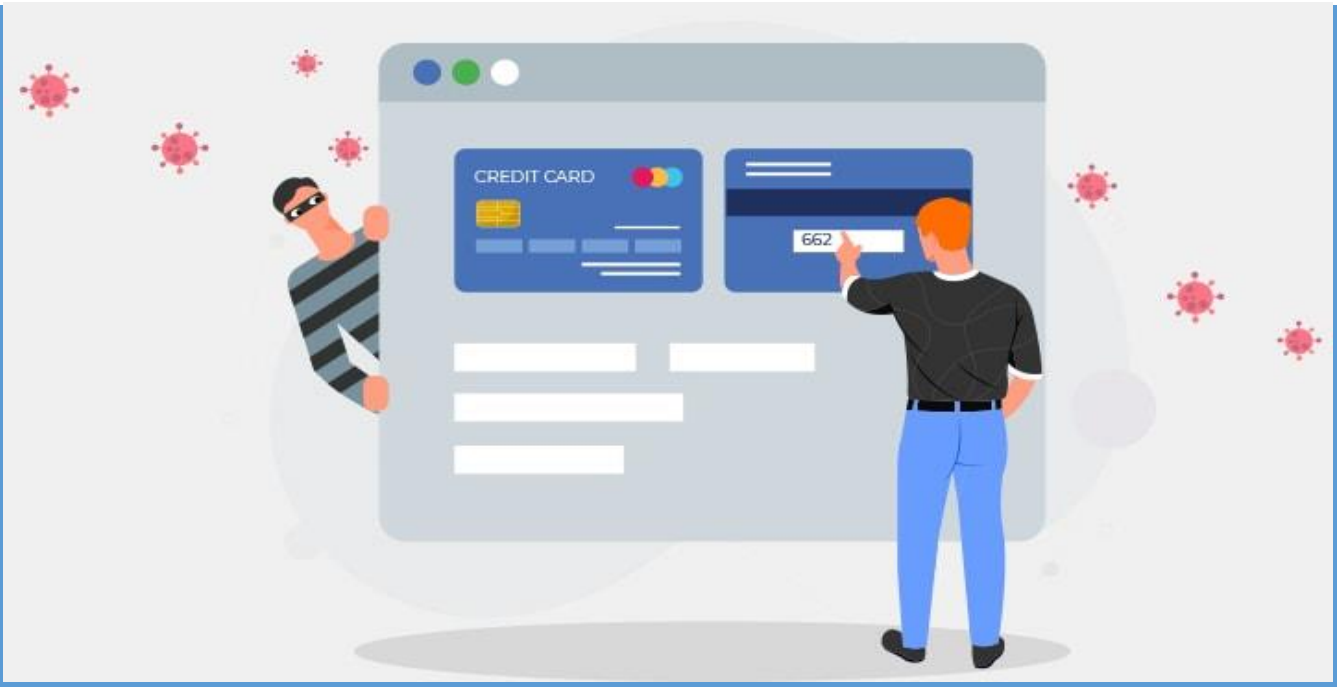
Marshall Consulting

IDENTITY FRAUD IDENTIFICATION



Wenyi Li, MSBA

Yue Li, MSBA

Xinyi Ou, MSBA

Yingying Qian, MSBA

Lingyi Xu, MSBA

Runfeng Zhang, MSBA

Date: March 25th, 2021

Professor Stephen Coggeshall
Project Advisor

Table of Content

Executive Summary

Identity fraud or identity theft is a common type of fraud where a suspect deliberately uses another person's personal identifying information without authorization or creates synthetic identity information in some way to conduct fraud or deception activities in favor of unlawful benefits. This type of fraud usually happens in the financial world, in particular, the banking industry.

This project will be focusing on identifying and predicting fraud applications in the identification application process by building a data-driven algorithm. The overall goal of the project is to find the most effective and efficient statistical analysis model to predict and identity fraud applications.

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 application fraud identification process

Our team has utilized many different algorithms to fit the data, including Logistic Regression, Random Forest, XGBoost, Light GBM, and Neural Network. 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 LightGBM Classifier *(n_estimators=600, max_depth=5, min_child_samples, num_leaves=20, subsample=0.6, colsample_bytree=0.8, learning_rate=0.05)* was selected as our best and final model. The FDR scores achieved by this model on training, testing, and OOT data are 56.17%, 55.61%, and 53.55%. The key statistics of the top 20 bins also show that our final model has a good performance in detecting fraud.

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	Application Data
Dataset Purpose	Generated to mimic actual application data and find application/identity fraud
Data Source	Came from an identity fraud prevention company
Time Period	From January 1, 2016 to December 31, 2016
# of Fields	10 fields in total – 8 categorical, 2 date
# of Record	1,000,000

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

Field Name	**Field Type**	**# records that have a value**	**% populated**	**# records with value zero**	**# unique values**	**Most common field value**
record	categorical*	1,000,000	100	0	1,000,000	N/A**
ssn	categorical*	1,000,000	100	0	835,819	999999999
firstname	categorical	1,000,000	100	0	78,136	EAMSTRMT
lastname	categorical	1,000,000	100	0	177,001	ERJSAXA
address	categorical	1,000,000	100	0	828,774	123 MAIN ST
zip5	categorical*	1,000,000	100	0	26,370	68138
homephone	categorical*	1,000,000	100	0	28,244	9999999999
fraud_label	categorical*	1,000,000	100	0	2	0

Table 1.2.2: Summary Statistics of Date Fields

Field Name	Field Type	# records that have a value	% populated	# records with value zero	# unique values	Most common field value	Min	Max
date	date*	1,000,000	100	0	365	20160816	20160101	20161231
dob	date*	1,000,000	100	0	42,673	19070626	19000101	20161031

* These fields should be of the categorical/date type but are stored as integers in the dataset.

** All values in the *record* field are unique, thus no such a most common field value.

1.3 Field Examples

1.3.1 Field "SSN"

Table 1.3.1: SSN

Description	Applicant's social security number.
Type	Categorical
Most Common Field Value	"999999999" occurred the most for 16935 times and is a frivolous value.

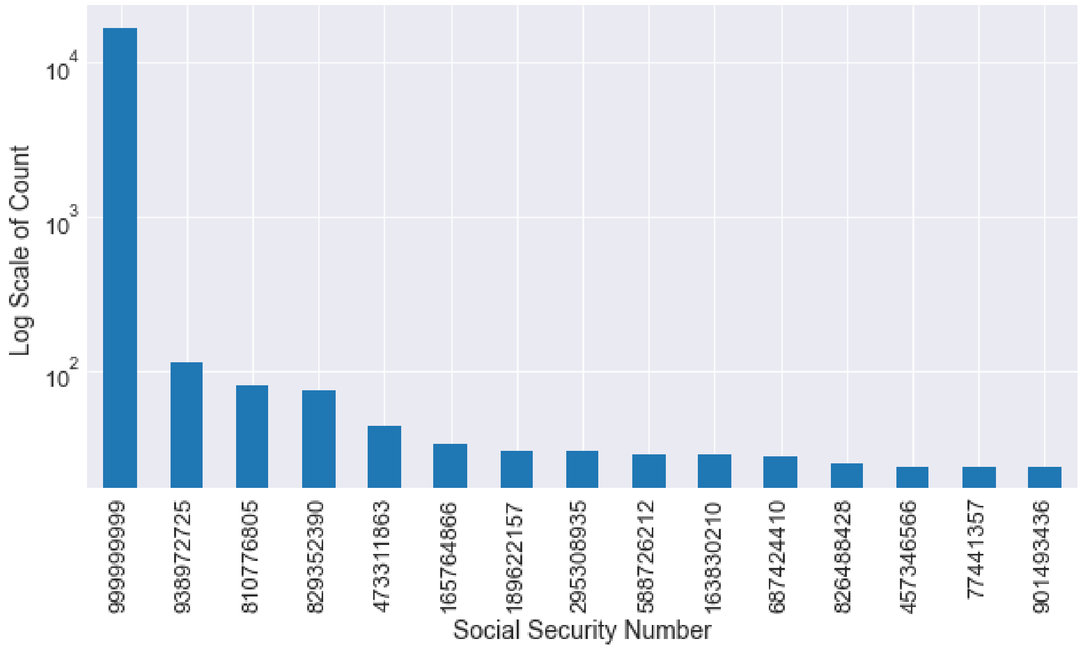


Figure 1.3.1: Frequency Distribution of the *ssn* Field
(Top 15 Most Common Values)

1.3.2 Field “address”

Table 1.3.2: address

Description	Applicant’s home address.
Type	Categorical
Most Common Field Value	“123 MAIN ST” occurred the most for 1079 times and is a frivolous value.

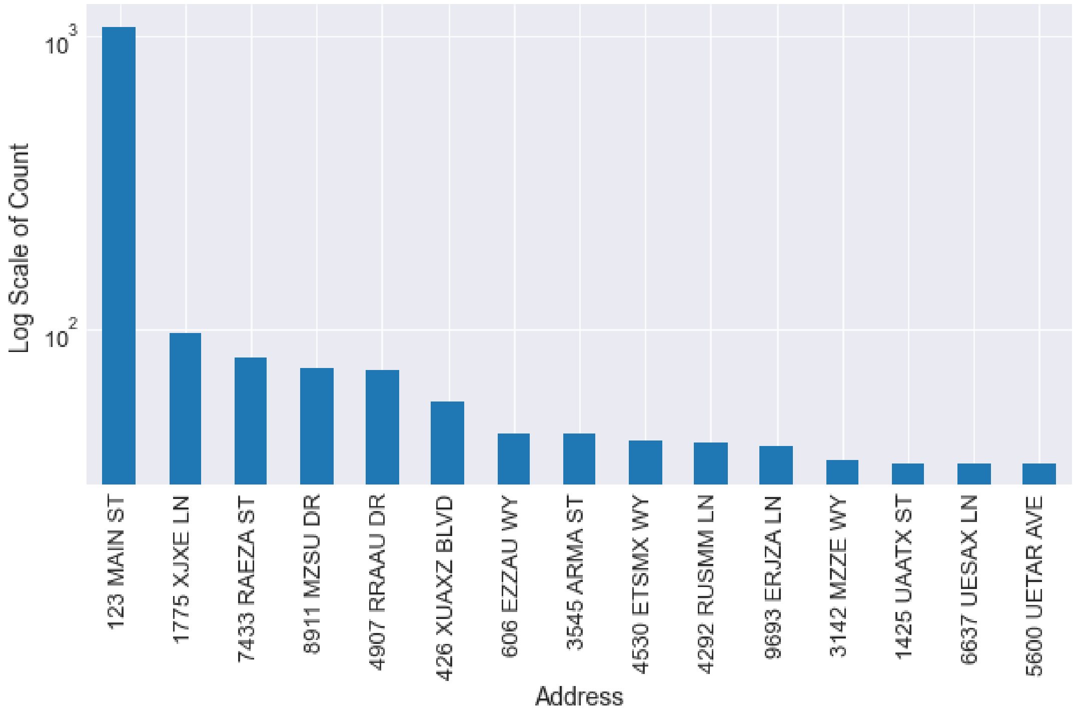


Figure 1.3.2: Frequency Distribution of the *address* Field
(Top 15 Most Common Values)

1.3.3 Field “dob”

Table 1.3.3: dob

Description	Applicant’s date of birth.
Type	Date
MIN	1900-01-01
MAX	2016-10-31
Most Common Field Value	“1907-06-26” occurred the most for 126568 times and is a frivolous value.

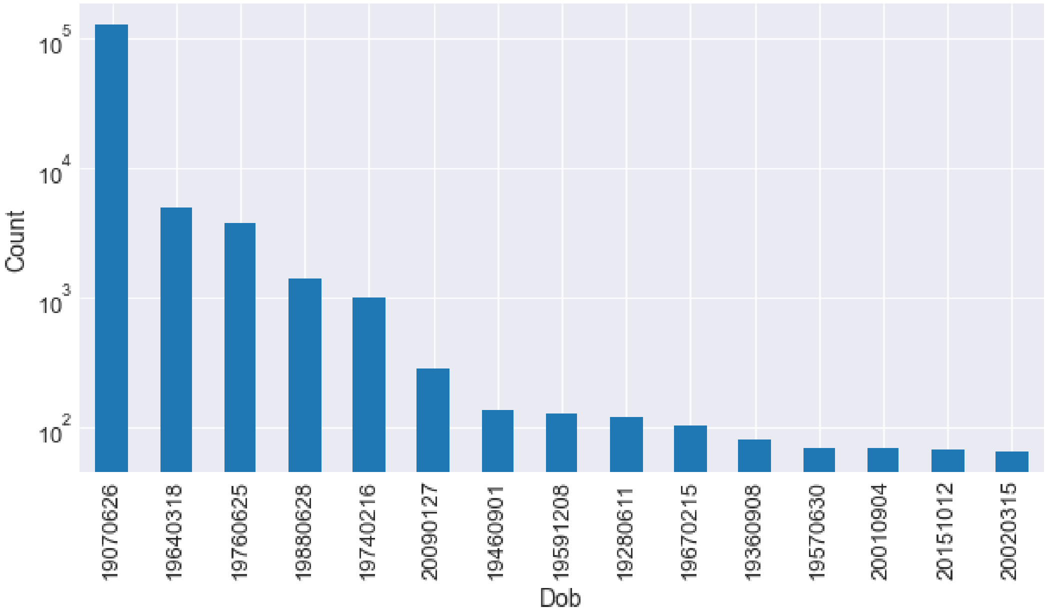


Figure 1.3.3.1: Frequency Distribution of the *dob* Field
(Top 15 Most Common Values)

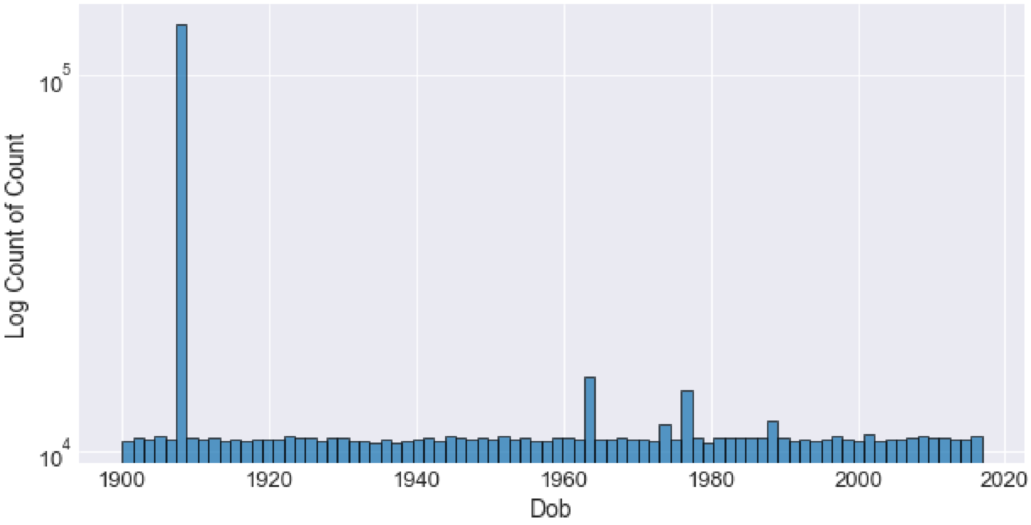


Figure 1.3.3.2: Frequency Distribution of the *dob* Field
(Sort by Date)

1.3.4 Field “homephone”

Table 1.3.4: homephone

Description	Applicant’s home phone.
Type	Categorical
Most Common Field Value	“9999999999” occurred the most for 78512 times and is a frivolous value.

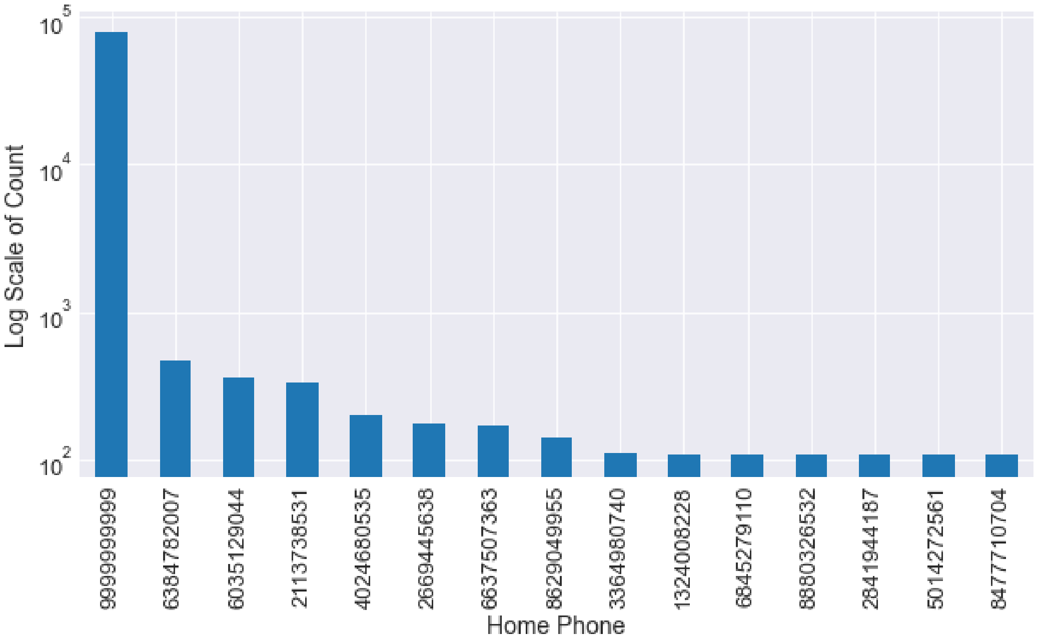


Figure 1.3.4: Frequency Distribution of the *homephone* Field
(Top 15 Most Common Values)

1.3.5 Field "fraud_label"

Table 1.3.5: fraud_label

Description	Whether the application is fraudulent ("1" for Yes, "0" for No)
Type	Categorical
Most common field value	"0" occurred the most for 985607 (98.6%) times.

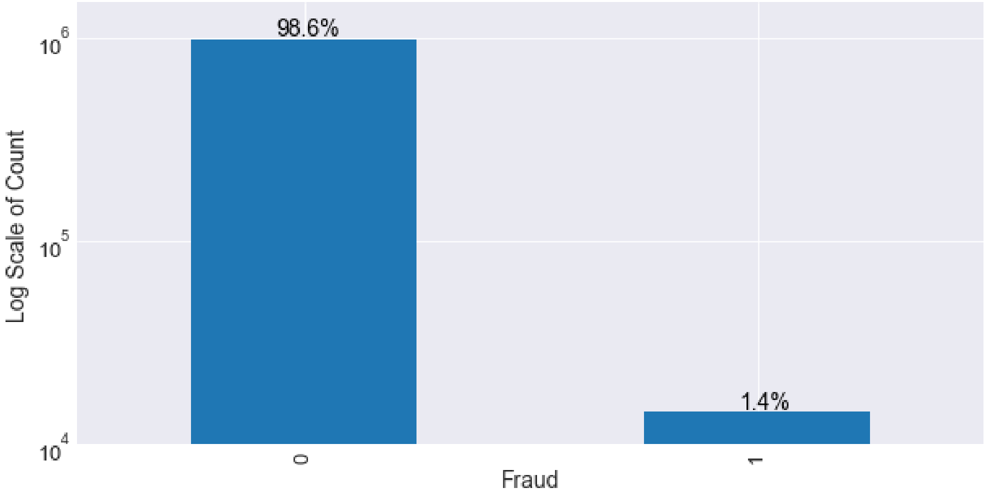


Figure 1.3.5: Frequency Distribution of the *fraud_label* Field

2 Data Cleaning

The core solution path for this identity fraud detection problem is to link applications that use the same or similar identity information. However, during the data exploration stage, we found four fields with frivolous values that are obviously placeholders for fields not collected. Consequently, these values will mistakenly connect unrelated applications and affect model effectiveness. To solve this problem, we need to replace these frivolous values with values that won't link unrelated applications. Therefore, a negative record number is used. Table 2.1 below shows how we replaced those frivolous values.

Table 2.1: Frivolous Values

Field	**Frivolous Value**	**Replaced With**
ssn	999999999	Negative record number
address	123 MAIN ST	Negative record number + "Record"
dob	19070626	Negative record number
homephone	9999999999	Negative record number

3 Feature Creation

A total of 517 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 517 Candidate Variables

Category	Description	# of Variables Created
Weekday Risk	Average fraud likelihood of each weekday	1
Days Since Last Seen	Number of days since the last appearance of the same PII or PII combination	36
Velocity	Number of records of the same PII or PII combination showed over a specific time window before	216
Relevant Velocity	Measures if there is a sudden increase of applications of the same PII or PII combination in a short period relative to a long-term frequency	144
Cross Entity Velocity	Number of unique PII or PII combination that appeared for a particular PII or PII combination over a specific time window	120

3.1 Weekday Risk

Taking into account of 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 Probability	0.0135	0.0141	0.0152	0.0150	0.0145	0.0150	0.0137

3.2 Days Since Last Seen

Since the goal of this project is to find identity fraud which is the act of misrepresenting which person you are, there are mainly two forms that an application seems suspicious: 1) one PII occurred many times with other different PIIs frequently; 2) different PIIs occurred with one same PII in short periods. Therefore, the frequency of a PII or PII combination being used is an essential indicator 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, 2016/1/1, to be specific, as its value. Figure 3.1 shows how the total 36 relevant variables are built from the existing PII and PII combinations.

# of days since the last appearance of

↓

PII	Description
‘ssn’	Social Security Number
‘address’	Address at street level
‘dob’	Date of birth
‘homephone’	Home phone number
‘name’	First name and last name
‘fulladdress’	Combination of ‘address’ and 5-digit zip code

+

PII Combination	
‘name_dob’	‘lastname_zip5’
‘ssn_firstname’	‘lastname_dob’
‘ssn_lastname’	‘lastname_homephone’
‘ssn_address’	‘lastname_fulladdress’
‘ssn_zip5’	‘address_dob’
‘ssn_dob’	‘address_homephone’
‘ssn_homephone’	‘address_name_dob’
‘ssn_name_dob’	‘zip5_dob’
‘ssn_fulladdress’	‘zip5_homephone’
‘firstname_address’	‘zip5_name_dob’
‘firstname_zip5’	‘dob_homephone’
’firstname_dob’	‘dob_fulladdress’
‘firstname_homephone’	‘homephone_fulladdress’
‘firstname_fulladdress’	‘homephone_name_dob’
‘lastname_address’	‘name_dob_fulladdress’

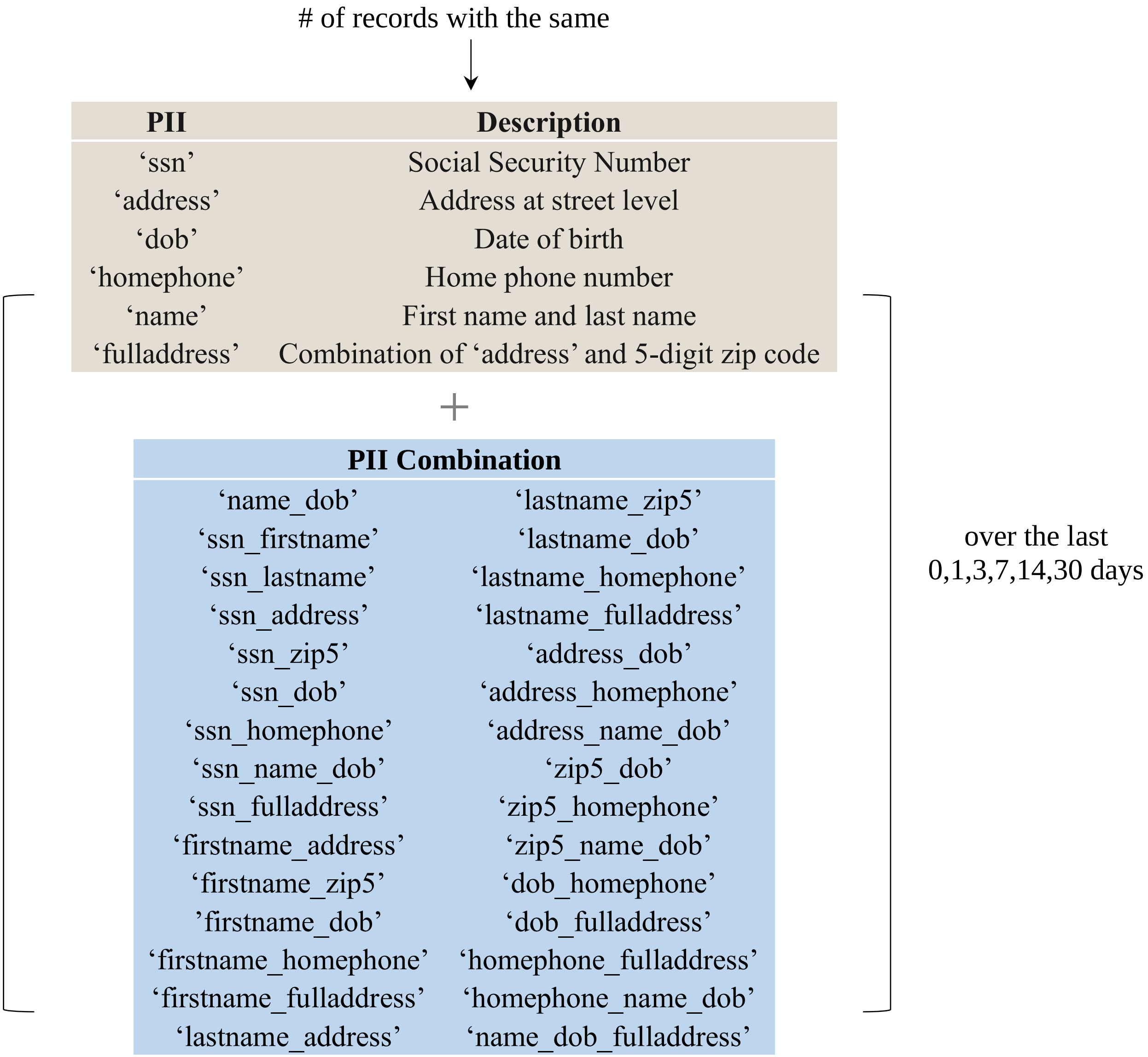
Figure 3.2: Creation of 36 *days since last seen* Variables

$* \ 6 + 30 = 36$ *days since last seen* variables

It’s worth noting that the records of the first couple weeks of 2016 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 2016/1/1, 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 bigger 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 application. Figure 3.3 shows how the total 216 relevant variables are built from the existing PII and PII combinations.



PII	Description
‘ssn’	Social Security Number
‘address’	Address at street level
‘dob’	Date of birth
‘homephone’	Home phone number
‘name’	First name and last name
‘fulladdress’	Combination of ‘address’ and 5-digit zip code

PII Combination	
‘name_dob’	‘lastname_zip5’
‘ssn_firstname’	‘lastname_dob’
‘ssn_lastname’	‘lastname_homephone’
‘ssn_address’	‘lastname_fulladdress’
‘ssn_zip5’	‘address_dob’
‘ssn_dob’	‘address_homephone’
‘ssn_homephone’	‘address_name_dob’
‘ssn_name_dob’	‘zip5_dob’
‘ssn_fulladdress’	‘zip5_homephone’
‘firstname_address’	‘zip5_name_dob’
‘firstname_zip5’	‘dob_homephone’
’firstname_dob’	‘dob_fulladdress’
‘firstname_homephone’	‘homephone_fulladdress’
‘firstname_fulladdress’	‘homephone_name_dob’
‘lastname_address’	‘name_dob_fulladdress’

Figure 3.3: Creation of 216 *velocity* Variables

$* \ (6 + 30) \times 6 = 216$ *velocity* variables

3.4 Relative Velocity

Based on the velocity variables, relative variables were created to describe further the sudden increase of the same PII applications 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 144 variables were created.

$$\text{Relative Velocity} = \frac{\#\ apps\ with\ that\ group\ seen\ in\ the\ past\ 1\ day}{\#\ apps\ with\ that\ same\ group\ seen\ in\ the\ past\ [3,7,14,30]\ days}$$

$*36 \times 4 = 144\ relative\ velocity$ variables

3.5 Cross Entity Velocity

Creating cross- entity velocity variables could help detect the frequency of PII combinations in a certain time window. For example, suppose for the same ssn, many different name_dob applications were recorded in a short time period. In that case, such applications have a higher possibility of being regarded as identity theft. Therefore, the greater the cross entity velocity variables are, the more likely an application is a potential fraud. If all the PII and PII combinations (36 variables) are selected to cross with, then the majority of cross entity velocity variables will be redundant. Therefore, only a small number of entities are chosen to cross with others. Figure 3.5 shows how a total of 120 cross entity velocity variables were calculated.

$$\text{Number of unique} \begin{bmatrix} 'ssn' \\ 'address' \\ 'dob' \\ 'homephone' \\ 'name \end{bmatrix} \text{for that particular} \begin{bmatrix} 'ssn' \\ 'address' \\ 'dob' \\ 'homephone' \\ 'name \end{bmatrix} \text{over the past} \begin{bmatrix} 0 \\ 1 \\ 3 \\ 7 \\ 14 \\ 30 \end{bmatrix} \text{days}$$

Figure 3.5: Creation of 120 *cross entity velocity* Variables

*A field does not cross with itself
$**5 \times 4 \times 6 = 120$ *cross entity velocity* 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 are using 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 (KS) score, and Fraud Detection Rate (FDR).

For this project, KS and FDR@3% were calculated to rank the 517 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?

KS 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 like in figure 4.3.1. The more different the curves, the better the variable for separating, and thus the more important the variable.

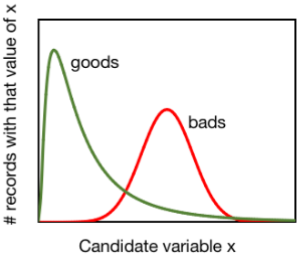


Figure 4.3.1.1: The distribution of goods and bads of candidate variable x

How does KS work?

For each candidate feature, two distributions of fraud (bad) and non-fraud (goods) records are built, respectively, and then the KS 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.

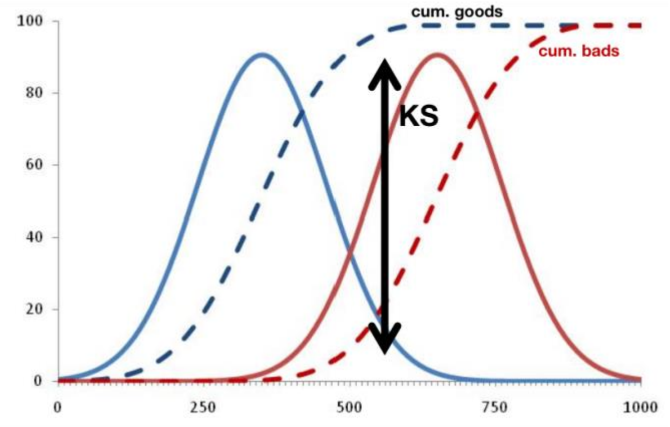


Figure 4.3.1.2: How to plot KS

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 catches 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 exists 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 KS and FDR, and the average ranking is used as 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. The Wrapper methodology instead is based on a specific machine learning algorithm that is 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 backward selection to select 20 to 30 or so variables out of 80 variables. The detailed process of backward selection is as follows:

1. Start with all 80 variables
2. Remove the variables one by one at each step, and ROC-AUC was used as the evaluation criterion. That is, after removing that variable, the remaining variables have the highest ROC-AUC
3. Repeat step 2 until any further removal leads to a dramatic ROC-AUC decrease or the number of remaining variables is under 20

After wrapper, a total of 30 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

Rank	Feature Name	Description
1	address_dob_1_day_cross_count	# different **dob** appeared alongside the given **address** in the past 1 day
1	address_homephone_lag14_count	# appearance with the given **address** + **homephone** combination seen in the past 14 days
1	address_lag14_count	# appearance with the given **address** seen in the past 14 days
1	address_lag1_count	# appearance with the given **address** seen in the past 1 day
1	address_lag1_lag7_avg	$\frac{\text{\# appearance with the given \textbf{address} seen in the past 1 day}}{\text{\# appearance with the same \textbf{address} seen in the past 7 days}}$
1	address_lag30_count	# appearance with the given **address** seen in the past 30 days
1	address_lag7_count	# appearance with the given **address** seen in the past 7 days
1	address_name_14_day_cross_count	# different **name** appeared alongside the given **address** in the past 14 days
1	address_name_30_day_cross_count	# different **name** appeared alongside the given **address** in the past 30 days
1	address_name_7_day_cross_count	# different **name** appeared alongside the given **address** in the past 7 days
1	address_ssn_14_day_cross_count	# different **ssn** appeared alongside the given **address** in the past 14 days
1	address_ssn_30_day_cross_count	# different **ssn** appeared alongside the given **address** in the past 30 days
1	fulladdress_lag1_count	# appearance with the given **full address** seen in the past 1 day
1	fulladdress_lag1_lag7_avg	$\frac{\text{\# appearance with the given \textbf{full address} seen in the past 1 day}}{\text{\# appearance with the same \textbf{full address} seen in the past 7 days}}$
1	fulladdress_lag30_count	# appearance with the given **full address** seen in the past 30 days
1	homephone_fulladdress_lag14_count	# appearance with the given **full address** + **homephone** combination seen in the past 14 days
1	homephone_fulladdress_lag7_count	# appearance with the given **full address** + **homephone** combination seen in the past 7 days
1	homephone_lag3_count	# appearance with the given **homephone** seen in the past 3 days
1	lastname_dob_lag14_count	# appearance with the given **lastname** + **dob** combination seen in the past 14 days
1	lastname_dob_lag30_count	# appearance with the given **lastname** + **dob** combination seen in the past 30 days
1	name_dob_lag14_count	# appearance with the given **name** + **dob** combination seen in the past 14 days
1	name_dob_lag30_count	# appearance with the given **name** + **dob** combination seen in the past 30 days
1	ssn_dob_lag14_count	# appearance with the given **ssn** + **dob** combination seen in the past 14 days
1	ssn_dob_lag30_count	# appearance with the given **ssn** + **dob** combination seen in the past 30 days
1	ssn_firstname_lag30_count	# appearance with the given **ssn** + **firstname** combination seen in the past 30 days
1	ssn_lag30_count	# appearance with the given **ssn** seen in the past 30 days
1	ssn_name_dob_lag14_count	# appearance with the given **ssn** + **name** + **dob** combination seen in the past 14 days
1	ssn_name_dob_lag30_count	# appearance with the given **ssn** + **name** + **dob** combination seen in the past 30 days
1	zip5_homephone_lag14_count	# appearance with the given **zip5** + **homephone** combination seen in the past 14 days
1	zip5_homephone _lag7_count	# appearance with the given **zip5** + **homephone** combination seen in the past 7 days

5 Modeling

To get a more robust model, we first preprocessed the dataset by performing undersampling and capping. Then we compared different algorithms with several combinations of parameters, including Logistic Regression, Random Forest, XGBoost, LightGBM, and Neural Network. To prevent overfitting, 10-fold cross-validation was implemented during this process. Randomness can exist in the training process of some models, such as Random Forest, XGBoost, and LightGBM. Therefore, we fit the same model on the same training set five times and calculate the average FDR at a 3% rejection rate. The average FDR score was then used to compare different models. LightGBM achieved the highest FDR on testing data and was selected as the final model. Details of different models are shown in the model performance table.

Table 5.0: Model Performance Summary

Model			Parameter							Average FDR (%) at 3%		
LogisticRegression	Iterations	Variables Selected	Penalty			C				Train	Test	OOT
	1	10	l2			0.01				35.91	34.16	31.52
	2	15	l2			0.01				35.35	33.69	31.22
	3	20	l2			0.01				52.32	51.27	49.25
	4	25	l2			0.01				52.32	51.32	49.20
	5	30	l2			0.01				52.33	51.35	49.16
RandomForest	Iterations	Variables Selected	n_estimators	max_depth	max_features	min_samples_leaf	min_samples_spli	criterion		Train	Test	OOT
	1	30	50	None	auto	1	2	gini		56.32	56.57	53.49
	2	30	50	50	5	10	5	entropy		56.03	56.46	53.52
	3	30	50	100	auto	1	5	entropy		56.32	56.52	53.57
	4	30	80	20	auto	10	10	gini		56.06	56.51	53.54
	5	30	80	50	5	10	10	entropy		56.08	56.48	53.50
	6	30	100	50	5	1	10	entropy		56.27	56.58	53.60
	7	30	100	None	auto	1	2	gini		56.34	56.46	53.49
	8	30	100	None	5	1	10	gini		56.27	56.56	53.62
XGBoost	Iterations	Variables Selected	eta		max_depth		min_child_weight			Train	Test	OOT
	1	30	0.3		3		1			56.06	56.37	53.45
	2	30	0.3		4		2			56.10	56.41	53.50
	3	30	0.2		4		2			56.07	56.44	53.45
	4	30	0.2		5		2			56.16	56.51	53.55
	5	30	0.1		5		1			56.15	56.44	53.52
	6	30	0.1		5		2			56.17	56.47	53.53
	7	30	0.05		5		1			56.18	56.52	53.55
	8	30	0.05		5		2			56.16	56.45	53.54
LightGBM	Iterations	Variables Selected	subsample	colsample_bytree	n_estimators	min_child_samples	max_depth	num_leaves	learning_rate	Train	Test	OOT
	1	30	1	0.8	500	1	0.8	20	0.1	56.22	56.54	53.55
	2	30	0.6	0.8	500	150	5	20	0.05	56.19	56.55	53.55
	3	30	0.6	0.8	600	200	5	20	0.05	56.17	56.61	53.55
	4	30	0.6	0.8	700	200	5	15	0.05	56.17	56.56	53.57
	5	30	0.6	0.8	800	200	5	15	0.01	56.12	56.50	53.52
	6	30	0.6	0.8	900	200	5	10	0.01	56.08	56.50	53.53
	7	30	0.6	0.8	1000	200	5	10	0.001	55.20	55.53	52.43
NeuralNetwork	Iterations	Variables Selected	Nodes	Layers	Epoch	Activation		Opimizer		Train	Test	OOT
	1	30	50	1	200	relu		adam		56.14	56.49	53.56
	2	30	50	1	200	relu		sgd		54.91	55.31	52.41
	3	30	50	1	400	logistic		adam		56.03	56.29	53.49
	4	30	50	1	400	logistic		sgd		54.45	54.98	51.80
	5	30	100	1	200	relu		adam		56.14	56.44	53.52
	6	30	100	1	200	relu		sgd		54.80	55.20	52.41
	7	30	100	1	400	logistic		adam		56.05	56.36	53.49
	8	30	100	1	400	logistic		sgd		54.17	54.90	51.99

5.1 Data Preprocessing

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

5.1.1 Weighting

Since the dataset is highly imbalanced, with only around 1.4% 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. First, we implemented 10-fold cross-validation on each model. Notice that we did not include the OOT data in cross-validation. Otherwise, it will cause data leakage and overfitting. The general process of k-fold cross-validation is as follows:

1. Randomly shuffle the dataset and then split it into k groups. (We use k=10 here.)
2. For each group, we keep it as the testing dataset. Then we fit the model on the remaining data and evaluate the model on testing data.
3. Check the evaluation score in each of the k rounds and make sure that they do not have great variance. (We use FDR at 3% rejection here.)
4. Calculate the average evaluation score.

Cross-validation 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. It is possible that certain observations 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, it may have significantly worse performance. Therefore, we can use cross-validation to make sure that our model will also have a good performance on new data.

If these 10 FDR scores do not have much variance, then we can be sure that our model is robust and go to the next step. Some algorithms have randomness in their training process. For example, Random Forest and Gradient Boosting Trees may use a randomly selected subsample of rows or columns to train each train. To take this randomness derived from the algorithm itself into account, we fit the model with the same parameters on the same training dataset five times. Then we calculated average FDR scores for training, testing, and OOT dataset.

We performed the steps above on various algorithms with many combinations of parameters, including Logistic Regression, Random Forest, XGBoost, LightGBM, and Neural Network. Eventually, LightGBM had the best performance on the testing 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 certain 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.

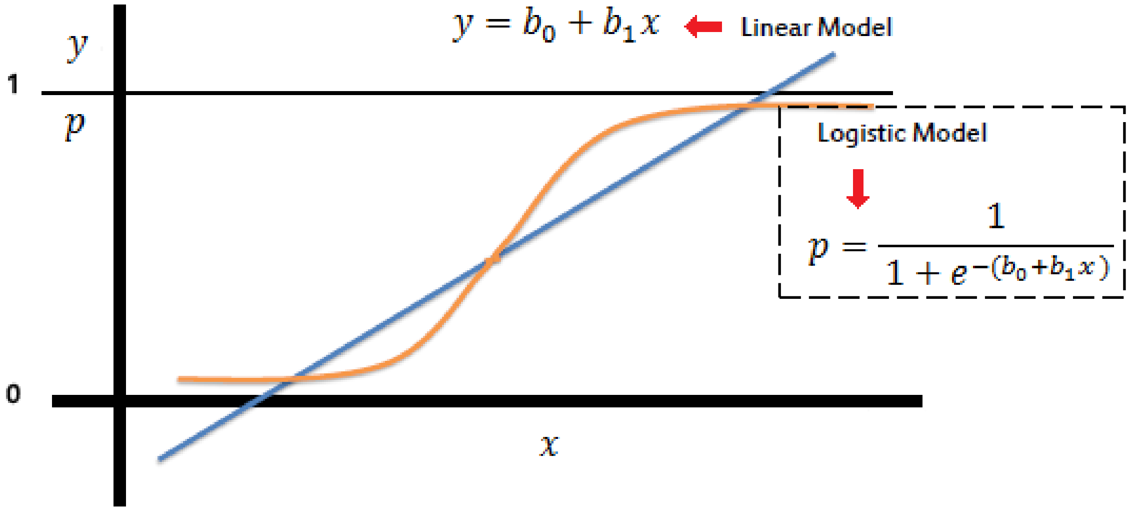


Figure 5.2.1: Logistic Model and Linear Model Illustration

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 application 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 5.2.1.1):

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	Number of CPU cores used when parallelizing over classes if multi_class='ovr'. We used the value "*-1*", which means using all processors.

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.

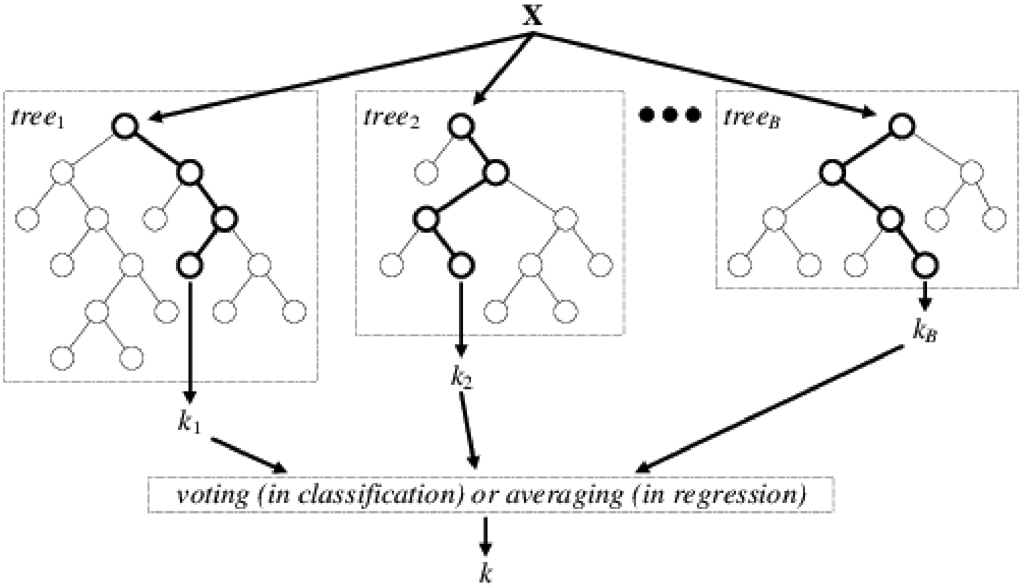


Figure 5.2.2: Random Forest Model Illustration

Given the flexibility available with this model, we tuned six parameters (see table 5.2.2.1):

Package: sklearn.ensemble.RandomForestClassifier

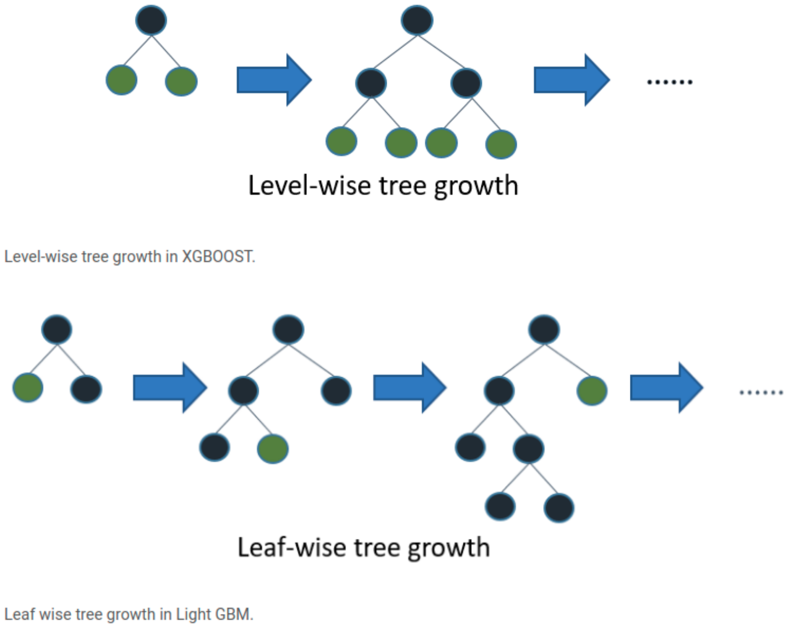
Table 5.2.2: Essential Paramemters in Random Forest

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, which allows LightGBM to reduce more loss. Also, LightGBM is often faster to train. In addition, LightGBM tends to occupy less memory. However, LightGBM’s leaf-wise growth could also lead to more over-fitting in training data. In reality, the accuracy of both models tend to be very similar in most cases. Therefore, we have decided to try both models.



Level-wise tree growth in XGBOOST.

Leaf wise tree growth in Light GBM.

Figure 5.2.3: XGBoost and LightGBM Model Illustration

We have tuned these parameters for XGBoost (see table 5.2.3.1).

Package: xgboost.XGBClassifier

Table 5.2.3: Essential Paramemters in XGBoost

Name	**Description**
max_depth	Maximum depth of the tree. Increasing this parameter will also increase the complexity of the model and the likelihood of overfitting.
min_child_weight	Minimum sum of instance weight needed in a child. If a tree partitioning step results in a tree node with the sum of instance weight less than this parameter, it will give up partitioning. Setting this parameter can help to prevent overfitting.
eta	Size of shrinkage, and it is also named as “learning rate”. After each boosting step, we can get the weights of new features, and eta shrinks the feature weights to prevent overfitting.

We have tuned these parameters for LightGBM (see table 5.2.3.2).

Package: lightgbm.LGBMClassifier

Table 5.2.3: Essential Paramemters in LightGBM

Name	**Description**
n_estimators	Number of boosted trees to fit.
max_depth	Maximum depth of the tree. Increasing this parameter will also increase the complexity of the model and the likelihood of overfitting.
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.
num_leaves	Maximum tree leaves for base learners.
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.
subsample	The percent of data used for training each tree.
colsample_bytree	The percent of columns used for training each tree.

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 are gathering loca- tions 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.

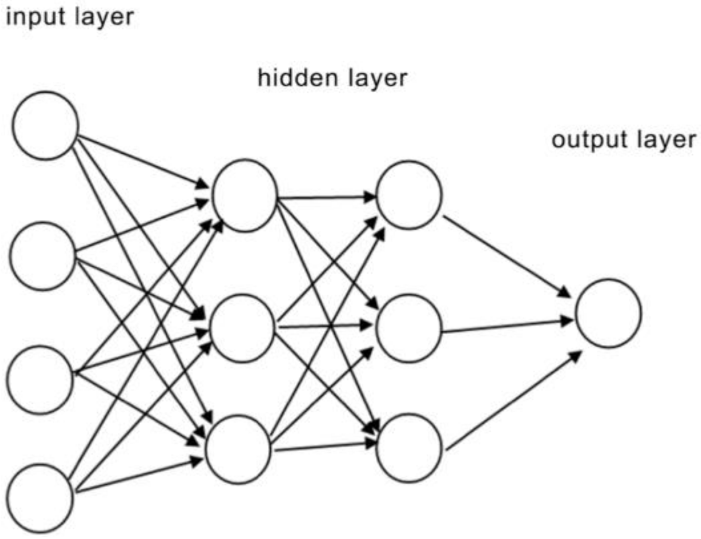


Figure 5.2.4: Neural Network Model Illustration

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

Package: sklearn.neural_network.MLPClassifier

Table 5.2.4: Essential Paramemters 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=*50, 100* 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***

6 Results

LightGBM (*n_estimators=600, max_depth=5, min_child_samples, num_leaves=20, subsample=0.6, colsample_bytree=0.8, learning_rate=0.05*) reached the highest FDR at 3% rejection rate on the testing data, and was selected as our final model. It achieved an FDR at 3% rejection rate of 56.17% on the training data, 55.61% on the testing data, and 53.55% on the OOT data. The key statistics of the top 20 bins in training, testing, and OOT data are shown in the tables below.

For each dataset, we first sorted the observations 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 bads 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, more than 70% of the records are actually a fraud. This shows that our model has a good performance in detecting fraud. Also, cumulative KS scores in the top 20 bins remain high. In each table, most of the KS scores are more than 40% and are over 50% 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 time period. The OOT data contains information from a time period that the model has never encountered before, so the model has the worst performance on it.

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

Training	# Records		# Goods		# Bads		Fraud Rate					
	596247		587632		8615		0.01444871					
	Bin Statistics					Cumulative Statistcs						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	5963	1390	4573	23.31%	76.69%	5963	1390	4573	0.24%	53.08%	52.85%	0.30
2	5963	5788	175	97.07%	2.93%	11926	7178	4748	1.22%	55.11%	53.89%	1.51
3	5963	5876	87	98.54%	1.46%	17889	13054	4835	2.22%	56.12%	53.90%	2.70
4	5963	5911	52	99.13%	0.87%	23852	18965	4887	3.23%	56.73%	53.50%	3.88
5	5963	5905	58	99.03%	0.97%	29815	24870	4945	4.23%	57.40%	53.17%	5.03
6	5963	5909	54	99.09%	0.91%	35778	30779	4999	5.24%	58.03%	52.79%	6.16
7	5963	5907	56	99.06%	0.94%	41741	36686	5055	6.24%	58.68%	52.43%	7.26
8	5963	5898	65	98.91%	1.09%	47704	42584	5120	7.25%	59.43%	52.18%	8.32
9	5963	5927	36	99.40%	0.60%	53667	48511	5156	8.26%	59.85%	51.59%	9.41
10	5963	5922	41	99.31%	0.69%	59630	54433	5197	9.26%	60.33%	51.06%	10.47
11	5963	5916	47	99.21%	0.79%	65593	60349	5244	10.27%	60.87%	50.60%	11.51
12	5963	5930	33	99.45%	0.55%	71556	66279	5277	11.28%	61.25%	49.97%	12.56
13	5963	5926	37	99.38%	0.62%	77519	72205	5314	12.29%	61.68%	49.40%	13.59
14	5963	5921	42	99.30%	0.70%	83482	78126	5356	13.30%	62.17%	48.88%	14.59
15	5963	5930	33	99.45%	0.55%	89445	84056	5389	14.30%	62.55%	48.25%	15.60
16	5963	5929	34	99.43%	0.57%	95408	89985	5423	15.31%	62.95%	47.64%	16.59
17	5963	5921	42	99.30%	0.70%	101371	95906	5465	16.32%	63.44%	47.12%	17.55
18	5963	5927	36	99.40%	0.60%	107334	101833	5501	17.33%	63.85%	46.52%	18.51
19	5963	5931	32	99.46%	0.54%	113297	107764	5533	18.34%	64.23%	45.89%	19.48
20	5963	5924	39	99.35%	0.65%	119260	113688	5572	19.35%	64.68%	45.33%	20.40

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

Testing	# Records		# Goods		# Bads		Fraud Rate					
	198749		195878		2871		0.014445356					
	Bin Statistics					Cumulative Statistcs						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	1988	447	1541	22.48%	77.52%	1988	447	1541	0.23%	53.67%	53.45%	0.29
2	1988	1929	59	97.03%	2.97%	3976	2376	1600	1.21%	55.73%	54.52%	1.49
3	1988	1964	24	98.79%	1.21%	5964	4340	1624	2.22%	56.57%	54.35%	2.67
4	1988	1968	20	98.99%	1.01%	7952	6308	1644	3.22%	57.26%	54.04%	3.84
5	1988	1977	11	99.45%	0.55%	9940	8285	1655	4.23%	57.65%	53.42%	5.01
6	1988	1970	18	99.09%	0.91%	11928	10255	1673	5.24%	58.27%	53.04%	6.13
7	1988	1975	13	99.35%	0.65%	13916	12230	1686	6.24%	58.73%	52.48%	7.25
8	1988	1970	18	99.09%	0.91%	15904	14200	1704	7.25%	59.35%	52.10%	8.33
9	1988	1967	21	98.94%	1.06%	17892	16167	1725	8.25%	60.08%	51.83%	9.37
10	1988	1983	5	99.75%	0.25%	19880	18150	1730	9.27%	60.26%	50.99%	10.49
11	1988	1975	13	99.35%	0.65%	21868	20125	1743	10.27%	60.71%	50.44%	11.55
12	1988	1973	15	99.25%	0.75%	23856	22098	1758	11.28%	61.23%	49.95%	12.57
13	1988	1978	10	99.50%	0.50%	25844	24076	1768	12.29%	61.58%	49.29%	13.62
14	1988	1977	11	99.45%	0.55%	27832	26053	1779	13.30%	61.96%	48.66%	14.64
15	1988	1975	13	99.35%	0.65%	29820	28028	1792	14.31%	62.42%	48.11%	15.64
16	1988	1978	10	99.50%	0.50%	31808	30006	1802	15.32%	62.77%	47.45%	16.65
17	1988	1978	10	99.50%	0.50%	33796	31984	1812	16.33%	63.11%	46.79%	17.65
18	1988	1972	16	99.20%	0.80%	35784	33956	1828	17.34%	63.67%	46.34%	18.58
19	1988	1975	13	99.35%	0.65%	37772	35931	1841	18.34%	64.12%	45.78%	19.52
20	1988	1972	16	99.20%	0.80%	39760	37903	1857	19.35%	64.68%	45.33%	20.41

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

OOT	# Records		# Goods		# Bads		Fraud Rate					
	166493		164107		2386		0.014330933					
	Bin Statistics					Cumulative Statistcs						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	1665	449	1216	26.97%	73.03%	1665	449	1216	0.27%	50.96%	50.69%	0.37
2	1665	1627	38	97.72%	2.28%	3330	2076	1254	1.27%	52.56%	51.29%	1.66
3	1665	1642	23	98.62%	1.38%	4995	3718	1277	2.27%	53.52%	51.25%	2.91
4	1665	1649	16	99.04%	0.96%	6660	5367	1293	3.27%	54.19%	50.92%	4.15
5	1665	1647	18	98.92%	1.08%	8325	7014	1311	4.27%	54.95%	50.67%	5.35
6	1665	1648	17	98.98%	1.02%	9990	8662	1328	5.28%	55.66%	50.38%	6.52
7	1665	1651	14	99.16%	0.84%	11655	10313	1342	6.28%	56.24%	49.96%	7.68
8	1665	1650	15	99.10%	0.90%	13320	11963	1357	7.29%	56.87%	49.58%	8.82
9	1665	1651	14	99.16%	0.84%	14985	13614	1371	8.30%	57.46%	49.16%	9.93
10	1665	1653	12	99.28%	0.72%	16650	15267	1383	9.30%	57.96%	48.66%	11.04
11	1665	1654	11	99.34%	0.66%	18315	16921	1394	10.31%	58.42%	48.11%	12.14
12	1665	1653	12	99.28%	0.72%	19980	18574	1406	11.32%	58.93%	47.61%	13.21
13	1665	1651	14	99.16%	0.84%	21645	20225	1420	12.32%	59.51%	47.19%	14.24
14	1665	1656	9	99.46%	0.54%	23310	21881	1429	13.33%	59.89%	46.56%	15.31
15	1665	1660	5	99.70%	0.30%	24975	23541	1434	14.34%	60.10%	45.76%	16.42
16	1665	1655	10	99.40%	0.60%	26640	25196	1444	15.35%	60.52%	45.17%	17.45
17	1665	1655	10	99.40%	0.60%	28305	26851	1454	16.36%	60.94%	44.58%	18.47
18	1665	1653	12	99.28%	0.72%	29970	28504	1466	17.37%	61.44%	44.07%	19.44
19	1665	1656	9	99.46%	0.54%	31635	30160	1475	18.38%	61.82%	43.44%	20.45
20	1665	1652	13	99.22%	0.78%	33300	31812	1488	19.38%	62.36%	42.98%	21.38

7 Conclusions

In this project, we created a machine learning model to identify fraudulent credit applications. We first conducted Exploratory Data Analysis (EDA) to create a Data Quality Report (DQR). Then, We cleaned our data according to the findings in the DQR. After that, we created around 500 features based on expert recommendations and research. Subsequently, we reduced the number of features in our data to 30 features using Kolmogorov-Smirnov (KS) filter and Recursive Feature Elimination (RFE). Next, we built a series of models by using five different machine learning algorithms and tuning the hyperparameters of these algorithms. Finally, we decided to use Light GBM, a Boosted Tree algorithm, to be the algorithm of our final model based on the accuracies on the testing dataset. Our final model reached a Fraud Detection Rate of 56.12% at 3% of the population, which means it can detect more than half of the fraud attempts given only the top 3% data that was sorted as suspicious by the fraud algorithm score.

The resulting model can be utilized in a credit card fraud detection system. We note that a similar model development process can be performed in related business domains such as insurance and telecommunications, to avoid or detect fraudulent activities.

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 to have the same statistical properties as real 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 10 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	Application Data
Dataset Purpose	Generated to mimic real application data and find application/identity fraud
Data Source	Came from an identity fraud prevention company
Time Period	From January 1, 2016 to December 31, 2016
# of Fields	10 fields in total – 8 categorical, 2 date
# of Record	1,000,000

Summary Statistics Table

All Fields can be treated as categorical except for 2 dates (date, dob). All 10 fields are fully populated. Key statistics of these fields are summarized as follows.

Table 8.1: Summary Statistics of Categorical fields

Field Name	Field Type	# records that have a value	% populated	# records with value zero	# unique values	Most common field value
record	categorical*	1,000,000	100	0	1,000,000	N/A**
ssn	categorical*	1,000,000	100	0	835,819	999999999
firstname	categorical	1,000,000	100	0	78,136	EAMSTRMT
lastname	categorical	1,000,000	100	0	177,001	ERJSAXA
address	categorical	1,000,000	100	0	828,774	123 MAIN ST
zip5	categorical*	1,000,000	100	0	26,370	68138
homephone	categorical*	1,000,000	100	0	28,244	9999999999
fraud_label	categorical*	1,000,000	100	0	2	0

Table 8.2: Summary Statistics of Date Fields

Field Name	Field Type	# records that have a value	% populated	# records with value zero	# unique values	Most common field value	Min	Max
date	date*	1,000,000	100	0	365	20160816	20160101	20161231
dob	date*	1,000,000	100	0	42,673	19070626	19000101	20161031

* These fields should be of the categorical/date type but are stored as integers in the dataset.

** All values in the *record* field are unique, thus no such a most common field value.

Field Description and Distribution

FIELD 1: RECORD

DESCRIPTION	Unique identifier for each record.
TYPE	Categorical
UNIQUE VALUES	1,000,000

FIELD 2: DATE

DESCRIPTION	The date when the application was submitted.
TYPE	Date
MIN	2016-01-01
MAX	2016-12-31

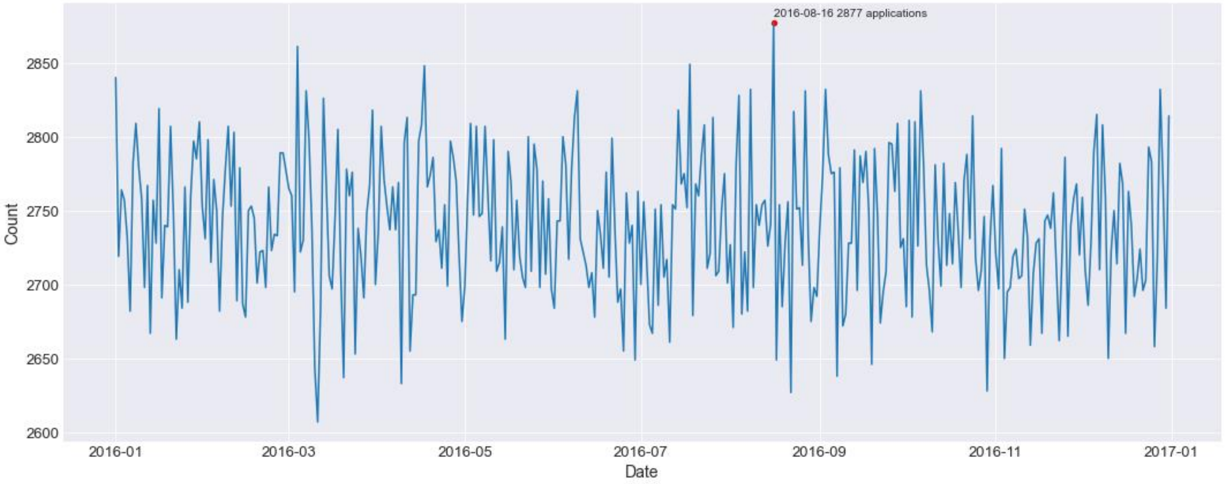


Figure 8.1: Daily Applications

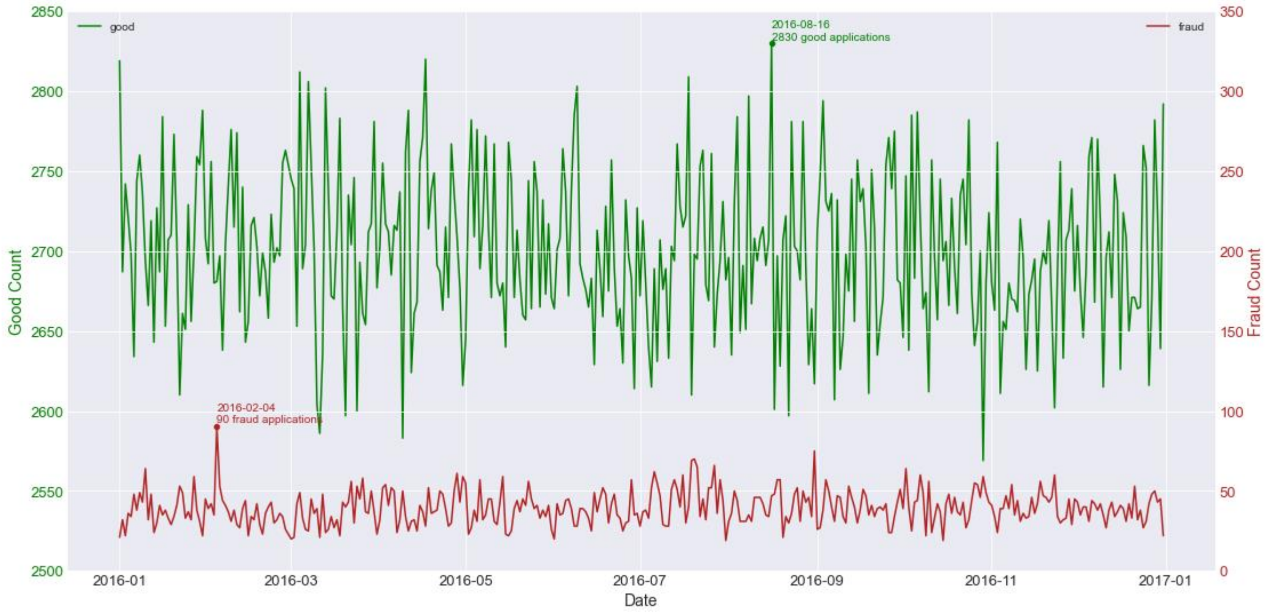


Figure 8.2: Daily Applications (Good vs. Fraud)

FIELD 3: SSN

DESCRIPTION	Applicant's social security number.
TYPE	Categorical
MOST COMMON FIELD VALUE	"999999999" occurred the most for 16935 times.

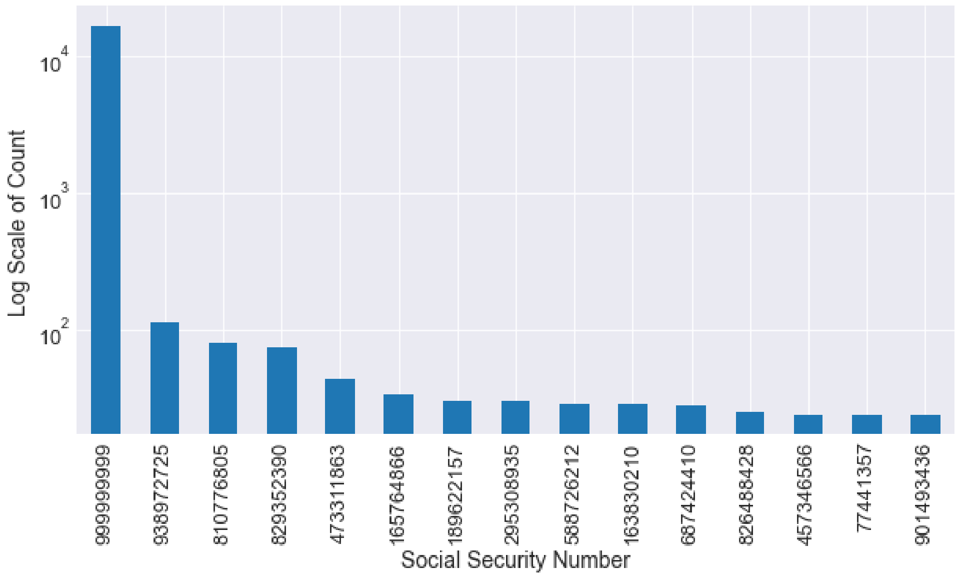


Figure 8.3: Frequency Distribution of the *ssn* Field
(Top 15 Most Common Values)

FIELD 4: FIRSTNAME

DESCRIPTION	Applicant’s first name.
TYPE	Categorical
MOST COMMON FIELD VALUE	“EAMSTRMT” occurred the most for 12658 times.

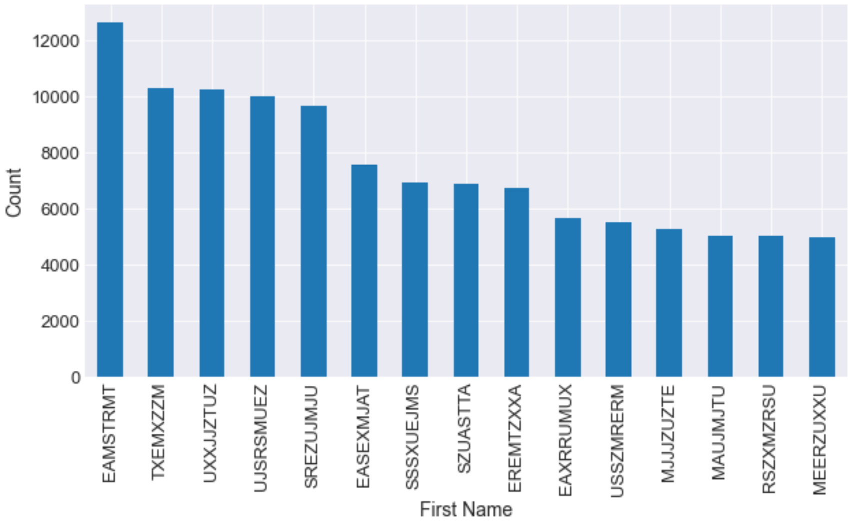


Figure 8.4: Frequency Distribution of the *firstname* Field
(Top 15 Most Common Values)

FIELD 5: LASTNAME

DESCRIPTION	Applicant’s last name.
TYPE	Categorical
MOST COMMON FIELD VALUE	“ERJSAXA” occurred the most for 8580 times.

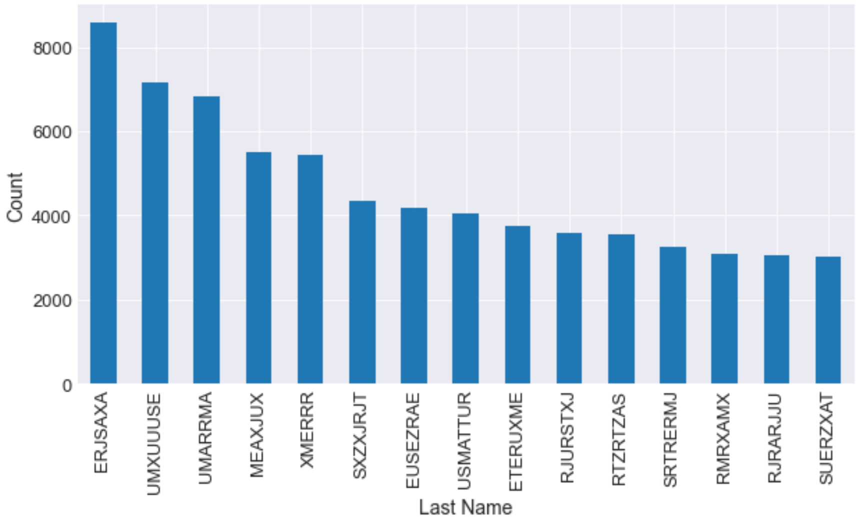


Figure 8.5: Frequency Distribution of the *lastname* Field
(Top 15 Most Common Values)

FIELD 6: ADDRESS

DESCRIPTION	Applicant’s home address.
TYPE	Categorical
MOST COMMON FIELD VALUE	“123 MAIN ST” occurred the most for 1079 times.

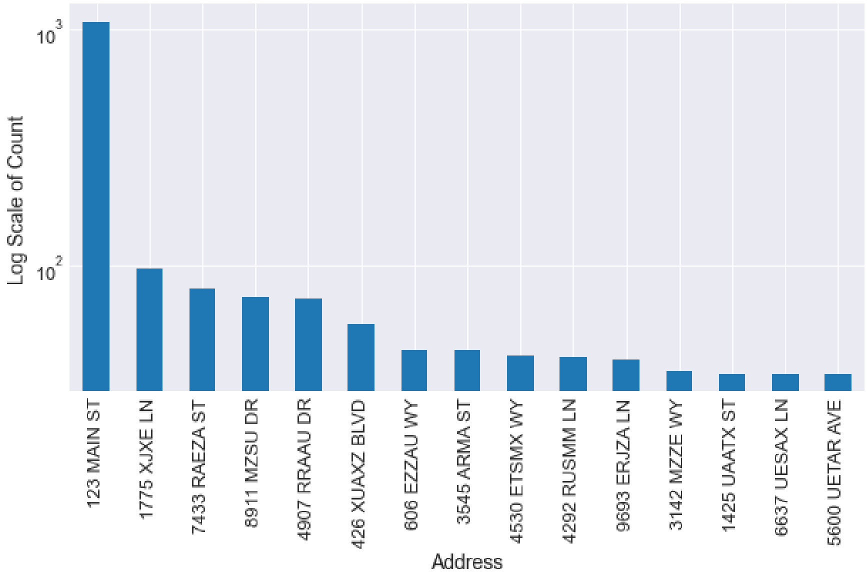


Figure 8.6: Frequency Distribution of the *address* Field
(Top 15 Most Common Values)

FIELD 7: ZIP5

DESCRIPTION	Applicant’s 5-digit zip code.
TYPE	Categorical
MOST COMMON FIELD VALUE	“68138” occurred the most for 823 times.

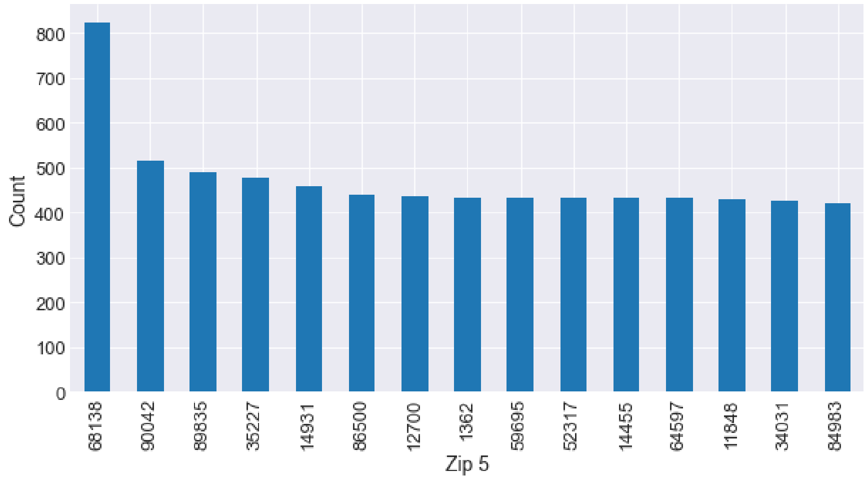


Figure 8.7: Frequency Distribution of the *zip5* Field
(Top 15 Most Common Values)

FIELD 8: DOB

DESCRIPTION	Applicant's date of birth.
TYPE	Date
MIN	1900-01-01
MAX	2016-10-31
MOST COMMON FIELD VALUE	"1907-06-26" occurred the most for 126568 times.

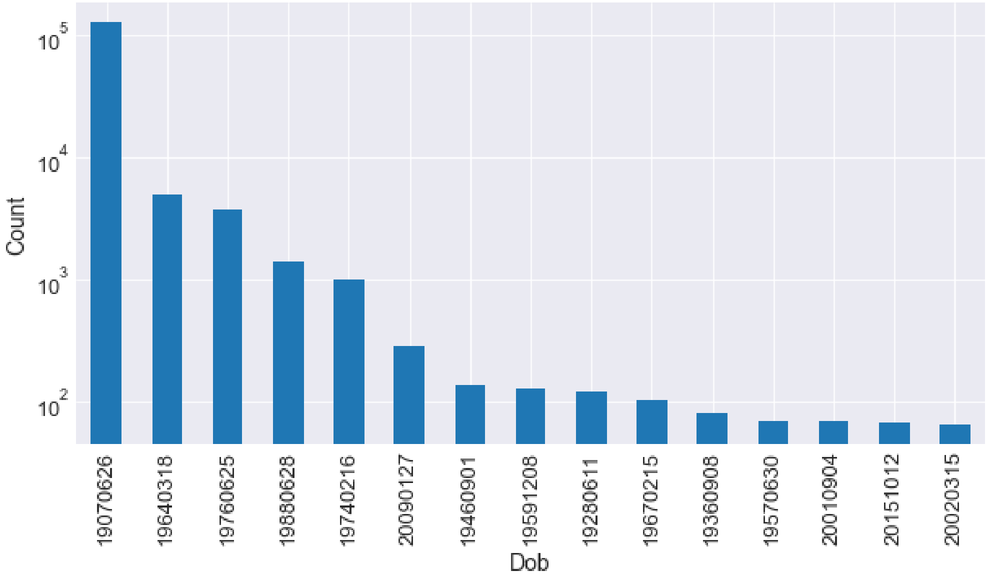


Figure 8.8: Frequency Distribution of the *dob* Field
(Top 15 Most Common Values)

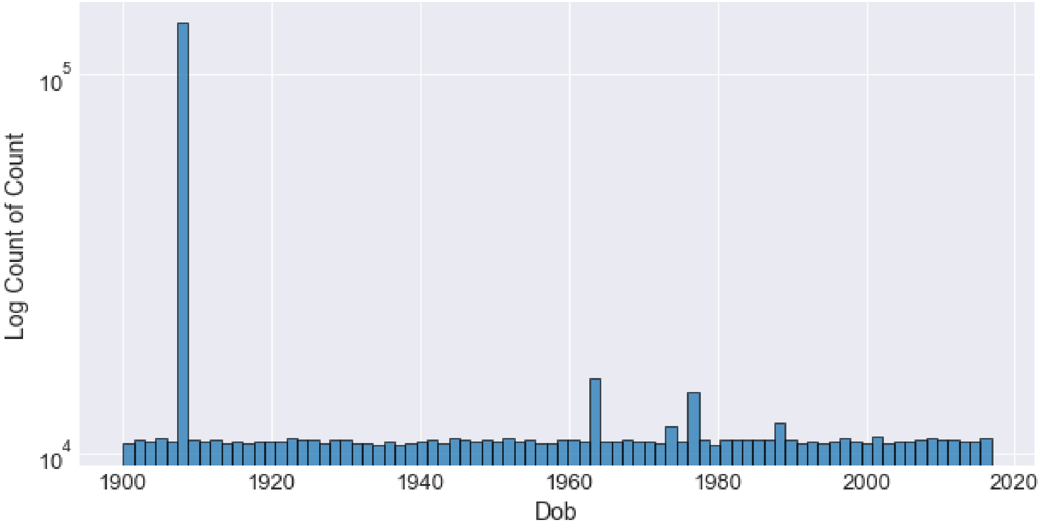


Figure 8.9: Frequency Distribution of the *dob* Field
(Sort by Date)

FIELD 9: HOMEPHONE

DESCRIPTION	Applicant’s home phone.
TYPE	Categorical
MOST COMMON FIELD VALUE	“9999999999” occurred the most for 78512 times.

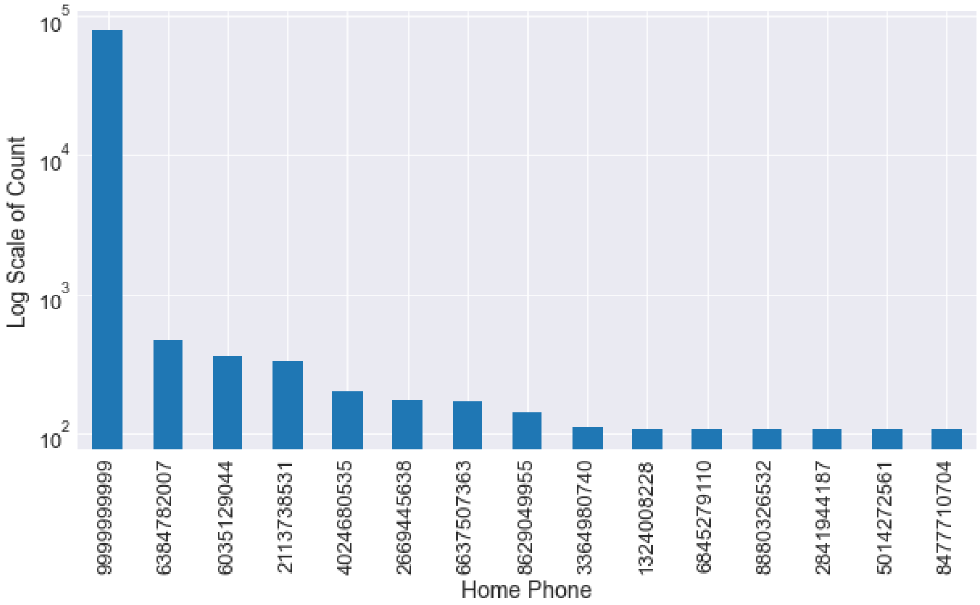


Figure 8.10: Frequency Distribution of the *homephone* Field
(Top 15 Most Common Values)

FIELD 10: FRAUD_LABEL

DESCRIPTION	Whether the application is fraudulent (“1” for Yes, “0” for No)
TYPE	Categorical
MOST COMMON FIELD VALUE	“0” occurred the most for 985607 (98.6%) times.

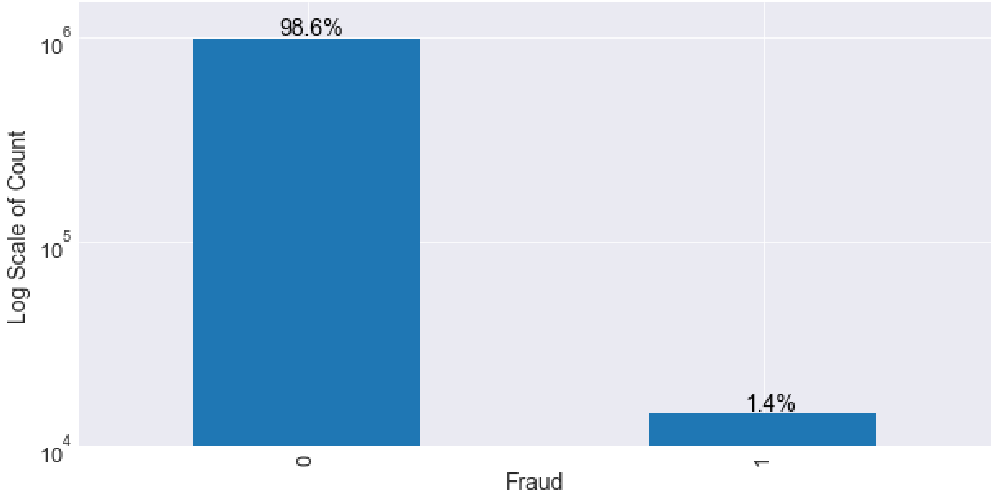


Figure 8.11: Frequency Distribution of the *fraud_label* Field

Appendix B: 80 Variables Selected after Filter

Variable	KS	FDR @3%	KS Rank	FDR Rank	Average Rank
fulladdress_lag30_count	0.3320	0.3341	519	520	519.5
address_lag30_count	0.3327	0.3327	520	519	519.5
fulladdress_#days_since	0.3235	0.3178	517	518	517.5
address_#days_since	0.3246	0.3085	518	515	516.5
fulladdress_lag14_count	0.3218	0.3146	515	517	516
fulladdress_lag7_count	0.3014	0.3106	513	516	514.5
address_lag14_count	0.3223	0.3045	516	513	514.5
address_lag7_count	0.3014	0.3081	514	514	514
address_ssn_30_day_cross_count	0.2807	0.2907	512	512	512
address_name_30_day_cross_count	0.2806	0.2904	511	511	511
address_lag3_count	0.2784	0.2877	509	510	509.5
fulladdress_lag3_count	0.2785	0.2873	510	509	509.5
address_name_14_day_cross_count	0.2758	0.2852	506	508	507
address_ssn_14_day_cross_count	0.2759	0.2851	507	507	507
address_dob_30_day_cross_count	0.2775	0.2812	508	504	506
address_dob_14_day_cross_count	0.2740	0.2836	505	506	505.5

address_ssn_7_day_cross_count	0.2726	0.2814	504	505	504.5
address_name_7_day_cross_count	0.2723	0.2808	503	503	503
address_dob_7_day_cross_count	0.2715	0.2804	502	502	502
address_ssn_3_day_cross_count	0.2637	0.2715	501	501	500.75
address_dob_3_day_cross_count	0.2633	0.2715	499	501	499.75
address_name_3_day_cross_count	0.2634	0.2711	500	499	499.5
address_lag1_count	0.2493	0.2574	498	498	498
fulladdress_lag1_count	0.2491	0.2567	497	497	497
address_name_1_day_cross_count	0.2435	0.2512	495	496	495.25
address_dob_1_day_cross_count	0.2436	0.2509	496	494	495
address_ssn_1_day_cross_count	0.2435	0.2512	494	496	494.75
zip5_homephone_lag30_count	0.2292	0.2382	492	493	492.5
address_homephone_lag30_count	0.2311	0.2372	493	492	492.5
homephone_fulladdress_lag30_count	0.2290	0.2370	491	491	491
ssn_dob_lag30_count	0.2285	0.2331	490	488	489
lastname_dob_lag30_count	0.2283	0.2333	489	489	489
name_dob_lag30_count	0.2276	0.2349	487	490	488.5
firstname_dob_lag30_count	0.2273	0.2325	486	486	486

address_homephone_#days_since	0.2279	0.2268	488	482	485
ssn_name_dob_lag30_count	0.2262	0.2326	483	487	485
ssn_lag30_count	0.2270	0.2320	485	485	485
ssn_firstname_lag30_count	0.2261	0.2319	481	484	482.5
zip5_homephone_#days_since	0.2264	0.2258	484	479	481.5
ssn_lastname_lag30_count	0.2260	0.2314	480	483	481.5
homephone_fulladdress_#days_since	0.2262	0.2261	482	480	481
address_homephone_lag14_count	0.2201	0.2264	479	481	480
name_dob_#days_since	0.2193	0.2243	475	478	476.5
zip5_homephone_lag14_count	0.2193	0.2240	476	476	475.75
ssn_dob_#days_since	0.2196	0.2236	477	474	475.5
homephone_fulladdress_lag14_count	0.2189	0.2242	473	477	475
lastname_dob_#days_since	0.2201	0.2211	478	470	474
ssn_name_dob_#days_since	0.2176	0.2227	469	472	470.5
ssn_#days_since	0.2185	0.2196	472	468	470
name_dob_lag14_count	0.2153	0.2240	464	476	469.75
ssn_firstname_#days_since	0.2178	0.2197	470	469	469.5
ssn_dob_lag14_count	0.2149	0.2230	463	473	468

firstname_dob_#days_since	0.2192	0.2159	474	462	468
ssn_lastname_#days_since	0.2175	0.2193	468	467	467.5
lastname_dob_lag14_count	0.2157	0.2190	466	465	465.5
ssn_name_dob_lag14_count	0.2135	0.2222	459	471	465
ssn_firstname_lag14_count	0.2138	0.2192	460	466	463
ssn_lag14_count	0.2144	0.2181	462	464	463
firstname_dob_lag14_count	0.2157	0.2153	465	461	463
ssn_lastname_lag14_count	0.2134	0.2177	458	463	460.5
zip5_homephone_lag7_count	0.2001	0.2118	452	460	456
address_homephone_lag7_count	0.2003	0.2105	453	458	455.5
homephone_fulladdress_lag7_count	0.1998	0.2106	451	459	455
firstname_dob_lag7_count	0.1946	0.2058	449	457	453
lastname_dob_lag7_count	0.1945	0.2054	448	456	452
name_dob_lag7_count	0.1941	0.2050	446	455	450.5
ssn_dob_lag7_count	0.1931	0.2039	445	454	449.25
ssn_lag7_count	0.1930	0.2039	444	454	448.75
ssn_firstname_lag7_count	0.1927	0.2036	443	452	447.5
ssn_lastname_lag7_count	0.1926	0.2036	442	451	446.25

ssn_name_dob_lag7_count	0.1925	0.2036	441	451	445.75
name_lag14_count	0.2045	0.1835	454	435	444.25
address_lag0_count	0.1868	0.1955	435	447	441
fulladdress_lag0_count	0.1868	0.1953	434	446	440
address_lag1_lag7_avg	0.1851	0.1977	429	449	439
fulladdress_lag1_lag7_avg	0.1852	0.1974	430	448	439
homephone_lag3_count	0.1949	0.1802	450	428	439
address_name_0_day_cross_count	0.1850	0.1935	428	444	435.75
address_dob_0_day_cross_count	0.1850	0.1937	426	445	435.5
address_ssn_0_day_cross_count	0.1850	0.1935	427	444	435.25