Card Transaction Fraud Identification

Wenyi Li, Yue Li, Xinyi Ou, Yingying Qian, Lingyi Xu, Runfeng Zhang
DSO 562 Fraud Analytics
5/5/21

Professor Stephen Coggeshall
Project Advisor

Marshall Consulting

Table of Content

Executive Summary

Card transaction fraud is a type of fraud committed during a transaction or payment using an unauthorized card, such as a credit card or debit card. This is one type of low-risk, high-profit criminal activity. This type of fraud usually happens in the retail industry, including both in-store and online transactions. There are two common types of fraud, which are card-not-present fraud and card-present fraud. Card-not-present fraud often occurs online mainly, where card-present fraud regularly happens at ATMs.

This project will be focusing on identifying and predicting card payment fraud in the purchasing process by building a data-driven machine learning algorithm. The overall goal of the project is to find the most effective and efficient statistical analysis model to predict and identify card transaction fraud.

The report details the creation and completion process of a supervised machine learning algorithm that can perform real-time fraud detection. Our team has decided to accomplish our overall project goal by completing the following five objectives:

1. Data cleaning – detect, correct, and remove inaccurate/corrupted/incomplete/duplicate data records from the dataset
2. Feature creation – construct new and insightful features from existing data entries to train a machine learning model
3. Feature selection – select a subset of features by reducing the number of input variables
4. Modeling – establish machine learning models, train and test the models to discover the most efficient and effective model
5. Model analysis and conclusion – analyze models created in the previous steps and make recommendations for future payment fraud identification process

Our team has utilized many different algorithms to fit the data, including Logistic Regression, Decision Tree, Neural Network, Support Vector Machine, Light GBM, Random Forest, and K-Nearest Neighbor. We have also tried several combinations of parameters for each model and compared the performance of different models based on FDR at a 3% rejection rate.

Among all these models, a Neural Network Classifier was selected as our best and final model. The FDR scores achieved by this model on training, testing, and OOT data are 81.1%, 84.8%, and 58.1%. The key statistics of the top 20% bins also show that our final model has a good performance in detecting fraud. With our best model, we would like to recommend a cutoff point to maximize the overall savings. At this point, we want to deny as few fraud transactions as possible and avoid losses due to not denying a sufficient number of transactions. In our model, our recommended cutoff point is about 2.5%, which will bring about 176,55 dollars of overall savings.

We also discussed the potential future improvements for our model, such as building more expert features and performing more hyperparameters tuning.

1 Data Description

1.1 File Description

The "applications data.csv" is a computer-generated dataset creating the same statistical properties as accurate application data. It stores 1,000,000 records of U.S. applications such as opening a credit card or a cell phone account. It covers ten fields, including application date, Social Security Number (SSN), applicant's name, address, zip code, date of birth, home phone number, and a label representing whether the application is a fraud or not. The dataset came from an identity fraud prevention company and was shared by Professor Stephen Coggeshall in February 2021.

Table 1.1: File Description

Dataset Name	Card Transaction Dataset
Dataset Purpose	Combined actual credit card purchases with manipulated fraud labels to serve as the raw data for building machine learning models to find the frauds
Data Source	Came from a U.S. government organization
Time Period	From January 1, 2010 to December 31, 2010
# of Fields	10 fields in total – 7 categorical, 1 text, 1 date, 1 numerical
# of Record	96,753

1.2 Summary Statistics Table

All Fields can be treated as categorical except for two dates (date, dob). All ten fields are fully populated. Key statistics of these fields are summarized as follows.

Table 1.2.1: Summary Statistics of Categorical Fields

Column Name	# of Records	% Populated	# records with value zero	Unique Values	Most Common Field Value	Data Type
Recnum*	96,753	100	0	96,753	N/A**	int64
Cardnum*	96,753	100	0	1,645	5142148452	int64
Merchnum*	93,378	96.51	231	13,091	930090121224	object
Merch Descirption	96,753	100	0	13,126	GSA-FSS-ADV	text
Merch state	95,558	98.76	0	227	TN	object
Merch zip*	92,097	95.19	0	4,567	38118	float64
Transtype	96,753	100	0	4	P	object
Fraud*	96,753	100	95,694	2	0	int64

Table 1.2.2: Summary Statistics of Numerical Fields

Column Name	# of Records	% Populated	Unique Values	Most Common Field Value	Minimum Value	Maximum Value	Mean	Data Type
Amount	96,753	100	34,909	3.62	0.01	3102045.53	427.89	float 64

Table 1.2.3: Summary Statistics of Date Fields

Column Name	# of Records	% Populated	Unique Values	Most Common Field Value	Minimum Value	Maximum Value	Data Type
Date	96,753	100	365	2010/2/28	2010/1/1	2010/12/31	datetime64[ns]

* These fields should be categorical but are stored as integers/floats/object in the dataset.

** All values in the *recnum* field are unique, thus not such a most common field value.

1.3 Field Examples

1.3.1 Field "cardnum"

Table 1.3.1: cardnum

Description	Credit card number of the transaction
Type	Categorical
Distribution	100% populated with 1,645 unique values. "5142148452" occurred the most for 1192 times.

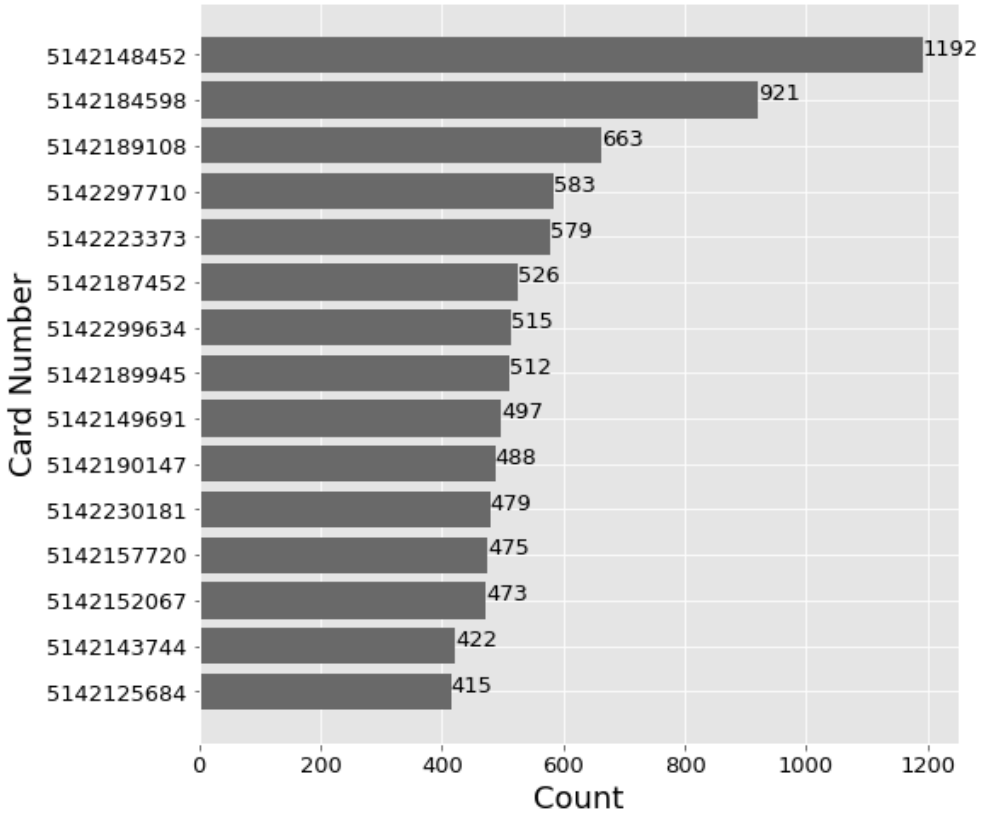

Figure 1.3.1: Frequency Distribution of the *cardnum* Field
(Top 15 Most Common Values)

1.3.2 Field “date”

Table 1.3.2: date

Description	The date that the payment was made
Type	Date
Distribution	100% populated with 365 unique values. The following plots compare the distributions of payment counts of fraud and non-fraud transactions each month, each day of the week, and each day.

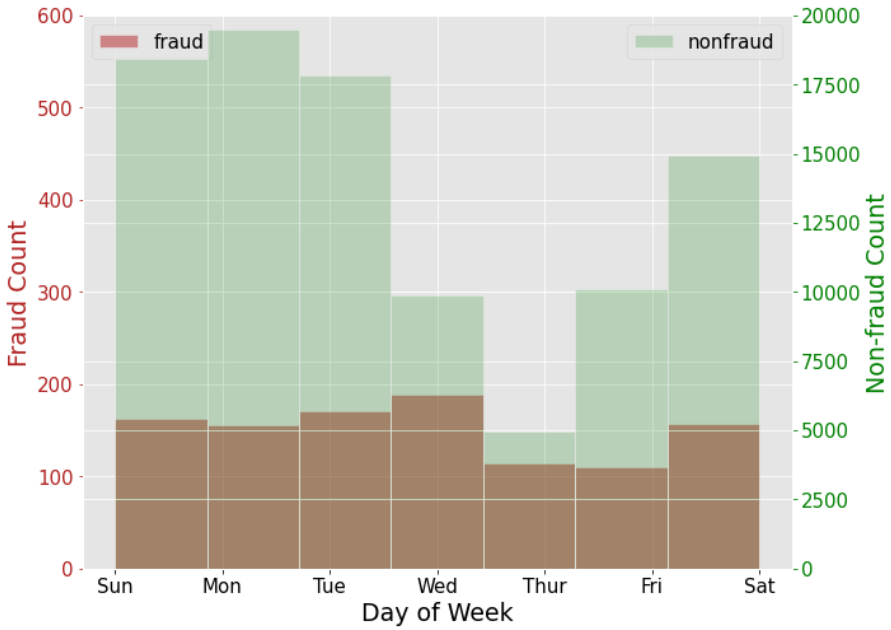

Figure 1.3.2.1: Payment Counts by Day of Week

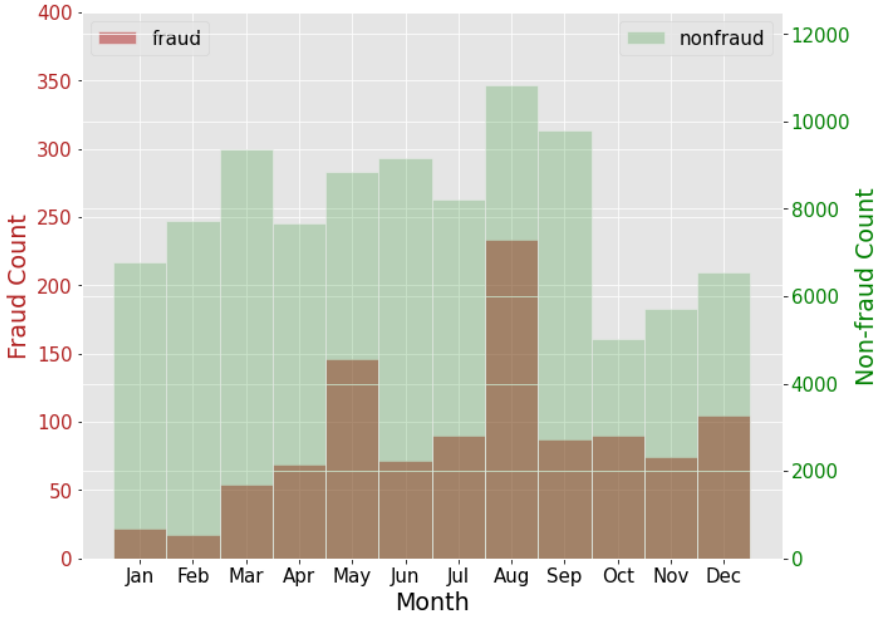

Figure 1.3.2.2: Payment Counts by Month

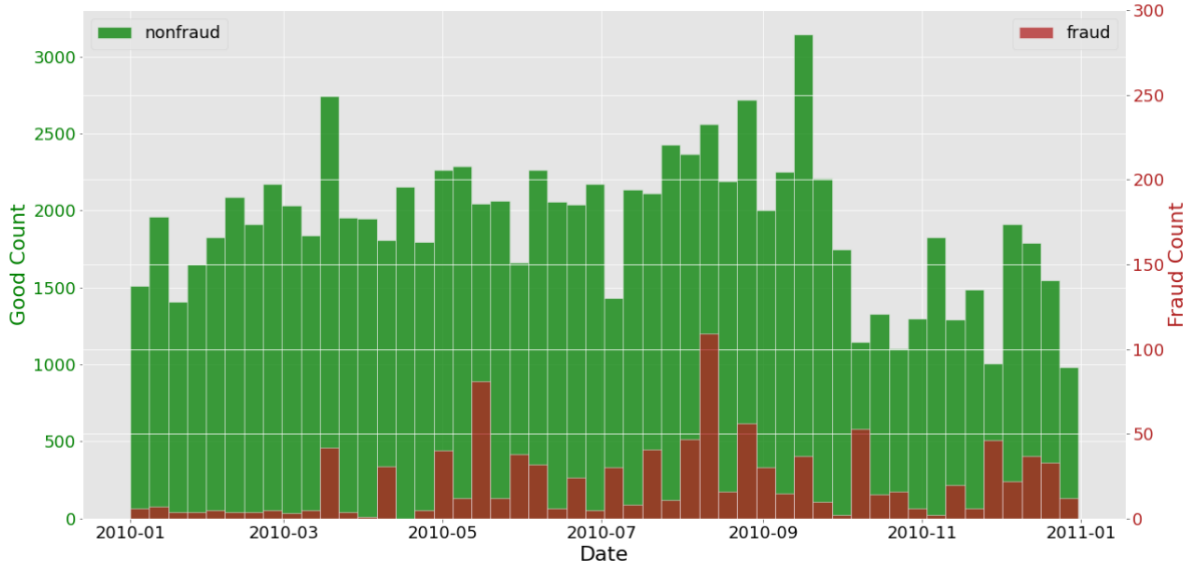

Figure 1.3.2.3: Payment Counts by Day

From the above graphs, we can see that the number of payments reached a peak on Monday (non-fraud) and Wednesday (fraud), respectively. Both fraud and non-fraud payments have the highest transactions in August.

1.3.3 Field "merch state"

Table 1.3.3: merch state

Description	**The abbreviations of the state where the purchase is made.**
Type	Categorical
Distribution	98.76% populated with 227 unique values, 51 of which stand for states in the U.S., 7 stands for states in Canada, 1 value U.S., 167 invalid numeric values, and other missing values. Transactions that take place in Canada do not have any fraud.

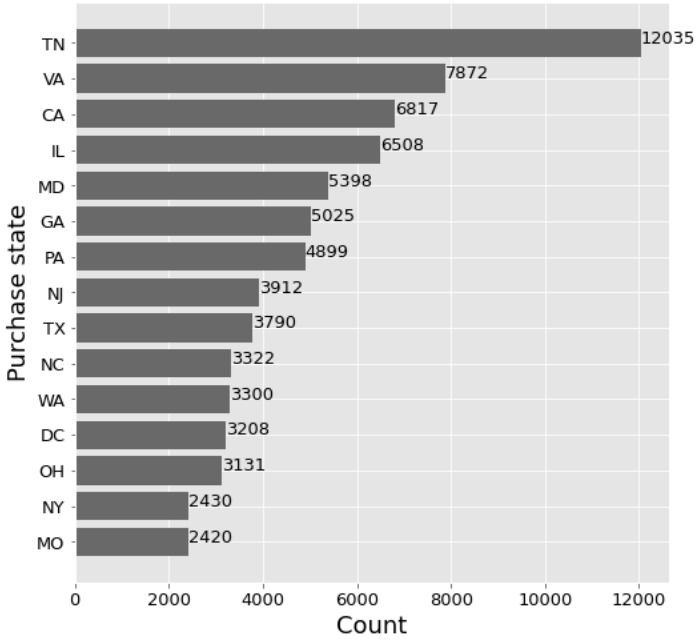

Figure 1.3.3.1: Frequency Distribution of the *merch state* Field
(Top 15 Most Common Values)

Figure 1.3.3.2: Distribution of Payment Counts by States

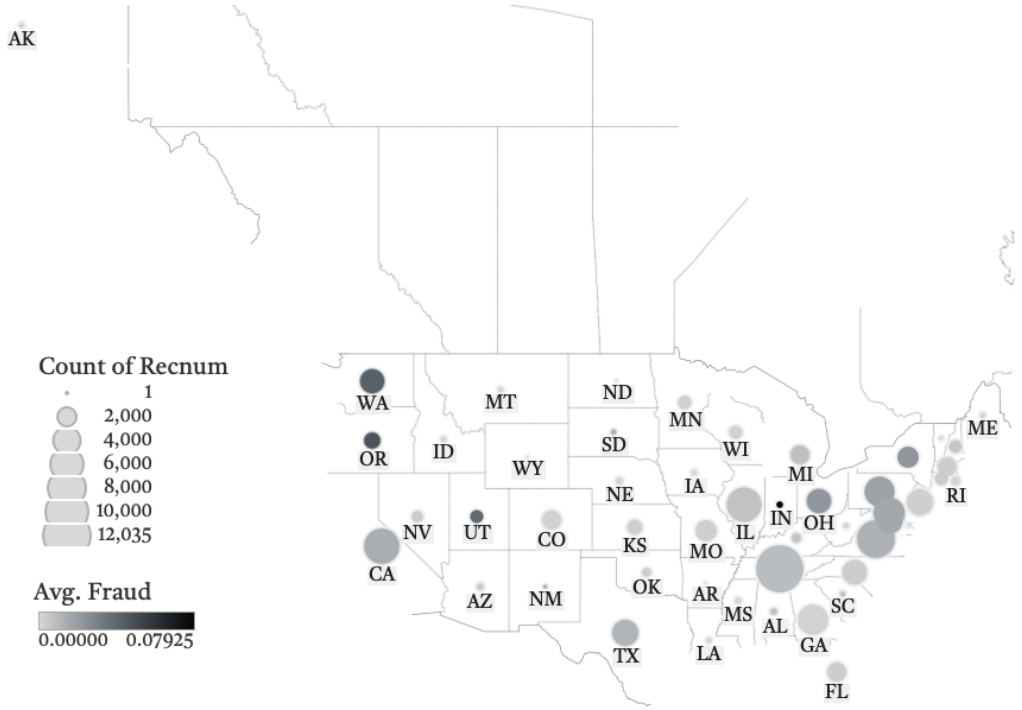

Figure 1.3.3.3: Distribution of Payment Counts by States

1.3.4 Field “amount”

Table 1.3.4: amount

Description	**The transaction amount of that record.**
Type	Numerical
Distribution	100% populated. The following table shows the distribution of the transaction amounts. Outlier (values that are not within 3 standard deviations of the mean) was removed for graphing purposes.

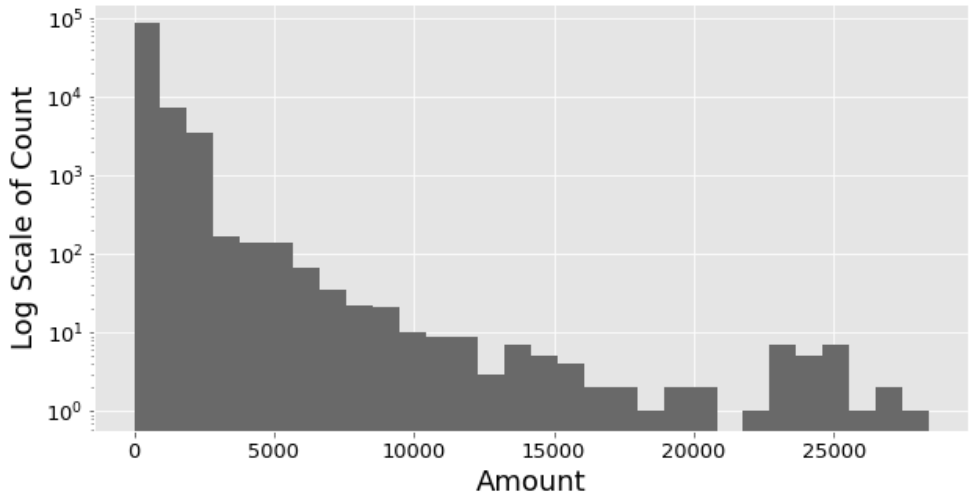

Figure 1.3.4: Frequency Distribution of the *amount* Field

1.3.5 Field “fraud”

Table 1.3.5: fraud

Description	**Labels represent whether the transaction is fraudulent (“1” for Yes, “0” for No).**
Type	Categorical
Most common field value	100% populated with 98.91% filled with value 0. The following plot shows the distribution of the fraud label.

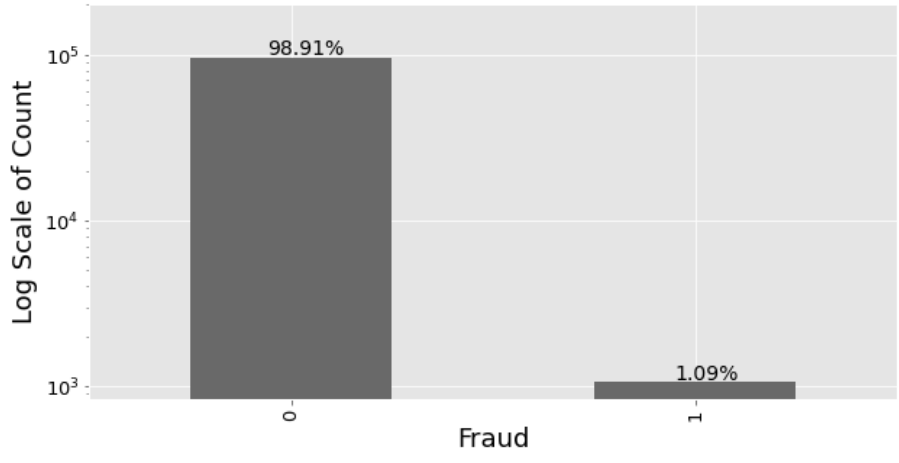

Figure 1.3.5: Frequency Distribution of the *fraud* Field

2 Data Cleaning

2.1 Remove Frivolous Records

We first removed one outlier with a large transaction amount, which is more than 6 standard deviations away from the mean. We also only kept transactions with Transtype equals "P."

2.2 Fill Missing Values

Three features are not 100% populated, including Merchant number, Merchant state, and Merchant zip. To fill these missing values, we tried three methods.

2.2.1 KNN Imputation

Since the function for this algorithm in Python can only take in numerical values, we first used an Ordinal Encoder to transform data. In other words, we use a series of integers to replace each unique value of a categorical variable. For each record with missing values, we used K nearest Neighbors to find k data points closest to it. After calculating the mean of these neighbors, we rounded it to an integer and then filled the missing value. After imputation, we transformed the encoded data back to the original form.

2.2.2 Mode Imputation

This method aims to use the most frequent value to fill missing values.

1) Merchnum
   We first grouped the data by merchants' descriptions and found the most frequent Merchnum for each unique description. Missing values in Merchnum were filled with these most frequent values based on the description of each record.
2) Merch state
   We used the similar method mentioned above to fill missing values in Merch state. The only difference is that we grouped the data by Merch zip and then by Merch description.
3) Merch zip
   We grouped the data by Merchnum and then by Merch description. Then we used the previous method to deal with missing values in Merch zip.
4) "Unknown"
   There were still some records with missing values, so we replaced them with a new category, "Unknown."

2.2.3 Mode Imputation Based on Time

This method is similar to the previous one, but it is more complicated. It also took time into consideration.

1) Merch state
   We first filled missing values in Merch state with the most frequent state within its zip code. Then based on the Merch description, we found the latest merch state appeared before each record, or the first state appeared later. Eventually, we filled the remaining missing values with the most frequent value of state on that day.
2) Merch zip

For each Merchnum or Merch description, we found the latest zip code appeared before each record, and the first zip code appeared after it. Then we used these zip codes to fill missing values. For the remaining missing values, we filled them with the most frequent zip code within each state. There are 40 records in states outside the U.S. in our dataset. We assigned a random zip code for each of these states.

3) Merchnum
We first filled missing values with the most recent Merchnum associated with each Merch description. Some Merch descriptions do not have any Merchnum associated with them, so we generated a new Merchnum for each of them for these descriptions.

2.2.4 Final Choice for Missing Value Imputation

Based on the result of each imputation method, we built three same sets of variables and ran the feature selection process. Among these three methods, mode imputation seemed to have the best performance. Based on this imputation method, candidate variables achieved high K.S. values, and the curve plotted in the wrapper also seemed to be relatively stable. Therefore, we eventually chose the mode imputation.

3 Feature Creation

A total of 875 candidate variables were created based on the existing PII field and PII field combinations to describe the structures inherent in the data, thus better quantifying the characteristics of fraud behaviors. Table 3.0 summarizes the description and the number of variables created in each category.

Table 3.0: Summary of 875 Candidate Variables

Category	Description	# of Variables Created
Weekday Risk	Average fraud likelihood of each day of the week	1
Days Since Last Seen	Number of days since the last appearance of the same PII or PII combination	12
Velocity	The number of records of the same PII or PII combination showed over a specific time window before	72
Relevant Velocity	Measures if there is a sudden increase of transactions of the same PII or PII combination in a short period relative to a long-term frequency	192
Amount	Measures of the amount at the same PII or PII combination over different time periods	576
Benford's Law	Analyzes the first digit of amounts to check for made up transactions	2

3.1 Weekday Risk

Considering the possibility that the likelihood of someone who commits fraud may vary from a different day of a week, a categorical variable that represents the weekday was created. Targeting Encoding was employed to encode this categorical variable.

Target Encoding (aka Risk Tables) means for each possible category assign a specific value to it. The value is usually the average of the dependent variables for all training and testing records (except for validation data) in that category. But when the records are not enough in the category, a better estimation would be to smoothly transit between the two measurements (categorical average and overall average), which is done by a logistic function below:

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

where $Y_{\text{high}}, Y_{\text{low}}$ denote the overall average and categorical average here, n_{mid} is the value of *n* where the smoothed value is halfway between Y_{high} and Y_{low}C is a measure of how quickly it transitions. The average probabilities of fraud by weekday are summarized in Table 3.1 below and the resulting new variable was named *dow_risk.*

Table 3.1: Average Probability of Fraud by Weekday

Weekday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Average Fraud	0.0087	0.0071	0.0098	0.0186	0.0260	0.0100	0.0096

3.2 Days Since Last Seen

Since the goal of this project is to find transaction fraud which is the act of unusual transaction activity, there are mainly three forms that a transaction seems suspicious: 1) one PII occurred many times with other different PIIs frequently; 2) different PIIs occurred with one same PII in short periods; 3) the amounts of transaction of one PII varies a lot in short periods. Therefore, the frequency of a PII or PII combination appeared and the amounts of transactions that occurred are essential indicators of fraudulent behaviors.

Day since last seen, which represents the number of days since the last appearance of the same PII or PII combination, is one of such indicators. A smaller value means a closer previous appearance of the same PII, and therefore a higher likelihood of fraud. If a PII has never occurred in the dataset before, its corresponding day since the last seen variable will take the number of days since the earliest date in the dataset, 2010/1/1, to be specific, as its value. Figure 3.1 shows how the total 12 relevant variables are built from the existing PII and PII combinations.

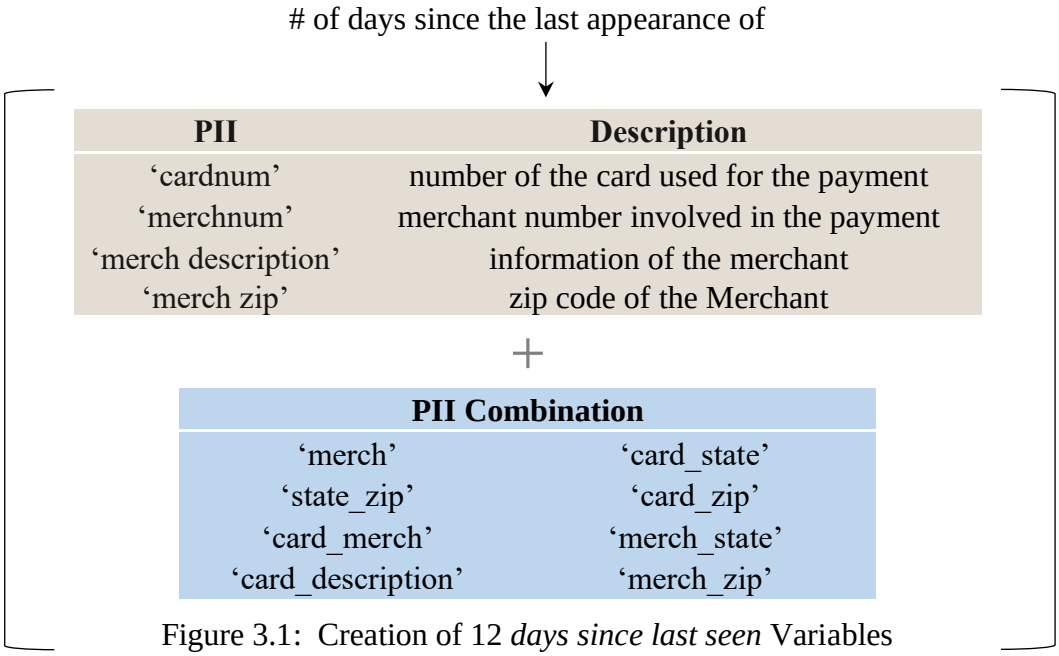

Figure 3.1: Creation of 12 *days since last seen* Variables

$* \ 4 + 8 = 12$ *days since last seen* variables
** We didn't include 'merch state' in the PII since it has many unknown values, leading to a much longer time to merge with other variables
*** 'Merch' refers to 'merchnum'+'merch description'+'merch state'+'merch zip.'

It's worth noting that the records of the first couple weeks of 2010 have small values in the *days since last seen* variables because the largest possible values these variables can take are

the number of days since 2010/1/1, which are small for early records. To avoid such misleading information being fed into the model, the first two weeks of records were excluded from modeling.

3.3 Velocity

Velocity is another way to capture the frequency of appearance of the same PII, which denotes the number of records of the same PII or PII combination showed over a certain time window before. A more considerable value means frequent applications of the same PII, and therefore a higher likelihood of fraud. The time window used were 0, 1, 3, 7, 14, and 30 days, with 0 days meaning the same day of the last transaction. Figure 3.2 shows how the total 72 relevant variables are built from the existing PII and PII combinations.

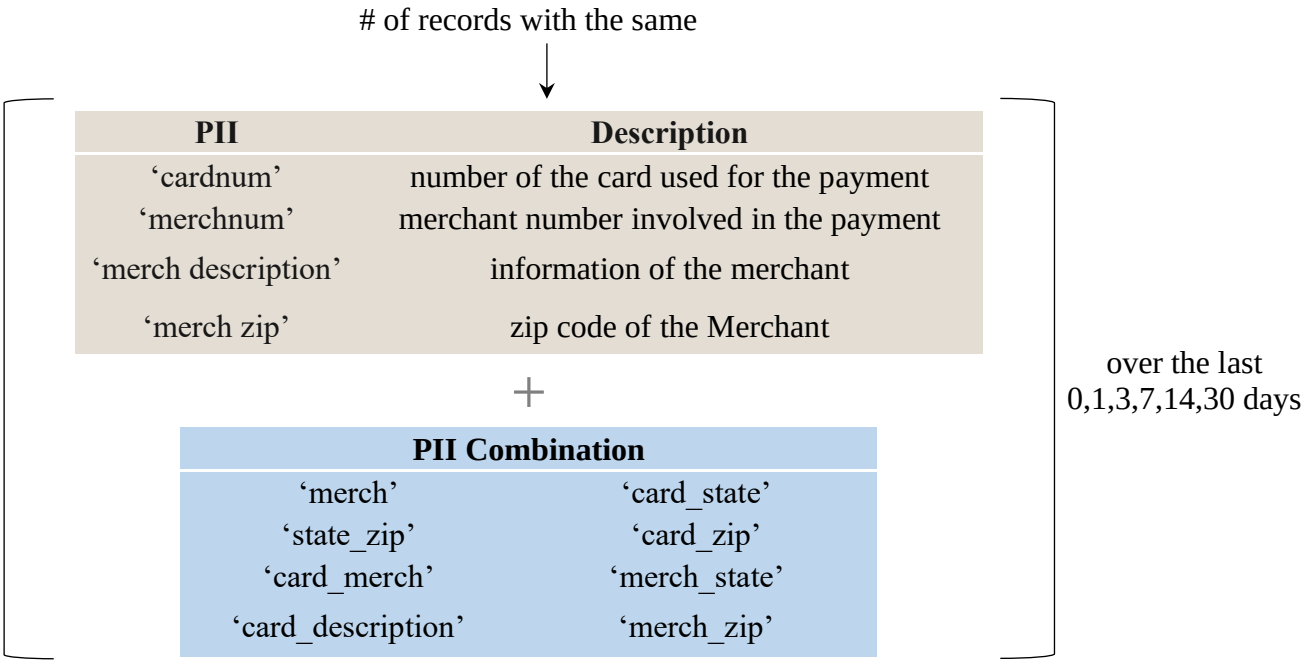

Figure 3.2: Creation of 72 *velocity* Variables

$*$ $(4 + 8) \times 6 = 72$ *velocity* variables
$**$ We didn't include 'merch state' in the PII since it has many unknown values, leading to a much longer time to merge with other variables
*** 'Merch' refers to 'merchnum'+'merch description'+'merch state'+'merch zip.'

3.4 Relative Velocity

Based on the velocity variables, relative variables were created to describe further the sudden increase of the same PII transactions in a short period relative to a long-term frequency. The rationale behind this is that given the same number of occurrences of the same PII in a set of time period (i.e., a 30-day window), a situation where all the occurrences happened on the same day is more likely to be a fraud than another situation where the occurrences were spread out on different dates. The detailed calculation process is shown below, and a total of 192 variables were created.

$$\text{Relative Velocity} = \frac{[number, amount]\ trans\ with\ that\ group\ over\ the\ past\ [0,1]\ days}{[number, amount]\ trans\ with\ same\ group\ over\ the\ past\ [3,7,14,30]\ days}$$

$* 2 \times 4 \times 2 \times 12 = 192\ relative\ velocity$ variables

3.5 Amount

Creating amount variables could help detect the unusual amount of transactions of PII combinations in a certain time window. For example, suppose for the same cardnum, the average amount of transactions recorded in a short time period is relatively high. In that case, such transactions have a higher possibility of being regarded as transaction fraud. Therefore, the greater the amount variables are, the more likely a transaction is a potential fraud. Figure 3.3 shows how a total of 120 cross entity velocity variables were calculated.

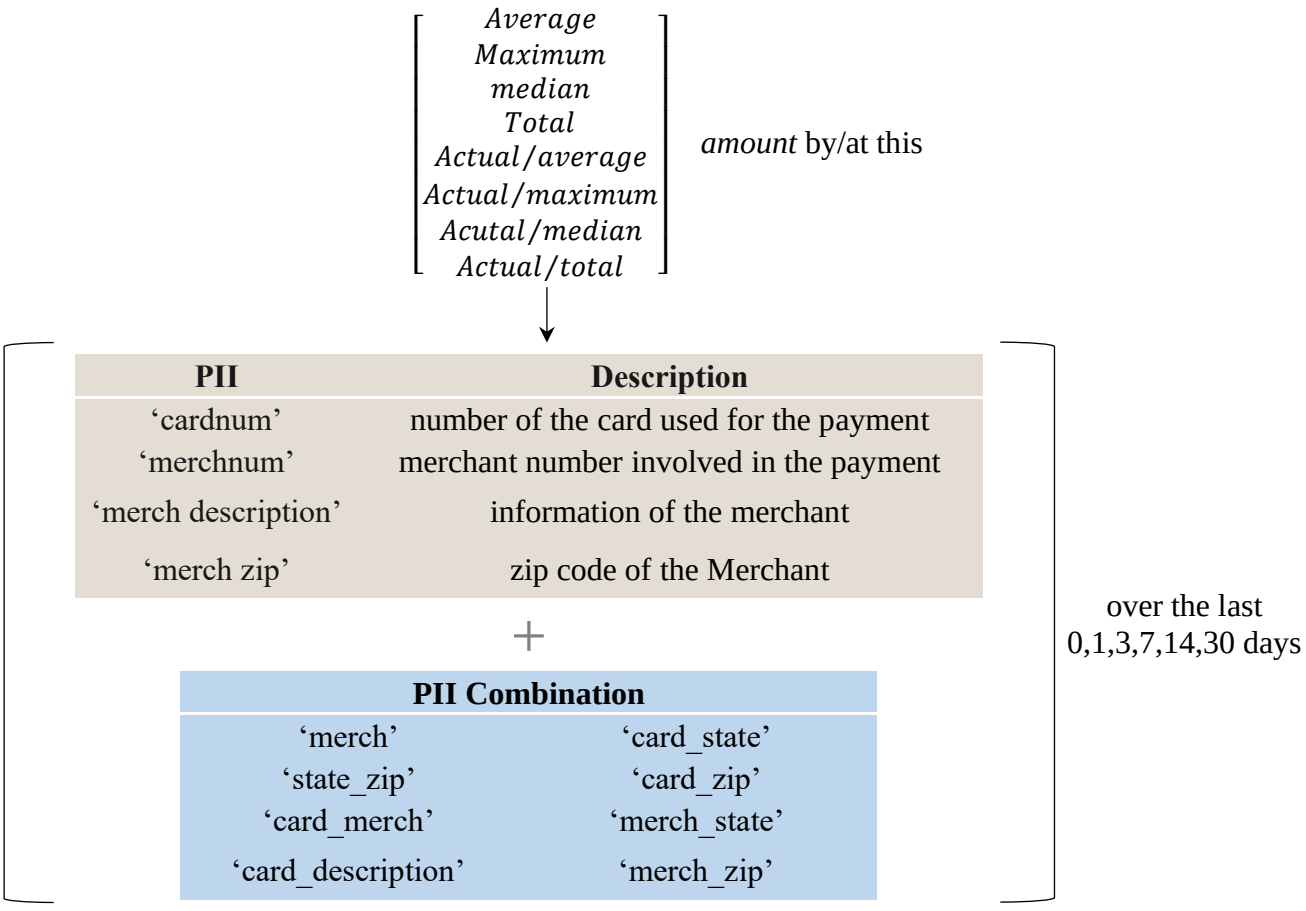

Figure 3.3: Creation of 576 *amount* Variables

$* 8 \times 12 \times 6 = 576$ *amount* variables
$**$ We didn't include 'merch state' in the PII since it has many unknown values, leading to a much longer time to merge with other variables
*** 'Merch' refers to 'merchnum'+'merch description'+'merch state'+'merch zip.'

3.6 Benford's Law

Benford's Law is the non-intuitive fact that the first digit of many measurements is not uniformly distributed, with 1 being the most frequent (~30%) and 9 being the least frequent. Therefore, if a fraudster makes up transactions, he/she usually doesn't know about Benford's Law, so the transaction amounts are uniformly distributed random numbers. By examining the distributions of the first digit of the amount for each cardholder ('cardnum') and merchant ('merchnum'), we could see if the amount distributions substantially violate Benford's Law. In case we don't have enough records for each bin, we just divided all the first digits (1-9) into two bins, a lower (1-2, takes up 47.7%) bin and a higher (3-9, takes up 52.3%) bin. For each cardnum or merchnum, we counted the number of first digits beginning with either 1 or 2 (n_{low}), then $n_{high} = n - n_{low}$. We could thus measure the unusualness by the following formula:

$$U = \max(R, 1/R)$$

where R = $\frac{52.3\%/47.7\% \times n_{low}}{n_{high}}$. Therefore, the greater the U is, the more likely a transaction is a potential fraud. Here, the new variable U will be added as one of the total variables.

4 Feature Selection

4.1 Why Apply Feature Selection

Feature Selection is a very critical component in a Data Scientist's workflow. When presented data with very high dimensionality, models usually choke because

1. Training time increases exponentially with a large number of features.
2. Models have an increased risk of overfitting when the number of features grows.
3. Models are hard to interpret due to the high complexity
4. Data is always sparse, and all points become outliers, thus needing exponentially more data to see true nonlinearities rather than noise

Feature selection methods help with these problems by reducing the dimensions without much loss of complete information. It also assists in making sense of the features and their importance. Therefore, feature selection was performed for this project which went through the list of candidate variables and found the best predictors for modeling.

4.2 Data Preprocessing

Before feature selection, the following three steps were taken to prepare the data.

4.2.1 Feature Scaling

Feature scaling helps differently towards different models.

1. **Gradient Descent Based Algorithms:** Having features on a similar scale can help the gradient descent converge more quickly towards the minima.

   Machine learning algorithms such as linear regression, logistic regression, neural network, etc. that use gradient descent as an optimization technique require data to be scaled. Take a look at the formula for gradient descent below:

   $$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

   The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, scaling the data before feeding it to the model is necessary.

2. **Distance-Based Algorithms:** scaling data before employing a distance-based algorithm so that all the features contribute equally to the result.

   Distance algorithms like KNN, K-means, and SVM are most affected by the range of features. This is because, behind the scenes, they use distances between data points to determine their similarity. If different features have different scales, there is a chance that higher weightage is given to features with higher magnitude. This will impact the performance of the machine learning algorithm.

Therefore, feature scaling is necessary before feature selection and modeling. For this project, z-scaling is performed for all features.

4.2.2 Drop the First Two Weeks Data

The first two weeks' data was dropped to avoid biased *days since the last seen* variables. The detailed reason was explained in section 3.2.

4.2.3 Dataset split

After dropping the first two weeks' data, the remaining data was split into two parts: modeling data and out-of-time (oot) validation data based on different time periods, that is, using the last two months' data as oot data and the other as modeling data. Then, put the modeling data as the only input of feature selection to avoid overfitting. After feature selection, the modeling data will be split further into training and testing data.

4.3 Filter

Why uses filters?

Filter methods use a statistical calculation to evaluate the relevance of the predictors outside of the predictive models and keep only the predictors that pass some criterion. The goal of the filter is to know how important each variable is by itself to predict y. Considerations when choosing filter methods are the types of data involved, both in predictors and outcome —— either numerical or categorical. Common filter methods for binary classification problems include Pearson Correlations, Mutual Information, univariate Kolmogorov-Smirnov (K.S.) score, and Fraud Detection Rate (FDR).

For this project, K.S. and FDR@3% were calculated to rank the 875 variables. The average ranking by two scores was then used to filter out the top 80 variables to be moved on to the next step of feature selection.

4.3.1 Kolmogorov-Smirnov (KS)

Why is KS a good filter?

K.S. is a simple and robust measure of how well two distributions are separated (goods vs. bads). For each candidate variable, plot the goods and bads separately as in figure 4.3.1. The more different the curves, the better the variable for separating, and thus the more important the variable.

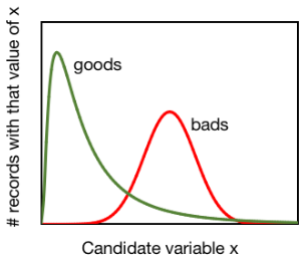

Figure 4.3.1.1: The distribution of goods and bads of candidate variable x

How does K.S. work?

For each candidate feature, two distributions of fraud (bad) and non-fraud (goods) records are built, respectively, and then the K.S. score is calculated based on the formula below.

$$KS = \max_{x} \int_{x_{min}}^{x} [P_{\text{goods}} - P_{\text{bads}}]\, dx$$

Figure 4.3.1.2 is drawn to explain the formula visually. After plotting the fraud (bad) and non-fraud of a candidate variable, start adding them up (cumulative distribution). The KS is then the maximum of the difference of the cumulative.

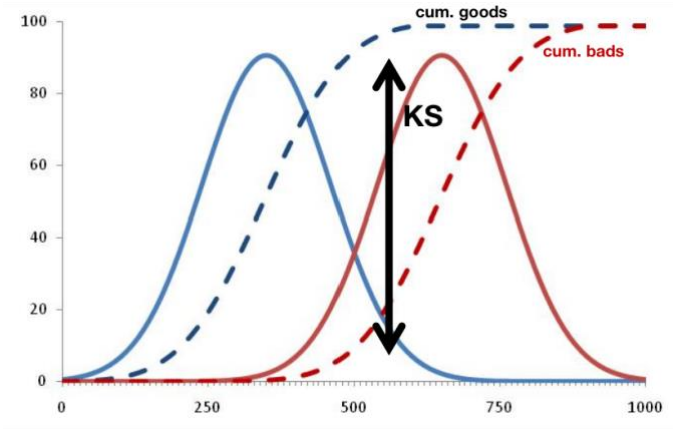

Figure 4.3.1.2: How to plot K.S.

4.3.2 Fraud Detection Rate (FDR) @3%

Why is FDR a good filter?

FDR is also used as a filter metric. It describes how many frauds can be caught within a certain population. FDR is very common in business applications. It's more robust and meaningful than the False Positive measure of goodness.

How does FDR work?

FDR describes what % of all the frauds are caught at a particular examination cutoff location. For Example, FDR 50% at 3% means the model captures 50% of all the frauds in 3% of the population. FDR is calculated as the number of true frauds in the bin, which are caught by the model, divided by the total number of true frauds in the entire dataset. FDR reflects how many frauds can be caught by a model with a fixed number of predicted positives.

$$\text{FDR@3\%} = \frac{\text{\# frauds caught at 3\% rejection rate}}{\text{total \# frauds in the dataset}}$$

4.3.3 Combine KS and FDR Rankings

In filter selection, each feature is scored and ranked by both K.S. and FDR, and the average ranking is used as the final ranking, with each metrics contributing 50% (see the formula below). The top 80 features (see Appendix B) are selected for further selection by wrapper methods.

$$\text{Average Rank of Scores} = \frac{\text{KS Rank (Descending)} + \text{FDR Rank (Descending)}}{2}$$

4.4 Wrapper

Why use wrapper?

While a filter is fast, it ignores interactions with classifiers. Instead, the Wrapper methodology is based on a specific machine learning algorithm used to fit on a given dataset. It considers the selection of feature sets as a search problem, where different combinations are prepared, evaluated, and compared to other combinations.

How does wrapper work?

Wrapper methods for feature selection can be divided into three categories: 1) Step forward feature selection; 2) Step backward feature selection; 3) Exhaustive feature selection. For this project, we used forward selection to select 10 variables out of 80 variables. Because compared to the backward selection, forward selection could reduce multicollinearity issues. Meanwhile, there is little FDR difference (around 0.02) between the two methods. Most importantly, forward selection performed well in the later modeling part.

The detailed process of forward selection is as follows:

1. Start with 1 variable whose FDR is the highest
2. Add the variable one by one at each step, and FDR was used as the evaluation criterion. That is, after adding that variable, the remaining variables have the highest FDR
3. Repeat step 2 until any further variable addition leads to a dramatic FDR decrease or the number of remaining variables reaches 80

After wrapper, a total of 10 variables were chosen to be used for modeling. The final features are listed in Table 4.4 below.

Table 4.4: Final Features for Modeling

Feature Name	Description
total_Amount_card_description_7	total amount by the given **card + merch description** combination seen in the past 7 days
max_Amount_card_state_14	max amount by the given **card + merch state** combination seen in the past 14 days
total_Amount_card_description_1	total amount by the given **card + merch description** combination seen in the past 1 day
total_Amount_card_description_0	total amount by the given **card + merch description** combination seen in the same day
max_Amount_card_state_30	max amount by the given **card + merch state** combination seen in the past 30 days
total_Amount_card_description_3	total amount by the given **card + merch description** combination seen in the past 3 days
total_Amount_Cardnum_3	total amount by the given **card** seen in the past 3 days
total_Amount_Cardnum_0	total amount by the given **card** seen in the same day
total_Amount_Cardnum_14	total amount by the given **card** seen in the past 14 days
max_Amount_Cardnum_3	max amount by the given **card** seen in the past 3 days

5 Modeling

We compared different algorithms with several combinations of parameters, including Logistic Regression, Neural Network, Light GBM, Decision Tree, Random Forest, SVM, and KNN. To prevent overfitting, we trained the model 10 times. Each time when we trained the model, we split the data into training and testing datasets randomly. Since our dataset is highly unbalanced, the ratio of goods and bads is about 90:1, so we performed oversampling on the training dataset. Then we fit the model on the oversampled training dataset. Eventually, we calculated the average FDR at a 3% rejection rate to compare different models. Neural Network achieved the highest FDR on out-of-time data and was selected as the final model. Details of other models are shown in the model performance table.

Marshall Consulting

Table 5.0: Model Performance Summary

Model	Parameters								FDR at 3%		
LogisticRegression	# Variables	penalty	C		solver		l1_ratio		Train	Test	OOT
	10	l1	100		liblinear		N/A		62.94	68.29	44.86
	10	l1	10		liblinear		N/A		63.69	66.45	45.31
	10	l2	1		lbfgs		N/A		59.73	68.25	41.84
	10	l2	10		lbfgs		N/A		62.48	68.53	46.2
	10	l2	100		liblinear		N/A		62.73	69.27	45.36
	10	l2	0.001		lbfgs		N/A		54.94	67.51	45.2
Neural Network	# Variables	max_iter	layer	nodes	solver		activation		Train	Test	OOT
	10	200	1	10	sgd		relu		82.74	80.18	52.51
	10	400	1	20	adam		relu		81.06	84.79	58.1
	10	200	1	10	adam		logistic		82.81	78.34	58.1
	10	200	1	10	sgd		logistic		79.91	77.42	49.72
	10	200	1	20	sgd		logistic		81.55	80.18	49.16
	10	200	1	20	adam		relu		82.71	84.33	53.63
	10	400	1	10	adam		relu		84.58	81.57	56.42
Light GBM	# Variables	learning_rate	n_estimators	max_depth	min_child_samples	subsample	col_sample_bytree	num_leaves	Train	Test	OOT
	10	0.1	500	2	50	1	1	3	87.98	84.10	37.10
	10	0.01	600	3	100	1	1	6	80.39	81.52	36.59
	10	0.01	700	4	150	0.8	0.8	15	87.44	84.29	38.60
	10	0.001	800	5	200	0.8	0.8	30	82.62	82.35	49.72
	10	0.001	900	5	250	0.6	0.6	30	83.66	81.34	43.97
	10	0.0001	1000	5	300	0.6	0.6	30	74.36	73.50	40.84
Decision Tree	# Variables	criterion	splitter	max_depth	max_leaf_nodes	min_samples_leaf	max_features	min_samples_split	Train	Test	OOT
	10	entropy	best	45	10	70	auto	20	60.05	65.67	37.99
	10	gini	best	20	10	70	auto	20	62.72	68.25	40.22
	10	gini	best	10	20	30	auto	20	70.01	74.51	40.67
	10	gini	best	10	10	30	auto	20	61.34	67.7	46.7
	10	gini	best	5	5	30	auto	30	56.58	60	39.55
	10	gini	random	5	5	30	None	30	53.21	63.09	40.5
Random Forest	# Variables	criterion	n_estimators	max_depth	min_samples_leaf		min_samples_split		Train	Test	OOT
	10	gini	50	100	20		100		88.53	84.84	48.94
	10	gini	60	200	10		100		88.53	83.78	49.94
	8	gini	60	500	10		100		85.6	83.18	52.46
	8	gini	60	1000	10		100		85.41	81.47	52.74
	8	gini	70	800	10		80		86.93	83.36	54.47
	8	gini	80	1000	10		50		88.99	84.7	56.93
SVM	# Variables		C		gamma		kernel		Train	Test	OOT
	10		1.082019792		2.96E-11		rbf		52.65	66.73	45.98
	10		0.1		2.96E-10		rbf		54.08	64.56	45.25
	10		0.1		2.96E-11		rbf		28.78	34.01	26.03
	10		10		2.96E-11		rbf		53.58	64.56	45.92
	10		10		2.96E-10		rbf		53.64	65.58	45.7
KNN	# Variables	n_neighbors	weights	algorithm	leaf_size	p	metric		Train	Test	OOT
	10	14	uniform	auto	30	2	minkowski		95.09	83.78	31.62
	10	8	uniform	kd_tree	30	2	minkowski		98.08	86.27	30.17
	7	14	uniform	kd_tree	30	2	minkowski		95.3	85.02	31.28
	7	14	uniform	auto	30	2	minkowski		95.3	85.9	31.12
	10	9	uniform	auto	30	2	euclidean		98.16	86.73	33.3
	10	16	uniform	auto	30	2	chebyshev		95.31	85.07	35.75

5.1 Data Preprocessing

Before modeling, preprocessing data is required to train a stronger and more robust model better. For this project, weighting and capping were employed.

5.1.1 Weighting

Since the dataset is highly imbalanced, with only around 1.09% of frauds, it is necessary to leverage resampling techniques to alleviate the effect of the imbalanced class size on the predictive power of classification models. Either oversampling the bads (fraud) or undersampling the goods (non-fraud) could help. Good/bad ratio between 1:1 to 10:1 is typically best. Here, undersampling was performed on the training data every time a model was fitted, and a good/bad ratio of 10:1 was applied.

5.1.2 Capping

Many variables have long tails, where high is bad. It is easier to learn from either the "very good" or "very bad" records compared to those in between. Capping excludes "very high" and "very low" data and forces models to focus on and learn from the remaining in-between areas (e.g., the slightly likely to be fraud records), which could improve the model's learning power. Here, all z-scaled variables were capped at six z-scores. Values more than six standard deviations away from the variable mean were dropped.

5.2 Model Training

The preprocessed data is now ready for model training and parameter tuning. Then we used a method that combines both cross-validation and bootstrapping to train our model. In general, we put the out-of-time data aside ran the following steps 10 times:

1) Randomly split the dataset (without out-of-time data) into train and test datasets.
2) Oversample the training dataset, increase the ratio of goods and bads to 10:1.
3) Fit the model on oversampled training data.
4) Calculate the FDR at a 3% rejection rate on the train, test, and out-of-time dataset.

Eventually, we calculated the average FDR for three datasets from the 10 models trained in previous steps.

This method can help us to test the generalizability of the model. Suppose we fit the model on training data and validate it on the testing data only once; certain observations may lead to a good performance. However, if we use the model to make predictions on the data that it has never seen before, say, OOT data may have significantly worse performance. Therefore, we can use both cross-validation and bootstrapping to ensure that our model will also have a good performance on new data.

We used this method to compare many different algorithms with various combinations of hyperparameters. Eventually, Neural Network had the best performance on the out-of-time data.

5.2.1 Logistic Regression

Logistic regression (or logit regression) is a predictive statistical model that utilizes logistic function to model the probability of output with two possible outcomes in terms of input. It is an applicable regression analysis model to use when the dependent variable is binary. The model intakes a number of parameters called independent variables and a binary dependent variable of 0 or 1. The label "1" usually represents an event that happens, or the observed object belongs to a particular class. The label "0" represents otherwise. This kind of model is also called a binary logistics regression model. A graph that indicates the basics of logistics regression is shown below.

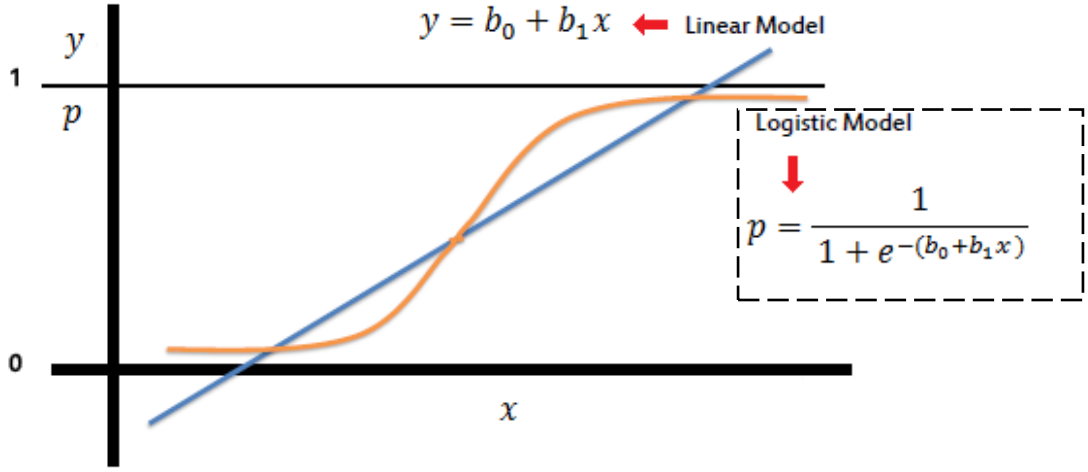

Figure 5.2.1: Logistic Model and Linear Model Illustration

Due to their simplicity and efficiency, logistics regression models are widely used in classification problems. In this identity fraud project, we decided to use the binary logistics regression model as a baseline model with all useful variables from our feature selection process as independent variables and fraud labels indicate whether the transaction is fraudulent (1) or non-fraudulent (0) in the dataset as our dependent variable.

Given the flexibility available with this model, we tuned the below parameters (see table 5.2.1):

Package: sklearn.linear_model.LogisticRegression

Table 5.2.1: Essential Paramemters in Logistic Regression

Name	**Description**
penalty	Used to specify the norm used in the penalization. We used the default value "*l2*".
C	Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization. We used the value of *0.01*.

n_jobs	Algorithm to use in the optimization problem. For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones. As this is a small dataset, we tried "*lbfas*" (default) and "*liblinear*."
l1_ratio	The Elastic-Net mixing parameter, with $0 \leq$ l1_ratio $\leq$ 1. Only used if penalty='elasticnet'. Setting l1_ratio=0 is equivalent to using penalty='l2', while setting l1_ratio=1 is equivalent to using penalty='l1'. For 0 < l1_ratio <1, the penalty is a combination of L1 and L2. We set l1_ratio = "*none*".

5.2.2 Random Forest

Random Forest builds a collection of many independent, strong trees and averages across them. Within each decision tree, a node is a predictor, and the leaf represents the number of samples in each class. For each tree and/or for each split iteration within a tree, the model uses only a randomly chosen subset of variables or records. After finishing building trees, the model combines all the results by averaging for regression or voting for a classification.

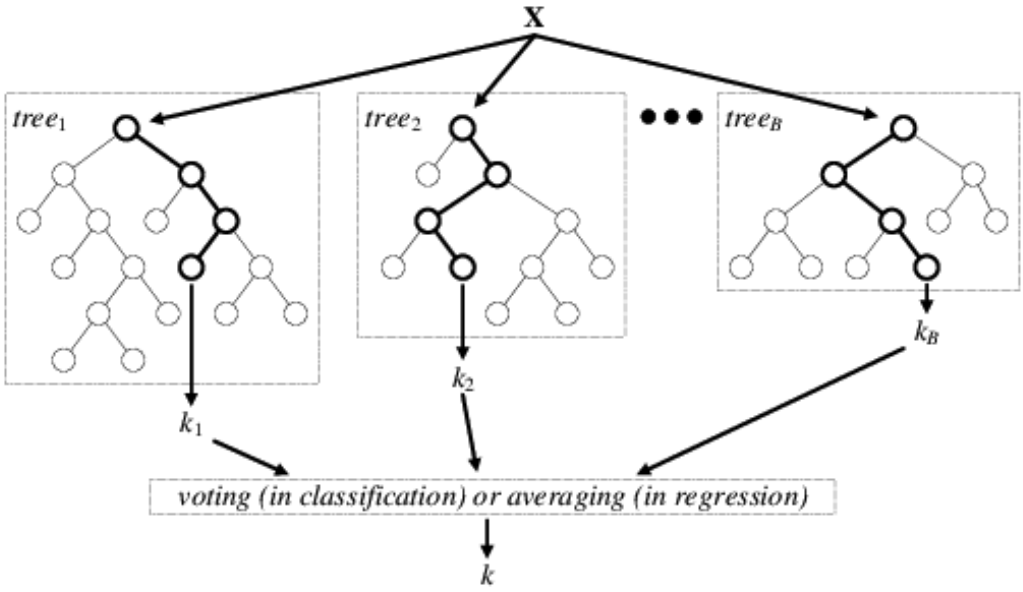

Figure 5.2.2: Random Forest Model Illustration

For the number of variables used in random forest, we tried both 10 variables and 8 variables because we found 10 variables sometimes cause overfit. Therefore, to prevent overfitting from happening, we reduce dimension by only using 8 variables. Given the flexibility available with this model, we tuned six parameters (see table 5.2.2):

Package: sklearn.ensemble.RandomForestClassifier

Table 5.2.2: Essential Parameters in Random Forest

Name	**Description**
n_estimators	The number of trees in the forest. At each iteration, we choose a number between *50 and 100.*
max_depth	The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
max_features	The number of features to consider when looking for the best split. In this project, we typically choose from max_features=*sqrt (n_features)* and max_features=*5*.
min_samples_leaf	The minimum number of samples required to be at a leaf node. We choose from *1 to 10* for this parameter.
min_samples_split	The minimum number of samples required to split an internal node. Our choices range from *2 to 10*.
criterion	The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

5.2.3 Boosted Tree

Boosted trees classifiers use the combination of results of a series of simple tree models to predict the outcome variable. Each subsequent tree in the series aims to capture the residual errors of the previous tree.

Currently, the most popular boosted trees models are XGBoost and LightGBM (LGBM). Tianqi Chen developed XGBoost in 2014. Microsoft released LightGBM in 2017. The most significant difference between these two models are the ways they grow their trees. While XGBoost grows its tree level-wise, LightGBM's trees grow leaf wise, allowing LightGBM to reduce more loss. Also, LightGBM is often faster to train. In addition, LightGBM tends to occupy less memory. Therefore, we have decided to use LightGBM (LGBM).

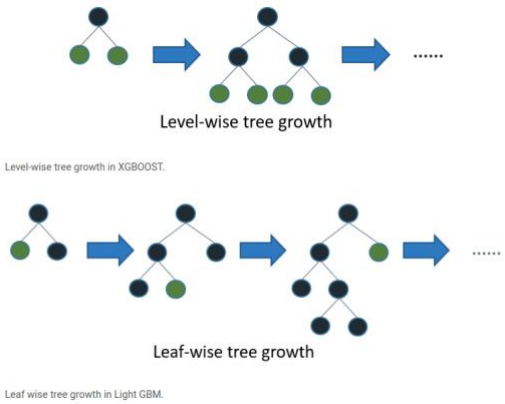

Figure 5.2.3: XGBoost and LightGBM Model Illustration

We have tuned these parameters for LightGBM (see table 5.2.3).

Package: lightgbm.LGBMClassifier

Table 5.2.3: Essential Paramemters in LightGBM

Name	**Description**
n_estimators	Number of boosted trees to fit. *We used n_estimators= 500, 600, 700, 800, 900, 1000.*
max_depth	Maximum depth of the tree. Increasing this parameter will also increase the complexity of the model and the likelihood of overfitting. *We used max_depth = 2,3,4,5*
min_child_samples	Minimum number of data needed in a child (leaf). A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. Setting this parameter can help to prevent overfitting. *We used min_child_samples = 50, 100, 150, 200, 250, 300*
num_leaves	Maximum tree leaves for base learners. *Num_leaves=3,6,15,30*
learning_rate	Size of shrinkage. After each boosting step, we can get the weights of new features, and eta shrinks the feature weights to prevent overfitting. *We used learning_rate = 0.1, 0.01, 0.001, 0.0001*
subsample	The percent of data used for training each tree. *Subsample = 1, 0.8, 0.6*
colsample_bytree	The percent of columns used for training each tree. *Colsample_tree = 1, 0.8, 0.6*

5.2.4 Neural Network

An artificial neural network (ANN) is an algorithm that shares a simulative structure of the neural network of the human brain. There are an input layer, hidden layers, and an output layer in the overall network architecture. Each of these layers contains nodes or neurons, which gather locations for the receipt and transmission of numerical signals from the previous to the next layer of the neural net. Each neuron embedded in the structure receives a signal from the nodes in the previous layer, applies a transfer function to the signal, and then outputs a new signal to the nodes in the next layer. The signal received from the previous layer is, in general, a linear combination of the outputs of the previous layer nodes. This combined linear combination signal, received by the node, is then passed through a transfer function, typically a sigmoid or logit function. Other transfer functions are also used, but the sigmoid/logit is the most common.

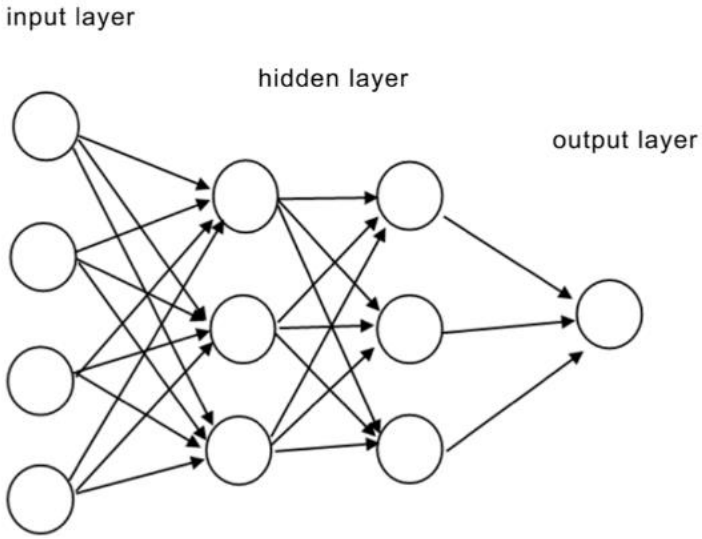

Figure 5.2.4: Neural Network Model Illustration

The parameters used for Neural Network are as below (see table 5.2.4).

Package: sklearn.neural_network.MLPClassifier

Table 5.2.4: Essential Parameters in Neural Network

Name	**Description**
hidden_layer_sizes	The ith element represents the number of neurons in the ith hidden layer. We use value nodes=10,20 and layers=*1*. ***{nodes, layers=1(Default)}***
activation	Activation function for the hidden layer. We use the values '*relu*' and '*logistic*.' ***{'identity,' 'logistic,' 'tanh,' 'relu'}***
solver	The solver for weight optimization. We use the values '*sgd'* and '*adam*.' ***{'lbfgs', 'sgd', 'adam'}, default='adam'}***
max_iter	Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd,' 'adam'), note that this determines the number of epochs (how many times each data point will be used). We use values *200 and 400*. ***default=200***

5.2.5 Support Vector Machine (SVM)

The objective of the support vector machine algorithm is to find a hyperplane in an n-dimensional space (n = the number of features) that distinctly classifies the data points. As illustrated by Figure 5.2.4, in two-dimensional space, this hyperplane is a line dividing a plane into two parts where each class lies on either side. SVM is especially effective in high-dimensional spaces.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane.

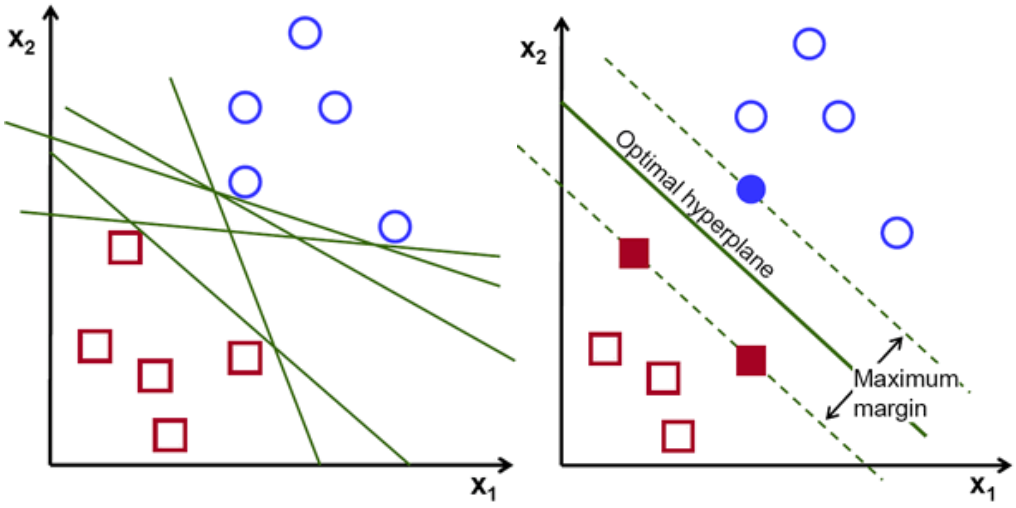

Figure 5.2.5: Support Vector Machine Model Illustration

The parameters used for the SVM are as below (see table 5.2.5).

Package: sklearn.svm.SVC

Table 5.2.5: Essential Parameters in SVM

Name	**Description**
C	Regularization parameter. The strength of the regularization is inversely proportional to C. It Must be strictly positive. The penalty is a squared l2 penalty. We tried *1.082*, *0.1*, and *10*.
gamma	Kernel coefficient for 'rbf,' 'poly' and 'sigmoid.' We used *2.96E-11*and *2.96E-10*.
kernel	Specifies the kernel type to be used in the algorithm. It must be one of *'linear,' 'poly,' 'rbf,' 'sigmoid,' 'precomputed'* or a *callable*. We used "*rbf*" (default).

5.2.6 Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test. Each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

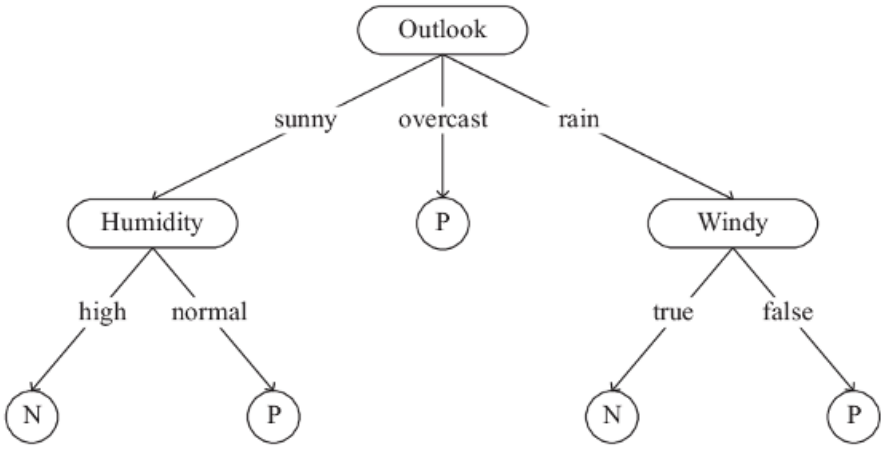

Figure 5.2.3: Decision Tree Model Illustration

Package: DecisionTreeClassifier

Table 5.2.3: Essential Parameters in Decision Tree

Name	**Description**
criterion	The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. *We tried criterion = entropy and gini*
splitter	The strategy used to choose the split at each node. *We tried splitter = best and random*
max_depth	The maximum depth of the tree. *We tried max_depth = 45, 20, 10, 5*
max_leaf_nodes	Grow a tree with max_leaf_nodes in best-first fashion. *We tried max_leaf_nodes = 10, 20, 5*
min_samples_leaf	The minimum number of samples required to be at a leaf node. *We tried min_samples_leaf = 70 and 30*
max_features	The number of features to consider when looking for the best split. *We tried max_features = auto, none*
min_samples_split	The minimum number of samples required to split an internal node. *We tried min_samples_split = 20 and 30*

5.2.7 K-nearest neighbors (KNN)

K-nearest neighbors (KNN) is a supervised machine learning algorithm that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data. KNN works by finding the distances between a query and all the examples in the data, selecting the specified number of examples (K) closest to the query, then votes for the most frequent label in our case of classification.

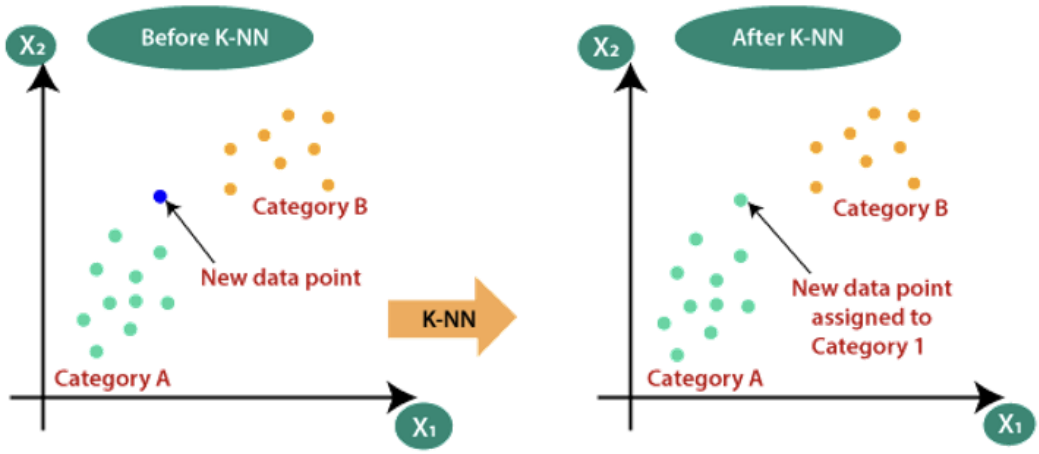

Figure 5.2.7: KNN Model Illustration

We have tuned these parameters for KNN (see table 5.2.7).

Package: sklearn.neighbors.KNeighborsClassifier

Table 5.2.7: Essential Parameters in KNN

Name	**Description**
n_neighbors	Number of neighbors to use by default for k neighbors queries. *We tried n_neighbors = 14, 8, 9, 16*
Weights	Weight function used in prediction: uniform, distance, callable. *We tried weights = uniform*
Algorithm	Algorithm used to compute the nearest neighbors: ball_tree, kd_tree, brute, auto. *We tried algorithm = auto and kd_tree*
leaf_size	Leaf size passed to BallTree or KDTree. *We tried leaf_size = 30*
P	Power parameter for the Minkowski metric. *We tried P = 2*
metric	Distance metric for searching neighbors. *We tried metric = minkowski, Euclidean and chebyshev*

6 Results

Neural Network (*max_iter=400, layer=1, nodes=20, solver='adam', activation='relu'*) reached the highest FDR at a 3% rejection rate on the out-of-time data and was selected as our final model. It achieved an FDR score at a 3% rejection rate of 81.06% on the training data, 84.79% on the testing data, and 58.1% on the OOT data. The key statistics of the top 20 bins in training, testing, and OOT data are shown in the tables below.

We first sorted the observations for each dataset by probabilities of being fraud predicted by our final model in descending order. Then each dataset was split into 100 bins, and the statistics of the top 20 bins are shown in the table. The green part on the left shows the statistics within a single bin, while the blue part on the right contains the cumulative statistics within the current bin and all the bins above it.

In each table, the number of bands detected in the first bin is the highest among all 100 bins and is significantly higher than the number in other bins. In the first 1% of the dataset with the highest probability of being a fraud, 54.4% of the records are fraud. This shows that our model has a good performance in detecting fraud. Also, cumulative K.S. scores in the top 20 bins remain high. For OOT data, most of the K.S. scores are more than 50% or even 60% in several top bins. This indicates that the final model can effectively tell the difference between goods and bads.

The final model performs the best on the training and testing dataset, but its performance is worse on the OOT data. This is expected because a model generally has better performance on the data that it has seen before. We have used the training data to offer information for the model in the training process, and the test data is from a similar period. The OOT data contains information from a period that the model has never encountered before, so it has the worst performance.

Table 6.1: Key Statistics of Top 20 Bins in Training Data

Training	# Records		# Goods		# Bads		Fraud Rate					
	60055		59580		475		0.007909416					
	Bin Statistics					Cumulative Statistcs						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	601	301	300	50.08%	49.92%	601	301	300	0.51%	63.16%	62.65%	1.00
2	601	551	50	91.68%	8.32%	1202	852	350	1.43%	73.68%	72.25%	2.43
3	601	574	27	95.51%	4.49%	1803	1426	377	2.39%	79.37%	76.98%	3.78
4	601	588	13	97.84%	2.16%	2404	2014	390	3.38%	82.11%	78.72%	5.16
5	601	587	14	97.67%	2.33%	3005	2601	404	4.37%	85.05%	80.69%	6.44
6	601	593	8	98.67%	1.33%	3606	3194	412	5.36%	86.74%	81.38%	7.75
7	601	596	5	99.17%	0.83%	4207	3790	417	6.36%	87.79%	81.43%	9.09
8	601	596	5	99.17%	0.83%	4808	4386	422	7.36%	88.84%	81.48%	10.39
9	601	593	8	98.67%	1.33%	5409	4979	430	8.36%	90.53%	82.17%	11.58
10	601	600	1	99.83%	0.17%	6010	5579	431	9.36%	90.74%	81.37%	12.94
11	601	600	1	99.83%	0.17%	6611	6179	432	10.37%	90.95%	80.58%	14.30
12	601	592	9	98.50%	1.50%	7212	6771	441	11.36%	92.84%	81.48%	15.35
13	601	599	2	99.67%	0.33%	7813	7370	443	12.37%	93.26%	80.89%	16.64
14	601	600	1	99.83%	0.17%	8414	7970	444	13.38%	93.47%	80.10%	17.95
15	601	599	2	99.67%	0.33%	9015	8569	446	14.38%	93.89%	79.51%	19.21
16	601	600	1	99.83%	0.17%	9616	9169	447	15.39%	94.11%	78.72%	20.51
17	601	600	1	99.83%	0.17%	10217	9769	448	16.40%	94.32%	77.92%	21.81
18	601	601	0	100.00%	0.00%	10818	10370	448	17.41%	94.32%	76.91%	23.15
19	601	600	1	99.83%	0.17%	11419	10970	449	18.41%	94.53%	76.11%	24.43
20	601	601	0	100.00%	0.00%	12020	11571	449	19.42%	94.53%	75.11%	25.77

Table 6.2: Key Statistics of Top 20 Bins in Testing Data

Testing	# Records		# Goods		# Bads		Fraud Rate					
	20158		19941		217		0.010764957					
	Bin Statistics					Cumulative Statistcs						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	202	67	135	33.17%	66.83%	202	67	135	0.34%	62.21%	61.88%	0.50
2	202	177	25	87.62%	12.38%	404	244	160	1.22%	73.73%	72.51%	1.53
3	202	193	9	95.54%	4.46%	606	437	169	2.19%	77.88%	75.69%	2.59
4	202	196	6	97.03%	2.97%	808	633	175	3.17%	80.65%	77.47%	3.62
5	202	198	4	98.02%	1.98%	1010	831	179	4.17%	82.49%	78.32%	4.64
6	202	199	3	98.51%	1.49%	1212	1030	182	5.17%	83.87%	78.71%	5.66
7	202	201	1	99.50%	0.50%	1414	1231	183	6.17%	84.33%	78.16%	6.73
8	202	200	2	99.01%	0.99%	1616	1431	185	7.18%	85.25%	78.08%	7.74
9	202	201	1	99.50%	0.50%	1818	1632	186	8.18%	85.71%	77.53%	8.77
10	202	199	3	98.51%	1.49%	2020	1831	189	9.18%	87.10%	77.91%	9.69
11	202	201	1	99.50%	0.50%	2222	2032	190	10.19%	87.56%	77.37%	10.69
12	202	201	1	99.50%	0.50%	2424	2233	191	11.20%	88.02%	76.82%	11.69
13	202	200	2	99.01%	0.99%	2626	2433	193	12.20%	88.94%	76.74%	12.61
14	202	202	0	100.00%	0.00%	2828	2635	193	13.21%	88.94%	75.73%	13.65
15	202	199	3	98.51%	1.49%	3030	2834	196	14.21%	90.32%	76.11%	14.46
16	202	201	1	99.50%	0.50%	3232	3035	197	15.22%	90.78%	75.56%	15.41
17	202	201	1	99.50%	0.50%	3434	3236	198	16.23%	91.24%	75.02%	16.34
18	202	202	0	100.00%	0.00%	3636	3438	198	17.24%	91.24%	74.00%	17.36
19	202	202	0	100.00%	0.00%	3838	3640	198	18.25%	91.24%	72.99%	18.38
20	202	199	3	98.51%	1.49%	4040	3839	201	19.25%	92.63%	73.37%	19.10

Table 6.3: Key Statistics of Top 20 Bins in OOT Data

OOT	# Records		# Goods		# Bads		Fraud Rate					
	12427		12248		179		0.01440412					
	Bin Statistics					Cumulative Statistcs						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	125	69	56	55.20%	44.80%	125	69	56	0.56%	31.28%	30.72%	1.23
2	125	97	28	77.60%	22.40%	250	166	84	1.36%	46.93%	45.57%	1.98
3	125	118	7	94.40%	5.60%	375	284	91	2.32%	50.84%	48.52%	3.12
4	125	123	2	98.40%	1.60%	500	407	93	3.32%	51.96%	48.63%	4.38
5	125	123	2	98.40%	1.60%	625	530	95	4.33%	53.07%	48.75%	5.58
6	125	123	2	98.40%	1.60%	750	653	97	5.33%	54.19%	48.86%	6.73
7	125	123	2	98.40%	1.60%	875	776	99	6.34%	55.31%	48.97%	7.84
8	125	124	1	99.20%	0.80%	1000	900	100	7.35%	55.87%	48.52%	9.00
9	125	123	2	98.40%	1.60%	1125	1023	102	8.35%	56.98%	48.63%	10.03
10	125	122	3	97.60%	2.40%	1250	1145	105	9.35%	58.66%	49.31%	10.90
11	125	123	2	98.40%	1.60%	1375	1268	107	10.35%	59.78%	49.42%	11.85
12	125	125	0	100.00%	0.00%	1500	1393	107	11.37%	59.78%	48.40%	13.02
13	125	125	0	100.00%	0.00%	1625	1518	107	12.39%	59.78%	47.38%	14.19
14	125	121	4	96.80%	3.20%	1750	1639	111	13.38%	62.01%	48.63%	14.77
15	125	121	4	96.80%	3.20%	1875	1760	115	14.37%	64.25%	49.88%	15.30
16	125	122	3	97.60%	2.40%	2000	1882	118	15.37%	65.92%	50.56%	15.95
17	125	125	0	100.00%	0.00%	2125	2007	118	16.39%	65.92%	49.54%	17.01
18	125	124	1	99.20%	0.80%	2250	2131	119	17.40%	66.48%	49.08%	17.91
19	125	124	1	99.20%	0.80%	2375	2255	120	18.41%	67.04%	48.63%	18.79
20	125	125	0	100.00%	0.00%	2500	2380	120	19.43%	67.04%	47.61%	19.83

6.1 Fraud Score Analysis

6.1.1 Cardnum

Employing the final model, we assigned each transaction with a fraud score, which is the probability of being fraudulent. Then, we found some card numbers having patterns that are interesting to explore. As illustrated by two graphs in Figure 6.1.1, there are more than 100 transactions associated with card 5142235211, where the last 32 transactions are highly likely to be fraudulent. Interestingly, by tracking the same card number over time, we found these 32 transactions happened within two days (11/25 and 11/26). Similarly, in August, a small bunching of transactions gives a slight rise in fraud scores. Overall, these patterns show card fraud usually happens in a very short time period with high frequency of transactions.

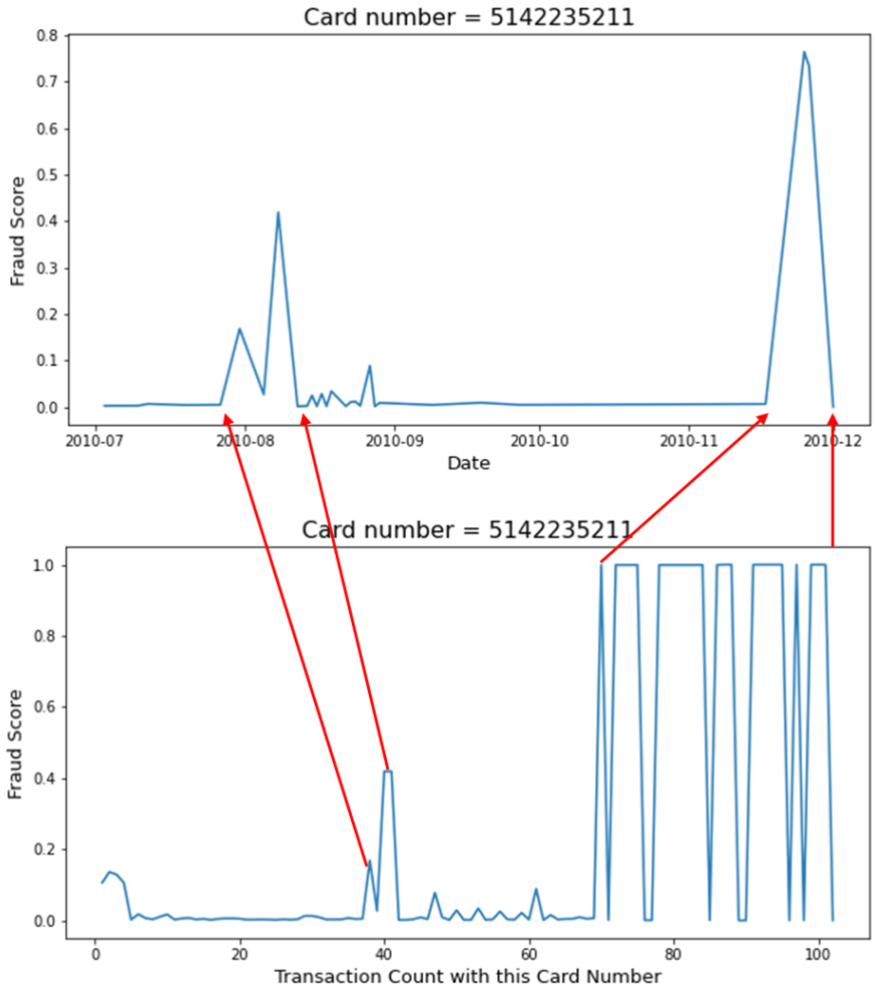

Figure 6.1.1: Fraud score analysis for Cardnum 5142235211 over time and count

6.1.2 Merchnum

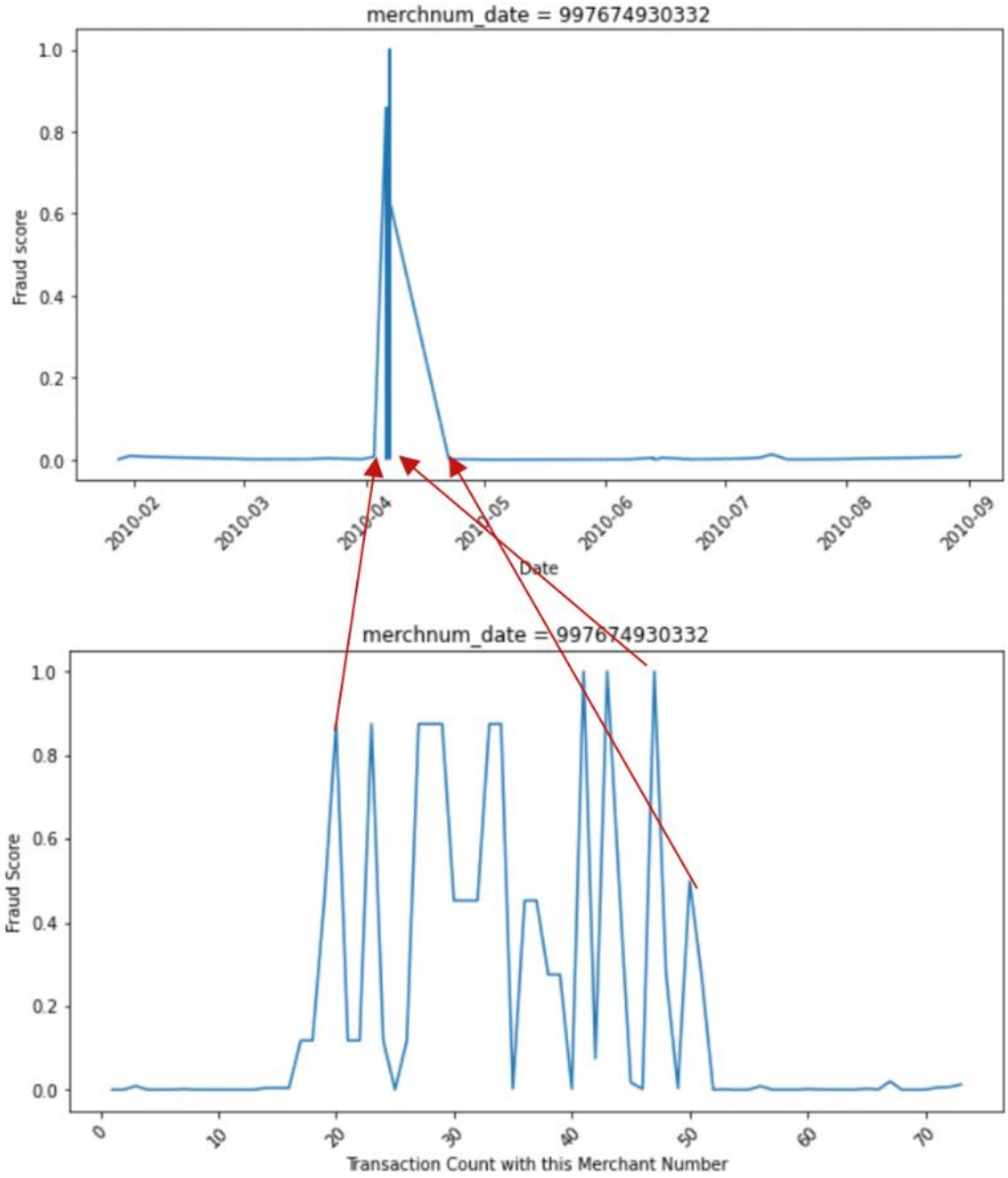

Figure 6.1.2: Fraud score analysis for Merchnum 997674930332 over time and count

In addition to card numbers, we also explored the patterns of merchant numbers by doing the same analysis as card numbers. As illustrated by two graphs in Figure 6.1.2, there are 73 transactions associated with merchant number 997674930332, where 3 transactions that have fraud scores of more than 0.9 are highly likely to be fraudulent. By tracking the same merchant number over time, we found that the 3 transactions happened on the same day (04/07). It's worth noting that 49 transactions happened within 2 months (03/01-05/01), leading to a rise in fraud score during this time period. In conclusion, identity fraud usually happens when a high frequency of transactions appears within a very short time period.

6.2 Fraud Saving Analysis

With our final model, we created a fraud-saving graph to illustrate the cost-saving impact if our model is implemented. For this graph, we assume that we save $2000 for each fraud transaction caught and lose $50 for declining each transaction. The graph's data is based on the result from implementing our final model on the out-of-time dataset. The OOT Population % bins are sorted by the highest to the lowest fraud score computed by our model.

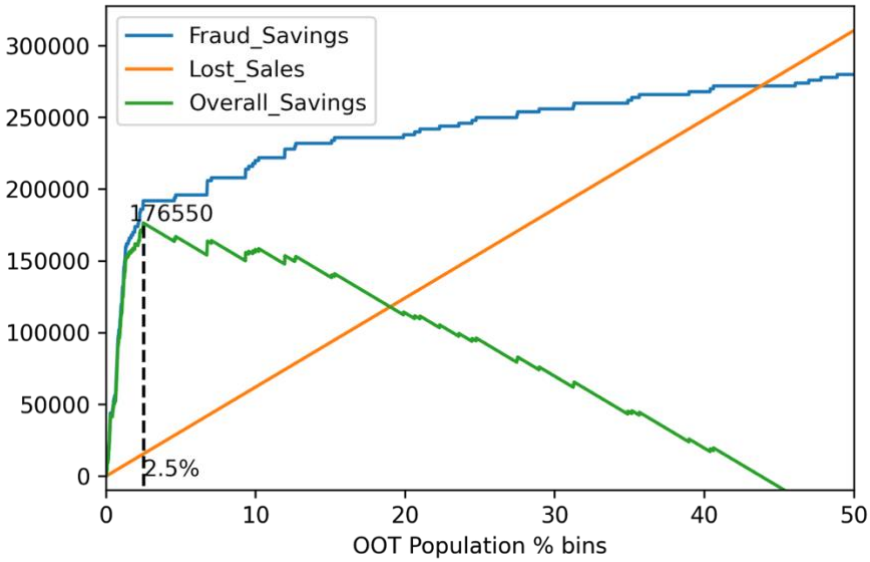

Figure 6.2: Fraud Saving Graph

From the graph, we can see that we will save the most by declining the top 2.5% of the population. Based on our assumption, we could save $176,550 from a total of 12,427 out-of-time transactions. Therefore, declining the top 2.5% of the most likely fraud transaction if our recommended cutoff.

7 Conclusions

In this project, we created a machine learning model to identify fraudulent credit card transactions. We first conducted Exploratory Data Analysis (EDA) to create a Data Quality Report (DQR). Then, we cleaned the data and filled missing values by comparing and choosing the optimal imputing method. After that, we created around 864 features based on expert recommendations and research. Subsequently, we reduced the number of features in our data to 10 features using Kolmogorov-Smirnov (K.S.) filter and Recursive Feature Elimination (RFE). Next, we built a series of models using five different machine learning algorithms and tuning the hyperparameters of these algorithms. Finally, we decided to use the Multi-layer Perceptron classifier, a Neural Net algorithm, to be the algorithm of our final model based on the fraud detection rate on the out-of-time dataset. Our final model reached a Fraud Detection Rate of 58.1% at 3% of the population, which means it can detect more than half of the fraud attempts given only the top 3% data sorted as suspicious by the fraud algorithm score.

The resulting model can be utilized in a credit card fraud transaction detection system. This system could help banks and credit card companies calculate each transaction's fraud score and decline the transaction based on a cutoff that they determined. Traits of fraud transactions evolve over time. Therefore, the companies should keep track of recent fraud transaction trends and evaluate the model frequently. If necessary, they should take action to update the model and adjust the cutoff. Eventually, by implementing this system and catching more fraud transactions, the companies can save money in the long term.

Every project has a time constraint. Here, we will provide some suggestions of improvement that we would have made if there is additional time:

1. Interview more experts to create more expert features
2. In features selection, test more features against the wrapper methodology
3. Include more varieties and combinations of hyperparameters in tuning our model
4. If possible, get extra data

References

Figure 5.2.1
https://www.saedsayad.com/logistic_regression.htm

Figure 5.2.2
https://www.researchgate.net/figure/Architecture-of-the-random-forest-model_fig1_301638643

Figure 5.2.3
https://medium.com/analytics-vidhya/gradient-boosting-lightgbm-xgboost-and-catboost-kaggle-challenge-santander-f3cf0cc56898

Appendix A: Data Quality Report

File Description

The "applications data.csv" is a computer-generated dataset creating the same statistical properties as accurate application data. It stores 1,000,000 records of U.S. applications such as opening a credit card or a cell phone account. It covers ten fields, including application date, Social Security Number (SSN), applicant's name, address, zip code, date of birth, home phone number, and a label representing whether the application is a fraud or not. The dataset came from an identity fraud prevention company and was shared by Professor Stephen Coggeshall in February 2021.

Table 8.0: File Description

Dataset Name	Card Transaction Dataset
Dataset Purpose	Combined actual credit card purchases with manipulated fraud labels to serve as the raw data for building machine learning models to find the frauds
Data Source	Came from a U.S. government organization
Time Period	From January 1, 2010 to December 31, 2010
# of Fields	10 fields in total – 7 categorical, 1 text, 1 date, 1 numerical
# of Record	96,753

Summary Statistics Table

All Fields can be treated as categorical except for two dates (date, dob). All ten fields are fully populated. Key statistics of these fields are summarized as follows.

Table 8.1: Summary Statistics of Categorical Fields

Column Name	# of Records	% Populated	# records with value zero	Unique Values	Most Common Field Value	Data Type
Recnum*	96,753	100	0	96,753	N/A**	int64
Cardnum*	96,753	100	0	1,645	5142148452	int64
Merchnum*	93,378	96.51	231	13,091	930090121224	object
Merch Descirption	96,753	100	0	13,126	GSA-FSS-ADV	text
Merch state	95,558	98.76	0	227	TN	object
Merch zip*	92,097	95.19	0	4,567	38118	float64
Transtype	96,753	100	0	4	P	object
Fraud*	96,753	100	95,694	2	0	int64

Table 8.2: Summary Statistics of Numerical Fields

Column Name	# of Records	% Populated	Unique Values	Most Common Field Value	Minimum Value	Maximum Value	Mean	Data Type
Amount	96,753	100	34,909	3.62	0.01	3102045.53	427.89	float64

Table 8.3: Summary Statistics of Date Fields

Column Name	# of Records	% Populated	Unique Values	Most Common Field Value	Minimum Value	Maximum Value	Data Type
Date	96,753	100	365	2010/2/28 0:00	2010/1/1 0:00	2010/12/31 0:00	datetime64[ns]

* These fields should be categorical but are stored as integers/floats/object in the dataset.

** All values in the *recnum* field are unique, thus not such a most common field value.

Field Description and Distribution

Field 1:

Name	**recnum**
Description	Unique identifier of each entry in the data
Type	Categorical
Distribution	100% populated with 96,753 unique values.

Field 2:

Name	**cardnum**
Description	Credit card number of the transaction
Type	Categorical
Distribution	100% populated with 1,645 unique values. “5142148452” occurred the most for 1192 times.

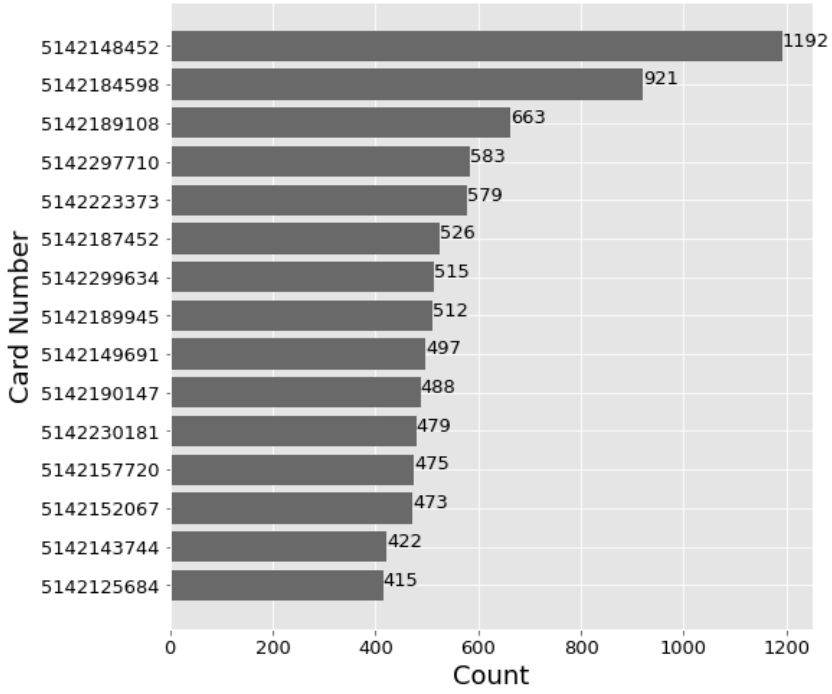

Figure 8.1: Frequency Distribution of the *cardnum* Field
(Top 15 Most Common Values)

Field 3:

Name	date
Description	The date that the payment was made
Type	Date
Distribution	100% populated with 365 unique values. The following plots present the most common date values and compare the distributions of payment counts of fraud and non-fraud transactions each month, each day of the week, and each day.

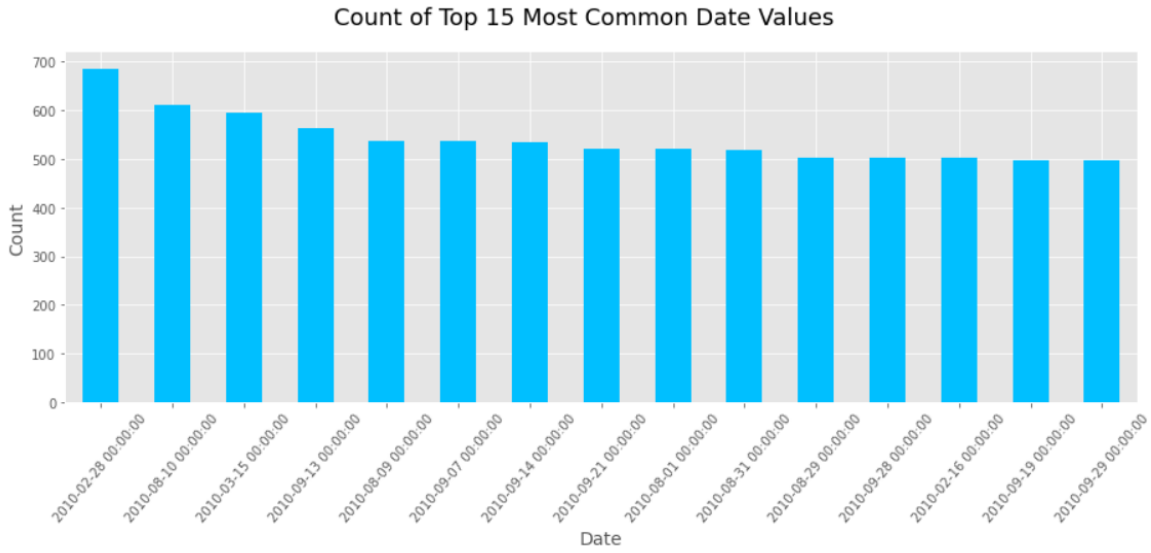

Figure 8.2: Frequency Distribution of the *date* Field
(Top 15 Most Common Values)

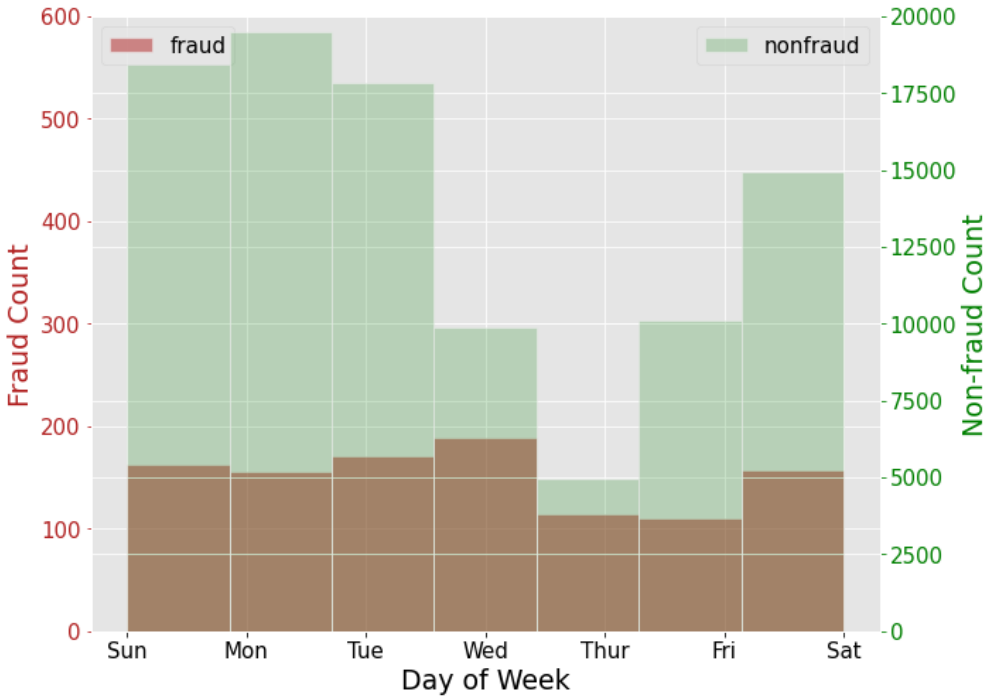

Figure 8.3: Payment Counts by Day of Week

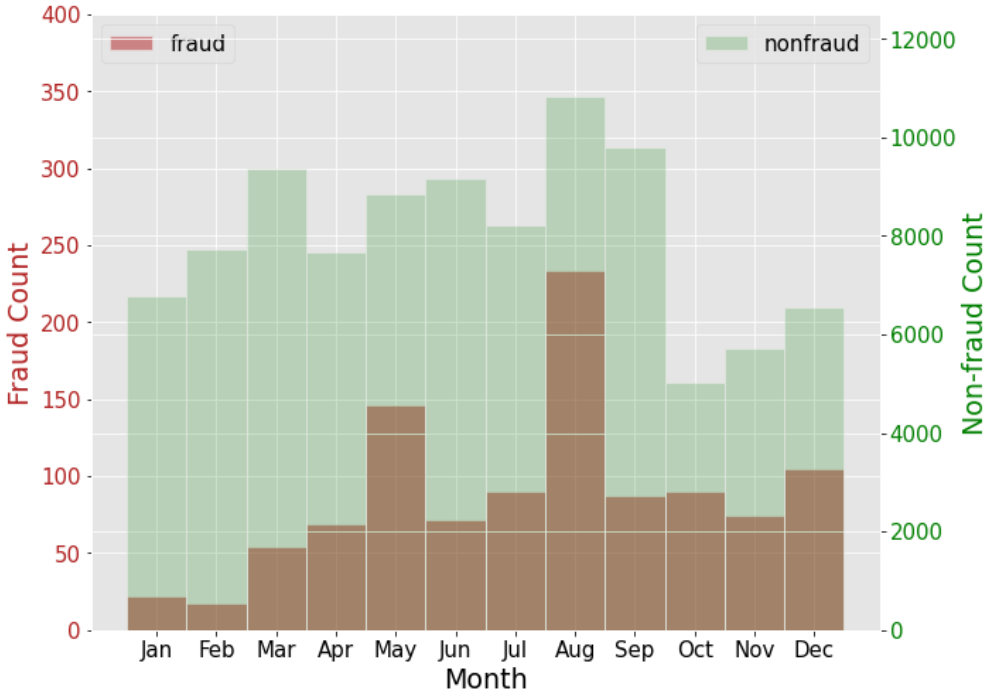

Figure 8.4: Payment Counts by Month

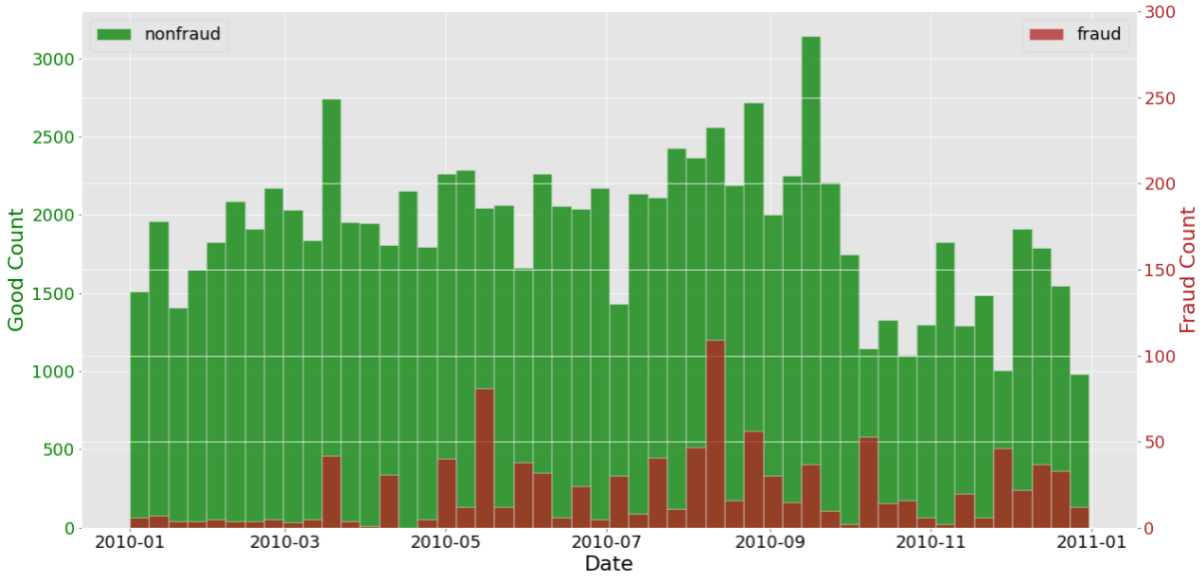

Figure 8.5: Payment Counts by Day

Field 4:

Name	**merchnum**
Description	Unique identification number of the Merchant where the transaction was made at
Type	Categorical
Distribution	96.51% populated with 13,091 unique values and 231 0s. The following table shows the number of times the same valid Merchant number appears, and the plot shows the top 15 most common Merchant numbers.

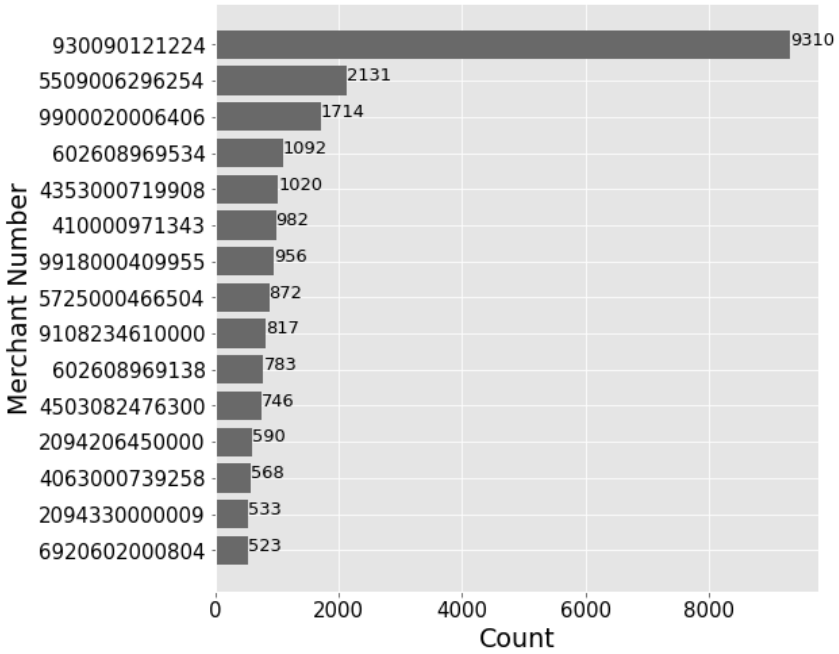

Figure 8.6: Frequency Distribution of the *merchnum* Field
(Top 15 Most Common Values)

Field 5:

Name	**merch description**
Description	The abbreviations of the state where the purchase is made.
Type	Categorical
Distribution	98.76% populated with 227 unique values, 51 of which stand for states in the U.S., 7 stands for states in Canada, 1 value U.S., 167 invalid numeric values, and other missing values. Transactions that take place in Canada do not have any fraud.

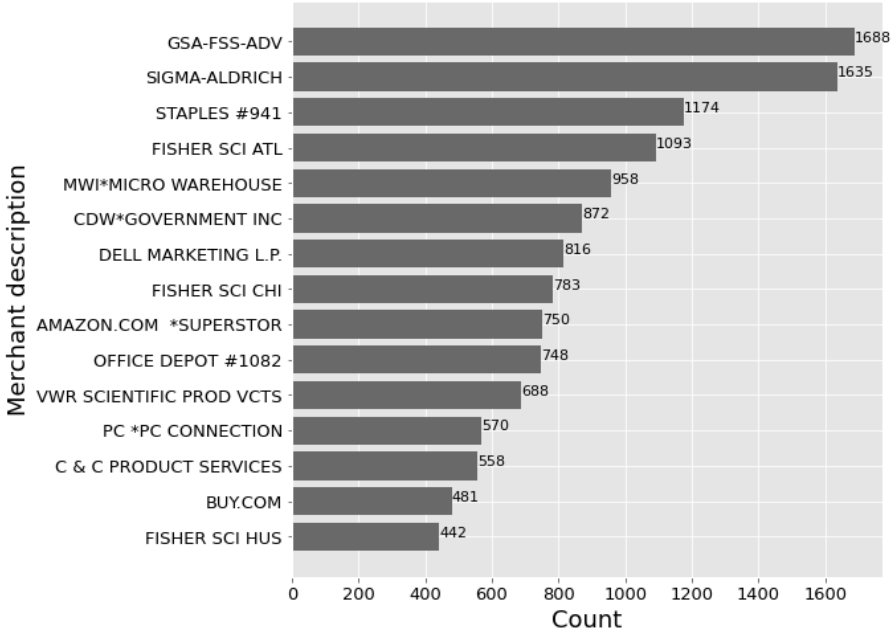

Figure 8.7: Frequency Distribution of the *merch description* Field
(Top 15 Most Common Values)

Field 6:

Description	**The abbreviations of the state where the purchase is made.**
Type	Categorical
Distribution	98.76% populated with 227 unique values, 51 of which stand for states in the U.S., 7 stands for states in Canada, 1 value U.S., 167 invalid numeric values, and other missing values. Transactions that take place in Canada do not have any fraud.

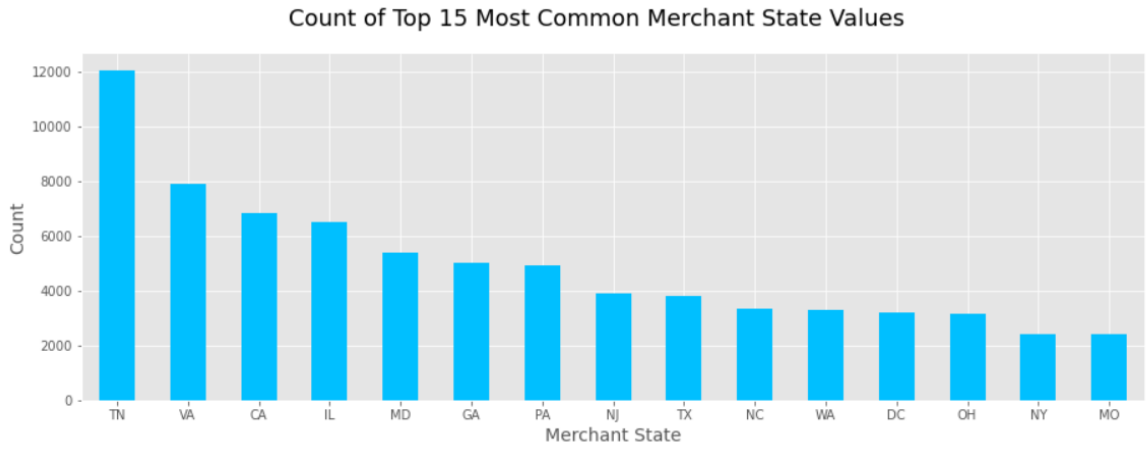

Figure 8.8: Frequency Distribution of the *merch state* Field
(Top 15 Most Common Values)

Figure 8.9: Distribution of Payment Counts by States

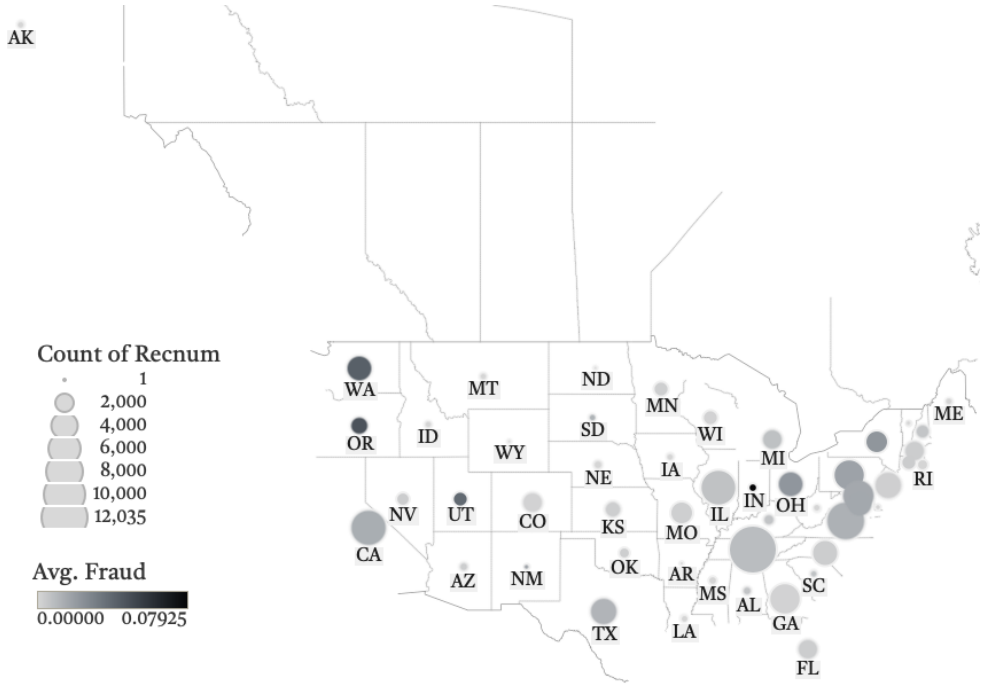

Figure 8.10: Distribution of Payment Counts by States

Field 7:

Name	**merch zip**
Description	5-digit Zip code of the Merchant.
Type	Categorical
Distribution	95.19% populated with 4,567 unique values. Among all the non-missing values, 91.2% (83,998) have valid 5-digit zip codes, 8.6% (7,973) have 4 digits that lack the first 0 digits and could be added later on. Others whose digits are below or equal to 3 are frivolous values. 99999 is also a frivolous value. The following table shows the distribution of valid zip codes, and the plot shows the top 20 zip codes. Although there is one zip code, "38118," which appears 10 times more than other values, it is in T.N., the most frequent state; hence we do not treat it as a frivolous value.

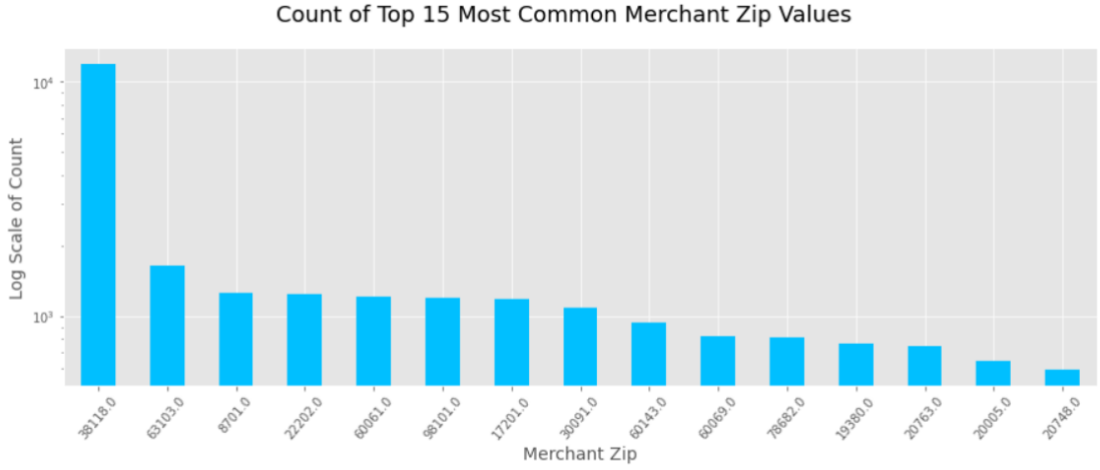

Figure 8.11: Frequency Distribution of the *merch zip* Field
(Top 15 Most Common Values)

Field 8:

Name	**transtype**
Description	Type of the transaction.
Type	Numerical
Distribution	100% populated with 4 unique values (P, A, D, Y), and value P (purchase) accounts for 99.63%. The following plot shows the distribution of these four transaction types.

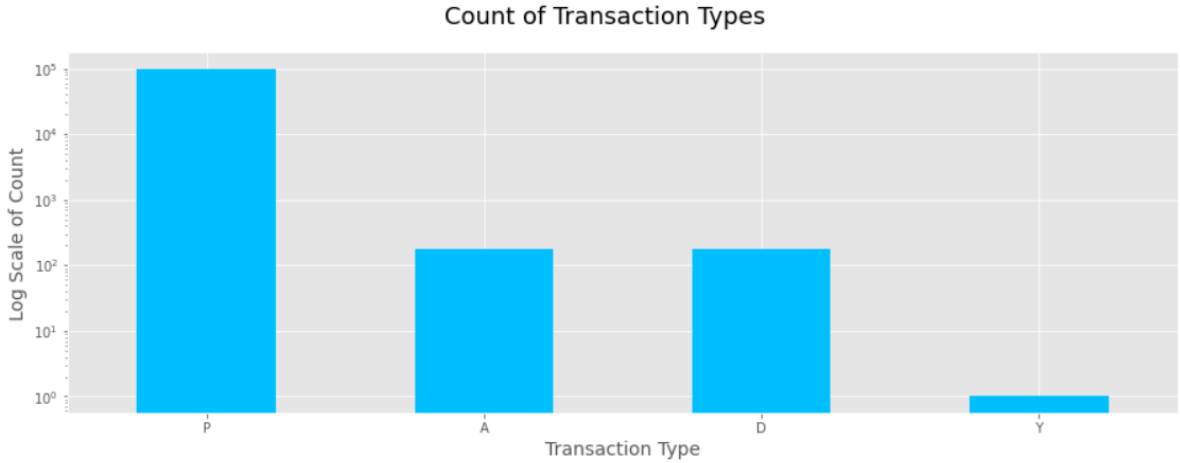

Figure 8.12: Frequency Distribution of the *transtype* Field
(Top 15 Most Common Values)

Field 9:

Description	**The transaction amount of that record.**
Type	Numerical
Distribution	100% populated. The following table shows the distribution of the transaction amounts. Outlier (values that are not within 3 standard deviations of the mean) was removed for graphing purposes.

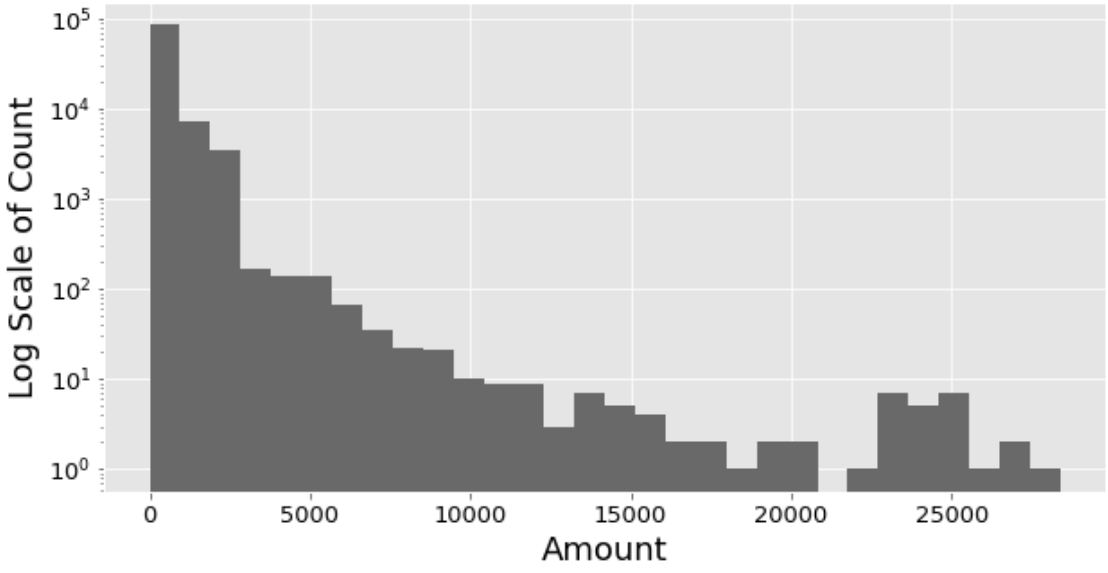

Figure 8.13: Frequency Distribution of the *amount* Field

Field 10:

Name	**fraud**
Description	Labels represent whether the transaction is fraudulent (“1” for Yes, “0” for No).
Type	Categorical
Most common field value	100% populated with 98.91% filled with value 0. The following plot shows the distribution of the fraud label.

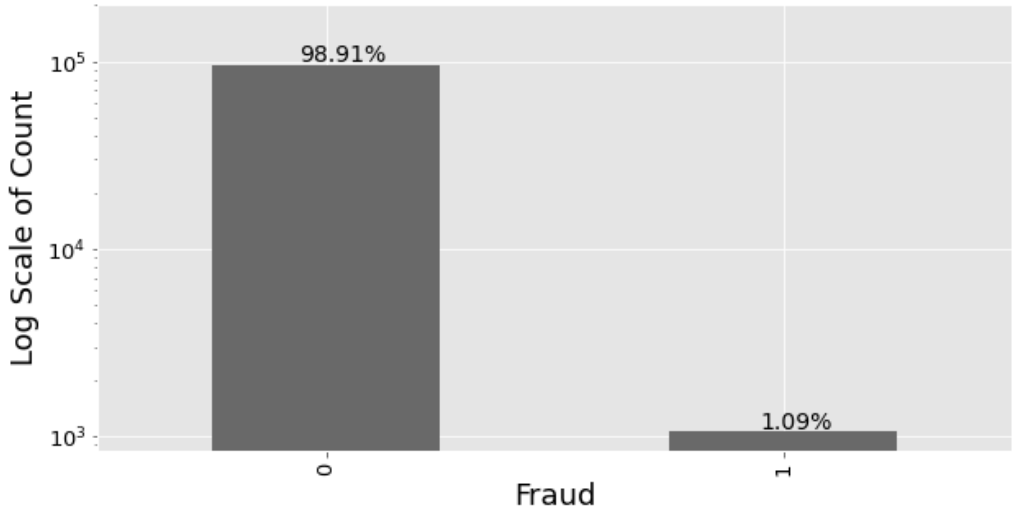

Figure 8.14: Frequency Distribution of the *fraud* Field

Marshall Consulting

Appendix B: 80 Variables Selected after Filter

Variable	KS	FDR @3%	K.S. Rank	FDR Rank	Average Rank
max_Amount_card_description_7	0.6629	0.4286	842	854.5	848.25
max_Amount_card_description_14	0.6664	0.4217	843	850.5	846.75
max_Amount_card_merch_14	0.6618	0.4217	840	850.5	845.25
max_Amount_card_zip_7	0.6599	0.4286	836	854.5	845.25
max_Amount_card_merch_7	0.6570	0.4309	833	856	844.5
max_Amount_card_zip_14	0.6617	0.4171	839	846	842.5
max_Amount_card_description_3	0.6582	0.4205	834	847.5	840.75
max_Amount_card_zip_3	0.6540	0.4217	830	850.5	840.25
max_Amount_card_state_7	0.6482	0.4217	825	850.5	837.75
max_Amount_card_merch_3	0.6519	0.4205	827	847.5	837.25
total_Amount_card_description_14	0.6889	0.3456	857	817	837
max_Amount_card_zip_30	0.6542	0.4101	831	842	836.5
max_Amount_card_description_30	0.6607	0.4067	837	836	836.5
max_Amount_card_state_3	0.6530	0.4113	828	843.5	835.75
max_Amount_card_merch_30	0.6566	0.4078	832	838.5	835.25

total_Amount_card_description_30	0.6870	0.3399	853	814	833.5
total_Amount_card_description_7	0.6701	0.3445	847	816	831.5
max_Amount_card_description_1	0.6419	0.4078	823	838.5	830.75
max_Amount_card_zip_1	0.6391	0.4090	820	841	830.5
max_Amount_card_state_1	0.6374	0.4078	819	838.5	828.75
max_Amount_card_merch_1	0.6342	0.4078	818	838.5	828.25
total_Amount_card_state_1	0.6670	0.3249	844	811	827.5
total_Amount_Cardnum_7	0.6032	0.4147	809	845	827
total_Amount_card_state_3	0.6841	0.2995	851	801	826
total_Amount_card_description_3	0.6612	0.3306	838	812.5	825.25
total_Amount_card_zip_3	0.6737	0.2995	849	801	825
total_Amount_card_state_14	0.6908	0.2938	858	791	824.5
total_Amount_card_zip_30	0.6875	0.2949	854	793.5	823.75
max_Amount_card_state_0	0.6225	0.3998	813	834.5	823.75
max_Amount_card_zip_0	0.6264	0.3952	815	831.5	823.25
max_Amount_card_state_14	0.6340	0.3882	817	829	823
total_Amount_card_merch_3	0.6684	0.2995	845	801	823

max_Amount_card_description_0	0.6264	0.3952	814	831.5	822.75
total_Amount_Cardnum_3	0.5889	0.4470	788	857.5	822.75
max_Amount_card_merch_0	0.6217	0.3952	812	831.5	821.75
total_Amount_card_merch_14	0.6880	0.2915	855	788.5	821.75
total_Amount_card_merch_30	0.6815	0.2949	850	793.5	821.75
total_Amount_card_state_7	0.6888	0.2892	856	787	821.5
total_Amount_card_zip_14	0.6869	0.2915	852	788.5	820.25
total_Amount_card_zip_1	0.6594	0.2995	835	801	818
total_Amount_card_description_1	0.6449	0.3226	824	809.5	816.75
total_Amount_card_merch_1	0.6536	0.2995	829	801	815
total_Amount_Cardnum_1	0.5763	0.4470	771	857.5	814.25
total_Amount_card_zip_7	0.6714	0.2788	848	779.5	813.75
total_Amount_card_state_30	0.6628	0.2880	841	785.5	813.25
total_Amount_card_merch_7	0.6698	0.2788	846	779.5	812.75
total_Amount_Cardnum_0	0.5795	0.4240	772	853	812.5
total_Amount_card_description_0	0.6319	0.3099	816	806	811
total_Amount_card_state_0	0.6418	0.2984	822	797.5	809.75

total_Amount_Cardnum_14	0.5804	0.4113	775	843.5	809.25
max_Amount_Merch description_0	0.5982	0.3537	799	819	809
total_Amount_card_zip_0	0.6484	0.2938	826	791	808.5
max_Amount_merch_state_0	0.5928	0.3537	793	819	806
total_Amount_card_merch_0	0.6413	0.2938	821	791	806
max_Amount_card_state_30	0.5994	0.3203	802	808	805
max_Amount_Merch_0	0.5903	0.3548	789	821	805
max_Amount_merch_zip_0	0.5835	0.3537	783	819	801
max_Amount_Merchnum_0	0.5859	0.3306	786	812.5	799.25
max_Amount_Cardnum_0	0.5801	0.3744	774	824	799
max_Amount_Merch_1	0.5826	0.3226	781	809.5	795.25
avg_Amount_card_state_3	0.6000	0.2788	804	779.5	791.75
avg_Amount_card_state_7	0.5942	0.2800	794	782.5	788.25
max_Amount_Merch description_1	0.5826	0.2961	780	795	787.5
avg_Amount_card_state_1	0.5922	0.2800	792	782.5	787.25
max_Amount_merch_state_1	0.5736	0.3007	768	804	786
avg_Amount_card_description_3	0.6007	0.2765	806	759.5	782.75

avg_Amount_card_zip_1	0.5917	0.2776	791	772	781.5
avg_Amount_card_merch_3	0.5998	0.2765	803	759.5	781.25
max_Amount_Cardnum_3	0.5603	0.3606	740	822	781
avg_Amount_card_zip_3	0.5992	0.2765	801	759.5	780.25
max_Amount_merch_zip_1	0.5719	0.2972	764	796	780
avg_Amount_card_merch_0	0.5868	0.2776	787	772	779.5
avg_Amount_card_zip_7	0.5979	0.2765	798	759.5	778.75
max_Amount_state_zip_0	0.5661	0.3030	752	805	778.5
avg_Amount_card_zip_0	0.5854	0.2776	785	772	778.5
avg_Amount_card_description_0	0.5844	0.2776	784	772	778
avg_Amount_card_description_30	0.6064	0.2753	810	745.5	777.75
avg_Amount_card_description_1	0.5943	0.2765	795	759.5	777.25
avg_Amount_Cardnum_1	0.5759	0.2834	770	784	777
avg_Amount_card_merch_30	0.6032	0.2753	808	745.5	776.75