Building and Evaluating a Supervised Model for Transaction Fraud Detection

Henry Ajagbawa, Zach Cummings, Frank Wang,
Vilhelm Sandberg, Sarah Russell, & Arjun Reddy

University of Southern California

May 9, 2021

Table of Contents

Executive Summary

Transaction fraud is when a stolen card or stolen payment data is used to make unauthorized transactions. As transactions happen in real time, there is a need to also identify and block these fraudulent transactions as they occur. It is important to be mindful that turning away a transaction that is not fraudulent will have negative business impacts, as this could upset a customer and ultimately lead to churn. Thus, the goal of this project is to identify as much fraud as possible, while limiting false positives.

In this project, we were provided with a dataset consisting of 96753 rows and 10 fields, including a fraud label which could be used to train a supervised machine learning algorithm to recognize records that should be classified as fraudulent credit card transactions. We began by performing a preliminary analysis of the data, and then built a supervised model to predict whether or not a record should be classified as fraud.

Initially, we performed a descriptive analysis by building a data quality report to ensure the data was properly organized and ready for further analysis. For each variable we were provided, we examined its distribution and summary statistics. After gaining an understanding of the data and ensuring its integrity, we began constructing additional features that would serve as candidate variables for our predictive model. These engineered features were designed to illuminate typical signals of fraud in real life scenarios.

Once we'd engineered over 300 additional candidate variables, we began the process of reducing dimensionality and identifying the few variables that best distinguished between fraudulent and non-fraudulent records. By first using a filter that combined the rank for the univariate KS and fraud detection rate scores for each variable, we were able to determine a subset of 80 variables that were good model input candidates. We further narrowed this list down to 15 using a Stepwise Forward Selection wrapper.

With our final 15 variables, we trained logistic regression, random forest, neural network and boosted tree models. We adjusted various hyperparameter settings to fine tune each model. After model training, we evaluated the models' performances on the training, testing, and out of time data and selected the model with the best performance on the out of time data. The model that performs the best on the out of time data, will most closely resemble how well the model will perform when applied in practice. Our final model, a random forest algorithm, achieved a Fraud Detection Rate at 3% of .6781. At this model's optimal Fraud Detection Rate (which is approximately 2%), we believe that we can save $1,392,200 annually.

We are confident that our data cleaning, feature engineering, feature reduction, and model building processes could be applied to almost any credit card transaction database and would result in a model that was able to detect fraudulent transactions before they were approved.

Data Description and Exploration

Before beginning feature engineering, feature selection, and modeling, we first performed exploratory data analysis to identify key trends in the data and ensure that the distributions in the provided data were reasonable and sensible. Below are the summary statistics for all original data fields:

Numerical Fields

Field Name	Field Type	# of Records	% Populated	# Zeros	Mean	Standard Deviation	Min	Max
Amount	numeric	96753	100	0	427.89	10006.14	0.01	3102045.53

Date Fields

Field Name	Field Type	# of Records	% Populated	# Zeros	Earliest Date	Latest Date	Most Common Date
Date	datetime	96753	100	0	1/1/2010	12/31/2010	2/28/2010

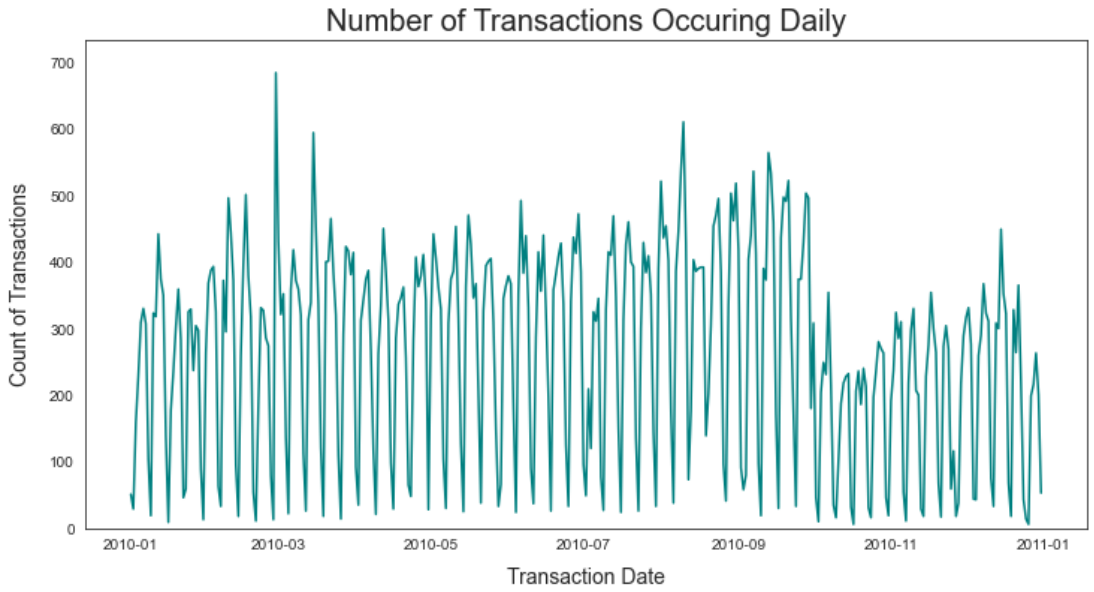
Categorical Fields

Field Name	Field Type	# of Records	% Populated	# Zeros	# Unique Values	Most Common Value
Recnum	categorical	96753	100	0	96753	N/A
Cardnum	categorical	96753	100	0	1645	5142148452
Merchnum	categorical	93378	96.51	0	13092	930090121224
Merch description	categorical	96753	100	0	13126	GSA-FSS-ADV
Merch state	categorical	95558	98.76	0	228	TN
Merch zip	categorical	96753	100	0	4568	38118
Transtype	categorical	96753	100	0	4	P
Fraud	categorical	96753	100	95694	2	0

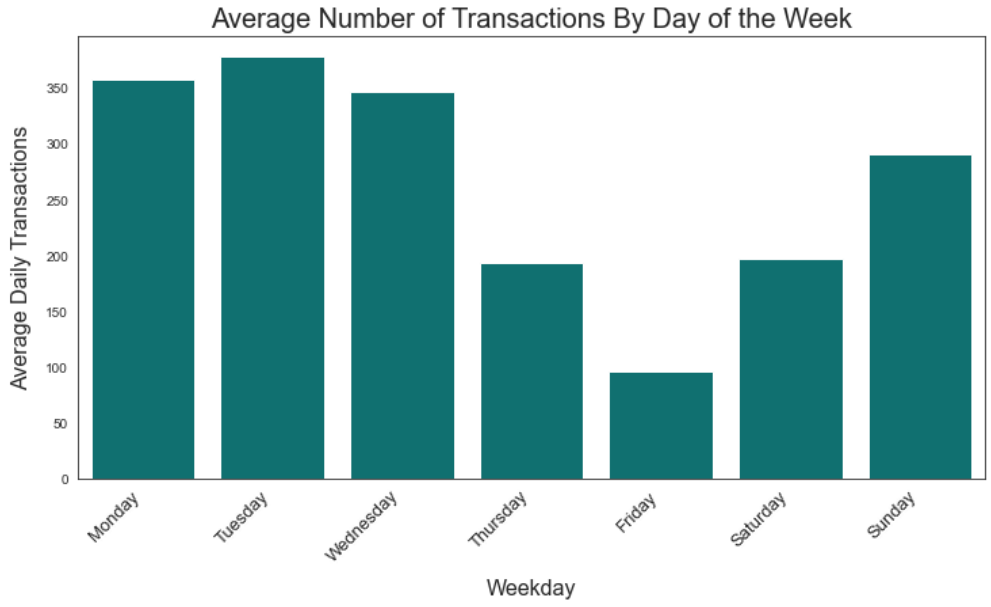
Next, we'll describe some of the key distributions in the original data.

Transaction frequency across time

The daily transaction distribution follows a cyclical pattern on a weekly time scale. Each week, transactions spike at the beginning of the week, and fall at the end of the week. After September 30, transaction quantity per day drops significantly. This is likely due to September 30 corresponding to the end of the fiscal year, when budgets reset.

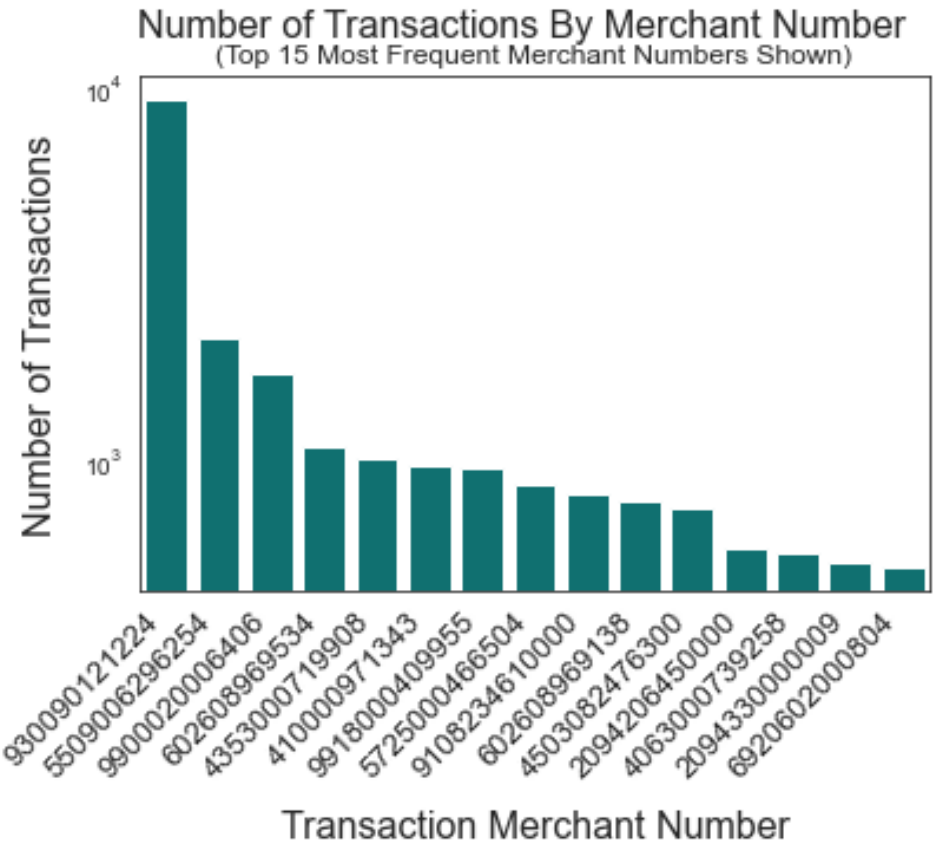


Transaction quantity tends to be higher on Monday, Tuesday, and Wednesday when the week begins, and then falls off at the end of the week.



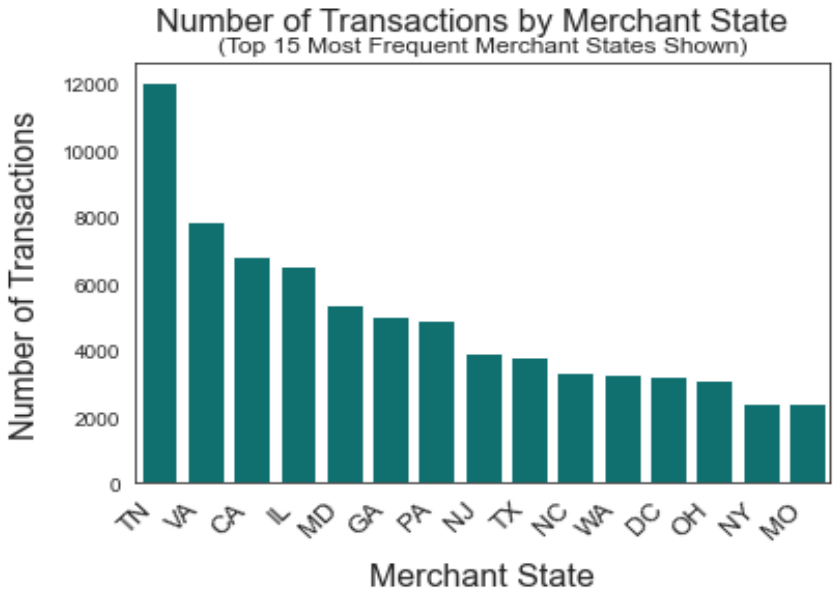
Merchant Number

Transaction count is extremely high for the top most transacted merchant number. The distribution is fairly even from there on out, but the top few merchants make up the vast majority of all transactions. Note the logarithmic y axis scale.



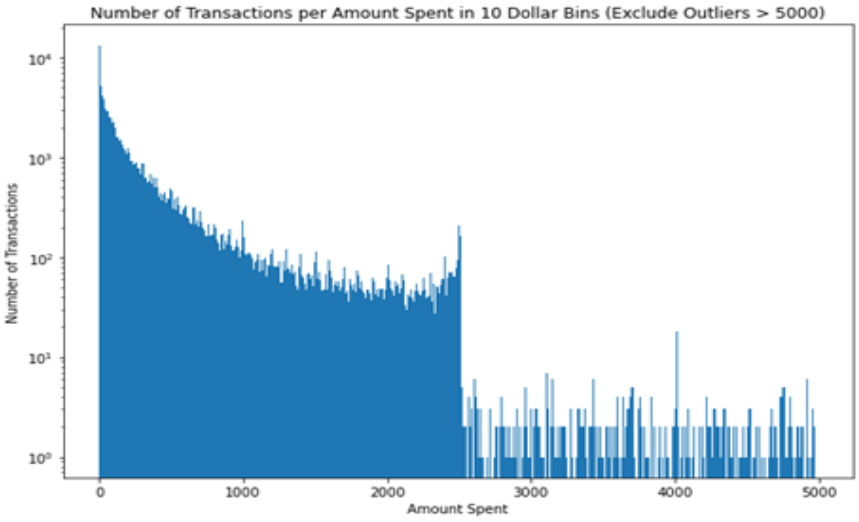
Transaction State

The highest quantity of transactions occurs in Tennesse, followed by Virginia, California, and Illinois. The distribution of transaction frequency per state falls off steadily after Tennessee, which has a relatively high number of transactions.



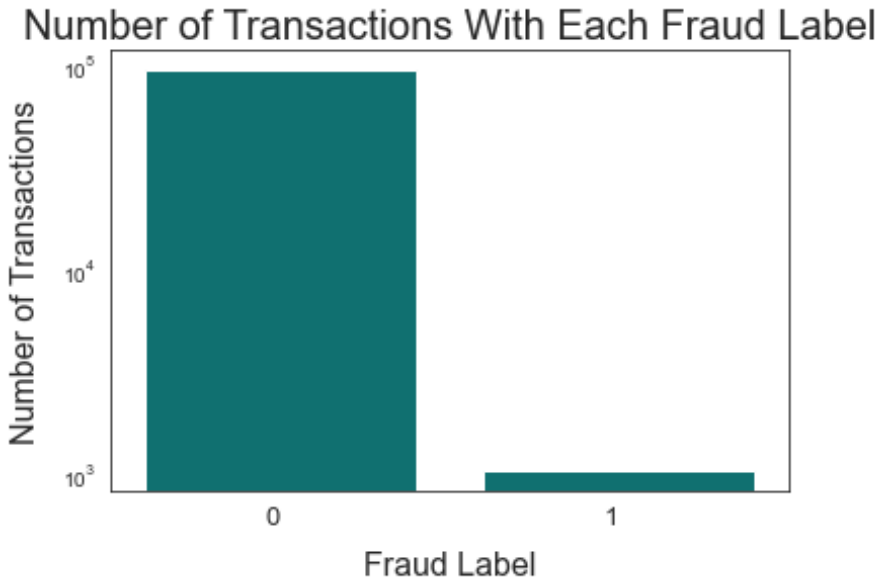
Transaction Amount

The amount of each transaction is right skewed, and most transactions are for relatively low amounts. There is a substantial dropoff in transaction frequency when amount passes $2,500. Please note the logarithmic scale on the y axis in the plot below.



Fraud

The vast majority of records are non-fradulent, with over 96,000 total transactions and only 1,059 fraudulent transactions. As a result, fraud transactions will be preserved at all costs during the data cleaning process. Please note the logarithmic scale on the y axis in the plot below.

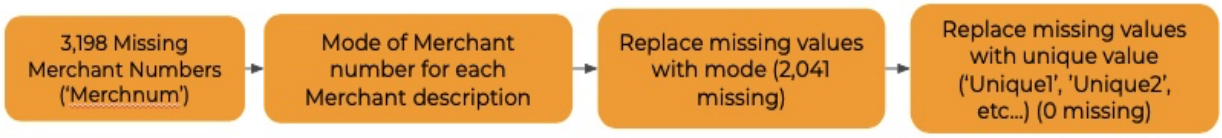


The full data quality report and visualizations of all distributions can be viewed in the report appendix.

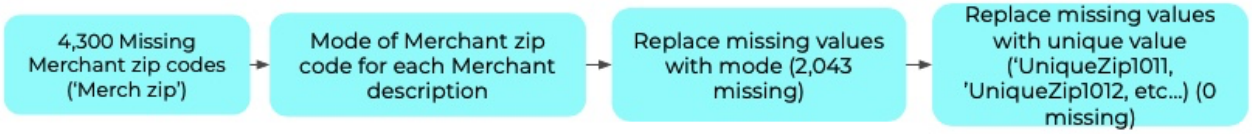
Data Cleaning

For this project, we were given a dataset of 96,753 transaction records, containing 1,059 fraudulent records. First, we removed large outliers and filtered the data so that we were only looking at records of transaction type 'P'. This left us with 96,397 records. When cleaning the data, we noticed that there were three fields('Merchnum', 'Merch state' and 'Merch zip') that contained missing values. Our strategy was to impute missing fields with the most common value of the missing field from a related field and then add unique identifiers for missing fields that we were not able to map back. A further description of how we did this for each of the fields is below:

'*Merchnum'*
To fill in missing values for the merchant number, we made a dataframe that displayed the most common merchandise number for each merchant description. From here, we were able to match records that had a merchant description but no merchant number, with the most common merchant number for the corresponding description. This process left us with 2002 missing merchant numbers. After this, we created unique values for each merchant number that was still missing. The unique identifier was created by tying the string 'Unique" to the recnum number.

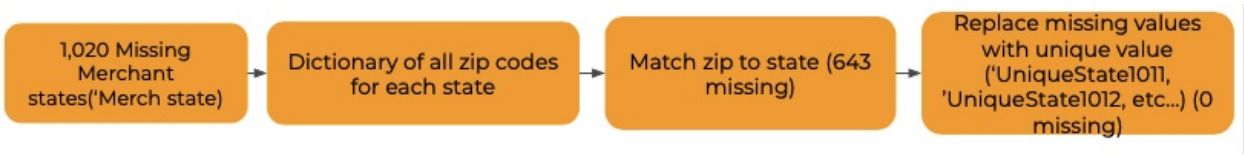


'Merch zip'
To fill in missing values for the merchant zip code, we created a dataframe that tracked the most common zip codes for each merchant description and then matched the missing zip codes back to that most common value when possible. This left us with 2024 missing zip codes. After this, we assigned unique identifiers to any remaining missing zip codes. The unique identifier is the term 'UniqueZip101' in combination with the index number.



'Merch state'
To address missing merchant states, we created a dictionary that kept track of all zip codes associated with each state. If a record had a zip code but not a state, we filled in the missing value using the state that matched with the zip code from our dictionary. This left us with 624 missing

states. For these, we created unique identifiers. The unique identifiers are the combination of the term ‘UniqueState101’ with the index number.



Once we had filled in the missing values, removed the large outlier, and sorted the data so that it only included records of transaction type, ‘P’, we were ready to move on to variable creation.

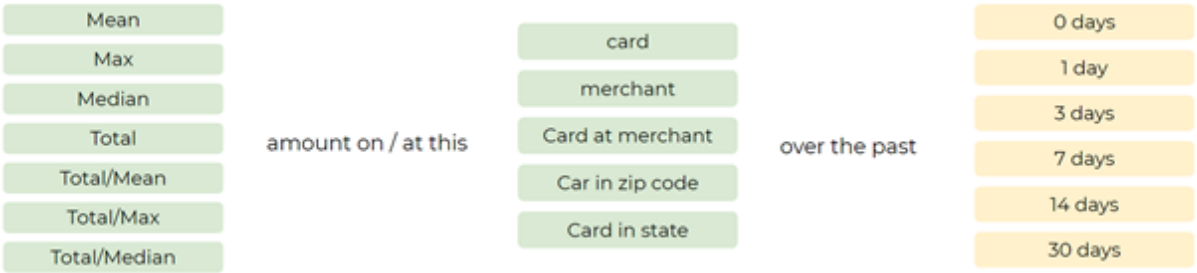
Feature Engineering

When creating fraud detection models, one can typically improve the model's ability to catch fraud substantially by engineering features that highlight various signals of fraudulent behavior that are observed in the real world. To give our model the best chance of success, we created two types of variables beyond the fields we were provided: features that mathematically rearrange the existing variables, and variables based on Benford's law concepts. For the first type of variable, we created four different subtypes: Amount Variables, Frequency and Day since Variables, Relative Velocity variables, and a Day of week risk table. Below, we provide detailed information about how we created these variables and the reasoning behind building them in the first place.

Amount Variables

Reason for creating the variable: The average, max, median, and total amount of money spent in transactions on a given card or at a given merchant over the past x number of days will likely change to unusual quantities as fraudsters rush to use stolen financial information before being caught.

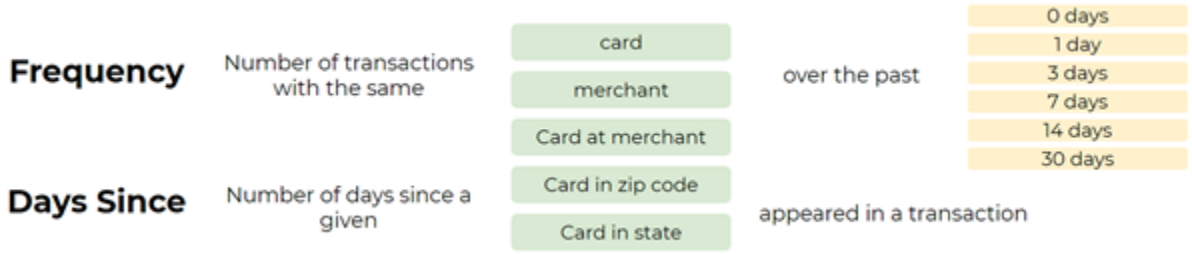
The variable: By creating variables that track various statistical values (Mean, Median and Total/ Mean) in amounts over different time periods on different cards and at different merchants, we tried to train the model to identify unusual spending patterns. Many of the fraudulent transactions are on a card that shows an unusually high average amount spent over the past 3 days compared to non-fraudulent transactions on the same card.



Frequency and Day since Variables

Reason for creating the variable: When a fraudster steals credit card information, it is obvious that the frequency of each transaction on a given card at a given merchant or location is likely to increase. The number of days since the card last used or the card with the same zip code appeared in a transaction may have more chance to reduce.

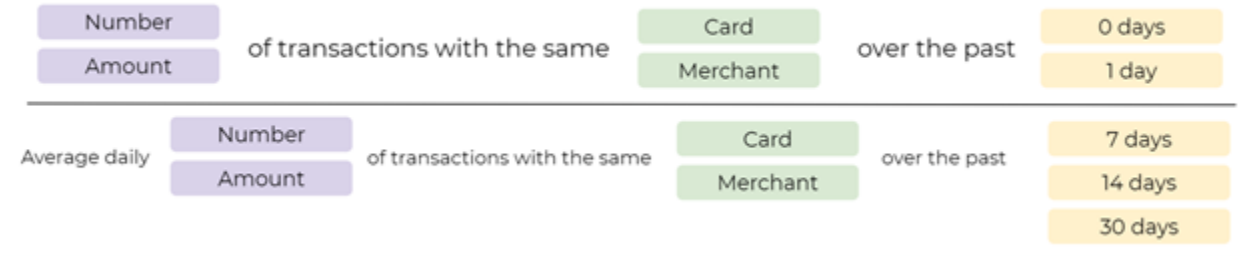
The variable: To detect these sometimes-subtle changes in card activity patterns, we used frequency variables that track the number of transactions made in give time period, and days since variables that investigate the number of days that have passed since the last purchase on a given card, at a given merchant, or on a given card in the same state, zip code, and merchant.



Relative Velocity Variables

Reason for creating the variable: Fraudsters often spend rapidly on the same card and at the same merchants once they have stolen credit card information. Therefore, we should be able to find fraud transaction when the number and total amount of transactions suddenly jumps up in a recent time relative to the average number and number of transactions in the less recent past.

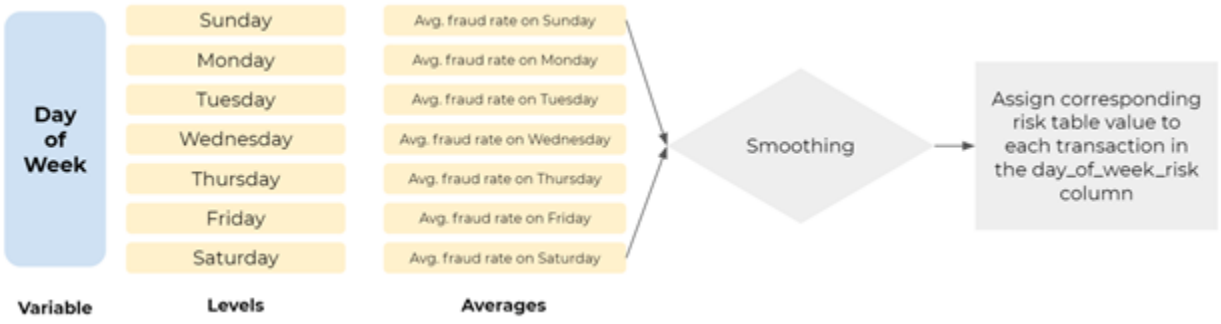
The variable: To detect these bursts in card activity, we used relative velocity variables that compared card usage and merchant usage in the last 0-1 days with the average usage across the last 7, 14, and 30 days. High relative velocities are suspicious.



Day of Week Risk Table

Reason for creating the variable: A risk table assigns the smoothed average of the value for the target variable for each level of a categorical variable. This allows us to incorporate categorical variables into our predictions in useful ways.

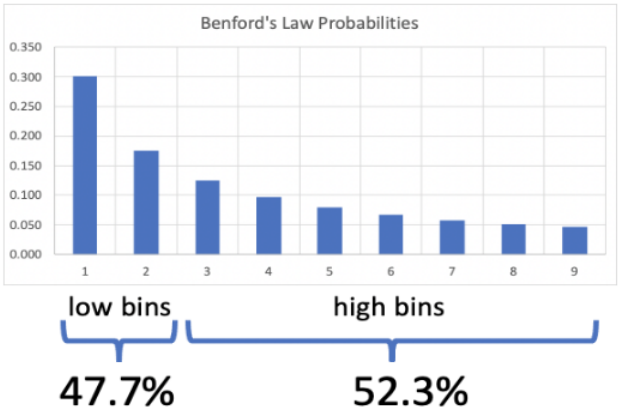
The variable: For each day of the week (each level of the Day of Week variable), we took the average fraud rate for that day of the week and ran it through a smoothing formula. The smoothing serves the purpose of pulling the risk table value for days of the week with a low sample size toward the overall average fraud rate. The amount of correction is proportional to the sample size for that day.



Benford's law Variables

Reason for creating the variable: As a merchant or a cardholder, fraudsters may construct many fraudulent transactions. When they just make up transactions, they typically will not be aware of the Benford's Law phenomenon and its implication on distributions of transactions. Thus, fraudulent transaction amounts tend to be uniformly distributed random numbers.

The variable: To detect these violations against Benford's Law, we examined the amount distributions for each merchant and cardholder and compared it with how it's expected to look if Benford's Law is followed. We achieved this by splitting the first digit distributions into 2 bins and we expect to see 52.3/47.7 = 1.096 ratio of digits (3 through 9) to (1 and 2).



1	0.301
2	0.176
3	0.125
4	0.0969
5	0.0792
6	0.0669
7	0.0580
8	0.0512
9	0.0458

47.7%

52.3%

Feature Selection

In the previous section, we discussed the creation of many candidate variables. However, to use every field as a variable in our final model candidates would introduce a host of modeling problems. Firstly, modeling with hundreds of variables is computationally expensive, and can take so much time to run that it becomes infeasible to test many combinations of hyperparameters. Additionally, using hundreds of predictors creates immense dimensionality in the input data, which in turn increases sparsity, making all points outliers and obscuring important non-linear data patterns. To reduce dimensionality to a more manageable set of input variables, we employed a feature selection process consisting of two main mechanisms: a filter and a wrapper.

Note that before beginning the feature selection process, we set aside an out of time (OOT) set, which includes all records with transaction dates after October 1. We also excluded the first two weeks of transactions from the feature selection process, since many of the features we engineered are temporally dependent, and may take time to fully develop into good predictors. For example, looking at the count of transactions in the last 14 days is the same as looking at the count of transactions in the last 7 days if you're evaluating the variables on the 7th day of the year.

Filter

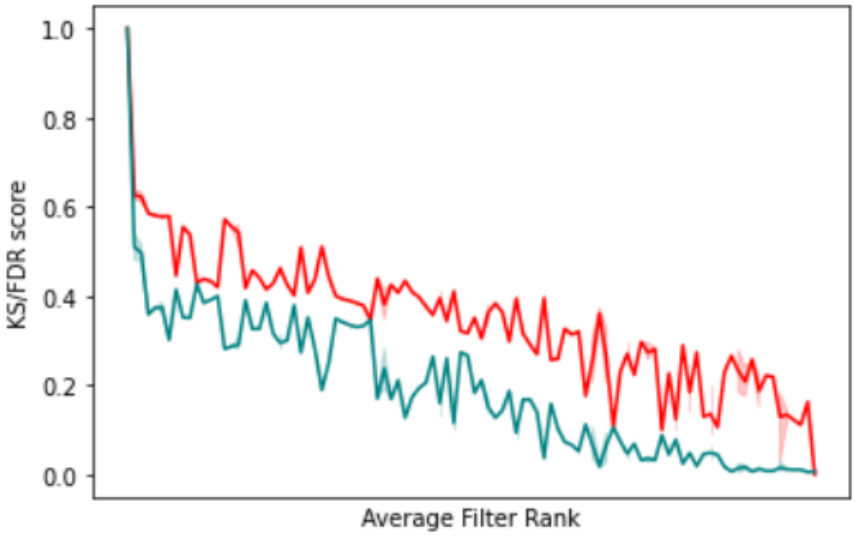
Since we began with 325 candidate variables, which is way too many to use for a good predictive model, we first utilized a filter process to substantially reduce dimensionality in a fast and computationally inexpensive way. We decided to keep the top 80 variables as ranked by our filter. The exact number of variables to keep after the filtering process is arbitrary, but the goal is to err on the side of caution; since the filters are simple measures of variable goodness, it's best to leave ample margin for good predictors to make it through the filter. The filter itself consisted of the average of two different scores: a Kolmogorov-Smirnov (KS) score and a Fraud Detection Rate at 3%.

When used for a binary classification problem, a KS score constructs two normalized distributions for each field. The distributions represent the density of records belonging to a given class for each value of the field in question. The score itself then measures how separate the two distributions are from each other. Said another way, a KS score is a simple way to measure how well a variable distinguishes between goods and bads, which is the key to a variable being helpful in a binary classification model.

The univariate fraud detection rate (FDR) at 3% indicates the percentage of fraudulent records that lie in the tails of a field's data distribution. To calculate univariate FDR at 3%, we take a

subset of the data that includes only the field of interest and the fraud label for each record. We then sort the subset by the field of interest and take the top and bottom 3% of those records. Finally, we count how many frauds we can detect in those 3% subsets and divide the number detected by the total number of frauds in the dataset. We then take the maximum detection rate between the top and bottom of the distribution, and this maximum value represents the univariate FDR at 3% for the field of interest.

We used these two filter methods to assign each candidate variable its own KS and FDR score. Then, we assigned each field a rank corresponding to those scores, and took the average of the two ranks as the final score. For example, if field A had the fifth highest KS score and 9th highest FDR score, its average rank would be the total number of variables minus 7 (in our specific case, the rank would be 318). We then sorted all the fields by this final score in descending order and kept only the top 80 fields. The chart below shows the FDR in teal and the KS score in red for each variable. The x axis is the variable's average rank, descending from left to right. In our filter, higher average ranks indicate higher performing variables.



The highest KS scores start just above 0.6, and the highest univariate FDRs at 3% start just over .5. Both scores descend generally as average rank decreases, although not monotone. The first variable has FDR and KS scores of 1; this variable was a copy of the fraud column, and was used as a test to ensure that the filters would correctly identify the fraud column as being a perfect predictor for itself. A random variable was also tested, and ranked 19th worst among the 325 variables that were filtered.

Wrapper

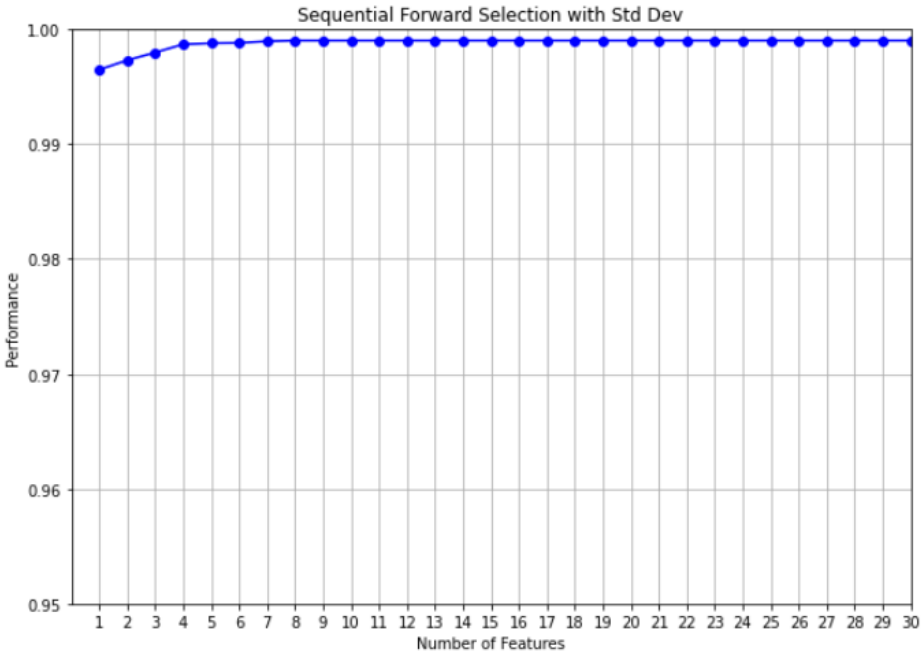
To further narrow down dimensionality, a more intelligent selection method is preferable, since more nuanced differences in importance to the model are harder to detect among the best

variables. To distinguish which subset of candidate variables of the 80 remaining were best to use for our model, we employed a Stepwise Forward Selection wrapper, explained below.

Wrappers are models "wrapped" around the feature selection process, and generally follow filters in the feature selection process. Stepwise forward selection wrappers operate by building a simple univariate model for all *p* candidate features, and ranking these models on some evaluation metric. In our forward selector, we evaluated the prediction accuracy (how many records did the model correctly classify out of all the records it saw?). Then, the variable that was used in the best performing univariate model advances to the next round, where all remaining *p-1* predictors are added to a model with the winning predictor from the first round. The variable that creates the best performing model in combination with the previous round's winning variable is then selected. In the third round, all *p-2* remaining predictors are tried in combination with the first two, and so on. In total, stepwise forward selection tries *p-squared* models, or in our case, 6,400 models. The predictors used to build models in each step k-1 are a subset of the variables used to build models in step k.

Our forward selection model utilized a decision tree classifier with criterion = 'gini', meaning each split's value was evaluated using the gini index, and max_depth = 10, meaning that no more than 10 splits could be made for each tree. Therefore, all 6,400 models built throughout the wrapping process were the same decision tree algorithm with different predictors.

Stepwise forward selectors inherently rank candidate variables by importance, since the more important a predictor is, the earlier the step is for which it appears in the final model. The most important variable as ranked by the wrapper appears in the first step's final model, and the least important doesn't appear in the final model for a step until the last step. When deciding how many candidate fields we wanted to keep, we examined a plot of the variance explained by each model, graphed by the number of features the model contained. As can be seen below, as the number of features increases (or, said another way, as we progress through the final models recommended at each step of the wrapper), the total variance explained increases, but plateaus around 8-10 predictors. However, to be safe (since the wrapper is still based on a sequence of very simple models), we decided to keep 15 variables for our final model.



As the number of features in the model increases, so too does the model's ability to explain variation in the data -- up to a certain point. At 8 predictors, additional variables add complexity without increasing model performance. However, since the wrapper is an inexact process, we decided to keep 15 predictors for our final model to err on the side of caution.

These 15 variables are:

median_amount_by_cardnum_merchnum_in_last_14_days
total_amount_by_cardnum_merchnum_in_last_30_days
median_amount_by_cardnum_merchnum_in_last_1_days
median_amount_by_cardnum_merchnum_in_last_3_days
median_amount_by_cardnum_merchnum_in_last_7_days
median_amount_by_cardnum_merchnum_in_last_30_days
total_amount_by_cardnum_merchnum_in_last_0_days
total_amount_by_cardnum_merchnum_in_last_14_days
total_amount_by_cardnum_merchnum_in_last_7_days
max_amount_by_cardnum_zip_in_last_30_days
mean_amount_by_cardnum_state_in_last_14_days
mean_amount_by_cardnum_zip_in_last_14_days
max_amount_by_cardnum_state_in_last_0_days
mean_amount_by_cardnum_merchnum_in_last_1_days
Amount
total_amount_by_Cardnum_in_last_0_days
Fraud

With our 15 final variables selected, we subset our original training, testing, and out-of-time data to include only those 15 fields along with the fraud label for each record. Combined, the three datasets contain 16 fields with 96,397 observations each.

Modeling Algorithms and Processes

In practice, it's always best to start the modeling process with a simple linear model. All subsequent non-linear models' performances can then be compared to the original linear model's baseline performance. We started by building a logistic regression model with our 15 selected predictors and default hyperparameter settings to obtain this baseline model.

Logistic Regression

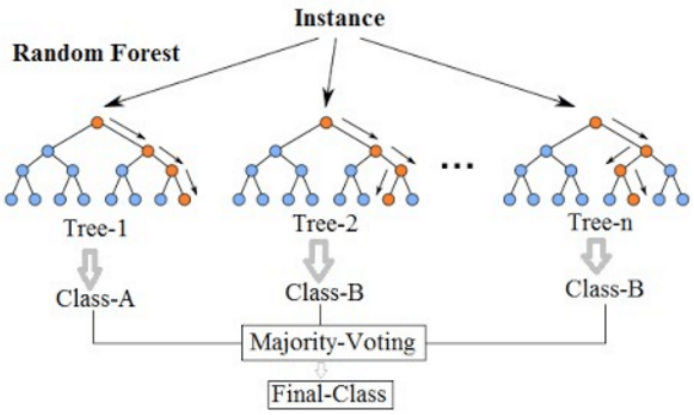
Logistic regression is the most commonly used linear classification model. It works by maximizing a likelihood function to estimate the coefficient parameters for each input variable in a sigmoid function which takes values between 0 and 1. Specifically, the logistic regression equation sets $P(Y=1 \mid X= x)$ equal to e raised to the power of each input value multiplied by each coefficient value divided by 1 + e raised to the same power. By seeing all the training record input values, logistic regression can estimate the coefficients that best allow the function to output high probabilities for actual class 1 records.

Next, we ran some non-linear models in an attempt to improve model performance on the test and out-of-time (OOT) data. Following are descriptions of the algorithms we trained.

Random Forest

A single decision tree models a data surface in discrete steps, and works by testing candidate cut points for each dimension in the model. At each candidate cutpoint, the impurity across the cut is stored, and so the first cut the tree makes is on the dimension that reduced impurity the most with its best candidate cutpoint. Then the tree continues to carve the remaining variable space, repeating the aforementioned process separately in each resulting data space after a cut is made.

A random forest in turn is an ensemble of strong decision trees. A tree's strength is primarily determined by its depth, and therefore its complexity. A strong tree can have many layers and partition the data space in complex and highly-fitted ways. A single strong tree is prone to overfitting the data by making cuts that are too specifically tailored to the training set, and therefore won't perform well in general on test data. Random forests both reduce the variability present in a single tree and ameliorate overfitting issues by, in classification problems, gathering "votes" from each tree in the ensemble for each test record. Each tree outputs which class it thinks is most likely for each test record, and the majority class vote from all trees gets assigned to that record.

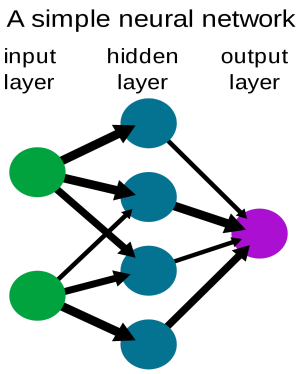


Some of the hyperparameters we adjusted when training random forest models include the number of trees in the ensemble, the maximum depth of each tree (fairly deep for random forests), the minimum number of samples required on either side of a partition for that split to occur in a given tree, and the criterion for evaluating the best split (either gini or entropy).

Neural Network

A neural network (neural net for short) is an algorithm designed to mimic the brain. Usually, neural nets consist of an input layer, a number of hidden layers greater than or equal to one, and an output layer, which in the case of a classification problem is the model's best guess at the record's class. Nodes in the hidden layer are analogous to neurons, and are sent signals from previous layers in the network that can be compared to impulses traveling along axons and synapses.

Each node performs a transformation or "activation" function on the linear combination of signals it receives from the previous layer, and outputs another signal to the next layer. The input signals are weighted, and the neural net trains by back-propagating errors and re-weighting the nodes. A training epoch constitutes one pass through the entire dataset. Eventually, each node will settle into a local optimum weight, and the model reaches a stable state.

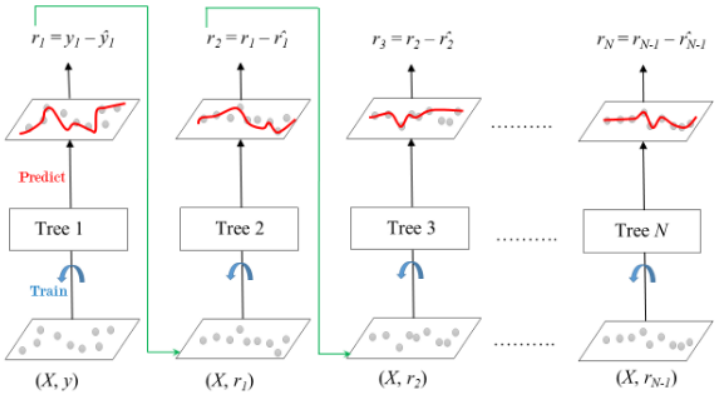


Some hyperparameters we adjusted when training neural nets include the number of hidden layers, the activation function type, the learning rate (which attempts to guide nodes into global optima over the course of training), the number of nodes per hidden layer, and the maximum number of training epochs.

Boosted Tree: Gradient Boosted Machine

Boosted trees, like random forest algorithms, use an ensemble of decision trees to formulate a final classification decision for each record. However, unlike random forests, boosted trees use far more trees, each with far less depth. Additionally, boosted trees hone in on errors made in previous trees and target those errors in the next tree. Because each tree is a weak learner, the algorithm is highly robust against overfitting.

Gradient boosting is a specific type of boosted tree algorithm that uses a gradient descent procedure to increasingly minimize the loss function with each new tree. Some hyperparameters that we tweaked for our GBM model were the number of trees used, the max depth of each tree, the loss function used, and the learning rate.

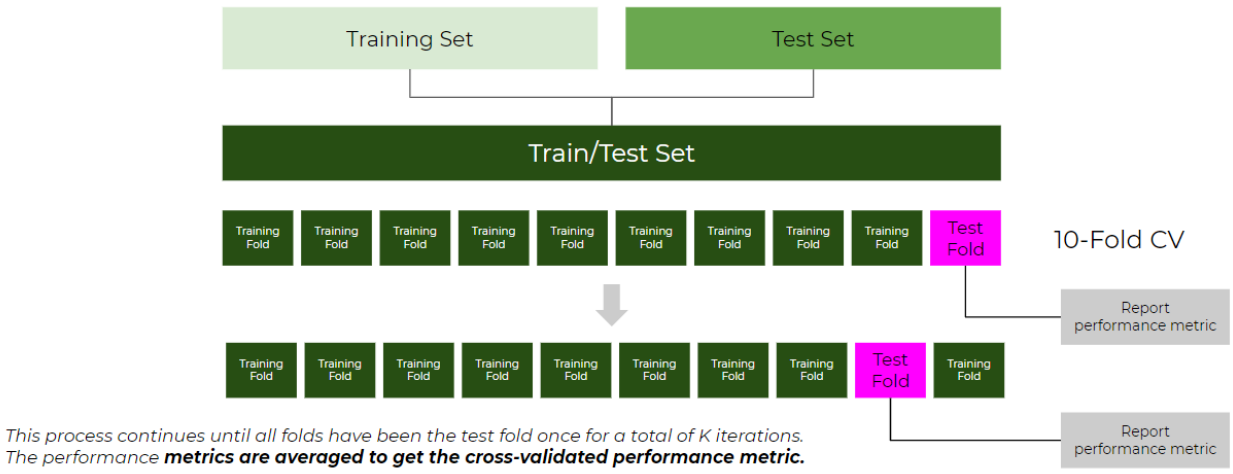


Model Evaluation

When comparing models built with different algorithms and hyperparameter settings, we needed a way to evaluate each model's performance using a metric related to the eventual use case for our model. If the evaluation metric doesn't match the use case, the chosen model may not actually perform the best on the task it's expected to perform out of all the candidate models.

With this in mind, we chose to evaluate models using the model Fraud Detection Rate at 3%. This metric calculates the percentage of the total fraud in the dataset that the model catches within the first 3% of records that the model tags as likely to be fraudulent. FDR @3% is calculated by ranking descending all records by the model's predicted probability that the record is fraudulent. Then, the first 3% of that list of sorted records are selected, and the total number of actual frauds caught within that 3% subset is divided by the total number of fraudulent records in the data. In our case, the data actually has 1,009 fraud records, so if a model catches 505 frauds in the top 3% of records ranked by its predicted fraud probability, its FDR @ 3% would be approximately .5, or 50%.

With so few fraud records, getting a stable estimate of fraud detection rate for a given model is difficult. This is due to the law of large numbers, which in this case suggests that with a small sample size of fraud records, it's less likely that the FDR will converge to its true value. The traditional mitigator for this issue is to use a technique called k-fold cross validation, shown in the diagram below:

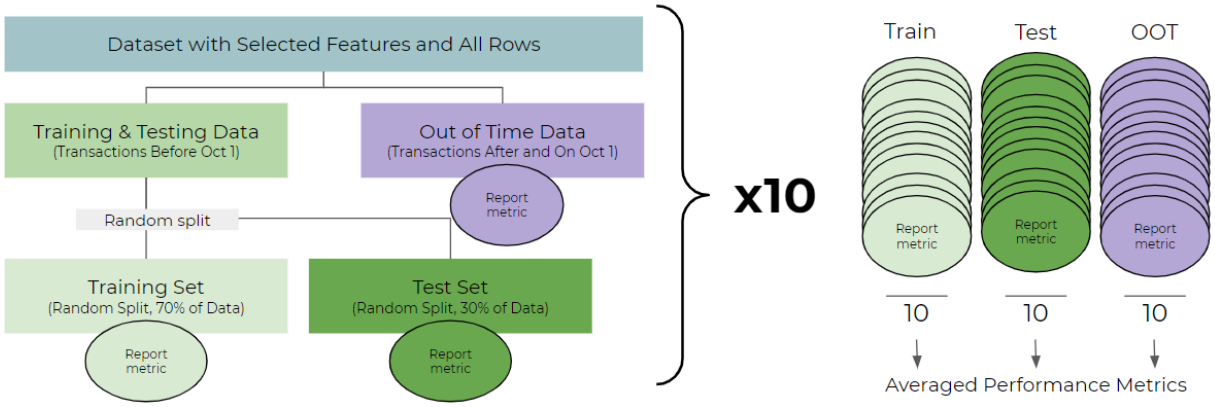


K-fold cross validation combines the training and testing data into one set, and then divides that data into K randomly shuffled folds. For classification problems, these folds are usually stratified, which means that each fold has approximately the same proportion of classes (or here, same

proportion of frauds) as the dataset at large. K-1 folds are used to train the model in each run, and then the model is evaluated on the remaining fold. A performance metric is calculated, and then the model iterates through the next run, rotating another fold into the test fold position.

However, in our case, we felt we didn't have enough fraud records for any given fold to produce a statistically significant evaluation metric. Decreasing the number of folds would increase the number of fraud records in each fold but reduce the number of metrics that we can average to get a final metric. Therefore, we decided to use an alternative evaluation method for averaging across multiple trials.

For our new method, in each trial, we began with a train/test set and out of time set. This split was made based on the date for each record; all records with a transaction date after October 1 were kept in the out of time set, and the rest of the transactions went to the train/test set. Then, for each trial, we randomly split the train/test set into a training and testing set using a random 70% subset for training and the remaining 30% for testing. We then trained each model on the training data, and evaluated its performance on all three datasets. For each trial, we reported the FDR @ 3% for each model on each of the three datasets. We conducted 10 trials and averaged the final performance across those 10 trials. At the end of the process we obtained a stable estimate of each model's performance on each of the three datasets. The process is outlined in the diagram below:



Results

Modeling Results

We initially built various models using the four different algorithms detailed above, and evaluated the results on each of these models' performance on the training, test, and OOT set. Due to computational limitations, we did not validate the results for these initial models across multiple trials. However, even without validation, it became clear that the tree based models -- random forest and gradient boosted trees -- performed far better than the logistic regression and neural network algorithms.

Model	Parameter				FDR at 3%		
Logistic Regression	Penalty	Max iteration	Number of varibles		TRAIN	TEST	OOT
1	None	2000	15		0.552	0.569	0.386
Random Forest	Estimators	Max depth	min sample split	criterion	TRAIN	TEST	OOT
1	50	100	2	gini	1.000	1.000	0.649
2	100	100	2	gini	1.000	1.000	0.552
3	150	100	2	gini	1.000	1.000	0.641
4	100	200	2	gini	1.000	1.000	0.618
5	50	500	2	gini	1.000	1.000	0.614
6	100	None	100	gini	1.000	1.000	0.560
7	100	200	2	entropy	1.000	1.000	0.591
Neural Networks	layers	Max iterations	learning rate	activation	TRAIN	TEST	OOT
1	10	20	constant	relu	0.526	0.547	0.185
2	5	30	constant	relu	0.486	0.498	0.216
3	15	40	constant	relu	0.398	0.431	0.232
4	5	40	adaptive	relu	0.371	0.400	0.193
5	5	40	constant	tanh	0.019	0.027	0.066
Gradient Boosting	Estimators	Max depth	loss	learning rate	TRAIN	TEST	OOT
1	500	3	deviance	0.1	0.992	0.978	0.533
2	750	4	deviance	0.1	0.977	0.960	0.506
3	500	3	exponential	0.1	1.000	1.000	0.564
4	500	3	deviance	0.01	0.865	0.865	0.637
5	1000	3	deviance	0.01	0.691	0.691	0.529

Modeling results and hyperparameter settings for 4 algorithms without validation. We used this initial test to select the two algorithms we thought would perform best to explore further.

Next, we explored additional models with varying hyperparameter settings for the two best performing algorithms -- boosted trees and random forest. However, in this round of modeling, we used our 10-trial performance validation described above in the model evaluation section to get an averaged performance metric for each model on each of the three datasets across 10 runs. The progression through this model performance evaluation process can be thought of as analogous to the filtering and wrapping process, in that we initially examined the results of all of our algorithms through non-validated results to save computational resources and time, and then used more computationally expensive validated results to find the small (but important) differences in performance between the best models. The results are as follows:

Model	trainingFDR	testingFDR	ootFDR
RandomForestClassifier(max_depth=2, n_estimators=50)	0.638	0.657	0.4770
RandomForestClassifier(max_depth=4)	0.837	0.836	0.6331
RandomForestClassifier(max_depth=6, n_estimators=150)	0.901	0.889	0.6565
RandomForestClassifier(max_depth=8)	0.985	0.946	0.6781
RandomForestClassifier(max_depth=10, n_estimators=50)	0.998	0.971	0.6751
RandomForestClassifier(min_samples_split=12)	1.000	0.981	0.6766
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_split=15, n_estimators=500)	**1.000**	**0.982**	**0.6781**
RandomForestClassifier(criterion='entropy', max_depth=15, n_estimators=500)	1.000	0.983	0.6770
RandomForestClassifier(criterion='entropy', max_depth=20, n_estimators=500)	1.000	0.982	0.6755
RandomForestClassifier(criterion='entropy', max_depth=25)	1.000	0.981	0.6710

Results for random forest algorithms with various hyperparameter settings on the train, test, and OOT set. Performance is evaluated as the FDR @ 3% averaged across 10 runs with random train/test splits in each run. The best model achieved an FDR at 3% of .6781 on the OOT data.

Model	trainingFDR	testingFDR	ootFDR
GradientBoostingClassifier(learning_rate=0.01, max_depth=6, n_estimators=500)	0.9871	0.9470	0.5815
GradientBoostingClassifier(learning_rate=0.01, max_depth=4, n_estimators=500)	0.9158	0.9005	0.5980
GradientBoostingClassifier(learning_rate=0.01, max_depth=4, n_estimators=700)	0.9416	0.9132	0.6017

Results for boosted tree algorithms with various hyperparameter settings on the train, test, and OOT set. Performance is evaluated as the FDR @ 3% averaged across 10 runs with random train/test splits in each run. The best model achieved an FDR at 3% of .6017 on the OOT data.

According to our validated performance metrics, the best performing model on the OOT data (which most closely matches how the model might be expected to perform on future data once it's deployed) is the random forest model with an entropy criterion, max depth of 10, minimum number of samples required for a split of 15, and 500 trees. Below are the fraud detection rates for this model at various percentages.

Training	# Records	# Goods	# Bads	Fraud Rate
	55206	54653	553	0.0100170

Bin Statistics						Cumulative Statistics						
Population Bin %	records	goods	Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	%Goods	%Bads (FDR)	KS	FPR
1	552	37	515	6.70%	93.30%	552	37	515	0.07%	93.13%	93.06	0.07
2	552	514	38	93.12%	6.88%	1104	551	553	1.01%	100.00%	98.99	1.00
3	552	552	0	100.00%	0.00%	1656	1103	553	2.02%	100.00%	97.98	1.99
4	552	552	0	100.00%	0.00%	2208	1655	553	3.03%	100.00%	96.97	2.99
5	552	552	0	100.00%	0.00%	2760	2207	553	4.04%	100.00%	95.96	3.99
6	552	552	0	100.00%	0.00%	3312	2759	553	5.05%	100.00%	94.95	4.99
7	552	552	0	100.00%	0.00%	3864	3311	553	6.06%	100.00%	93.94	5.99
8	552	552	0	100.00%	0.00%	4416	3863	553	7.07%	100.00%	92.93	6.99
9	552	552	0	100.00%	0.00%	4968	4415	553	8.08%	100.00%	91.92	7.98
10	552	552	0	100.00%	0.00%	5520	4967	553	9.09%	100.00%	90.91	8.98
11	552	552	0	100.00%	0.00%	6072	5519	553	10.10%	100.00%	89.90	9.98
12	552	552	0	100.00%	0.00%	6624	6071	553	11.11%	100.00%	88.89	10.98
13	552	552	0	100.00%	0.00%	7176	6623	553	12.12%	100.00%	87.88	11.98
14	552	552	0	100.00%	0.00%	7728	7175	553	13.13%	100.00%	86.87	12.97
15	552	552	0	100.00%	0.00%	8280	7727	553	14.14%	100.00%	85.86	13.97
16	552	552	0	100.00%	0.00%	8832	8279	553	15.15%	100.00%	84.85	14.97
17	552	552	0	100.00%	0.00%	9384	8831	553	16.16%	100.00%	83.84	15.97
18	552	552	0	100.00%	0.00%	9936	9383	553	17.17%	100.00%	82.83	16.97
19	552	552	0	100.00%	0.00%	10488	9935	553	18.18%	100.00%	81.82	17.97
20	552	552	0	100.00%	0.00%	11040	10487	553	19.19%	100.00%	80.81	18.96
21	552	552	0	100.00%	0.00%	11592	11039	553	20.20%	100.00%	79.80	19.96
22	552	552	0	100.00%	0.00%	12144	11591	553	21.21%	100.00%	78.79	20.96
23	552	552	0	100.00%	0.00%	12696	12143	553	22.22%	100.00%	77.78	21.96
24	552	552	0	100.00%	0.00%	13248	12695	553	23.23%	100.00%	76.77	22.96
25	552	552	0	100.00%	0.00%	13800	13247	553	24.24%	100.00%	75.76	23.95
26	552	552	0	100.00%	0.00%	14352	13799	553	25.25%	100.00%	74.75	24.95
27	552	552	0	100.00%	0.00%	14904	14351	553	26.26%	100.00%	73.74	25.95
28	552	552	0	100.00%	0.00%	15456	14903	553	27.27%	100.00%	72.73	26.95
29	552	552	0	100.00%	0.00%	16008	15455	553	28.28%	100.00%	71.72	27.95
30	552	552	0	100.00%	0.00%	16560	16007	553	29.29%	100.00%	70.71	28.95

Performance statistics for our winning random forest model as reported on the training data.

Testing	# Records	# Goods	# Bads	Fraud Rate
	23661	23424	237	0.010016

Bin Statistics						Cumulative Statistics						
Population Bin %	records	goods	Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	%Goods	%Bads (FDR)	KS	FPR
1	236	28	208	11.86%	88.14%	236	28	208	0.12%	87.76%	87.64	0.13
2	236	215	21	91.10%	8.90%	472	243	229	1.04%	96.62%	95.59	1.06
3	236	234	2	99.15%	0.85%	708	477	231	2.04%	97.47%	95.43	2.06
4	236	233	3	98.73%	1.27%	944	710	234	3.03%	98.73%	95.70	3.03
5	236	236	0	100.00%	0.00%	1180	946	234	4.04%	98.73%	94.70	4.04
6	236	236	0	100.00%	0.00%	1416	1182	234	5.05%	98.73%	93.69	5.05
7	236	236	0	100.00%	0.00%	1652	1418	234	6.05%	98.73%	92.68	6.06
8	236	236	0	100.00%	0.00%	1888	1654	234	7.06%	98.73%	91.67	7.07
9	236	236	0	100.00%	0.00%	2124	1890	234	8.07%	98.73%	90.67	8.08
10	236	236	0	100.00%	0.00%	2360	2126	234	9.08%	98.73%	89.66	9.09
11	236	236	0	100.00%	0.00%	2596	2362	234	10.08%	98.73%	88.65	10.09
12	236	236	0	100.00%	0.00%	2832	2598	234	11.09%	98.73%	87.64	11.10
13	236	236	0	100.00%	0.00%	3068	2834	234	12.10%	98.73%	86.64	12.11
14	236	236	0	100.00%	0.00%	3304	3070	234	13.11%	98.73%	85.63	13.12
15	236	236	0	100.00%	0.00%	3540	3306	234	14.11%	98.73%	84.62	14.13
16	236	236	0	100.00%	0.00%	3776	3542	234	15.12%	98.73%	83.61	15.14
17	236	236	0	100.00%	0.00%	4012	3778	234	16.13%	98.73%	82.61	16.15
18	236	236	0	100.00%	0.00%	4248	4014	234	17.14%	98.73%	81.60	17.15
19	236	236	0	100.00%	0.00%	4484	4250	234	18.14%	98.73%	80.59	18.16
20	236	236	0	100.00%	0.00%	4720	4486	234	19.15%	98.73%	79.58	19.17
21	236	236	0	100.00%	0.00%	4956	4722	234	20.16%	98.73%	78.58	20.18
22	236	236	0	100.00%	0.00%	5192	4958	234	21.17%	98.73%	77.57	21.19
23	236	236	0	100.00%	0.00%	5428	5194	234	22.17%	98.73%	76.56	22.20
24	236	235	1	99.58%	0.42%	5664	5429	235	23.18%	99.16%	75.98	23.10
25	236	236	0	100.00%	0.00%	5900	5665	235	24.18%	99.16%	74.97	24.11
26	236	236	0	100.00%	0.00%	6136	5901	235	25.19%	99.16%	73.96	25.11
27	236	236	0	100.00%	0.00%	6372	6137	235	26.20%	99.16%	72.96	26.11
28	236	236	0	100.00%	0.00%	6608	6373	235	27.21%	99.16%	71.95	27.12
29	236	236	0	100.00%	0.00%	6844	6609	235	28.21%	99.16%	70.94	28.12
30	236	236	0	100.00%	0.00%	7080	6845	235	29.22%	99.16%	69.93	29.13

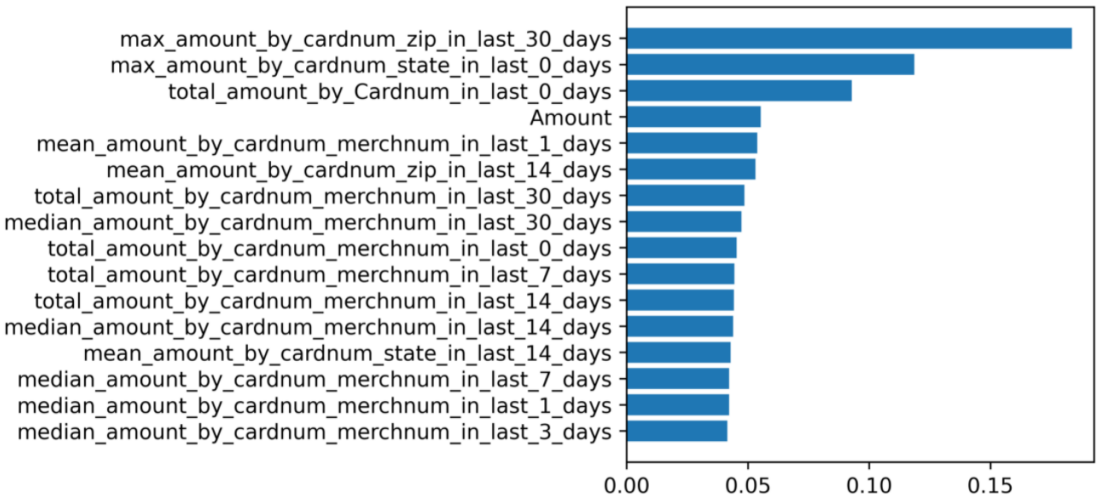
Performance statistics for our winning random forest model as reported on the test data.

OOT	# Records	# Goods	# Bads	Fraud Rate
	17530	17261	269	0.015345

Bin Statistics						Cumulative Statistics						
Population Bin %	records	goods	Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	%Goods	%Bads (FDR)	KS	FPR
1	175	59	116	33.71%	66.29%	175	59	116	0.34%	43.12%	42.78	0.51
2	175	116	59	66.29%	33.71%	350	175	175	1.01%	65.06%	64.04	1.00
3	175	169	6	96.57%	3.43%	525	344	181	1.99%	67.29%	65.29	1.90
4	175	172	3	98.29%	1.71%	700	516	184	2.99%	68.40%	65.41	2.80
5	175	173	2	98.86%	1.14%	875	689	186	3.99%	69.14%	65.15	3.70
6	175	174	1	99.43%	0.57%	1050	863	187	5.00%	69.52%	64.52	4.61
7	175	171	4	97.71%	2.29%	1225	1034	191	5.99%	71.00%	65.01	5.41
8	175	175	0	100.00%	0.00%	1400	1209	191	7.00%	71.00%	64.00	6.33
9	175	174	1	99.43%	0.57%	1575	1383	192	8.01%	71.38%	63.36	7.20
10	175	172	3	98.29%	1.71%	1750	1555	195	9.01%	72.49%	63.48	7.97
11	175	173	2	98.86%	1.14%	1925	1728	197	10.01%	73.23%	63.22	8.77
12	175	173	2	98.86%	1.14%	2100	1901	199	11.01%	73.98%	62.96	9.55
13	175	171	4	97.71%	2.29%	2275	2072	203	12.00%	75.46%	63.46	10.21
14	175	173	2	98.86%	1.14%	2450	2245	205	13.01%	76.21%	63.20	10.95
15	175	169	6	96.57%	3.43%	2625	2414	211	13.99%	78.44%	64.45	11.44
16	175	172	3	98.29%	1.71%	2800	2586	214	14.98%	79.55%	64.57	12.08
17	175	175	0	100.00%	0.00%	2975	2761	214	16.00%	79.55%	63.56	12.90
18	175	175	0	100.00%	0.00%	3150	2936	214	17.01%	79.55%	62.54	13.72
19	175	173	2	98.86%	1.14%	3325	3109	216	18.01%	80.30%	62.29	14.39
20	175	175	0	100.00%	0.00%	3500	3284	216	19.03%	80.30%	61.27	15.20
21	175	174	1	99.43%	0.57%	3675	3458	217	20.03%	80.67%	60.64	15.94
22	175	175	0	100.00%	0.00%	3850	3633	217	21.05%	80.67%	59.62	16.74
23	175	172	3	98.29%	1.71%	4025	3805	220	22.04%	81.78%	59.74	17.30
24	175	170	5	97.14%	2.86%	4200	3975	225	23.03%	83.64%	60.61	17.67
25	175	175	0	100.00%	0.00%	4375	4150	225	24.04%	83.64%	59.60	18.44
26	175	172	3	98.29%	1.71%	4550	4322	228	25.04%	84.76%	59.72	18.96
27	175	175	0	100.00%	0.00%	4725	4497	228	26.05%	84.76%	58.71	19.72
28	175	174	1	99.43%	0.57%	4900	4671	229	27.06%	85.13%	58.07	20.40
29	175	175	0	100.00%	0.00%	5075	4846	229	28.07%	85.13%	57.06	21.16
30	175	175	0	100.00%	0.00%	5250	5021	229	29.09%	85.13%	56.04	21.93

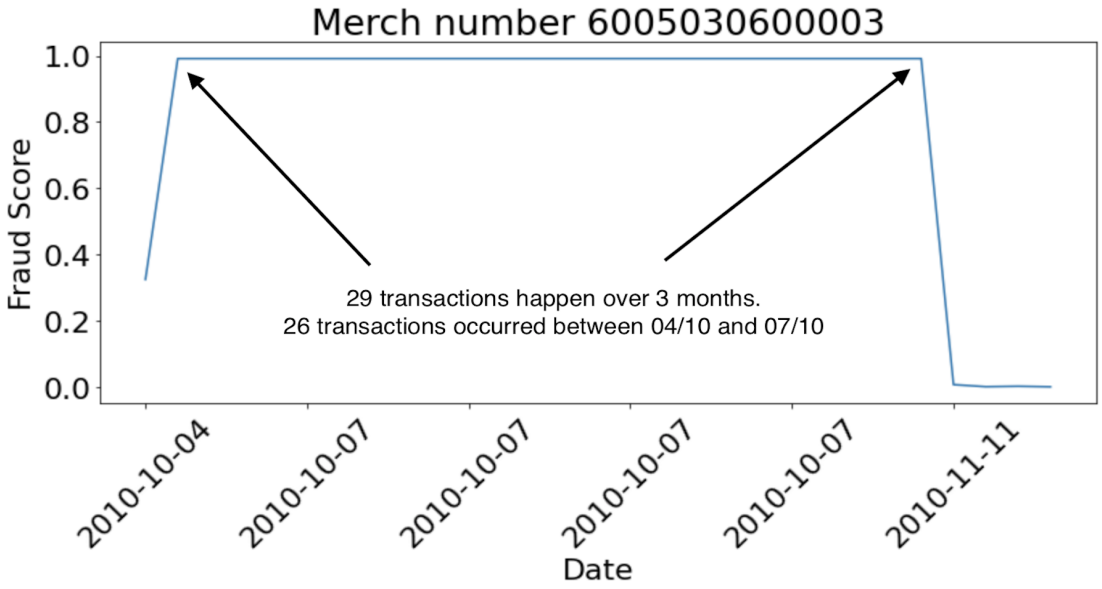
Performance statistics for our winning random forest model as reported on the OOT data.

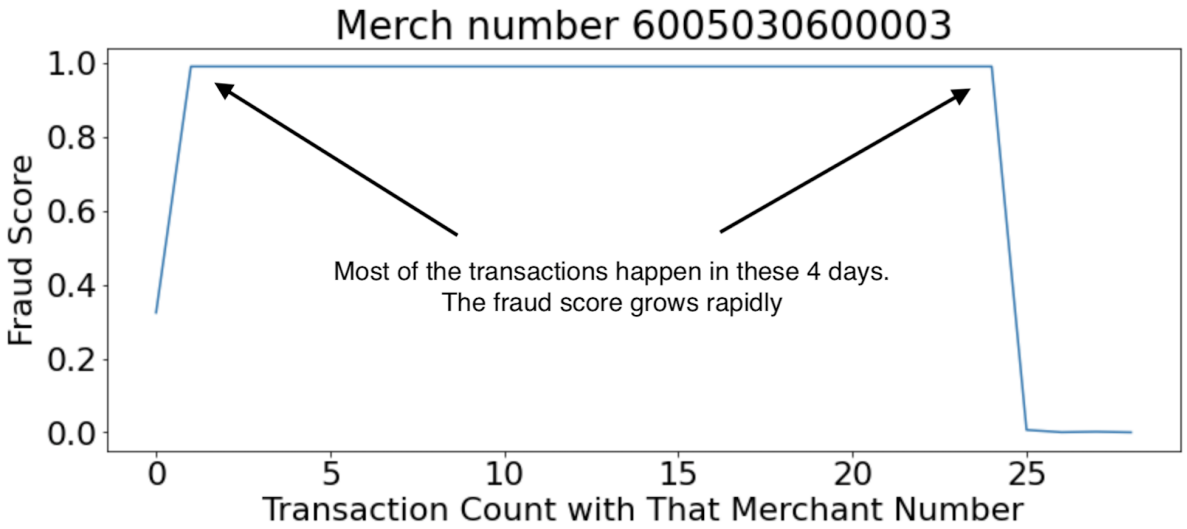
The most important variables for our final model are shown in the chart below. Each variable's bar length represents its relative importance to predicting fraud in the final model.

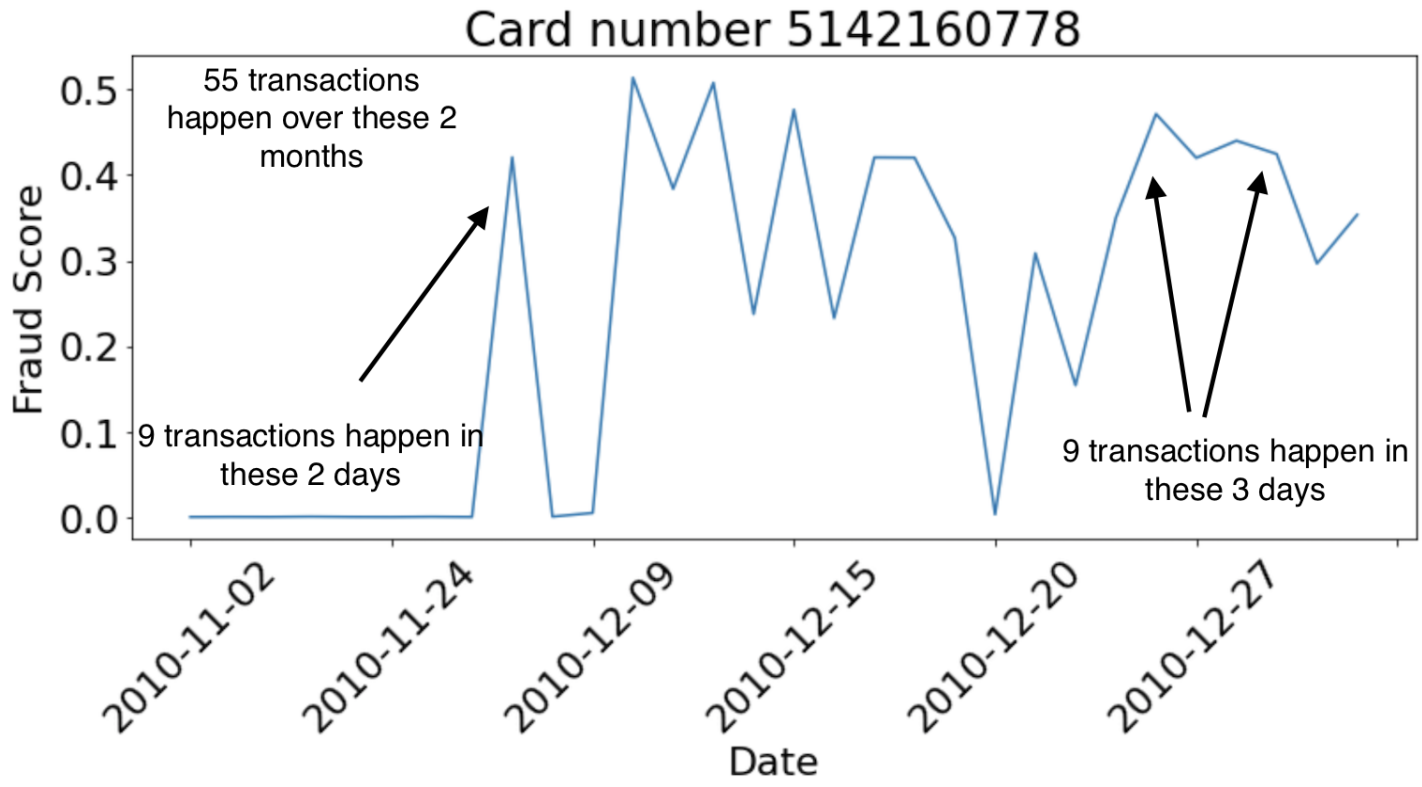


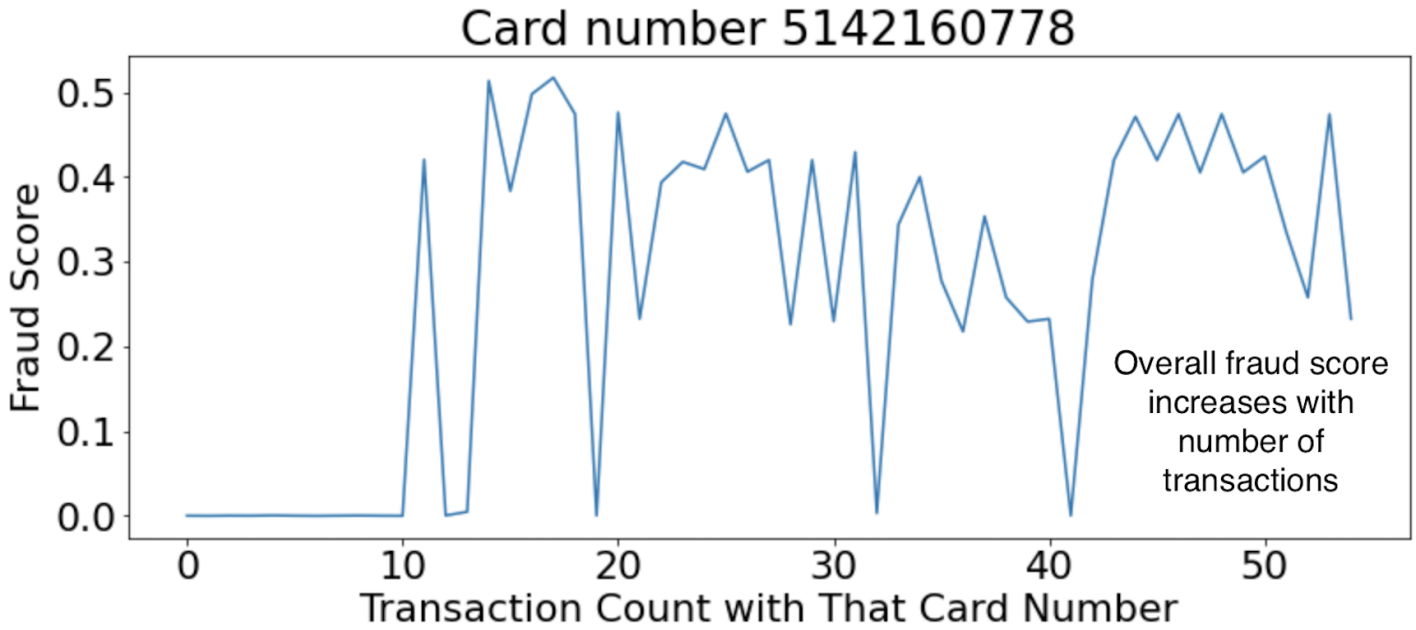
Relative feature importance of all variables in the winning model. The most important variables for predicting which records are fraudulent are the maximum amount spent on a card in a given zip code in the last month, and the maximum amount spent on a card in the past day in a state.

The following charts show how and why the model is effective in detecting fraudulent transactions. A typical signal of fraud is a sudden jump in transaction frequency on a given card or at a given merchant. For the first few transactions on a given card or merchant, the model doesn't tag them as fraudulent (otherwise, every time a transaction occurred at a new merchant or card number in a given time period, the model would tag those transitions as fraudulent, which would lead to many false positives). However, as the transaction frequency continues to tally up over a short period of time relative to non-fraudulent transactional behavior, to model takes notice, and the fraud score increases substantially for those transactions. The charts below demonstrate the relationship between the transactional behavior in the data and the fraud score the model assigns to those transactions.



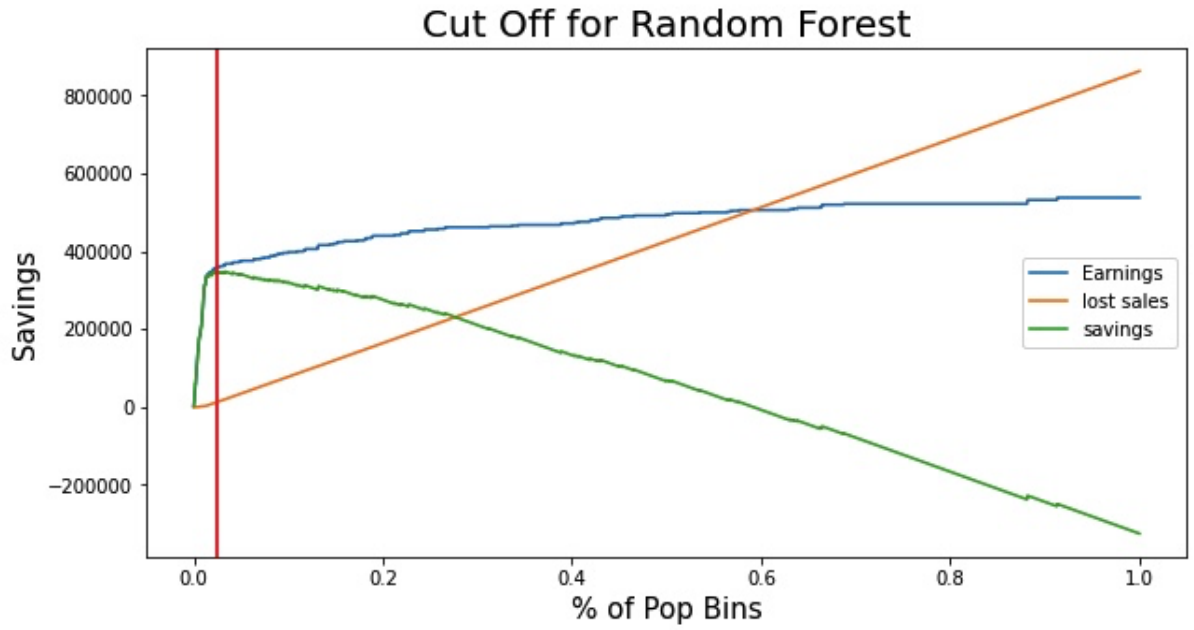



Card number 5142160778
55 transactions happen over these 2 months
9 transactions happen in these 2 days
9 transactions happen in these 3 days
Fraud Score
0.0
0.1
0.2
0.3
0.4
0.5
2010-11-02
2010-11-24
2010-12-09
2010-12-15
2010-12-20
2010-12-27
Date


Card number 5142160778
Overall fraud score increases with number of transactions
Fraud Score
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50
Transaction Count with That Card Number

Financial Results

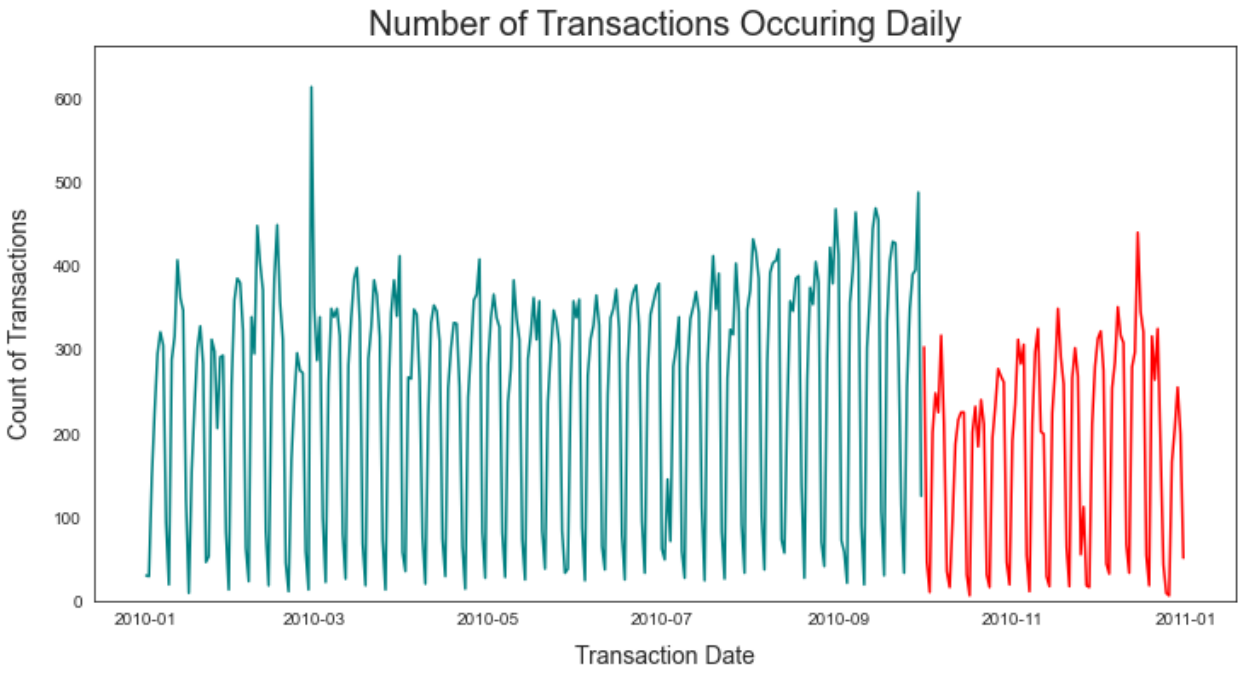
In order to maximize the cost savings we created a plot that shows our expected earnings versus our expected losses using our random forest model. This was done with the assumptions that every fraud found would save us $2,000 and every false positive would cost us $50. We plotted these assumptions against the percentage of the population to find the ideal Fraud Detection Rate. This ideal Fraud Detection Rate is approximately 2.39% and is illustrated by the red vertical line in the graph below.



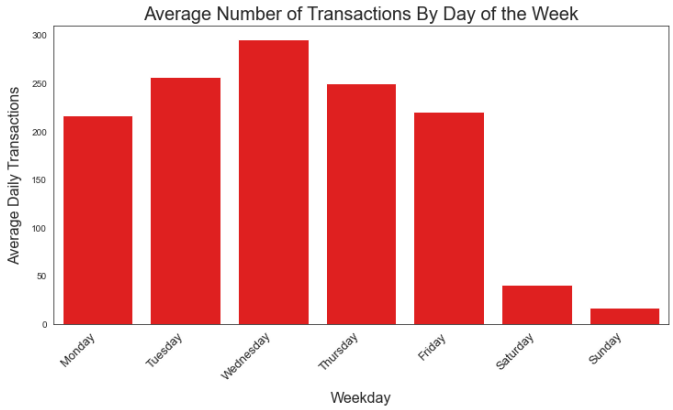
Based off of our best performing model we thus recommend a 2.39% cutoff for the Fraud Detection Rate. This saved $348,050 based on the out of time data and so we estimate that it would save $1,392,200 annually.

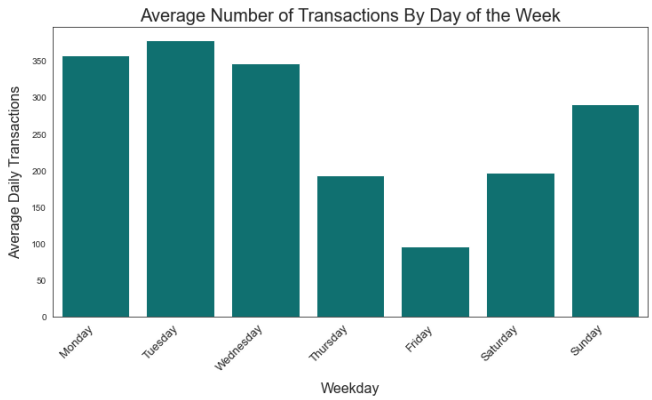
Post-Hoc Analysis and Possible Improvements

After evaluating our models' performance both analytically and financially, we considered potential improvements for our model that could be made in the future before deployment. The most obvious issue with our current models is that the OOT performance is so much worse than performance on the training and testing data. To examine a potential cause for this problem, we performed a post-hoc exploratory data analysis comparing the distributions of data in the train/test set and the OOT set. We found that on many of the original data dimensions, the distributions are quite different for the two sets. In all visualizations below, the red line represents data in the OOT set, and the teal line represents data in the train/test set.

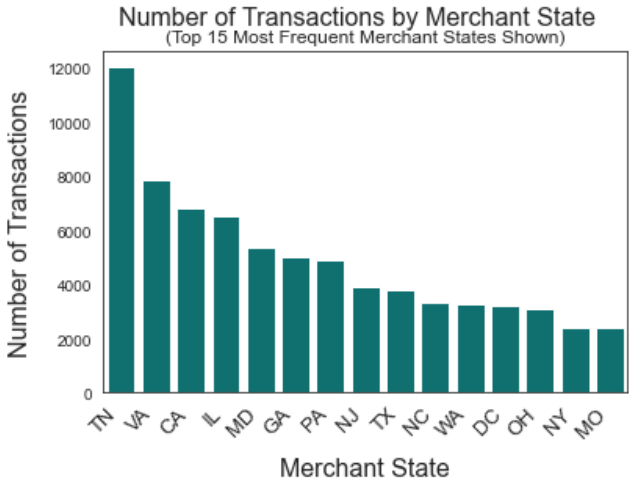


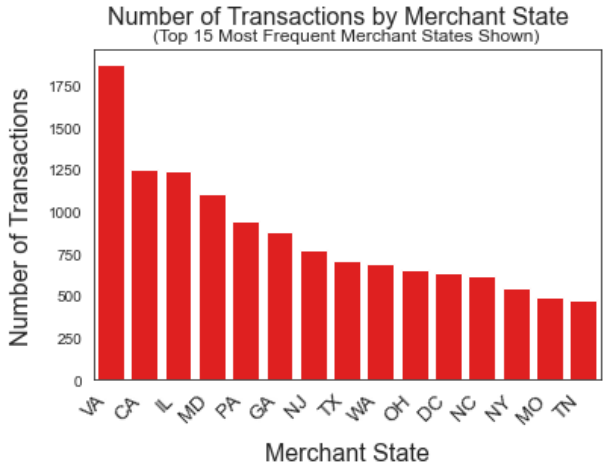
The quantity of daily transactions is significantly lower in the OOT data on average than in the train/test data.





The days of the week distribution is fairly different, with transactions on the weekends decreasing substantially in the OOT set relative to the train/test set.





The number of transactions in the state of Tennessee (TN) decrease significantly relative to other states in the OOT data vs. the train/test data.

As evidenced above, there are clear differences in the data distributions on at least some of the data dimensions for the two sets of data. We posit that this happens because the behavioral patterns that generated this data are different for the OOT set than for the train/test set. The date on which we split the OOT data (October 1) also happens to coincide with the end of the fiscal year the day before (September 30). Since these transactions come from corporate accounts, it's reasonable to say that as budgets reset, the spending patterns on these accounts change substantially on the same date that we split our data.

Because of this unfortunate coincidence, the model may be training on slightly different behavioral patterns than the ones it's asked to recognize in the OOT set. The model performs well on the test set because the test set is comprised of data generated by similar behavioral patterns as the data the model is being trained on, while performance on the OOT set dips substantially as the patterns the model was trained to recognize become muddier and less clearly associated with fraud.

We propose four solutions to ameliorate this perplexing issue. They are as follows:

A. Obtain multiple years of data and train two different models. In this method, one model is responsible for learning about patterns and making predictions in the first 9 months, and the second model is responsible for the last three months of the calendar year. We'd need multiple years of data to accomplish this, since we can't train a model effectively on just three months of data with so few frauds. The drawback to this approach is that the data from further past years would become increasingly less relevant to newer fraud patterns.

B. After choosing an algorithm and carefully reviewing all code, train the model on all available data, including OOT. In doing so, we can expose the model to the patterns that arise in the last three months of data. However, this is a risky strategy because we won't be able to evaluate the model's performance before deployment.

C. Decrease the size of the OOT dataset to 1 or 2 months. Train the model on some of the current OOT records while still leaving a few to test the model on. To increase the stability of FDR estimates at a given percentage on these few fraud records, we can use cross validation or bootstrapping to sample additional records.

D. Weight input records differently depending on their transaction date. If we want the model to be a little bit better at predicting on the OOT data, we can sacrifice some training experience on the pre-October 1 data for additional training on the post Oct 1 data. By more heavily weighting input variables for records that occur later in the year, we might get slightly worse performance on the first 9 months of data, but we might improve net performance across the entire year.

Conclusion

In this project, we built a model to predict credit card transaction fraud. As our data was labelled, this was a supervised learning task. Our approach incorporated most of the end-to-end data science framework, including data cleaning, feature engineering, feature selection, modelling and evaluation.

We began with exploratory data analysis in the form of a data quality report, delivering insights such as summary statistics on every variable, split up into 3 fields: Date, Numerical and Categorical. Following this, we transitioned onto our feature engineering stage where we created over 300 additional variables. We then scaled the variables to ensure they contributed equally to the models. For the feature selection, we applied a filter based on the univariate KS score and fraud detection rate (at 3%) scores to first reduce our candidate variables to 80 and then a Stepwise Forward Selection wrapper to finally reduce our variables to 15.

In the modelling stage, we executed a range of algorithms, starting with a linear model: logistic regression as our baseline. We then implemented some non-linear models including Random Forests, Neural Networks and Gradient Boosted Trees to see which would perform the best or more specifically, to see which model best predicted credit transaction fraud.

As mentioned above in the results section, our best performing model was a Random Forest algorithm (with a FDR at 3% of 0.6781 on the Out of Time Data). Deployment of this model at the optimal cutpoint -- meaning the optimal top percentage of the model's transactions ranked descending by probability of fraud to actually act on as if they are fraudulent -- is projected to save $1,392,200 annually.

With more time and resources, there are several ways in which we could improve our current model's performance. First, our dataset is extremely imbalanced (which is expected given the nature of our task is fraud prediction), with the fraudulent credit card transactions accounting for about 1.1% of the total credit card transactions in our dataset. Given this is what we are trying to predict, future efforts might be improved by collecting more fraudulent transactions data. In this case, increasing the number of fraudulent transactions data by a factor of 5 for example, would still result in a very imbalanced dataset but it would give us a lot more fraudulent transaction labels to input into our Machine Learning models and thus might give them a better chance of performing better.

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.

Appendix

DQR

Data Description

The data contained in the Credit Transactions dataset is information from 96,573 credit card transactions from January 01, 2010 to December 31, 2010. This data primarily included labels of fraud and not-fraud to help us generate predictions of fraudulent transactions.

Data Summary

Numeric Fields:

Column Name	count	mean	Standard Deviation	min	Median	max	% populated	Unique Values	# of Zeroes
Recnum	96753.0	48377.000000	27930.329635	1.00	48377.00	96753.00	100.0	96753	0
Amount	96753.0	427.885677	10006.140302	0.01	137.98	3102045.53	100.0	34909	0

Categorical Fields:

column_name	Number of Values	Unique Number of Values	Top Value	% Populated
Cardnum	96753	1645	5142148452	100.000000
Merchnum	93378	13091	930090121224	99.965117
Merch description	96753	13126	GSA-FSS-ADV	100.000000
Merch state	95558	227	TN	99.987649
Merch zip	96753	4568	38118.0	100.000000
Transtype	96753	4	P	100.000000
Fraud	96753	2	0	100.000000

Date Fields:

column	unique_num_of_dates	most_common_date	first_date	last_date	% Filled
Date	365	2010-02-28	2010-01-01	2010-12-31	100.0

Numeric Field Exploration:

Field 1:

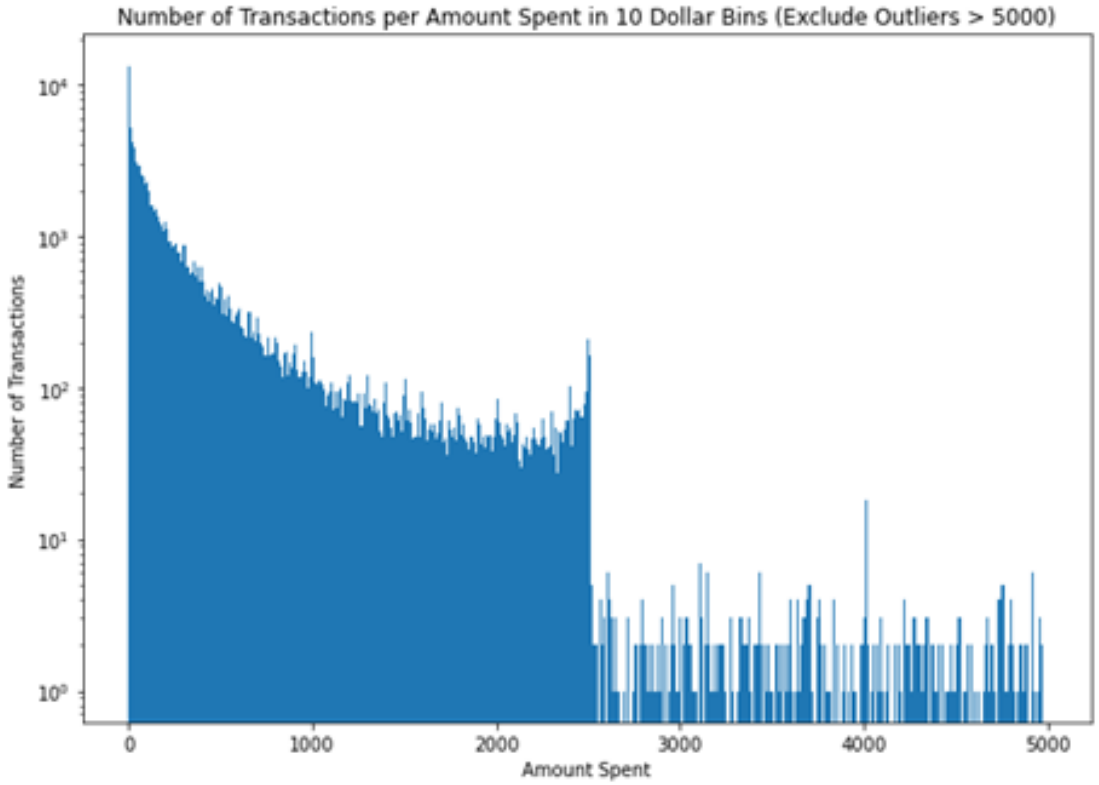
Name: Recnum

Description: Tim eorder value of the data.

Field 2:

Name: Amount

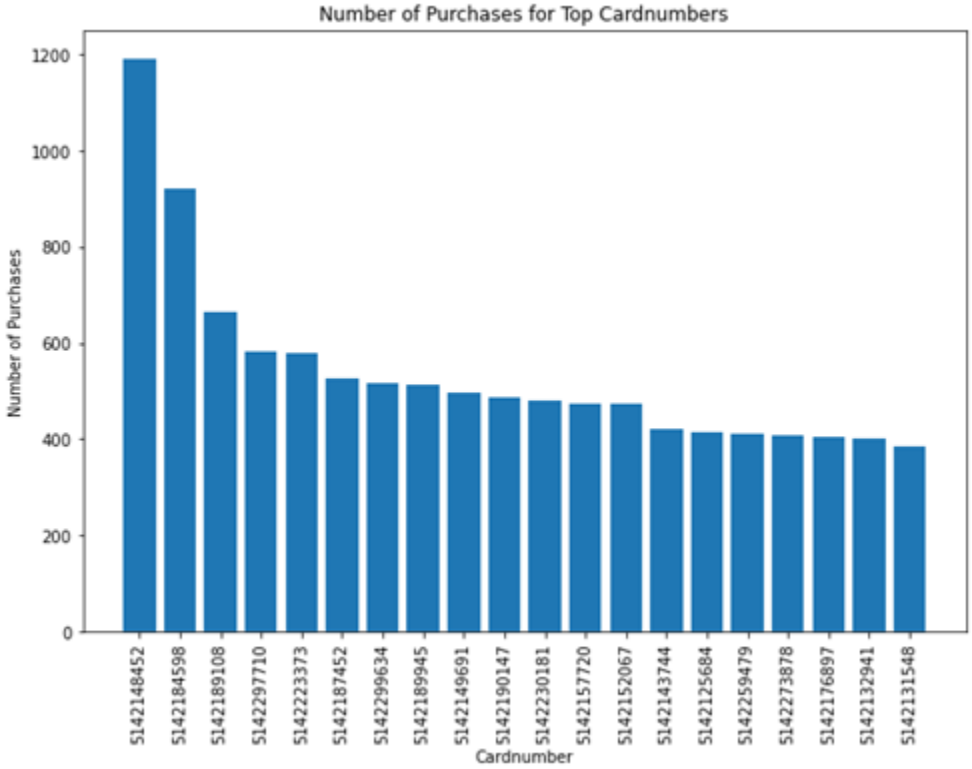
Description: Amount spent on the credit card purchase. Note the logarithmic scale on the y axis.



Field 3:

Name: CardNum

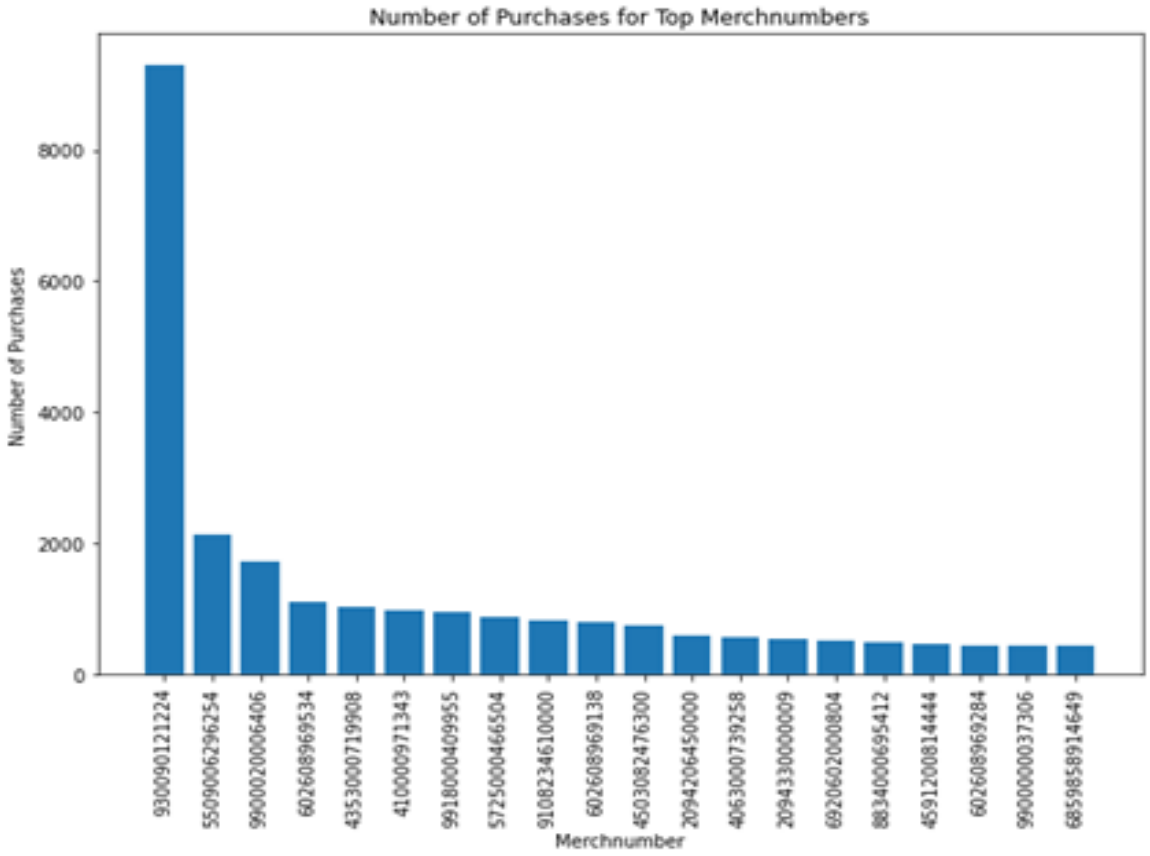
Description: CreditCard Number



Field 4:

Name: Merchnum

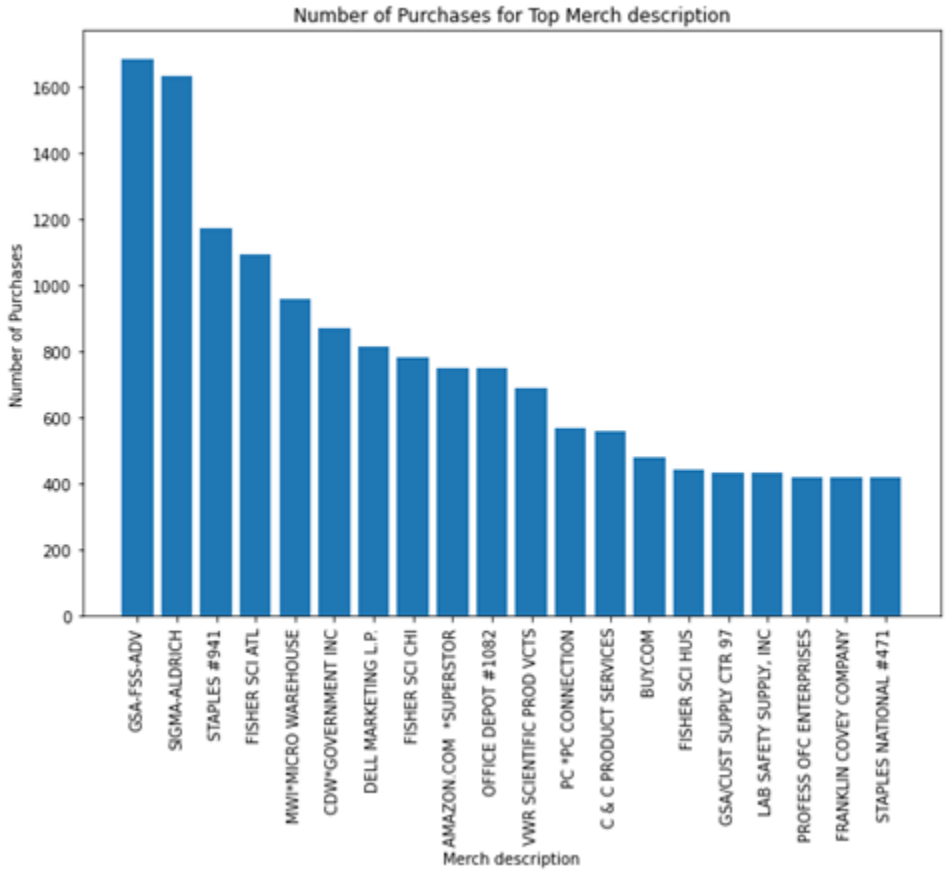
Description: The identification number of the merchant associated with the transaction.



Field 5:

Name: Merch description

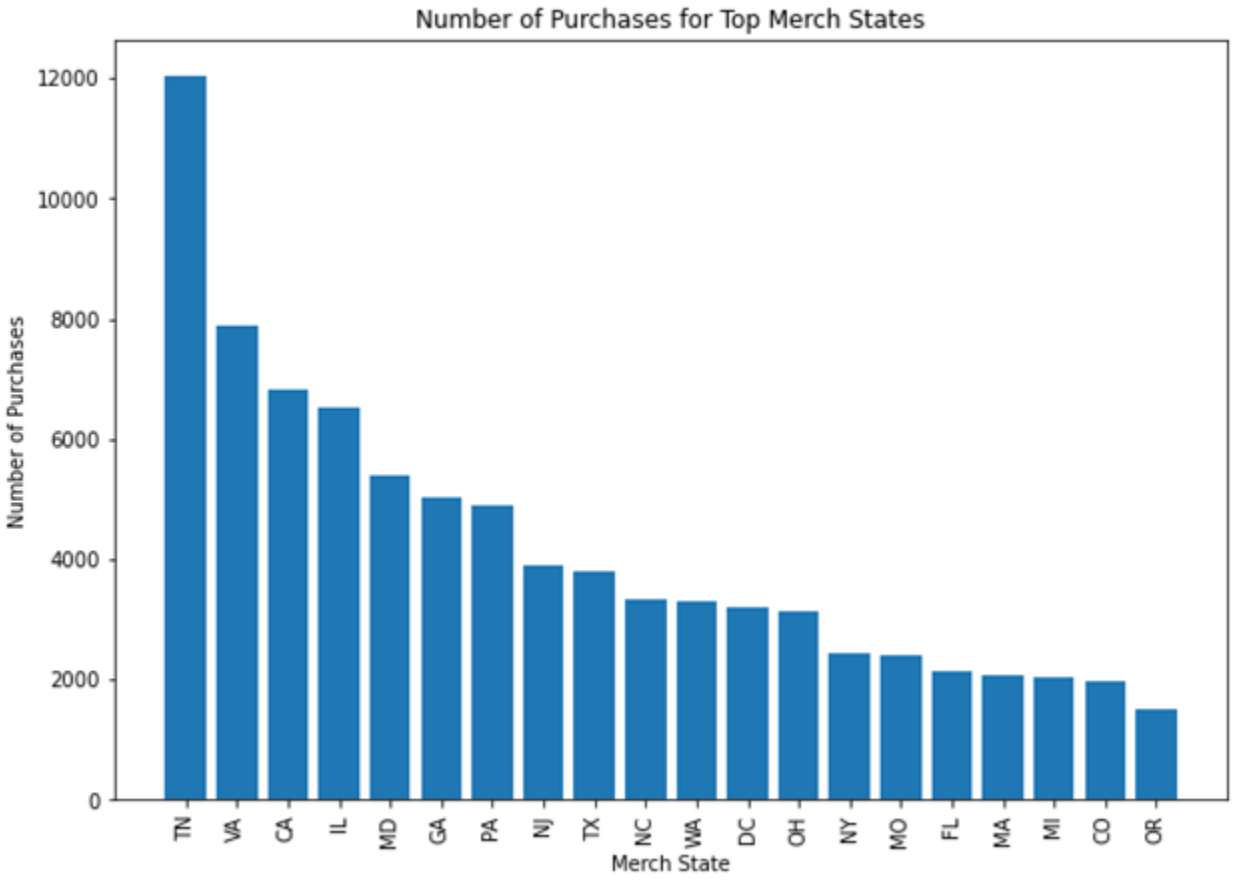
Description: Description of the merchant related to this purchase.



Field 6:

Name: Merch States

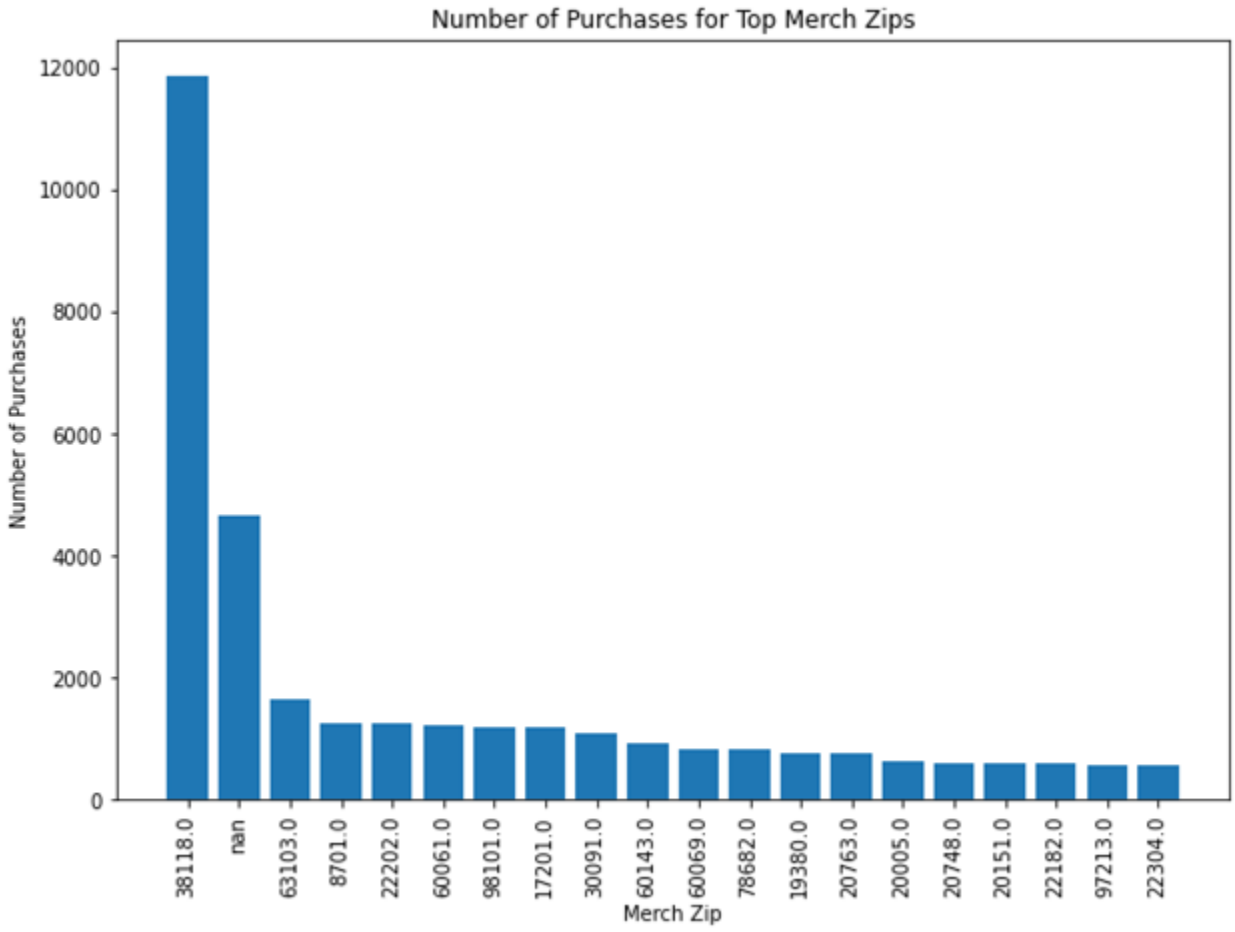
Description: State the merchant is located in.



Field 7:

Name: Merch Zip

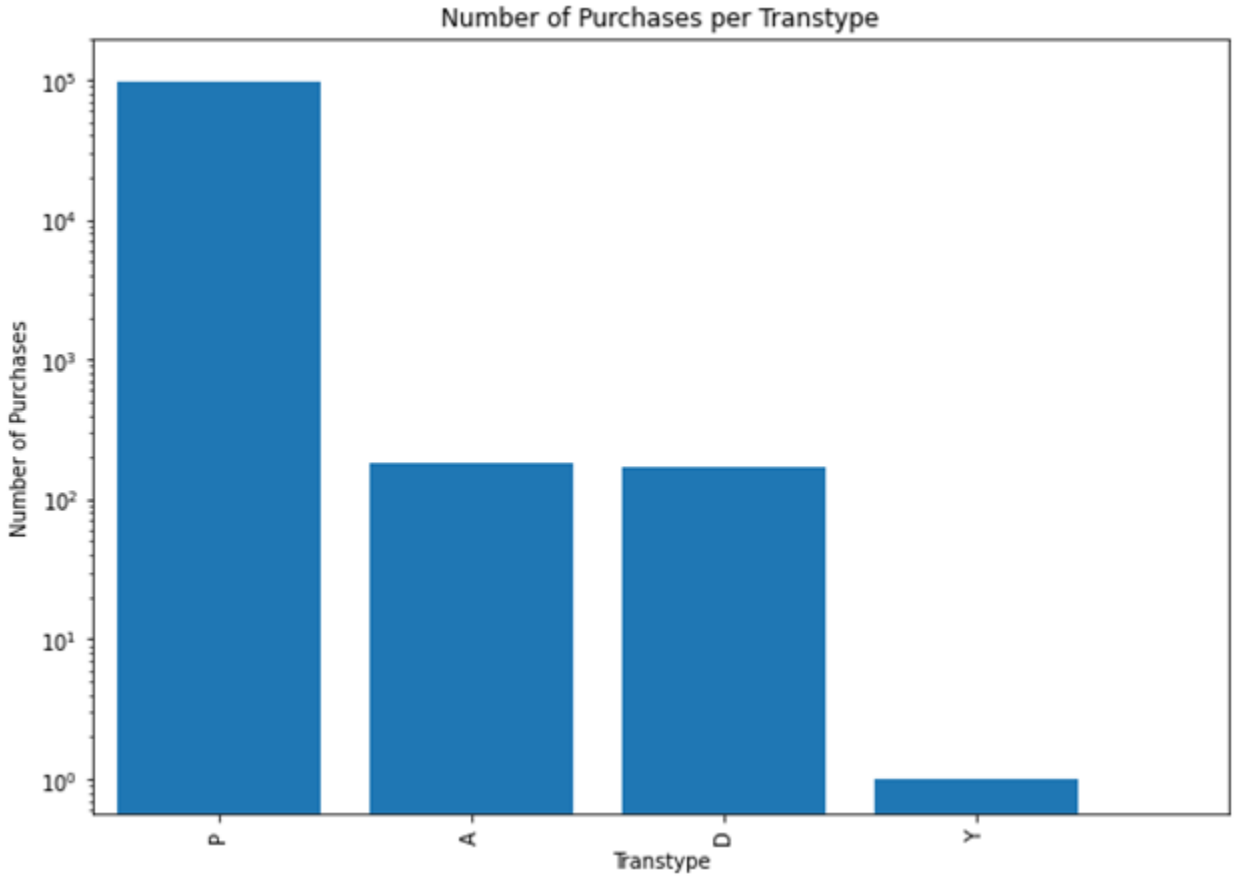
Description: Zip code the merchant is located in.



Field 8:

Name: Transtype

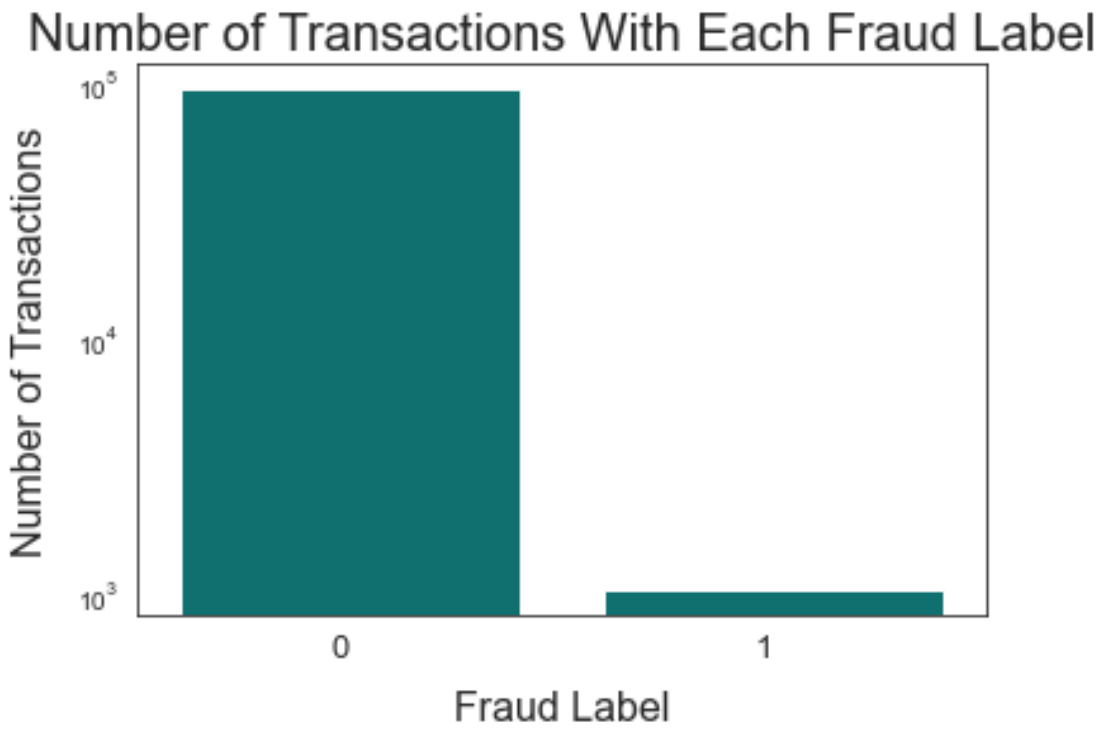
Description: Type of transaction type. Note the logarithmic scale on the y axis.



Field 9:

Name: Fraud Label

Description: Whether a purchase was a fraud or not. 0 indicates no fraud while 1 indicates Fraud. Note the logarithmic scale on the y axis.



Field 10:

Name: Date

Description: Date of purchase.