Supervised Fraud Detection Model on Transactions Data

DSO 562 - Fraud Analytics
Spring 2022

Team Members: Lukas Frei, Jentrie Grissom, Jiahui Tian & Jenna Pakkanen
Team # 107

Date: 05/05/2022

Table of Contents

Executive Summary

There are many types of credit card fraud. The financial technology company Brex broadly divides them into card-present and card-not-present fraudulent activities. In card-present fraud, the fraudster takes physical control of the card by, for example, stealing it or intercepting it in the mail before attempting to use it in-person. Card-not-present fraud, on the other hand, occurs when the fraudster knows the card details without needing physical access to the card before attempting to make purchases online or by phone.

Due to the COVID-19 pandemic, physical credit card transactions have decreased while online transactions, due to business requirements and health reasons, have increased. However, even before the pandemic, the share of transactions that do not require a card to be physically present has been increasing with the rising popularity of online shopping. This is reflected in fraudulent credit card transaction volumes as a 2018 study conducted by the Federal Reserve illustrated a decline of card-present fraud in the U.S.A. from $3.68 billion in 2015 to $2.91 billion in 2016. The card-not-present fraud volume in the same time period increased from $3.4 billion to $4.57 billion. We aim to build an effective machine learning algorithm to predict the nature of a credit card transaction (being fraudulent or not) to help reduce the losses incurred due to fraud.

The data evaluated in this report contains actual credit card purchases from a US government organization, originally published on a website for a book on Forensic Analytics by Mark Nigrini. The 1,059 fraudulent transactions were added by a domain expert based on their experience with credit card fraud. The data frame has 10 fields and 97,753 records with transactions between January 1st, 2006, and December 31st, 2006, as expressed by the field 'Date.'

In this report, we started with a brief data quality report to give the audience an overview of the dataset, emphasizing selected field distributions. Then, we described the data cleaning process on exclusions, outliers, and missing values. Next, we explained our feature engineering process that created 747 variables and applied the filter and wrapper to select the 20 most significant variables. Based on the top 20 variables, we explored a linear algorithm (logistic regression) and a few non-linear algorithms, including Decision Tree, Random Forest, Boosted Trees, Neural Network. After fine tuning hyperparameters, we selected a tuned Neural Network algorithm as our final model. The financial institution is expected to save $1,231,800 per annum if our algorithm is implemented.

In the end, our final model achieved a fraud detection rate of 71.90% on training data, 71.27% on test data, and 60.89% on out-of-time data by examining only 3% of the data among all the records. In the future, with the availability of more data, we will explore more features, hyperparameters and down sampling and/or up sampling imbalance data on more powerful machines to continue to improve our model performance.

Description of the Data

1. High-Level Data Description

The data evaluated in this report contains actual credit card transaction information from a US government organization from January 1, 2006 to December 31, 2006.

The dataset includes 96,753 transaction records and ten fields that correspond to the individual transactions, for example, card number, merchant number, and amount. Additionally, the data includes a binary fraud label indicating whether the application has or is connected to compromised elements. A record with a fraud label of 1 is considered a fraudulent application, while a record with a fraud label of 0 is deemed normal or not fraudulent. A description of each field in the dataset and its summary statistics can be found in figures 1, 2, and 3. For a full description of all of the data fields in the dataset, please refer to Appendix A for the Data Quality Report.

2. Summary Tables

Field Name	% Populated	# of Unique Values	Minimum	Maximum	Mean	Standard Deviation
amount	100%	34,909	0.01	3,102,045.50	427.90	10,006.10

Numeric Fields Table

Field Name	% Populated	# of Unique Values	Most Common Values
recnum	100%	96,753	N/A
cardnum	100%	1,645	5142148452
date	100%	365	2006-02-28
merchnum	96.5%	13,092	930090121224
merch description	100%	13,126	GSA-FSS-ADV
merch state	98.7%	228	TN
merch zip	95.2%	4,568	38118
transtype	100%	4	P

fraud	100%	2	0

Categorical Fields Table

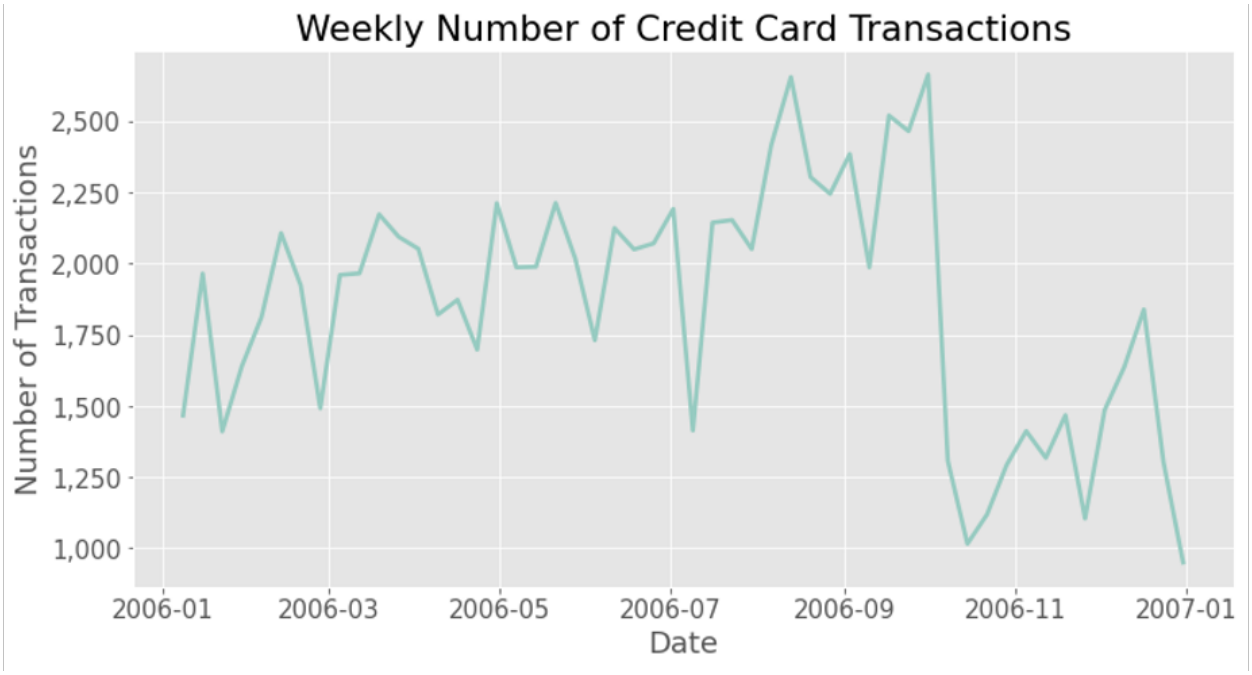
Field Name	Description
amount	The dollar amount of the transaction
recnum	A unique integer assigned to each record 1 through 96,75
cardnum	The card number associated with each transaction
date	The date the transaction took place
merchnum	A 12-digit number assigned to each merchant
merch description	A text description including unique details about each merchant
merch state	The abbreviation of the state where each merchant is based
merch zip	The ZIP code of the merchant's location at the time of the transaction
transtype	A code denoting the type of each transaction (types = P, A, D and Y)
fraud	A binary label assigned to each transaction (0 = good transaction, 1 = bad transaction)

Field Description Table

3. Selected Field Distributions

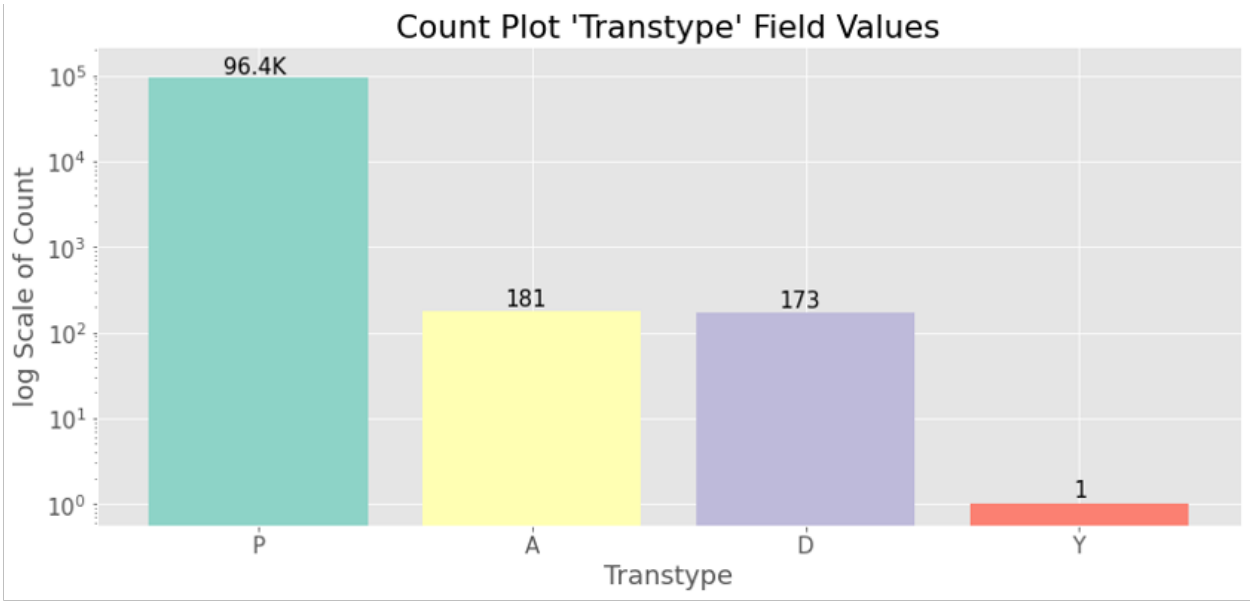
Field: 'date'

Looking at the number of transactions over time as expressed by the field 'date,' aggregated by calendar week, one can observe a relatively normal distribution of transaction activity for the first eight months of the year. The distribution of weekly transactions below shows that transaction activity tends to decrease at the beginning of each fiscal quarter, evidenced by the significant dips in January, April, July and October. While that pattern remains constant throughout the entire year, the decrease in transaction activity at the beginning of October, the fourth quarter, is significantly sharper and the number of weekly transactions reaches its lowest point.



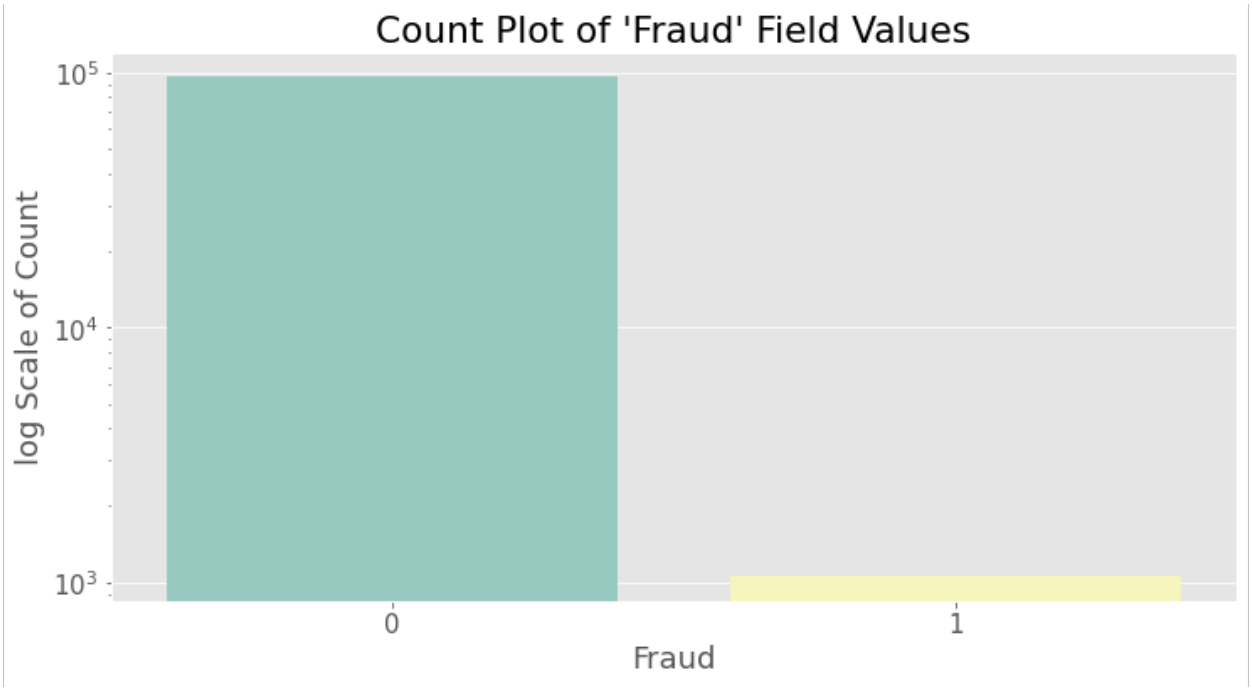
Field: 'transtype'

The transtype field assigns each record a single-character value "A", "D", "P" or "Y", denoting the type of transaction associated with the record. For our analysis, we were only interested in evaluating purchase transactions, type "P" records, to detect potential fraud, so in order to isolate the relevant data, we removed the remaining transaction types from the dataset. After excluding the 355 "A", "D" and "Y" records, we were left with 96,398 "P" type records for analysis.



Field: 'fraud'

The 'fraud' field is a binary variable indicating whether the application was fraudulent. A record with a "1" indicates that the transaction was fraudulent and a record with a "0" indicates that the record is not fraudulent. Of the total 96,753 records 1,059 were flagged as fraudulent.



Fraud Label	Number of Transactions
0	95,694
1	1,059

Data Cleaning

The data cleaning process for this project involved filtering transactions and removing outliers, and dealing with missing values.

Outliers

There was a single outlier in the dataset, evident by an anomalous transaction amount. While the median of the Amount field was $137.98, this outlier referred to a transaction of $3,102,046. After removing the outlier, there are 96,752 records in the dataset.

Filtering

The dataset contained transactions of various types, recorded as ‘P’, ‘A’, ‘D’, ‘Y’ in order of occurrence in the dataset. As we are only interested in purchases, we filtered out 355 transactions not labeled as ‘P’.

Missing values

During the data quality report, we noticed that there are missing values in Merchnum, Merch state and Merch zip fields.

Mapping-Based Filling

- Merchnum: We grouped records with their Merch Description and used the mode of each group to fill in the missing values based on the assumption that the same merchant description represents the same merchant number.
- Merch State: We grouped records with their merchant zip; however, due to the missing values in the Merch Zip field, this is not sufficient to impute all missing values in the Merch State field. To deal with some of the remaining missing values in the Merchant State field, we used census data to map zip codes to states. Afterwards, we further used the model per merchant number and merchant description groups to fill in the missing values.
- Merch Zip: Similarly, we used the mode of the non-null values per Merch num and Merch description group to fill in missing values.

These steps significantly decreased missing values in the dataset.

Remaining missing values

We filled remaining missing values with ‘Unknown’ before moving onto the candidate variable creation stage of our project. In this stage, all the missing values were properly filled.

Candidate Variables

Before a supervised model can be built, we must create a large set of candidate variables by selecting the data features that we believe have a relationship to the likelihood that a transaction is fraudulent. We created these variables from the original data fields, focusing on the speed and regularity of transactions.

We first developed 12 new entities by concatenating data fields from the original dataset:

Entities:

Entity Name	Description
Cardnum	Card
Merchnum	Merchant
Card_merchn	Card at this merchant
Card_zip	Card in this zip
Card_state	Card in this state
Merch_state	Merchant in this state
Merch_zip	Merchant in this zip
Card_merch_zip	Card at this merchant in this zip
Card_merch_state	Card at this merchant in this state
Card_state_zip	Card in this state in this zip
Merch_state_zip	Merchant in this state in this zip
Merch_description	Merchant description

Days Since Candidate Variables

It is common that fraudsters operate in surges, filing multiple applications in a short period of time, regularly using the same personal identifying information. To capture this activity, we created this set of numerical variables that indicate the number of days since the last application with a given data field has been seen. Visually, the days since equation looks like this:

Days since	*{'Cardnum', 'Merchnum', 'Card_merch', 'Card_zip', 'Card_state', 'Merch_state', 'Merch_zip', 'Card_merch_zip', 'Card_merch_state', 'Card_state_zip', 'Merch_state_zip', 'Merch_description'}*	**last seen**

For these variables, a lower calculated value, meaning fewer days since a particular variable has been seen on transactions, indicates a higher likelihood of fraudulent activity.

Velocity Candidate Variables

The velocity candidate variables capture the rate of the number of applications at that entity over the past 0, 1, 3, 7, 14 and 30 days. Because fraudsters are often active in measurable periods of time, we created these velocity variables to identify the speed at which particular elements appear in applications during given time periods. These variables are numeric and allow us to identify the surges in which fraudsters are active. Visually, the velocity equation looks like this:

# of records with the same	*{'Cardnum', 'Merchnum', 'Card_merchn', 'Card_zip', 'Card_state', 'Merch_state', 'Merch_zip', 'Card_merch_zip', 'Card_merch_state', 'Card_state_zip', 'Merch_state_zip', 'Merch_description'}*	**over the last *{0,1,3,7,14,30}* days**

We measured each of the original candidate variables over periods of 0, 1, 3, 7, 14 and 30 days. Here, higher velocity, meaning a particular element appears frequently, indicates a higher likelihood of fraudulent activity.

Relative Velocity Candidate Variables

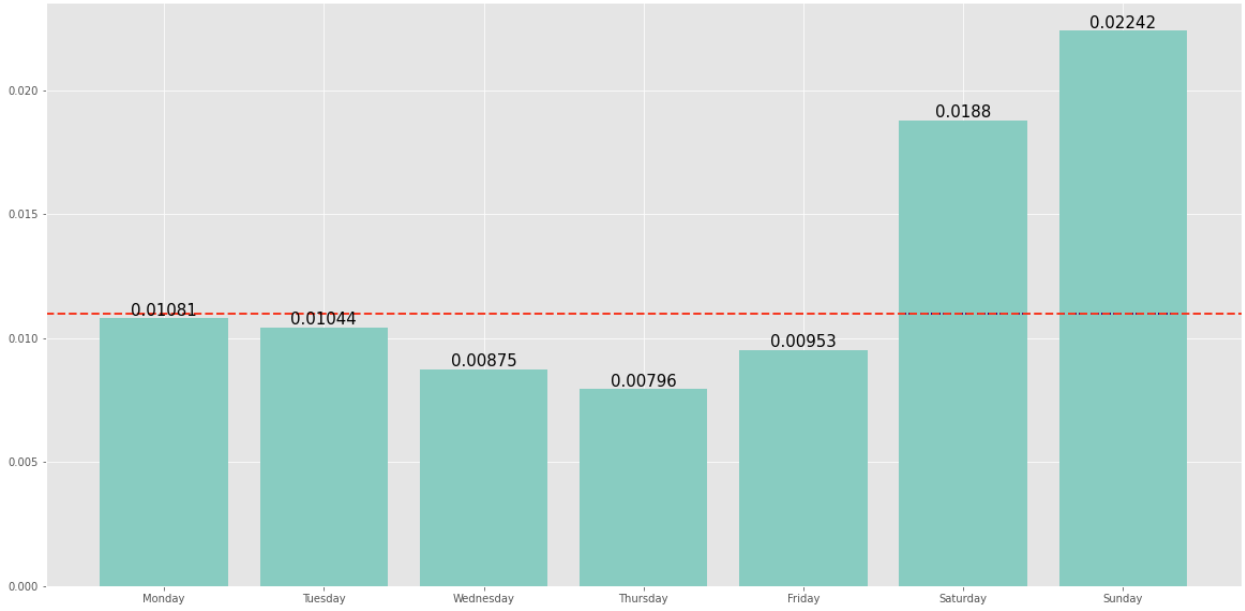
The relative velocity candidate variables are a set of numeric variables that indicate the number of transactions at the entity in a short window of time compared to the number of transactions at that entity over a longer window of time. Because fraudsters are typically active in bursts, we are interested in determining if a specific variable in the dataset appears on more application records during a given time period compared to what is normal. Visually, the relative velocity equation looks like this:

$$Relative\ Velocity = \frac{\#of\ \ applications\ with\ entity\ in\ the\ recent\ past}{\#\ of\ applications\ with\ the\ same\ entity\ in\ the\ past\ \{1,3,7,14,30\}\ days}$$

We applied this equation to our set of initial candidate variables, using 0 days and 1 day as our numerator or "recent past." For our longer window of time, the denominator of the equation, we used 1 day, 3 days, 7 days, 14 days, and 30 days. Like the velocity variables, a higher relative velocity, meaning a particular element appears more frequent than what is normal, indicates a higher likelihood of fraudulent activity.

Risk Table Candidate Variable

The day-of-the-week effect relates to the observation of returns that vary across days of the week in a persistent way. Because this could be another factor in detecting fraudulent transactions, we created a candidate variable to monitor the likelihood of fraudulent activity occurring on each day of the week. To avoid dimensionality expansion and overfitting, we performed target encoding (risk tables) with smoothing to transform categorical values into numerical values.



Amount Variables

Often times, the dollar amount of a transaction can be indicative of fraudulent activity. For example, a merchant who typically spends $200 per day suddenly spending $100,000 on one transaction would likely signal unusual behavior. Because fraudulent transactions tend to have higher amounts compared to innocuous transactions, we created amount candidate variables to

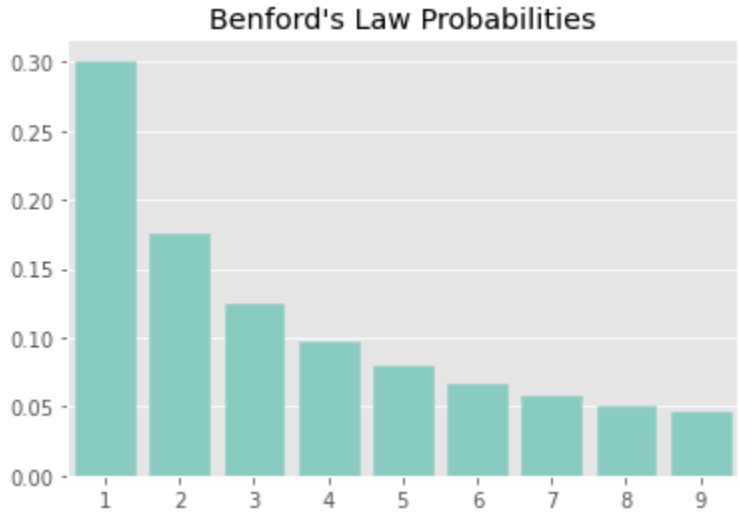
capture the statistics of the amount of each transaction in the dataset for the categorical data fields over the past 0,1, 3, 7, 14 and 30 days. Statistics included in our variable creation are average, maximum, median, total, actual/average, actual/maximum, actual/median and actual/total. Visually, the amount equation looks like this:

{Average Maximum Median Total Actual/average Actual/maximum Actual/median Actual/total}	Amount by/at this	*{'Cardnum', 'Merchnum', 'Card_merchn', 'Card_zip', 'Card_state', 'Merch_state', 'Merch_zip', 'Card_merch_zip', 'Card_merch_state', 'Card_state_zip', 'Merch_state_zip', 'Merch_description'}*	Over the past	*{0, 1, 3, 7, 14, 30}* days

The idea behind these variables is that some transactions may only slightly modify one of the attributes, in order to bypass fraud detection.

Benford's Law Variables

Benford's law is the non-intuitive fact that the first digit of many measurements is not uniformly distributed, and the first digit "1" appears about 30% of the time. Benford's law states that in naturally occurring sets of numbers, the distribution of the leading digit, *d*, is given by $P(d) = \log_{10}(1 + 1/d)$, where the likelihood of the leading digit d has a higher probability of being a lower digit than a higher digit. The Benford's law distribution is as follows:



Incorporating this theory into our fraud model is particularly useful in capturing unexpected distributions. For example, when a fraudster generates fraudulent transactions, they do not know about Benford's law, so the transaction amounts are uniformly distributed random numbers. With this type of candidate variable, we can look at the amount distributions for each cardholder and merchant to see if the amount distributions substantially violate Benford's law. The goal with this type of variable is to measure unusualness, quantifying how different the first digit distribution is from the Benford's law distribution.

Candidate Variable Creation Summary:

Variable Type	Variable Count
Days Since	12
Velocity	72
Relative Velocity	72
Day of Week Risk	1
Amount	576
Benford's Law	2
Counts by Entity	12
Total Number of Candidate Variables	**747**

Feature Selection Process

After creating candidate variables, the dimensionality of the dataset increased significantly. Adding the candidate variables made the dataset sparse, increased the computational power required to process it, and reduced the explainability of results. Therefore, we applied filter and wrapper techniques to reduce dimensionality and identify the variables with the highest predictive relevance.

Filter

Filtering is a fast and straightforward method to examine the correlation between each independent variable and the dependent variable. No machine learning algorithms are required for this step.

All independent candidate variables were considered in the filter stage, along with the dependent variable "Fraud." However, the last two months and the first two weeks of records were excluded in this stage. November and December 2006 records were held out as the out-of-time (OOT) data and later used to test the model performance when creating the models. For the first two weeks, the lack of historical data impedes fraud detection; therefore, it is reasonable to remove the first two weeks of data.

The Kolmogorov-Smirnov (KS) approach was applied as the filter. The KS approach plots non-fraudulent transactions and fraudulent transactions for each variable and measures the degree of separation between the two. In general, the more separate the two curves are, the more relevant the variable is as it has a greater capacity to distinguish between the non-fraudulent and fraudulent transactions. The univariate KS score was calculated for each individual variable, and all variables were ranked by their KS scores in descending order. As a result of the KS filter, the top 80 variables were retained for further analysis, while the remaining variables were filtered out.

Wrapper

In the wrapper stage, a machine learning algorithm was trained using all the features resulting from the filter stage. Then, we monitored the model performance (as measured by the fraud detection rate) to find a smaller set of variables that are highly correlated with the fraud label.

A forward selection wrapper was used. More specifically, we selected a LGBClassifier algorithm. We started with no variables selected in our initial model. Then, we iteratively added the variable that achieved the best overall model performance (lowest fraud detection rate in our case) when added to the existing variables at each step. This helped remove correlations among

independent variables and made sure the best variable was selected given the existing variables in the current model at each iteration. This process was repeated until the top 20 variables with the highest multivariate importance were generated. The 20 variables were then used to build preliminary machine learning models.

The following table includes the top 20 final variables in rank order of multivariate importance with their univariate KS score:

Variables	**KS Score**
Card_merch_total_7	0.6819
Card_state_max_3	0.6483
Merchnum_max_0	0.6028
Card_zip_max_3	0.6414
Cardnum_total_7	0.6002
Merch_state_max_3	0.5769
Merchnum_max_3	0.5692
Merch_state_max_0	0.6070
Merchnum_max_1	0.5066
Card_zip_total_7	0.6768
Merch_zip_max_1	0.5822
Cardnum_max_0	0.5853
Card_avg_1	0.5720
Cardnum_avg_0	0.5702
Merch_state_max_1	0.5908
Merch_zip_max_0	0.5959
Merch_zip_max_3	0.5692
Merch_description_max_3	0.5894

Merch_description_max_0	0.6093
Merch_description_max_1	0.5977

After applying the filter and wrapper, we selected the 20 variables that are most relevant to the fraud label prediction. These 20 variables will be used in the next stage to explore different model algorithms and fine tune hyperparameters that best fit the dataset to achieve a higher fraud detection rate. The 5 most powerful variables are Card_merch_total_7, Card_state_max_3, Merchnum_max_0, Card_zip_max_3 and Cardnum_total_7.

Model Algorithms

The dataset is split into training, test, and OOT data. Application records from 2006-11-01 to 2006-12-31 are held out as OOT data to evaluate the model performances, whereas transaction records from 2006-01-01 to 2006-10-31 are randomly split as training and test dataset. The training data and test data account for 70% and 30% of all the applications from 2006-01-01 to 2006-10-31 respectively. The randomized allocation between training and test datasets ensures the generalization of the model and reduces the risk of overfitting. The data for the last two months in 2006 is not used to build models, but can test the effectiveness of models when predicting unseen data.

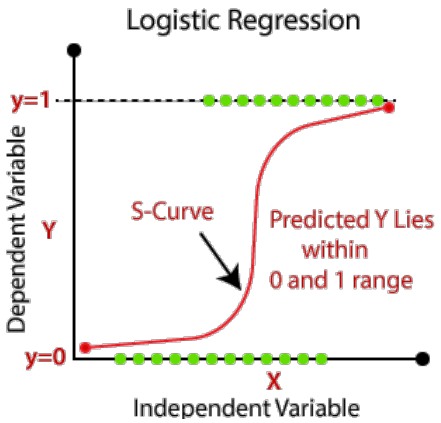
We manually performed a cross-validation with 10 iterations on each run and evaluated the model performances based on average fraud detection rate on training, test and OOT data.

We explored five machine learning algorithms of varying complexity (underfitting, overfitting, and balance between bias and variance) before selecting our final model. The section below contains a high-level description of each algorithm along with an overview of the top five performing model configurations, as measured by the respective algorithm's average FDR at 3% on the out-of-time data. Each table also contains the model's average FDR at 3% on the training and test datasets, the names and configurations of explored parameters, as well as the number of variables used in each model.

Logistic Regression

Logistic regression is a linear statistical model which uses a logistic function to estimate probabilities of a transaction being fraudulent. This is the simplest model explored and serves as a baseline for more complex, non-linear models explored below.

$$f(x) = \frac{L}{1 + e^{-k(x - x_0)}}$$



Where L is the maximum value of the curve (i.e. 1)

Logistic regression model exploration:
Descriptions of the hyperparameters fine-tuned:

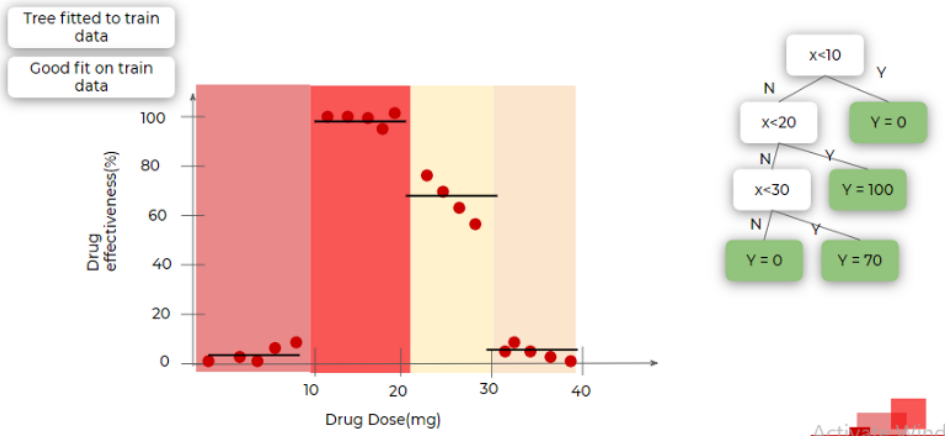
- Penalty: penalty for having too many variables
- C: inverse of regularization strength
- max_iter: the maximum number of iterations taken for the solvers to converge
- solver: algorithm used in an optimization problem

		Parameters				**Avg. FDR at 3%**		
Top 5 Runs	**# of Variables**	**penalty**	**C**	**max_iter**	**solver**	**trn**	**tst**	**oot**
(default)	10	l2	1	100	lbfgs	64.79%	66.21%	31.12%
1	10	l2	0.2	150	lbfgs	64.48%	63.71%	36.99%
2	10	elasticnet	0.3	150	saga	64.40%	64.57%	35.75%
3	10	l1	0.6	100	saga	64.15%	64.72%	33.41%
4	10	elasticnet	0.1	100	saga	64.33%	64.30%	33.35%
5	15	l2	1.3	100	lbfgs	64.01%	63.67%	31.40%

The best performance in logistic regression is approximately 37%.

Decision Tree

The decision tree model is a supervised machine learning model which learns through simple decision rules from the training data. It uses the features that work best for the data to split the dataset into smaller and smaller subsets, until no more choices can be made. The model is simple and straightforward, but is sensitive to outliers and relatively weak in making good predictions. The figure below provides a graphical and tree representation of a simple decision tree model. The graph on the left represents the model separating the space into boxes and assigning an output for each box (represented by the black horizontal line). The illustration on the right shows a tree structure representation of the same set of rules.



Decision tree model exploration:

Descriptions of the hyperparameters fine-tuned:

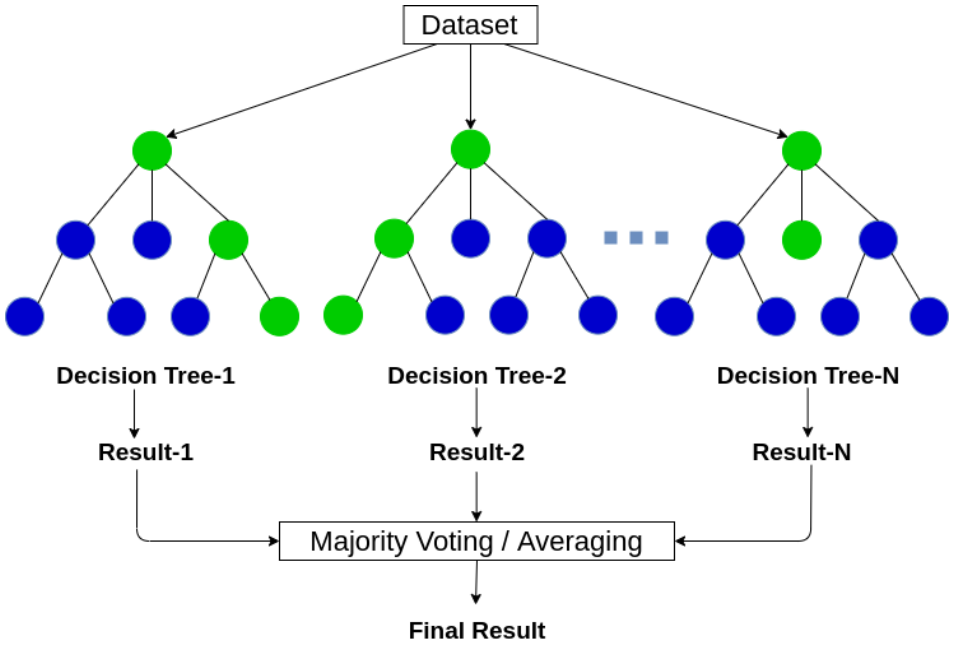
- max_depth: the maximum depth a tree can split to
- min_samples_leaf: the minimum number of samples to be at an internal node before the next split
- min_samples_split: the minimum number of samples required to split an internal node
- criterion: the function to measure the quality of split
- Splitter: the strategy used to choose the split at each node

		Parameters					Avg. FDR at 3%		
Top 5 Runs	**# of Variables**	**max_depth**	**min_samples_leaf**	**min_samples_split**	**criterion**	**Splitter**	**trn**	**tst**	**oot**
(default)	10	None	1	2	gini	best	100%	60.87%	28.49%
1	10	None	180	200	gini	best	71.61%	69.77%	46.65%
2	10	30	50	10	gini	best	84.06%	75.25%	44.92%
3	10	None	200	200	gini	best	72.56%	67.69%	44.30%
4	10	30	180	200	entropy	best	72.77%	69.07%	43.35%
5	10	None	150	150	entropy	best	73.26%	69.86%	41.62%

The best performance in the decision tree model is 46.65%.

Random Forest

A random forest supervised machine learning model is a collection of multiple independent decision trees which are combined by averaging (for regression problems) or voting (for classification problems). Specifically, A random forest model builds a bootstrapped dataset (with replacement) and creates a decision tree with only a random subset of variables from the bootstrapped dataset. The process is repeated multiple times to create a variety of unrelated trees. Each tree is used to make intermediate prediction results, and the final prediction on a record is made based on the results from the majority of the trees in a binary classification problem or average of the numerical values in a regression problem.
A random forest combats issues of overfitting and sensitivity in decision trees.



The figure above provides a visual representation of the model.

Random forest model exploration:

Descriptions of the hyperparameters fine-tuned:

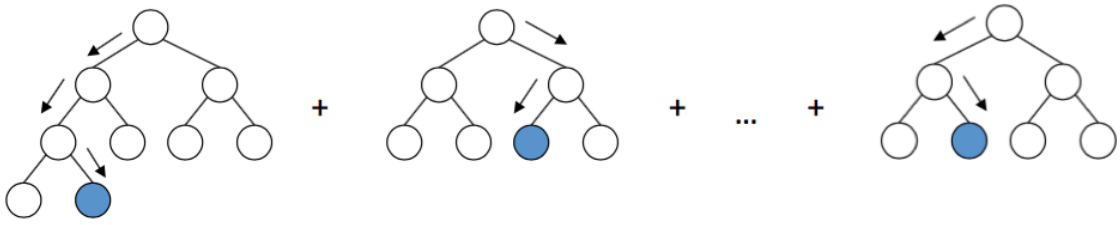
- max_depth: the maximum depth a tree can split to
- min_samples_leaf: the minimum number of samples to be at an internal node before the next split
- criterion: the function to measure the quality of a split
- n_estimators: the number of trees in the forest
- min_samples_split: the minimum number of samples to split an internal node

		Parameters					Avg. FDR at 3%		
Top 5 Runs	**# of Variables**	**max_depth**	**min_samples_leaf**	**criterion**	**n_estimators**	**min_samples_split**	**trn**	**tst**	**oot**
(default)	10	None	1	gini	100	2	100.00%	83.04%	46.15%
1	10	6	30	gini	50	50	76.05%	74.47%	48.00%
2	10	10	2	auto	10	10	88.31%	81.81%	46.76%
3	10	5	30	gini	50	50	72.33%	69.76%	45.31%
4	10	50	4	entropy	80	10	86.02%	79.72%	44.08%
5	10	5	50	gini	80	100	72.77%	69.00%	43.74%

The best performance in the random forest model is 48.00%.

Boosted Trees

Boosted tree models are another example of improvements to the decision tree model. Boosted trees are a sum of many weak, shallow trees. As an additional tree is added, the summed model improves.



Boosting works similarly to a Taylor Series, which starts with a basic approximation then adds terms to arrive at the closest approximation. Below is an example of a Taylor Series:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

When boosting, every additional term in the equation is a new model, and it is predicting the residual error at that level of models. Examples of boosted trees include: CatBoost, XGBoost, LightGBM. The model exploration below uses LightGBM. While LightGBM is prone to overfitting, it is efficient in training models with high accuracy, and is capable of handling large datasets.

Boosted tree model exploration:

Descriptions of the hyperparameters fine-tuned:

- max_depth: the maximum depth a tree can split to
- learning_rate: the impact of each tree on the final model
- num_leaves: the number of leaves to consider to best split the dataset
- n_estimators: the number of leaves boosted trees

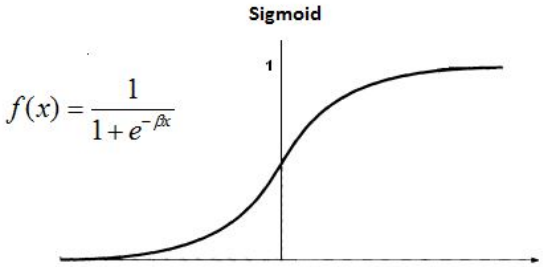
		Parameters				**Avg. FDR at 3%**		
Top 5 Runs	**# of Variables**	**max_depth**	**learning_rate**	**num_leaves**	**n_estimators**	**trn**	**tst**	**oot**
(default)	10	-1	0.1	31	100	99.48%	79.77%	45.14%
1	10	4	0.03	30	200	81.45%	76.93%	49.61%
2	10	4	0.004	30	400	82.80%	78.00%	49.00%
3	10	6	0.001	30	500	74.21%	70.75%	48.10%
4	10	4	0.002	30	400	75.94%	72.76%	47.93%
5	10	4	0.001	25	400	73.93%	70.37%	47.66%

The highest performance on oot data in the boosting tree model is 49.61%, however, it suffers from overfitting. Therefore, the best performance in the boosting tree model is the 4th run, with 47.93% FDR on oot data.

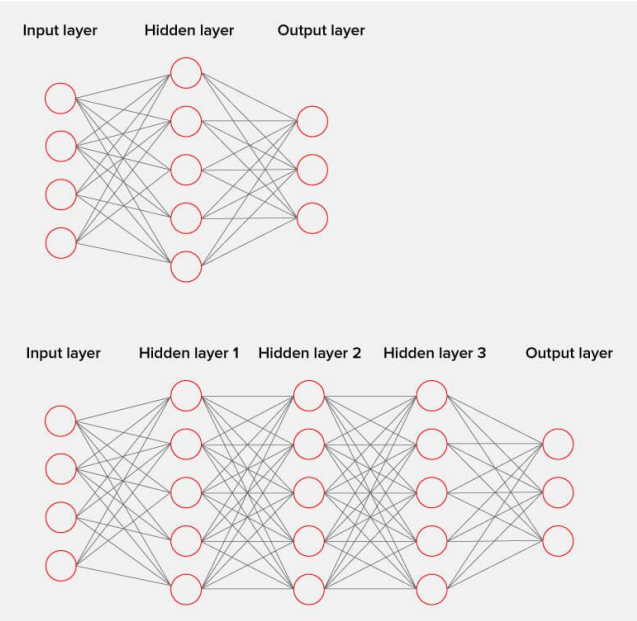
Neural Network

The neural network machine learning model is a supervised model that was inspired by the human brain, more specifically the way that signals are passed between neurons through connectors called axons. Similarly to this biological structure, the neural net model consists of an input layer (independent variables), hidden layers (set of nodes) and an output layer (dependent

variable). The model learns as nodes receive signals from a previous layer's nodes, in other words as it propagates inputs, weights and biases.



The activation function applied at each hidden layer can be logistic (most commonly used) or any other function.



The figure above provides an illustration of a shallow (top) and a deep neural network (bottom).
Neural net model exploration:

Descriptions of the hyperparameters fine-tuned:

- hidden_layer_sizes: the number of neurons in a particular hidden layer
- activation: activation function for the hidden layers
- solver: serves for weight optimization
- learning_rate_init: the initial learning rate used
- alpha: a parameter for L2 penalty (regularization term)

		Parameters					Avg. FDR at 3%		
Top 5 Runs	**# of Variables**	**hidden_layer_sizes**	**activation**	**solver**	**learning_rate_init**	**alpha**	**trn**	**tst**	**oot**
(default)	10	(100,)	relu	adam	0.001	0.0001	72.12%	70.92%	50.28%
1	10	(10, 10)	relu	adam	0.02	0.002	70.33%	68.59%	56.59%
2	10	(20, 20)	relu	adam	0.002	0.002	71.79%	69.05%	55.92%
3	10	(10,10)	relu	adam	0.05	0.002	68.46%	67.69%	54.80%
4	10	(15)	logistic	lbfgs	0.1	0.001	72.61%	70.24%	54.64%
5	5	(30, 30)	relu	adam	0.01	0.001	73.90%	69.30%	54.02%

The best performance in the neural network model is 56.59%.
The following table summarizes our exploration results (underfitting, overfitting, balance between bias and variance) for all the five model algorithms:

Model	Parameters							Average FDR at 3%		
	Iteration	# Variables	penalty	C	solver	verbose	max_iter	Train	Test	OOT
Logistic Regression	1	15	l2	1	lbfgs	0	100	64.11%	63.77%	30.17%
	2	15	l2	0.1	lbfgs	1	800	62.14%	62.38%	31.73%
	3	15	none	0.5	saga	0	200	63.11%	61.66%	30.67%
	4	15	l2	1.3	lbfgs	0	100	64.01%	63.67%	31.40%
	5	15	l2	1.6	lbfgs	0	200	64.34%	62.73%	31.01%
	6	15	l1	0.8	liblinear	0	300	64.14%	63.49%	30.28%
	7	10	l1	0.6	saga	0	100	64.15%	64.72%	33.41%
	8	10	elasticnet	1	saga	0	100	64.33%	64.30%	33.35%
	9	10	l2	0.2	lbfgs	0.1	150	64.48%	63.71%	36.99%
	10	10	elasticnet	0.3	saga	0.5	150	64.40%	64.57%	35.75%
	Iteration	# Variables	criterion	max_depth	min_samples_leaf	min_samples_split	splitter	Train	Test	OOT
Decision Tree	1	15	gini	3	2	2	random	54.73%	54.10%	30.28%
	2	15	gini	3	5	5	best	56.95%	56.36%	33.80%
	3	15	entropy	10	5	2	random	86.50%	70.84%	34.30%
	4	15	gini	10	10	3	best	96.67%	73.45%	40.67%
	6	15	gini	25	20	5	best	92.26%	74.33%	41.17%
	7	10	entropy	25	20	3	best	91.22%	73.11%	40.78%
	8	10	gini	30	20	5	best	92.61%	73.99%	42.35%
	9	10	gini	50	50	10	best	84.06%	75.25%	44.92%
	10	10	gini	None	5	5	best	100.00%	68.65%	33.58%
	11	5	gini	None	50	20	best	81.61%	73.22%	42.01%
	12	10	gini	None	100	100	best	77.99%	72.14%	41.73%
	13	10	entropy	None	120	100	best	75.80%	69.18%	40.83%
	14	10	entropy	None	150	150	best	73.26%	69.86%	41.62%
	15	10	gini	None	200	200	best	72.56%	67.69%	44.30%
	16	10	gini	None	180	200	best	71.61%	69.77%	46.65%
	17	10	entropy	None	180	200	best	72.77%	69.00%	43.35%
	Iteration	# Variables	n_estimators	max_depth	min_samples_leaf	min_samples_split	criterion	Train	Test	OOT
Random Forest	1	15	10	5	20	5	entropy	71.33%	69.70%	40.22%
	2	15	50	10	30	30	gini	85.89%	80.32%	41.34%
	3	15	80	15	10	10	gini	94.90%	83.35%	44.75%
	4	15	100	20	30	10	entropy	92.72%	83.05%	41.90%
	5	15	100	30	50	5	gini	92.76%	82.23%	42.51%
	6	10	10	5	5	5	gini	72.58%	71.73%	42.18%
	7	10	50	10	10	10	gini	88.31%	81.81%	46.76%
	8	10	80	20	50	10	entropy	86.92%	79.72%	44.08%
	9	10	100	30	5	5	entropy	100.00%	83.67%	48.77%
	10	5	40	30	20	20	gini	90.33%	78.87%	42.68%
	11	5	50	15	5	5	gini	98.49%	79.24%	45.08%
	12	10	50	6	80	80	gini	74.26%	71.01%	44.30%
	13	10	60	20	50	80	gini	74.54%	72.13%	43.97%
	14	10	40	None	80	80	gini	72.77%	69.00%	43.35%
	15	10	50	None	50	60	gini	72.33%	69.76%	45.31%
	16	10	50	Noe	30	50	gini	76.05%	74.47%	48.00%
	Iteration	# Variables	n_estimator	num_leaves	max_depth	learning_rate		Train	Test	OOT
Boosting Tree	1	15	100	30	3	0.1		85.60%	77.81%	45.59%
	2	15	100	20	3	0.01		72.34%	67.87%	46.76%
	3	15	300	50	4	0.001		74.11%	70.85%	45.92%
	4	15	500	40	4	0.02		91.68%	81.42%	48.21%
	5	15	800	60	6	0.002		89.15%	78.88%	50.39%
	6	15	1000	100	6	0.006		95.63%	81.56%	48.55%
	7	10	100	10	3	0.1		85.49%	78.61%	43.07%
	8	10	200	20	4	0.05		91.11%	80.49%	47.98%
	9	10	400	30	4	0.002		76.44%	73.85%	48.18%
	10	10	700	30	6	0.002		87.88%	77.73%	50.45%
	11	5	700	30	5	0.01		89.00%	75.86%	44.25%
	12	10	500	50	4	0.002		77.11%	74.07%	44.92%
	13	10	300	60	4	0.001		73.47%	71.09%	45.98%
	14	10	400	30	4	0.002		75.94%	72.76%	47.93%
	15	10	200	30	4	0.03		81.45%	76.93%	49.61%
	16	10	600	30	4	0.005		82.70%	77.02%	47.93%
	17	10	500	30	4	0.001		74.21%	70.75%	48.10%
	18	10	400	30	4	0.004		83.80%	78.00%	49.00%
	19	10	400	default	4	0.005		73.00%	69.93%	45.31%
	20	10	400	25	4	0.001		73.93%	70.37%	47.66%
	21	10	600	100	4	0.005		82.70%	77.53%	47.99%
	22	10	400	60	3	0.003		71.25%	67.88%	47.27%
	Iteration	# Variables	layer	nodes	activation	solver	learning_rate_init	alpha	Train	Test	OOT
Neural Network	1	15	1	3	relu	adam	0.01	0.001	66.24%	64.35%	40.73%
	2	15	1	10	relu	adam	0.001	0.001	66.83%	65.65%	46.76%
	3	15	1	20	logistic	lbfgs	0.001	0.001	75.87%	71.11%	46.76%
	4	15	2	10	relu	lbfgs	0.02	0.001	74.02%	68.96%	47.71%
	5	15	2	15	relu	adam	0.05	0.001	68.53%	65.99%	52.68%
	6	15	2	20	relu	adam	0.002	0.001	76.60%	72.53%	48.32%
	7	10	1	3	relu	lbfgs	0.001	0.001	66.91%	64.96%	41.23%
	8	10	1	10	relu	lbfgs	0.005	0.001	70.34%	69.85%	53.52%
	9	10	1	15	logistic	lbfgs	0.1	0.001	72.61%	70.24%	54.64%
	10	10	1	20	relu	lbfgs	0.001	0.001	72.21%	69.07%	53.52%
	11	10	2	10	relu	adam	0.01	0.001	72.64%	69.83%	52.46%
	12	10	2	10	relu	adam	0.05	0.002	68.46%	67.69%	54.80%
	13	10	2	20	relu	adam	0.002	0.001	71.79%	69.05%	55.92%
	14	5	2	20	relu	adam	0.002	0.001	75.69%	73.43%	51.23%
	15	5	2	20	relu	adam	0.08	0.001	76.06%	71.33%	52.74%
	16	5	2	25	relu	adam	0.02	0.001	70.30%	68.98%	53.74%
	17	5	2	30	relu	adam	0.01	0.002	73.90%	69.30%	54.02%
	18	10	2	10	relu	adam	0.02	0.002	70.33%	68.59%	56.59%

Results

Models	AVG FDR at 3%		
	trn	test	oot
Logistic Regression	64.48%	63.71%	36.99%
Decision Trees	71.61%	69.77%	46.65%
Random Forest	76.05%	74.47%	48.00%
Boosted Trees	75.94%	72.76%	47.93%
Neural Networks	70.33%	68.59%	56.59%

Based on model exploration performed above, we selected Neural Network as our final model. The rationale behind this decision was the high performance of the Neural Networks algorithm, as measured by the average FDR at 3%. As the goal of our model algorithm is to reduce financial losses due to fraudulent credit transactions, we trade model prediction accuracy with model explainability.

The final configuration of the Neural Networks model used is as follows:

# of Variables	hidden_layer_sizes	learning_rate_init	solver	activation	alpha
10	(10, 10)	0.02	adam	relu	0.002

Using these hyperparameter values, we then proceeded to evaluate the model on the training dataset only before training it on the training dataset again but evaluating it on the basis of the test dataset. Finally, we trained the model on the combined train and test datasets before measuring the performance using the OOT data. As opposed to the model tuning section, we did not average the performances across multiple runs but performed the following evaluations on one run only. Using the aforementioned approach, the model performed as follows on the training, testing, and out-of-time data:

Train	# Records	# Goods	# Bads	Fraud Rate
	59,010	58,395	615	1.0532%

	Bin Statistics					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	590	261	329	44.24%	55.76%	590	261	329	0.45%	53.76%	53.31	0.79
2	590	518	72	87.80%	12.20%	1180	779	401	1.33%	65.52%	64.19	1.94
3	590	551	39	93.39%	6.61%	1770	1330	440	2.28%	71.90%	69.62	3.02
4	590	573	17	97.12%	2.88%	2360	1903	457	3.26%	74.67%	71.41	4.16
5	590	572	18	96.95%	3.05%	2950	2475	475	4.24%	77.61%	73.38	5.21
6	591	576	15	97.46%	2.54%	3541	3051	490	5.22%	80.07%	74.84	6.23
7	590	584	6	98.98%	1.02%	4131	3635	496	6.22%	81.05%	74.82	7.33
8	590	583	7	98.81%	1.19%	4721	4218	503	7.22%	82.19%	74.97	8.39
9	590	580	10	98.31%	1.69%	5311	4798	513	8.22%	83.82%	75.61	9.35
10	590	583	7	98.81%	1.19%	5901	5381	520	9.21%	84.97%	75.75	10.35
11	590	590	0	100.00%	0.00%	6491	5971	520	10.22%	84.97%	74.74	11.48
12	590	587	3	99.49%	0.51%	7081	6558	523	11.23%	85.46%	74.23	12.54
13	590	581	9	98.47%	1.53%	7671	7139	532	12.22%	86.93%	74.70	13.42
14	590	584	6	98.98%	1.02%	8261	7723	538	13.22%	87.91%	74.68	14.36
15	591	583	8	98.65%	1.35%	8852	8306	546	14.22%	89.22%	74.99	15.21
16	590	585	5	99.15%	0.85%	9442	8891	551	15.22%	90.03%	74.81	16.14
17	590	587	3	99.49%	0.51%	10032	9478	554	16.23%	90.52%	74.29	17.11
18	590	587	3	99.49%	0.51%	10622	10065	557	17.24%	91.01%	73.78	18.07
19	590	587	3	99.49%	0.51%	11212	10652	560	18.24%	91.50%	73.26	19.02
20	590	584	6	98.98%	1.02%	11802	11236	566	19.24%	92.48%	73.24	19.85

Test	# Records	# Goods	# Bads	Fraud Rate
	25,290	25,025	265	1.0478%

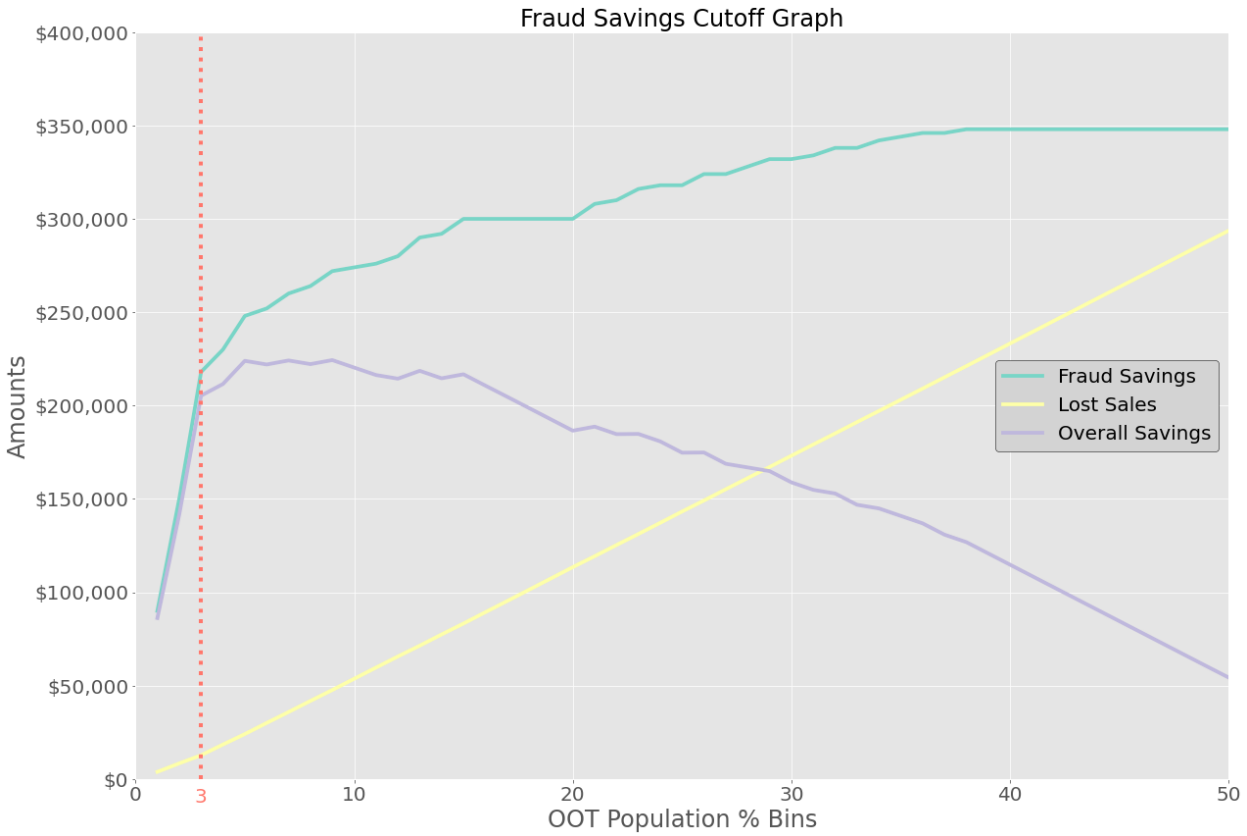
	Bin Statistics					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	253	119	134	47.04%	52.96%	253	119	134	0.48%	50.00%	49.52	0.89
2	253	215	38	84.98%	15.02%	506	334	172	1.33%	64.18%	62.84	1.94
3	253	234	19	92.49%	7.51%	759	568	191	2.27%	71.27%	69.00	2.97
4	253	245	8	96.84%	3.16%	1012	813	199	3.25%	74.25%	71.00	4.09
5	252	241	11	95.63%	4.37%	1264	1054	210	4.21%	78.36%	74.15	5.02
6	253	248	5	98.02%	1.98%	1517	1302	215	5.20%	80.22%	75.02	6.06
7	253	250	3	98.81%	1.19%	1770	1552	218	6.20%	81.34%	75.14	7.12
8	253	248	5	98.02%	1.98%	2023	1800	223	7.19%	83.21%	76.02	8.07
9	253	248	5	98.02%	1.98%	2276	2048	228	8.18%	85.07%	76.89	8.98
10	253	250	3	98.81%	1.19%	2529	2298	231	9.18%	86.19%	77.01	9.95
11	253	253	0	100.00%	0.00%	2782	2551	231	10.20%	86.19%	76.00	11.04
12	253	252	1	99.60%	0.40%	3035	2803	232	11.20%	86.57%	75.37	12.08
13	253	251	2	99.21%	0.79%	3288	3054	234	12.21%	87.31%	75.11	13.05
14	253	249	4	98.42%	1.58%	3541	3303	238	13.20%	88.81%	75.61	13.88
15	253	253	0	100.00%	0.00%	3794	3556	238	14.21%	88.81%	74.59	14.94
16	252	250	2	99.21%	0.79%	4046	3806	240	15.21%	89.55%	74.34	15.86
17	253	253	0	100.00%	0.00%	4299	4059	240	16.22%	89.55%	73.33	16.91
18	253	252	1	99.60%	0.40%	4552	4311	241	17.23%	89.93%	72.70	17.89
19	253	247	6	97.63%	2.37%	4805	4558	247	18.22%	92.16%	73.95	18.45
20	253	253	0	100.00%	0.00%	5058	4811	247	19.23%	92.16%	72.94	19.48

OOT	# Records	# Goods	# Bads	Fraud Rate
	12,097	11,918	179	1.4797%

	Bin Statistics					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	121	76	45	62.81%	37.19%	121	76	45	0.64%	25.14%	24.50	1.69
2	121	91	30	75.21%	24.79%	242	167	75	1.40%	41.90%	40.50	2.23
3	121	87	34	71.90%	28.10%	363	254	109	2.13%	60.89%	58.76	2.33
4	121	115	6	95.04%	4.96%	484	369	115	3.10%	64.25%	61.15	3.21
5	121	112	9	92.56%	7.44%	605	481	124	4.04%	69.27%	65.24	3.88
6	121	119	2	98.35%	1.65%	726	600	126	5.03%	70.39%	65.36	4.76
7	121	117	4	96.69%	3.31%	847	717	130	6.02%	72.63%	66.61	5.52
8	121	119	2	98.35%	1.65%	968	836	132	7.01%	73.74%	66.73	6.33
9	121	117	4	96.69%	3.31%	1089	953	136	8.00%	75.98%	67.98	7.01
10	121	120	1	99.17%	0.83%	1210	1073	137	9.00%	76.54%	67.53	7.83
11	121	120	1	99.17%	0.83%	1331	1193	138	10.01%	77.09%	67.08	8.64
12	121	119	2	98.35%	1.65%	1452	1312	140	11.01%	78.21%	67.20	9.37
13	121	116	5	95.87%	4.13%	1573	1428	145	11.98%	81.01%	69.02	9.85
14	121	120	1	99.17%	0.83%	1694	1548	146	12.99%	81.56%	68.58	10.60
15	121	117	4	96.69%	3.31%	1815	1665	150	13.97%	83.80%	69.83	11.10
16	121	121	0	100.00%	0.00%	1936	1786	150	14.99%	83.80%	68.81	11.91
17	120	120	0	100.00%	0.00%	2056	1906	150	15.99%	83.80%	67.81	12.71
18	121	121	0	100.00%	0.00%	2177	2027	150	17.01%	83.80%	66.79	13.51
19	121	121	0	100.00%	0.00%	2298	2148	150	18.02%	83.80%	65.78	14.32
20	121	121	0	100.00%	0.00%	2419	2269	150	19.04%	83.80%	64.76	15.13

Fraud Savings Calculation with Suggested Score Cutoff

Based on the assumption that $2,000 would be saved for every fraud caught (green curve) and $50 loss would incur for every false positive incident (yellow curve) The plot below shows the fraud savings, losses due to frauds and overall savings (purple curve).



The overall savings at 3% cutoff for the 2-month oot data are $205,300 using our Neural Network algorithm. The annualized overall savings are expected to be $1,231,800. Despite the fact that the overall savings are not maximized at 3% cutoff, we would still recommend it in order to ensure a seamless customer experience, which, in turn, could lead to improved word-of-mouth referrals and an increase in the number of customers.

Conclusions

In this report we worked with a dataset of 96,753 total credit card transaction records. We explored the dataset through summary tables and distributions on selected fields. During the data cleaning process, we removed outliers, narrowed our data to include only relevant purchase records, and filled in missing data using a variety of imputation techniques.

In the feature engineering stage, we created 7 types of variables including days since, velocity, relative velocity, amount, two Benford's Law variables and one risk table variable. Following our candidate variable creation, we then applied a filter and wrapper on our 747 total variables to generate the top 20 variables with the highest KS scores.

After completing the feature selection process, we used our top 20 variables to build model algorithms. The dataset was split into training, test and OOT data and a variety of models were explored. We used a linear logistic regression model as a baseline, and then explored non-linear model algorithms including Decision Tree, Random Forest, Boosted Trees (Light Gradient Boosting Machine) and Neural Networks.

We ultimately selected the Neural Net as our final model due to the efficiency in model training and high performance in fraud detection rate of 3%. Our final model contained 10 variables, 2 layers, 10 nodes and used a rectified linear unit activation function with an Adam solver. The average train, test and OOT rates from our results table were 71.90%, 71.27% and 60.89% respectively. The overall expected savings per annum for financial institutions are $1,231,800 if our model algorithm is implemented.

In the future, we would like to continue enhancing our model using more robust data and stronger machines that have the capability to deal with larger datasets. Because the volume of identifiable fraudulent activity was low, having a larger, more expansive dataset would allow us to explore additional hyperparameters to increase the fraud detection rate. We would also consider down sampling non-fraudulent activities and/or up sampling the fraudulent activities to handle imbalanced data. With this information, we would be able to generate a more comprehensive analysis and ultimately reduce potential fraud losses using our model.

Appendix I: Data Quality Report

1. High-Level Data Description

The data evaluated in this report contains actual credit card purchases from a US government organization, originally published on a website for a book on Forensic Analytics by Mark Nigrini. The 1,059 fraudulent transactions were added by a domain expert based on their experience with credit card fraud. The data frame has 10 fields and 97,753 records with transactions between January 1st, 2006, and December 31st, 2006, as expressed by the field 'Date.'

2. Summary Tables

Field Name	% Populated	Min	Max	Mean	Stdev	% 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

Table 1: Numeric Fields

Field Name	% Populated	# Unique Values	Most Common Value
Recnum	100	96,753	No single most common value, 96,753 value(s) share the most common count of 1
Cardnum	100	1,645	5142148452
Merchnum	96.51	13,091	930090121224

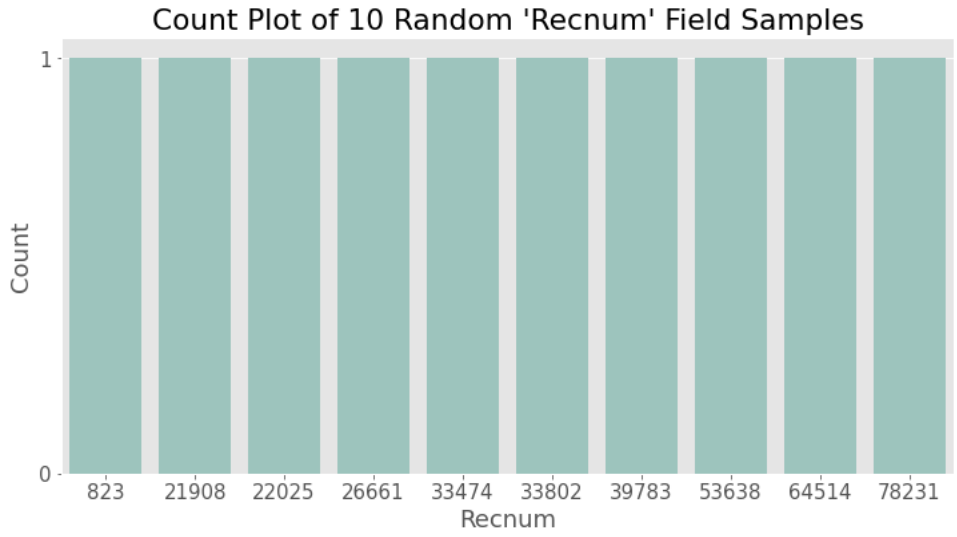
Merch description	100	13,126	GSA-FSS-ADV
Merch state	98.76	227	TN
Merch zip	95.19	4,567	38118
Transtype	100	4	P
Fraud	100	2	0

Table 2: Categorical Fields

3. Individual Field Investigations

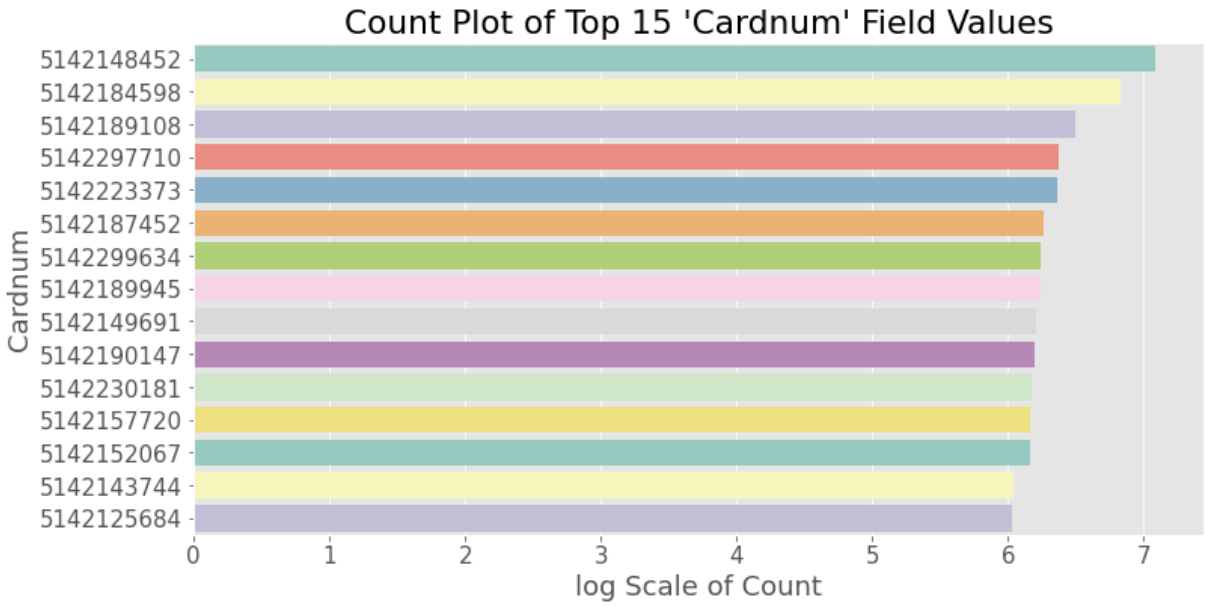
Field: 'Recnum'

96,753 unique values, one for each record of the data frame. The minimum value is '1' and the maximum value is '96,753.' The field seems to serve as a unique identifier for each record in the data frame.



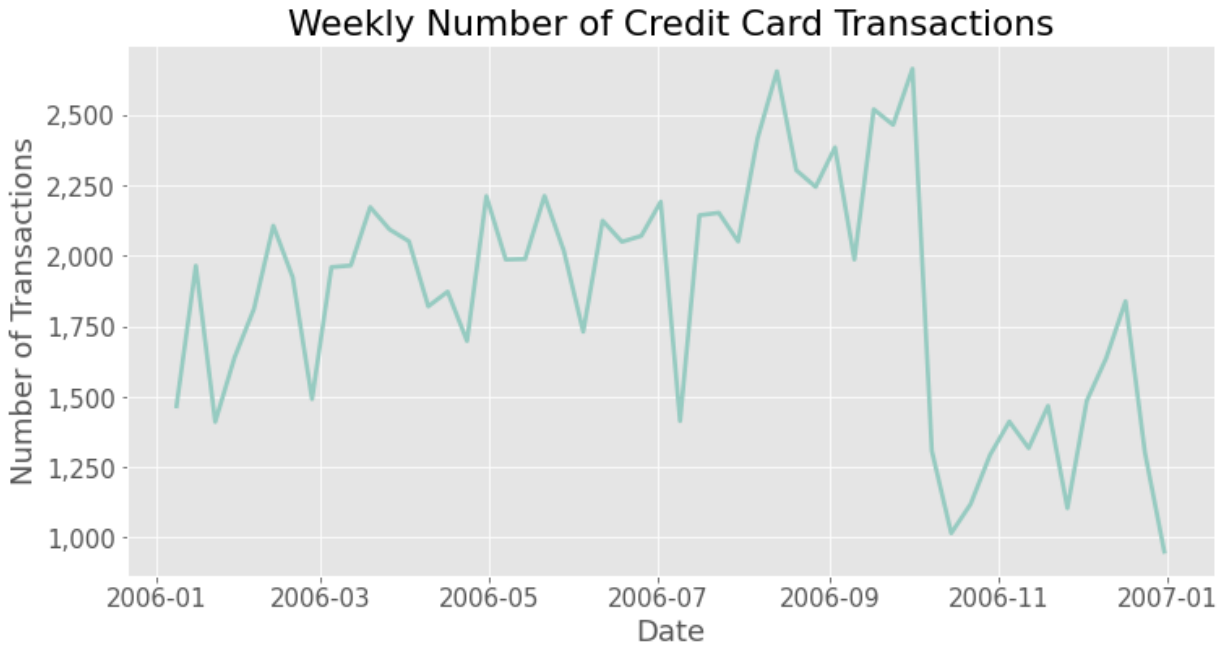
Field: 'Cardnum'

The 'Cardnum' field represents the credit card numbers associated with a transaction. Looking at the fifteen most common credit card numbers in the data set, two insights can be drawn. First, except for the two most common credit card numbers, the value counts for the remaining top fifteen credit card value counts are relatively similar. Second, given that all top fifteen credit card numbers begin with '51,' we can identify that all seem to be issued by MasterCard. This is based on the fact that the so-called Issuer Identification Number (IIN), which makes up the first two digits of credit card numbers, falls within MasterCard's range between 51 and 55 (Investopedia).



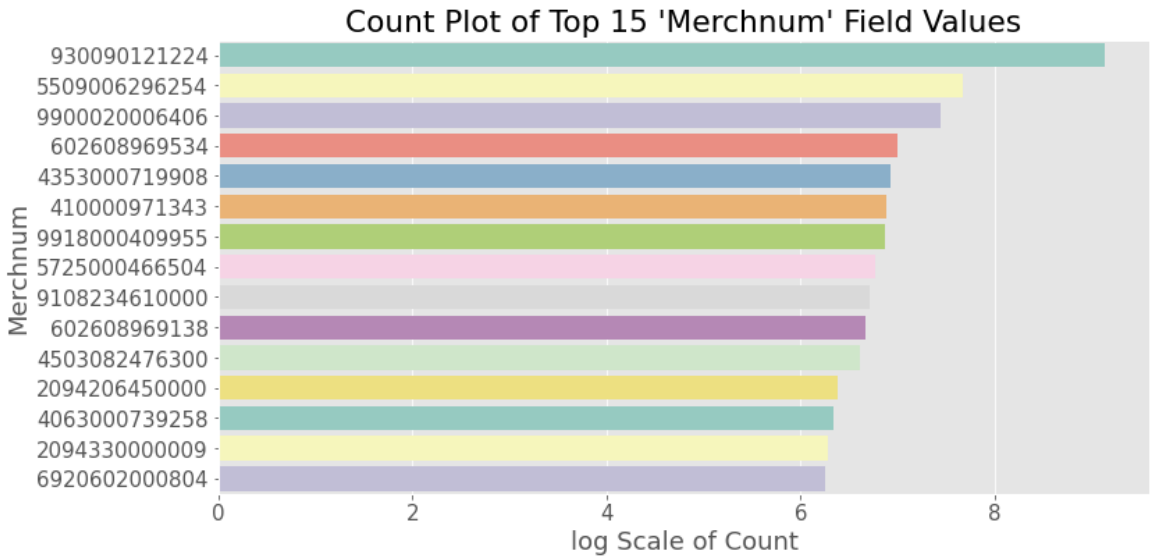
Field: 'Date'

The 'Date' field contains the date of the transaction. Looking at the weekly number of transactions for all full seven-day weeks in 2006, once can observe an increase in transactions until around October of 2006, before a sharp drop to a lower level towards the end of 2006.



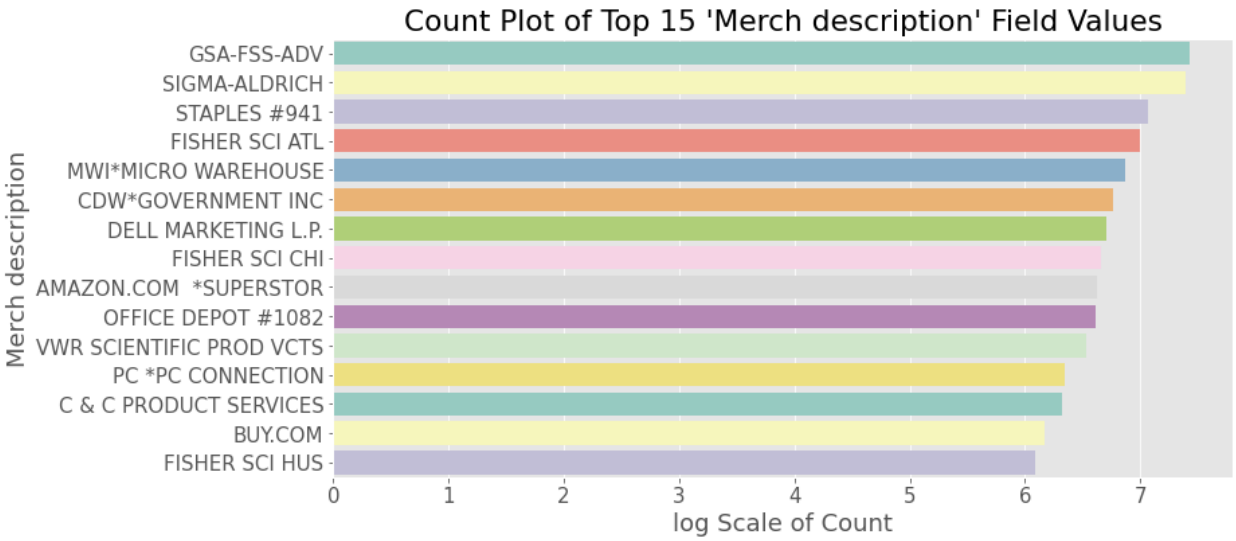
Field: 'Merchnum'

The field 'Merchnum' contains a merchant number. As evident from the count plot, the most common value '930090121224' is far more common than any other merchant number. One can also observe that the merchant numbers differ in length.



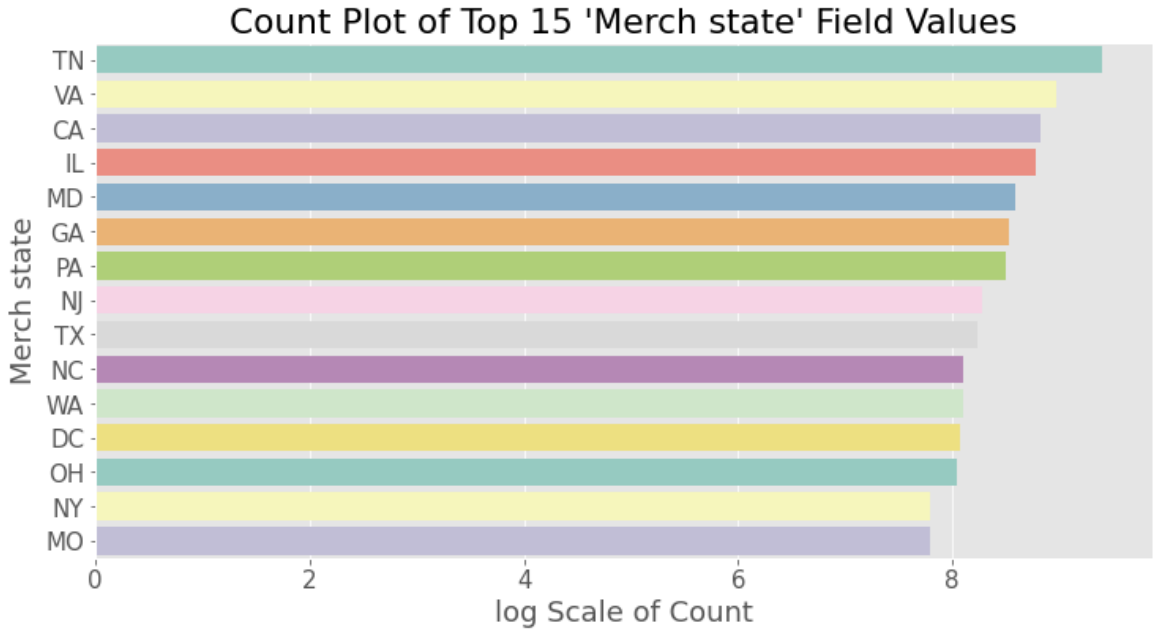
Field: ‘Merch description’

The field ‘Merch description’ contains merchant descriptions. There are indications of both online (‘BUY.COM’) and offline transactions (‘STAPLES #941’).



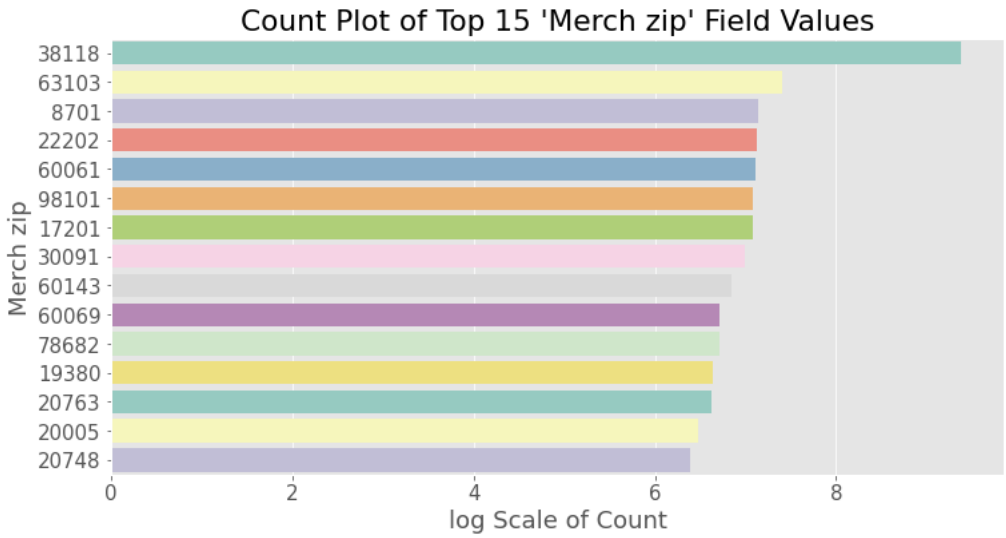
Field: ‘Merch state’

The field ‘Merch state’ contains US state name abbreviations. It is not specified whether these relate to the location of the transaction or the merchant’s headquarters in case of online transactions.



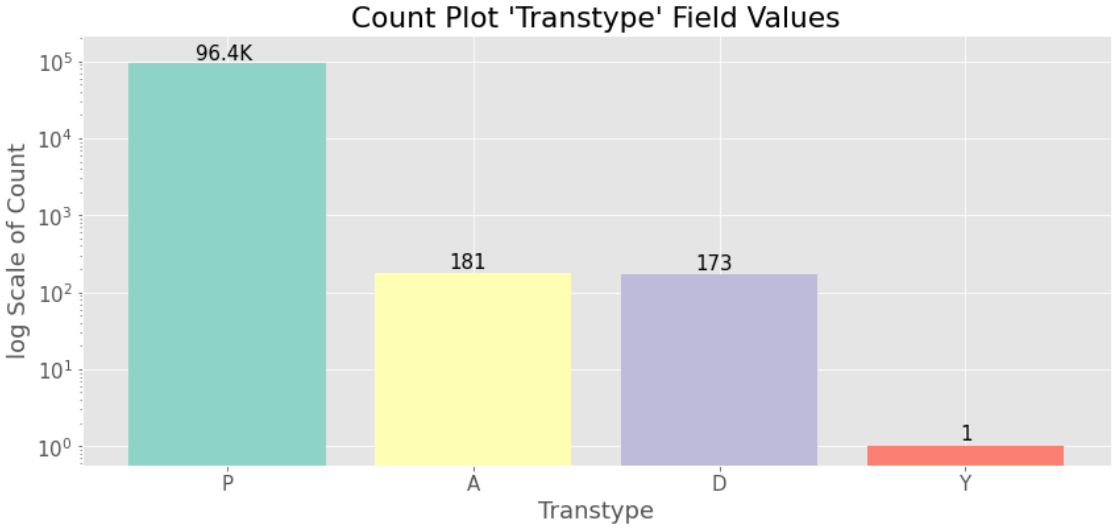
Field: ‘Merch zip’

The field ‘Merch zip’ contains a merchant zip code. One can observe that not all values seem to have five digits, as evident by the third most common value ‘8701.’



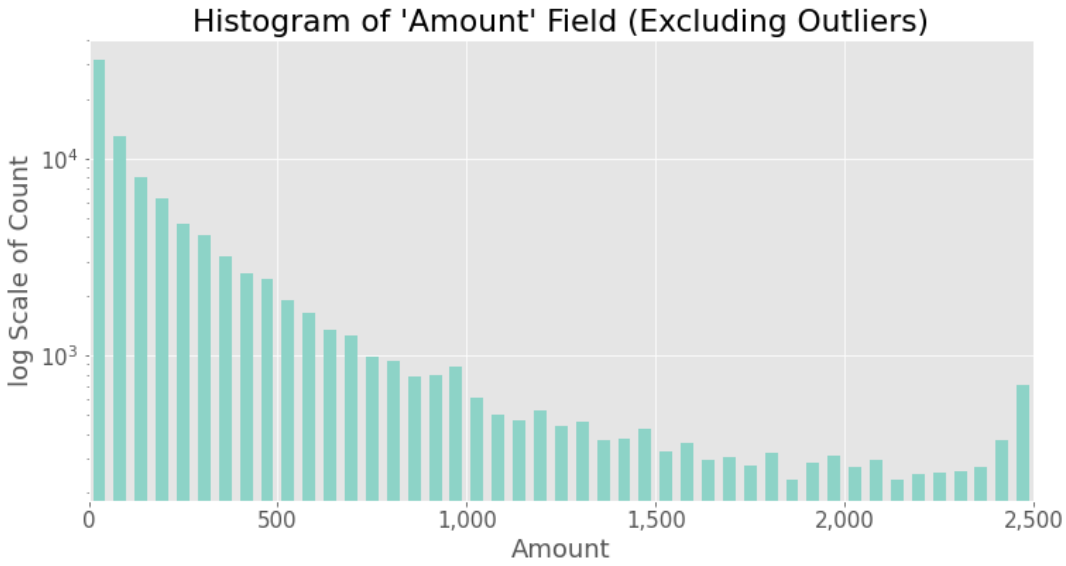
Field: ‘Transtype’

The field ‘Transtype’ refers to transaction types, which are not further specified. The value ‘P’ could refer to the purchase of an item or service.



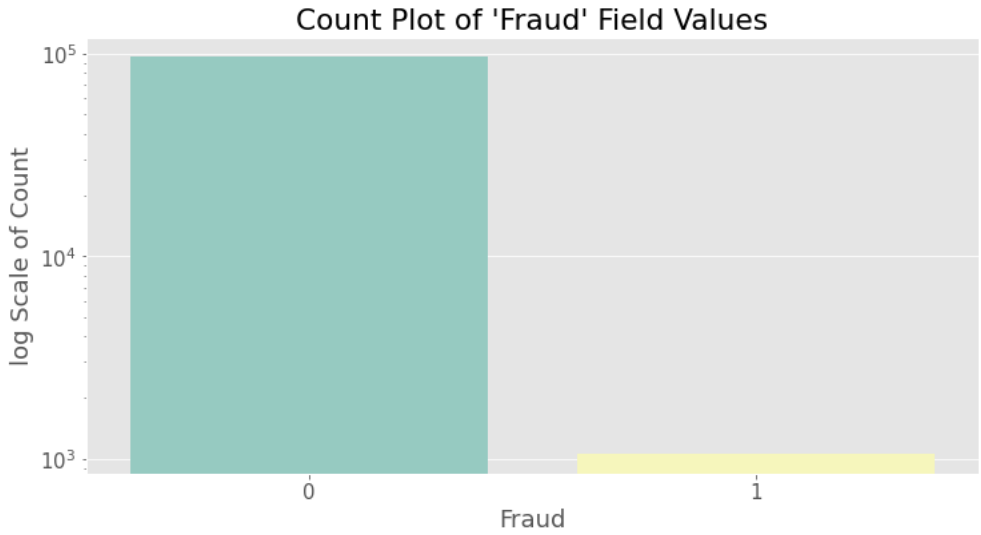
Field: ‘Amount’

The field ‘Amount’ contains the transaction amount, presumably in USD. Looking at the histogram of values, excluding values larger than 2,500 (99.2% of values), one can observe that most transactions amounts are rather small. In fact, 78.2% of all transactions have an amount less than or equal to 500.

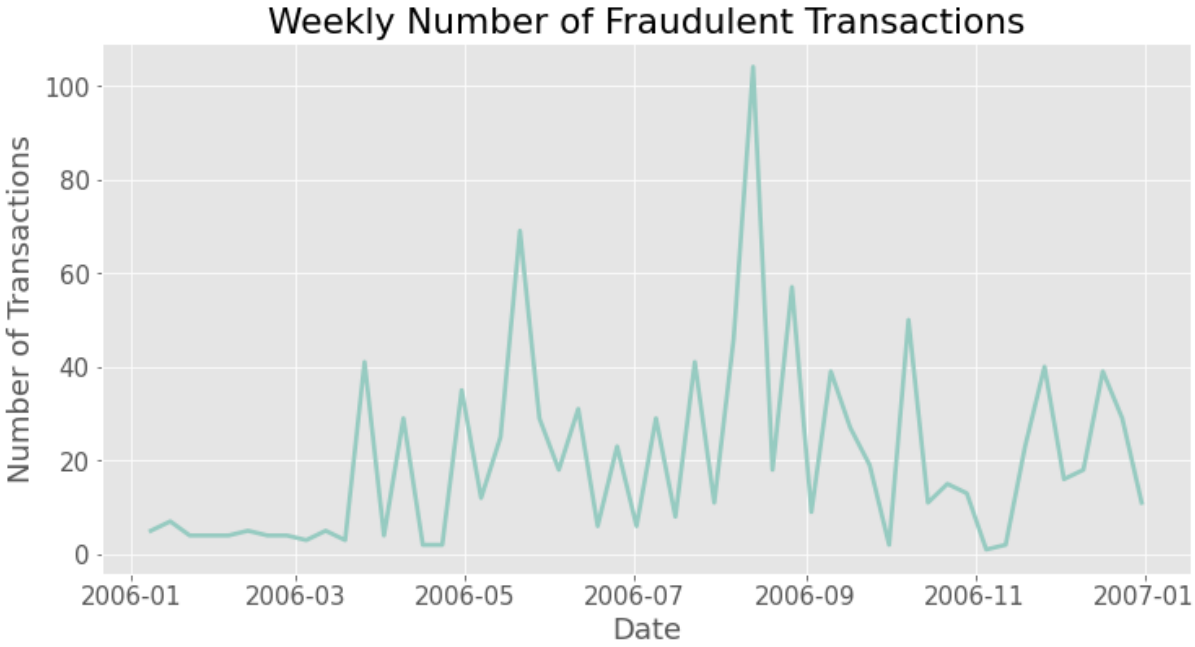


Field: ‘Fraud’

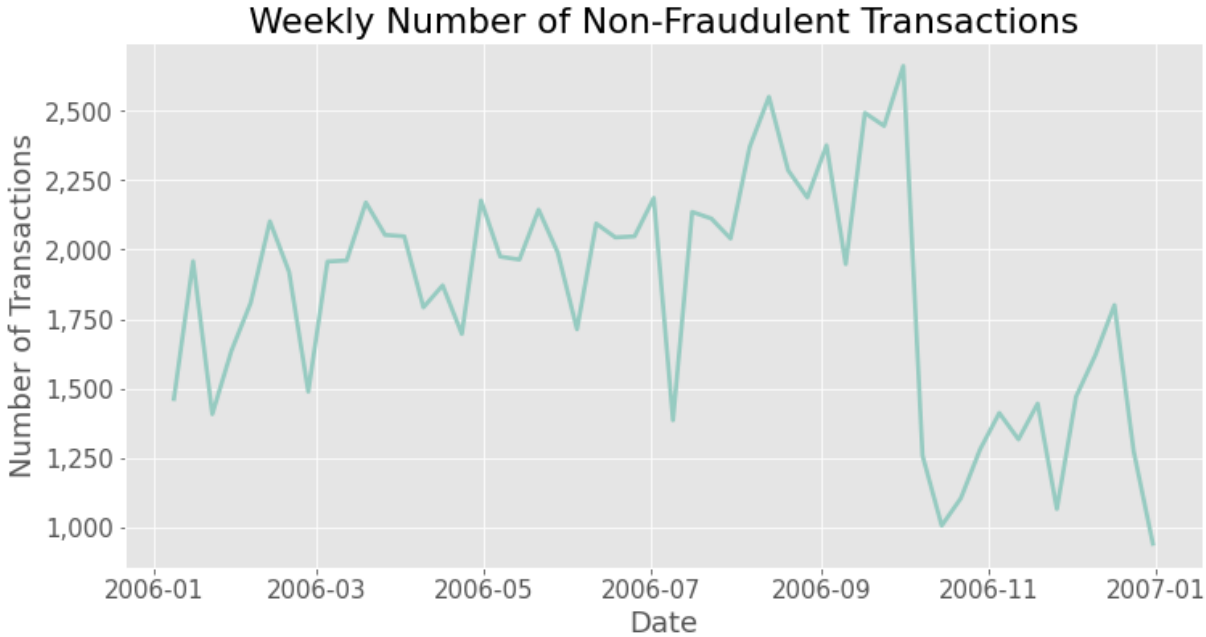
The ‘Fraud’ field is a binary variable indicating whether the application was fraudulent.



As there are significantly more non-fraudulent than fraudulent applications, it makes sense to separately examine both cases over time.



Fraudulent transactions seem to have peaked in August of 2006. Generally, there seem to be rather few fraudulent transactions from January through mid-March of 2006.



The plot of weekly non-fraudulent transactions closely mirrors that of the total transactions above.