Fraud Detection in Credit Card Transactions

Team 22
Christina Acojedo
Mrinal Gupta
Athanasios Rompokos
Guyane Valian
Min Zhu

Project 3 Report
DSO562 | Fraud Analytics
Spring 2020
May 3, 2020

Table of Contents

Executive Summary

GSA SmartPay: Supporting your Mission
GSA SmartPay 3
United States of America
U. S. Government Tax Exempt
FOR OFFICIAL GOVERNMENT USE ONLY
1234 5678 9012 3456
12/31/26
JOHN SMITH
Purchase . Travel . Fleet . Integrated . Tax Advantage

Credit card fraud is a burden for organizations across the globe. Specifically, $24.26 billion were lost due to credit card fraud worldwide in 2018, according to shiftprocessing.com. In this report, our goal was to build an effective and efficient model to predict fraud. We analyzed a real-world dataset that contained a list of government related credit card transactions over the 2010 calendar year. The data presented a supervised problem as it included a column showing the transaction's fraud label (whether a transaction was fraudulent or not). It also contained identifying information about each transaction such as the credit card number, merchant, merchant state, etc. The dataset had 96,753 records and 10 data fields. We first described and visualized each of the 10 data fields, cleaned the dataset, and filled in missing values. Then we created many variables and performed feature selection. Finally, we created a variety of models using several algorithms (both linear and nonlinear) and highlighted our results.

Each of the 10 data fields in the dataset were analyzed and an exploratory data analysis was performed. There were nine categorical variables and one numeric variable. We provided a histogram or table for each data field to get a better feel for its distribution. Many of the variables had a right skewed distribution. Overall, the dataset had 1,059 fraudulent transactions of the total 96,753 records (1.09%).

The dataset required cleaning before we could move to the variable creation phase. Of particular note, we removed one outlier record due to its extremely high value. Also, we only analyzed records that had a transaction type equal to "P" (purchase) to work with a more focused dataset. Lastly, the "Merchnum", "Merch state", and "Merch zip" data fields contained missing values which necessitated the use of carefully crafted data imputation techniques.

Once the dataset was cleaned, we attempted to create as many candidate variables as possible. There were six different types of variables: amount variables, frequency variables, days-since variables, velocity change variables, Benford's Law variables, and a day-of-the-week risk table variable.

After the variable creation process, we performed feature selection via filter and wrapper methods in order to determine the best variables for use in the development of machine learning models to predict potentially fraudulent activity. Kolmogorov-Smirnov (KS) and Fraud Detection Rate (FDR) at 3% were used to eliminate less effective variables as a filtering method, while the average results of three different tree-based methods was used as our wrapper method. Once the final variables were chosen, the data was divided into three sections for model development and analysis: training, testing, and out-of-time (OOT).

To find the best model, the variables were tested in several different models. First, since we had a supervised binary classification problem, logistic regression was used as a base model to predict fraud. Logistic regression caught 44.80% of fraud at a 3% FDR. Next, nonlinear models such as random forest, boosted trees, and neural network were used and compared against the logistic regression model. Parameter tuning was also performed on each model to get better results. Overall, our best model was random forest which caught 59.21% fraud at a 3% FDR after parameter tuning.

Description of Data

The dataset used in this analysis contained a list of credit card transactions over the 2010 calendar year. The records are a combination of actual credit card transactions from a government entity and synthetic credit card transactions to be used to locate fraud. It contained 96,753 transaction records and 10 fields. The data presented a supervised problem as it included a column called "Fraud" which took on a binary value of 1 or 0. A record with a value of 1 represented a fraudulent transaction while a record with a value of 0 represented a normal transaction. There was a total of 1,059 records with a label of 1 and 95,694 records with a label of 0. The summary tables of all the data fields is provided below in figure 1.

Categorical Summary Table

Data Field	Num Records w/ a Value	Percent Populated	Num Unique Values	Most Common Value
Recnum	96,753	100%	96,753	N/A
Cardnum	96,753	100%	1,645	5142148452
Date	96,753	100%	365	2010-02-28
Merchnum	93,378	96.5%	13,092	930090121224
Merch description	96,753	100%	13,126	GSA-FSS-ADV
Merch state	95,558	98.8%	228	TN
Merch zip	92,097	95.2%	4,568	38118
Transtype	96,753	100%	4	P
Fraud	96,753	100%	2	0

Numerical Summary Table

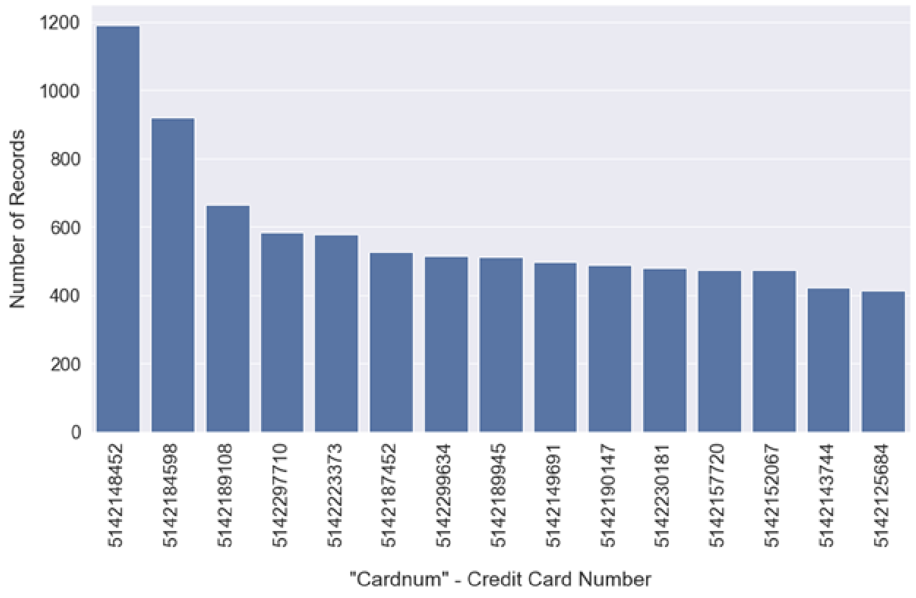
Data Field	Num Records w/ a Value	Percent Populated	Num Unique Values	Num Records w/ a Value of Zero	Mean	Std Dev	Min	Max
Amount	96,753	100%	34,909	0	$427.89	$10,006.14	$0.01	$3,102,046

Figure 1. Summary statistics of categorical and numerical variables.

Furthermore, we highlighted four critical data fields of the total ten: "Cardnum", "Merch description", "Amount", and "Fraud". The description and distribution of these four data fields are outlined below. For a full description of all the data fields in the dataset, please refer to Appendix A for the Data Quality Report (DQR).

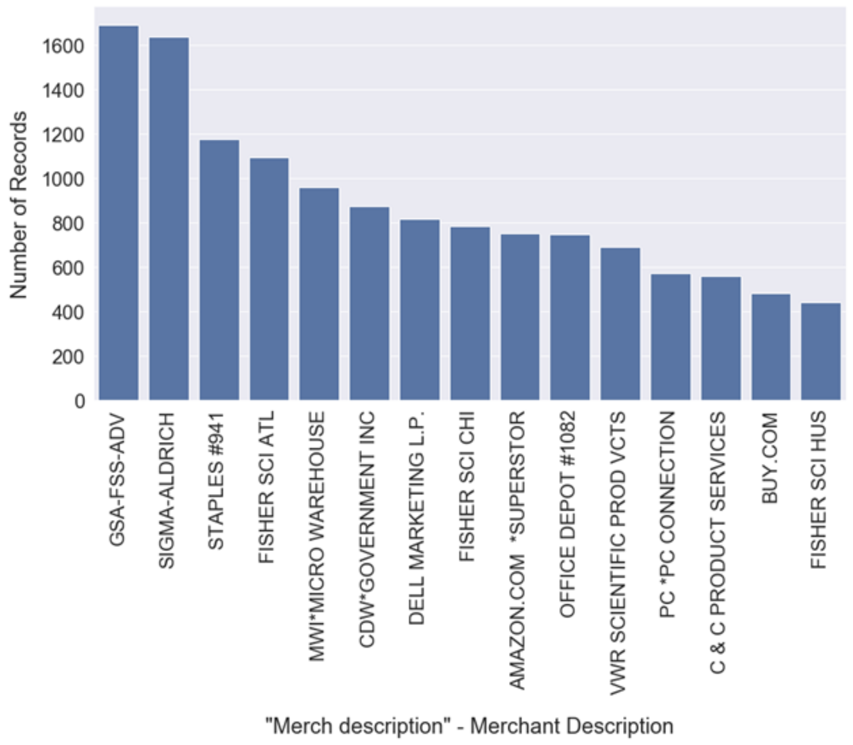
Cardnum Data Field

"Cardnum" represents a 10-digit version of the full 16-digit credit card number that was used in the transaction. It includes the first four digits and last six digits of the credit card number only. This field was 100% populated and had 1,645 unique values. The most common value was "5142148452" at about 1,200 records followed by "5142184598" coming in second at about 900 records. The distribution was skewed to the right.



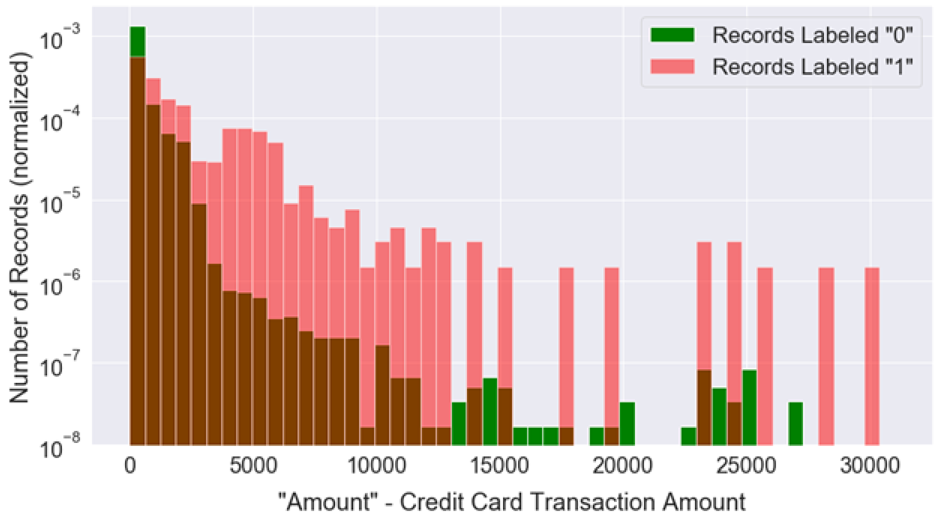
Merch Description Data Field

This data field showed the merchant name/description for each credit card transaction. “Merch description” was 100% populated with 13,126 unique values. The most common value was “GSA-FSS-ADV” followed by “SIGMA-ALDRICH” with both at just above 1,600 records. Similar to “Cardnum”, this field also had a right-skewed distribution.



Amount Data Field

“Amount” represented the amount of each credit card transaction in the dataset. This field was 100% populated with no zero values. It was the only numeric field in the dataset provided. Its mean was about \$427 with a minimum value of \$0.01, and a maximum value of \$3,102,046. This seemed to be a wide range of amount values. With more exploration, we realized that the maximum value was an amount in Mexican Pesos and was deemed an outlier. We discuss this outlier further in the next section of this report titled Data Cleaning. Additionally, we witnessed another right-skewed distribution as seen below. The distribution is broken down by fraudulent records (in red) and not fraudulent records (in green). As the transaction amounts became higher, the number of normal transactions decreased whereas the number of fraudulent transactions only slightly decreased and then plateaued.



Fraud Data Field

This field showed whether a transaction was fraudulent or not. A record was coded with a “1” if the transaction was fraudulent or coded with a “0” if it was not fraudulent. This was the dependent variable in our analysis. Out of the 96,753 variables, 1.09% of records were fraudulent.

Value of “Fraud”	Number of Transactions
0	95,694
1	1,059

Data Cleaning

Since the dataset used in our analysis was real-world data, we discovered that it contained noisy records and three critical data fields with missing or erroneous values. We therefore determined it was necessary to exclude the noisy records for a proper analysis and to use data imputation techniques before fully analyzing the dataset for potential fraud cases. This resulted in the exclusion of 356 records from the dataset and an assessment and cleaning of the following data fields for missing or erroneous values: "Merch state", "Merch zip", and "Merchnum". A description of the exclusions and the data imputation conducted for each of these data fields is discussed in the subsections below.

Exclusions

There were two main types of records we discovered that needed to be excluded from the dataset. The first type was an outlier found in the "Amount" data field. The record with a "Recnum" value of "52715" contained an "Amount" value that was discovered to be annotated in Mexican Pesos. As a result, the value was recorded as "3,102,045.53" for the respective "Amount" data field. Due to this discrepancy, we determined it was necessary to exclude this record since there was a high potential for this record to cause noise and/or unnecessary issues in our analysis.

The second type of record that needed to be excluded were those records containing anything but a "P" value in the "Transtype" data field. Each record in the dataset was coded with a 1-character categorical value for the type of transaction associated with that record. Since we were only concerned with evaluating the purchase or "P" type records in detecting potential fraud cases with the various credit card transactions, we determined that any record without a "P" value for the "Transtype" data field would likely result in noise and/or cause unnecessary issues in our analysis. Thus, we excluded the 355 records that contained an "A", "D", "or "Y" value in the "Transtype" data field so that only "P" type transactions remained.

In removing the outlier record and the 355 records that did not contain a "P" value for the "Transtype" data field, we ended up excluding a total of 356 records before proceeding with our analysis. We were therefore left with 96,397 records for analysis.

Merch state

The "Merch state" data field had 1,021 records with a missing value. Since this data field had the fewest number of records that required cleaning, we opted to start our data imputation with this data field. To begin, we took a unique approach by using a free database of 33,099 US zip codes provided by SimpleMaps.com that can be found at https://simplemaps.com/data/us-zips. We opted to use this method as our first step because we sought to fill the missing values for the "Merch state" data field in the most accurate manner possible. Once this was complete, we proceeded to fill the remaining missing values using more traditional methods. The overall steps and procedures we used were as follows:

STEP 1 → We downloaded the "Basic" version of the US zip codes database from SimpleMaps.com and then extracted the zip codes and their corresponding states from the database into a Python dictionary where the zip codes served as the dictionary keys and the states served as the dictionary values. We then mapped this dictionary to the "Merch zip" data field to fill the corresponding missing values in the “Merch state” data field. This step reduced the number of records with missing values in this data field to 976 records.

STEP 2 → Since there were missing values that still remained after Step 1, we moved to simply using the “Merch zip” data field as a focal point to fill missing values in the “Merch state” data field. We wanted to continue using the zip code as much as possible in an attempt to fill the missing values for the “Merch state” in the most accurate manner possible. However, because the “Merch zip” data field also contained missing values, we needed to temporarily remove the records that had a missing value for the “Merch zip” data field to avoid erroneous groupings that can occur when using a “dummy” value and to prevent errors that can occur with size mismatches when grouping data fields in Python dataframes that are not fully filled. Once this was complete, we were then able to fill the missing values in the “Merch state” data field with the most common value after grouping the dataset by the “Merch zip” data field. Given that we had temporarily removed the aforementioned records, we needed to devise a way to properly map our results to the main dataframe. We therefore created a Python dictionary of our results where the “Recnum” of a record served as the dictionary key and the corresponding “Merch state” value served as the dictionary value. This dictionary was then mapped to the “Recnum” data field in the main dataframe to fill the corresponding missing values in the “Merch state” data field. This step reduced the number of records with missing values in this data field to 959 records.

STEP 3 → Since there were missing values that still remained after Step 2, we then filled the missing values with the most common value after grouping the dataset first by the “Merch description” data field, then by the “Cardnum” data field. The reasoning for this grouping was based on the idea that the owner of the “Cardnum” may frequent the same business for purchases. This step reduced the number of records with missing values in this data field to 904 records.

STEP 4 → Since there were missing values that still remained after Step 3, we then filled the missing values with the most common value after grouping the dataset first by the “Cardnum” data field, then by the “Date” data field. The reasoning for this grouping was based on the idea that the owner of a “Cardnum” would make purchases within the same general area on the same day. A notable risk with this grouping is that it is unable to accommodate for unique circumstances such as a “Cardnum” owner using that particular card while traveling, or a “Cardnum” owner using that particular card for online or over the phone purchases to merchants that may be geographically dispersed. Nevertheless, we understood these risks and felt they would have a minimal negative impact, so we proceeded forward. This step reduced the number of records with missing values in this data field to 597 records.

STEP 5 → Since there were missing values that still remained after Step 4, we then filled the missing values with the most common value after grouping the dataset simply by the “Merch description” data field. This step reduced the number of records with missing values in this data field to 129 records.

STEP 6 → Since there were missing values that still remained after Step 5, we then manually inspected the remaining records for any other potential groupings. Unfortunately, we could not find any other logical groupings and so we decided to begin repeating our steps starting with Step

3. A repeat of Step 3 did not produce any results, but a repeat of Step 4 was able to reduce the number of records with missing values in this data field to 126 records.

STEP 7 → Since there were missing values that still remained after Step 6, we then repeated Step 5, but this did not produce any results. We therefore opted to simply group by the "Cardnum" data field based on the idea that the owner of a "Cardnum" may make most of his/her purchases within the same general area. Using this method, we were able to reduce the number of records with missing values in this data field to 31 records.

STEP 8 → Since there were missing values that still remained after Step 7, we conducted another manual inspection of the remaining records. At this point, we were unable to find any other logical mappings or groupings. We therefore filled the missing values for the 31 remaining records with their corresponding "Recnum" in an attempt to prevent any adverse linkages. At this point, the entire "Merch state" data field was filled.

Merch zip

The "Merch zip" data field had 4,301 records with a missing value. To fill the missing values, we used the following logic:

STEP 1 → In an attempt to fill the missing values for this data field in the most accurate manner possible, we wanted to start by using the "Merchnum" and "Merch state" data fields. However, because the "Merchnum" data field also contained missing values, we needed to temporarily remove the records that had a missing value for the "Merchnum" data field to avoid erroneous groupings that can occur when using a "dummy" value and to prevent errors that can occur with size mismatches when grouping data fields in Python dataframes that are not fully filled. Once this was complete, we were then able to fill the missing values in the "Merch zip" data field with the most common value after grouping the dataset first by the "Merchnum" data field, then by the "Merch state" data field. Given that we had temporarily removed the aforementioned records, we needed to devise a way to properly map our results to the main dataframe. We therefore created a Python dictionary of our results where the "Recnum" of a record served as the dictionary key and the corresponding "Merch zip" value served as the dictionary value. This dictionary was then mapped to the "Recnum" data field in the main dataframe to fill the corresponding missing values in the "Merch zip" data field. This step reduced the number of records with missing values in this data field to 2,474 records.

STEP 2 → Since there were missing values that still remained after Step 1, we then filled the missing values with the most common value after grouping the dataset first by the "Merch description" data field, then by the "Merch state" data field. This step reduced the number of records with missing values in this data field to 2,151 records.

STEP 3 → Since there were missing values that still remained after Step 2, we then filled the missing values with the most common value after grouping the dataset simply by the "Merch description" data field. This step reduced the number of records with missing values in this data field to 2,030 records.

STEP 4 → Since there were missing values that still remained after Step 3, we then filled the missing values with the most common value after grouping the dataset simply by the "Merch state" data field. This step reduced the number of records with missing values in this data field to 71 records.

STEP 5 → Since there were missing values that still remained after Step 4, we then filled the missing values with the most common value after grouping the dataset first by the "Cardnum" data field, then by the "Date" data field. The reasoning for this grouping was based on the idea that the owner of a "Cardnum" would make purchases within the same general area on the same day. A notable risk with this grouping is that it is unable to accommodate for unique circumstances such as a "Cardnum" owner using that particular card while traveling, or a "Cardnum" owner using that particular card for online or over the phone purchases to merchants that may be geographically dispersed. Nevertheless, we understood these risks and felt they would have a minimal negative impact, so we proceeded forward. This step reduced the number of records with missing values in this data field to 54 records.

STEP 6 → Since there were missing values that still remained after Step 5, we then manually inspected the remaining records for any other potential groupings. Unfortunately, we could not find any other logical groupings and so we decided to begin repeating our steps starting with Step 2. A repeat of Step 2 was able to reduce the number of records with missing values in this data field to 46 records.

STEP 7 → Since there were missing values that still remained after Step 6, we then repeated Step 3, but this did not produce any results. We therefore moved on to repeating Step 4 where we were able to reduce the number of records with missing values in this data field to 31 records.

STEP 8 → Since there were missing values that still remained after Step 7, we then repeated Step 5, but this did not produce any results. We therefore conducted another manual inspection of the remaining records, but we were unable to find any other logical mappings or groupings. Hence, we filled the missing values for the 31 remaining records with their corresponding "Recnum" in an attempt to prevent any adverse linkages. At this point, the entire "Merch zip" data field was filled.

Merchnum

The "Merchnum" data field had 3,199 records with a missing value and 53 records with an erroneous value of "0" as its merchant number. To clean this data field, we used the following logic:

STEP 1 → With a combination of missing and erroneous values for this data field, we opted to convert the erroneous values to missing values so that there would be only one type of value to clean. Thus, we replaced all 53 of the erroneous values of "0" to a missing or "NaN" value. This increased our total number of records with missing values for this data field to 3,252 records.

STEP 2 → In an attempt to fill the "Merchnum" values in the most accurate manner possible, we started filling missing values with the most common value after grouping the dataset first by the "Merch description" data field, then by the "Merch zip" data field. This step reduced the number of records with missing values in this data field to 2,169 records.

STEP 3 → Since there were missing values that still remained after Step 2, we then filled the missing values with the most common value after grouping the dataset first by the “Merch description” data field, then by the “Merch state” data field. This step reduced the number of records with missing values in this data field to 2,110 records.

STEP 4 → Since there were missing values that still remained after Step 3, we then filled the missing values with the most common value after grouping the dataset simply by the “Merch description” data field. This step reduced the number of records with missing values in this data field to 2,095 records.

STEP 5 → At this point, we were unable to determine any more logical groupings or mappings for this data field. From a technical perspective, we could have easily made a variety of groupings or mappings to fill the missing values, but we concluded that not only was it wrong to take that course of action, but it could also cause adverse linkages and subsequent errors in our analysis. Hence, we filled the missing values for the 2,095 remaining records with their corresponding “Recnum” in an attempt to prevent any adverse linkages. At this point, the entire “Merchnum” data field was filled.

Candidate Variables

In this section, we developed additional candidate variables from the data fields in the original dataset that helped us detect potential fraud records. Crafting candidate variables is a bit of an artform and necessitates not only an understanding of the data and the different modeling techniques, but also domain knowledge of the subject area to include the business processes and customer or human behaviors surrounding that subject. For detecting potential fraud in the card transaction dataset, building candidate variables that relate to the amount, periodicity and frequency of transactions as they pertain to detecting transaction fraud are of great importance. Therefore, we built a number of candidate variables based on the data fields in the original dataset as they applied to the concepts of the monetary amount of the transactions (amount variables), the frequency in which transactions were conducted (frequency variables), the number of days since the last time a transaction was seen (days since variables), and the speed in which transactions were seen over a certain period of time in relation to what was considered normal (velocity change variables). Additionally, we created variables based on Benford's law of large numbers and the fraud rate per day (day-of-week risk table).

Initial Variable Creation

As an initial step before building numerical candidate variables that relate to the amount, periodicity and speed, we first needed to craft additional categorical variables that would serve as the basis for these properties. This meant carefully thinking through logical combinations of the original data fields that fraudsters could possibly use in their transactions.

It is possible from a programming perspective to create any combination and concatenation of the original data fields, namely "Cardnum", "Merchnum", "Merch description", "Merch state" and "Merch zip". This method would result in 25 additional variables, which would subsequently generate hundreds more variables in later calculations. Such an approach could also result in resource-intensive computations that may necessitate server and/or cloud computing capabilities. We therefore opted to judiciously select rational and reasonable combinations of the original data fields and then manually create three concatenations:

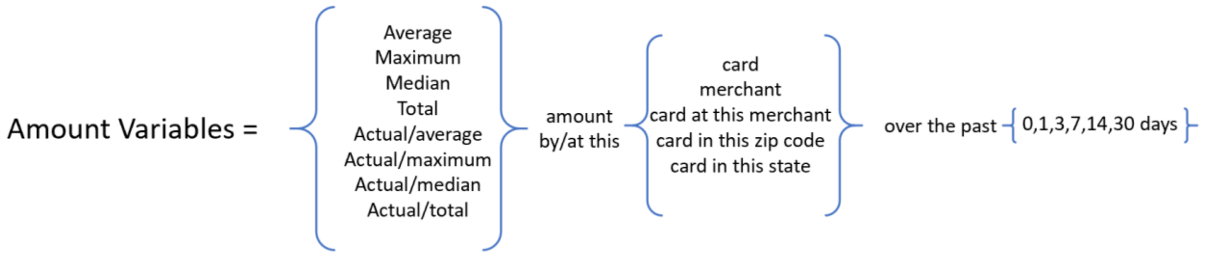
Candidate Variable	Concatenation of Original Data Fields
Cardnum_|_merchnum	Cardnum, Merchnum
Cardnum_|_zip	Cardnum, Merch zip
Cardnum_|_state	Cardnum, Merch state

Figure 2. Categorical Candidate Variables.

To perform these concatenations, we simply ensured each of the original data fields were converted to a string data type and then we manually concatenated the data fields by adding them together. With the creation of the three categorical candidate variables, we were then prepared to use them as the foundation for developing the numerical candidate variables for the amount, frequency, days-since, and velocity change variables.

Amount Variables

The amount variables are numeric variables and capture the statistics of the monetary amount of each transaction in the dataset for the categorical data fields and candidate variables. More specifically, the amount variables include the average, maximum, median and total amount of the transaction for each of the five categorical variables over the past 0, 1, 3, 7, 14, and 30 days (0 indicates the same day) and thus we generated 120 variables. Next, we normalized the actual transaction amount of each data field with each of the above variables and produced another 120 variables. We therefore generated 240 amount variables. The formula used to create the amount variables is shown below:



Fraudulent transactions tend to have higher amounts compared to non-fraudulent transactions for the same entity; hence, an increase in the average amount over a time window would possibly indicate fraud. A snapshot of some of the velocity change candidate variables are provided in figure 3 below. For the full list of the velocity change candidate variables, please see Appendix B.

	Amount Candidate Variables		Amount Candidate Variables
1	Cardnum_totalamount0_Amount2	**121**	Cardnum_totalamount_0_actual
2	Cardnum_totalamount1_Amount2	**122**	Cardnum_totalamount_1_actual
3	Cardnum_totalamount3_Amount2	**123**	Cardnum_totalamount_3_actual
4	Cardnum_totalamount7_Amount2	**124**	Cardnum_totalamount_7_actual
5	Cardnum_totalamount14_Amount2	**125**	Cardnum_totalamount_14_actual
6	Cardnum_totalamount30_Amount2	**126**	Cardnum_totalamount_30_actual
7	Merchnum_totalamount0_Amount2	**127**	Merchnum_totalamount_0_actual

Figure 3. Days-Since Candidate Variables.

Let us now present an example to further clarify the amount variables. In figure 4, we present the transactions of “Cardnum” value of “5142225701” between 01/01/2010 and 01/08/2010. The variable “Cardnum_totalamount3_Amount2” denotes the total transaction amount of a record over a 3-day period and the variable “Cardnum_totalamount3_Amount2” the actual transaction amount normalized by the total amount over a 3-day period. For the data field with index number 48 the total amount over the last three days is the same amount itself ($32.80) as it is the first transaction for that “Cardnum” and hence the “Cardnum_totalamount3_actual” variable equals to 1. The next data field for this “Cardnum” is on 01/08/2010 with amount of $\$1{,}035$ and the total amount for the last three days is $\$32.80 + \$1{,}035 = \$1{,}067.80$. The actual amount over the total amount in the last three days is $\$1{,}035/\$1{,}067.80 = 0.9692$ as seen in “Cardnum_totalamount_3_actual”. The same applies for the next data field with index number 231. Finally, the next transaction took place on 01/08/2010, five days later and we can see that the total amount over the last three days variable is reset.

Index	Date	Cardnum	Amount	Cardnum_totalamount3 _Amount2	Cardnum_totalamount_3 _actual
48	01/01/2010	5142225701	32.80	32.80	1
113	01/03/2010	5142225701	1,035	1067.80	0.96928257
152	01/03/2010	5142225701	50.83	1118.63	0.04543951
231	01/03/2010	5142225701	600	1718.63	0.34911528
1429	01/08/2010	5142225701	98.40	98.40	1

Figure 4. Amount Candidate Variables for Cardnum.

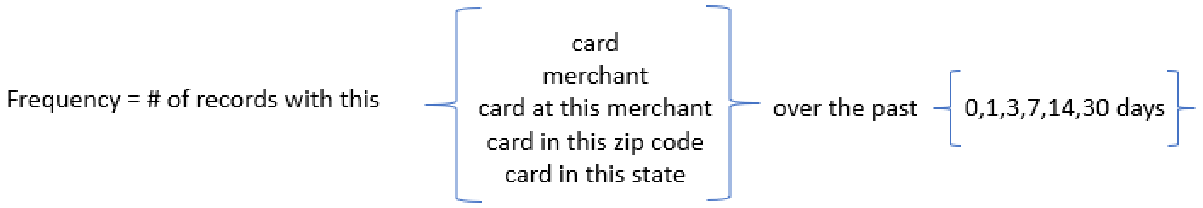
Frequency Variables

The frequency variables are numeric in nature and capture the speed at which categorical data fields and candidate variables were seen in a dataset for a particular transaction record. Since fraudsters often operate in bursts of activity, calculating the frequency in which transactions are seen is one way of finding these bursts of activity as we seek to identify potentially fraudulent transactions. A higher value calculated for a particular time period means there is increased risk and a stronger likelihood of fraud. The general equation for calculating the value for a velocity variable is as follows:

$$Frequency \ = \# \ of \ records \ (transactions) \ \ with \ the \ \boldsymbol{same \ data \ field} \ over \ the \ last \ X \ days$$

$$Frequency \ = \# \ of \ records \ (transactions) \ with \ the \ \boldsymbol{same \ variable} \ over \ the \ last \ X \ days$$

where X is an integer value from 0 to ∞ and the $\# \ of \ records$ is restricted to counting only the current record at hand and any records in the past (i.e. the $\# \ of \ records$ cannot include the count of records forward in time as it relates to time of day). It is important to note that a value of 0 for X represents the time period used for transaction records seen within the same day. From a more visual perspective, this equation can be viewed as follows:



For instance, in calculating the frequency of the “Cardnum” data field over the last 1-day period, the value of the frequency variable for a particular record only counts the current transaction record and any previous transactions records in the dataset that came before the current transaction record. In figure 5 below, we see that for the transaction records with a “Cardnum” value of “5142224817” in the dataset, the frequency variable containing the number of records over a 1-day period is called “Cardnum_velocity1_Date” and it contains the respective values for each transaction record. For index number 38283, the value of “Cardnum_velocity1_Date” can only include the current record and those records from 05/23/2010 up to the current record. Since there were two records on 05/22/2010 and two previous records on 05/23/2010 before index number 38283, the value for the velocity in “Cardnum_velocity1_Date” is 4 for index number 38283.

Index	Date	Cardnum	Cardnum_velocity0_Date	Cardnum_velocity1_Date
35870	05/15/2010	5142224817	1	1
37806	05/22/2010	5142224817	1	1
38057	05/22/2010	5142224817	2	2
38189	05/23/2010	5142224817	1	3
38283	05/23/2010	5142224817	2	4
38444	05/23/2010	5142224817	3	5
38632	05/24/2010	5142224817	1	4
38893	05/24/2010	5142224817	2	5
39563	05/26/2010	5142224817	1	1

Figure 5. 0-Day and 1-Day Candidate Frequency Variables for Cardnum.

The equation and procedure described above was used for each of the 3 categorical candidate variables we created and for the "Cardnum" and "Merchnum" data fields. For the value of X, we used time periods of 0, 1, 3, 7, 14, and 30 days. After calculating all the frequency variables for the five data fields and categorical candidate variables, we ended up with 30 frequency candidate variables. The list of the frequency candidate variables is presented in figure 6 below.

	Frequency Candidate Variable		Frequency Candidate Variable
1	Cardnum_velocity0_Date	16	Cardnum_velocity1_Date
2	Cardnum_velocity3_Date	17	Cardnum_velocity7_Date
3	Cardnum_velocity14_Date	18	Cardnum_velocity30_Date
4	Merchnum_velocity0_Date	19	Merchnum_velocity1_Date
5	Merchnum_velocity3_Date	20	Merchnum_velocity7_Date
6	Merchnum_velocity14_Date	21	Merchnum_velocity30_Date
7	Cardnum_|_zip_velocity0_Date	22	Cardnum_|_zip_velocity1_Date
8	Cardnum_|_zip_velocity3_Date	23	Cardnum_|_zip_velocity7_Date
9	Cardnum_|_zip_velocity14_Date	24	Cardnum_|_zip_velocity30_Date
10	Cardnum_|_state_velocity0_Date	25	Cardnum_|_state_velocity1_Date
11	Cardnum_|_state_velocity3_Date	26	Cardnum_|_state_velocity7_Date
12	Cardnum_|_state_velocity14_Date	27	Cardnum_|_state_velocity30_Date
13	Cardnum_|_merchnum_velocity0_Date	28	Cardnum_|_merchnum_velocity1_Date
14	Cardnum_|_merchnum_velocity3_Date	29	Cardnum_|_merchnum_velocity7_Date
15	Cardnum_|_merchnum_velocity14_Date	30	Cardnum_|_merchnum_velocity30_Date

Figure 6. Table of Frequency Candidate Variables

Days-Since Candidate Variables

The days-since candidate variables are numeric in nature and capture the number of days since the last time a categorical data field or candidate variable was seen for a particular transaction record. Since fraudsters often operate in bursts of activity, calculating the number of days between transactions is another way of finding and characterizing these bursts of activity as we seek to identify potentially fraudulent transactions. Apart from the value of 0, a lower calculated value for this type of variable means there is increased risk and a higher likelihood of fraud. The general equation for calculating the value for a days-since variable is as follows:

$$days\text{-}since \;=\; \#\,of\ days\ since\ a\ \boldsymbol{data\ field}\ was\ last\ seen$$

$$days\text{-}since \;=\; \#\,of\ days\ since\ a\ \boldsymbol{variable}\ was\ last\ seen$$

From a more visual perspective, this equation can be viewed as follows:

$$\text{Days} - \text{since} = \#\text{ of days since this} \left\{ \begin{array}{c} \text{card} \\ \text{merchant} \\ \text{card at this merchant} \\ \text{card in this zip code} \\ \text{card in this state} \end{array} \right\} \text{was seen}$$

Figure 7 provides the data fields and categorical candidate variables that served as the basis for the five days-since variables we calculated for our analysis.

	Days-Since Candidate Variable	Base Data Field or Variable
1	Cardnum_daysSince	Cardnum
2	Merchnum_daysSince	Merchnum
3	card-merch_daysSince	Cardnum-Merchnum
4	card-state_daysSince	Cardnum-Merch state
5	card-zip_daysSince	Cardnum-Merch zip

Figure 7. Days-Since Candidate Variables.

As an example, let us revisit the transaction records with a “Cardnum” value of “5142224817” in the dataset as seen in figure 8 below. On 01/03/2010, it was the very first day we saw the “Cardnum” value of “5142224817” in the dataset. Thus, index number 83 had a days-since value of 0. However, since the “Cardnum” value of “5142224817” was used over after two days on 01/05/2010 and then on 01/07/2010, each of the subsequent index numbers have a days-since value of two because there was only two days or less since that card number was last seen. Similarly, the card was used again on 01/19/2010; hence, the transaction with index number 4049 had a days-since value of 12.

Index	Date	Cardnum	Cardnum_daysSince
83	01/03/2010	5142224817	0
610	01/05/2010	5142224817	2
704	01/05/2010	5142224817	2
1148	01/07/2010	5142224817	2
1234	01/07/2010	5142224817	2
1389	01/07/2010	5142224817	2
4049	01/19/2010	5142224817	12

Figure 8. Days-Since Candidate Variables for Cardnum.

Velocity Change Candidate Variables

The velocity change candidate variables are numeric in nature and capture the speed at which a categorical data field or candidate variable was seen for a transaction record over a certain period of time in relation to what was considered normal. This candidate variable is a comparative type of variable where we analyze how often we see a data field or variable during a short period of time (e.g. 0-day or 1-day time periods) in comparison to how often we see that data field or variable over a longer period of time (e.g. 7-day, 14-day, or longer time periods). We are therefore using this type of variable as an indication of fraud by determining if we are seeing more transaction records with a particular data field or variable within a short period of time versus how often we would normally see a transaction record with that data field or variable. A higher value for the velocity change variables correlates to increased risk and higher likelihood of fraudulent activity. The general equation for calculating the value for the velocity change variable is as follows:

$$\text{Velocity change Variables} = \frac{\begin{pmatrix}\text{Number}\\ \text{Amount}\end{pmatrix} \text{ of transactions with same } \begin{pmatrix}\text{card}\\ \text{merchant}\end{pmatrix} \text{ over the past } \begin{pmatrix}\text{0 days}\\ \text{1 day}\end{pmatrix}}{\text{Average daily } \begin{pmatrix}\text{number}\\ \text{amount}\end{pmatrix} \begin{matrix}\text{of transactions}\\ \text{with same}\end{matrix} \begin{pmatrix}\text{card}\\ \text{merchant}\end{pmatrix} \text{ over the past } \begin{pmatrix}\text{7 days}\\ \text{14 days}\\ \text{30 days}\end{pmatrix}}$$

The equation above was used for each of the three categorical candidate variables we created and for the “Cardnum” and “Merchnum” data fields. For the numerator (recent past) values, we used the 0-day and 1-day velocity candidate variable values. For the denominator (past days) values, we used the 7-day, 14-day, and 30-day frequency candidate variable values. After calculating all the velocity change variables for the data fields and categorical candidate variables, we ended up with 60 velocity change candidate variables. A list of the velocity change candidate variables is provided in figure 9 below. For the full list of the velocity change candidate variables, please see Appendix B.

	Velocity Change Candidate Variable		Velocity Change Candidate Variable
1	Cardnum_0_dayamount_div_7_dayamount_velchange	**21**	Cardnum_0_dayvel_div_7_dayvel_velchange
2	Cardnum_0_dayamount_div_14_dayamount_velchange	**22**	Cardnum_0_dayvel_div_14_dayvel_velchange
3	Cardnum_0_dayamount_div_30_dayamount_velchange	**23**	Cardnum_0_dayvel_div_30_dayvel_velchange
4	Cardnum_1_dayamount_div_7_dayamount_velchange	**24**	Cardnum_1_dayvel_div_7_dayvel_velchange
5	Cardnum_1_dayamount_div_14_dayamount_velchange	**25**	Cardnum_1_dayvel_div_14_dayvel_velchange
6	Cardnum_1_dayamount_div_30_dayamount_velchange	**26**	Cardnum_1_dayvel_div_30_dayvel_velchange
7	Merchnum_0_dayamount_div_7_dayamount_velchange	**27**	Merchnum_0_dayvel_div_7_dayvel_velchange
8	Merchnum_0_dayamount_div_14_dayamount_velchange	**28**	Merchnum_0_dayvel_div_14_dayvel_velchange
9	Merchnum_0_dayamount_div_30_dayamount_velchange	**29**	Merchnum_0_dayvel_div_30_dayvel_velchange
10	Merchnum_1_dayamount_div_7_dayamount_velchange	**30**	Merchnum_1_dayvel_div_7_dayvel_velchange
11	Merchnum_1_dayamount_div_14_dayamount_velchange	**31**	Merchnum_1_dayvel_div_14_dayvel_velchange
12	Merchnum_1_dayamount_div_30_dayamount_velchange	**32**	Merchnum_1_dayvel_div_30_dayvel_velchange
13	Cardnum_|_zip_0_dayamount_div_7_dayamount_velchange	**33**	Cardnum_|_zip_0_dayvel_div_7_dayvel_velchange
14	Cardnum_|_zip_0_dayamount_div_14_dayamount_velchange	**34**	Cardnum_|_zip_0_dayvel_div_14_dayvel_velchange
15	Cardnum_|_zip_0_dayamount_div_30_dayamount_velchange	**35**	Cardnum_|_zip_0_dayvel_div_30_dayvel_velchange
16	Cardnum_|_zip_1_dayamount_div_7_dayamount_velchange	**36**	Cardnum_|_zip_1_dayvel_div_7_dayvel_velchange
17	Cardnum_|_zip_1_dayamount_div_14_dayamount_velchange	**37**	Cardnum_|_zip_1_dayvel_div_14_dayvel_velchange
18	Cardnum_|_zip_1_dayamount_div_30_dayamount_velchange	**38**	Cardnum_|_zip_1_dayvel_div_30_dayvel_velchange
19	Cardnum_|_state_0_dayamount_div_7_dayamount_velchange	**39**	Cardnum_|_state_0_dayvel_div_7_dayvel_velchange
20	Cardnum_|_state_0_dayamount_div_14_dayamount_velchange	**40**	Cardnum_|_state_0_dayvel_div_14_dayvel_velchange

Figure 9. Velocity Change Candidate Variables.

Benford's Law Variables

Since Benford's law can assist in detecting fraud, we utilized Benford's law to create an extra candidate variable (feature) for the credit card transaction dataset. Benford's law states that in naturally occurring sets of numbers, such as physical constants, the distribution of the leading digit d is given by $P(d) = log_{10}\left(1+\frac{1}{d}\right)$, where the likelihood of the leading digit d has a higher probability of being a lower digit (e.g. 1, 2, or 3) than a higher digit (e.g. 7, 8, 9). The Benford's law distribution is shown in figure 10.

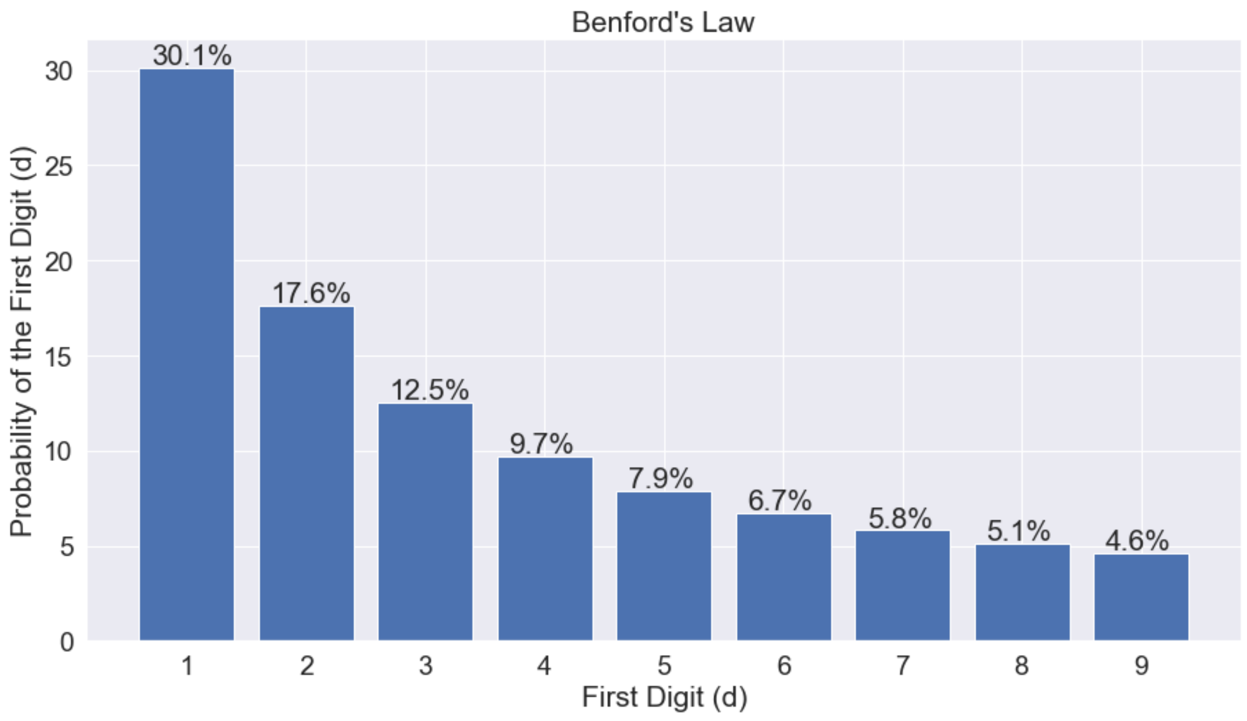


Figure 10. Benford's Law Distribution.

Benford's law often applies well to entities that have units; hence, it can be applied to monetary transactions and it is commonly used to detect fraudulent activity in accounting or financial data. For instance, in a scenario with no fraudulent transactions, it is expected that the distribution of leading digits for the transaction amounts would follow the Benford's law distribution. If the distribution of the leading digits fails to follow Benford's law logarithmic curve, then the collection of numbers can be characterized as violating Benford's law. With a violation of Benford's law, there is a possibility of fraudulent activity associated with that collection of numbers.

In applying Benford's law within the accounting realm, it is important to understand that there are limitations to its usefulness. Of particular note, Benford's law is only useful when there is a large quantity of numbers being created or altered by a potential fraudster. Benford's law will therefore be unable to detect any possible fraudulent activity if only a small quantity of numbers were created or altered within a collection of numbers. Additionally, Benford's law may not be useful in detecting possible fraudulent activity when the numbers in a dataset are generated by a variety of different people where the majority of the individuals in that group of people are honest and not committing fraud. Benford's law is therefore best in detecting possible fraudulent activity when there is a single entity (i.e. a single merchant or a single person) generating or making up a lot of the numbers in a collection of numbers.

To detect violations of Benford's law on this relatively small dataset, we needed to determine a separate measure of unusualness to quantify how much the distributions of the leading digits in the "Amount" data field violated Benford's law for each of the "Cardnum" or "Merchnum" values. The reason for this separate measure of unusualness was to help account for the small quantity of records and for the fact that there would not be enough records to fill all nine bins of the leading digit values. Thus, we called this modified Benford's law measurement U and we reduced the quantity of bins (i.e. we widened the bins) to having only two bins where a low bin contained leading digit values of 1 and 2, and a high bin contained leading digit values of 3 through 9. The specific steps are outlined below:

1) As a preprocessing step, we removed all the records where the "Merch description" data field was associated with "FEDEX". This step was necessary since there was a high count of "FEDEX" transactions with a leading digit of 3. After consulting with our field domain experts, we were advised to remove those particular records as they would negatively impact the Benford's law calculations.
2) For each "Merchnum" and "Cardnum" value, we found the number of leading digits in the "Amount" data field greater than or equal to 3 (n_{high}) and the number of leading digits in the "Amount" data field less than or equal to 2 (n_{low}). We then identified these quantities as the high and low bins where the high bin was set to the value of $n_{high}$and the low bin was set to the value of n_{low}. Also, if either n_{high}or $n_{low}$was 0, they were set to 1.
3) From a theoretical perspective, the ratio of high bins to low bins is n_{high}/n_{low}= 1.096. Thus, the following quantities were further defined:
   a) $R = 1.096\frac{n_{low}}{n_{high}}$, which should be close to 1 for non-fraudulent activity.
   b) $U = max(R, 1/R)$ which should be close to 1 regardless of which of n_{high} or n_{low} is greater.
   c) $U^* = 1 + \frac{U-1}{1+e^{-t}}, t = \frac{n-n_{mid}}{c}$. This is a smoothing formula used on the modified Benford's law measurement U to address entities (merchants or card numbers) that do not have a sufficient number of transactions. The parameters used for t were set to $n_{mid} = 15$ and $c = 3$. The value obtained from the calculation of U^* then became the final value for our Benford's law measurement.
4) Finally, we assigned the "FEDEX" associated records a Benford's law variable value of 1.

Based on the above, a U^* value close to 1 indicated that the activity associated with merchant or credit card transaction followed Benford's law and thus was unlikely to be fraudulent. In contrast, a high value of U^* served as a higher indication of potentially fraudulent activity associated with a particular merchant or credit card number. For this analysis the value for U^* was stored in the following Benford's Law candidate variables:

	Benford's Law Candidate Variables
1	Cardnum_U*
2	Merchnum_U*

Figure 11. Benford's Law Distribution Candidate Variables.

As an example, let us take a look at a particular “Cardnum” value and its respective calculations for the Benford’s law candidate variable as shown in figure 12. For the “Cardnum” of 5142204384, we can see that since all of the transactions have a leading digit of 2, the n_{low} count is increased by 1 for each transaction, whereas n_{high} is set to 1 in every data field in order to avoid a value of 0 when conducting the calculation of R. We can then observe that the “Cardnum_U*” candidate variable is increasing as the number of transactions with a leading digit of 2 increases. This increase in value for the “Cardnum_U*” candidate variable means that this particular “Cardnum” value is starting to deviate from the Benford’s Law distribution and is trending towards potentially fraudulent activity.

Date	Cardnum	Amount	n_{low}	n_{high}	R	U	n	Cardnum_U*
01/02/2010	5142204384	20.15	1	1	1.096	1.096	2	1.0012
01/02/2010	5142204384	20.15	2	1	0.548	1.8248	3	1.0148
01/02/2010	5142204384	20.15	3	1	0.3653	2.7372	4	1.0433
01/02/2010	5142204384	20.15	4	1	0.274	3.6496	5	1.0913
01/02/2010	5142204384	20.15	5	1	0.2192	4.562	6	1.1689
01/02/2010	5142204384	20.15	6	1	0.1827	5.4745	7	1.2907
01/02/2010	5142204384	20.15	7	1	0.1566	6.3869	8	1.4762
01/02/2010	5142204384	28.13	8	1	0.137	7.2993	9	1.7509
01/02/2010	5142204384	20.15	9	1	0.1218	8.2117	10	2.1457
01/02/2010	5142204384	20.15	10	1	0.1096	9.1241	11	2.6948

Figure 12. Benford’s Law Candidate Variables for Cardnum.

Day-of-Week Risk Table Variable

Analyzing credit card transaction activity associated with the day-of-week when a transaction occurred can sometimes assist in detecting fraud. We therefore created a candidate variable for the day-of-week associated with each transaction in the dataset. However, since this information is categorical in nature, we needed to convert the various days of the week to a numerical type in order to enable proper machine learning modeling. To do so, we encoded the day-of-week for each transaction by using a risk table (also known as target encoding).

The general method for using risk tables as a means to encode categorical information simply replaces the categorical information with the average of the target value associated with each category. The “target value” is the dependent variable or the value our model is trying to predict. This allows us to encode an arbitrary number of categories without increasing the dimensionality of our data. Unfortunately, by using the average value of the dependent variable, there is some target leakage that occurs that can cause overfitting. To avoid overfitting when using risk tables, we must remove the validation data before conducting any calculations for the risk tables. Additionally, we must also ensure that we have a statistically sufficient number of records in each category and use a smoothing formula when we do not have enough records in a particular category.

In applying this concept to the dataset for our analysis, we first transformed the “Date” data field to the name of day-of-week by using panda’s “.dt.weekday_name” method on our dataset so that this value could serve as our categories. We then removed the validation data to help prevent overfitting which left us with the transaction records that occurred before 11/1/2010 to build the risk table.

For each day of the week, we summed up the records whose fraud label was 1 and divided that sum by the count of all the transaction records that occurred on the same day of the week. Thus, we calculated the fraud ratio for each day of the week.

Next, a smoothing formula was applied in order to account for scenarios in which there were an insufficient number of records in a particular category. For the smoothing formula, we set $c = 4$ and $n_{mid} = 20$. Y_{low} is the fraud rate of a certain "Day_of_week", Y_{high} is the overall fraud rate.

$$Value = Y_{low} + \frac{Y_{high} - Y_{low}}{1 + e^{-(n-n_{mid})/c}}$$

The following is an extract of the "Day_of_week" risk table from all the records:

Index	Date	Day_of_week
1	01/01/2020	0.026
52	01/02/2020	0.01
81	01/03/2020	0.0096
240	01/04/2020	0.0087
469	01/05/2020	0.0071
778	01/06/2020	0.0098
1108	01/07/2020	0.0186

Figure 13. Day of Week Risk Table.

Feature Selection Process

Feature selection is the process in statistics that aims to reduce the number of features (dimensions) of input variables. Feature selection is performed in such a way so that the most statistically important features are kept from the original input. Also, features that are either highly correlated with others or not significant to perform an accurate prediction are ignored. Feature selection is pivotal in modern statistical learning as it often allows reduced computational costs and increased model performance. There are three main methods of feature selection:

1) Filter Methods: Filter methods use statistical measures to evaluate the relationship (correlation) of two distributions and measure the correlation between the distribution of each of the classes of each feature and the dependent variable. The features that are chosen are the ones with the highest correlation with the dependent variable.
2) Wrapper Methods: Wrapper methods utilize statistical models to evaluate the performance of each feature (or a subset of features) based on a performance metric (accuracy, AUC, f1 score, etc.). A common wrapper method is recursive feature elimination, in which a model recursively uses smaller and smaller sets of features until a desired number of features is reached.
3) Embedded Methods: Embedded methods perform feature elimination as the model is built. A common embedded method for feature selection is regularization, in which a norm is included in the loss function of a statistical model to penalize the number of features used.

Filter Methods

For our analysis, we performed feature selection using the Kolmogorov-Smirnov distance and fraud detection rate.

Kolmogorov–Smirnov (KS) distance:

Kolmogorov-Smirnov (KS) distance measures the maximum distance between two distributions to determine how well the distributions are separated. A higher KS distance value correlates to a better separation between the two distributions and therefore a better feature in the context of feature selection. For this analysis, we used the KS distance metric by calculating the univariate KS value as a filtering method to aid in determining which features provide a better separation between the “Fraud” values of 1 and 0. Meaning, for each numerical candidate variable, we generated the distribution of the two classes (1 and 0) based on the dependent variable ("Fraud” data field). Subsequently, we measured the KS distance between the distributions of the two classes for each of the numerical candidate variables. More formally,

$$KS = max_x = \sum_{x_{min}}^{x} (P_{goods} - P_{bads})$$

We then rank ordered the KS distance value from high to low for each of the numerical candidate variables and used this ranking to evaluate the importance of each variable.

Fraud Detection Rate (FDR) at 3%:

The second metric we used for filtering the features was the Fraud Detection Rate (FDR). In general, the FDR is the percentage of all the frauds that are detected up to a particular cutoff point. In the context of this analysis for feature selection, we used a cutoff threshold of 3% and calculated the univariate FDR for each numerical candidate variable. The FDR at 3% was determined by first sorting the numerical candidate variables in descending order, and then computing the percentage of frauds in the top 3%. We then assigned a rank for each of the numerical candidate variables and used this ranking as a means to evaluate the importance of each variable.

In our analysis, we obtained the univariate KS distance and the univariate FDR at 3% for each numerical candidate variable and used their average rank to serve as a final score for the importance of each candidate variables. Subsequently, we discarded the lowest 260 ranked numerical candidate variables, reducing the total number of features from 340 to 80. In figure 14 we present the KS, FDR and final ranking score normalized from 0 to 1 in descending order (higher ranking score indicates higher feature importance). The score threshold is shown by the green dashed line and indicates the score cutoff of the top 80 features. Also, note that the "Fraud" column had KS and FDR scores of 1 which validates our filter method.

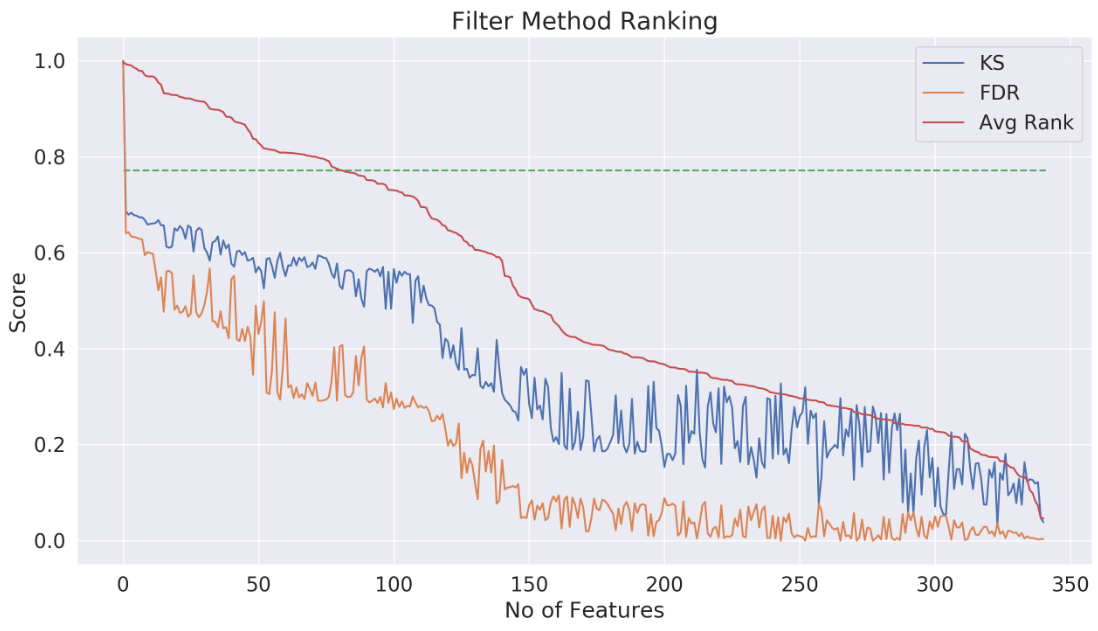


Figure 14. Filter Method Scores.

Wrapper Methods

We used Decision Trees, Random Forest and Boosted Trees as our models for the wrapper method for feature (candidate variable) selection. These models are simple to implement with a relatively low computational cost for their default parameters. Common measures for feature importance in these models are the mean decrease in accuracy (misclassification) or the mean decrease in node impurity (Gini index), which represents how well the trees split the data. We evaluated feature importance as the decrease in node impurity weighted by the probability of reaching that node in each model. Similar to the filter method, we then ranked the feature importance score for each model and averaged the ranking to obtain a final rank for each feature. With our final ranking we then kept the top 30 features for our models.

In figure 15, we present the final score of the wrapper method for the 80 features in descending order.

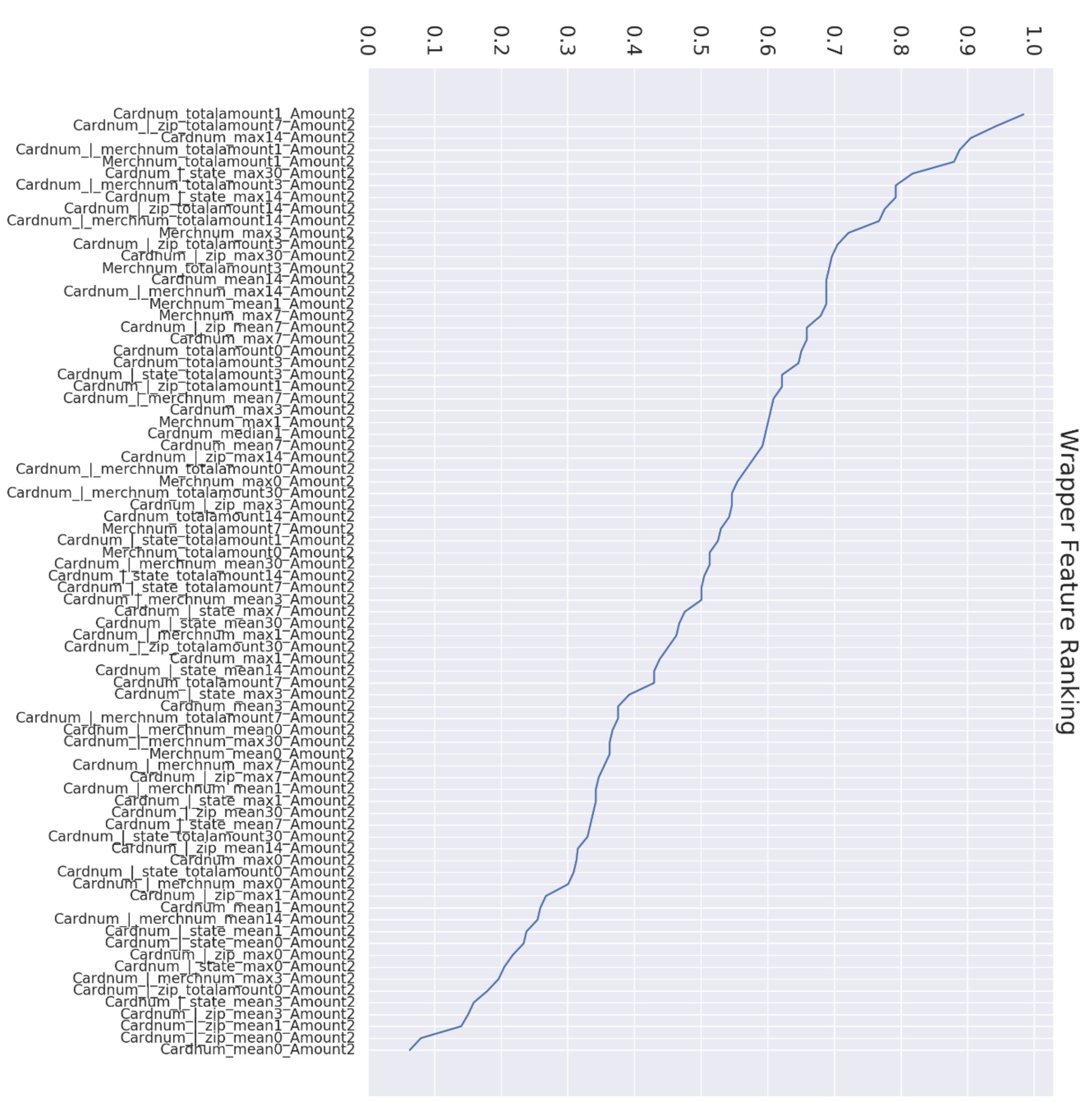


Figure 15. Wrapper Method Scores.

The final variables that were selected are presented below.

Final Variables	
Cardnum_totalamount1_Amount2	Cardnum_|_zip_totalamount7_Amount2
Cardnum_max14_Amount2	Cardnum_|_merchnum_totalamount1_Amount2
Merchnum_totalamount1_Amount2	Cardnum_|_state_max30_Amount2
Cardnum_|_merchnum_totalamount3_Amount 2	Cardnum_|_state_max14_Amount2
Cardnum_|_zip_totalamount14_Amount2	Cardnum_|_merchnum_totalamount14_Amount 2
Merchnum_max3_Amount2	Cardnum_|_zip_totalamount3_Amount2
Cardnum_|_zip_max30_Amount2	Merchnum_totalamount3_Amount2
Cardnum_mean14_Amount2	Cardnum_|_merchnum_max14_Amount2
Merchnum_mean1_Amount2	Merchnum_max7_Amount2
Cardnum_|_zip_mean7_Amount2	Cardnum_max7_Amount2
Cardnum_totalamount0_Amount2	Cardnum_totalamount3_Amount2
Cardnum_|_state_totalamount3_Amount2	Cardnum_|_zip_totalamount1_Amount2
Cardnum_|_merchnum_mean7_Amount2	Cardnum_max3_Amount2
Merchnum_max1_Amount2	Cardnum_median1_Amount2
Cardnum_mean7_Amount2	Cardnum_|_zip_max14_Amount2

Figure 16. Final Variables.

In a quick review of the variables that were chosen, we noticed that all the variables were related to the "Amount" data field. A likely reason for this occurrence may be due to the fact that these variables not only include the monetary amount of a transaction, but also inherently capture the number of transactions in the average, median and total amount variables.

Model Algorithms

Logistic Regression

Logistic regression is one of the simplest and commonly used machine learning algorithms for binary classification problems. It is derived from a logit model which tries to predict the log of odds of a model as a linear combination of the predictor(s):

$$log\big(O(Y = 1|X = x)\big) = \beta_0 + \beta_1 X$$

With some rearrangement, we can obtain the formula for logistic regression:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Logistic regression follows a sigmoid function. It describes and estimates the relationship between one dependent binary variable and the independent variables. The sigmoid function gives an ‘S’ shaped curve that can take any real-valued number and map it into a value between 0 and 1 as seen in figure 17 below. If the output of the sigmoid function is more than 0.5, we can classify the outcome as 1, and if it is less than 0.5, we can classify it as 0.

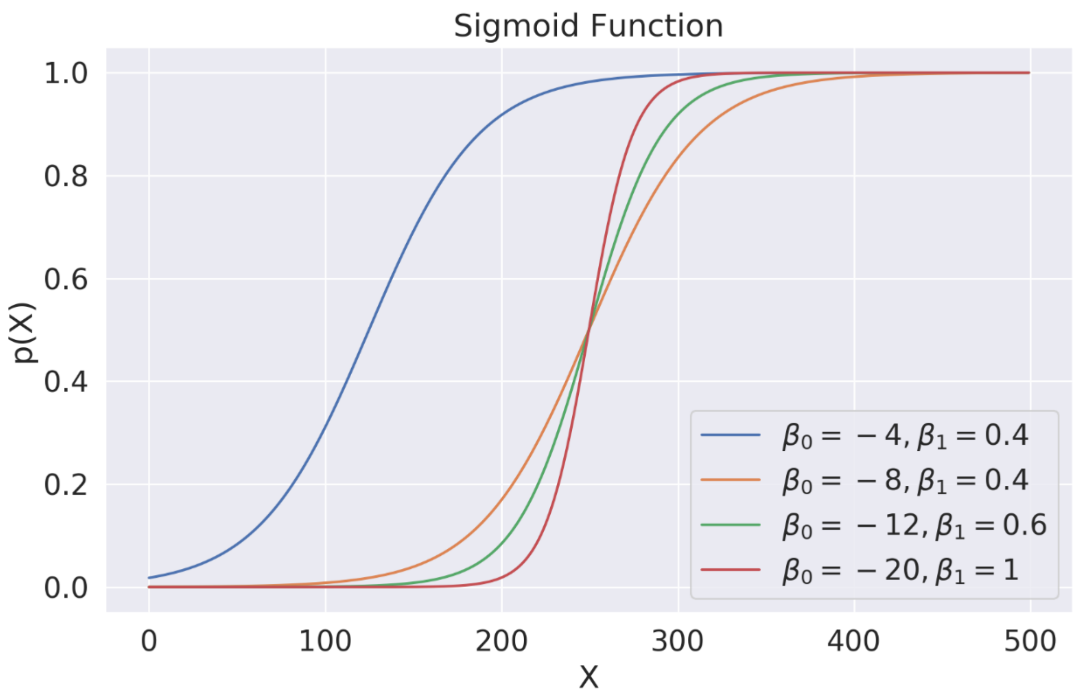


Figure 17. Sigmoid Function.

Parameters control the shape and location of the sigmoid curve, where β_0 controls the location of midpoint and β_1 controls the rise of the slope.

For our fraud analysis, we predicted the fraud label using logistic regression, ranked the records by probability in descending order and calculated the FDR at 3% for training, testing and OOT data separately. We used 30 variables and we tried different combinations of parameters to locate the model of best performance. Below we present the average FDR results for different logistic regression models. The parameters of our best logistic regression model were:

cv=10, Cs=0.001, L2, class_weight= {0:1,1:91}

Logistic Regression	# of Variables	CV	class_weight	Cs	Regularization	TRAIN FDR	TEST FDR	OOT FDR
1	30	10	(1, 91)	0.001	L2	0.7146	0.7058	0.4480
2	30	10	(1, 91)	1	L2	0.6500	0.6481	0.3259
3	30	10	(1, 91)	10	L1	0.6925	0.7120	0.4190
4	30	10	(1, 91)	10	L2	0.7016	0.7008	0.3923
5	30	10	(1, 91)	100	L2	0.7001	0.7032	0.4245

Figure 18. Logistic Regression Model Results.

Boosted Trees

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

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

As the boosted tree algorithm progresses, it applies a weight (w_t) to each data record in the dataset relative to the accuracy of the output. The weights (w_t) for a record are therefore dependent on whether a record was misclassified with weights (w_t) increasing for misclassified records as it is subsequently misclassified throughout the algorithm so that each iterative decision tree in the series can place a heavier importance on that record to increase the likelihood of properly classifying the record. The overall idea is that by fitting a series of weak learners to the residuals (i.e. misclassified results), we slowly improve the model in areas where it does not perform well. The ultimate goal is to minimize the losses or the residual error in the objective function to improve prediction results and accuracy. Refer to figure 19 below for a visualization of boosted trees.

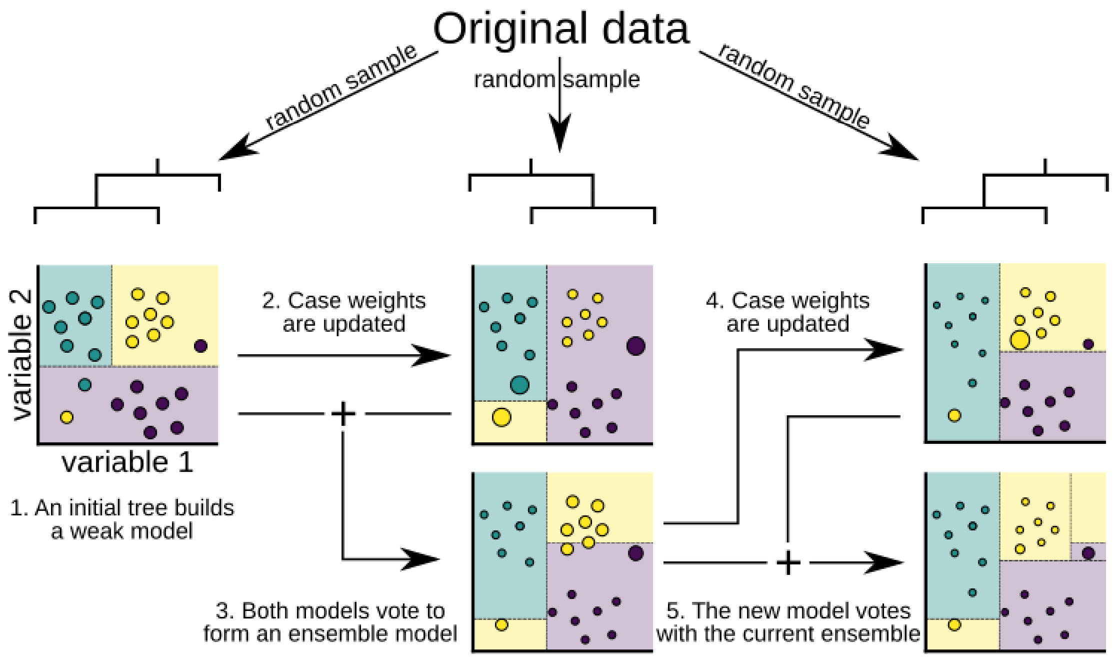

Original data
random sample
random sample
random sample
variable 2
variable 1
1. An initial tree builds a weak model
2. Case weights are updated
3. Both models vote to form an ensemble model
4. Case weights are updated
5. The new model votes with the current ensemble

Figure 19. Boosted Trees Algorithm.

There are a variety of boosted tree algorithms readily available for use within many of the popular machine learning libraries. For instance, Scikit-Learn and XGBoost are two popular machine learning libraries that contain boosted tree algorithms. For this dataset, we opted to use the XGBClassifier package from XGBoost's machine learning library for our boosted tree algorithm.

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

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

and more time will be needed for the model to finish training. Common values for the learning rate are 0.01 and 0.001. However, when tuning this parameter, higher values may prove to be more optimal if the lower values show to result in little to no improvement.

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

In applying the XGBClassifier to the dataset, we used a short list of values for each of the parameters above to conduct an initial analysis of the performance of the algorithm on the dataset. We also wanted to conduct an initial attempt at dealing with the unbalanced classes by downscaling the majority class to 75% and using the “scale_pos_weight” parameter with a value obtained by using the $sum(negative\ instances)\ /\ sum(positive\ instances)$. Lastly, we ran 10 iterations with different random states used for the training and testing splits on the dataset for each set of parameters and averaged the FDR results over those 10 iterations due to the inherent variances that can occur with random initializations. The results are shown in figure 20 below.

Boosted Tree	Num of Variables	Num of Trees	Max Depth	Learning Rate	TRAIN FDR	TEST FDR	OOT FDR
1	30	600	3	0.01	0.8839	0.8226	0.5704
2	30	600	4	0.01	0.9377	0.8663	0.5274
3	30	600	5	0.01	0.9753	0.8843	0.4704
4	30	800	3	0.01	0.9181	0.8494	0.5536
5	30	800	4	0.01	0.9562	0.8954	0.5078
6	30	800	5	0.01	0.9928	0.8931	0.4240
7	30	1000	3	0.01	0.9343	0.8736	0.5430
8	30	1000	4	0.01	0.9789	0.8931	0.5151
10	30	600	3	0.001	0.7008	0.6743	0.4983
11	30	600	4	0.001	0.7700	0.7176	0.4603
12	30	600	5	0.001	0.8400	0.7234	0.4145
13	30	800	3	0.001	0.7082	0.6678	0.4983
14	30	800	4	0.001	0.7952	0.7287	0.4514
15	30	800	5	0.001	0.8623	0.7460	0.3961
16	30	1000	3	0.001	0.7237	0.6812	0.5084
17	30	1000	4	0.001	0.8059	0.7295	0.4626
18	30	1000	5	0.001	0.8616	0.7590	0.4134
19	30	600	3	0.0001	0.6442	0.6276	0.4067
20	30	600	4	0.0001	0.7092	0.6778	0.4849
21	30	800	3	0.0001	0.6633	0.6383	0.4777
22	30	800	4	0.0001	0.7053	0.6693	0.5330
23	30	1000	3	0.0001	0.6595	0.6467	0.4665
24	30	1000	4	0.0001	0.7138	0.6828	0.5173

Figure 20. Boosted Trees Results.

In addition to the results above, we attempted to use cross-validation and parameter tuning with Scikit-Learn's GridSearchCV package to evaluate the XGBClassifier with additional values for each of the three main parameters seen in figure 20. Unfortunately, we found this method caused overfitting and did not enable our model to improve. Given the lack of improvement and the results from our initial analysis on the other algorithms, we opted to forego any further detailed hyperparameter tuning for boosted trees and the XGBClassifier model.

Neural Network

Neural networks are statistical models widely used today in many applications, including classification, image processing, and speech processing among others. The neural network is based on the perceptron algorithm originally introduced in the 1950s and consists of multiple layers of perceptrons (or neurons). The neural network tries to imitate the function of a human brain, i.e., try to learn things, distinguish patterns and make decisions by training, in the same way that a human brain does.

A neural network consists of multiple layers of neurons:

- an input layer that is formed by the independent variables
- hidden layers
- the output layer which is formed by the dependent variable

Each of the neurons of each layer is connected to all neurons of the next layer, i.e., the output of a neuron on the i-th layer is the input of the neurons in the $(i+1)$-th layer. Each neuron has an optional threshold and an activation function, and each neuron connection has an associated weight that represents the significance of this connection. Each neuron receives as input a weighted signal and outputs the result of its activation function of that weighted signal. Subsequently, this output is multiplied by the weight of the neuron's output connection and is propagated to all the neurons in the next layer. The activation function serves to scale the input of the neuron and to provide a smooth, differentiable transition as input values change. Common activation functions are relu, sigmoid and tanh. In figure 21, the architecture of a neural network with 3 hidden layers containing 5, 4 and 3 nodes respectively is presented.

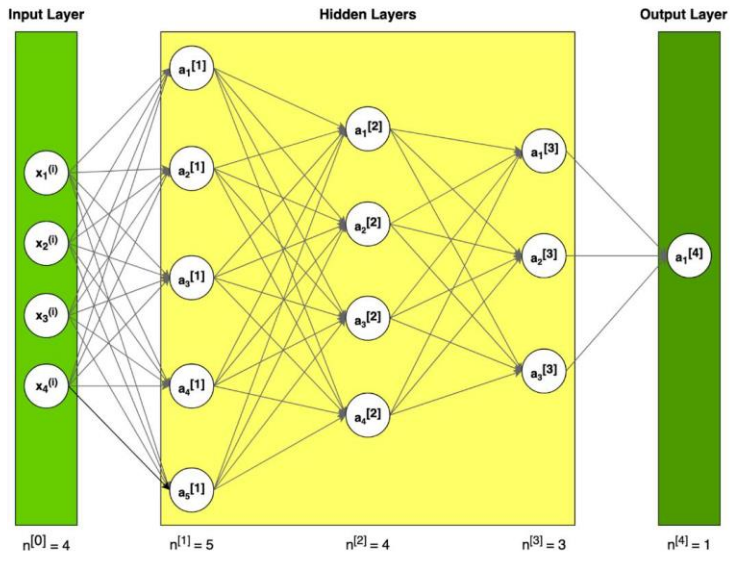


Figure 21. Neural Network Architecture.

A neural network tries to predict the dependent variable (or label) of an input vector (independent variable). This is achieved by calculating the appropriate weights of the connections of the neurons in the network. Thus, the weights become a variable that we are trying to optimize over all available data in the input so that we have more accurate predictions as possible. For that matter, we define a loss function which is parameterized with the weights. More specifically, the loss function is defined as a convex function of the difference of the predicted and real dependent variables (estimation error) and our aim is to find the weights that produce the global minimum of the loss function, i.e., the weights that result to the minimum estimation error. Common loss functions used in neural networks are the mean square error (MSE), mean absolute error (MAE), binary crossentropy (BCE) etc. The optimal weights are found using the gradient descent algorithm (and its variations) in the loss function for each data point in the input. An iteration through all the data points of the input is called epoch. In figure 22 we can see the loss function of a neural network over 800 epochs.

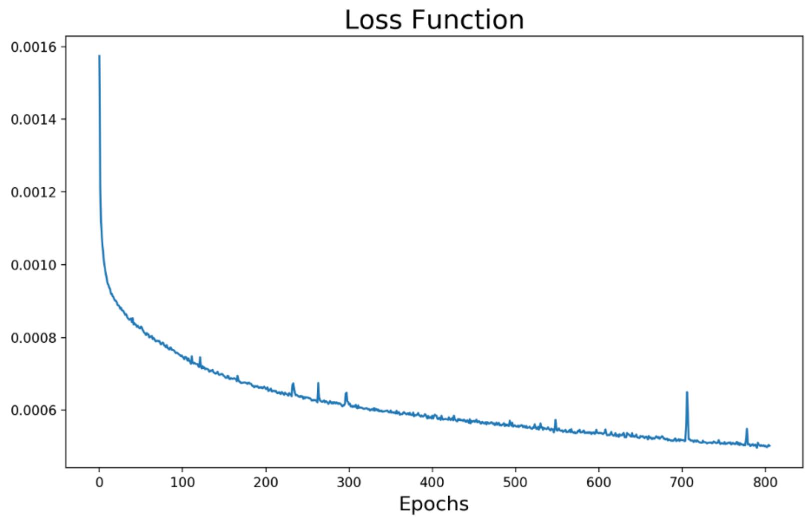


Figure 22. Loss function for Neural Network.

A neural network has a significant number of hyperparameters. To obtain the best results possible, we performed hyperparameter tuning with exhaustive search over a grid of parameters. The parameters tuned were:

- *Number of neurons in the hidden layer*: The number of neurons used in the hidden layer.
- *Number of hidden layers in the network.*
- *Batch size*: The number of samples that are used in one training pass of the network.
- *Number of epochs*: Number of times that the network will be trained with the total number of samples.
- *Optimization algorithm*: The algorithm used to train the network and minimize the loss function.
- *Learning rate and momentum of the optimization algorithm*: Parameters that affect the accuracy and convergence time of the optimization algorithm.
- *Neuron activation function*: The activation function used in each neuron in each layer.

- *Regularization* on the weights of the network.
- *Dropout rate*: Percentage of neurons that can be removed from training at each batch iteration. Both dropout rate and regularization are used to avoid overfitting in a neural network.

For the training process, initially we normalized the training set to mean of 0 and standard deviation of 1 and clipped the range of values between -5 and 5, which significantly improved our results. Given that our input is a numerical matrix, we chose to train a simple network architecture to detect fraud, namely a neural network with one hidden layer containing a few number of nodes. Moreover, we set the class weights to 0.5 for normal and 8 for fraud classes to tackle the imbalance between the classes. Indeed, a network with one hidden layer with 30 nodes trained over 20 epochs produced better FDR results compared to more complicated neural networks. For instance, we trained a second neural network with two hidden layers, where the second hidden layer contained 300 nodes; and a third neural network that was trained over 150 epochs. Unfortunately, these more complicated neural networks caused issues such as overfitting and decreased performance.

In addition to attempting more complicated neural networks, we added dropout rate of 0.2 and L2 regularization in the network's weights to avoid overfitting in our model, a tactic that resulted in similar FDR values with our original simple network.

Our next step was to tune the optimization algorithm and its learning rate. The default algorithm used was Adaptive Moment Estimation (adam), a variation of stochastic gradient descent (SGD) which has a momentum term to speed up the convergence. In our setting, adam outperformed SGD. We also tuned the algorithm's learning rate, which is an important step in neural network training, as it affects the algorithm's ability to converge to the loss function's global optimum which in turn significantly affects the model's metrics results. The results of the FDR for learning rates ranging from 10^{-4} to 1 are presented in figure 23. The optimal learning rate was set to 10^{-2} as it achieved the maximum FDR.

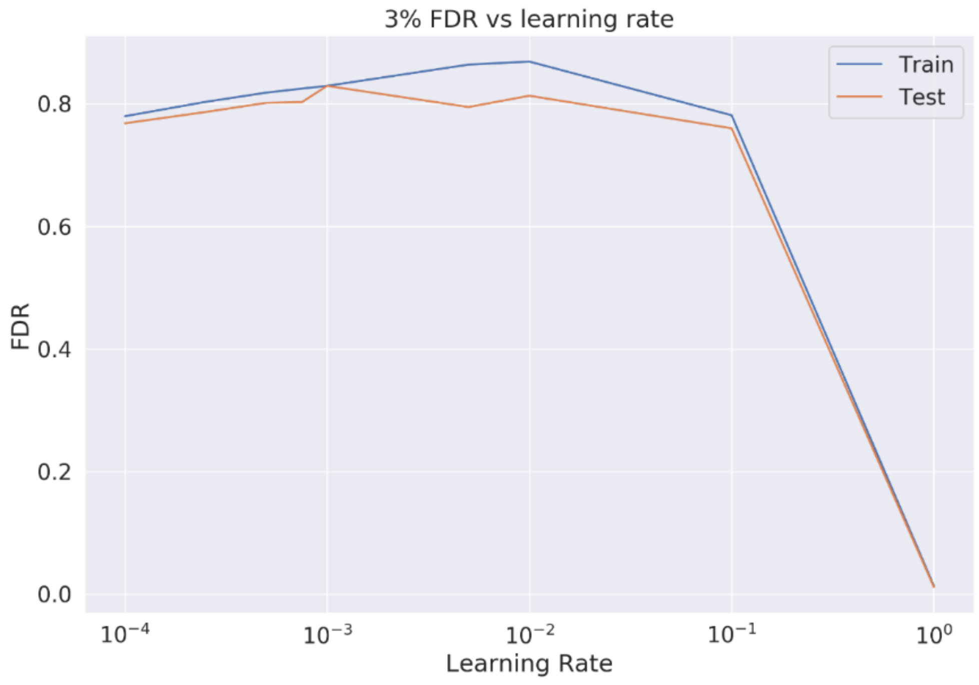


Figure 23. Learning Rate Tuning.

Any further hyperparameter tuning or adjusting the model's decision threshold failed to improve our results as they did not efficiently address our dataset's heavily imbalanced classes. Finally, to further improve our model's performance we analyzed the model's loss function. A common loss function is binary crossentropy defined as follows.

$$BCE = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

Namely, y is the label and $\hat{y}$ is the predicted score of each data entry of our model. When $\hat{y} \to y$, i.e. the predicted probability (score) approaches the actual label, BCE decreases and when the opposite occurs it increases. In our models so far, we have used a weighted version of BCE, in which each term of the equation is multiplied by a weight. That way we assign more importance in classifying correctly one class over the other, which is crucial in an imbalanced dataset. This method achieved improved and consistent results, i.e. it increased the FDR and reduced its the variance over 10 training iterations with different splits of train and test set. In figure 24 we present BCE and weighted BCE. We observed that this weighted version of BCE is skewed towards 1, i.e. it penalizes misclassification of the "Fraud" class.

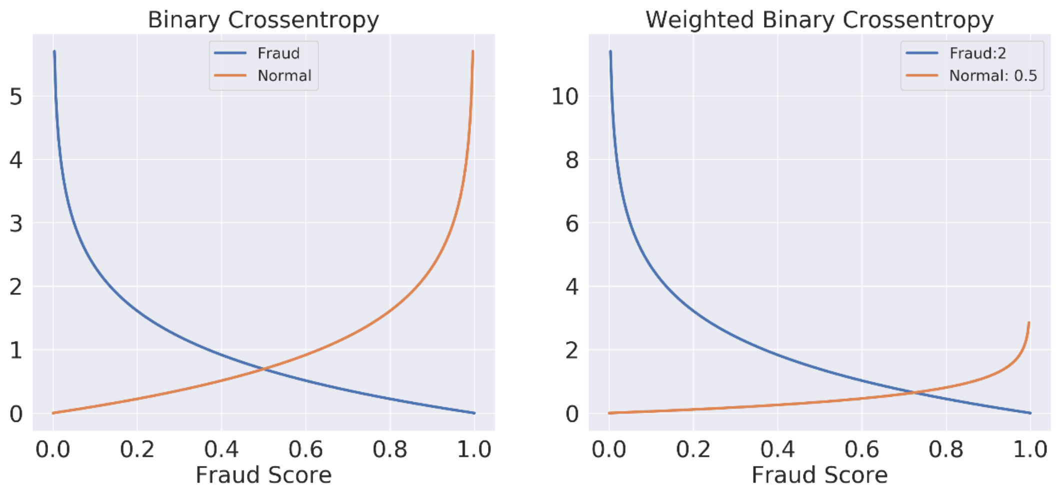


Figure 24. Binary Crossentropy.

Finally, we present the results of our training process for the neural network in figure 25. We see in the following figure that the higher the model complexity the lower is the model performance. In our setting, a simple neural network with a tuned learning rate resulted to the highest FDR.

Neural Network	Layers	Nodes	Epochs	Class weights	Train FDR	Test FDR	OOT FDR	Specifications
1	1	30	20	(0.5,8)	0.7564	0.7862	0.5642	adam, BCE
2	1	150	20	(0.5,8)	0.7130	0.7231	0.5140	adam, BCE
3	1	30	100	(0.5,8)	0.7304	0.7246	0.4749	adam, BCE
4	2	(20,10)	20	(0.5,8)	0.6854	0.6831	0.3911	adam, BCE
5	1	30	20	(0.5,8)	0.7550	0.7692	0.5028	adam, BCE, dropout, L2
6	1	30	20	(0.5,8)	0.6008	0.6154	0.2961	SGD, BCE
7	1	30	20	(0.5,8)	0.7954	0.7231	0.5866	adam, learning rate=0.01, BCE

Figure 25. Neural Network Results.

Random Forest

Random Forest is a statistical algorithm which is a slight improvement from a bagging algorithm. In a bagging algorithm, a number of decision trees are made by taking bootstrapped samples of rows from the dataset involving all the possible number of predictors. However, this algorithm still contains the problem of higher variance from the decision trees. This is because, all the decision trees contain the most powerful predictor among all which induces high correlation between the predictions of the individual predictions of decision trees. As a result, the problem of high variance in the result persists. To overcome this problem, random forest does not involve all the predictors in making the decision trees. It randomly selects the number of predictors and subsequently makes the decision trees. This helps in making a sequence of uncorrelated trees where the most powerful predictor is not always present in every decision tree. This helps in reducing the variance and helps in getting rid of overfitting. Traditionally, we choose the number of predictors, $m = \sqrt{p}$, where p is the total number of predictors in the data. Figure 26 illustrates the differences between decision tree and random forest. All features are included in decision trees, whereas a random subset of the features is included in each tree of the random forest.

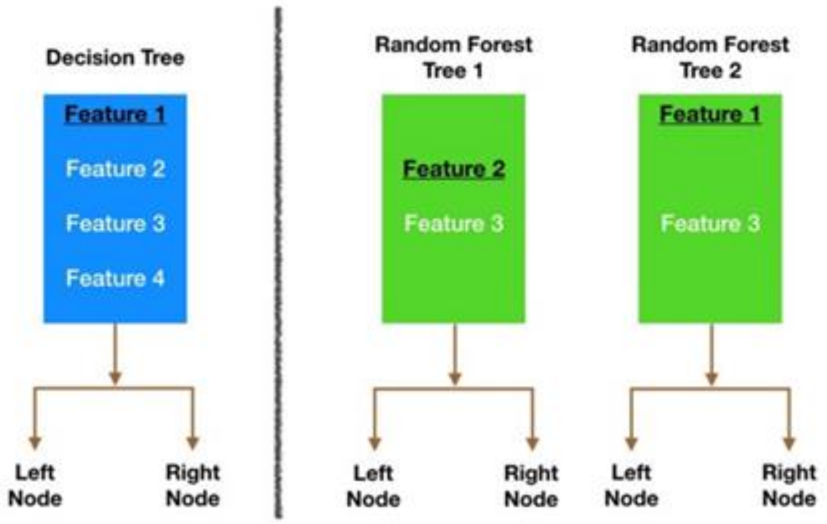


Figure 26. Decision Trees vs. Random Forest Trees

Random forest includes various hyperparameters which we can tune in order to improve our accuracy measure. The following is a short description of all the hyperparameters:

- *Number of Estimators*: The number of trees that the algorithm will create, where the most common class predicted by each individual tree is the final prediction of the algorithm. Too many estimators in the model tends to cause overfitting. The number should be chosen such that there is little or less improvement in the testing accuracy.
- *Max_features*: The maximum number of features that the algorithm can choose to make each decision tree. The most common number of features to be selected are given by $\sqrt{p}$, where p is the total number of predictors.
- *Max_depth*: The depth of each tree in the forest. Higher depth results in higher splits in each tree which may lead to overfitting.

- *Min_samples_leaf*: The minimum number of samples required to be at a leaf node.
- *Min_samples_split*: The minimum number of samples required to split an internal node.
- *Class_weight – "balanced_subsample"*: Used to balance the majority and minority classes of every bootstrapped sample for every tree grown.

The following table is the average of the 10 FDR results of our random forest model with different combinations of hyperparameters:

Random Forest	# of Variables	# of Trees	# of Features	Depth	class_weight	TRAIN FDR	TEST FDR	OOT FDR
1	30	500	6	20	(1, 91)	1	0.9196	0.5698
2	30	500	6	20	"balanced_subsample"	1	0.8967	0.5653
3	30	50	5	8	(1, 91)	0.9637	0.8056	0.4478
4	30	50	5	8	"balanced_subsample"	0.9689	0.8158	0.3575
5	30	50	5	8	None	0.8630	0.8320	0.5787
6	30	100	5	8	None	0.8660	0.8150	0.5788
7	30	150	5	8	None	0.8647	0.8289	0.5770
8	30	200	5	8	None	0.8635	0.8333	0.5815
9	30	200	5	8	(1, 91)	0.9648	0.8024	0.4857
10	30	200	5	9	None	0.8865	0.8291	0.5837
11	30	200	5	10	None	0.9136	0.8535	0.5800

Figure 27. Random Forest Results

Results

Model	Parameter					Average FDR(%) at 3%		
Logistic Regression	# of Variables	CV	Class weights	Cs	Regularization	TRAIN	TEST	OOT
1	30	10	(1, 91)	0.001	L2	0.7146	0.7058	0.4480
2	30	10	(1, 91)	1	L2	0.6500	0.6481	0.3259
3	30	10	(1, 91)	10	L1	0.6925	0.7120	0.4190
4	30	10	(1, 91)	10	L2	0.7016	0.7008	0.3923
5	30	10	(1, 91)	100	L2	0.7001	0.7032	0.4245
Boosted Trees	# of Variables	# of Trees	Max Depth	Learning Rate		TRAIN	TEST	OOT
1	30	600	3	0.01		0.8839	0.8226	0.5704
2	30	600	4	0.01		0.9377	0.8663	0.5274
3	30	600	5	0.01		0.9753	0.8843	0.4704
4	30	800	3	0.01		0.9181	0.8494	0.5536
5	30	800	4	0.01		0.9562	0.8954	0.5078
6	30	800	5	0.01		0.9928	0.8931	0.4240
7	30	1000	3	0.01		0.9343	0.8736	0.5430
8	30	1000	4	0.01		0.9789	0.8931	0.5151
9	30	1000	5	0.01		1.0000	0.9103	0.4436
10	30	600	3	0.001		0.7008	0.6743	0.4983
11	30	600	4	0.001		0.7700	0.7176	0.4603
12	30	600	5	0.001		0.8400	0.7234	0.4145
Neural Network	# of Layers	# of Nodes	# of Epochs	Class weights	Notes	TRAIN	TEST	OOT
1	1	30	20	(0.5,8)	adam, binary cross	0.7564	0.7862	0.5642
2	1	150	20	(0.5,8)	adam, BCE	0.7130	0.7231	0.5140
3	1	30	100	(0.5,8)	adam, BCE	0.7304	0.7246	0.4749
4	2	(20,10)	20	(0.5,8)	adam, BCE	0.6854	0.6831	0.3911
5	1	30	20	(0.5,8)	adam, BCE, dropout, l2	0.7550	0.7692	0.5028
6	1	30	20	(0.5,8)	SGD, BCE	0.6008	0.6154	0.2961
7	1	30	20	(0.5,8)	adam, learning rate=0.01, BCE	0.7954	0.7231	0.5866
Random Forest	# of Variables	# of Trees	# of Features	Depth	class weight	TRAIN	TEST	OOT
1	30	500	6	20	(1, 91)	1.0000	0.9196	0.5698
2	30	500	6	20	"balanced_subsample"	1.0000	0.8967	0.5653
3	30	50	5	8	(1, 91)	0.9637	0.8056	0.4478
4	30	50	5	8	"balanced_subsample"	0.9689	0.8158	0.3575
5	30	50	5	8	None	0.8630	0.8320	0.5787
6	30	100	5	8	None	0.8660	0.8150	0.5788
7	30	150	5	8	None	0.8647	0.8289	0.5770
8	30	200	5	8	None	0.8635	0.8333	0.5815
9	30	200	5	8	(1, 91)	0.9648	0.8024	0.4857
10	30	200	5	9	None	0.8865	0.8291	0.5837
11	30	200	5	10	None	0.9136	0.8535	0.5800

Figure 28. Results of All Models.

Final Model – Random Forest

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

Hyperparameter Selection

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

GridSearchCV Trial 1

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

Best parameters: Max_features = 5; N_estimators = 500; Max_depth = 20

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

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

GridSearchCV Trial 2

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

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

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

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

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

Figure 29. Random Forest Results for Various Hyperparameters

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

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

Figure 30. Random Forest Results of 10 Runs

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

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

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

Figure 31. Random Forest – Best Model Training Set Results.

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

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

Figure 32. Random Forest – Best Model Testing Set Results.

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

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

Figure 33. Random Forest – Best Model OOT Validation Set Results.

Plots

Cutoff plot

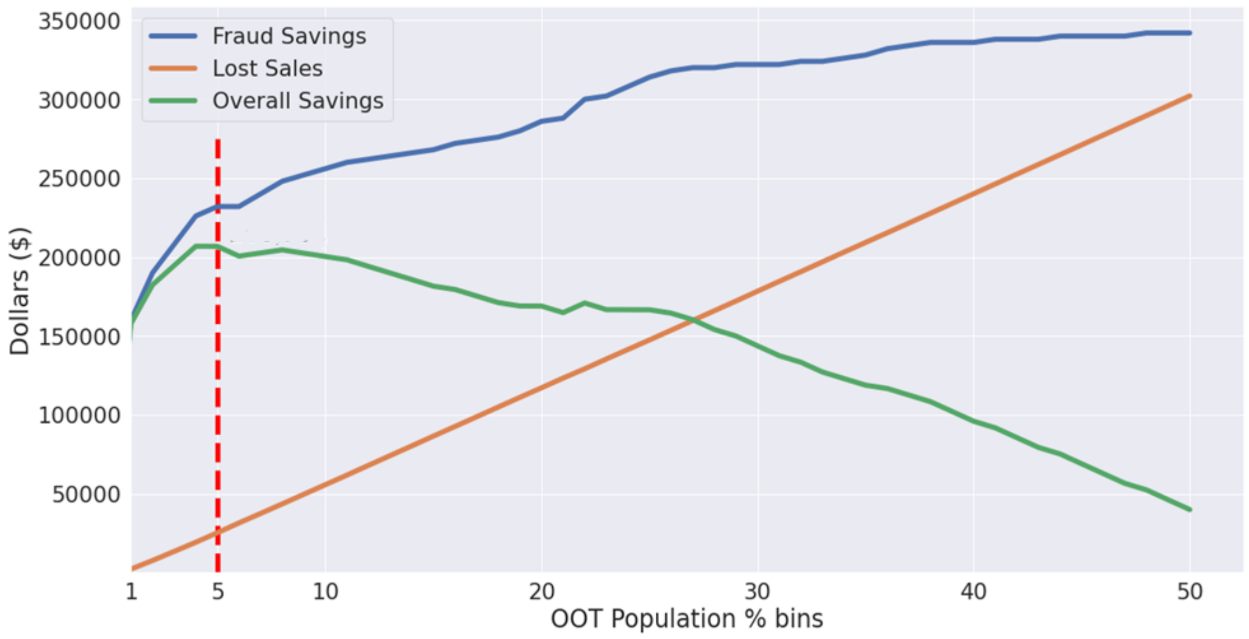


Figure 34. FDR Cutoff.

To put our model into business practice, we generated the above plot to provide a recommendation for the FDR cutoff point, i.e. any transaction with a score above the threshold at that cutoff point would be classified as fraudulent. Assuming $2,000 gain for every fraud that is detected (true positive, TP) and $50 loss for every non-fraud that is flagged as a fraud (false positive, FP), we plotted the Fraud Savings (blue), the Lost Sales (orange) and the Fraud Savings (green) for the out of time dataset. According to our analysis, we recommend a cutoff point at 5% as this threshold maximizes the profits (P) calculated as follows:

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

Time plots

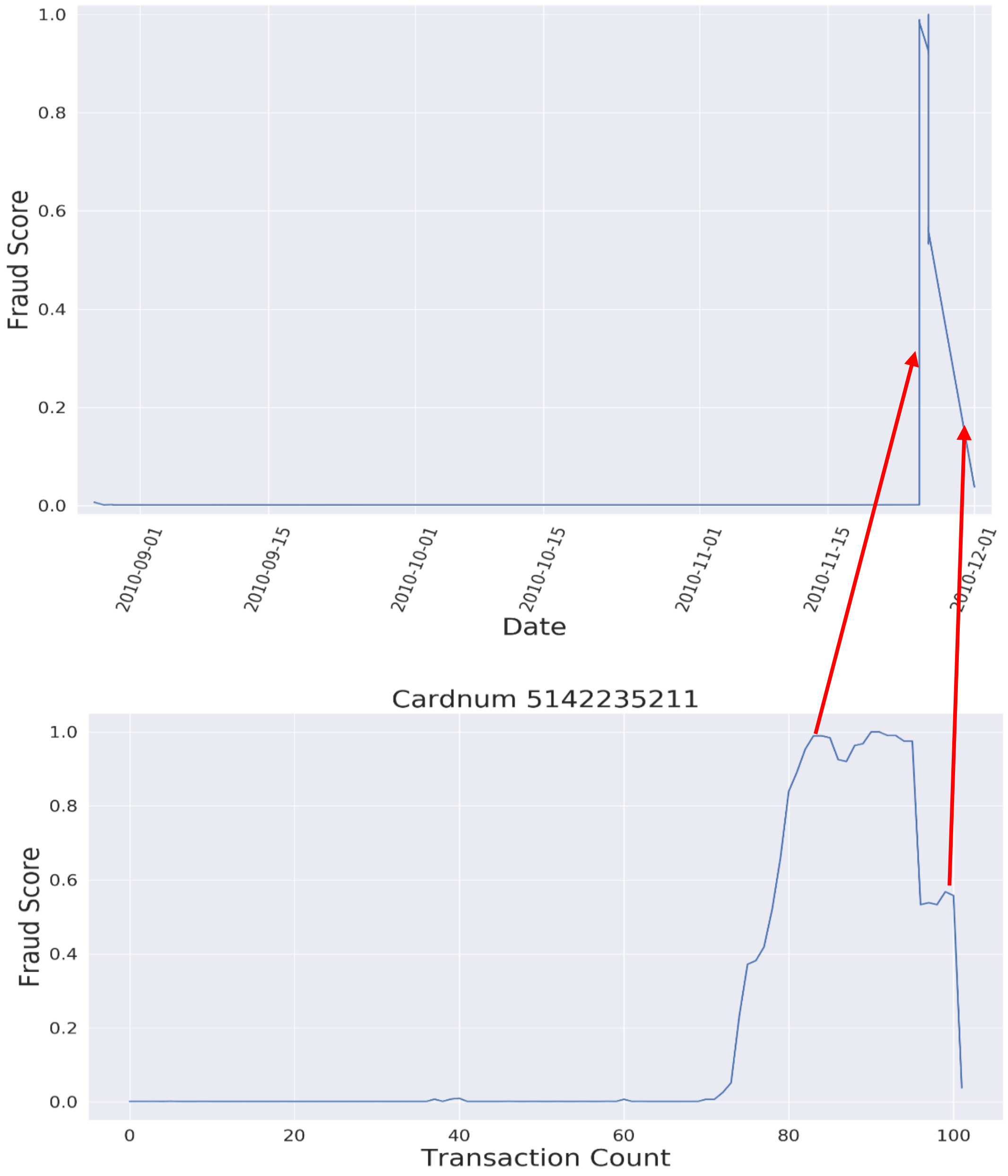


Figure 35. Transaction counts vs. Fraud score (Cardnum).

From our analysis, we found there was a correlation between transaction counts and fraud scores. When there is a burst of transaction activity, the fraud score rises rapidly. We illustrate this with the “Cardnum” value of “5142235211”. There was a burst of activity on 11/25/2010 containing 18 transactions and an additional 15 transactions on 11/26/2010. From the graph above, we can see the fraud score rose steeply in accordance with the time period.

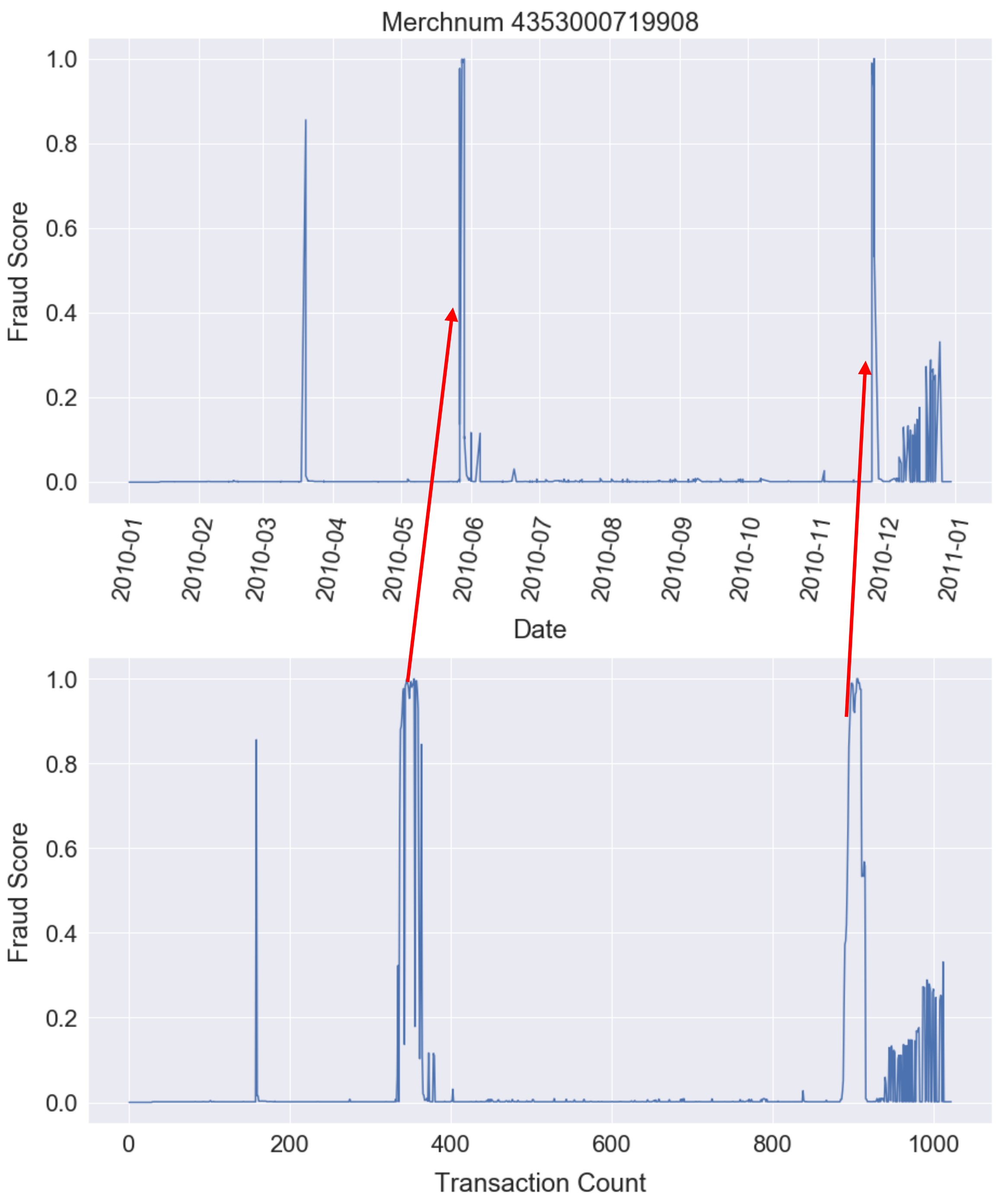


Figure 36. Transaction counts vs. Fraud Score (Merchnum).

Here is another example showing potentially fraudulent activity from the "Merchnum" data field containing the value of "4353000719908". There was a burst of activity with 34 transactions from 05/27/2010 to 05/29/2010 and another burst of activity with 32 transactions from 11/25/2010 to 11/26/2010. Similar to the "Cardnum" example, we can see in the graph above that the fraud score rose steeply in accordance with the time period.

Conclusions

A comprehensive analysis of credit card transaction fraud cases was performed. First, the data was cleaned and all missing values were filled logically. Then, over 300 candidate variables were created and feature selection was performed (filter and wrapper methods) to pick the best variables. The variables were then used in several models: logistic regression, boosted trees, random forest, and neural networks. Our best model to predict fraud was random forest which resulted in an 83.33% FDR at 3% for the testing dataset and a 59.21% FDR at 3% for the OOT dataset.

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

Appendix A: Data Quality Report (DQR)

Data Overview

Description: The data analysis contained in this report is primarily from an unidentified government organization in Tennessee. The dataset is of real government credit card transactions that was made public by the government organization for accounting purposes. However, the original dataset was not labeled and so Professor Stephen Coggeshall meticulously analyzed the data and created a binary label for each record in such a manner where the dataset is representative of realistic credit card transaction labels. Additionally, the records labeled "1" were carefully crafted by Professor Coggeshall based on his extensive subject matter expertise in fraudulent signals and activity. With each record containing a binary label, the dataset lends itself well to binary classification analysis.

Data Source: A government organization in Tennessee and Professor Stephen Coggeshall.

Data Time Period: 01/01/2010 to 12/31/2010.

Number of Data Fields: 10

Number of Records: 96,753

Number of Records Labeled "0": 95,694

Number of Records Labeled "1": 1,059

Name of Data File: "card transactions.xlsx"

Size of Data File: 7 MB

Summary Tables

The tables below provide summary information for the numerical and categorical data found in the dataset. Within the 10 data fields there are nine categorical data fields and one numerical data field. Section 2.1 contains the summary table for the nine categorical data fields and Section 2.2 contains the summary table for the one numerical data field.

Categorical Summary Table

Data Field	Num Records w/ a Value	Percent Populated	Num Unique Values	Most Common Value
Recnum	96,753	100%	96,753	N/A
Cardnum	96,753	100%	1,645	5142148452
Date	96,753	100%	365	2010-02-28
Merchnum	93,378	96.5%	13,092	930090121224
Merch description	96,753	100%	13,126	GSA-FSS-ADV
Merch state	95,558	98.8%	228	TN
Merch zip	92,097	95.2%	4,568	38118
Transtype	96,753	100%	4	P
Fraud	96,753	100%	2	0

Numerical Summary Table

Data Field	Num Records w/ a Value	Percent Populated	Num Unique Values	Num Records w/ a Value of Zero	Mean	Std Dev	Min	Max
Amount	96,753	100%	34,909	0	$427.89	$10,006.14	$0.01	$3,102,046

Data Field Exploration

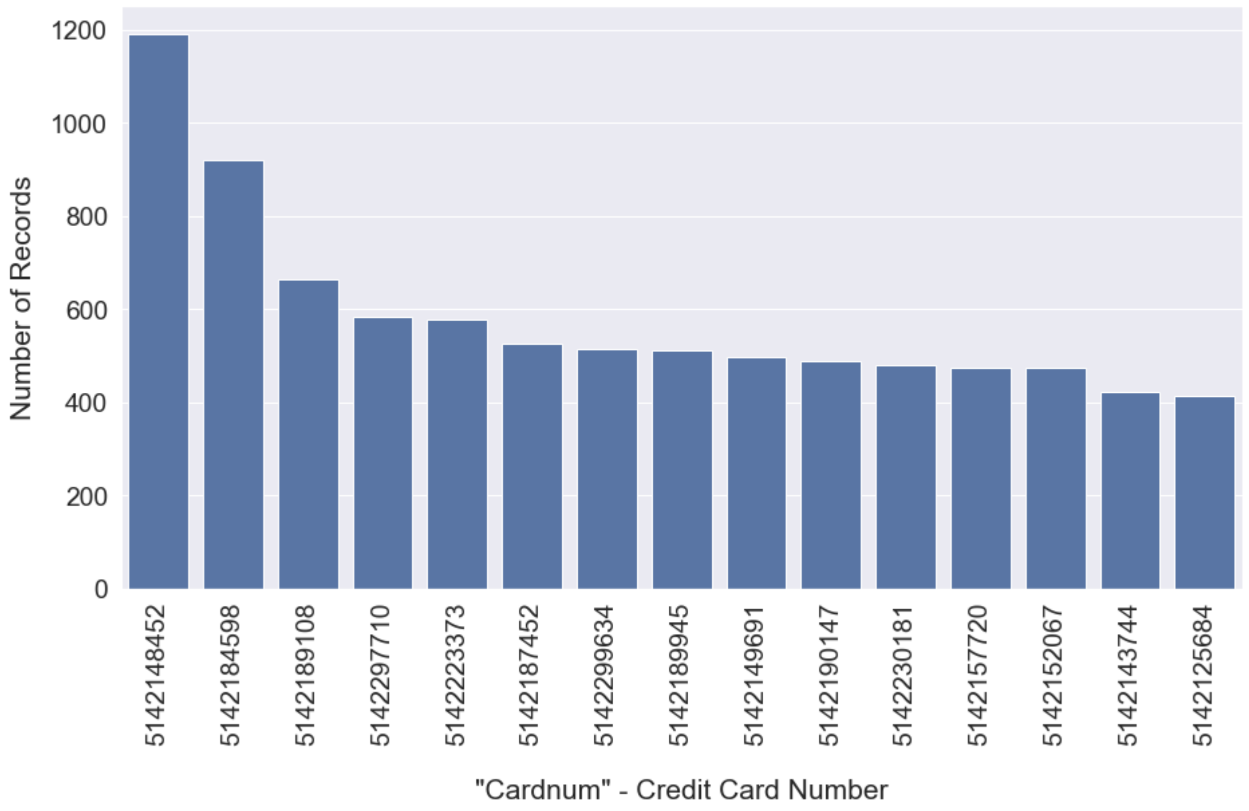
The subsections below provide additional detailed information about each data field within the dataset. The data fields are described in the order in which they appear.

Field 1: Recnum

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

Field 2: Cardnum

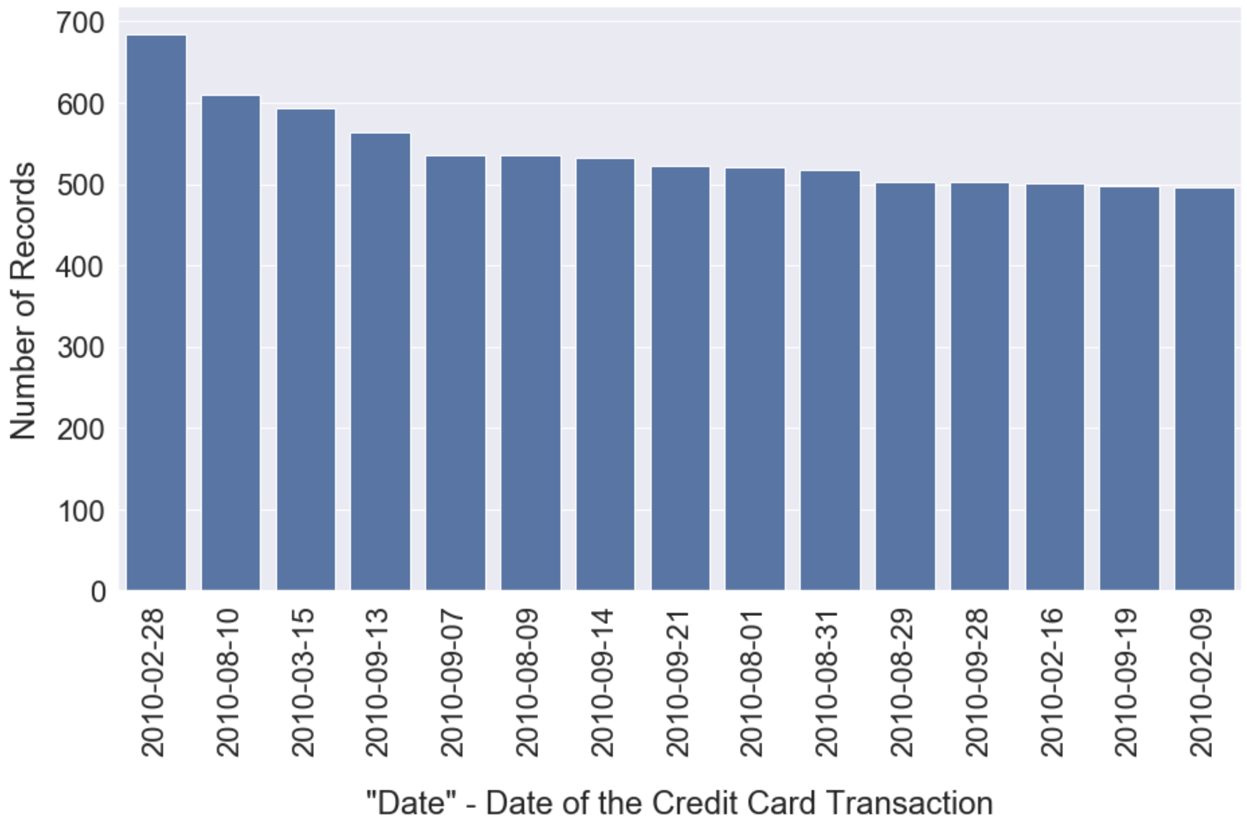
Description: A 10-digit categorical data field containing a shortened version of the credit card number for the card transaction record. The value is missing six internal digits from the true credit card number. The data field is 100% populated with 1,645 unique values. The bar chart below shows the top 15 values for the "Cardnum" data field.



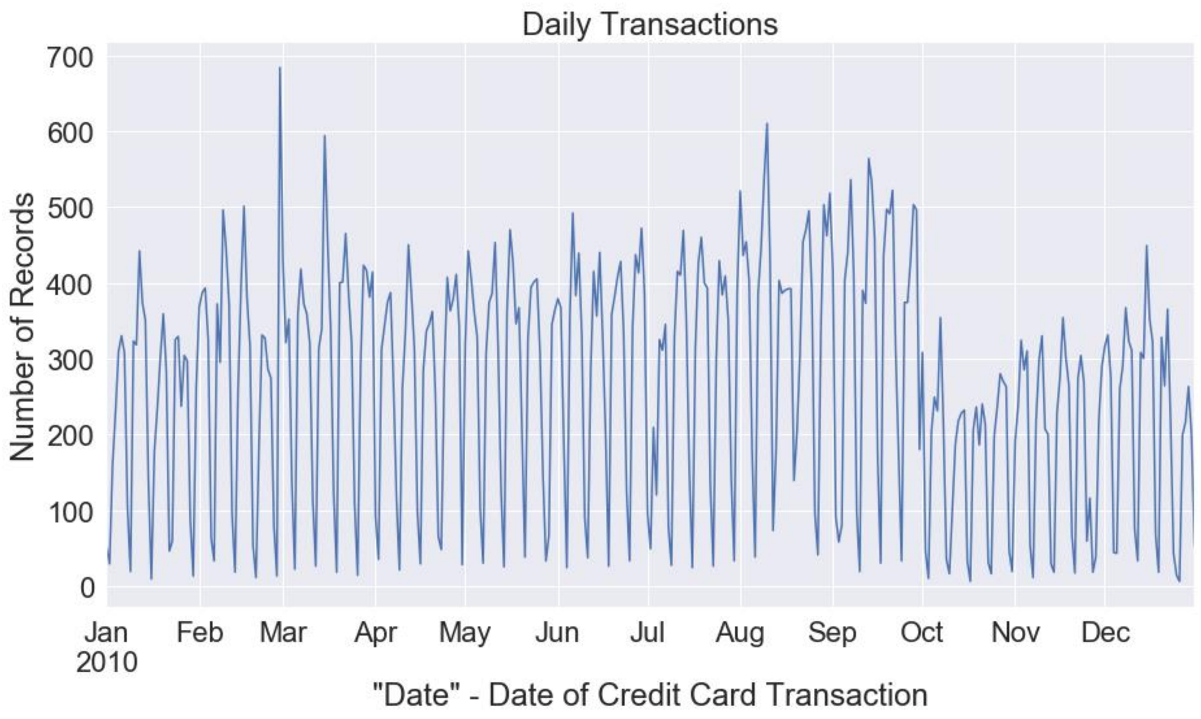
Field 3: Date

Description: A data field containing the date of the card transaction with a raw data format of MM/DD/YYYY. Upon importing the dataset, this data field has a converted format of YYYY-MM-DD. This data field is 100% populated and there are 365 unique values representing the 2010 calendar year. It is important to note that the dataset contains both weekly and annual seasonality. For the weekly seasonality, the data shows natural seasonality with regards to weekday and weekend activity. For the annual seasonality, the data shows activity related to government fiscal years. Government fiscal years start in October of the previous calendar year and end in September of the following calendar year. Thus, the 2010 government fiscal year started on 10/01/2009 and ended on30/09/2010. The 2011 government fiscal year started on 10/01/2010 and ended on 30/09/2011. The dataset therefore contains records for both the 2010 and 2011 government fiscal years.

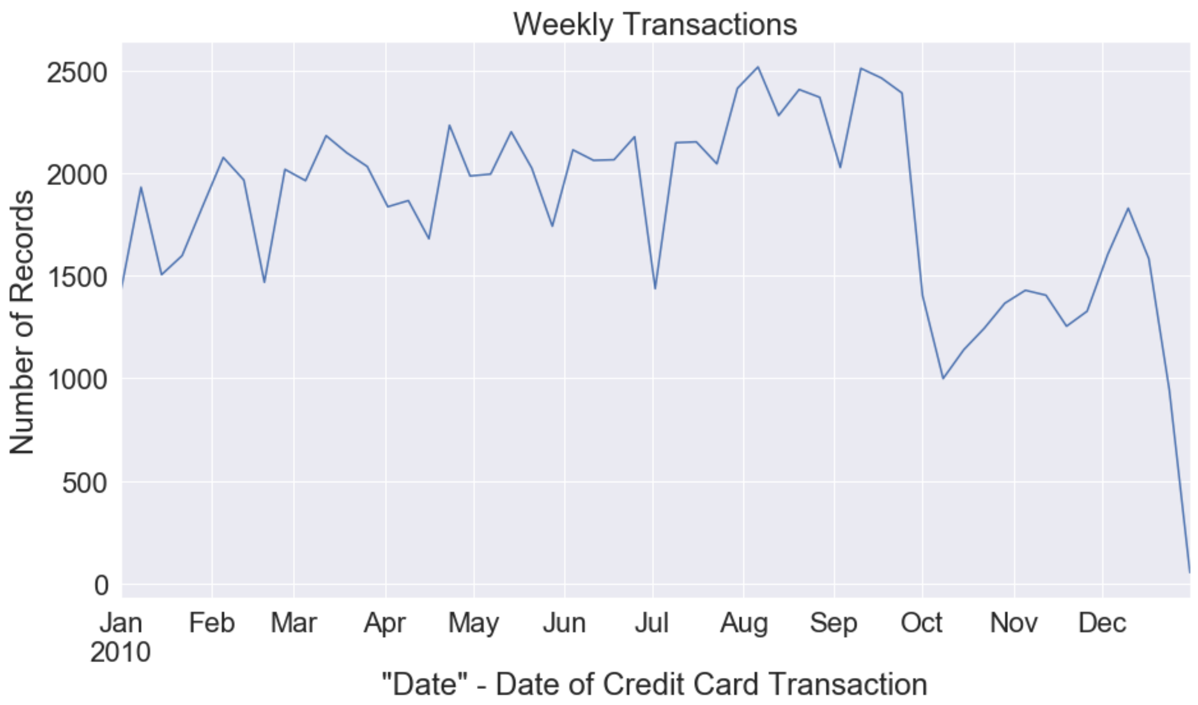
The bar chart below shows the top 15 values for the "Date" data field. Given that the 2010 government fiscal year ended on 30/09/2010, we can see that a majority of the top 15 values are towards the end of the 2010 fiscal year in the months of August and September.



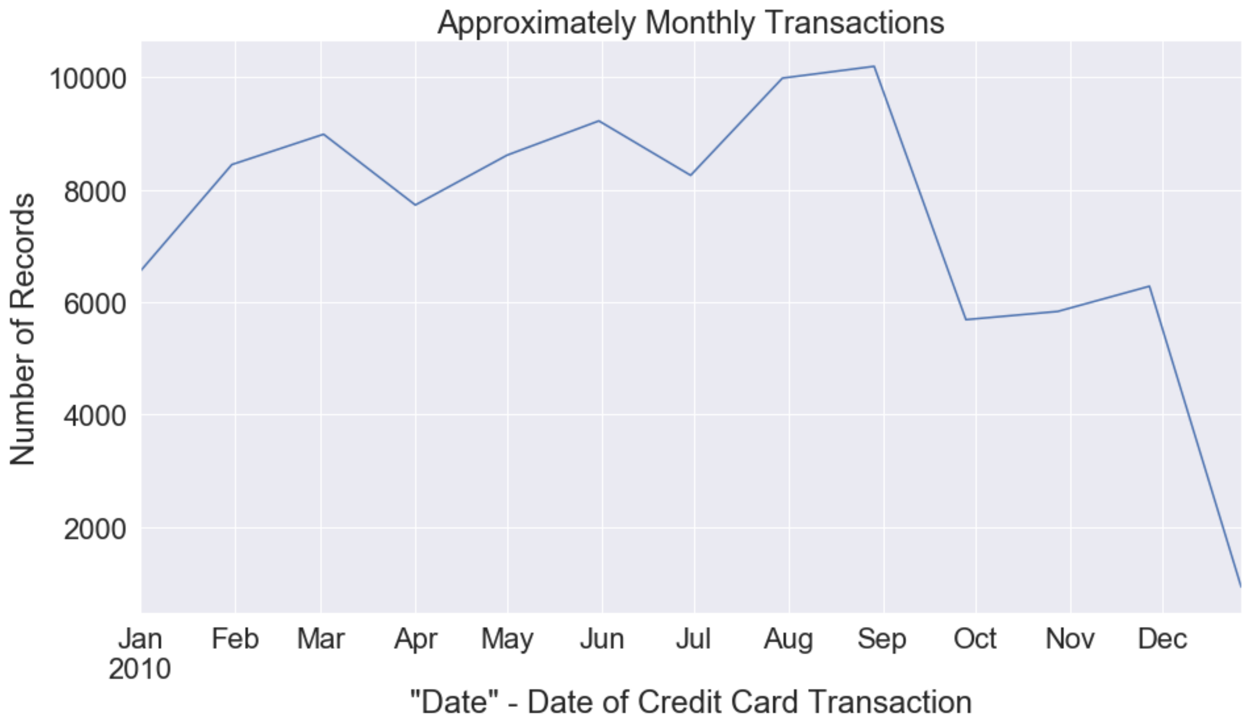
To view the weekly seasonality trends, the Daily Transactions graph below shows 52 peaks and troughs to represent the 52 weeks in a calendar year with the weekend activity depicted as the troughs.



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

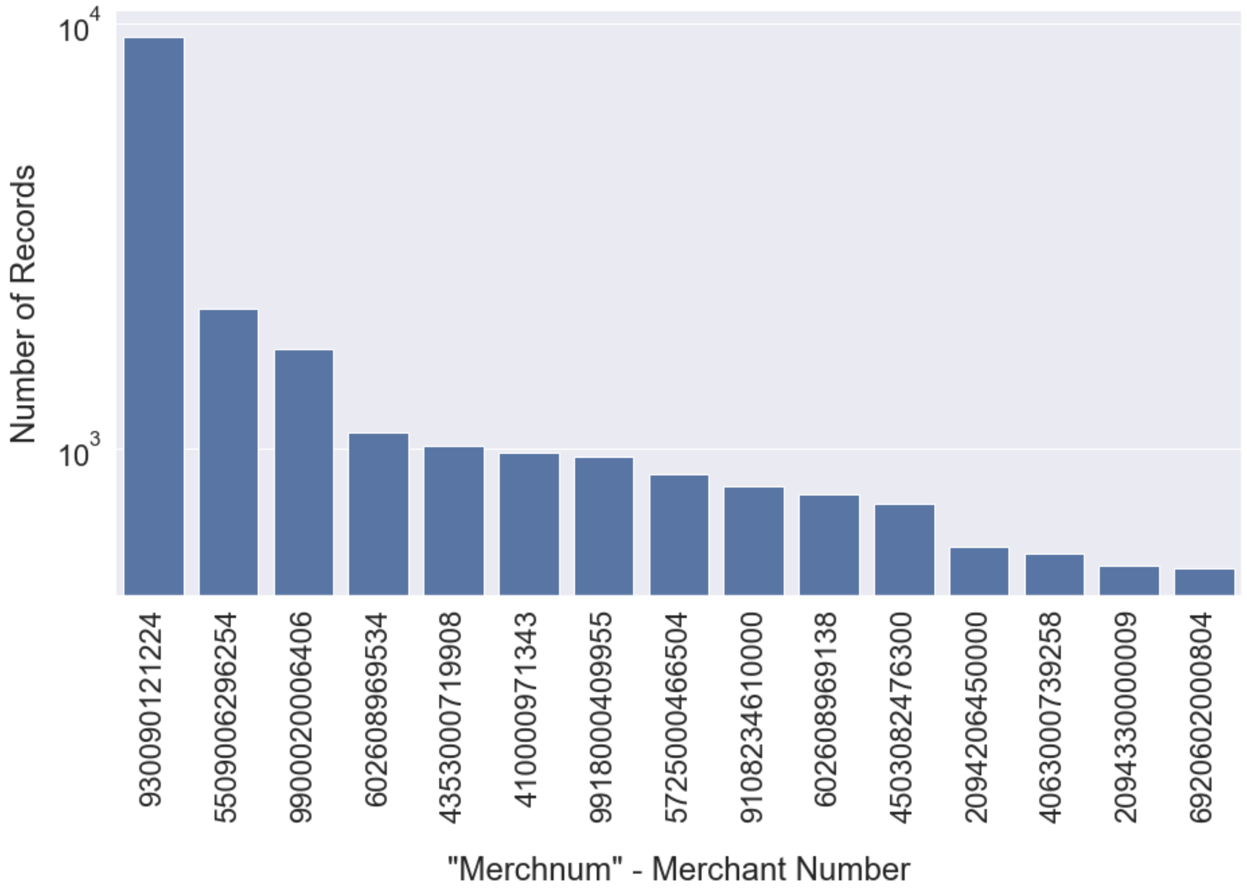


Additionally, the Monthly Transactions graph below provides a slightly different view of the annual seasonality in the data as well as the quarterly seasonality. With regards to the quarterly seasonality, there is increased activity towards the end of each quarter (March, June, September), and then a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



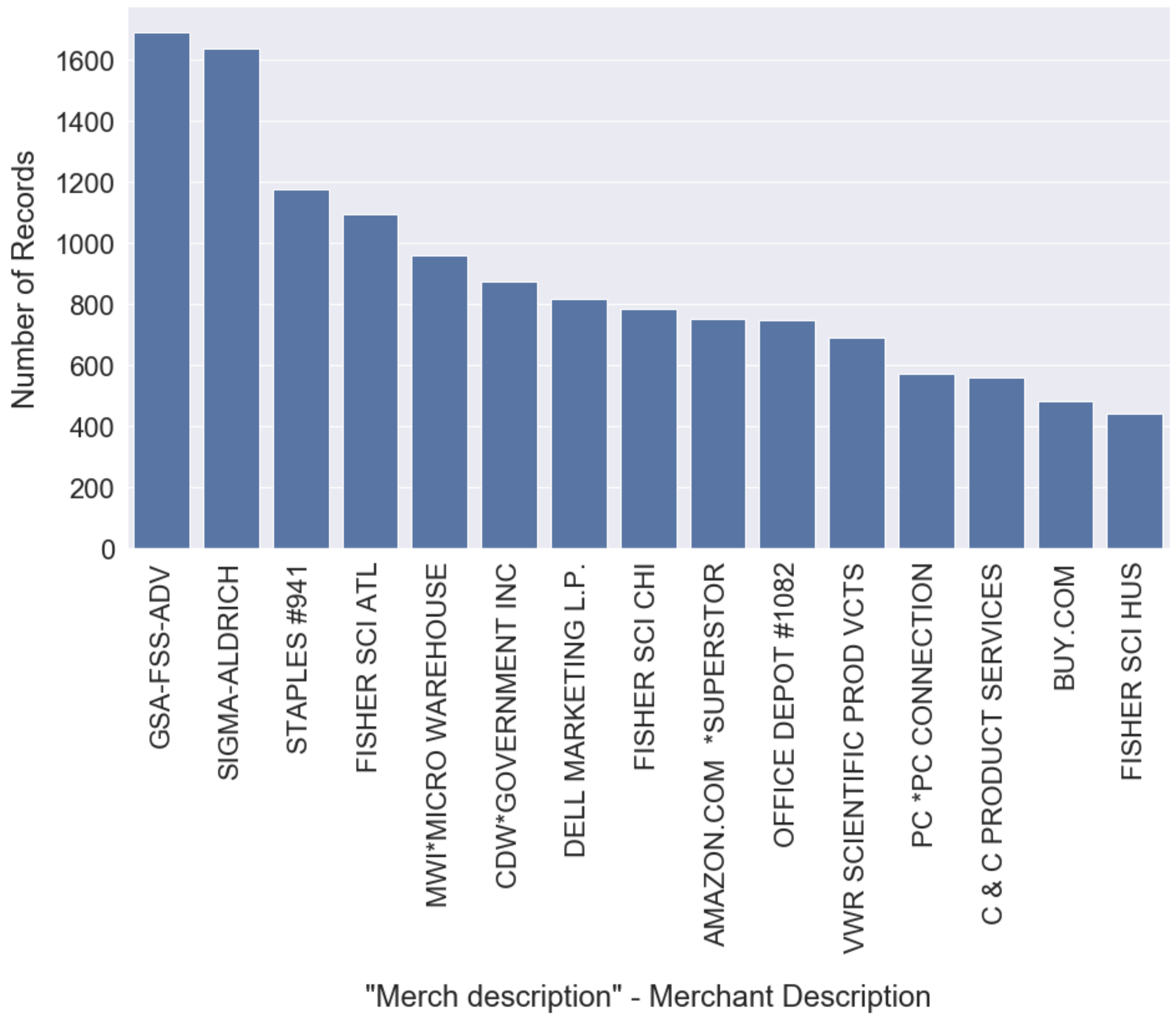
Field 4: Merchnum

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



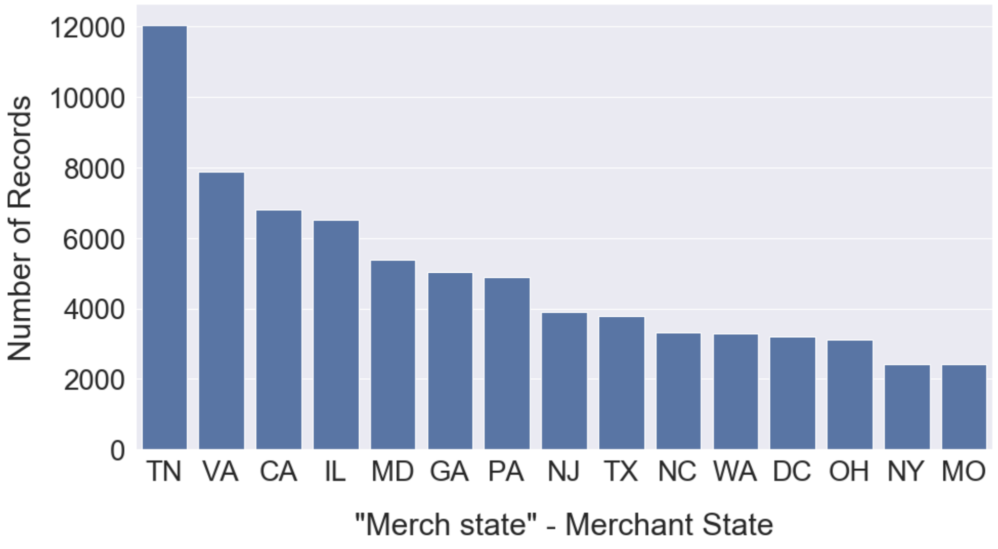
Field 5: Merch description

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



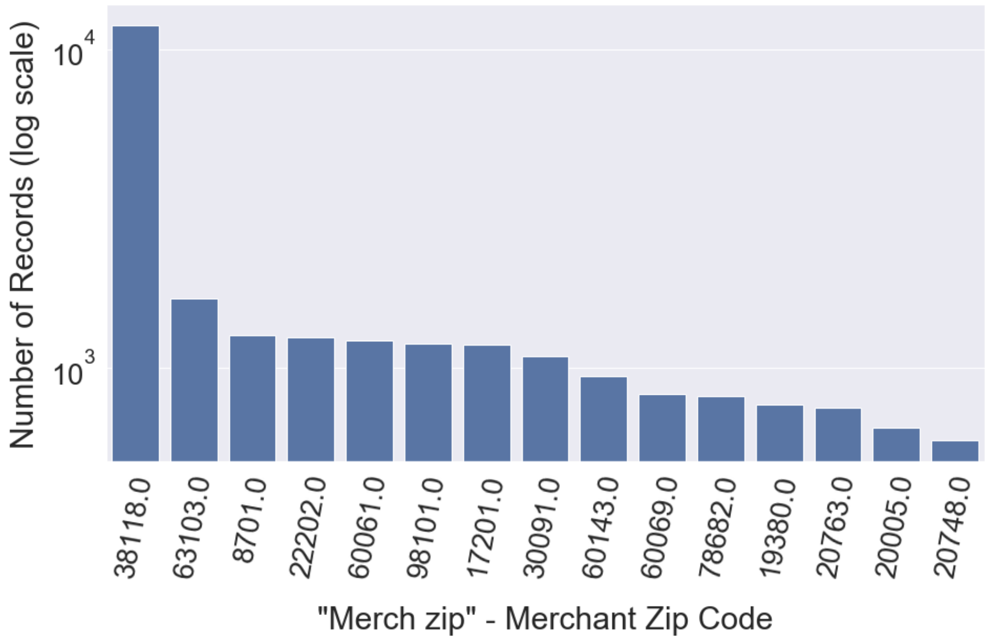
Field 6: Merch state

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



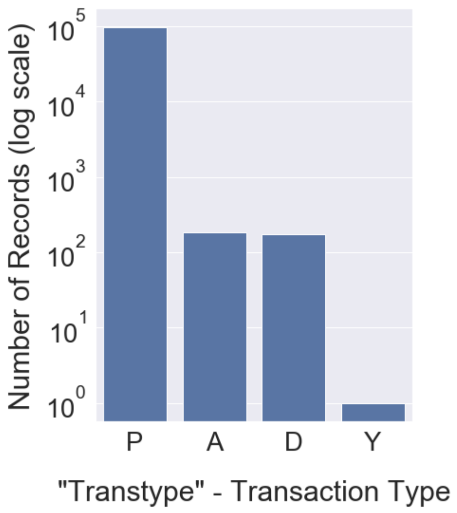
Field 7: Merch zip

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



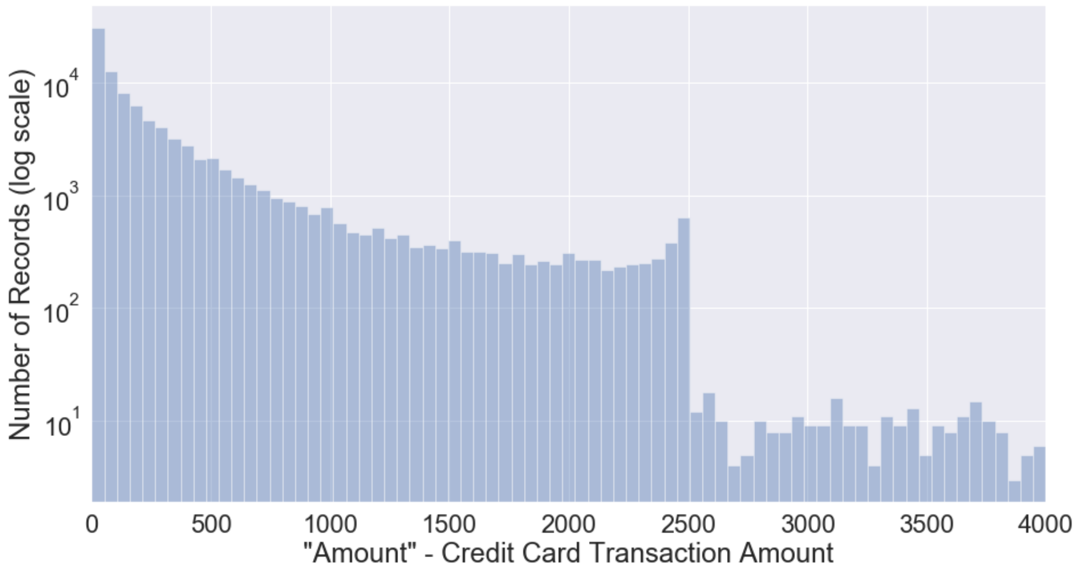
Field 8: Transtype

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

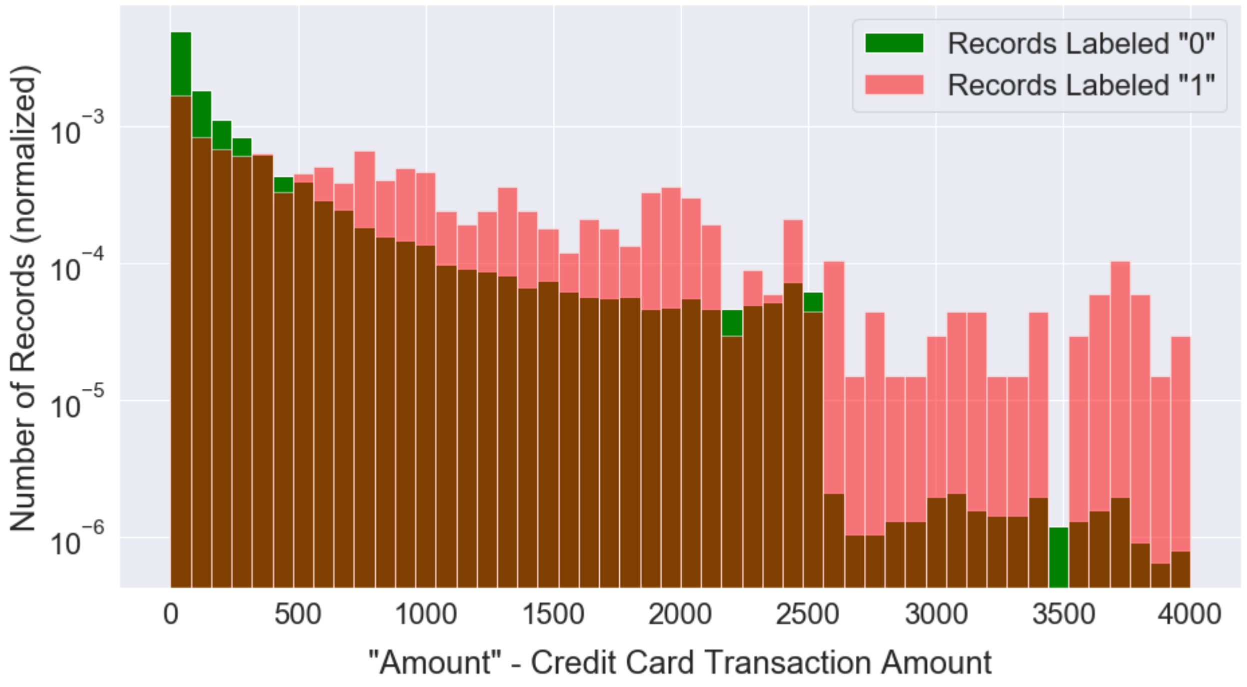


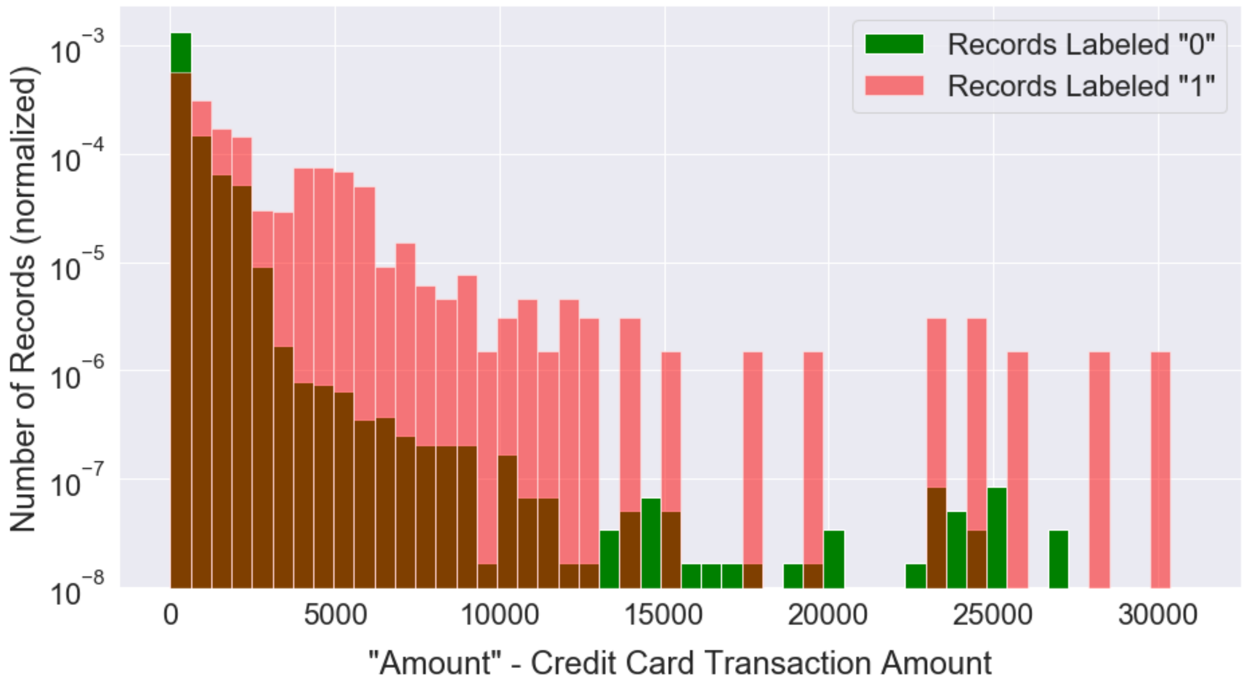
Field 9: Amount

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



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





Field 10: Fraud

Description: A binary data field used to label the card transaction record as either a zero or one. There are 95,694 records labeled as “0” and 1,059 records labeled as “1” in the dataset. This data field is 100% populated.

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

Appendix B: Candidate Variables

	Amount Candidate Variables		Amount Candidate Variables
1	Cardnum_totalamount0_Amount2	121	Cardnum_totalamount_0_actual
2	Cardnum_totalamount1_Amount2	122	Cardnum_totalamount_1_actual
3	Cardnum_totalamount3_Amount2	123	Cardnum_totalamount_3_actual
4	Cardnum_totalamount7_Amount2	124	Cardnum_totalamount_7_actual
5	Cardnum_totalamount14_Amount2	125	Cardnum_totalamount_14_actual
6	Cardnum_totalamount30_Amount2	126	Cardnum_totalamount_30_actual
7	Merchnum_totalamount0_Amount2	127	Merchnum_totalamount_0_actual
8	Merchnum_totalamount1_Amount2	128	Merchnum_totalamount_1_actual
9	Merchnum_totalamount3_Amount2	129	Merchnum_totalamount_3_actual
10	Merchnum_totalamount7_Amount2	130	Merchnum_totalamount_7_actual
11	Merchnum_totalamount14_Amount2	131	Merchnum_totalamount_14_actual
12	Merchnum_totalamount30_Amount2	132	Merchnum_totalamount_30_actual
13	Cardnum_|_zip_totalamount0_Amount2	133	Cardnum_|_zip_totalamount_0_actual
14	Cardnum_|_zip_totalamount1_Amount2	134	Cardnum_|_zip_totalamount_1_actual
15	Cardnum_|_zip_totalamount3_Amount2	135	Cardnum_|_zip_totalamount_3_actual
16	Cardnum_|_zip_totalamount7_Amount2	136	Cardnum_|_zip_totalamount_7_actual
17	Cardnum_|_zip_totalamount14_Amount2	137	Cardnum_|_zip_totalamount_14_actual
18	Cardnum_|_zip_totalamount30_Amount2	138	Cardnum_|_zip_totalamount_30_actual
19	Cardnum_|_state_totalamount0_Amount2	139	Cardnum_|_state_totalamount_0_actual
20	Cardnum_|_state_totalamount1_Amount2	140	Cardnum_|_state_totalamount_1_actual
21	Cardnum_|_state_totalamount3_Amount2	141	Cardnum_|_state_totalamount_3_actual
22	Cardnum_|_state_totalamount7_Amount2	142	Cardnum_|_state_totalamount_7_actual
23	Cardnum_|_state_totalamount14_Amount2	143	Cardnum_|_state_totalamount_14_actual
24	Cardnum_|_state_totalamount30_Amount2	144	Cardnum_|_state_totalamount_30_actual
25	Cardnum_|_merchnum_totalamount0_Amount2	145	Cardnum_|_merchnum_totalamount_0_actual
26	Cardnum_|_merchnum_totalamount1_Amount2	146	Cardnum_|_merchnum_totalamount_1_actual
27	Cardnum_|_merchnum_totalamount3_Amount2	147	Cardnum_|_merchnum_totalamount_3_actual
28	Cardnum_|_merchnum_totalamount7_Amount2	148	Cardnum_|_merchnum_totalamount_7_actual
29	Cardnum_|_merchnum_totalamount14_Amount2	149	Cardnum_|_merchnum_totalamount_14_actual
30	Cardnum_|_merchnum_totalamount30_Amount2	150	Cardnum_|_merchnum_totalamount_30_actual
31	Cardnum_median0_Amount2	151	Cardnum_median_0_actual
32	Cardnum_median1_Amount2	152	Cardnum_median_1_actual
33	Cardnum_median3_Amount2	153	Cardnum_median_3_actual
34	Cardnum_median7_Amount2	154	Cardnum_median_7_actual
35	Cardnum_median14_Amount2	155	Cardnum_median_14_actual
36	Cardnum_median30_Amount2	156	Cardnum_median_30_actual
37	Merchnum_median0_Amount2	157	Merchnum_median_0_actual
38	Merchnum_median1_Amount2	158	Merchnum_median_1_actual
39	Merchnum_median3_Amount2	159	Merchnum_median_3_actual
40	Merchnum_median7_Amount2	160	Merchnum_median_7_actual
41	Merchnum_median14_Amount2	161	Merchnum_median_14_actual

42	Merchnum_median30_Amount2	**162**	Merchnum_median_30_actual
43	Cardnum_|_zip_median0_Amount2	**163**	Cardnum_|_zip_median_0_actual
44	Cardnum_|_zip_median1_Amount2	**164**	Cardnum_|_zip_median_1_actual
45	Cardnum_|_zip_median3_Amount2	**165**	Cardnum_|_zip_median_3_actual
46	Cardnum_|_zip_median7_Amount2	**166**	Cardnum_|_zip_median_7_actual
47	Cardnum_|_zip_median14_Amount2	**167**	Cardnum_|_zip_median_14_actual
48	Cardnum_|_zip_median30_Amount2	**168**	Cardnum_|_zip_median_30_actual
49	Cardnum_|_state_median0_Amount2	**169**	Cardnum_|_state_median_0_actual
50	Cardnum_|_state_median1_Amount2	**170**	Cardnum_|_state_median_1_actual
51	Cardnum_|_state_median3_Amount2	**171**	Cardnum_|_state_median_3_actual
52	Cardnum_|_state_median7_Amount2	**172**	Cardnum_|_state_median_7_actual
53	Cardnum_|_state_median14_Amount2	**173**	Cardnum_|_state_median_14_actual
54	Cardnum_|_state_median30_Amount2	**174**	Cardnum_|_state_median_30_actual
55	Cardnum_|_merchnum_median0_Amount2	**175**	Cardnum_|_merchnum_median_0_actual
56	Cardnum_|_merchnum_median1_Amount2	**176**	Cardnum_|_merchnum_median_1_actual
57	Cardnum_|_merchnum_median3_Amount2	**177**	Cardnum_|_merchnum_median_3_actual
58	Cardnum_|_merchnum_median7_Amount2	**178**	Cardnum_|_merchnum_median_7_actual
59	Cardnum_|_merchnum_median14_Amount2	**179**	Cardnum_|_merchnum_median_14_actual
60	Cardnum_|_merchnum_median30_Amount2	**180**	Cardnum_|_merchnum_median_30_actual
61	Cardnum_mean0_Amount2	**181**	Cardnum_mean_0_actual
62	Cardnum_mean1_Amount2	**182**	Cardnum_mean_1_actual
63	Cardnum_mean3_Amount2	**183**	Cardnum_mean_3_actual
64	Cardnum_mean7_Amount2	**184**	Cardnum_mean_7_actual
65	Cardnum_mean14_Amount2	**185**	Cardnum_mean_14_actual
66	Cardnum_mean30_Amount2	**186**	Cardnum_mean_30_actual
67	Merchnum_mean0_Amount2	**187**	Merchnum_mean_0_actual
68	Merchnum_mean1_Amount2	**188**	Merchnum_mean_1_actual
69	Merchnum_mean3_Amount2	**189**	Merchnum_mean_3_actual
70	Merchnum_mean7_Amount2	**190**	Merchnum_mean_7_actual
71	Merchnum_mean14_Amount2	**191**	Merchnum_mean_14_actual
72	Merchnum_mean30_Amount2	**192**	Merchnum_mean_30_actual
73	Cardnum_|_zip_mean0_Amount2	**193**	Cardnum_|_zip_mean_0_actual
74	Cardnum_|_zip_mean1_Amount2	**194**	Cardnum_|_zip_mean_1_actual
75	Cardnum_|_zip_mean3_Amount2	**195**	Cardnum_|_zip_mean_3_actual
76	Cardnum_|_zip_mean7_Amount2	**196**	Cardnum_|_zip_mean_7_actual
77	Cardnum_|_zip_mean14_Amount2	**197**	Cardnum_|_zip_mean_14_actual
78	Cardnum_|_zip_mean30_Amount2	**198**	Cardnum_|_zip_mean_30_actual
79	Cardnum_|_state_mean0_Amount2	**199**	Cardnum_|_state_mean_0_actual
80	Cardnum_|_state_mean1_Amount2	**200**	Cardnum_|_state_mean_1_actual
81	Cardnum_|_state_mean3_Amount2	**201**	Cardnum_|_state_mean_3_actual
82	Cardnum_|_state_mean7_Amount2	**202**	Cardnum_|_state_mean_7_actual
83	Cardnum_|_state_mean14_Amount2	**203**	Cardnum_|_state_mean_14_actual
84	Cardnum_|_state_mean30_Amount2	**204**	Cardnum_|_state_mean_30_actual
85	Cardnum_|_merchnum_mean0_Amount2	**205**	Cardnum_|_merchnum_mean_0_actual

86	Cardnum_|_merchnum_mean1_Amount2	**206**	Cardnum_|_merchnum_mean_1_actual
87	Cardnum_|_merchnum_mean3_Amount2	**207**	Cardnum_|_merchnum_mean_3_actual
88	Cardnum_|_merchnum_mean7_Amount2	**208**	Cardnum_|_merchnum_mean_7_actual
89	Cardnum_|_merchnum_mean14_Amount2	**209**	Cardnum_|_merchnum_mean_14_actual
90	Cardnum_|_merchnum_mean30_Amount2	**210**	Cardnum_|_merchnum_mean_30_actual
91	Cardnum_max0_Amount2	**211**	Cardnum_max_0_actual
92	Cardnum_max1_Amount2	**212**	Cardnum_max_1_actual
93	Cardnum_max3_Amount2	**213**	Cardnum_max_3_actual
94	Cardnum_max7_Amount2	**214**	Cardnum_max_7_actual
95	Cardnum_max14_Amount2	**215**	Cardnum_max_14_actual
96	Cardnum_max30_Amount2	**216**	Cardnum_max_30_actual
97	Merchnum_max0_Amount2	**217**	Merchnum_max_0_actual
98	Merchnum_max1_Amount2	**218**	Merchnum_max_1_actual
99	Merchnum_max3_Amount2	**219**	Merchnum_max_3_actual
100	Merchnum_max7_Amount2	**220**	Merchnum_max_7_actual
101	Merchnum_max14_Amount2	**221**	Merchnum_max_14_actual
102	Merchnum_max30_Amount2	**222**	Merchnum_max_30_actual
103	Cardnum_|_zip_max0_Amount2	**223**	Cardnum_|_zip_max_0_actual
104	Cardnum_|_zip_max1_Amount2	**224**	Cardnum_|_zip_max_1_actual
105	Cardnum_|_zip_max3_Amount2	**225**	Cardnum_|_zip_max_3_actual
106	Cardnum_|_zip_max7_Amount2	**226**	Cardnum_|_zip_max_7_actual
107	Cardnum_|_zip_max14_Amount2	**227**	Cardnum_|_zip_max_14_actual
108	Cardnum_|_zip_max30_Amount2	**228**	Cardnum_|_zip_max_30_actual
109	Cardnum_|_state_max0_Amount2	**229**	Cardnum_|_state_max_0_actual
110	Cardnum_|_state_max1_Amount2	**230**	Cardnum_|_state_max_1_actual
111	Cardnum_|_state_max3_Amount2	**231**	Cardnum_|_state_max_3_actual
112	Cardnum_|_state_max7_Amount2	**232**	Cardnum_|_state_max_7_actual
113	Cardnum_|_state_max14_Amount2	**233**	Cardnum_|_state_max_14_actual
114	Cardnum_|_state_max30_Amount2	**234**	Cardnum_|_state_max_30_actual
115	Cardnum_|_merchnum_max0_Amount2	**235**	Cardnum_|_merchnum_max_0_actual
116	Cardnum_|_merchnum_max1_Amount2	**236**	Cardnum_|_merchnum_max_1_actual
117	Cardnum_|_merchnum_max3_Amount2	**237**	Cardnum_|_merchnum_max_3_actual
118	Cardnum_|_merchnum_max7_Amount2	**238**	Cardnum_|_merchnum_max_7_actual
119	Cardnum_|_merchnum_max14_Amount2	**239**	Cardnum_|_merchnum_max_14_actual
120	Cardnum_|_merchnum_max30_Amount2	**240**	Cardnum_|_merchnum_max_30_actual