Credit Card Fraud Detection

HarvardX Final Capstone Project

K. H. Hogan

21 February 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

Even before downloading the data, I could see the variables are a mixture of date, categorical, numeric, and geospatial data. Examining the variables:

```
Rows: 1,296,675
Columns: 23
$ X1                    <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
$ trans_date_trans_time <dttm> 2019-01-01 00:00:18, 2019-01-01 00:00:44, 20...
$ cc_num                <dbl> 2703186189652095, 630423337322, 3885949205766...
$ merchant              <chr> "fraud_Rippin, Kub and Mann", "fraud_Heller, ...
$ category              <chr> "misc_net", "grocery_pos", "entertainment", "...
$ amt                   <dbl> 5.0, 107.2, 220.1, 45.0, 42.0, 94.6, 44.5, 71...
$ first                 <chr> "Jennifer", "Stephanie", "Edward", "Jeremy", ...
$ last                  <chr> "Banks", "Gill", "Sanchez", "White", "Garcia"...
$ gender                <chr> "F", "F", "M", "M", "M", "F", "F", "M", "F", ...
$ street                <chr> "561 Perry Cove", "43039 Riley Greens Suite 3...
$ city                  <chr> "Moravian Falls", "Orient", "Malad City", "Bo...
$ state                 <chr> "NC", "WA", "ID", "MT", "VA", "PA", "KS", "VA...
$ zip                   <dbl> 28654, 99160, 83252, 59632, 24433, 18917, 678...
$ lat                   <dbl> 36, 49, 42, 46, 38, 40, 38, 39, 40, 37, 41, 3...
$ long                  <dbl> -81, -118, -112, -112, -79, -75, -101, -79, -...
$ city_pop              <dbl> 3495, 149, 4154, 1939, 99, 2158, 2691, 6018, ...
$ job                   <chr> "Psychologist, counselling", "Special educati...
$ dob                   <date> 1988-03-09, 1978-06-21, 1962-01-19, 1967-01-...
$ trans_num             <chr> "0b242abb623afc578575680df30655b9", "1f76529f...
$ unix_time             <dbl> 1325376018, 1325376044, 1325376051, 132537607...
$ merch_lat             <dbl> 36, 49, 43, 47, 39, 41, 37, 39, 40, 37, 40, 4...
```

```
$ merch_long            <dbl> -82, -118, -112, -113, -79, -76, -100, -79, -...
$ is_fraud              <dbl> 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 19:12:22   1st Qu.:    180042946491000   1st Qu.:   10
 Median :2019-10-03 07:35:47   Median :   3521417320840000   Median :   48
 Mean   :2019-10-03 12:47:28   Mean   : 417192042080000000   Mean   :   70
 3rd Qu.:2020-01-28 15:02:55   3rd Qu.:   4642255475290000   3rd Qu.:   83
 Max.   :2020-06-21 12:13:37   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   :  88824   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
```

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 $80, 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. When I check the date ranges I see that the training and test sets were split by dates where the test set is the last six months.

Training set: 2019-01-01 00:00:18, 2020-06-21 12:13:37. 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. I will recombine data then repartition into 80/20 training and test splits based on random sampling method.

Once resampled, I have both training and test sets covering two years. Tabling the proportion of is_fraud for shows 0.52% fraud transactions.

```
     0      1
0.9948 0.0052
```

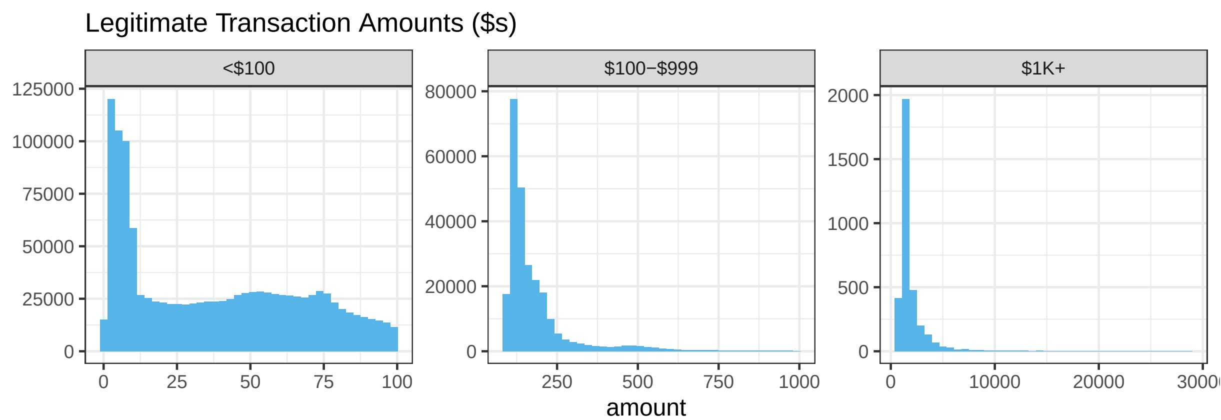
Now that the training set is repartitioned, I want to explore the difference between legitimate and fraudulent transactions to better understand to data.

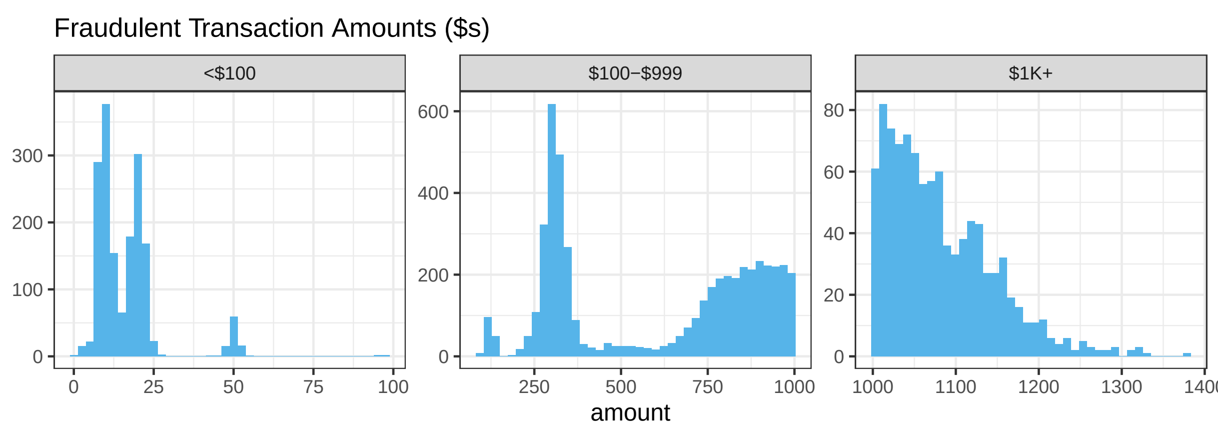
is_fraud	avg_trans	med_trans	amt	n	pct_amt	pct_n
0	68	47	99655913	1474187	96	99.48
1	534	419	4124923	7728	4	0.52

The cost of fraud in the training set is $4,124,923 and 4% of the total amount which is eight times higher than the percent of number of transactions. The average fraud transaction is much higher at $534 making amount a likely predictor. Since the average for fraudulent transaction is much higher, I need to investigate further and will summarize the transaction amounts into segments (bins).

bins	is_fraud	amt	n	pct_amt
<$100	0	45502779	1212517	99.94
<$100	1	27654	1697	0.06
$100-$999	0	47353151	258238	93.98
$100-$999	1	3030715	5045	6.02
$1K+	0	6799984	3432	86.44
$1K+	1	1066554	986	13.56

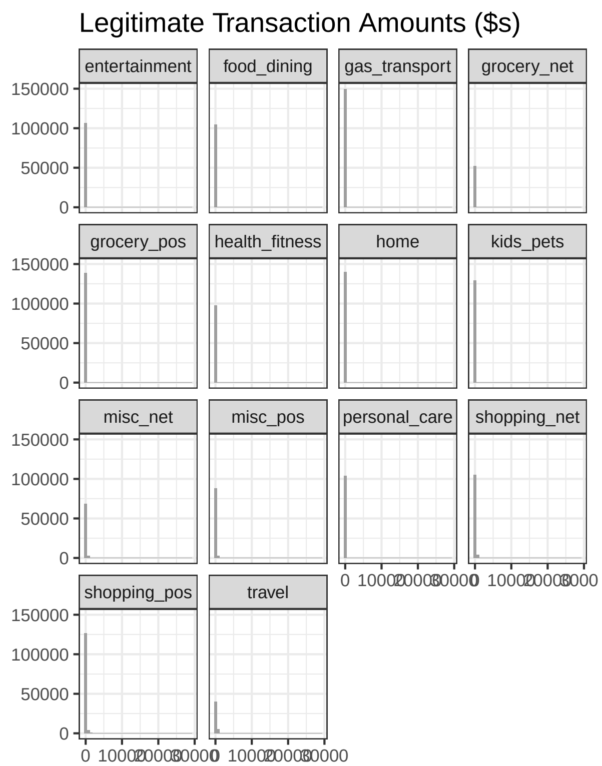
Since over 1.2 million transactions are under $100 this segment will overwhelm the visualization of the amount distribution if viewed together. When I segment transaction amounts into bins and allow the scales to float according to its bin, I can achieve a better understanding of that segment's distribution.

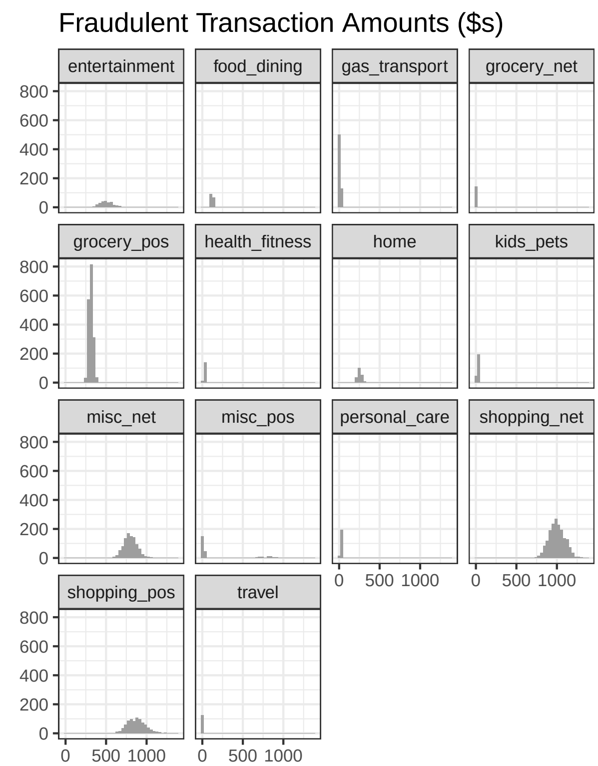




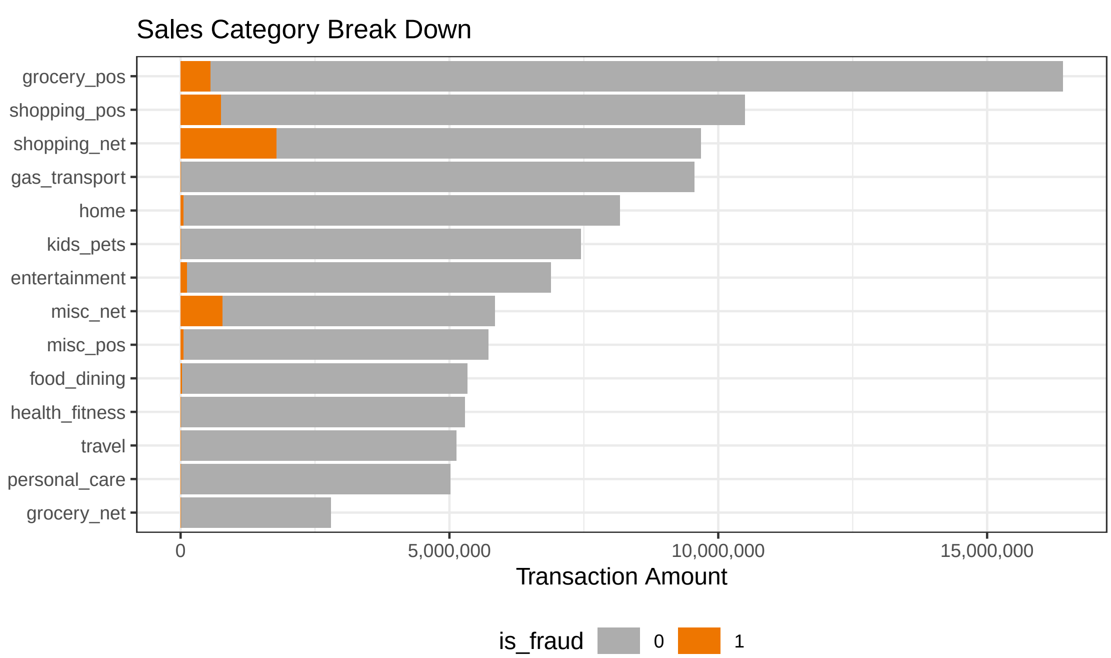
While the majority of legitimate transactions are under $100, 78% of fraudulent transactions are over $100. The amount of the transaction is definitely a predictor.

Next I want to explore categories. What are the actual transaction amounts by category, and how do they vary?

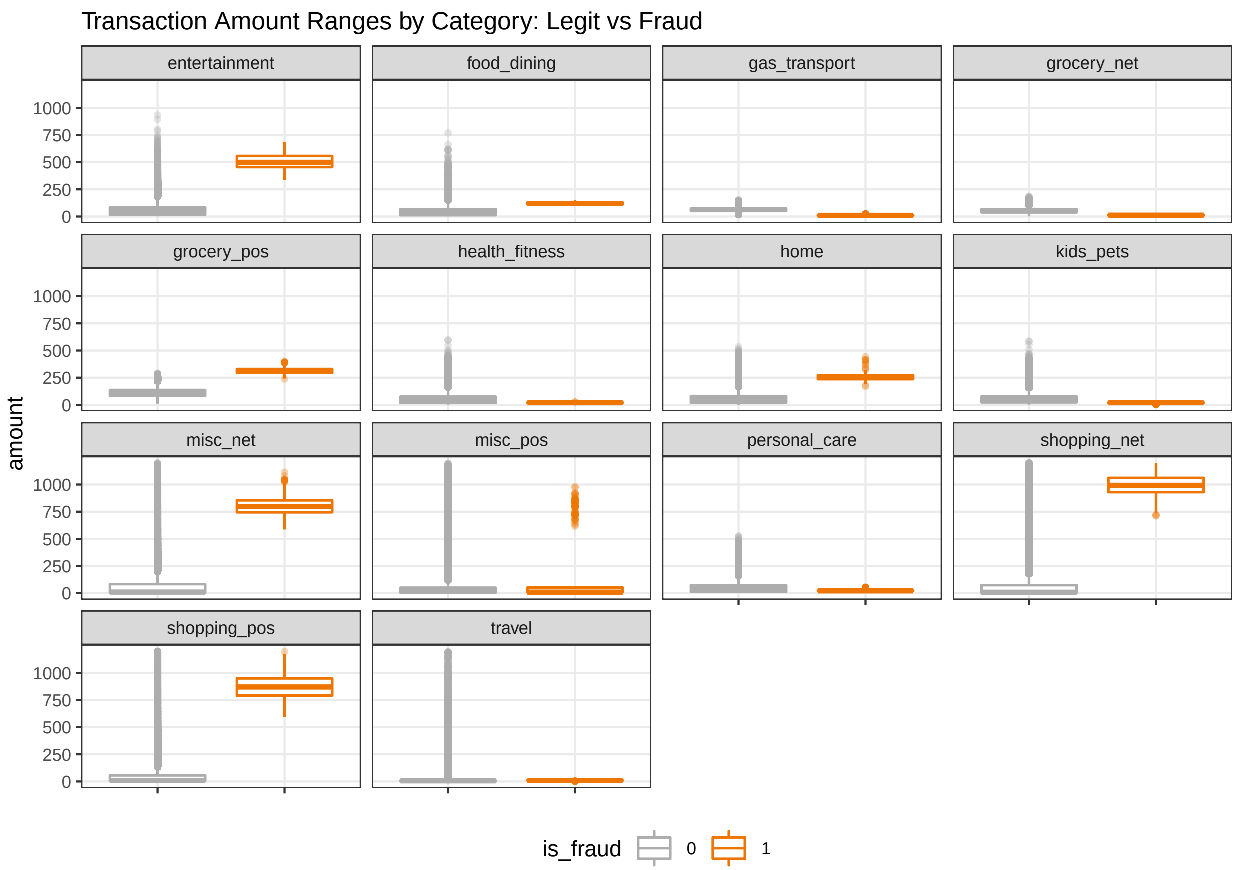


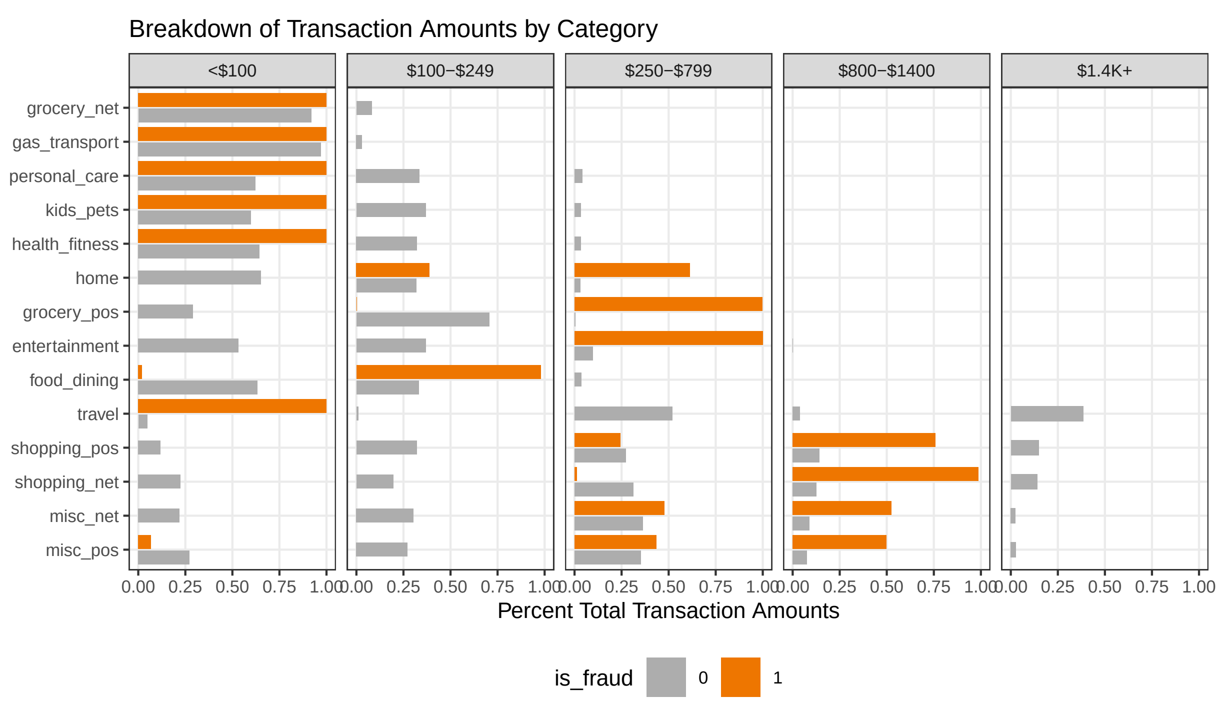


Category amount distribution differs drastically for fraudulent transactions but remains similar for otherwise.



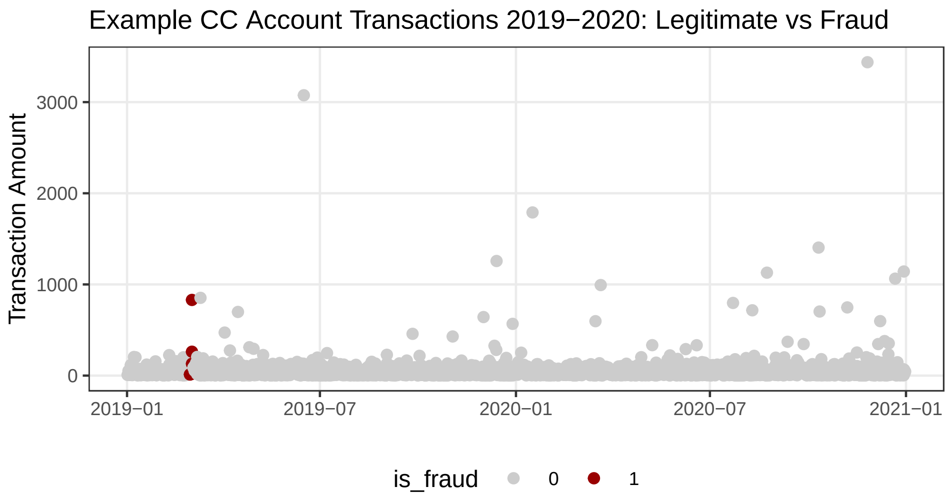
These charts show that fraud transactions are very to specific ranges within categories. Looking at the fraud amounts range by category, it is possible to use amount bins to better predict fraud.



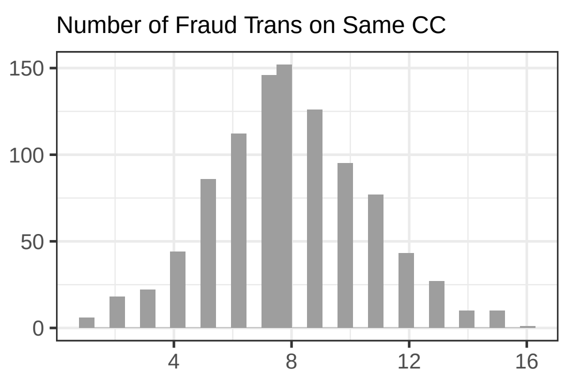


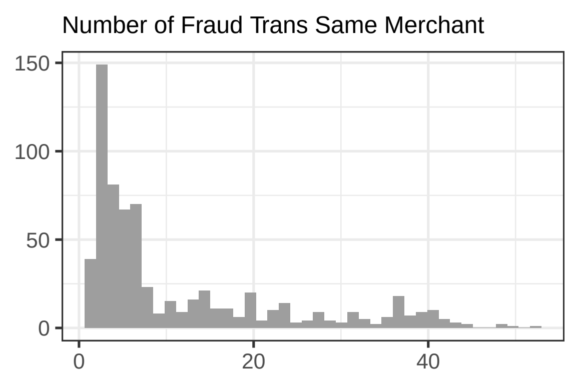
The transaction amount and category are very good predictors in this dataset.

I believe date and time elements will provide important insights. Visualizing all transactions on an account with fraud will help get an idea of fraud/legit timing.

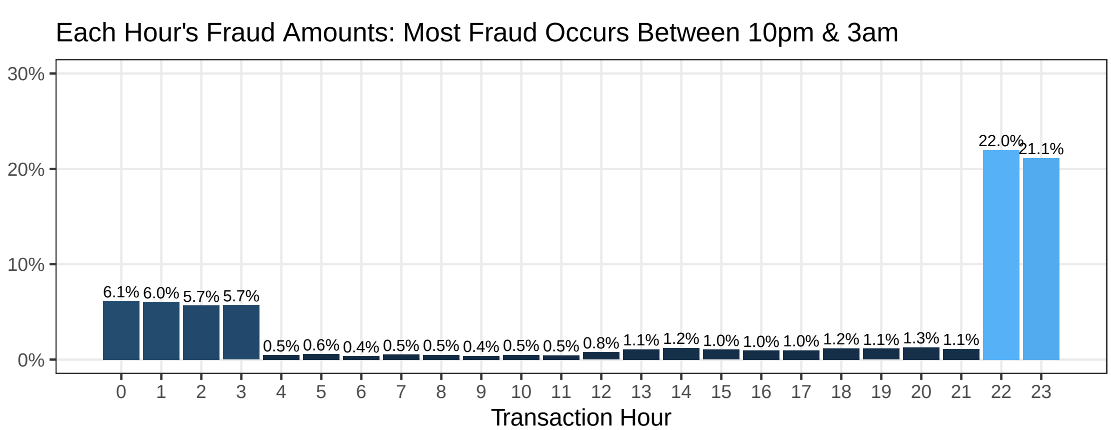


It appears that fraud happens quickly in groups. Timing and frequency both appear to be predictors. When looking at number of fraud charges by account and merchant, it is apparent that fraud is repeated significantly on both.

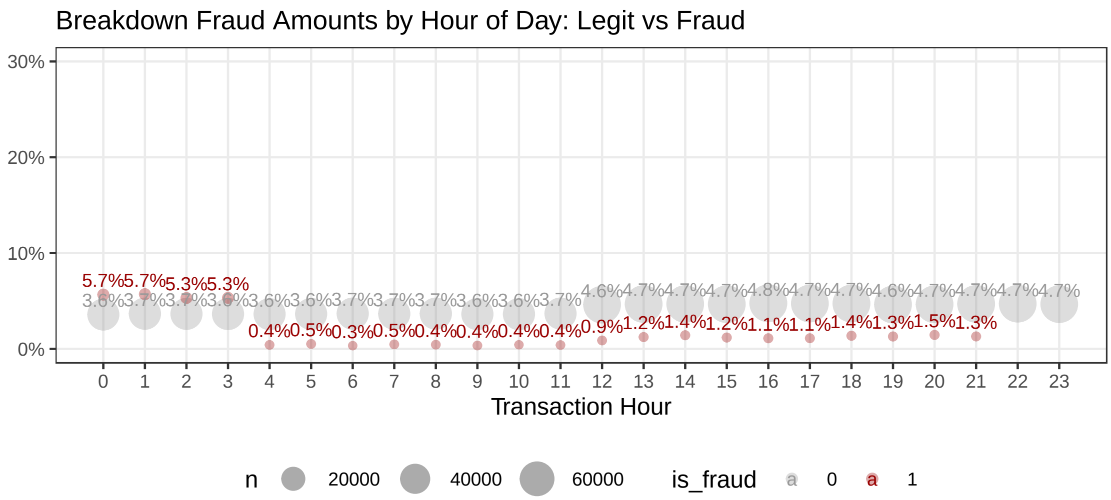


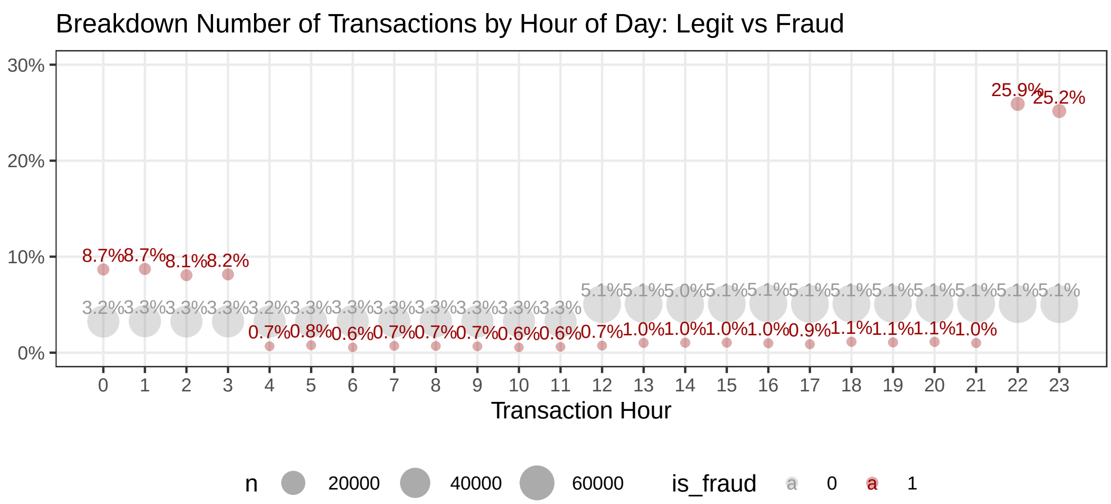


The trans_date_trans_time column should be explored in parts. First step is to explore transaction time. When viewing the percentage transaction amount for each hour, after 10pm has much higher rate of fraud.

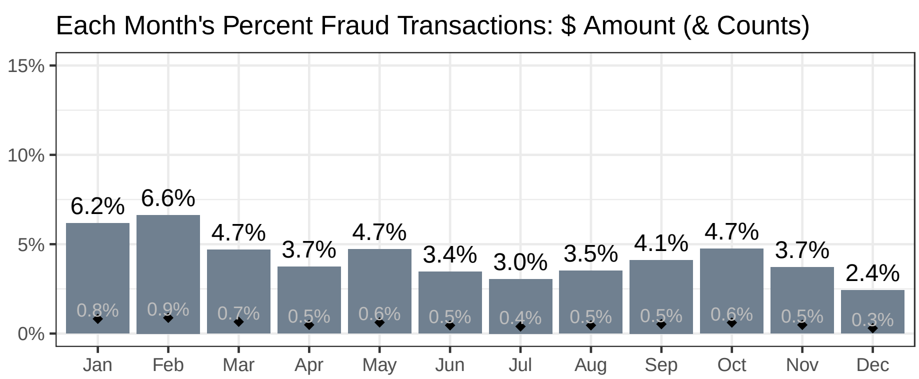


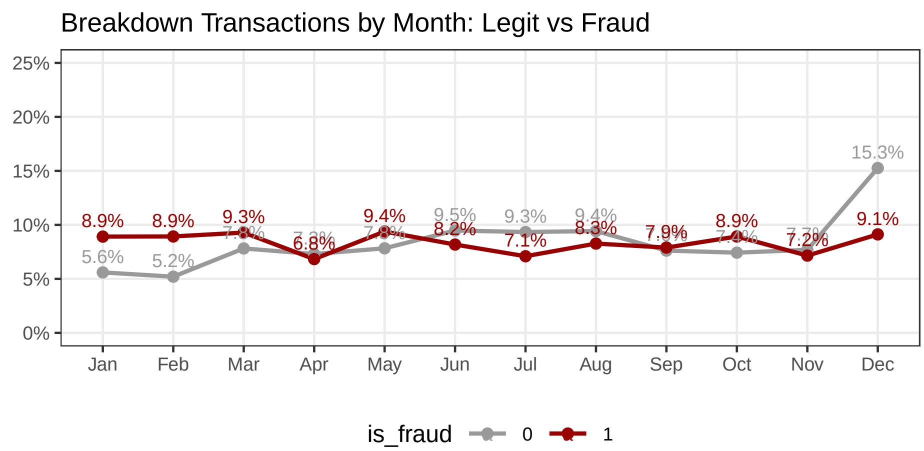
When viewing daily percentage transactions by hour (where 24 hours = 100%), it is clear that fraudulent transactions are mostly perpetuated late at night, whereas legitimate transactions stay at a steady rate.



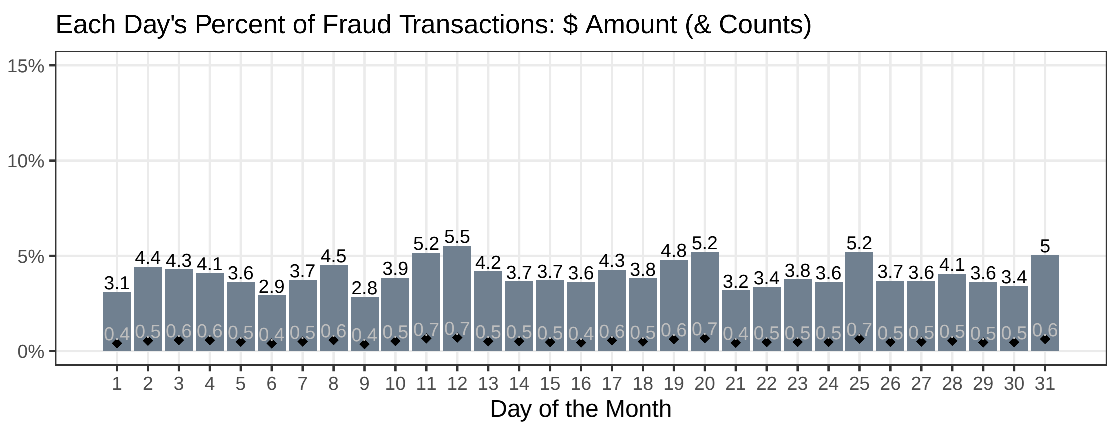


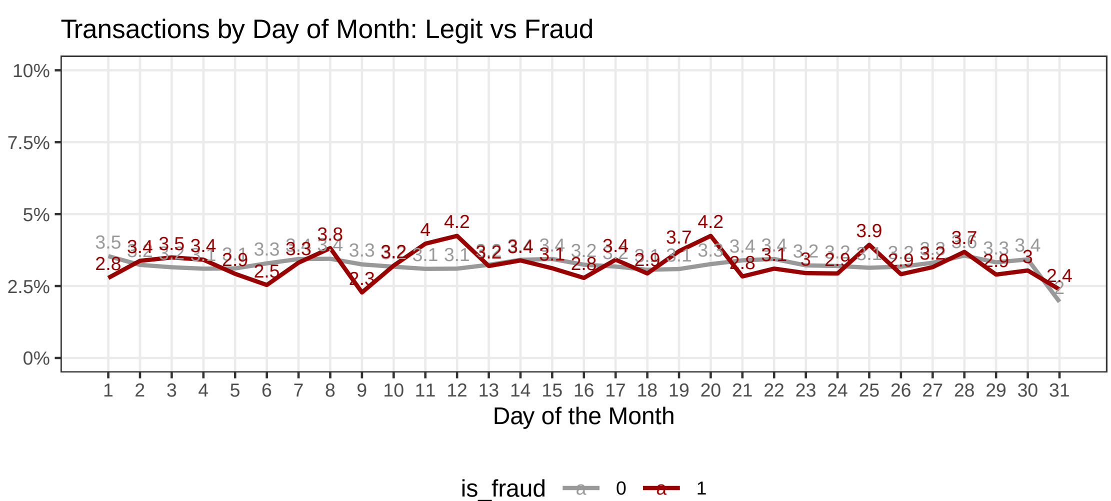
Next date elements are broken apart to look for trends. It is important to look for trends in fraudulent transactions within the date elements, and to check percentages of fraud vs legitimate charges in the date parts as well. Since fraud transaction amount rates are higher, I will focus of the percentage of $ amounts.



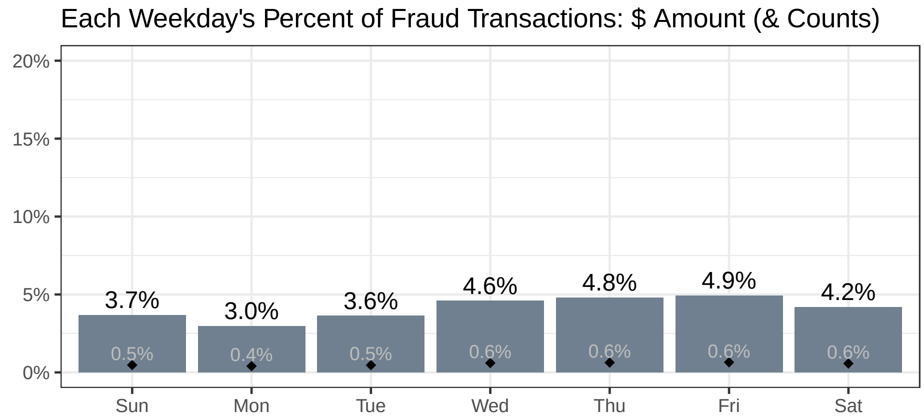


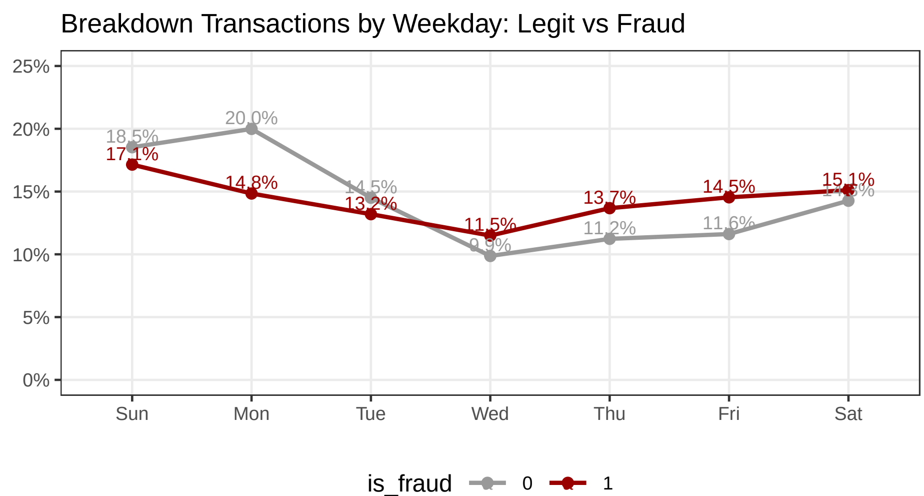
The percent of fraud transactions varies by month, and fraud/legitimate transactions show different trends during the year. Next I will look for trends within the month. Are certain days of the month more likely to have fraud charges?





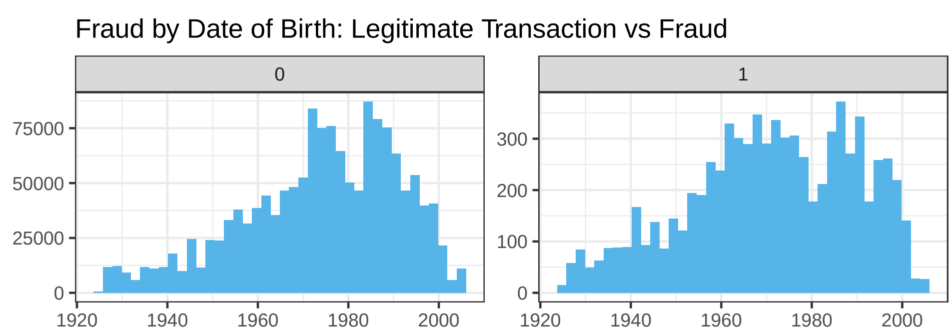
Although the transactions by day line chart is difficult to read, it does show a slight difference in the daily trends of fraud and legitimate charges. Once again legitimate transactions remain relatively steady throughout the month, but fraud dips up and down by day. Looking at weekday below, there is only slight differences between the percent of fraud by weekday. But compared with legitimate transactions, there appears to be an opposite charge trends by day of the week.





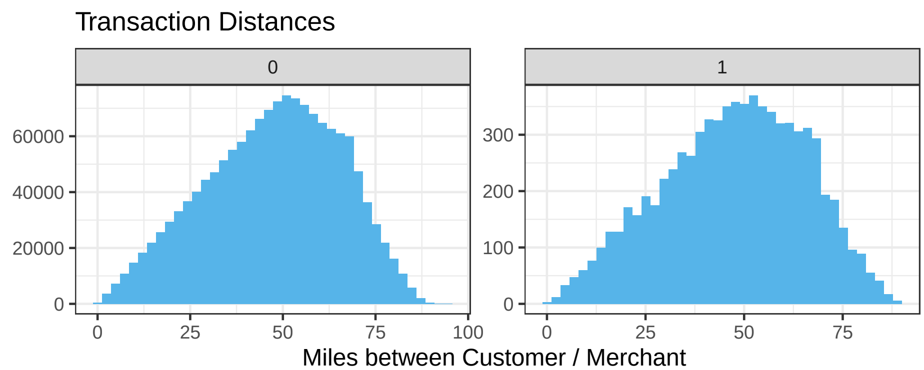
Not all variables have predicting power. For instance, neither gender nor age appear to be a good predictor.

is_fraud	gender	amt	n	pct_amt	pct_n
1	F	1948927	3938	0.47	0.51
1	M	2175996	3790	0.53	0.49

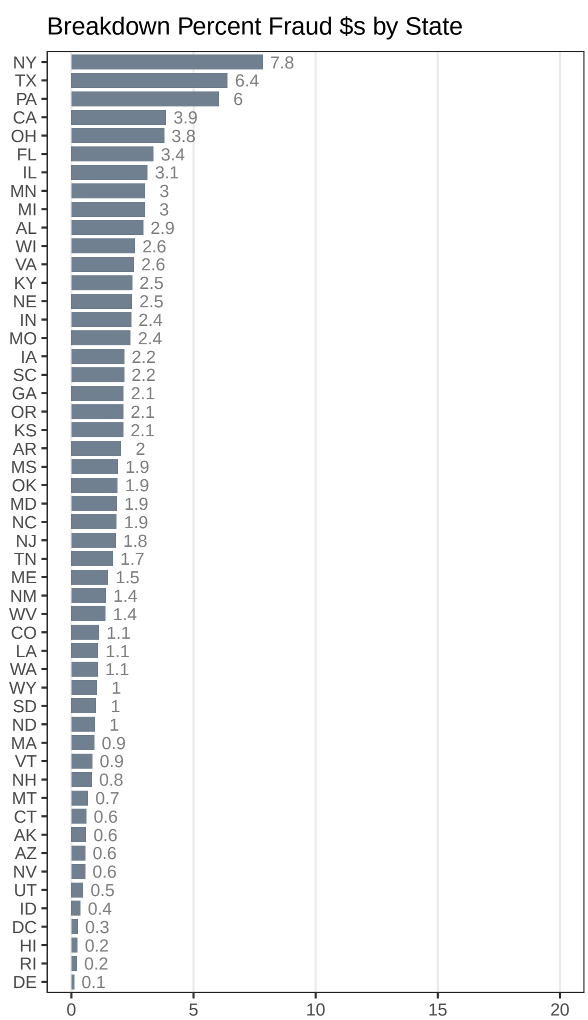


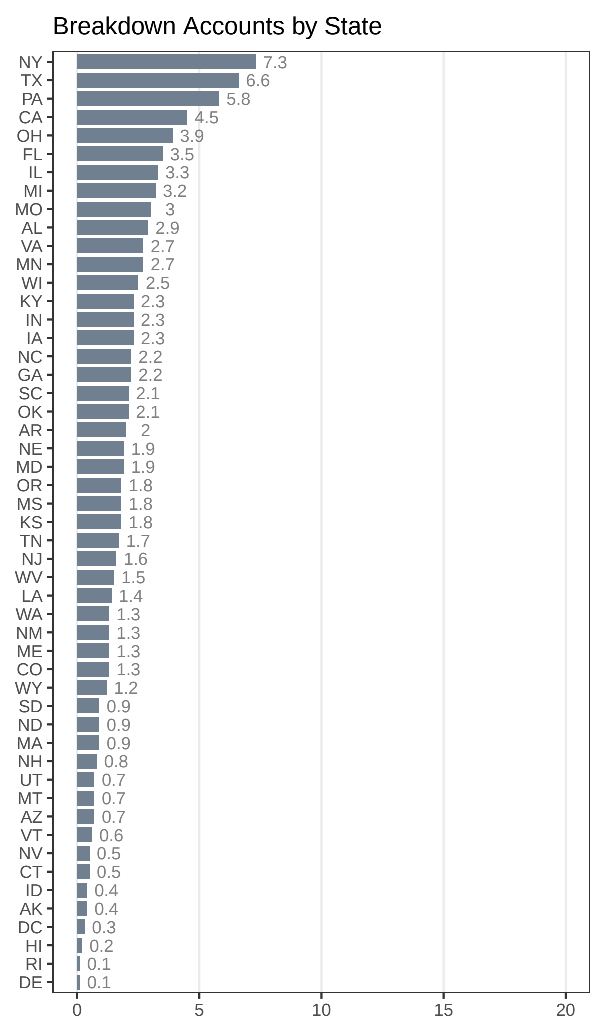
In real-life fraudulent transaction are often to merchants that are very far from the customer. With the longitude and latitude for both the customer and the merchant, we can calculate and analyze distance.

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: 0.04 Max: 94.65

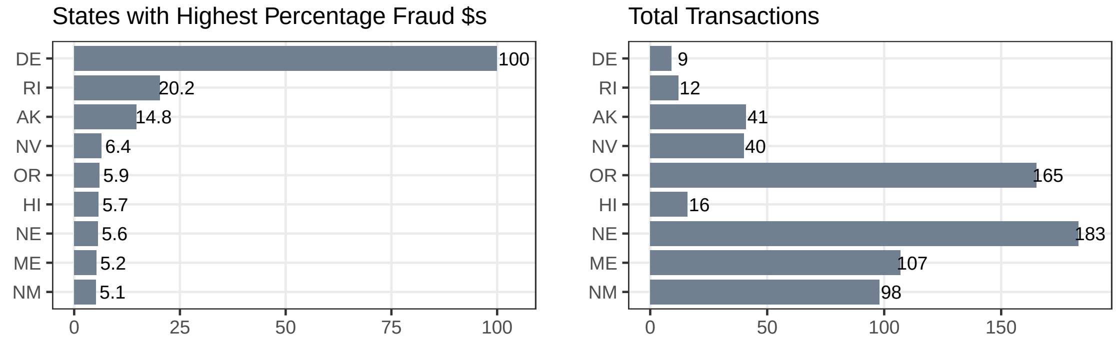


Lastly when looking at the breakdown of fraud by customer state, fraud rates are consistent with percent of accounts by state.





Looking at the states with the highest fraud rates, it appears the top three are small population states. A closer inspection of Delaware's transactions shows 100% fraud and only nine transactions. The states with greater than five percent fraud have very few transactions.



A closer inspection of Delaware's transactions shows 100% fraud and only nine transactions. The states with greater than five percent fraud have very few transactions. State would not be a good predictor. I believe I can construct an effective model with amount, category and date parts.

Data Cleaning

Before building the models, the dataset needs to be prepped to include the relevant predictors. To reduce the size of the dataset, I will remove unused columns and convert the date and time elements into features. I will keep the predictors to amount, category and transaction time and date parts: hour, month, day of month, and weekday and as factors. I will keep merchant and cc_num as possibilities.

Unfortunately, the synthetic card numbers were created fictionally which reduced their usefulness 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, the final digits are unique account ids. Here the credit card numbers here are between 11 to 19 digits and do not follow this pattern. I will change cc_num to a character variable to be handled appropriately by the algorithms. Using unique identifiers such as account numbers can lead to over training. Since fraud is usually repeated, I believe cc_num has usable predicting power. Although a recency or frequency feature once fraud is detected might provide a better predictor, for simplicity I will keep cc_num and will fit models with and without for comparison.

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

Modeling Methods

I will compare three algorithms presented in HarvardX's Machine Learning course for best for anomaly prediction: two Classification and Regression Tree (CART) models using rpart and randomForest and also a 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 **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. The defaults are 500 trees and square root of number of variables.[3][4]

Logistic Regression:
I will use the glm function 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, it is 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: I originally planned to use the caret function to train all models utilizing its cross-validation feature. Unfortunately, with a over a million observations, caret took hours to run and did not significantly improve the model compared rpart and glm.

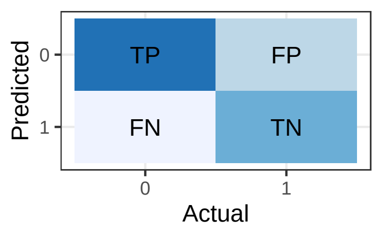
Model Evaluation

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 fraud transaction amounts (the cost) is eight times higher than the number of fraud transactions. In real life, fraudulent charges also produce additional customer service costs.[6] Therefore, I will use a combination of confusion matrix metrics and cost analysis to evaluate model performance.

Predictions have four possibilities in the following models:

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

Costs:

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

Model Building

First I calculate lost revenue when no fraud is detected. I will evaluate three versions of each algorithm with different variables and parameters, then choose then best performing construct of each for final evaluation.

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

The cost of not detecting fraud = $ 435007.06.
Accuracy with no correct fraud predictions: 99.48 %.

Rpart Models

Rpart Model 1: To begin I will include all possible predictors and assess results and variable importance. Date and time parts are included as factors. *Formula: rpart(is_fraud ~ ., data = train2, method = "class")*

Rpart Model 1 Confusion Matrix:

	0	1
0	147379	335
1	40	438

Rpart Model 1 Results:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
Rpart All Vars	274152	160855	32003	0.63	0.07	0.57	0.92

The results are not very good with only 63% of fraud amounts and just over half of transactions detected. Although with 92% NPV, at least the model isn't incorrectly flagging very many legitimate transactions. 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 I will evaluate variable importance.

Rpart Model 1 Variable Importance:

merchant	3554.6
category	2605.7
bins	2605.7
amt	2066.7
cc_num	1834.4
trans_hour	1032.1
day	2.9

I find it unlikely that merchant should top the list or that category and bins are improving the model more than amount. I suspect the high-level categorical variables are not performing well. Including a constructed feature like bins is probably inhibiting rpart's ability to calculate best splits.

Rpart Model 2: For a simpler model, I include only amount, category and date parts as factors. I am not including bins to allow rpart partitioning to calculate best split value.
Formula: rpart(is_fraud ~ amount + category + hour + month + day + weekday, data = train2, method = "class")

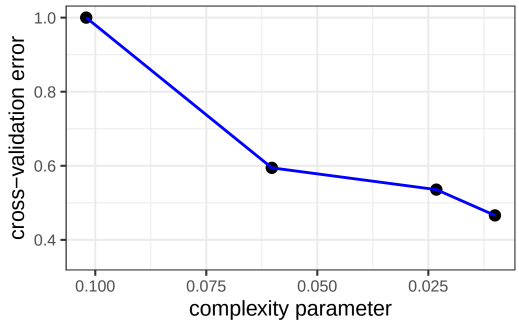
Rpart Model 2 Confusion Matrix:

	0	1
0	147319	265
1	100	508

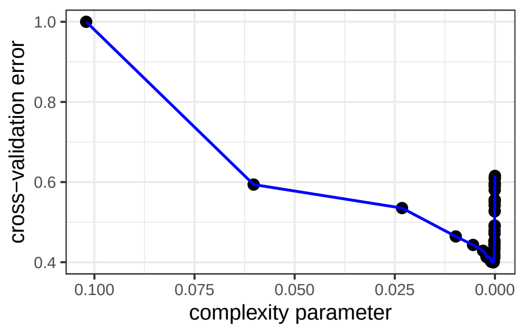
Rpart Model 2 Results:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
Rpart Basic	332542	102465	78829	0.76	0.18	0.66	0.84

This is a significant improvement over the first model with 76% of fraud dollars predicted. With such a large dataset with high-level variables, does the complexity parameter need to be tuned to improve the model? When we plot the cp against the xerror, it does appear that we could improve the model by decreasing the complexity parameter.[7]



Rpart Model 2 Tuning CP: I am setting minimum split and complexity parameters to zero to determine which cp value minimizes the cross-validated error (xerror), then I will prune the model accordingly with prune.rpart function. *Formula: rpart(is_fraud ~ amount + category + hour + month + day + weekday, data = train2, minsplit = 0, cp = 0, method = "class")*



It appears that the xerror is minimized much below the complexity parameter default of .01 and there are multiple xerror values below .4. Minimum xerror:

	CP	nsplit	rel error	xerror	xstd
21	0.000489	78	0.341481	0.4	0.007576
22	0.000479	84	0.337168	0.4	0.007576
24	0.000383	100	0.329978	0.4	0.007576

Rpart Model 2 Tuned CP Confusion Matrix:

	Rpart Basic Model		Rpart Basic Tuned	
	0	1	0	1
0	147319	265	147347	250
1	100	508	72	523

Rpart Model 2 Tuned CP Results Compared:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
Rpart Basic	332542	102465	78829.3	0.76	0.18	0.66	0.84
Rpart Basic Tuned	334801	100206	56715.7	0.77	0.13	0.68	0.88

Tuning the complexity parameter slightly improved saved amount ≈ \$9,000. It had a much larger impact reduced false negatives/misclassified amount by over \$21,000.

Rpart Model 3: I am interested in running the model with cc_num. In most models using a unique identifier is not advised, but in this case the cc_num has thousands of observations attached to it, and due to the repeat natural of fraud charges identifying fraud on a cc_num could be a valid predictor. *Formula: rpart(is_fraud ~ amount + category + hour + month + day + weekday + cc_num, data = train2, method = "class")*

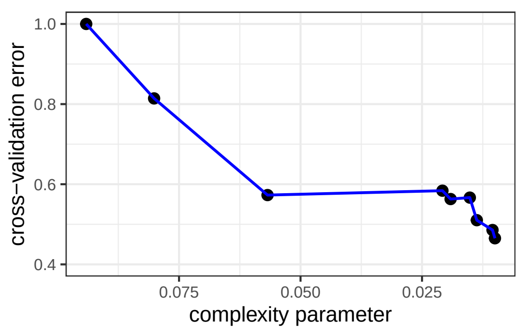
Rpart Model 2 & 3 Confusion Matrix Compared:

	Rpart Basic Model		Rpart with CCs	
	0	1	0	1
0	147319	265	147329	266
1	100	508	90	507

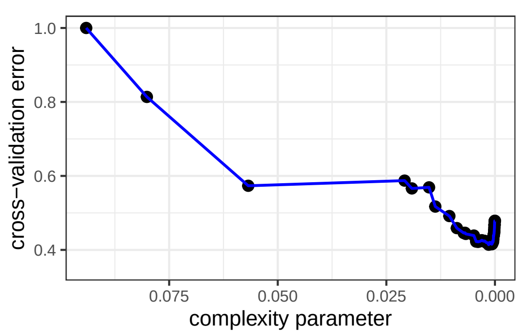
Rpart Model 2 & 3 Results Compared:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
Rpart Basic	332542	102465	78829.3	0.76	0.18	0.66	0.84

The rpart model with credit card numbers performed close to but not as good as the basic model. Plotting the complexity parameter again, shows the error was still decreasing when the cp parameter was reached.



Rpart Model 3 Tuning CP: Reruning model with with minimum split and complexity parameters to zero to determine which cp value minimizes the cross-validated error (xerror), then using new cp value to pruning with prune.rpart. *Formula: rpart(is_fraud ~ amount + category + hour + month + day + weekday + cc_num, data = train2, minsplit = 0, cp = 0, method = "class")*



Rpart Model 3 CP Tuned Confusion Matrix Compared:

	Rpart with CCs		Rpart CCs Tuned	
	0	1	0	1
0	147329	266	147363	241
1	90	507	56	532

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
Rpart w CCs	327393	107614.1	86883.8	0.75	0.2	0.66	0.85
Rpart CCs Tuned	340096	94911.3	44026.4	0.78	0.1	0.69	0.90

The tuned model with card numbers performed the best of the three catching 78% of fraudulent charged amounts. Not all predictors improved rparts performance and default complexity parameter did not perform best. I will use this rpart model with cc_nums for final validation and model comparison.

GLM Models

To begin I am comparing two models 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, first we create the fit model then calculate probability estimates with: *predict.glm(fit_glm, test2, type = "response")* and finally create a vector of predicted outcomes based on a threshold.

GLM Model 1: The first model will include amount and category interaction plus date parts as factors. It does not include the additional calculated bins variable. Predicted outcomes based on the threshold > .5. *Formula: glm(is_fraud ~ amount * category + hour + month + day + weekday, data = train2, family = "binomial").*

GLM Model 2: The second model will be a multivariate linear model including our calculated feature bins but not modeling for any interaction. *Formula: glm(is_fraud ~ amount + category + bins + hour + month + day + weekday, data = train2, family = "binomial").*

GLM Models 1 & 2 Confusion Matrix:

	GLM cat*amt		GLM cat+amt+bins	
	0	1	0	1
0	147339	376	147350	365
1	80	397	69	408

GLM Models 1 & 2 Results:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
GLM cat*amt	223209	211798	156292.5	0.51	0.36	0.51	0.83
GLM cat+amt+bins	299996	135011	60386.7	0.69	0.14	0.53	0.86

Model 1, even with interaction between category and amount, did not perform very well. Model 2 without interaction performed better when I introduced a feature 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 interaction plus date parts as factors. I will evaluate the model's estimates at thresholds of .5 and .4 to compare results. *Formula: glm(is_fraud ~ amount * category * bins + hour + month + day + weekday, data = train2, family = "binomial").*

GLM Model 3 Confusion Matrix:

	GLM cat*amt*bins > .5		GLM cat*amt*bins > .4	
	0	1	0	1
0	147357	233	147340	219
1	62	540	79	554

GLM Model 3 Results:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
Glm bins >.5	349281	85726.4	47885.0	0.80	0.11	0.70	0.90
Glm bins >.4	360252	74755.0	59982.5	0.83	0.14	0.72	0.88

Reducing the probability threshold increased the number of correct fraud predictions and saved about $11,000 more, but it increased false positives by $12,000. 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 the sale. With today's automation banks can send a text allowing the cardholder to approve or deny the suspicious transaction. I would think this makes the false positives preferable over false negatives.

RandomForest Models

RandomForest Model 1: The main tuning parameters 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 with replacement will generate more randomness. Due to machine memory limits, I decided to start with a very low number of trees (51) but kept the default variable parameter.
Formula: randomForest(is_fraud ~ ., data = train2, ntree = 51, replacement = TRUE, importance = TRUE)

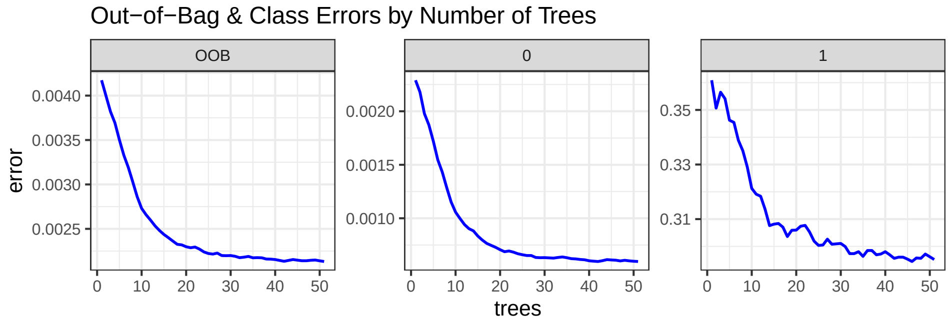
RandomForest Model 1 Confusion Matrix:

	0	1
0	147346	226
1	73	547

RandomForest Model 1 Results:

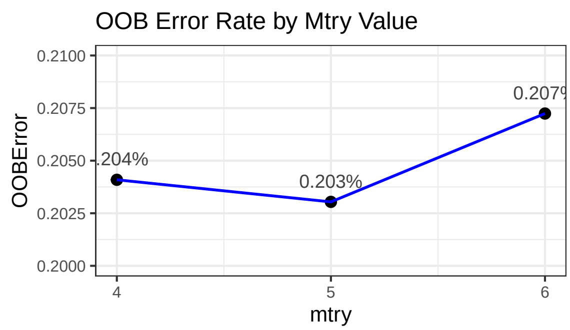
Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
RandomForest 51	358254	76753.4	58142.5	0.82	0.13	0.71	0.88

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 = 0.213238 % and Fraud Class Error = 29.446441 %. I want to look at error rates for higher values of mtry to see if I can improve the results. 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, legitimate transactions are flattening out around 30 trees, but the fraud class is still declining and has a much higher error rate.

RandomForest Model 1 Tuning Mtry: I will use the the tuneRF function. TuneRF takes a starting mtry input and returns the OOB error for a step factor above and below. I am increasing the number of trees slightly. *tuneRF(train2[-6], train2$is_fraud, mtryStart = 5, ntreeTry = 75, stepFactor = .9)*



In actuality, I am more interested in the err.rate for fraud class and the cost results than the OOB Error, but 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 the 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, I am limited on the number of trees I can run and this will inhibit performance and also re-producibility of results. I will train the next randomForest model, with 251 trees (just over half of the default) and use mtry = 4.

RandomForest Model 2: Model was fit for 251 decision trees, sampling four out of ten features: amount, category, bins, factored date/time parts, and full trans_date_trans_time.
Formula: randomForest(is_fraud ~ ., data = train2, ntree = 251, mtry = 4, replacement = TRUE, importance = TRUE)

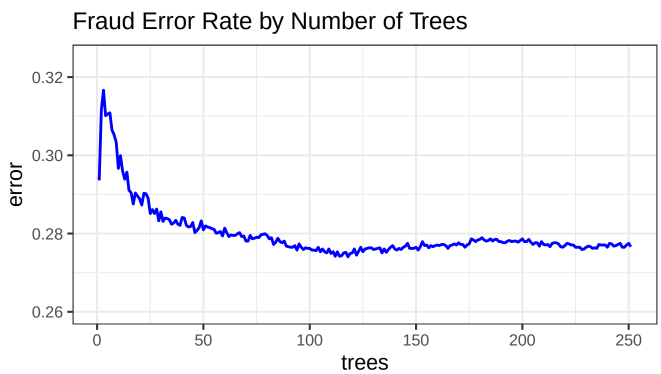
RandomForest Model 2 Confusion Matrix:

	0	1
0	147350	217
1	69	556

RandomForest Model 1 & 2 Results:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
RandomForest 51	358254	76753.4	58142.5	0.82	0.13	0.71	0.88
RandomForest 251	363887	71119.8	55898.6	0.84	0.13	0.72	0.89

Not surprisingly the randomforest models performed the best. Surprising, they ran much faster than the glm models. It was also interesting to see that increasing variables per tree did reduce false fraud predictions, it also slightly reduced the number of correct predictions and missed fraud. Looking at the err rate for the model:



Adding more trees could still improve the model, but the error rate appears to be stabilizing. I will use the RandomForest 251 trees model for final validation.

Note on Variation of RandomForest Results: Even with setting a seed, there is still randomness introduced in the algorithm and the results change slightly when re-run. (Setting the seed did create reproducible models results when run on the same day.) During testing, I ran multiple randomForest 251 models with mtry at 4 and 5 although I did not include the code and results here to save time (on an already lengthy project). The mtry=5 models had a lot of variation in their results and in the class 1 error. They sometimes performed thousands of dollars better or worse. Mtry=4 models had less variation both results and class error. My assumption is that the mtry=5 sometimes picked up very good trends and also over-fit trees. This means that while 251 trees can produce very good results, the number of trees is too low to fit a stable model. Unfortunately, my laptop errors out at 300 trees. Online I have seen rf models with 1000 and 1500 trees which would produce more stable and better results!

Final Validation

For final validation of the chosen models, I am running each on the validation set containing 20% of original data not used in training the models or data exploration.

- Rpart with tuned complexity parameter including credit card numbers as categorical variables: *Formula: rpart(is_fraud ~ amount+category+hour+month+day+weekday+cc_num, cp=0.000383, method="class")*
- Glm with interaction between category, amount, and bins with predicted outcomes > 0.4: *Formula: glm(is_fraud ~ amount * category * bins+hour+month+day+weekday, family="binomial")*
- RandomForest with 251 trees sampling 4 predictors: *Formula: randomForest(is_fraud ~ ., ntree=251, mtry=4, replacement=TRUE, importance=TRUE)*

Final Validation Results: Even with only half the default number of trees, random forest performed the best and was able to correctly predict 83% of the fraudulent transactions costs. Glm was very close 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	368396	608	368306	565	368343	565
1	160	1315	250	1358	213	1358

Final Models Cost Results Comparison:

Model	AmtSaved	FraudMissed	MisClassified	SavedPct	MisClassPct	Specificity	NPV
No Fraud Predicted	0	996491	0	0.00	0.00	0.00	0.00
Rpart CCs Tuned	783629	212862	120485	0.79	0.12	0.68	0.89
Glm amt*cat*bins >.4	822379	174111	195610	0.83	0.20	0.71	0.84
RandomForest 251	829538	166953	171176	0.83	0.17	0.71	0.86

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).