperparameters, we opted to use grid search with cross validation. We conducted this method twice using the following set of hyperparameters:

GridSearchCV Trial 1:

Hyperparameters Tested	Values
Max_features	5, 6, 7, 8
N_estimators	100, 200, 300, 400, 500, 600, 700
Max_depth	10, 20, 30, 40
Class_weight	default, balanced_subsample

Best Parameters: Max_features = 5, N_estimators = 500; Max_depth = 20

Train FDR	Test FDR	OOT FDR
100	92.03	55.86
With class_weight = 'balanced_subsample'		
100	88.7	55.97

The above table shows the results from the first trial where we found that we were overfitting with an 8% deficit in the testing set from the training set. Also, we did not want to trust a model which was getting a perfect 100% Training FDR as it demonstrates some signs of overfitting.

GridSearchCV Trial 2:

For the next trial, we ran another round of grid search but with a less complex set of parameters as shown below:

Hyperparameters Tested	Values
Max_features	5, 6, 7, 8
N_estimators	50, 100, 150, 200, 250
Max_depth	3, 4, 5, 6, 7, 8, 9,10
Class_weight	default, balanced_subsample

Best Parameters: Max_features = 5, N_estimators = 200; Max_depth = 8, class_weight = default

The following table shows the FDR results that we obtained using the best parameter results:

Max_features	N_estimators	Max_depth	Train	Test	OOT
5	50	8	0.8630	0.8320	0.5787
5	100	8	0.8660	0.8150	0.5787
5	200	8	0.8635	0.8333	0.5815
5	200	4	0.7658	0.7571	0.5301
5	200	5	0.8055	0.7827	0.5720
5	200	6	0.8398	0.7918	0.5753
5	200	9	0.8865	0.8290	0.5837
5	200	10	0.9136	0.8535	0.5800

Figure 11. Random forest results from various hyperparameters

The following table shows the FDR of the 10 runs of the best hyperparameters, and we can see that the results for training, testing, and OOT are very consistent and much higher than the other models:

Run #	Train	Test	OOT
1	0.8709	0.8279	0.5698
2	0.8777	0.8450	0.5865
3	0.8478	0.8800	0.5754
4	0.8717	0.8230	0.5810
5	0.8641	0.8054	0.5810
6	0.8559	0.8522	0.5810
7	0.8600	0.8181	0.5865
8	0.8539	0.8361	0.6033
9	0.8707	0.8210	0.5586
10	0.8630	0.8250	0.5921
AVG	**0.8635**	**0.8333**	**0.5815**

Figure 12. Random forest results from 10 runs

The following results are the in-depth analysis of the final Random Forest model for training, testing, and out-of-time datasets:

Training	# Records	# Goods	# Bads	Fraud Rate
	56442	55816	626	0.011091

	Bin Statistics					Cumulative Statistics						
Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	564	60	504	10.64	89.36	564	60	504	0.11	80.51	80.4	0.12
2	565	534	31	94.51	5.49	1129	594	535	1.06	85.46	84.4	1.11
3	564	551	13	97.7	2.3	1693	1145	548	2.05	87.54	85.49	2.09
4	565	549	16	97.17	2.83	2258	1694	564	3.03	90.1	87.06	3
5	564	553	11	98.05	1.95	2822	2247	575	4.03	91.85	87.83	3.91
6	565	555	10	98.23	1.77	3387	2802	585	5.02	93.45	88.43	4.79
7	564	559	5	99.11	0.89	3951	3361	590	6.02	94.25	88.23	5.7
8	564	559	5	99.11	0.89	4515	3920	595	7.02	95.05	88.02	6.59
9	565	564	1	99.82	0.18	5080	4484	596	8.03	95.21	87.17	7.52
10	564	561	3	99.47	0.53	5644	5045	599	9.04	95.69	86.65	8.42
11	565	562	3	99.47	0.53	6209	5607	602	10.05	96.17	86.12	9.31
12	564	564	0	100	0	6773	6171	602	11.06	96.17	85.11	10.25
13	564	563	1	99.82	0.18	7337	6734	603	12.06	96.33	84.26	11.17
14	565	561	4	99.29	0.71	7902	7295	607	13.07	96.96	83.9	12.02
15	564	564	0	100	0	8466	7859	607	14.08	96.96	82.88	12.95
16	565	565	0	100	0	9031	8424	607	15.09	96.96	81.87	13.88
17	564	563	1	99.82	0.18	9595	8987	608	16.1	97.12	81.02	14.78
18	565	564	1	99.82	0.18	10160	9551	609	17.11	97.28	80.17	15.68
19	564	564	0	100	0	10724	10115	609	18.12	97.28	79.16	16.61
20	564	564	0	100	0	11288	10679	609	19.13	97.28	78.15	17.54

Figure 13. Random forest - Best model training set results

Testing	# Records	# Goods	# Bads	Fraud Rate
	24190	23948	242	0.01

	Bin Statistics					Cumulative Statistics						
Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	242	64	178	26.4463	73.5537	242	64	178	0.2672	73.5537	73.2865	0.3596
2	242	218	24	90.0826	9.9174	484	282	202	1.1776	83.4711	82.2935	1.396
3	242	234	8	96.6942	3.3058	726	516	210	2.1547	86.7769	84.6222	2.4571
4	242	238	4	98.3471	1.6529	968	754	214	3.1485	88.4298	85.2813	3.5234
5	242	238	4	98.3471	1.6529	1210	992	218	4.1423	90.0826	85.9403	4.5505
6	241	237	4	98.3402	1.6598	1451	1229	222	5.132	91.7355	86.6036	5.536
7	242	242	0	100	0	1693	1471	222	6.1425	91.7355	85.5931	6.6261
8	242	241	1	99.5868	0.4132	1935	1712	223	7.1488	92.1488	84.9999	7.6771
9	242	240	2	99.1736	0.8264	2177	1952	225	8.151	92.9752	84.8242	8.6756
10	242	241	1	99.5868	0.4132	2419	2193	226	9.1573	93.3884	84.2311	9.7035
11	242	241	1	99.5868	0.4132	2661	2434	227	10.1637	93.8017	83.638	10.7225
12	242	242	0	100	0	2903	2676	227	11.1742	93.8017	82.6274	11.7885
13	242	241	1	99.5868	0.4132	3145	2917	228	12.1806	94.2149	82.0343	12.7939
14	242	241	1	99.5868	0.4132	3387	3158	229	13.1869	94.6281	81.4412	13.7904
15	241	241	0	100	0	3628	3399	229	14.1933	94.6281	80.4348	14.8428
16	242	242	0	100	0	3870	3641	229	15.2038	94.6281	79.4243	15.8996
17	242	242	0	100	0	4112	3883	229	16.2143	94.6281	78.4138	16.9563
18	242	241	1	99.5868	0.4132	4354	4124	230	17.2206	95.0413	77.8207	17.9304
19	242	241	1	99.5868	0.4132	4596	4365	231	18.227	95.4545	77.2276	18.8961
20	242	240	2	99.1736	0.8264	4838	4605	233	19.2292	96.281	77.0518	19.7639

Figure 14. Random forest - Best model training set results

OOT	# Records	# Goods	# Bads	Fraud Rate
	12427	12248	179	0.0144

	Bin Statistics					Cumulative Statistics						
Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	124	51	73	41.1290	58.8710	124	51	73	0.4164	40.7821	40.3657	0.6986
2	125	103	22	82.4000	17.6000	249	154	95	1.2573	53.0726	51.8153	1.6211
3	124	113	11	91.1290	8.8710	373	267	106	2.1799	59.2179	57.0379	2.5189
4	124	118	6	95.1613	4.8387	497	385	112	3.1434	62.5698	59.4265	3.4375
5	124	119	5	95.9677	4.0323	621	504	117	4.1150	65.3631	61.2482	4.3077
6	125	124	1	99.2000	0.8000	746	628	118	5.1274	65.9218	60.7944	5.3220
7	124	120	4	96.7742	3.2258	870	748	122	6.1071	68.1564	62.0493	6.1311
8	124	120	4	96.7742	3.2258	994	868	126	7.0869	70.3911	63.3042	6.8889
9	124	122	2	98.3871	1.6129	1118	990	128	8.0830	71.5084	63.4254	7.7344
10	125	125	0	100.0000	0.0000	1243	1115	128	9.1035	71.5084	62.4049	8.7109
11	124	124	0	100.0000	0.0000	1367	1239	128	10.1159	71.5084	61.3924	9.6797
12	124	123	1	99.1935	0.8065	1491	1362	129	11.1202	72.0670	60.9469	10.5581
13	125	121	4	96.8000	3.2000	1616	1483	133	12.1081	74.3017	62.1936	11.1504
14	124	124	0	100.0000	0.0000	1740	1607	133	13.1205	74.3017	61.1812	12.0827
15	124	124	0	100.0000	0.0000	1864	1731	133	14.1329	74.3017	60.1688	13.0150
16	124	121	3	97.5806	2.4194	1988	1852	136	15.1208	75.9777	60.8568	13.6176
17	125	124	1	99.2000	0.8000	2113	1976	137	16.1332	76.5363	60.4031	14.4234
18	124	124	0	100.0000	0.0000	2237	2100	137	17.1457	76.5363	59.3907	15.3285
19	124	122	2	98.3871	1.6129	2361	2222	139	18.1417	77.6536	59.5119	15.9856
20	124	122	2	98.3871	1.6129	2485	2344	141	19.1378	78.7709	59.6331	16.6241

Figure 15. Random forest - Best model OOT Validation Set Results

Cutoff plot:

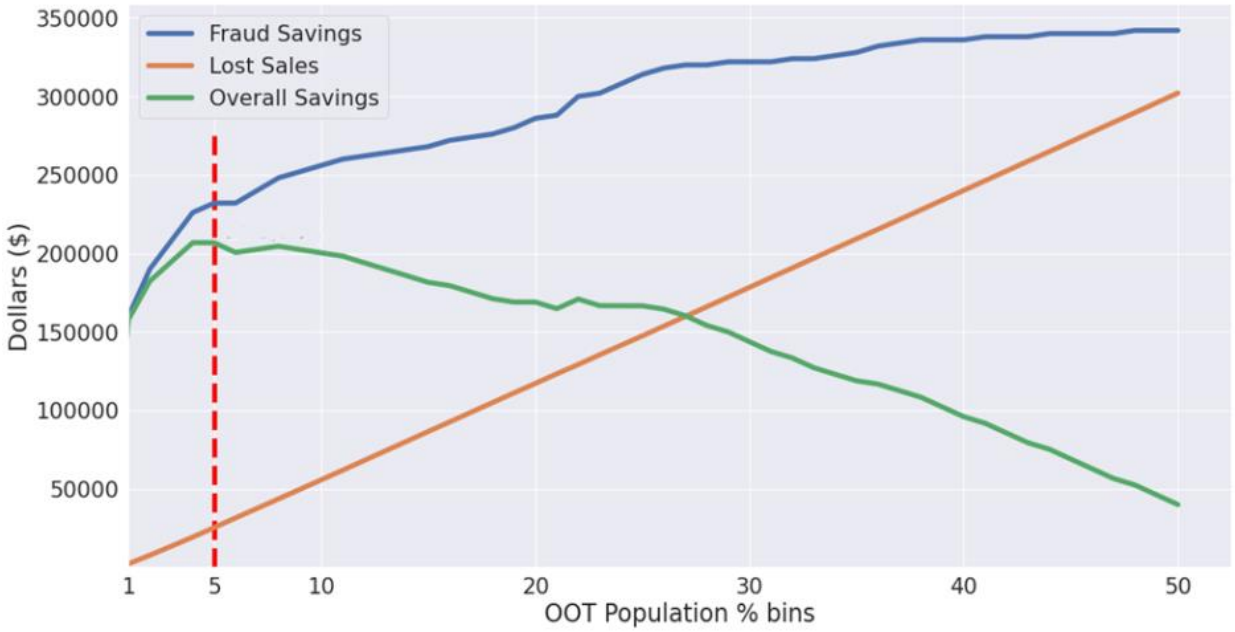


Figure 16. Cutoff Financial Results

To put our model into business practice, we generated the above plot to provide a recommendation for the FDR cutoff point, i.e. any transaction with a score above the threshold at that cutoff point would be classified as fraudulent. Assuming $2,000 gain for every fraud that is detected (true positive, TP) and $50 loss for every non-fraud that is flagged as a fraud (false positive, FP), we plotted the Fraud Savings (blue), the Lost Sales (orange) and the Fraud Savings (green) for the out

of time dataset. According to the analysis, we recommended a **cutoff point at 5%** as this threshold maximizes the profits (P) calculated as follows:

$$P = 2000 * TP - 50 * FP$$

We used the OOT overall savings to extrapolate the annual savings, i.e **$1.56 million**.

Conclusions

A comprehensive analysis of credit card transaction fraud cases was performed. First, the data was cleaned, and all missing values were filled logically. Then, over 2,000 candidate variables were created, and feature selection was performed (filter and wrapper methods) to pick the best 20 variables. The variables were then used in several models: logistic regression, boosted trees, random forest, and neural networks. Our best model to predict fraud was random forest which resulted in an 83.33% FDR at 5% for the testing dataset and a 59.21% FDR at 5% for the OOT dataset.

With more time and resources, we would further explore different techniques to improve our fraud detection performance. In our model building, we found that compared to other algorithms, random forest performed better in reducing variance in the out of time results. We would investigate why this is the case. Furthermore, in order to manage the imbalanced dataset, we applied weights to each class and down sampled the majority class. In the future, we would seek to apply other methods such as up sampling the minority class and adjusting the decision threshold. Moreover, we found it was quite challenging to work with a small dataset containing only a small fraction of fraudulent activity. Thus, we would attempt to collect more data for a more robust analysis.

We could have also explored additional combinations of hyperparameter settings for the models we tried and explored a wider range of algorithms. Additionally, if we didn't have computational constraints, we could have supercharged our feature engineering stage, by potentially creating thousands of additional variables, which likely would have led to greater model performance.

Finally, we hope to further consult experts to generate a more comprehensive collection of candidate variables.

Appendix

Data Quality Report (DQR)

The data analysis contained in this report is primarily from an unidentified government organization in Tennessee for over about 10 years (01/01/2006 to 12/31/2007). The dataset is of real government credit card transactions that were made public by the government organization for accounting purposes. There are 96,753 of records and 10 fields - RECNUM, CARDNUM, DATE, MERCHNUM, MERCH DESCRIPTION, MERCHANT STATE, MERCHANT ZIP, TANSTYPE, AMOUNT, FRAUD. All of them are identity fields except for the record number and fraud label.

Field summary Table

Numeric Fields

Field Name	% Populated	Min	Max	Mean	Std Dev	%Zero
Date	100	2006-01-01	2006-12-31	NA	NA	0
Amount	100	0.01	3,102,045.53	427.89	10,006.14	0

Figure 1

Categorical Variables

Field Name	% Populated	# Unique Values	Most Common Value
Recnum	100	96,753	Unique for all records
Cardnum	100	1,645	5142148452
Merchnum	96.51	13092	930090121224
Merch description	100	13126	GSA-FSS-ADV
Merch state	98.76	228	TN
Merch zip	95.19	4568	38118
Transtype	100	4	P
Fraud	100	2	0

Figure 2

Fields Descriptions

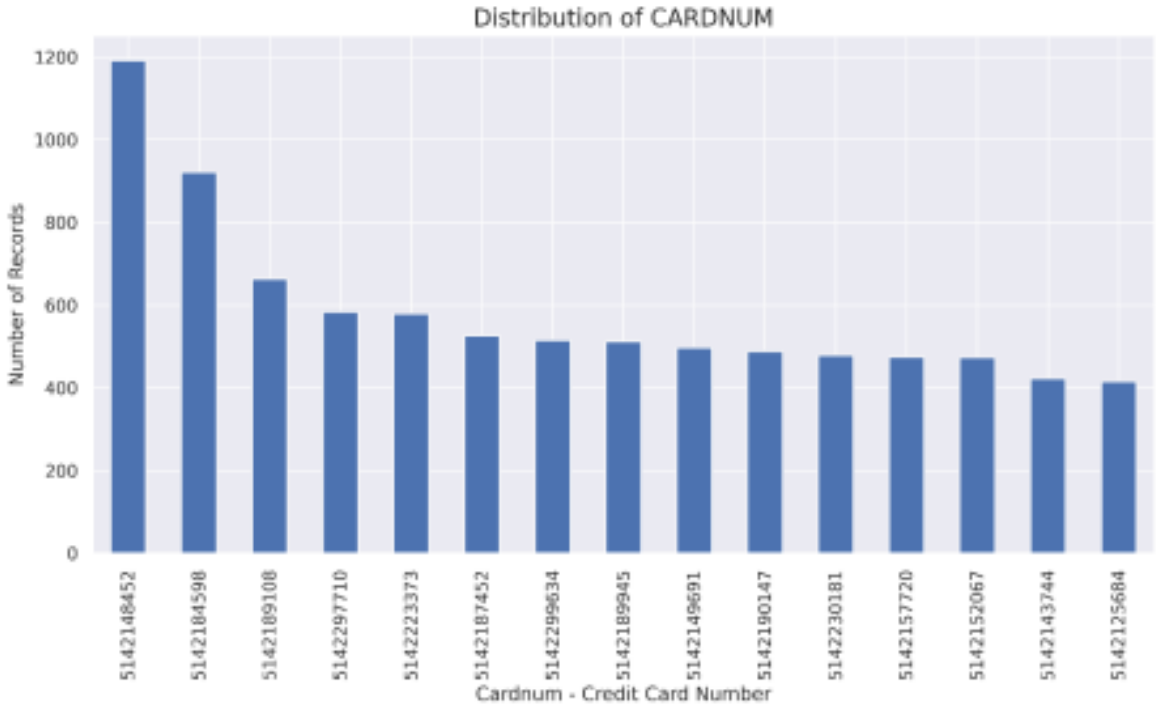
1. RECNUM:

a. **Description:** A categorical data field containing an integer representing the unique card transaction record number identifier from 1 to 96,753. All records in the dataset contain a record number

2. CARDNUM:

a. **Description:** A 10-digit categorical data field containing a shortened version of the credit card number for the card transaction record. The value is missing six internal digits

from the true credit card number. The data field is 100% populated with 1,645 unique values. The bar chart below shows the top 15 values for the "Cardnum" data field.



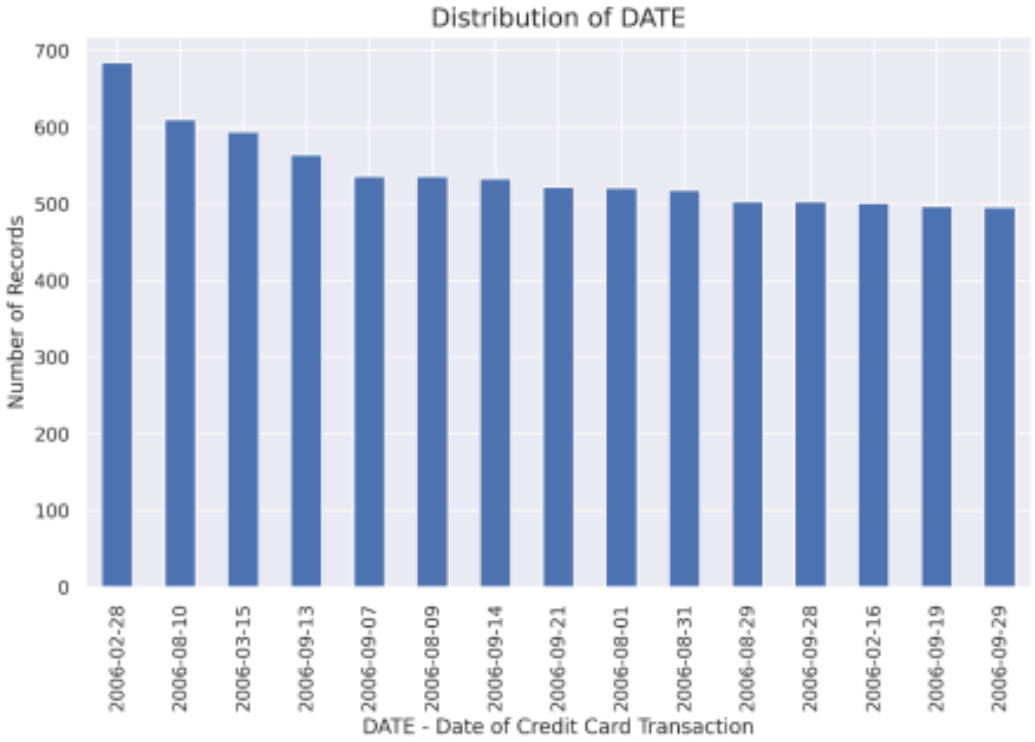
3. <u>DATE:</u>

a) **<u>Description:</u>** A data field containing the date of the card transaction with a raw data format of MM/DD/YYYY. Upon importing the dataset, this data field has a converted format of YYYY-MM DD. This data field is 100% populated and there are 365 unique values representing the 2006 calendar year.

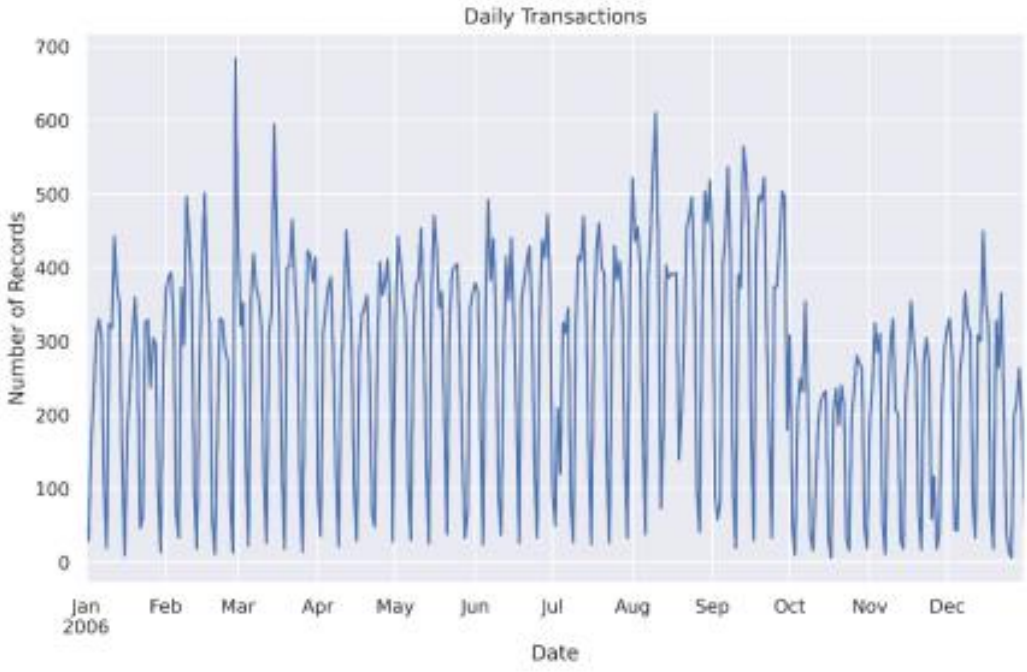
b) It is important to note that the dataset contains both weekly and annual seasonality. For the weekly seasonality, the data shows natural seasonality with regards to weekday and weekend activity.

c) For the annual seasonality, the data shows activity related to government fiscal years. Government fiscal years start in October of the previous calendar year and end in September of the following calendar year. Thus, the 2006 government fiscal year started on 10/01/2006 and ended on 30/09/2006. The 2007 government fiscal year started on 10/01/2006 and ended on 30/09/2007. The dataset therefore contains records for both the 2006 and 2007 government fiscal years.

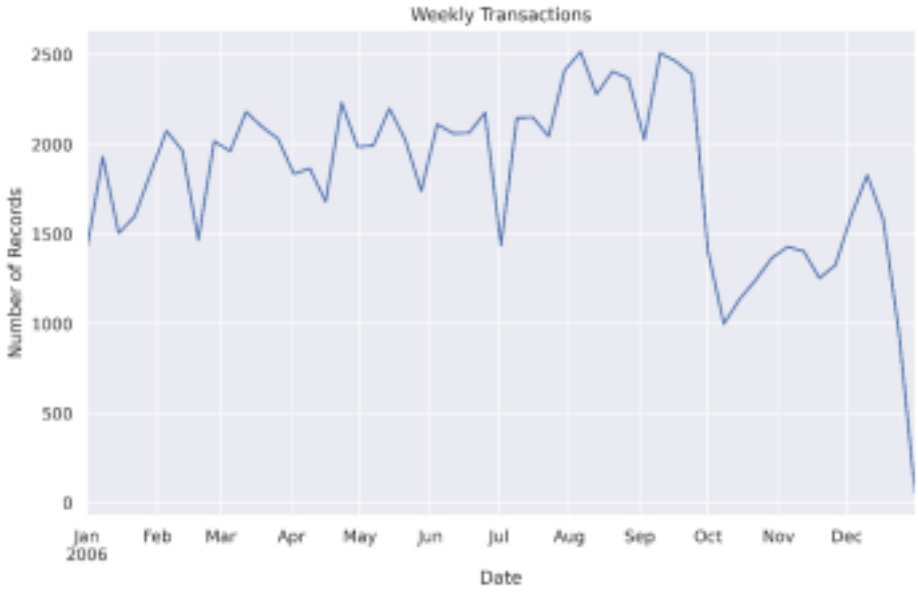
d) The bar chart below shows the top 15 values for the "Date" data field. Given that the 2006 government fiscal year ended on 30/09/2006, we can see that a majority of the top 15 values are towards the end of the 2006 fiscal year in the months of August and September.



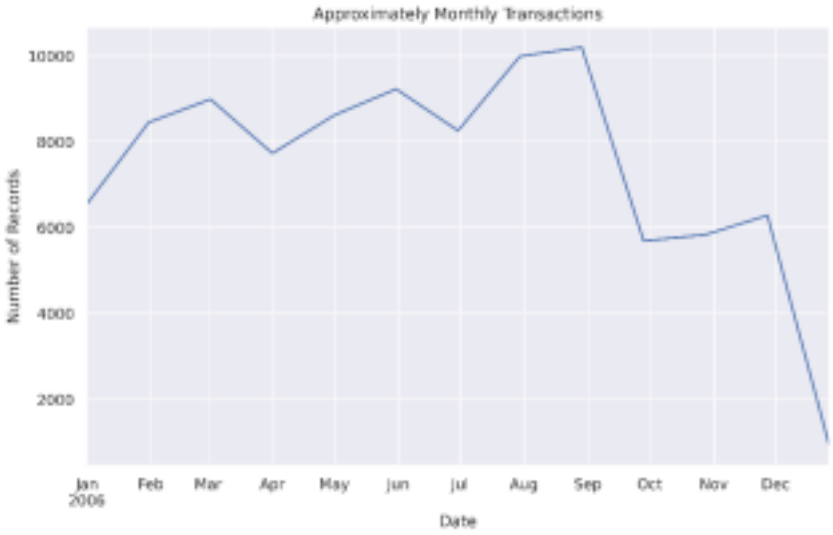
To view the weekly seasonality trends, the Daily Transactions graph below shows 52 peaks and troughs to represent the 52 weeks in a calendar year with the weekend activity depicted as the troughs.



To view the annual seasonality, the Weekly Transactions graph below shows increased activity towards the end of the government fiscal year in September and then a sharp drop-in activity at the beginning of the next government fiscal year in October.

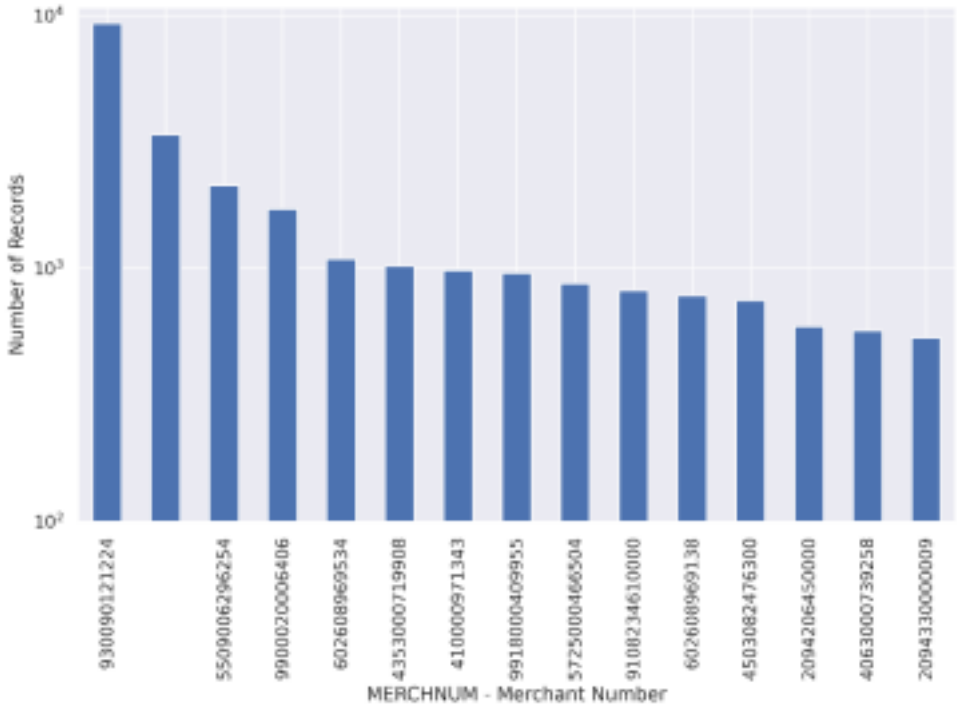


Additionally, the Monthly Transactions graph below provides a slightly different view of the annual seasonality in the data as well as the quarterly seasonality. With regards to the quarterly seasonality, there is increased activity towards the end of each quarter (March, June, September), and then a drop-in activity at the beginning of each quarter (January, April, July, October).



4. <u>MERCHNUM:</u>

a) **<u>Description:</u>** A categorical data field containing a number identifier for the merchants in the dataset. There are 13,092 unique merchant numbers. This data field is 96.5% populated with only 3,375 records missing a value. The bar chart below provides the top 15 merchant numbers in the dataset.

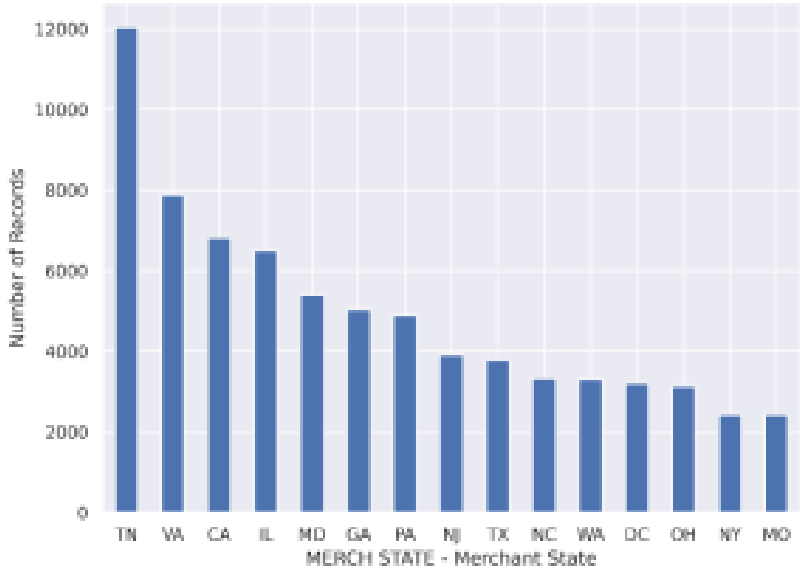


5. MERCH DESCRIPTION:

a) **Description:** A categorical data field containing a short text description of the merchant. This data field is 100% populated and there are 13,126 unique values for the merchant description. The bar chart below provides the top 15 merchant descriptions used for the records in the dataset.

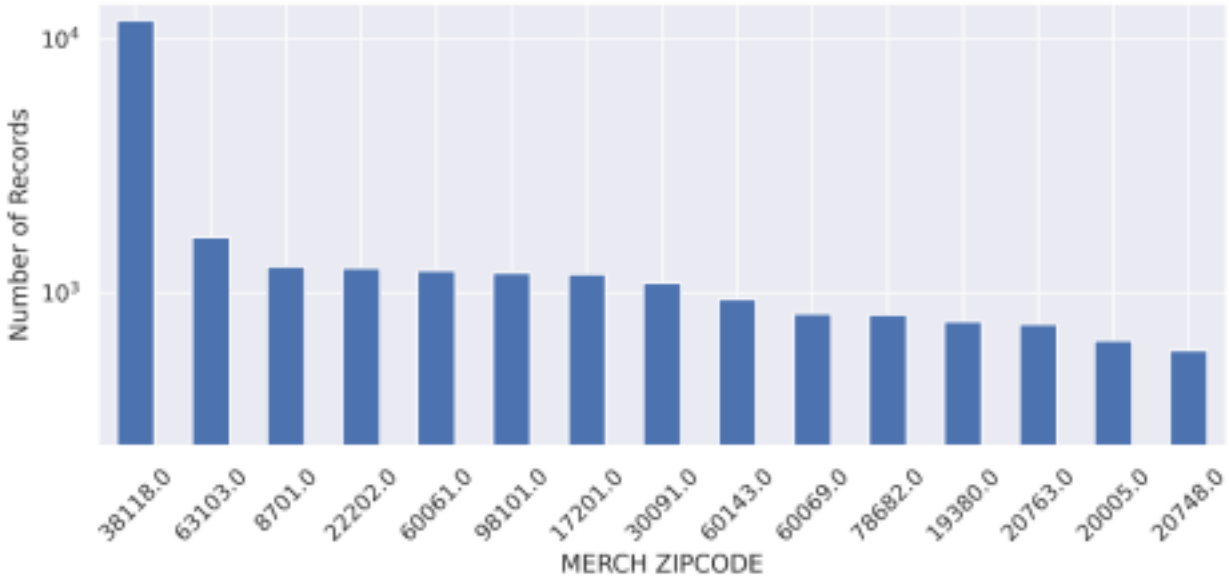
6. MERCH STATE:

a) **Description:** A categorical data field containing either a 2-character or 3-digit value to represent the state associated with the merchant's location. There are 228 unique values for this data field in which 59 of the unique values use 2-character values and 168 of the unique values use 3-digit values. This data field is 98.8% populated with only 1,195 records missing a value. The bar charts below show the top 15 values for the "Merch state" data field.



7. MERCH ZIP:

a) **Description:** A 5-digit categorical data field containing the zip code associated with the merchant's location. It is important to note that if a value has less than 5 digits, then the value has a leading zero(s). There are 4,568 unique values for this data field, and it is 95.2% populated with only 4,656 records missing a value. The bar chart below shows the top 15 values for the "Merch zip" data.

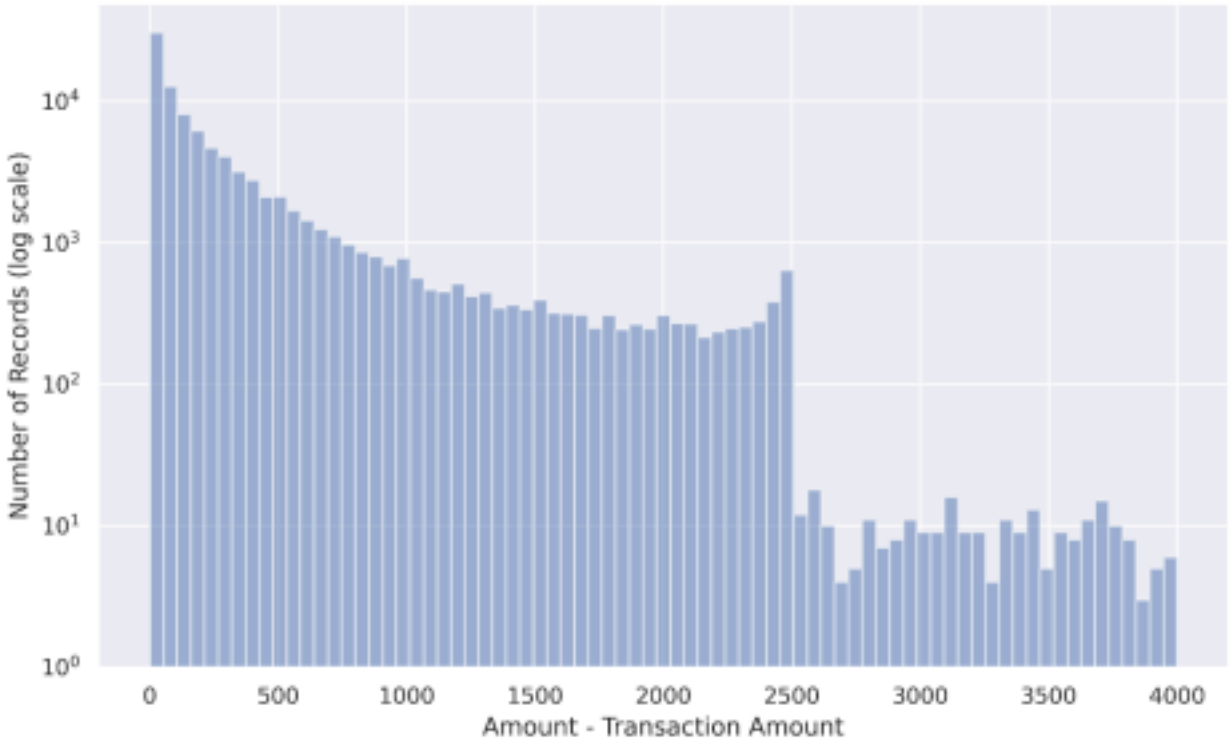


8. TRANSTYPE:

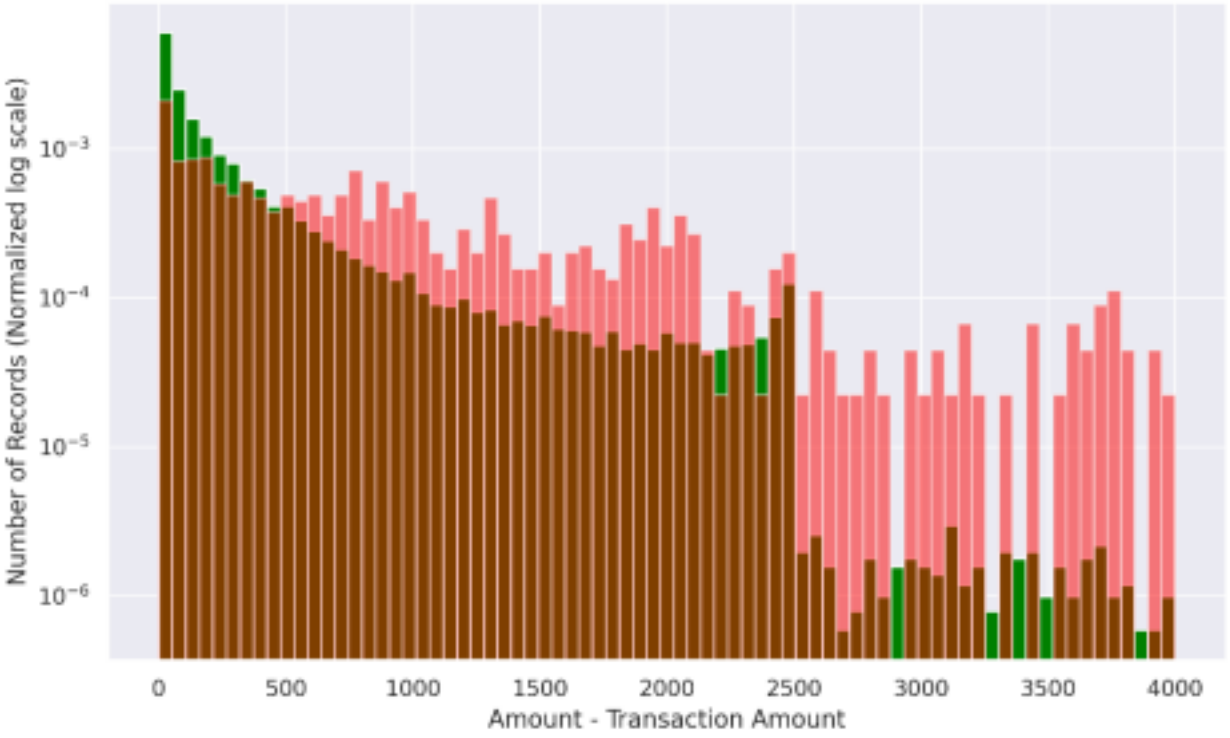
a) **Description:** A 1-character categorical data field containing the transaction type for the record. This data field is 100% populated with only four different values (P, A, D, or Y). There are 99.6% or 96,398 records containing a value of "P" for the transaction type, where "P" stands for purchase. The bar chart below shows all values for the "Transtype" data field.

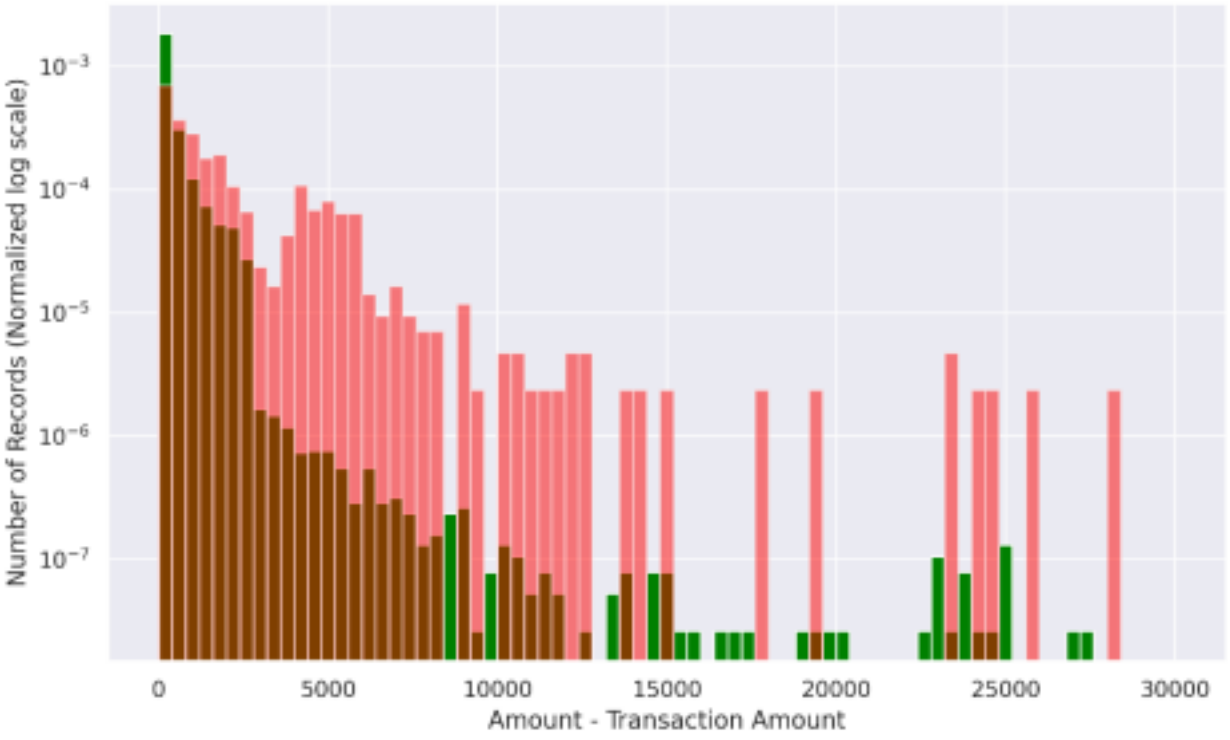
9. <u>AMOUNT:</u>

a) **<u>Description:</u>** A numerical data field containing the dollar amount charged to the credit card for that particular record. This data field is 100% populated and the values range from $0.01 to $3,102,045.53. However, it is important to note that the bulk of the purchases are below approximately $2,500 due to government purchase cards having a single purchase limit of $2,500 in 2010. The distribution plot below shows the dollar amounts for the credit card transactions in the dataset up to a value of $4,000 with the notable drop off depicted at/near the single purchase limit of $2,500.



In addition, the distribution plots below show the differences in the amounts charged to the credit cards between the records labeled with a “0” (green) and “1” (red) in the “Fraud” data field. The first distribution plot shows the amounts charged up to $4,000, while the second distribution plot shows the amounts charged up to $30,000.



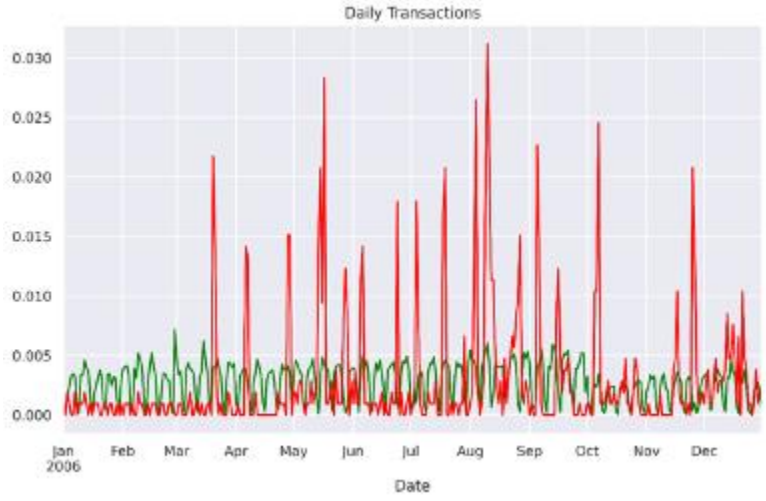


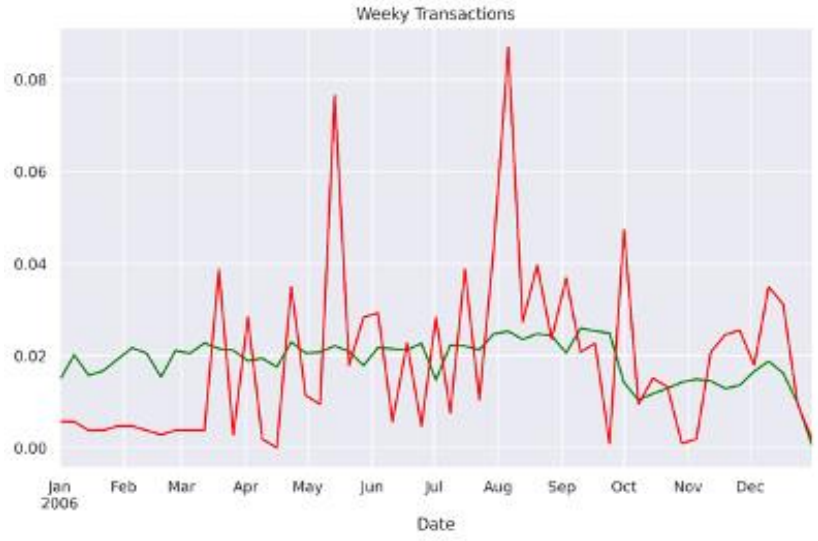
10. <u>FRAUD:</u>

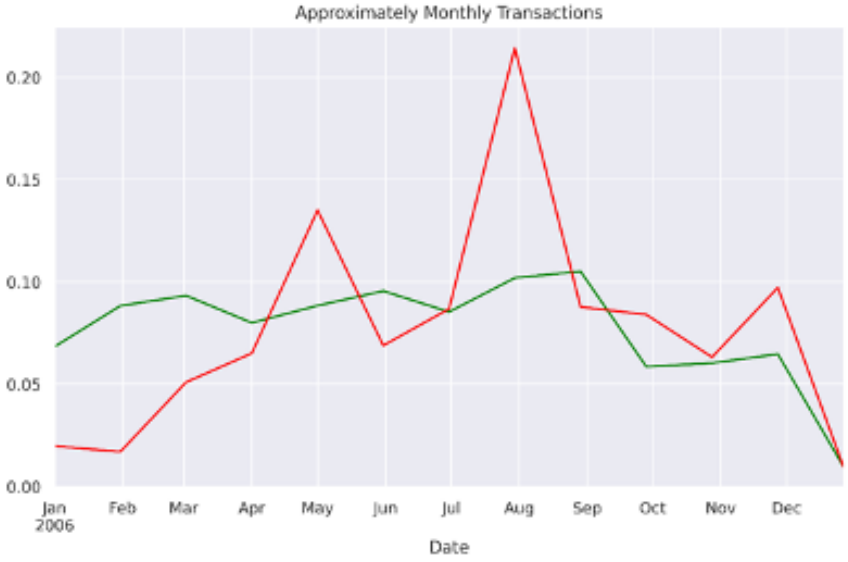
a) **<u>Description:</u>** A binary data field used to label the card transaction record as either a zero or one. There are 95,694 records labeled as "0" and 1,059 records

labeled as “1” in the dataset. This data field is 100% populated.

MISC. DISTRIBUTIONS OF DAILY – WEEKLY – MONTHLY WITH FRAUD CLASSIFIER:




Weeky Transactions
0.08
0.06
0.04
0.02
0.00
Jan
2006
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date


Approximately Monthly Transactions
0.20
0.15
0.10
0.05
0.00
Jan
2006
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date