Credit Card Fraud Detection

Wilsven Leong

July 11, 2021

Introduction

Credit card fraud losses total billions of dollars each year and is a major problem for financial institutions, customers and merchants. The 2018 Nilson Report estimated over $9 billion in fraudulent credit card transactions in the United State alone. Banks process thousands of transactions every minute and possess huge data sets, making good fraud detection models both necessary and possible. Since credit card companies cannot release real client data due to confidentiality, I chose a synthetic dataset from Kaggle to explore the issue. While synthetic data will not show real-world trends or unearth new predictors, it still provides excellent practice for analysis and machine learning modeling to detect anomalies and trends.

The dataset was generated using a Sparkov Data Generation Tool containing 23 real-life variables. It contains two years of data for 1000 cardholders. The set can be found at kaggle.com/kartik2112/fraud-detection. The data has already been split into training and test sets. The training set is large with over 1.2 million rows.

I will explore the data to look for trends to detect fraud. Once relevant features are chosen, I will compare several algorithms presented in HarvardX's Machine Learning course. The biggest problem for fraud detection models are imbalanced datasets. Credit card datasets are very large with few fraudulent transactions. Therefore, instead of overall accuracy, the focus of this project will be the process of choosing predictors, comparing and tuning algorithms for predicting the minority class and cost-saving results.

Data Analysis

Data Exploration

By examining the variables, we can see the variables are a mixture of date, categorical, numeric, and geospatial data:

```
Rows: 1,296,668
Columns: 23
$ X1                    <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14~
$ trans_date_trans_time <dttm> 2019-01-01 00:00:18, 2019-01-01 00:00:44, 2019-~
$ cc_num                <dbl> 2703186189652095, 630423337322, 38859492057661, ~
$ merchant              <chr> "fraud_Rippin, Kub and Mann", "fraud_Heller, Gut~
$ category              <chr> "misc_net", "grocery_pos", "entertainment", "gas~
$ amt                   <dbl> 5.0, 107.2, 220.1, 45.0, 42.0, 94.6, 44.5, 71.7,~
$ first                 <chr> "Jennifer", "Stephanie", "Edward", "Jeremy", "Ty~
$ last                  <chr> "Banks", "Gill", "Sanchez", "White", "Garcia", "~
$ gender                <chr> "F", "F", "M", "M", "M", "F", "F", "M", "F", "F"~
$ street                <chr> "561 Perry Cove", "43039 Riley Greens Suite 393"~
$ city                  <chr> "Moravian Falls", "Orient", "Malad City", "Bould~
$ state                 <chr> "NC", "WA", "ID", "MT", "VA", "PA", "KS", "VA", ~
```

```
$ zip                  <dbl> 28654, 99160, 83252, 59632, 24433, 18917, 67851,~
$ lat                  <dbl> 36, 49, 42, 46, 38, 40, 38, 39, 40, 37, 41, 39, ~
$ long                 <dbl> -81, -118, -112, -112, -79, -75, -101, -79, -80,~
$ city_pop             <dbl> 3495, 149, 4154, 1939, 99, 2158, 2691, 6018, 147~
$ job                  <chr> "Psychologist, counselling", "Special educationa~
$ dob                  <date> 1988-03-09, 1978-06-21, 1962-01-19, 1967-01-12,~
$ trans_num            <chr> "0b242abb623afc578575680df30655b9", "1f76529f857~
$ unix_time            <dbl> 1325376018, 1325376044, 1325376051, 1325376076, ~
$ merch_lat            <dbl> 36, 49, 43, 47, 39, 41, 37, 39, 40, 37, 40, 40, ~
$ merch_long           <dbl> -82, -118, -112, -113, -79, -76, -100, -79, -80,~
$ is_fraud             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

The variables are a mixture of customer, merchant and transaction specific data. Customer related variables include: first & last name, gender, multiple columns of address information, date of birth, and job. Transaction variables are: date and time, card number, purchase category, amount, id, unix time, and if fraud. Merchant variables include: name, latitude and longitude. There is also a row id.

There are quite a few redundant variables. First/last names and cc_num all identify for the individual account. There are seven variables related to the account address: street address, city, zip code, state, city population, longitude and latitude. The street address of a customer can be a good predictor of fraud. It isn't a logical predictor in this dataset since the is no transaction processing data included. Instead, I will focus on cc_nums, longitude, and latitude.

Summary of numeric and date variables:

```
 trans_date_trans_time             cc_num                          amt
 Min.   :2019-01-01 00:00:18   Min.   :        60416207185   Min.   :    1
 1st Qu.:2019-06-03 19:12:05   1st Qu.:    180042946491000   1st Qu.:   10
 Median :2019-10-03 07:32:34   Median :   3521417320840000   Median :   48
 Mean   :2019-10-03 12:45:25   Mean   : 417187580599000000   Mean   :   70
 3rd Qu.:2020-01-28 14:54:14   3rd Qu.:   4642255475290000   3rd Qu.:   83
 Max.   :2020-06-21 12:10:56   Max.   :4992346398069999616   Max.   :28949

      zip           city_pop            dob                is_fraud
 Min.   : 1257   Min.   :     23   Min.   :1924-10-30   Min.   :0.00
 1st Qu.:26237   1st Qu.:    743   1st Qu.:1962-08-13   1st Qu.:0.00
 Median :48174   Median :   2456   Median :1975-11-30   Median :0.00
 Mean   :48801   Mean   :  88825   Mean   :1973-10-03   Mean   :0.01
 3rd Qu.:72042   3rd Qu.:  20328   3rd Qu.:1987-02-22   3rd Qu.:0.00
 Max.   :99783   Max.   :2906700   Max.   :2005-01-29   Max.   :1.00
 NA's   :1       NA's   :1         NA's   :1            NA's   :1
```

The summary provides several important revelations. Even though transaction amounts have a large range from $1 to ~$28,000, the mean is only $70. Since the third quartile starts at $83, it is apparent that most transactions are under $100.

The date range is 2019-01-01 to 2020-06-21 not the expected two years. The Kaggle page stated the data covered two years from 2019 to 2020. However, when you check the date ranges, you can see that the training and test sets were split by dates. This meant that the test set consists of transactions in the last six months.

Training set: 2019-01-01 00:00:18, 2020-06-21 12:10:56.

Test set: 2020-06-21 12:14:25, 2020-12-31 23:59:34.

This seems like a poor sampling method especially for time trends. Therefore, I will recombine the data from the train and test sets and re-partition into 80/20 training and test splits based on random sampling method. Let's check for NA values in is_fraud column.

```
## [1] 1
```

Once resampled, there are now both training and test sets covering two years. Tabulating the proportion of is_fraud shows 0.52% transactions of transactions are fraudulent.

```
##
##      0      1
## 0.9948 0.0052
```

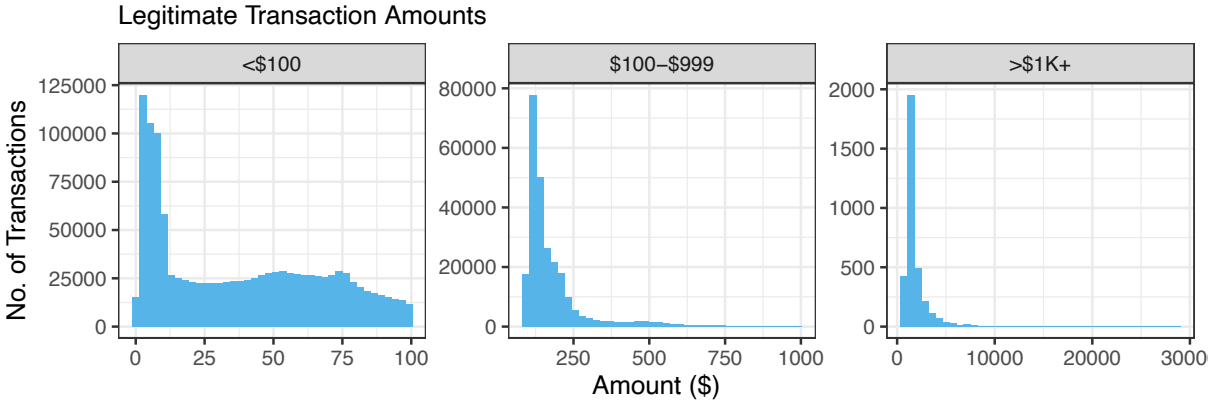
Now that the training set is re-partitioned, let's explore the difference between legitimate and fraudulent transactions to better understand the data.

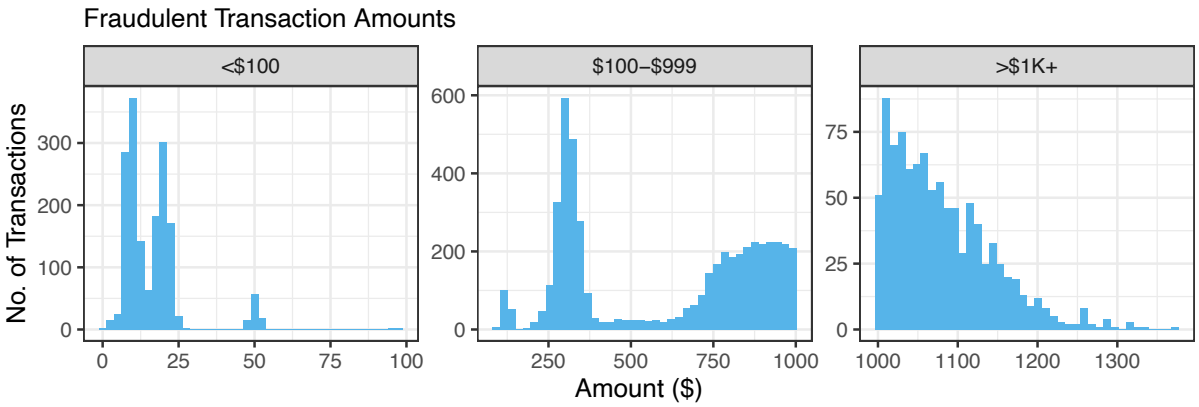
is_fraud	avg_trans	med_trans	amt	n	pct_amt	pct_n
0	68	47	99622852	1474179	96	99.48
1	535	433	4132810	7729	4	0.52

The cost of fraud in the training set is ~$4,000,000 and makes up 4.0% of the total amount transacted. This is eight times higher than the percent of number of transactions. The average fraud transaction is also much higher at ~$500, making amount a likely predictor. Since the average amount for fraudulent transactions is much higher, it is important to investigate further and summarize the transaction amounts into segments (bins).

bins	is_fraud	amt	n	pct_amt	pct_n
<$100	0	45492059	1212368	99.94	99.86
<$100	1	27622	1685	0.06	0.14
$100-$999	0	47369360	258380	93.98	98.08
$100-$999	1	3034484	5053	6.02	1.92
>$1K+	0	6761433	3431	86.33	77.59
>$1K+	1	1070704	991	13.67	22.41

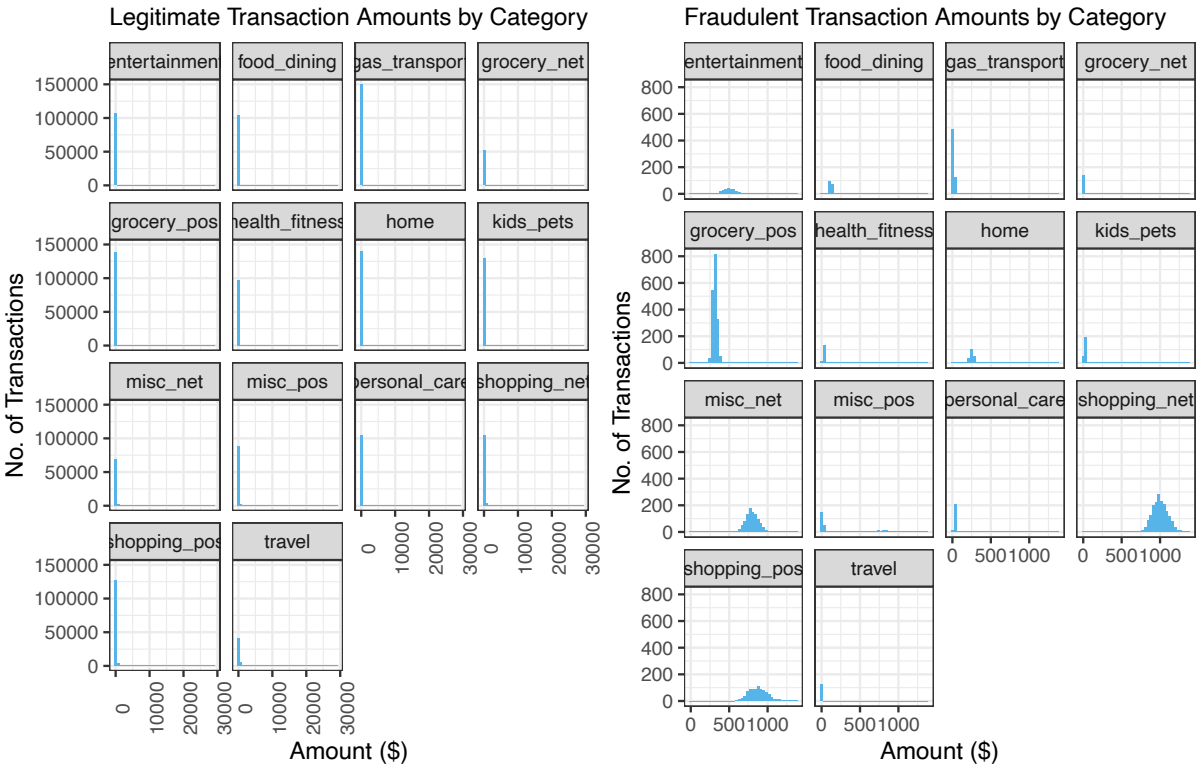
Since over 1.2 million transactions are under $100 this segment will overwhelm the visualization of the amount distribution if viewed together. When transaction amounts are segmented into bins and the scales are allowed to float to its respective bins, a better understanding of the distributions can be achieved.



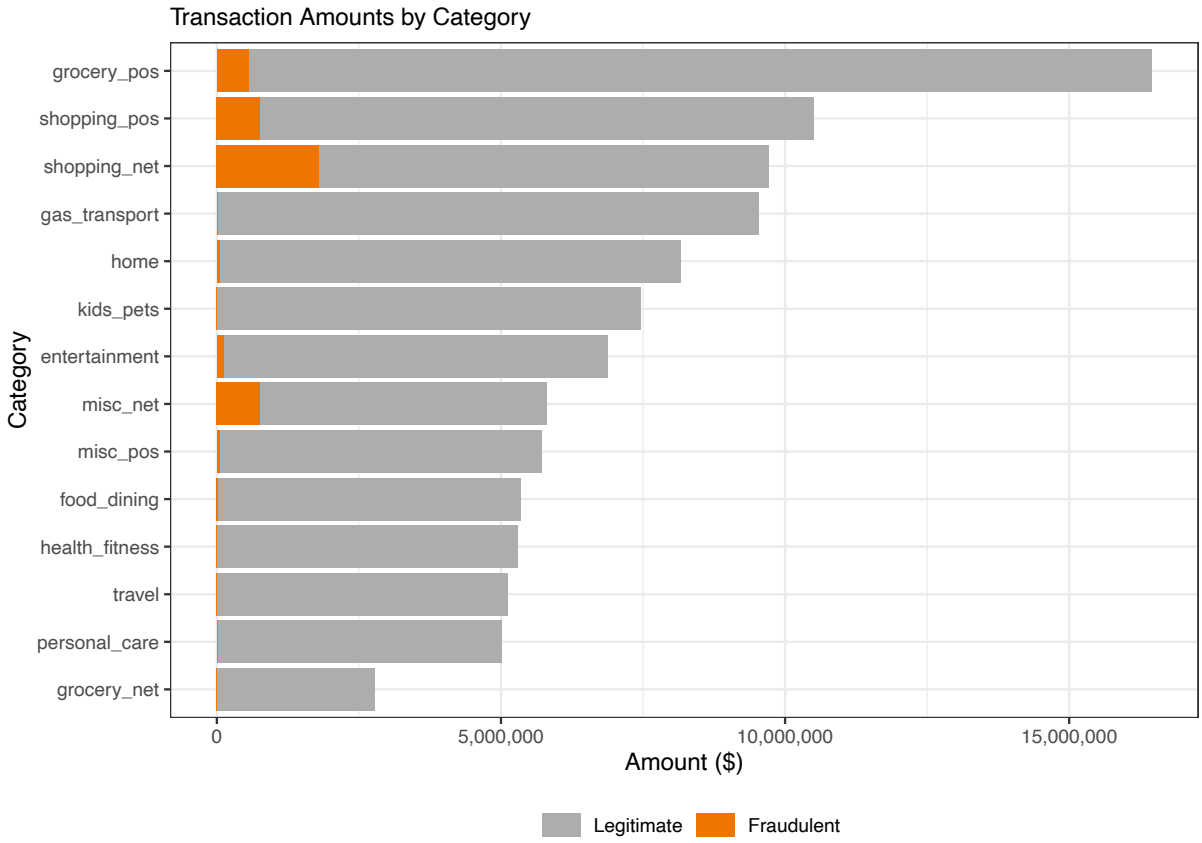


While the majority of legitimate transactions are under $100, 78.2% of fraudulent transactions are over $100. The amount of the transaction is definitely a predictor.

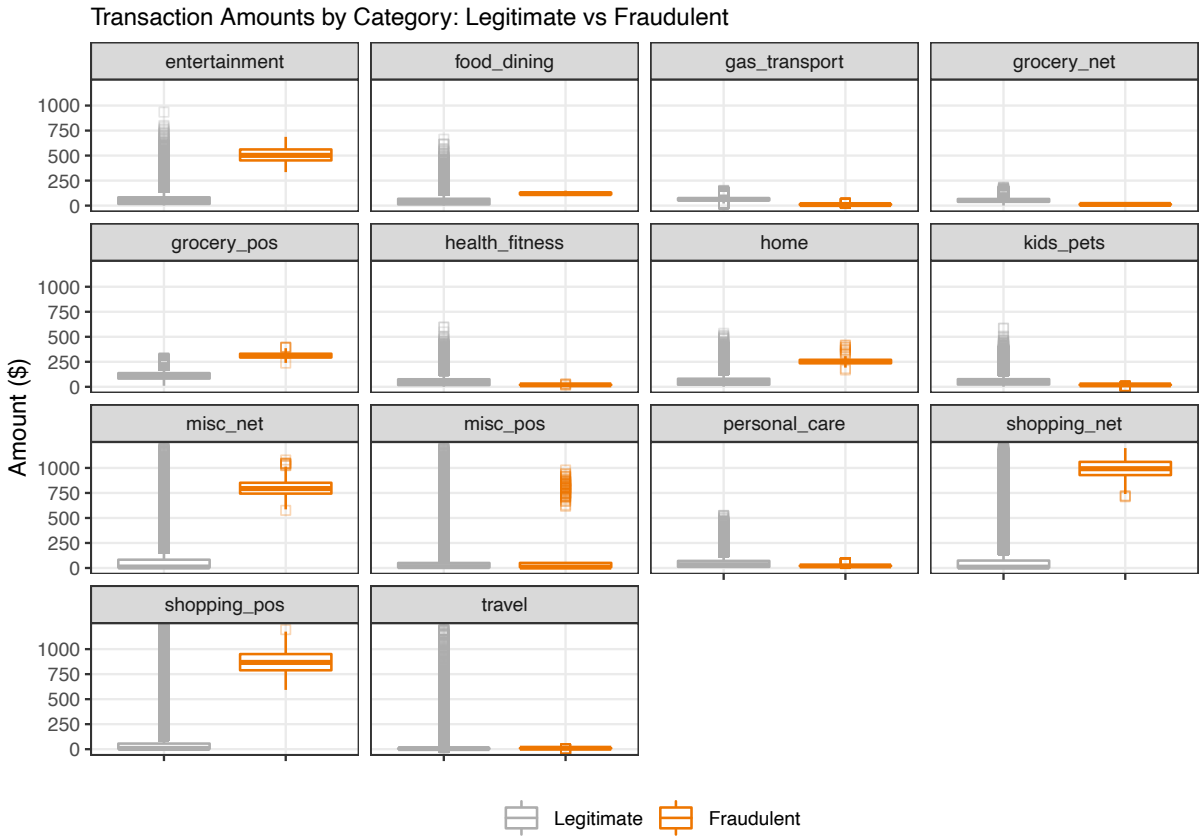
Next, let's explore categories. What are the actual transaction amounts by category, and how do they vary?

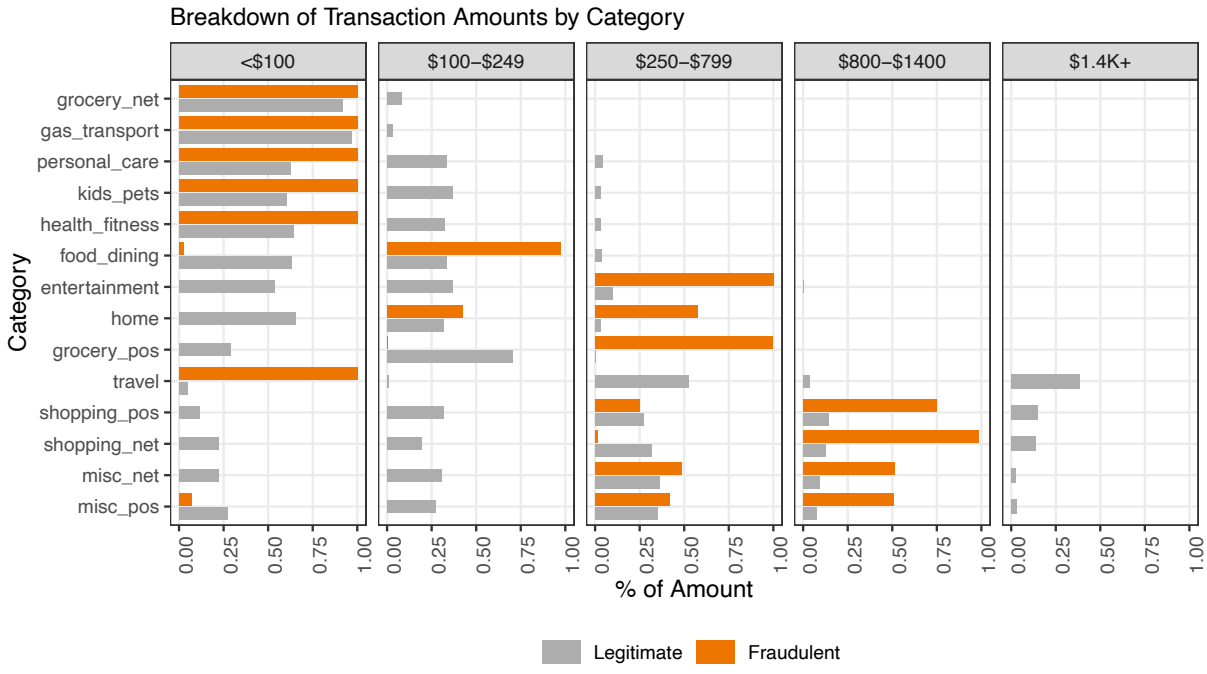


The distributions for transaction amount by category differ drastically for fraudulent transactions but remains similar otherwise.



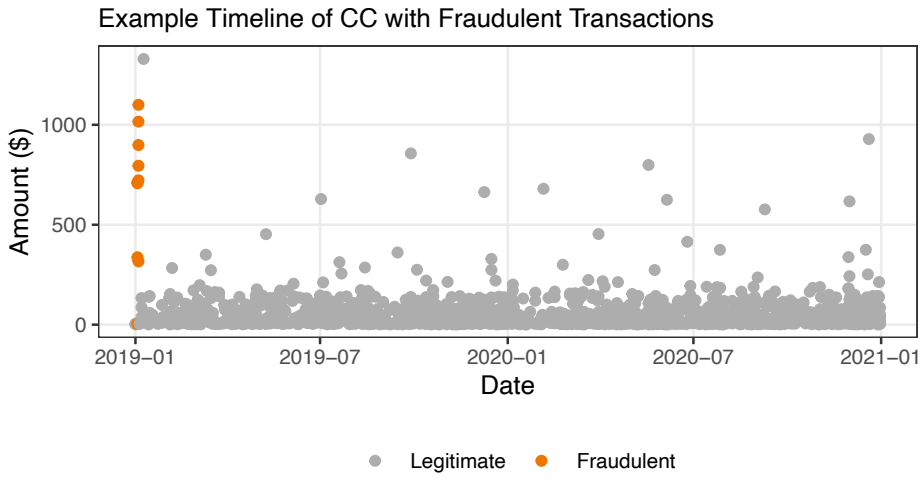
The plot above shows that fraudulent transactions are very specific to certain ranges within categories. Looking at the fraudulent amounts range by category, it is possible to use amount bins to better predict fraud.





The transaction amounts and categories are very good predictors in this dataset.

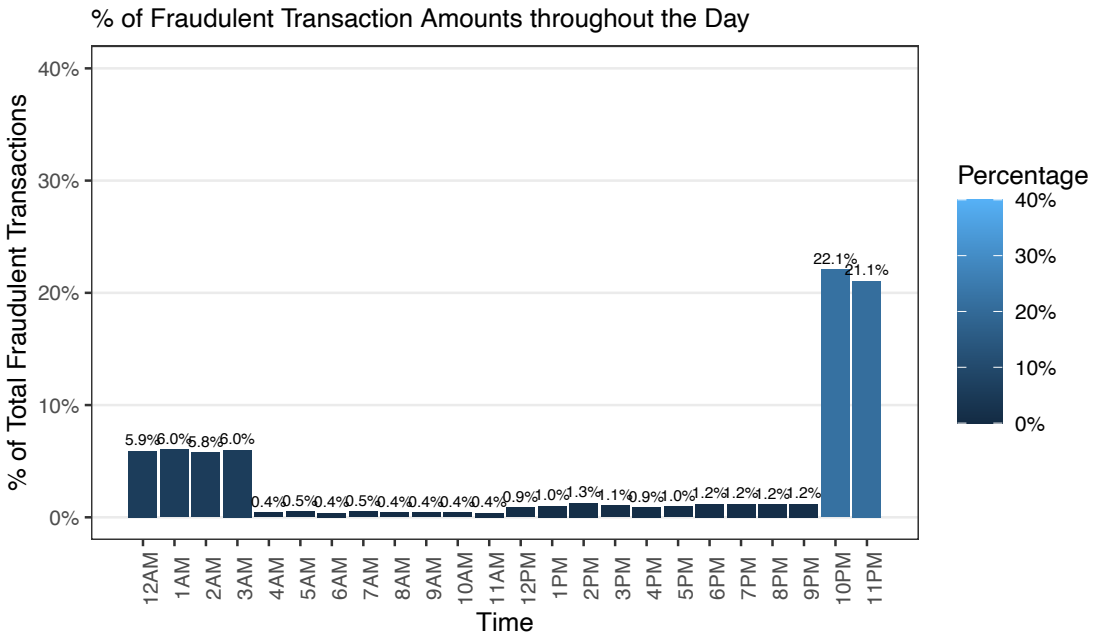
It is very likely that date and time elements also provide important insights. Therefore, visualizing all transactions on an account with fraudulent transactions will help get an idea of the relationship between timing and the nature of transactions.



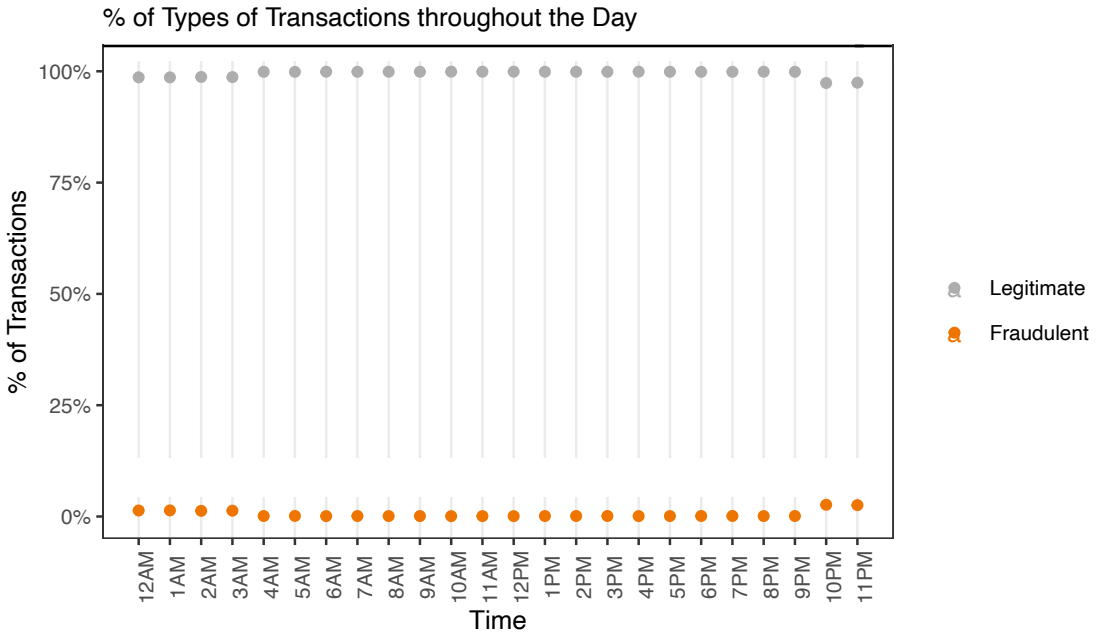
From the scatter plot above, it appears as though fraudulent transactions happen in batches in very quick successions (i.e. within a couple of days). This shows that timing and frequency of transactions are potential predictors.

Therefore, the date and time column should be explored more. We can first explore transaction timings. When visualizing the percentages of transaction amounts that are fraudulent over a timescale of an hour interval throughout the day, it seems that between 10PM to 3AM, fraudulent transaction amounts are substantially higher.

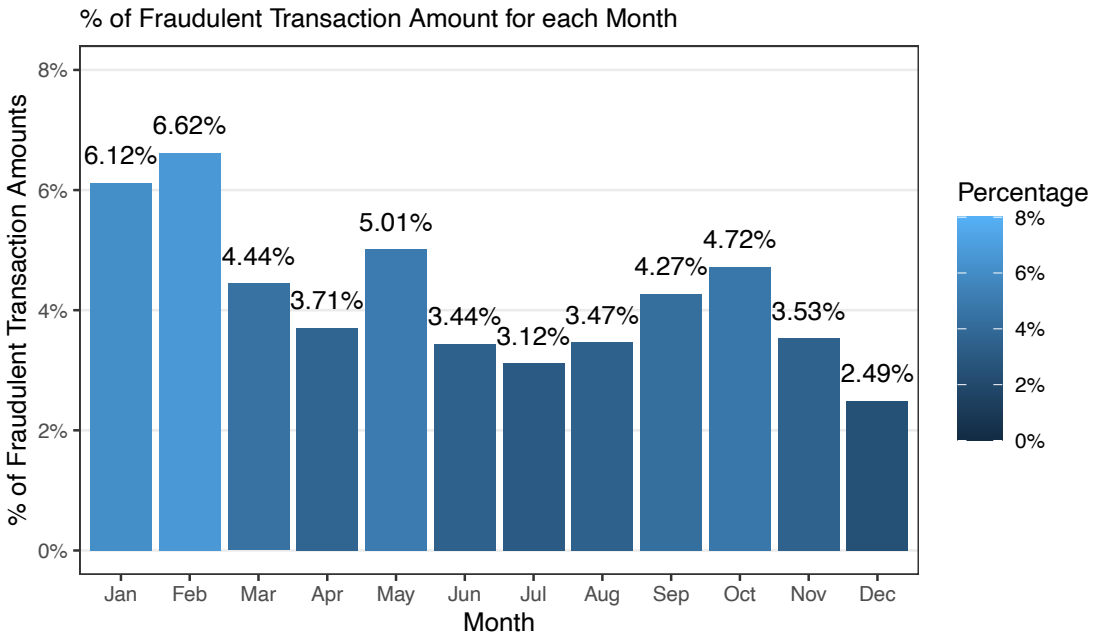
The likelihood for fraudulent transactions is especially prevalent between 10PM to 11PM, as shown in the plot below. With approximately 30.0% of transaction amounts being frauds, this is as much as 30 times more than other timings between 4AM to 9PM.



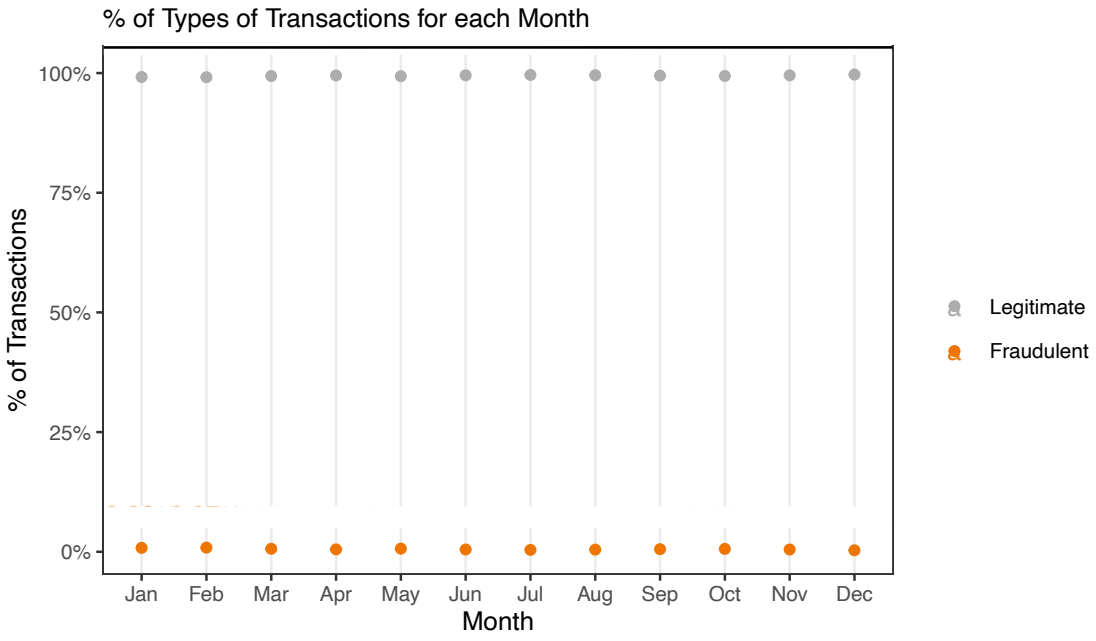
Unsurprisingly, if we plot the number of transactions that are fraudulent, we can see that the proportion of fraudulent transactions between 10PM to 3AM occur at higher rates (approximately 10 times) than other times throughout the day.



Next, let us take a look into the monthly proportion of fraudulent transaction amounts.

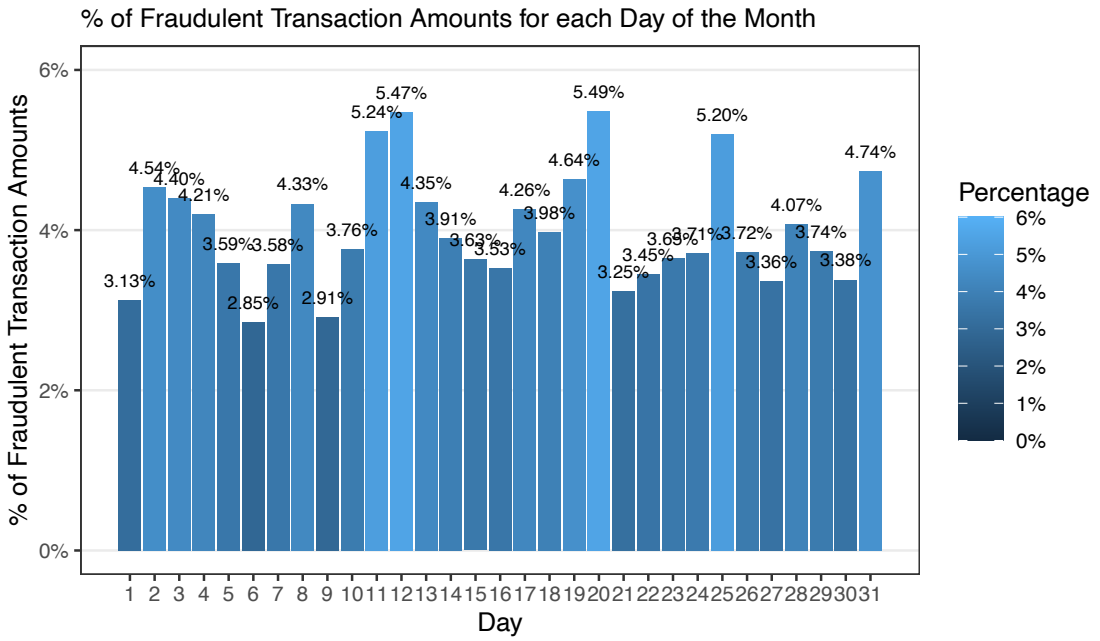


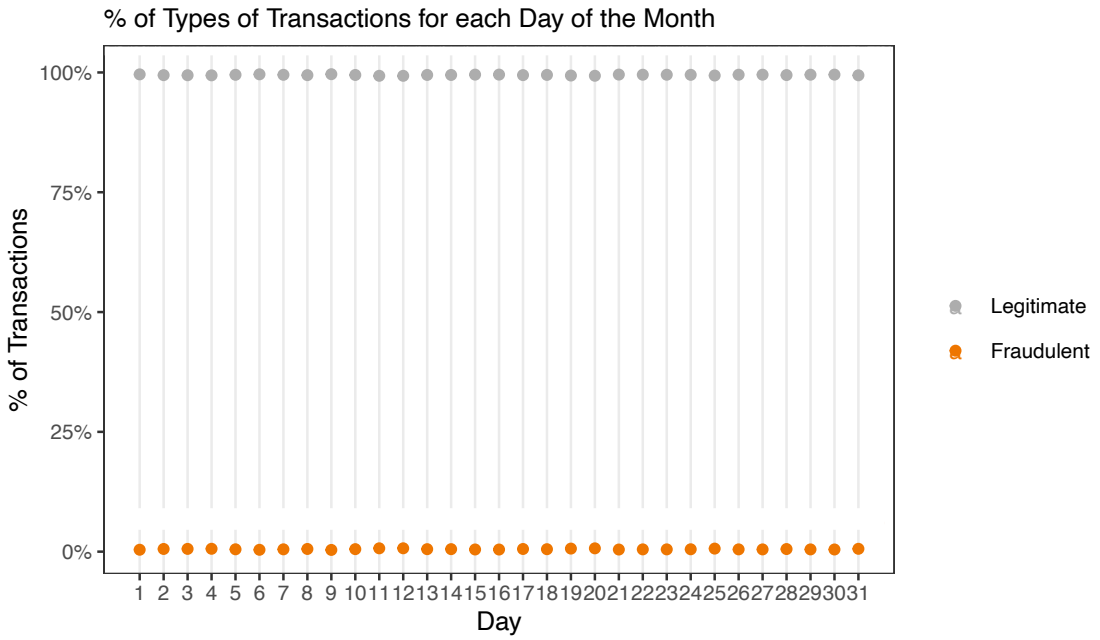
While the monthly trend in the plot above is less obvious than the hourly trend, we can still clearly see that there are higher percentages of fraudulent transaction amounts in the months of January, February and May.



The trend for the monthly proportion for the types of transactions is even harder to pinpoint than the previous plot. However, there are some obvious months where the proportion for fraudulent transactions are higher, like in January and February.

Next, let us take a look at how certain days of a month relate to fraudulent transactions.



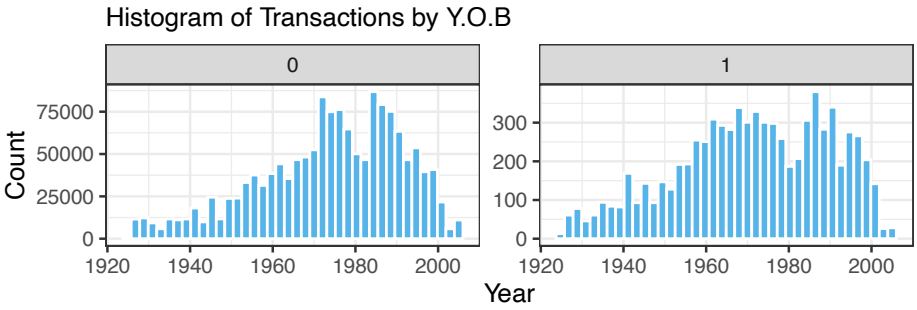


The trend for the proportion of types of transactions for each day of the month is difficult to spot. In the plot for proportion of fraudulent transaction amounts for each day of the month, we can see that some days have higher proportions of fraud.

While there may be a slight trend, it is not obvious and just like the monthly visualizations, it seems that the best date time predictor for fraudulent transactions is the hourly timescale.

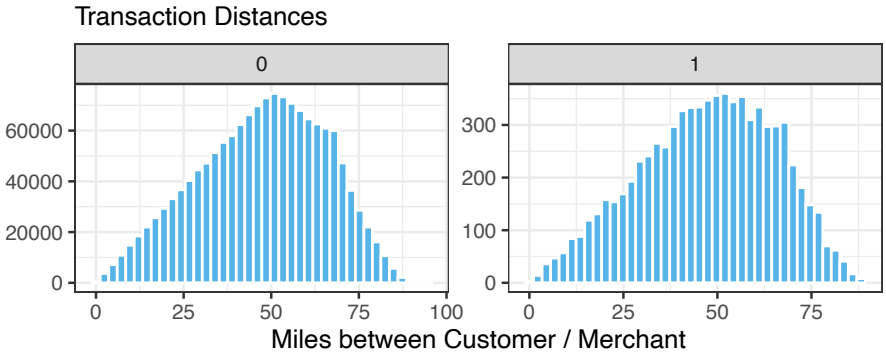
Similarly, there are variables with zero predictive power as well. For instance, neither gender nor age appear to be predictors for fraudulent transactions.

is_fraud	gender	amt	n	pct_amt	pct_n
1	F	1932903	3914	47	51
1	M	2199907	3815	53	49

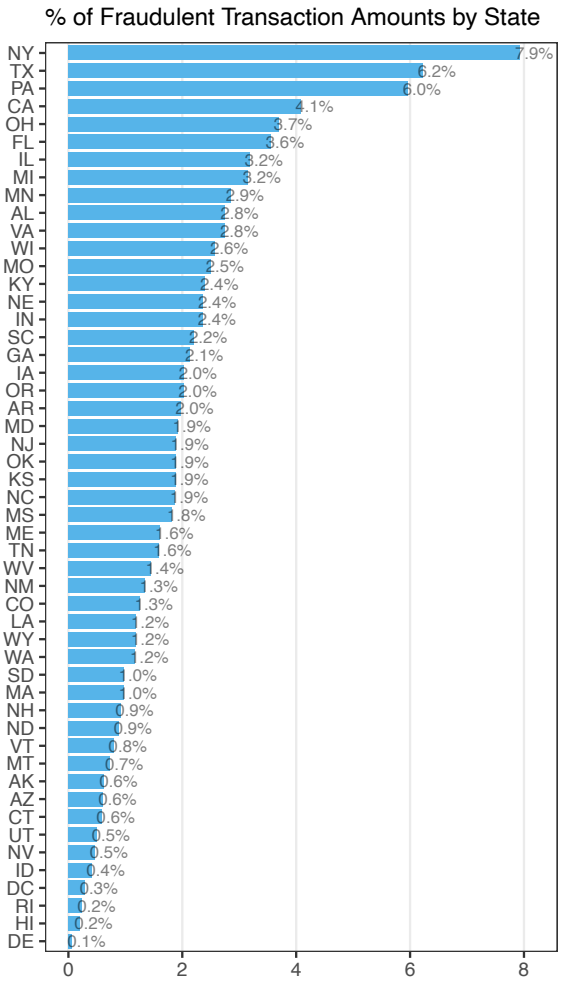


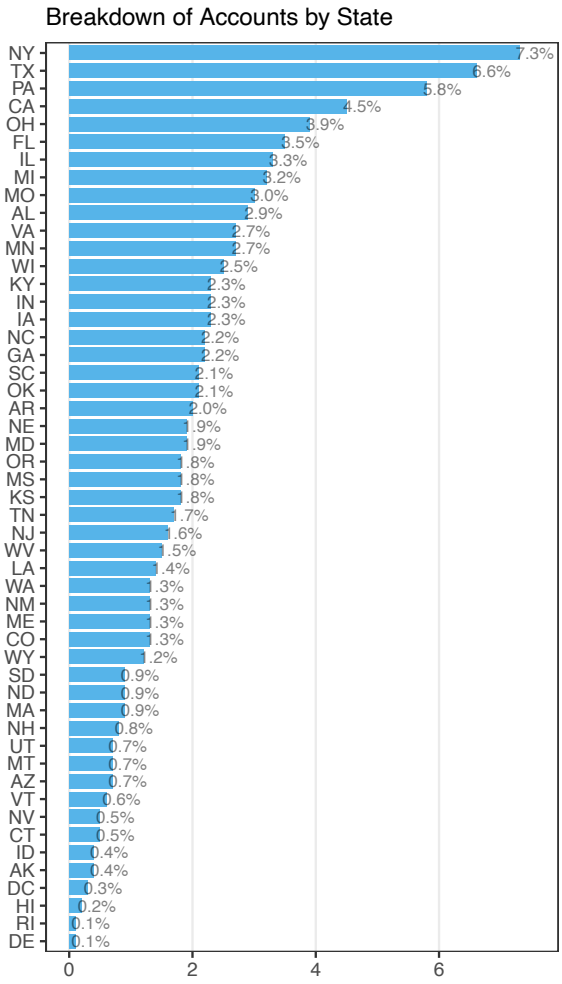
In most cases, fraudulent transactions are often between merchants and customers who are very far from one another. Using the longitude and latitude data for both the customer and the merchant, we can calculate and analyze distance.

In this synthetic dataset, all transactions are within 100 miles of the customer address, and no trend for fraud can be can be ascertained. Transaction distances Range: Min: 0.01 Max: 94.65



Lastly when looking at the breakdown of fraud by customer state, fraud rates are consistent with percent of accounts by state.





Data Preparation

Before building the models, the dataset needs to be prepped to include the relevant predictors. To reduce the size of the dataset, unused columns will be removed and the date and time elements converted into features. Predictors like amount, category, transaction date and time (hour, month, day of month, and weekday) will be kept. The merchant and cc_num will be kept as possible predictors.

Unfortunately, the synthetic card numbers were created fictionally which reduced their utility for modeling. In reality card numbers are between 13 to 16 digits. The first digit represents the card type and digits two through six identify the institution and the final digits are unique account ids. The credit card numbers in this dataset are between 11 to 19 digits and do not follow convention. Therefore, the cc_num will be converted to a character variable to be handled appropriately by the algorithms. It is important to note that by using unique identifiers such as account numbers, it can lead to over training. Since frauds are usually repeated, cc_num has usable predicting power.

To effectively train and tune models, it is necessary to partition the training set to avoid over-training. A 90/10 train/test split will be used to include as much of the data as possible since there are so few fraudulent transactions.

```
##
##         0         1
```

```
## 1326761    6956

##
##       0       1
## 147418    773
```

Modeling Methods

Three algorithms best for anomaly predictions will be presented: two Classification models and a Regression Tree (CART) model. The models used for this are rpart, randomForest and logistic regression model using glm. CART algorithms work by predicting an outcome or classification and are commonly used in fraud detection. Logistic regression models use linear regression to determine the probability of a binary outcome.

Rpart:
Rpart creates a decision tree through Recursive PARTitioning to predict the class of the target variable. Rpart repeatedly subsets predictors into non-overlapping regions (partitions) at decision nodes which create the largest and most uniform subset. This can be described as partition $\mathbf{x}$, predictor j, and value s where rpart splits observations into two regions $R_1(j,s)$ and $R_2(j,s)$. Mathematically represented as:

$$R_1(j,s) = \{\mathbf{x} \mid x_j < s\} \quad and \quad R_2(j,s) = \{\mathbf{x} \mid x_j \geq s\}$$

Rpart chooses j and s which minimize the residual sum of squares (RSS). Partitioning continues until minimum value of improvement in RSS, referred to as complexity parameter (cp), is reached. Rparts cp default is .01 but can be tuned.[1][2]

Random Forest:
RandomForest is an ensemble CART algorithm which creates large numbers of decision trees with different subsets of variables then aggregates the predictions. The algorithm builds B trees resulting in models $T_1, T_2, ..., T_B$. For each observation randomForest, predicts $\hat{y}_j$ from T_j. In a classification outcome, prediction $\hat{y}$ is the majority vote among $\hat{y}_1, ...\hat{y}_T$. Both the number of trees (ntree) and number of variables to use per tree (mtry) are editable parameters and can be tuned. The defaults are 500 trees and square root of number of variables.[3][4]

Logistic Regression:
The glm function will be used to compute a logistic regression model which predicts the conditional probability of an outcome: $Pr(Y = 1|X = x)$. To ensure the estimate is between 0 to 1, glm's family function is set l to binomial to apply the logit transformation, $g(p) = log\frac{p}{1-p}$. This will create a regression model:

$$g\{p(x_1, x_2, .., x_n)\} = g\{Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2, ..., X_n = x_n)\} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

After the model fits estimates for $\beta_0 + \beta_1 x_1 + ... + \beta_n x_n$, the predict.glm function calculates the conditional probabilities. To obtain a prediction, I must define a decision rule to produce a vector of predicted outcomes based on the threshold (such as $\hat{y} > .5$).

Logistic regression is limited in its modeling capability. Simply stated, glm calculates the relationship between features and outcome on a linear plane which means it cannot model non-linear relationships.[5] Furthermore, glm cannot handle categorical variables with many levels. Although a limited and simplistic approach, glm does allow the algorithm to model interaction between variables. Based on the above analysis, the relationship between category and amount appears to be the best indicator of fraud in the data. In this case the equation will change to include an coefficient for the interaction for amount and category:

$$g\{p(amt, cat)\} = \beta_0 + \beta_1 amt + \beta_2 cat + \beta_3 amt * cat$$

.

Modeling Issues: The caret function was used to train all models by utilizing its cross-validation feature. Unfortunately, with over a million observations, caret took hours to run and did not significantly improve the model as compared to rpart and glm.

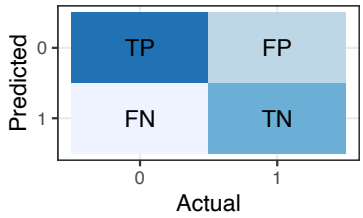
Model Evaluation

Accuracy is a poor evaluation metric for imbalanced datasets. For instance, this dataset contains only 0.5% fraudulent transactions, even a model that predicts no fraud will have a 99.5% accuracy. Credit card companies utilize fraud detection algorithms to prevent revenue loss. As shown during analysis, the percentage of fraudulent transaction amounts (cost) is eight times higher than the number of fraudulent transactions. In real life, fraudulent charges also produce additional customer service costs.[6] Therefore, a combination of confusion matrix metrics and cost analysis will be used to evaluate model performance.

Predictions have four possibilities in the following:

- True Positive (TP): legitimate predicted for actual legitimate transaction
- False Positive (FP): legitimate predicted for actual fraudulent transaction
- False Negative (FN): fraud predicted for actual legitimate transaction
- True Negative (TN): fraud predicted for actual fraudulent transaction

In the confusion matrix of a fraud detection model, is_fraud = 0 is a legitimate transaction (positive outcome), and is_fraud = 1 is fraudulent (negative outcome).



Evaluation Metrics:

- Specificity: The proportion correct fraud predictions to actual fraud, also called True Negative Rate (TNR). Specificity in an imbalanced dataset is a better metric than accuracy. The formula is as shown: $TN/(TN + FP)$.
- Negative Predictive Value (NPV): The proportion correct fraud predictions to all fraud predicted. NPV shows if the model is incorrectly identifying legitimate transactions. The formula is as shown: $TN/(TN + FN)$.

Costs:

- Amount Saved: Amount ($) of fraud correctly predicted (TN)
- Fraud Missed: Amount ($) of fraud missed or loss (FP)
- Misclassified: Amount ($) incorrectly predicted as fraud (FN)

Model Building

Firstly, lost revenue when no fraud is detected is calculated. Three versions of each algorithm with different variables and parameters will be evaluated. The best performing construct of each version will be chosen for the final evaluation.

No Fraud Predicted: No fraud was predicted, all positive outcomes assumed. (All is_fraud = 0.)

The cost of not detecting fraud = $ 421382.01.
Accuracy with no correct fraud predictions: 99.48 %.

Rpart Models

Rpart Model 1: To begin, all possible predictors will be included. The results and variable importance will also be accessed.

Rpart Model 1 Confusion Matrix:

	0	1
0	147358	308
1	60	465

Rpart Model 1 Results:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Rpart Model 1	285986	135396	53199	0.68	0.13	0.6	0.89

The results are not very impressive with only 67.9% of fraud amounts saved and just 60.2% actual fraudulent transactions detected. NPV result is 88.6% which shows that the model does a fairly decent job at not incorrectly flagging legitimate transactions as fraud.

One of the benefits of rpart is its ease of interpretability when plotting the decision tree. Unfortunately, a model with a high level of classifiers such as this does not make a readable decision tree. Instead, evaluation of variable importance will be used instead.

Rpart Model 1 Variable Importance:

merchant	3327.6
bins	2654.6
amt	2513.4
cc_num	2435.4
category	2367.8
hour	1183.5
day	1.3

It is surprising that merchant is at the top the list and that category and bins are improving the model more than amount. This may be because the high-level categorical variables are not performing well. By including a constructed feature like bins, rpart's ability to calculate best splits is most likely inhibited.

Rpart Model 2: For a simpler model, only amount, category and date parts are included as predictors. Binning is not included in order to allow rpart partitioning to calculate best split value.

Rpart Model 2 Confusion Matrix:

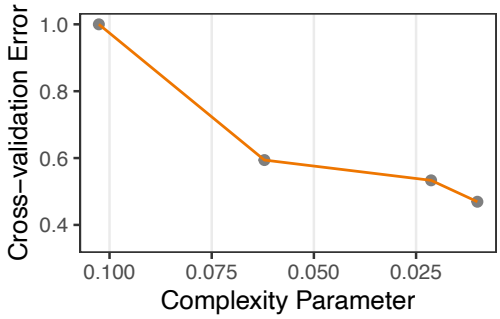
	0	1
0	147322	262
1	96	511

Rpart Model 2 Results:

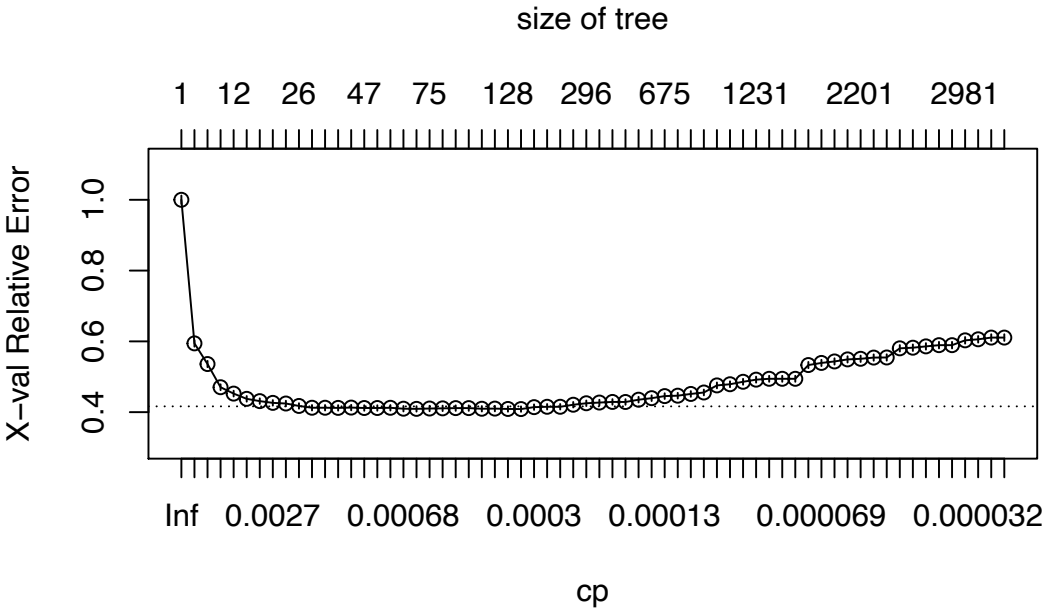
Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Rpart Model 2	324320	97062	79829	0.77	0.19	0.66	0.84

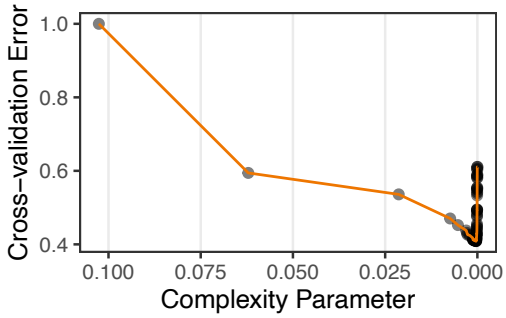
This is a significant improvement over the first model with 77.0% of fraudulent transaction amounts detected and specificity of 66.1%. With such a large dataset and high-level variables, should the complexity parameter (cp) be tuned to improve the model?

When we plot the cp against the cross-validation error, it does appear that we could improve the model by decreasing the complexity parameter.[7]



Rpart Model 2 Tuning CP: Setting minimum split and complexity parameters to zero to determine which cp value minimizes the cross-validation error (xerror), then prune the model accordingly with prune.rpart function.





It appears that the xerror is minimized much below the complexity parameter default of 0.01 and there are multiple xerror values below 0.4.

Rpart Model 2 & Rpart Model 2 Tuned CP Confusion Matrices Compared:

	Rpart Model 2		Rpart Model 2: Tuned	
	0	1	0	1
0	147322	262	147343	240
1	96	511	75	533

Rpart Model 2 Tuned CP Results Compared:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Rpart Model 2	324320	97062	79829	0.77	0.19	0.66	0.84
Rpart Model 2: Tuned	334396	86986	58461	0.79	0.14	0.69	0.88

Tuning the complexity parameter slightly improved the saved amount by over $10,000. In proportion wise, the tuned model helped saved 79.4% of all fraudulent transaction amounts. It had a much larger impact on reducing misclassified (false negative) amounts by over $20,000.

Rpart Model 3: Let's run the model with cc_num. In most models using a unique identifier is not advised. However, in this case, the cc_num has thousands of observations associated to it. Furthermore, because of the nature of fraud charges where fraudulent transactions are more likely to repeat on the same credit card, identifying fraud on a cc_num could be a valid predictor.

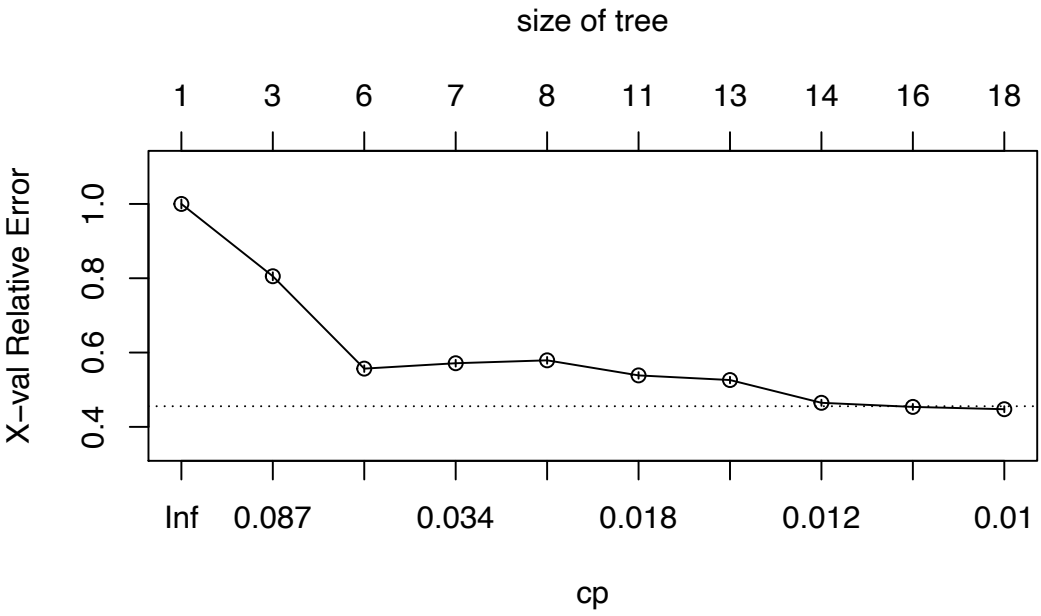
Rpart Model 2 & 3 Confusion Matrices Compared:

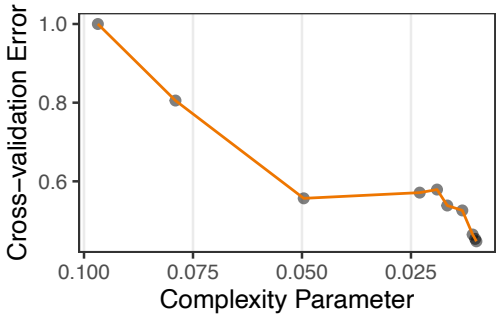
	Rpart Model 2		Rpart Model 3: CC	
	0	1	0	1
0	147322	262	147352	277
1	96	511	66	496

Rpart Model 2 & 3 Results Compared:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Rpart Model 2	324320	97062	79829	0.77	0.19	0.66	0.84

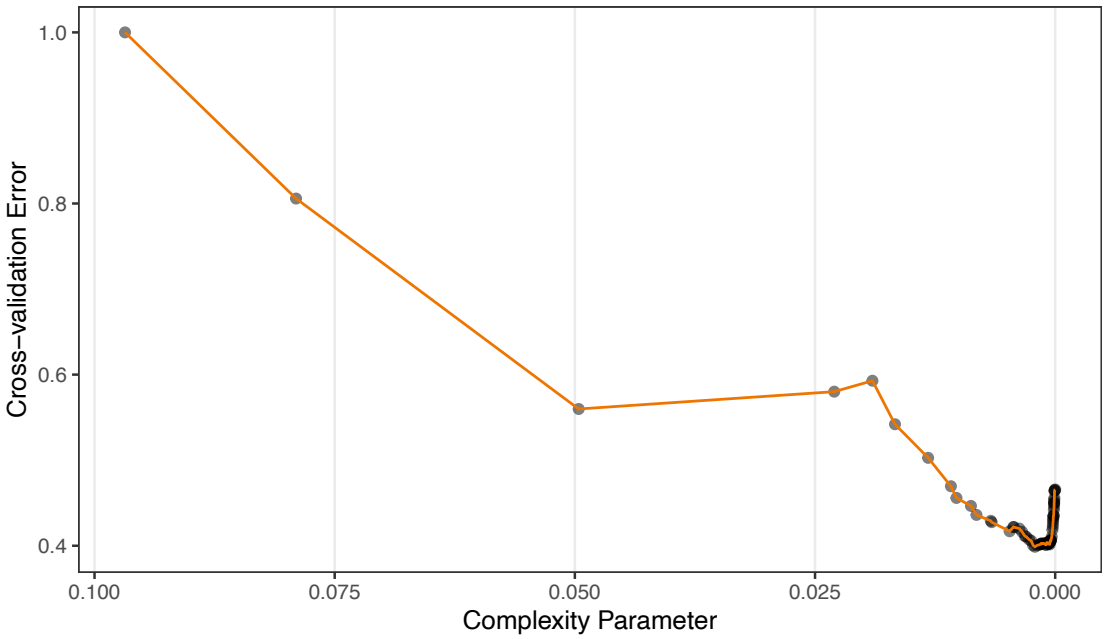
The rpart model with credit card numbers performed close to but not as good as the Rpart Model 2 which included amount, category and data variables as predictors. By plotting the complexity parameter again, we can show that the default cp is not optimized.





The rpart model with credit card numbers performed close to but not as good as the Rpart Model 2 with amount, category and data variables as predictors. Plotting the complexity parameter again, shows the error was still decreasing when the cp parameter was reached.

Rpart Model 3 Tuning CP: Setting minimum split and complexity parameters to zero to determine which cp value minimizes the cross-validation error (xerror), then prune the model accordingly with prune.rpart function.



Rpart Model 3 & Rpart Model 3 CP Tuned Confusion Matrices Compared:

	Rpart Model 3: CC		Rpart Model 3: Tuned	
	0	1	0	1
0	147352	277	147362	253
1	66	496	56	520

Rpart Model 3 CC & 3 Tuned CP Results Compared:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Rpart Model 3: CC	311314	110068	79246	0.74	0.19	0.64	0.88
Rpart Model 3: Tuned	323673	97709	64228	0.77	0.15	0.67	0.90

Unsurprisingly, the tuned model with card numbers performed the better than the untuned model with 76.8% of fraudulent transaction amounts saved. However, it still did not beat the tuned Rpart Model 2 with amount, category and date variables as predictors. Therefore, the Rpart Model 2 with amount, category and date variables as predictors will be used for the final validation and model comparison.

GLM Models

To begin, two models will be compared to emphasize the limitations of glm models in machine learning. Since glm cannot handle high levels of categorical variables, merchant and cc numbers cannot be used. With glm models, we can first create the fit model then calculate probability estimates. Finally, a vector of predicted outcomes based on a threshold can then be created.

GLM Model 1: The first model will include amount and category interaction along with date variables as factors. It does not include the additional calculated bins variable. Predicted outcomes based on the threshold > 0.5.

GLM Model 2: The second model will be a multivariate linear model including the calculated feature bins but not modeling for any interaction.

GLM Models 1 & 2 Confusion Matrices:

	GLM Model 1: category * amount		GLM Model 2: category + amount	
	0	1	0	1
0	147340	365	147330	368
1	78	408	88	405

GLM Models 1 & 2 Results:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
GLM Model 1: category * amount	224717	196665	164152	0.53	0.39	0.53	0.84
GLM Model 2: category + amount	288976	132406	77052	0.69	0.18	0.52	0.82

Even with interaction between category and amount, GLM Model 1 did not perform very well. GLM Model 2 without interaction performed better when a feature (i.e. bins) was introduced to capture the differences in fraud amounts in different categories.

Linear models will only estimate relationships of the features provided. Rpart performed better without the added bin construct because the algorithm calculates its own splits. For glm we must know our data well and provide appropriate predictors that best fit the algorithm and data structure.

GLM Model 3: This model will include amount/category/bin interactions plus date variables as factors. The model's estimates will then be evaluated at thresholds of 0.5 and 0.4 for comparison.

GLM Model 3 Confusion Matrices:

	GLM Model 3: > 0.5		GLM Model 3: > 0.4	
	0	1	0	1
0	147354	237	147328	211
1	64	536	90	562

GLM Model 3 Results:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
GLM Model 3: > 0.5	336038	85345	55266	0.80	0.13	0.69	0.89
GLM Model 3: > 0.4	357158	64224	71679	0.85	0.17	0.73	0.86

Reducing the probability threshold increased the number of correct fraud predictions and saved about $18,000 more, but it increased false positives (i.e. misclassified) by more than $20,000. This is where companies must decide on a trade-off. Is it better to catch more fraud at the risk of denying some legitimate transactions and possibly upsetting customers and losing customers. With today's automation, banks can send a text allowing the cardholder to approve or deny the suspicious transaction. Therefore, this report strongly believes that it is more important to reduce false negative (i.e. fraud undetected/missed) than false positives.

RandomForest Models

RandomForest Model 1: The main tuning parameter for random forest are the number of trees (ntree) and the number of variables to sample per tree (mtry). RandomForest defaults to 500 trees and the square root of the number of columns in the formula. Sampling can be done with or without replacement although more randomness will be generated with replacement. Due to machine memory limits, A very low number of trees (51) is opted.

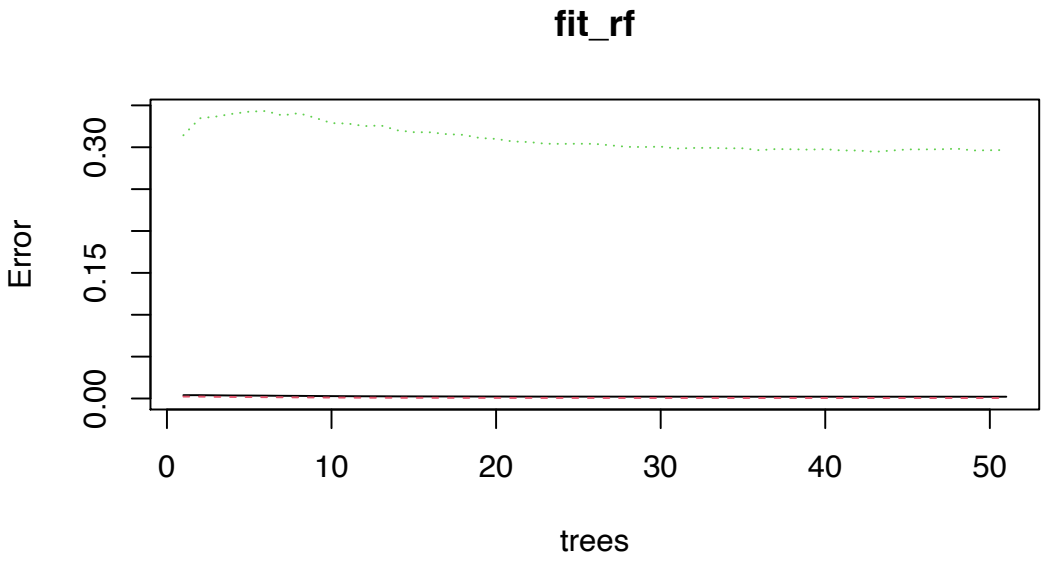
RandomForest Model 1 Confusion Matrix:

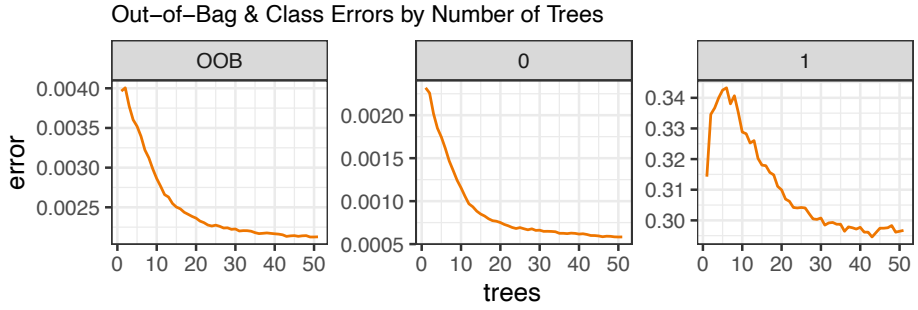
	0	1
0	147351	234
1	67	539

RandomForest Model 1 Results:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Random Forest Model 1	343851	77531	59419	0.82	0.14	0.7	0.89

The first randomForest model performed extremely well even with only 51 trees. Plotting the error rate by class vs number of trees to see if there is room for improvement.[8]

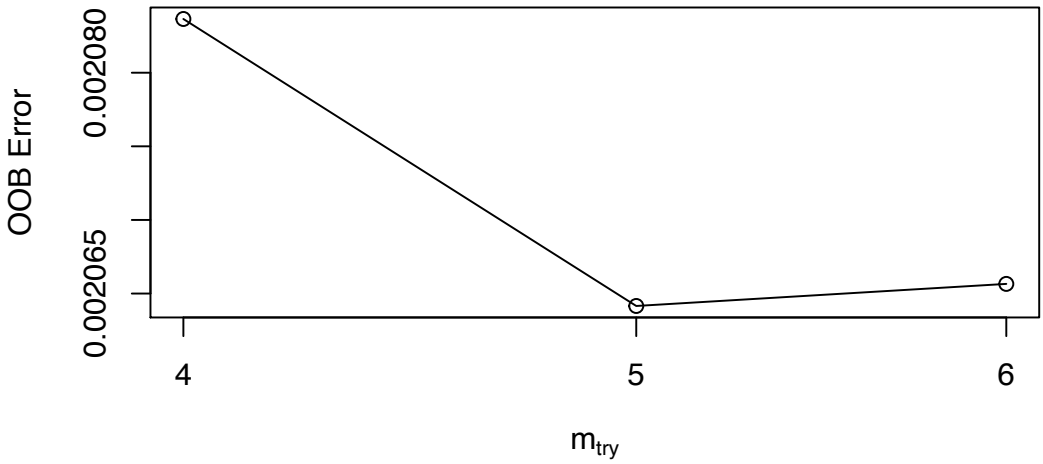


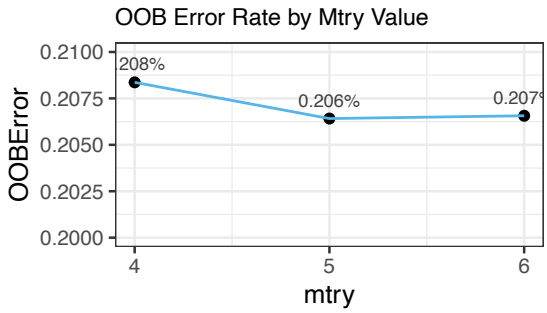


At mtry = 3 the minimum OOBError = 0.21 % and Fraud Class Error = 29.46 %. It would be interesting to look at error rates for higher values of mtry to see if the results can be improved. In an imbalanced dataset, as with accuracy, OOB error is not particularly helpful as it is skewed to the majority class. In the graph above, you can see that legitimate transactions are gradually plateauing after 30 trees, but the plot for fraudulent transactions is still declining and has a much higher error rate.

RandomForest Model 1 Tuning mtry: The tuneRF function will be used to tune the model. The tuneRF function takes in a starting mtry input and returns the OOB error for a step factor above and below. The number of trees will also be increased slightly in this model.

```
## mtry = 5  OOB error = 0.21%
## Searching left ...
## mtry = 6     OOB error = 0.21%
## -0.00073 0.05
## Searching right ...
## mtry = 4     OOB error = 0.21%
## -0.0094 0.05
```





In actuality, we are more interested in the err.rate for fraud class and the cost results than the OOB Error. However, this does show that different values of mtry do perform better than the default. Using more trees

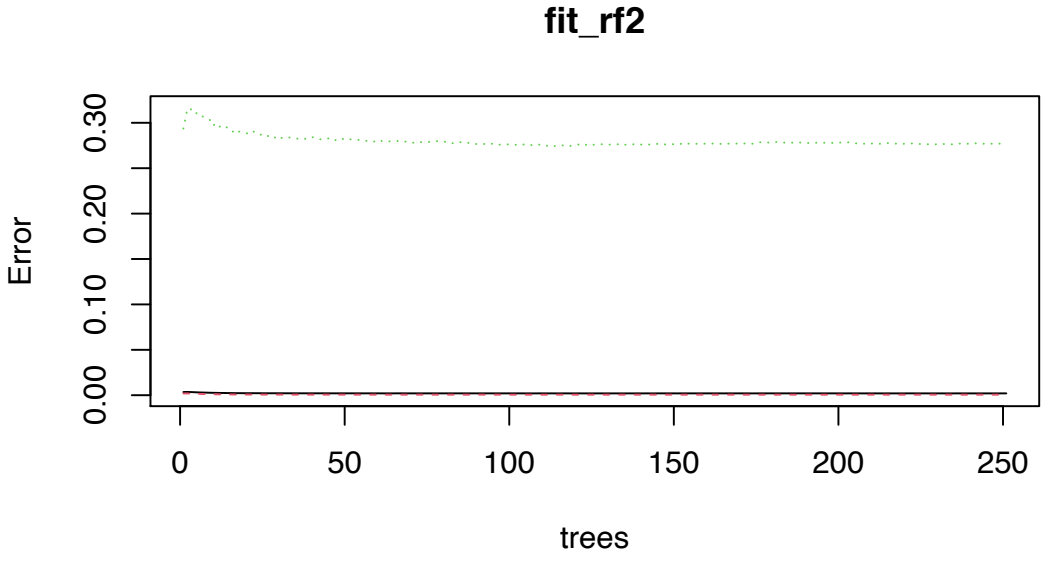
would produce better results, but tuneRF is a very time consuming function. Since the OOBError value changes only at a hundredth of a percent, these models should produce very similar values. Using a smaller number of predictors is supposed to allow randomForest to pick up trends between predictors that would be less noticeable with the full set. Unfortunately, due to technical restrictions and computational limitations, the number of trees that can ran is restricted and this will inhibit the performance and also reproducibility of the results. Therefore, the next randomForest model will be trained with only 251 trees (just over half of the default) and use mtry = 4.

RandomForest Model 2 Confusion Matrix:

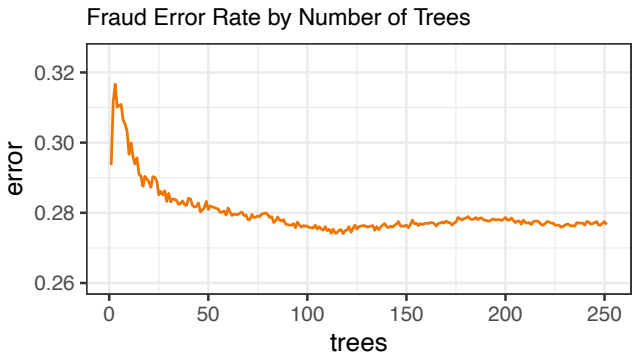
	0	1
0	147403	80
1	15	693

RandomForest Model 1 & 2 Results:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
Random Forest Model 1	343851	77531.0	59419.1	0.82	0.14	0.7	0.89
Random Forest Model 2	395932	25450.3	12741.6	0.94	0.03	0.9	0.98



Unsurprisingly, the randomforest models performed the best. It was also interesting to see that by increasing the variables per tree, false fraud predictions was reduced. It also slightly reduced the number of correct predictions and missed fraud. Looking at the error rate for the model:



Adding more trees could still improve the model, but the error rate appears to be stabilizing. Therefore, the RandomForest Model 2 with 251 trees will be used for the final validation.

Note on Variation of RandomForest Results: Even with setting a seed, there is still randomness introduced in the algorithm and the results may change slightly wand differ from run to run (setting the seed did create reproducible models results when ran on the same day). During testing, multiple randomForest 251 models with mtry of 4 and 5 were ran. The code and results were not included to save time, especially when this is a already lengthy project. The mtry = 5 models had a lot of variation in their results and class 1 error. They sometimes performed thousands of dollars better or worse. The mtry = 4 models had less variation in both results and class error. A possible assumption could be that the mtry = 5 sometimes picked up very good trends and which caused over-fitting of the trees. This meant that while 251 trees can produce very good results, the number of trees is too low to fit a stable model. Unfortunately, due to computational limitations, rf models with large number of trees were not feasible. However, there are rf models online with 1000 and 1500 trees which produced much stable and better results!

Final Validation

Each of the validation set containing 20% of original data which is not used in training the models or data exploration, will be fed to the chosen models for final validation.

- Rpart with tuned complexity parameter
- GLM with interaction between category, amount, and bins with predicted outcomes > 0.40
- RandomForest with 251 trees sampling 4 predictors

Final Validation Results: Even with only half the default number of trees, random forest was able to perform the best and correctly predict 94.0% of the fraudulent transactions amounts. GLM was very close behind saving just $7000 less by choosing predictions > .4, but it had the most misclassified amount overall. Even though rpart saved the least amount, it performed best at not misclassifying legitimate transactions.

Final Models Confusion Matrices Comparison:

	Rpart		GLM		RandomForest	
	0	1	0	1	0	1
0	368358	583	368323	542	368471	252
1	198	1339	233	1380	85	1670

Final Models Cost Results Comparison:

Model	AmtSaved	FraudMissed	Misclassified	SavedPct	MisclassPct	Specificity	NPV
No Fraud Predicted	0	421382.0	0.0	0.00	0.00	0.00	0.00
Rpart Model 2: Tuned CP	782476	206127.4	162512.8	0.79	0.16	0.70	0.87
GLM Model 3: > 0.4	813620	174983.2	182611.7	0.82	0.18	0.72	0.86
Random Forest Model 2	925690	62913.4	68710.1	0.94	0.07	0.87	0.95

Conclusion

With fraud detection, companies must decide on a balance between true positives (specificity), false positives (misclassified), false negatives and the resulting costs. As seen in the confusion matrices, adjusting models to increase the number of true fraud predictions often impacts missed fraud predictions and false fraud predictions. More importantly revenue saved or lost can vary more dramatically than the confusion matrix results show. These models could continue to be tuned and improved, Unfortunately, and with only 251 trees randomForest does not return consistent results. Even this limited model, would have saved my synthetic credit card company about 83% which I think is a pretty successful beginning to credit card fraud detection model.

Although this was a synthetic dataset without real-life predictors and cannot be used as an actual fraud detection model, many insights are able to be gained. The size of the dataset did cause complications. Larger processing capability and memory would improve modeling. Also, additional cost-sensitive algorithms could be explored or synthetic over-sampling techniques such as SMOTE which may improve results. Instead of rpart the party package could be used which models conditional probabilities more effectively.

In the end, even with this highly-imbalanced dataset and limited predictors, several models were created that had significant cost saving capabilities. Overall, I found this to be a very educational project into anomaly detection algorithms and effective metrics.

References

[1] Irizarry, Rafael. (2021). Machine Learning, Section 31.10: Classification and regression trees (CART). Introduction to Data Science.

[2] Therneau, T.M., Atkinson, E.J. (April 11, 2019). An Introduction to Recursive Partitioning Using the RPART Routines

[3] Irizarry, Rafael. (2021). Machine Learning, Section 31.11: Random forests. Introduction to Data Science.

[4] Liaw, Andy. (March 25, 2018). Package 'randomForest'.

[5] Irizarry, Rafael. (2021). Machine Learning, Section 31.3: Logistic Regression. Introduction to Data Science.

[6] Brownlee, Jason. (February 7, 2020). Cost-Sensitive Learning for Imbalanced Classification.

[7] Kabacoff, R.I. (2017). Tree Bases Models.

[8] Brownlee, Jason. (February 5, 2016). Tune Machine Learning Algorithms in R (random forest case study).