Credit Card Fraud Detection Algorithm

Gaetan Rieben

UC San Diego

May 13, 2024

Table of Contents

Executive Summary

This report details the creation of an advanced algorithm designed to detect credit card fraud, analyzing a dataset of 97,852 credit card transactions from a US government entity in 2010. Fraud pattern in credit card transaction is phenomena that evolves rapidly as fraudsters change their strategies and behaviors over time.

The process began with a thorough examination of the data, identifying and correcting anomalies and filling in missing information, particularly in merchant numbers and states, to ensure data integrity for accurate fraud detection. Also, we separated the dataset into three distinct parts: A training set, and a test set. Those datasets are used to train and build a model using past data. We then keep the last 2 months of transaction as an "out-of-time" – validation set. This out-of-time validation tests the model against new and unseen data. As our fraudsters techniques evolves rapidly, we generally keep the latest transactions to confirm the efficiency of our model.

In developing the fraud detection model, we employed feature engineering to create over 3,000 variables from different combinations of data points, such as card numbers and transaction locations. These variables capture key transaction metrics, but only the top 20 most influential ones were selected using advanced filtering techniques. These were then analyzed through a machine learning model, the LGBM Classifier, which proved to be highly effective in identifying fraudulent transactions without overfitting. In facts, in the top 3% of transactions in our out-of-time (unseen data), it catches 73% of the fraudulent transactions.

The deployment of this model demonstrated significant financial benefits, saving the organization approximately $40 million annually by detecting fraud. While there are minor losses due to incorrect fraud alerts, amounting to about $400,000, the net gain is substantial. Moving forward, it is imperative to continue refining the model through additional machine learning strategies and regular updates to adapt to new fraud patterns, ensuring the model's long-term effectiveness and accuracy.

Description of the data

The dataset is **Card Transactions**, which contains **credit card transactions** of a US government organization. The transactions are only related to business purposes. The data came from real credit card transactions made **over the year 2010**. There are **10 fields** and **97’852 records**. A complete Data Quality Report can be found in the appendix of this report.

Summary Tables

The below table includes summary statistics for each field of our data and help us to have a better understanding of the structure of the dataset we will work with in our task of building a credit card fraud detection algorithm.

Numeric Fields Table

Field name	# Records with value	% Populated	# Zeros	Min	Max	Mean	Standard Deviation	Most common value
Amount	97852	100	0	0.01	3102046	425.47	9949.85	3.62

Date Fields Table

Field Name	# Records have values	% Populated	# zeros	# unique values	Most Common	Min	Max
Date	97,852	100.00	0	365	2/28/10	1/1/10	9/9/10

Categorical Fields Table

Field Name	# Records have values	% Populated	# zeros	# unique values	Most Common
Recnum	97,852	100.00	0	97852	1
Cardnum	97,852	100.00	0	1645	5142148452
Merchnum	94,455	96.53	3,397	13091	930090121224

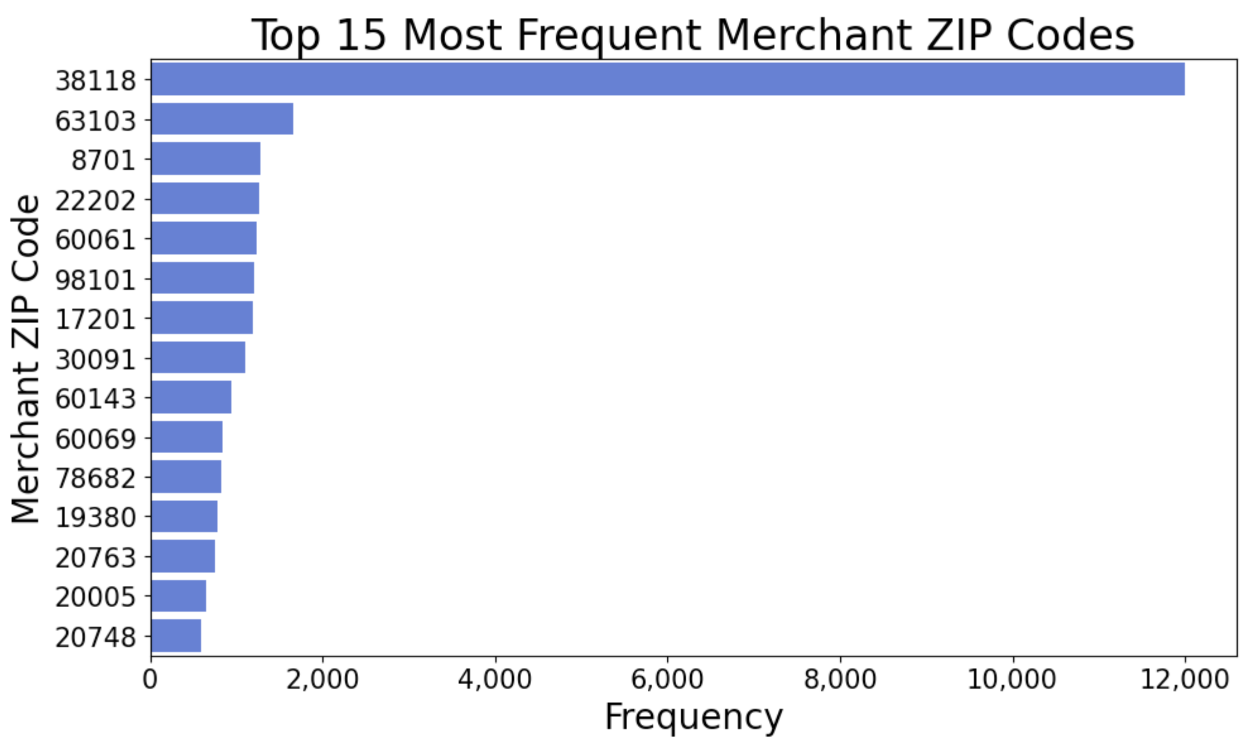
Merch description	97,852	100.00	0	13126	GSA-FSS-ADV
Merch state	96,649	98.77	1,203	227	TN
Merch zip	93,149	95.19	4,703	4567	38118
Transtype	97,852	100.00	0	4	P
Fraud	97,852	100.00	0	2	0

Data distributions

In this section, you will find some important data distribution that we found in our dataset.

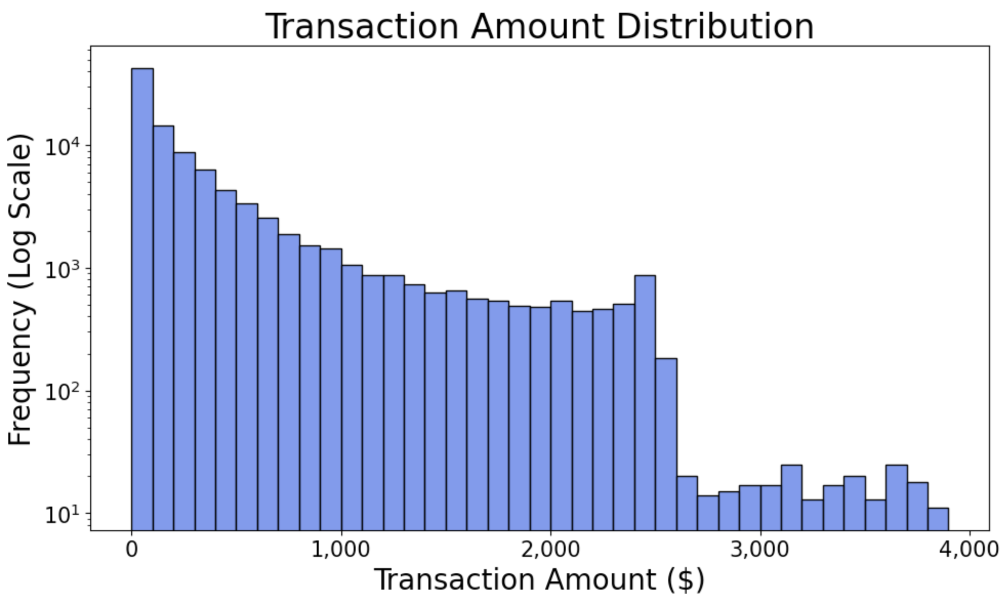
Field Name: Merch Zip

Description: Merchant's zip code. The distribution shows the top 15 field values of merchant's zip code. The most common zip code is 38118, with a total count of 11'998.



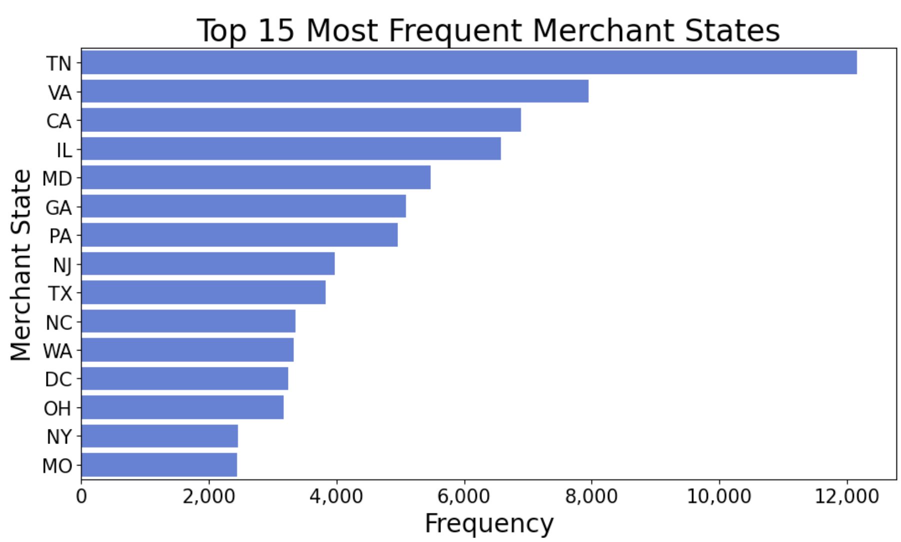
Field Name: Amount

Description: Amount of the transaction. The distribution shows the distribution of the transaction amounts. We can see that the distribution is moderate right-skewed, meaning that larger transactions are less frequent.



Field Name: Merch state

Description: State in which the merchant operates. The distribution shows the top 15 field values of merchant's state. The most common state is Tennessee, with a total count of 12'169 transactions.



Data Cleaning

In this section, we focus on getting the data ready for analysis. Imagine data as raw ingredients that need to be cleaned and prepped before cooking. We want to remove any bits that don't belong, handle any odd pieces (outliers), and fill in any gaps (missing information).

Removing Unusual Records

From the vast pool of data, we spotted some entries that raises our interest in whether we should keep those records or not:

- **A very large transaction:** There's a transaction that's way larger than the rest—over 3 million! It sticks out from the rest, and after talking it over with the team leaders, we've decided to set it aside, especially since it's not linked to any fraudulent activity.
- **Types of transactions:** Our dataset shows four types of transactions, but we're only confident about what "P" stands for—probably "Purchase." So, to keep things clear, we're only going to focus on these "P" transactions, as agreed with the management.

Filling in Missing Information

In our dataset, we have discovered that three fields have missing values. To build a robust algorithm, it is needed to deal with these missing values. Here's how we're addressing that:

- **Merchant Numbers:**
  1. If a transaction is missing the Merchant number, we first look at what the transaction was for (Merch description) and match it with similar transactions that have a number. This step take care of 1,164 records.
  2. If it's for a credit or debit adjustment, we're simply marking it as "unknown”. This step takes care of 694 records.
  3. For all other cases, we're creating unique numbers based on the transaction description and keeping track of these in a special list. As such, for each unique merchant description, a merchant number is created. For all remaining missing Merchant Numbers, we will look at the description and match it with its unique number.
- **Merchant States:**
  1. We've made a list that connects ZIP codes to states using our records, which helps us find the missing state info for each transaction. If a transaction is missing the Merchant State, we first look at the zip code and match it with its state using the list.
  2. If that doesn't work, we try matching it with the merchant number. We create a list that maps each Merchant number with its state. We look at the merchant number and match it with its corresponding state from the mapping list.
  3. We can also use the transaction description as a clue, using the same technique of mapping. After these three mapping steps, we are left with 293 records with missing values.

4. After that, if we still don't know, we'll label it as "unknown."

We also want to point out transactions that didn't happen locally by marking any non-U.S. transactions as "Foreign."

- **Zip Codes:**
  1. Like merchant states, we create two lists based on merchant numbers and descriptions to figure out the missing ZIP codes. For any missing zip code, we use merchant number or merchant description and match it with its zip code using the mapping list. This step reduces the number of missing values from 4,347 to 2,625 records.
  2. If some are still missing, we assign the ZIP code that is the most populated in the recorded state as per our dataset. As such, we need to create a list containing one zip code (The most frequent one in the state) for each state. We are then left with 1,216 null records.
  3. Any leftovers will be noted as "unknown."

Now, with our data all neat and tidy—free from outliers and missing info—we can move on to the next exciting part: transforming our data to highlight the important bits and pieces that will help in our analysis.

Variable Creation

In fraud detection, the concept of an "entity" is central to identifying and analyzing transactions. An entity refers to a unique combination of identifiers that characterizes a transaction. For example, a typical entity could be defined by concatenating a card number with a merchant number.

Consider this scenario: as a credit card user, you might frequently transact with the same merchant. However, if a transaction is registered with a new merchant or at a zip code that is significantly distant from your usual transaction locations, this could raise a flag for potential fraud.

To effectively monitor for fraudulent activities, we define unique entities by combining different transaction attributes such as card number, merchant number, location, date, etc. By creating as many of these entities as possible, we can closely observe transaction patterns and quickly identify any anomalies that deviate from the norm. This method helps in isolating unusual patterns from typical activities, enabling more precise and effective fraud detection.

In this section, we prepare a table with a list of groups of variables that I have created using different entities. First, I would like to introduce you the list of entities considered for this task. In the table, you will find the variables created and the entities used.

List of entities
Cardnum, Merchnum, card_merch, card_zip, card_state, merch_zip, merch_state, state_des, Card_Merchdesc, Card_dow, Merchnum_desc, Merchnum_dow, Merchdesc_dow, Card_Merchnum_desc, Card_Merchnum_Zip, Card_Merchdesc_Zip, Merchnum_desc_State, Merchnum_desc_Zip, merchnum_zip, Merchdesc_State, Merchdesc_Zip, Card_Merchnum_State, Card_Merchdesc_State

Description	**Entities**	**Variable Created**
Target encoded variable Average fraud percentage of categorical variable. Entity: Merch State, Merch ZIP, Day of the week, Month	Merch State, Merch Zip, Day of the week, Day of the month	4
Day since Number of days since the last transaction with that entity is seen	All	23
Frequency Number of records with the same entity over the last {0,1,3,7,14,30,50} days	All	161

Amount statistic Summary statistics including minimum, maximum, median and total amount over the last {0,1,3,7,14,30,50} days with the same entity	All	644
Transactional ratios Ratios of the actual amount of the transaction compared to summary statistics from the same entity over the last {0,1,3,7,14,30,60} days	All	644
Daily Transaction Frequency Ratios Ratio of the number of records for the same entity on the last {0,1} day(s) to the average daily number of records for those entities over the last {7,14,30,60} days	All	184
Daily Transaction Amount Ratios Ratio of the total transaction amount for the same entity on the last {0,1} day(s) to the average daily total amount for those entities over the last {7,14,30,60} days.	All	184
Frequency-Day-Since Ratios Velocity of transactions for an entity, considering both the frequency of recent transactions and the recency of activity. For each entity, it is the ratio of the number of transactions in the last {0,1} days relative to the average daily transactions over longer periods {7,14,30,60} days, further normalized by the number of days since the last transaction plus one.	All	184
Transactional Amount Variability Metrics This set of metrics quantifies the fluctuation in transaction amounts for an entity within specific recent time windows (0, 1, 3, 7, 14, 30 days). For each entity and time window, we have the average variability, the maximum and the median	All	414
Number of unique transactions by pairs of entities Count of unique transactions involving different pairs of entities within a specified time window {1,3,7,14, 30, 60}	All	696
Target encoded of pairs of entities (Variable created by me) Average fraud percentage of the pair of entities from the previous group (cell above)	All	116

Transaction Amount Bin Bins created with an ordinal number from 1 to 5, where 1 corresponds to the lowest range of amounts and 5 to the highest range of amounts.	Amount	1
Foreign Transaction The record is flagged as being a foreign transaction if the zip code of the merchant is not our list of US zip codes	Zip	1

Feature Selection

In this section, we use different techniques (filters and wrappers) to select the most important variables from all the variables we have created previously, about 3000 variables.

First, we used a filter, as it considers each candidate variable one at a time and by itself as a predictor. It is a **univariate** feature selection method. It sorts the candidate variables by their univariate importance and ignore correlations. We use a univariate KS metric to measure the performance of these variables.

After running the filter, we get a sorted list of important variables. The length of the list depends how many variables I indicated my filter to return.

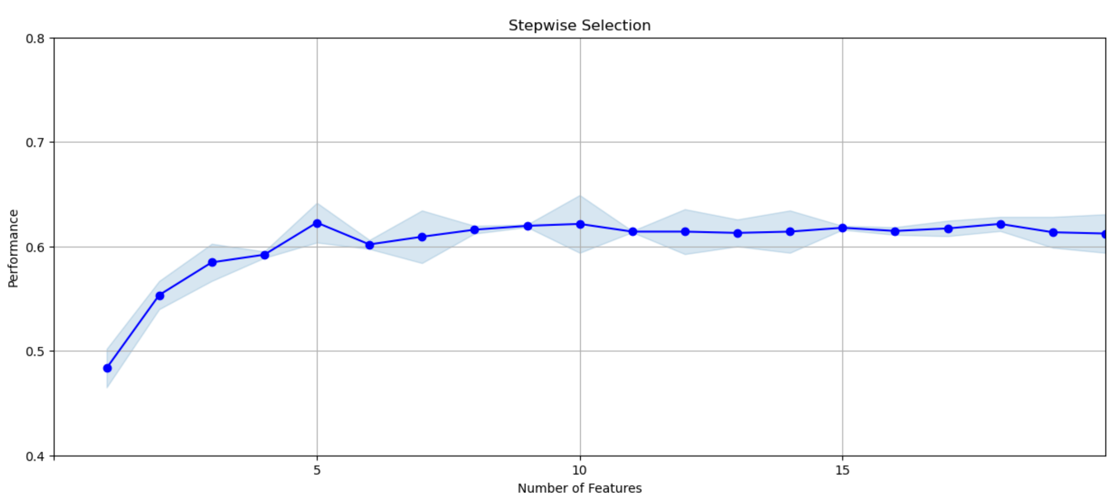
Based on these filtered variables, we are running a wrapper. A wrapper feature selection method is **multivariate.** It considers candidate variables in groups as predictor, and it makes a sorted list of candidate variable by their combined importance. It removed correlation but it takes longer to run the process. A wrapper method has a model "wrapped" around the process. As such, we will use two different models: A Random Forest, and a Lightboost model.

Around these models, I will use either a forward selection or a backward selection.

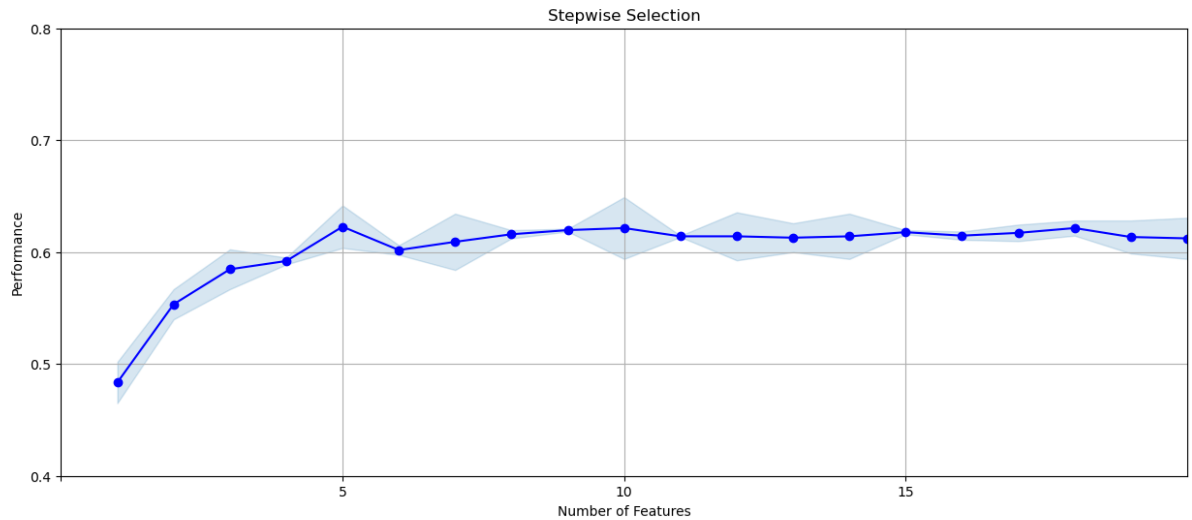
<u>First step:</u>

To avoid a long computational run in the wrapper, I am first limiting my filter to **50 variables**. We will then use the listed variables to run 4 different models and compare their performance. We will then use the best model to further increase the number of filtered variables to compare when it reached saturation in terms of performance.

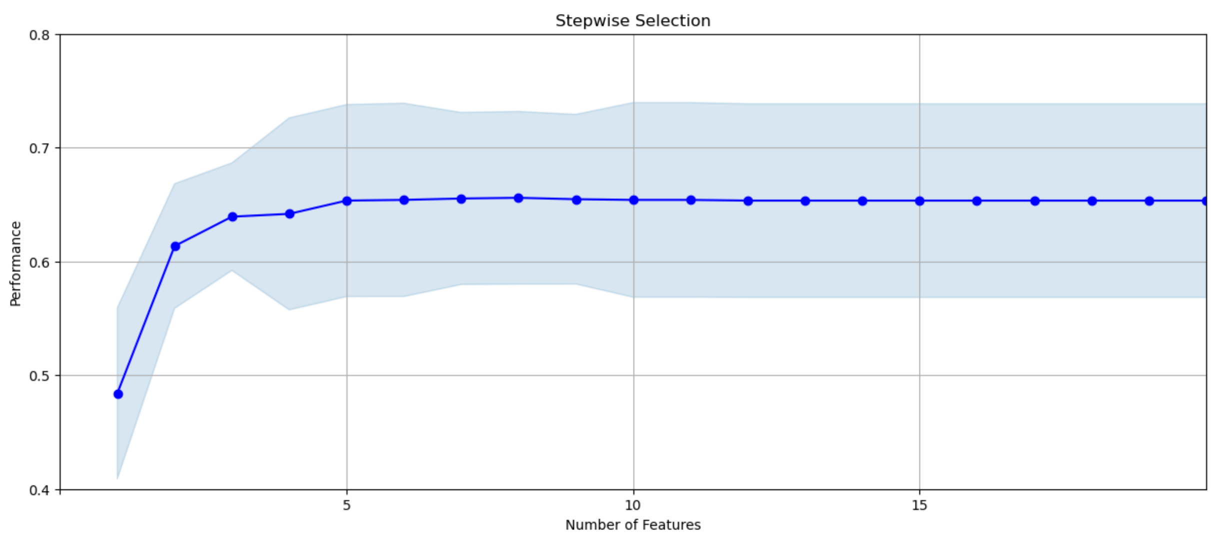
Random Forest - Forward Selection



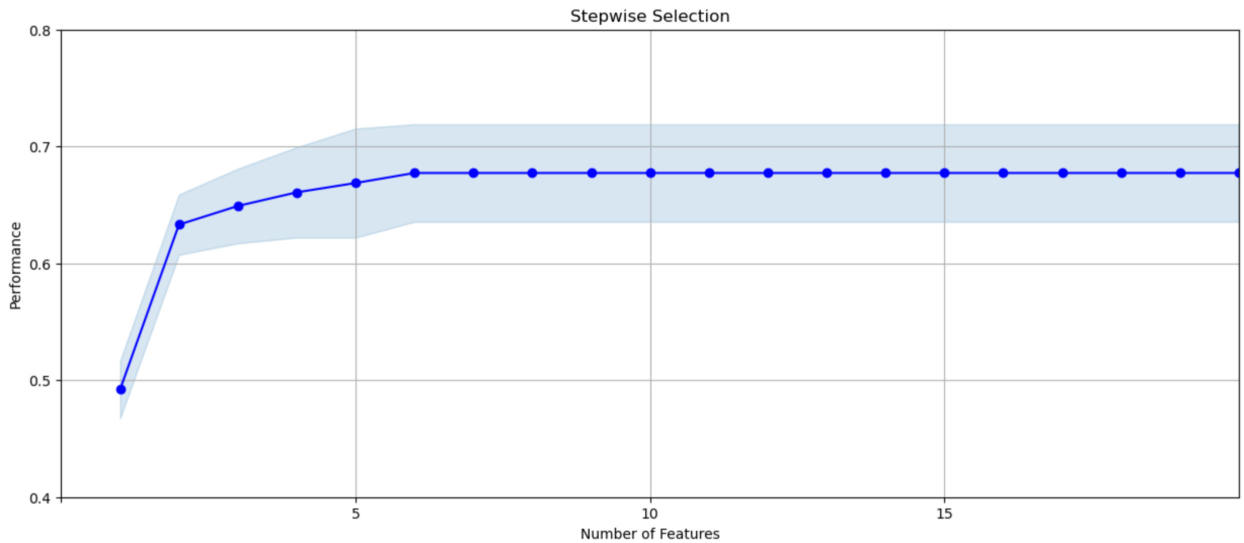
Random Forest – Backward Selection



LGBM - backward Selection



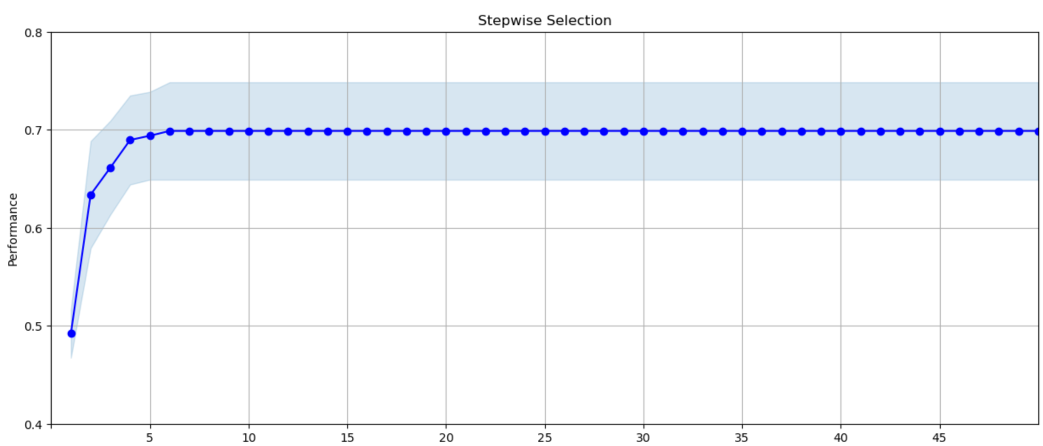
LGBM - Forward Selection



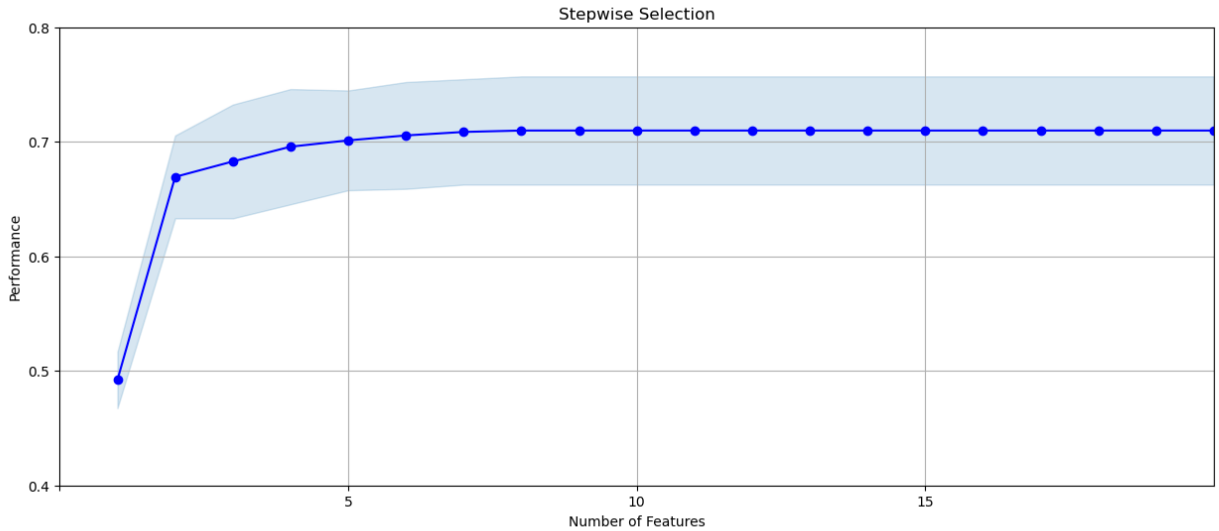
From these 4 models, we can easily see that Random Forest selection is weaker than a LGBM classifier. When comparing forward selection or backward selection of a LGBM classifier, we can conclude that the wrapper is a forward selection provides a higher performance, with much less noise than the backward selection method.

As such, we will increase the number of filtered variables and see when our model will reach saturation. In that case, we will use our last model to achieve a similar saturation to makes our final feature list.

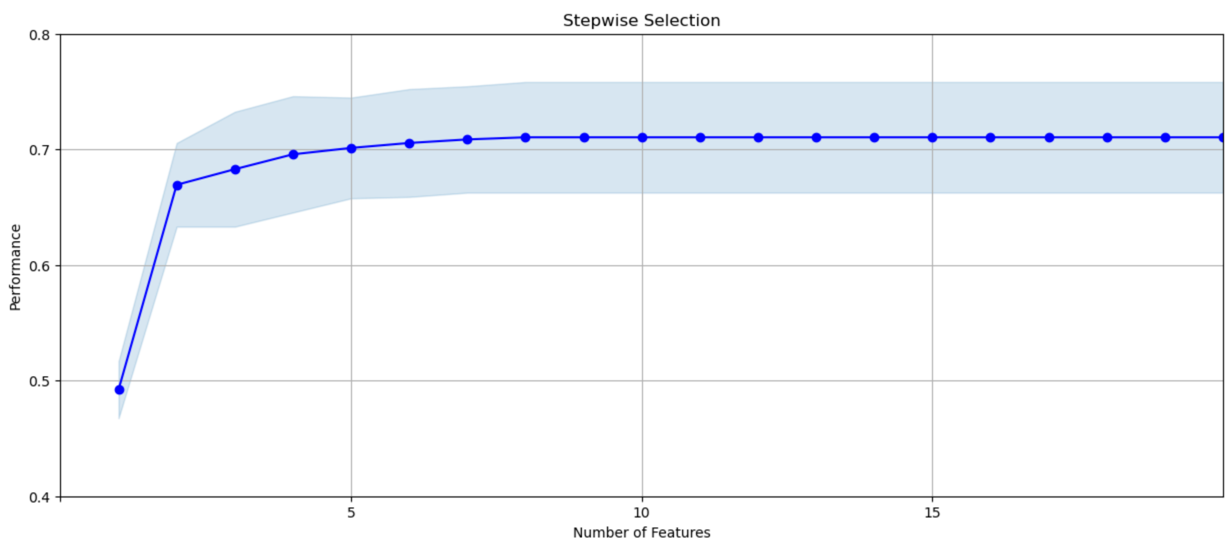
LGBM – Forward Selection with 100 filtered variables



LGBM – Forward Selection with 150 filtered variables



LGBM – Forward Selection with 200 filtered variables



Comparing the wrapper using 150 and 200 filtered variables, we can easily see that their saturation is similar. Hence, we believe that the model of choice should use **a list of 150 filtered variables.**

Below is the list of variables sorted by our wrapper to be considered to build our model (top 20 variables).

wrapper order	variable	filter score
1	Cardnum_unique_count_for_card_state_1	0.47606661
2	Card_Merchdesc_total_7	0.32463085
3	Cardnum_count_1_by_30_sq	0.4282289
4	Cardnum_max_14	0.31882556
5	Card_dow_vdratio_0by7	0.46796104
6	card_state_max_7	0.32913208
7	card_zip_count_1_by_60_sq	0.31482154
8	Merchnum_desc_total_3	0.30858598
9	Cardnum_unique_count_for_card_state_3	0.46641034
10	Cardnum_actual/toal_1	0.45971518
11	Cardnum_unique_count_for_card_state_7	0.44596675
12	Cardnum_count_14	0.44544343
13	Card_dow_count_14	0.44340533
14	Cardnum_unique_count_for_card_zip_7	0.43824177
15	Cardnum_unique_count_for_Merchnum_7	0.43693761
16	Cardnum_day_since	0.43216913
17	Card_dow_day_since	0.43216913
18	Cardnum_unique_count_for_card_state_14	0.42364178
19	Cardnum_actual/max_1	0.41890169
20	Cardnum_unique_count_for_card_zip_14	0.41645645

Model selection

In this section, We are exploring different non-linear models (Machine Learning) to find the model that best apply to our fraud case. As such, we am exploring 4 different models with different hyperparameters that reduces the number of errors to the maximum, hence, increasing its performance.

Decision Tree

A decision tree is a type of predictive modeling tool used in statistics, data mining, and machine learning. It visually and explicitly represents decisions and decision making. Here's how it works:

1. **Structure**: A decision tree is a flowchart-like structure where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

2. **Operation**: In decision-making terms, a decision tree starts at the root node and splits the data on the feature that results in the most significant information gain (IG) or the greatest reduction in impurity (like Gini impurity or entropy in the context of classification trees). This process is repeated recursively on each derived subset in a recursive manner called recursive partitioning.

3. **Advantages**:

    1. Easy to understand and interpret. Trees can be visualised.
    2. Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed.
    3. The cost of using the tree (i.e., making predictions) is logarithmic in the number of data points used to train the tree.

4. **Disadvantages**:

    1. Decision trees can create overly complex trees that do not generalize well from the training data (this is known as overfitting).
    2. They can be unstable because small variations in the data might result in a completely different tree being generated.
    3. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Random Forest

A random forest is an ensemble learning method used for classification, regression, and other tasks that operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Here's how it works:

1. **Ensemble Learning**: Random forest is a type of ensemble learning technique, which combines the predictions from multiple machine learning algorithms to make more accurate predictions than any individual model. It builds upon the idea that a group of weak learners can come together to form a strong learner.

2. **Construction**:

   - **Bootstrap Aggregating (Bagging)**: Random forests create individual trees using different subsets of the data. Each tree is built from a sample drawn with replacement (known as bootstrap sampling) from the training set.
   - **Feature Randomness**: When splitting a node during the construction of the tree, the choice of the split is no longer the best among all features. Instead, the split that is best among a random subset of features is chosen. This adds diversity to the model, hence increasing the robustness.

3. **Functioning**: Once the forest is built, it makes predictions by averaging the predictions of all the individual trees for regression tasks or using a majority vote for classification tasks. This multiple tree approach helps to reduce the risk of overfitting which is common in regular decision trees and improves accuracy.

4. **Advantages**:

   - **Accuracy**: Random forests achieve a high level of accuracy in many tasks and provide a reliable feature importance estimate.
   - **Robustness**: They are less likely to overfit than a single decision tree.
   - **Versatility**: They can handle both numerical and categorical data, can be used for both regression and classification tasks, and do a reasonably good job of handling missing data.

5. **Disadvantages**:

   - **Complexity**: Random forests create a lot of trees (sometimes hundreds or thousands) and can be quite complex to understand and interpret compared to a single decision tree.
   - **Computationally Intensive**: They require more computational resources and are slower to train than a single decision tree.

- **Less Intuitive**: The model results are more difficult to interpret than those of decision trees.

LGBM Classifier

The **LGBMClassifier** is an implementation of the LightGBM framework, which stands for Light Gradient Boosting Machine. LightGBM is an efficient and scalable gradient boosting framework that uses tree-based learning algorithms and is designed for distributed and efficient training, particularly on large datasets. Here's how it operates:

1. **Gradient Boosting Framework**: LightGBM is part of the gradient boosting family, where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It's an iterative technique that sequentially builds new models to improve upon the previous ones.

2. **Efficient Handling of Large Data**: It is designed to be distributed and efficient with the following advancements:

    - **Gradient-based One-Side Sampling (GOSS)**: This is a technique where the algorithm filters out the data instances to find a subset for training that will provide the greatest gains. It keeps all the instances with large gradients (hard to fit) and randomly selects instances with small gradients (easy to fit).
    - **Exclusive Feature Bundling (EFB)**: This method reduces the number of features in a sparse dataset by combining mutually exclusive features, thus reducing the dimensionality without significant loss of information.

3. **Tree Learning Algorithm**: LightGBM uses a histogram-based algorithm that buckets continuous feature values into discrete bins which speeds up the training process and reduces memory usage. This approach is different from traditional gradient boosting methods that grow trees level-wise. LightGBM grows trees leaf-wise, choosing the leaf it believes will yield the most reduction in loss, leading to better accuracy.

4. **Advantages**:

    - **Speed and Efficiency**: Processes data faster and is more efficient than many other implementations of gradient boosting, making it well-suited for large datasets.
    - **Lower Memory Usage**: Its histogram-based algorithms use less memory by bucketing continuous features into discrete bins.
    - **Higher Efficiency with Sparse Data**: Through techniques like EFB, it handles sparse data better than other gradient boosting methods.

5. **Disadvantages**:

   - **Overfitting Risk**: While it performs well on large datasets, like all boosting methods, it is prone to overfitting on small datasets. Proper tuning of parameters is required to avoid this.
   - **Complex Parameter Tuning**: It has a lot of hyperparameters that need tuning, which can be a challenging task without a good understanding of how each parameter affects the learning.

Neural Network

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Here's how they operate:

1. **Structure**: A typical neural network consists of a layer of input neurons (where data enters), one or more hidden layers, and an output layer (where decisions or predictions are made). Each neuron in one layer connects to neurons in the next layer with varying degrees of connection strength, represented by weights.

2. **Feedforward and Backpropagation**:

   - **Feedforward Network**: In this type of network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network.
   - **Backpropagation**: This is the training algorithm used for adjusting the weights of the connections in the network. During training, the network makes predictions (forward pass), calculates the error of the predictions compared to the actual target values, and then goes back through the network to adjust the weights (backward pass) in a way that minimizes the error.

3. **Activation Function**: Activation functions are critical in neural networks as they decide whether a neuron should be activated or not, helping the network learn complex patterns in the data. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), and softmax.

4. **Learning Process**: During training, neural networks use a method called gradient descent to iteratively adjust the weights of the connections. By minimizing a loss function (which measures prediction errors), the network learns to make increasingly accurate predictions.

5. **Advantages**:

   - **Flexibility and Versatility**: Neural networks can model complex non-linear relationships and interactions between variables, making them extremely flexible in handling a wide variety of modeling tasks.
   - **High Performance on Large Datasets**: They often deliver superior predictive performance when compared to other models, especially on large and complex datasets.

6. **Disadvantages**:

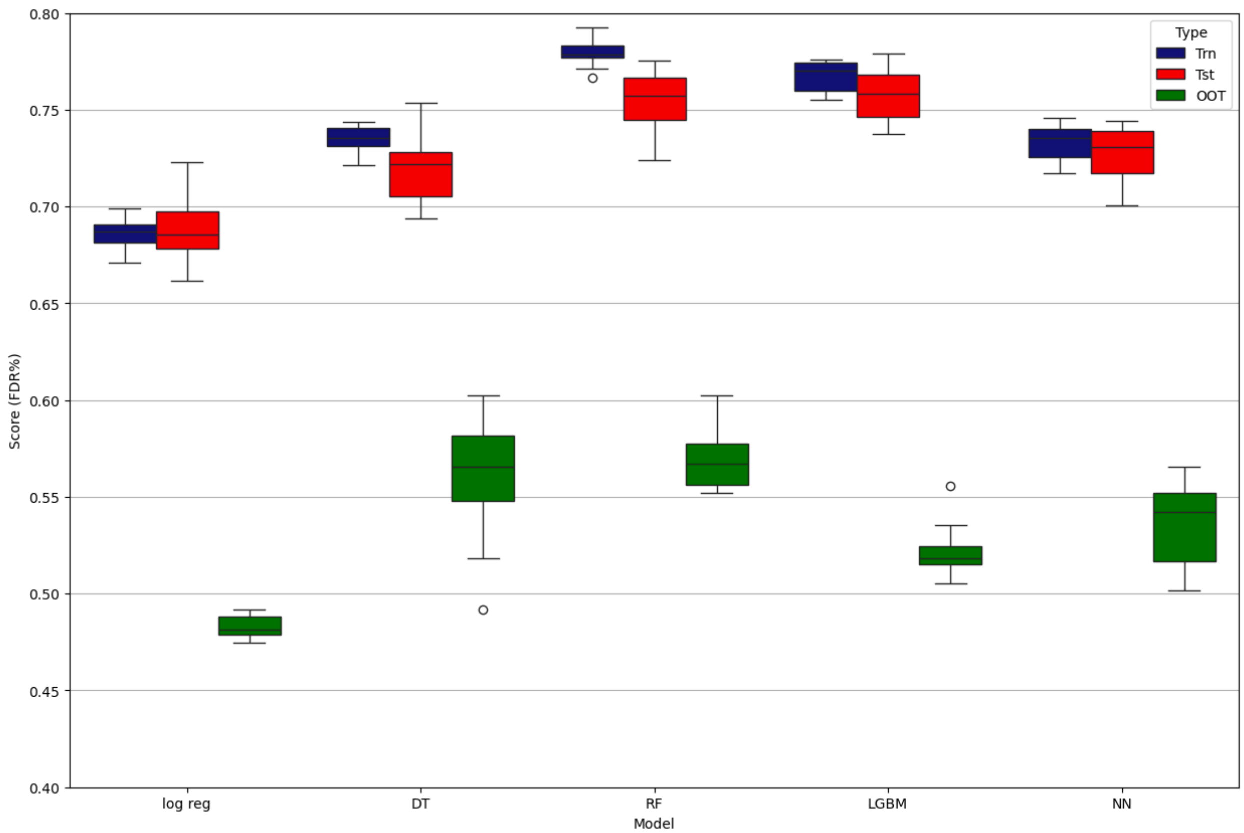
   - **Opaque Nature ("Black Box")**: Neural networks can be difficult to interpret, which means understanding how they make decisions can be challenging.
   - **Computationally Intensive**: They require significant computational resources and time to train, particularly as networks become deeper and more complex.
   - **Prone to Overfitting**: Without proper regularization techniques, neural networks tend to overfit especially when the training data is not sufficient or not representative of the general population.

Below, you find a summary table with different parameters and their performance on the training set, the test set, and the out-of-time set. Our aim is to maximize the performance on the test set, avoiding overfitting and having a performance of at least 0.50 in the out-of-time. For each model, the best one is underlined in red.

Model	Parameters				Average FDR at 3%		
Decision Tree	Criterion	max_depth	min_samples_leaf	Splitter	Train	Test	OOT
1	gini	2	1	best	0.5714	0.5577	0.4104
2	gini	5	1	best	0.7085	0.6916	0.5393
3	gini	5	2	best	0.7105	0.6961	0.5296
4	gini	5	2	random	0.6611	0.6517	0.4771
5	gini	10	2	best	0.7877	0.7065	0.4845
6	gini	15	1	best	0.8644	0.6946	0.4168
7	gini	15	60	best	0.7659	0.7251	0.5727
8	gini	50	100	best	0.7373	0.7165	0.5525
9	gini	80	50	best	0.7874	0.7472	0.5538
Random Forest	criterion	max_depth	min_samples_leaf	n_estimators	Train	Test	OOT
1	gini	4	1	100	0.7251	0.7107	0.5538
2	gini	10	2	100	0.8644	0.788	0.5612
3	gini	10	2	200	0.8616	0.7983	0.5639
4	gini	20	2	200	1	0.8316	0.5959
5	gini	5	2	200	0.7432	0.7192	0.5572
6	gini	5	2	50	0.7382	0.726	0.5535
7	gini	10	5	50	0.8516	0.7879	0.5626
8	gini	10	60	50	0.7484	0.7368	0.5535
9	gini	20	60	50	0.7784	0.7573	0.5609
10	gini	20	60	100	0.781	0.757	0.5666
LGBClassifier	sub_samples	max_depth	learning_rate	n_estimators	Train	Test	OOT
1	1	5	0.1	50	0.8743	0.8155	0.5848
2	1	5	0.1	100	0.9178	0.8185	0.5777
3	1	2	0.1	100	0.7656	0.7606	0.5356
4	1	2	0.1	200	0.7902	0.7661	0.536
Neural Network	activation	alpha	hidden layer	solver	Train	Test	OOT
1	relu	0.01	4,4	adam	0.648	0.6446	0.4723
2	relu	0.01	3,3	adam	0.7038	0.7004	0.4983
3	tanh	0.01	3,3	adam	0.7215	0.7136	0.5245
4	tanh	0.1	3,3	adam	0.717	0.7133	0.5101
5	tanh	0.1	5,5	adam	0.7339	0.7228	0.5215
6	relu	0.0001	100	adam	0.8189	0.7771	0.5326

To select the best model, it is helpful to plot the performance of each one of them on a box plot. It allows to see visually if there is any over-fitting and help to compare the overall performance of each one of them.

Although we have a slight overfitting in the Random Forest model, we have good performance on the out-of-time. However, when looking at the LGBM model, we avoid overfitting, and we still get a performance above 0.50 in the out-of-time. Given the stability of this model, we believe this is the model of choice.



Final model performance

As seen in the previous section, our final model is a LGBM Classifier, as it avoids over-fitting and has an overall good performance for our fraud detection task. Below, you will find further details about the final model, and what hyperparameters have been used to maximize its performance.

1. **Subsample = 1**:
    - **Description**: This hyperparameter controls the fraction of the total training set that is randomly sampled (without replacement) to build each tree. It is also known as the bagging fraction.
    - **Purpose**: Using a subsample value less than 1 can lead to faster training times and can also help prevent overfitting by introducing more randomness into the model training process. A value of **1** means that the model uses all available training data to build each tree.

2. **max_depth = 2**:
    - **Description**: This hyperparameter specifies the maximum depth of each tree that is grown during the learning process.
    - **Purpose**: Limiting the depth of the tree helps prevent the model from becoming overly complex and overfitting to the training data. A **max_depth** of **2** means each tree in the ensemble can have at most two levels. This creates simpler, more generalizable decision rules.

3. **learning_rate = 0.1**:
    - **Description**: Also known as the shrinkage rate or step size, this parameter controls how much the contribution of each tree influences the final outcome.
    - **Purpose**: A smaller learning rate requires more trees to be included in the model (increasing **n_estimators**) but can lead to better performance through more gradual learning and less risk of overfitting. A learning rate of **0.1** is a moderate choice, balancing model training speed and the need to fit complex patterns.

4. **n_estimators = 100**:
    - **Description**: This hyperparameter specifies the number of boosting stages the model has to go through, or in other words, the number of trees to use in the ensemble.
    - **Purpose**: More trees can lead to a more accurate model but also increase the risk of overfitting and the computational cost. A value of **100** trees is typically a good starting point to balance model complexity and training efficiency.

We will now explore the statistics of our model in the training, the test and the out-of-time dataset. Those statistics allows us to have a clear view on the distribution of non-fraud records (Goods) misclassified as fraudulent, the number of fraudulent transactions (Bads) – correctly classified.

For this analysis, we assume that flagging a fraudulent transaction implies 400 dollars saving, and a non-fraudulent that has been wrongly flagged by our algorithm as a loss of 20 dollars. A summary of the financial impact is included in our statistics table.

Training	Bin Statistic					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	597	48	549	8.04020101	91.959799	597	48	549	0.08210602	44.8896157	44.8075097	0.08743169
2	597	294	303	49.2462312	50.7537688	1194	342	852	0.58500539	69.6647588	69.0797534	0.40140845
3	597	499	98	83.5845896	16.4154104	1791	841	950	1.43856588	77.6778414	76.2392755	0.88526316
4	596	547	49	91.7785235	8.22147651	2387	1388	999	2.37423239	81.6843827	79.3101503	1.38938939
5	597	554	43	92.7973199	7.20268007	2984	1942	1042	3.3218727	85.2003271	81.8784544	1.86372361
6	597	583	14	97.6549414	2.34505863	3581	2525	1056	4.31911873	86.3450531	82.0259344	2.39109848
7	597	577	20	96.6499162	3.35008375	4178	3102	1076	5.3061015	87.9803761	82.6742746	2.88289963
8	597	584	13	97.8224456	2.17755444	4775	3686	1089	6.30505807	89.0433361	82.738278	3.38475666
9	597	582	15	97.4874372	2.51256281	5372	4268	1104	7.30059356	90.2698283	82.9692347	3.86594203
10	596	585	11	98.1543624	1.84563758	5968	4853	1115	8.30126067	91.1692559	82.8679953	4.35246637
11	597	592	5	99.1624791	0.83752094	6565	5445	1120	9.31390158	91.5780867	82.2641851	4.86160714
12	597	590	7	98.8274707	1.17252931	7162	6035	1127	10.3231214	92.1504497	81.8273283	5.35492458
13	597	592	5	99.1624791	0.83752094	7759	6627	1132	11.3357623	92.5592805	81.2235182	5.85424028
14	597	585	12	97.9899497	2.01005025	8356	7212	1144	12.3364294	93.5404742	81.2040448	6.3041958
15	597	593	4	99.3299832	0.67001675	8953	7805	1148	13.3507809	93.8675388	80.516758	6.79878049
16	596	592	4	99.3288591	0.67114094	9549	8397	1152	14.3634218	94.1946034	79.8311817	7.2890625
17	597	594	3	99.4974874	0.50251256	10146	8991	1155	15.3794838	94.4399019	79.0604181	7.78441558
18	597	593	4	99.3299832	0.67001675	10743	9584	1159	16.3938352	94.7669665	78.3731313	8.26919758
19	597	596	1	99.8324958	0.16750419	11340	10180	1160	17.4133183	94.8487326	77.4354143	8.77586207
20	597	589	8	98.6599665	1.3400335	11937	10769	1168	18.4208276	95.5028618	77.0820343	9.22003425

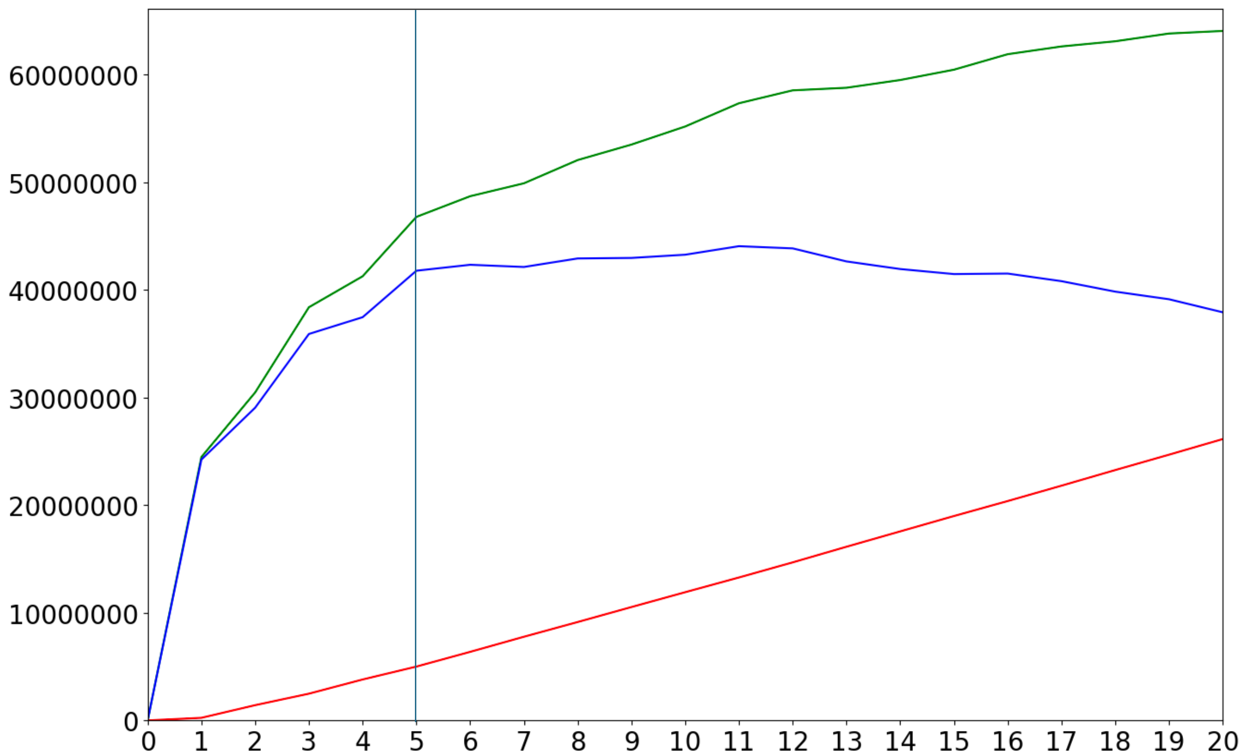
Test	Bin Statistic					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
0	0	0	0	0	0	0	0	0	0	0	0	0
1	256	32	224	12.5	87.5	256	32	224	0.12772921	42.5047438	42.3770146	0.14285714
2	256	136	120	53.125	46.875	512	168	344	0.67057837	65.2751423	64.6045639	0.48837209
3	255	214	41	83.9215686	16.0784314	767	382	385	1.52476749	73.0550285	71.530261	0.99220779
4	256	236	20	92.1875	7.8125	1023	618	405	2.46677045	76.8500949	74.3833244	1.52592593
5	256	240	16	93.75	6.25	1279	858	421	3.42473955	79.886148	76.4614085	2.03800475
6	256	244	12	95.3125	4.6875	1535	1102	433	4.39867481	82.1631879	77.764513	2.54503464
7	256	247	9	96.484375	3.515625	1791	1349	442	5.38458468	83.8709677	78.4863831	3.0520362
8	255	245	10	96.0784314	3.92156863	2046	1594	452	6.36251148	85.7685009	79.4059895	3.52654867
9	256	251	5	98.046875	1.953125	2302	1845	457	7.3643875	86.7172676	79.3528801	4.03719912
10	256	251	5	98.046875	1.953125	2558	2096	462	8.36626352	87.6660342	79.2997706	4.53679654
11	256	250	6	97.65625	2.34375	2814	2346	468	9.36414801	88.8045541	79.4404061	5.01282051
12	256	252	4	98.4375	1.5625	3070	2598	472	10.3700156	89.5635674	79.1935518	5.50423729
13	255	254	1	99.6078431	0.39215686	3325	2852	473	11.3838662	89.7533207	78.3694545	6.02959831
14	256	255	1	99.609375	0.390625	3581	3107	474	12.4017084	89.943074	77.5413656	6.55485232
15	256	249	7	97.265625	2.734375	3837	3356	481	13.3956013	91.2713472	77.8757459	6.97713098
16	256	254	2	99.21875	0.78125	4093	3610	483	14.409452	91.6508539	77.2414019	7.47412008
17	256	254	2	99.21875	0.78125	4349	3864	485	15.4233026	92.0303605	76.6070579	7.96701031
18	255	251	4	98.4313725	1.56862745	4604	4115	489	16.4251786	92.7893738	76.3641952	8.41513292
19	256	255	1	99.609375	0.390625	4860	4370	490	17.4430208	92.9791271	75.5361063	8.91836735
20	256	254	2	99.21875	0.78125	5116	4624	492	18.4568714	93.3586338	74.9017623	9.39837398

OOT	Bin Statistic					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
0	0	0	0	0	0	0	0	0	0	0	0	0
1	122	20	102	16.3934426	83.6065574	122	20	102	0.16757436	34.3434343	34.17586	0.19607843
2	123	98	25	79.6747967	20.3252033	245	118	127	0.98868873	42.7609428	41.772254	0.92913386
3	122	89	33	72.9508197	27.0491803	367	207	160	1.73439464	53.8720539	52.1376592	1.29375
4	122	110	12	90.1639344	9.83606557	489	317	172	2.65605362	57.9124579	55.2564043	1.84302326
5	123	100	23	81.300813	18.699187	612	417	195	3.49392543	65.6565657	62.1626402	2.13846154
6	122	114	8	93.442623	6.55737705	734	531	203	4.44909929	68.3501684	63.9010691	2.61576355
7	122	117	5	95.9016393	4.09836066	856	648	208	5.4294093	70.03367	64.6042607	3.11538462
8	123	114	9	92.6829268	7.31707317	979	762	217	6.38458316	73.0639731	66.6793899	3.51152074
9	122	116	6	95.0819672	4.91803279	1101	878	223	7.35651445	75.0841751	67.7276606	3.93721973
10	122	115	7	94.2622951	5.73770492	1223	993	230	8.32006703	77.4410774	69.1210104	4.3173913
11	123	114	9	92.6829268	7.31707317	1346	1107	239	9.27524089	80.4713805	71.1961396	4.63179916
12	122	117	5	95.9016393	4.09836066	1468	1224	244	10.2555509	82.1548822	71.8993313	5.01639344
13	122	121	1	99.1803279	0.81967213	1590	1345	245	11.2693758	82.4915825	71.2222067	5.48979592
14	122	119	3	97.5409836	2.45901639	1712	1464	248	12.2664432	83.5016835	71.2352403	5.90322581
15	123	119	4	96.7479675	3.25203252	1835	1583	252	13.2635107	84.8484848	71.5849742	6.28174603
16	122	116	6	95.0819672	4.91803279	1957	1699	258	14.235442	86.8686869	72.6332449	6.58527132
17	122	119	3	97.5409836	2.45901639	2079	1818	261	15.2325094	87.8787879	72.6462785	6.96551724
18	123	121	2	98.3739837	1.62601626	2202	1939	263	16.2463343	88.5521886	72.3058542	7.37262357
19	122	119	3	97.5409836	2.45901639	2324	2058	266	17.2434018	89.5622896	72.3188878	7.73684211
20	122	121	1	99.1803279	0.81967213	2446	2179	267	18.2572266	89.8989899	71.6417633	8.16104869

Financial Curves

With a fraud algorithm, we aim to deny as few transactions as possible but also to get a good overall savings. To do so, we can plot the financial impact on a graph with three different lines. The green light is the amount of fraud caught (as per our financial assumptions), the red line implies the lost revenue when we do block a transaction that was not a fraudulent one, and the blue line implies the overall savings.

Our out-of-time dataset representing only 2 months out of a full year of records, and we assume to have 100,000 samples out of 10 million transactions a year. Hence, we multiply the out-of-time by (12/2) * (10,000,000 / 100'000). Below is the financial impact of our simulation.



We recommend a cutoff as far as left as possible, but away from the sharp increase that we observe. Looking at this plot, we do recommend assigning a cutoff at 5% (5% of the transactions). At this score, we will maximize the overall saving by blocking most of the fraudulent transactions, without the risk of losing revenue by wrongly flagging a record. The transactions at the right of the cutoff represents the percentage of transaction that will not be blocked.

Summary

This report documents the development of a sophisticated algorithm for detecting credit card fraud, utilizing a dataset comprising credit card transactions from a US government organization from the year 2010. The dataset contains 97,852 records across ten fields, including transaction amount, merchant information, and fraud status.

To build a fraud detection algorithm, we went through key steps required to build a robust model. At first, we explored the dataset and its distribution, allowing to prepare the data for the most important step – building a model. As such, we identified and removed unusual records, such an outlier transaction. We addressed missing information, particularly in merchant numbers and states, by devising methods to infer missing data effectively. To build a model, we split the dataset into three sets: One for training, another one for testing, and an out-of-time. Out-of-time validation tests the model against new and unseen data. As our fraudsters techniques evolves rapidly, we generally keep the latest transactions to confirm the efficiency of our model.

One of the most important steps being feature engineering, we created various numbers of variables by analyzing entities, such as combination of card number with location. We created variables using those entities with important metrics in credit card transactions. Building metrics such as transactions frequency or variability, the number of days since a last transaction help to build a robust model. We created more than 3,000 variables through this key step, but not all of them are important and decisive to detect fraud. As such, we used some filtering techniques and build a wrapper around a non-linear model – LGBM Classifier. This method shortlisted a list of the most 20 important variables.

Using those variables, we explored different machine learning techniques, aiming to maximize its overall performance, and avoiding an overfitting issue. Out of the 4 nonlinear models, we found out that a LGBM Classifier is the most performant to detect frauds. However, it is important to underline that it can be achieve with a thorough selection of hyperparameters.

Using this model, we have explored its statistics, comparing the number of frauds detected and the ones which were wrongly flagged. Plotting those financial impacts, we can conclude that our fraud algorithm saves the company 40,000,000 dollars a year. Although the company might lose about 400,0000 of revenues with wrongly blocked transactions, it saves about 40,400,000 dollars on fraudulent transactions. In facts, in the top 3% of transactions in our out-of-time (unseen data), it catches 73% of the fraudulent transactions.

Although the model is robust, exploring additional machine learning techniques or experimenting further with hyperparameters could enhance its effectiveness. Moreover, it is crucial to emphasize the necessity of continuously monitoring and retraining the model to ensure its long-term performance.

Appendix: Data Quality Report

Data Quality Report

1. Data Description

The dataset is **Card Transactions**, which contains **credit card transactions** of a US government organization. The transactions are only related to business purposes. The data came from real credit card transactions made **over the year 2010**. There are **10 fields** and **97'852 records**.

2. Summary Tables

Numeric Fields Table

Field name	# Records with value	% Populated	# Zeros	Min	Max	Mean	Standard Deviation	Most common value
Amount	97852	100	0	0.01	3102046	425.47	9949.85	3.62

Date Fields Table

Field Name	# Records have values	% Populated	# zeros	# unique values	Most Common	Min	Max
Date	97,852	100.00	0	365	2/28/10	1/1/10	9/9/10

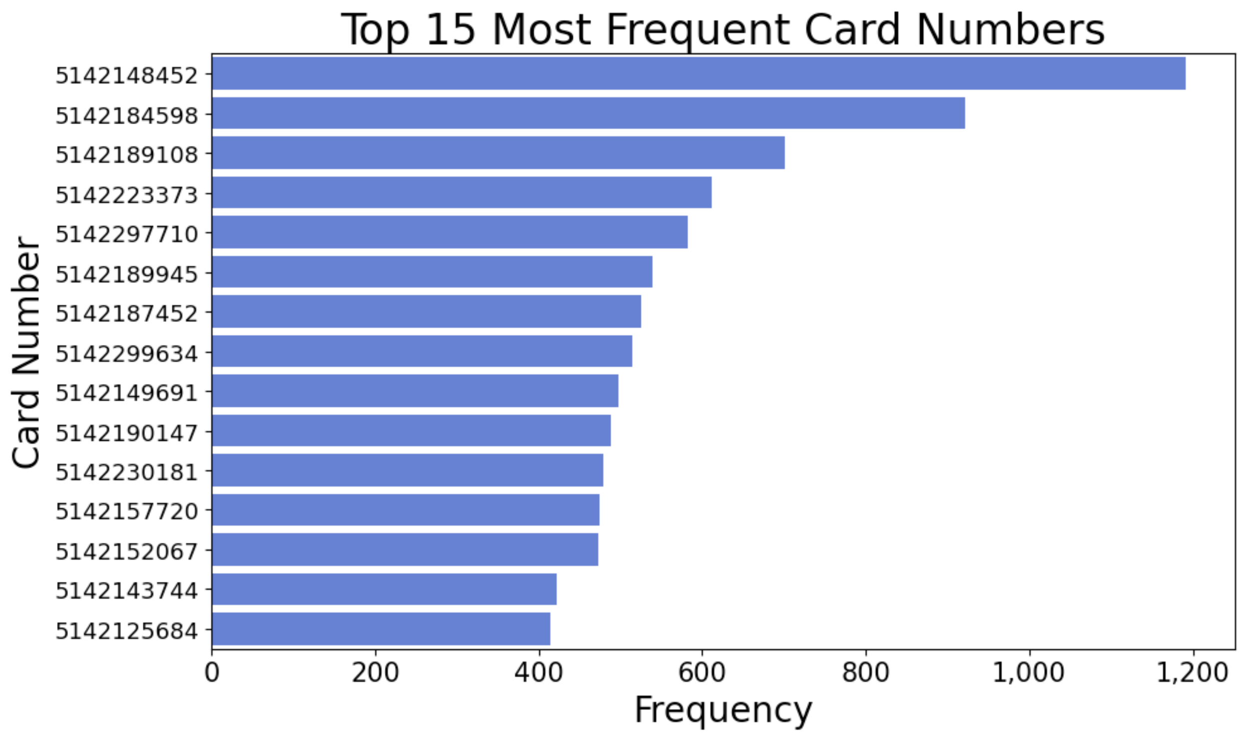
Categorical Fields Table

Field Name	# Records have values	% Populated	# zeros	# unique values	Most Common
Recnum	97,852	100.00	0	97852	1
Cardnum	97,852	100.00	0	1645	5142148452
Merchnum	94,455	96.53	3,397	13091	930090121224
Merch description	97,852	100.00	0	13126	GSA-FSS-ADV
Merch state	96,649	98.77	1,203	227	TN
Merch zip	93,149	95.19	4,703	4567	38118
Transtype	97,852	100.00	0	4	P
Fraud	97,852	100.00	0	2	0

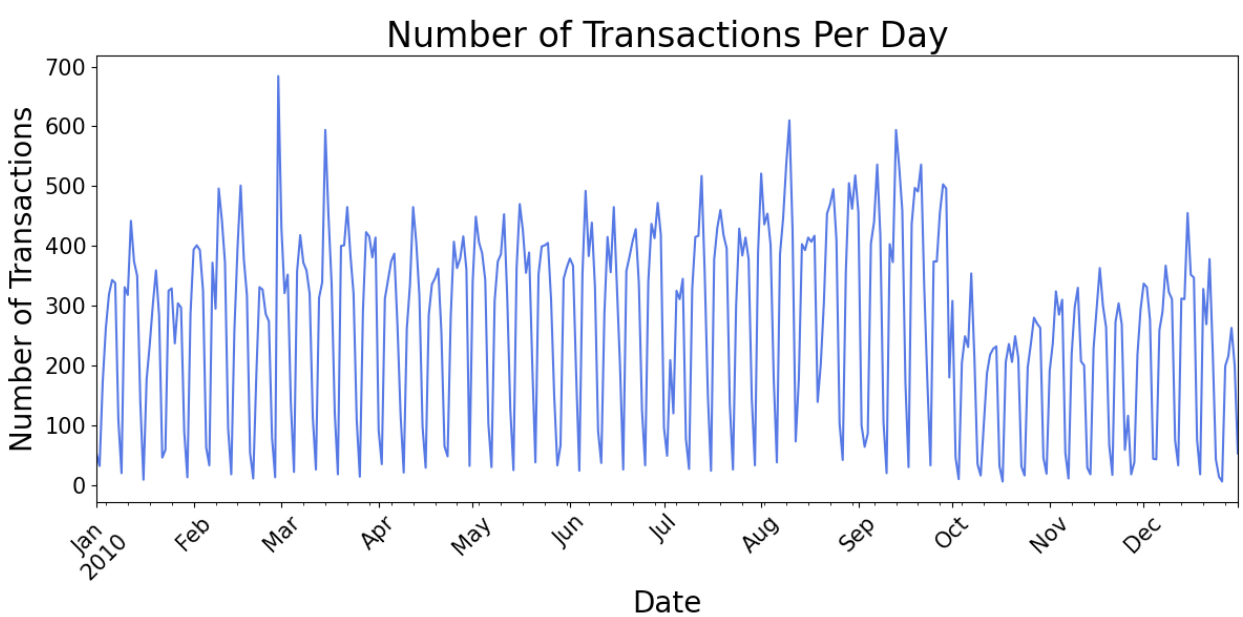
3. Visualization of Each Field

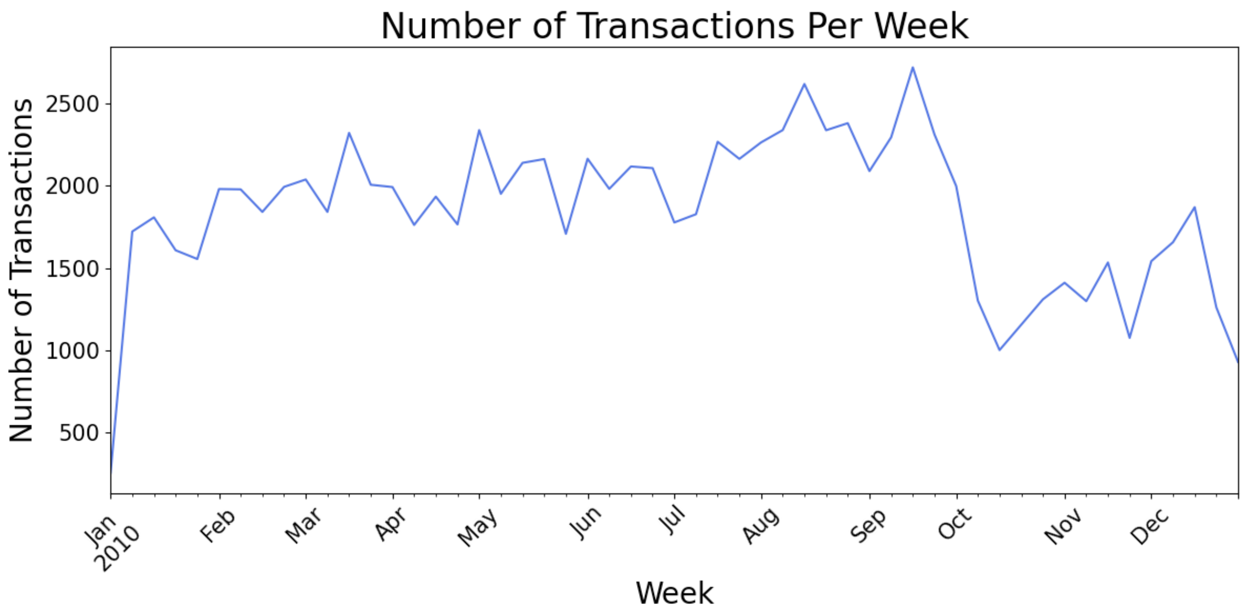
1) **Field Name: Recnum**
   Description: Ordinal unique positive integer for each application record, from 1 to 97’852.

2) **Field Name: Cardnum**
   Description: Card number used for the transaction observation. We can see that the most used card is the one with number “5142148452”, as this card is related to 1’192 transactions.



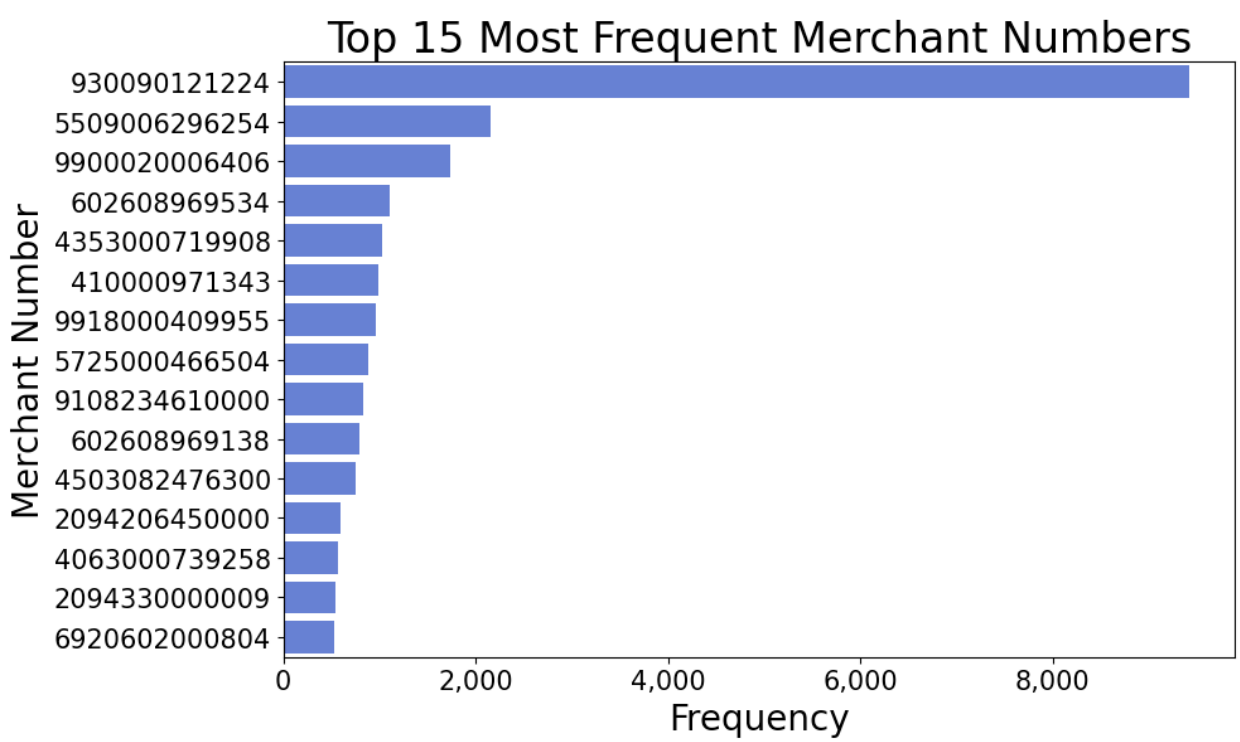
3) **Field Name: Date**
Description: Date of the transaction. The first distribution shows the number of transactions per day during the year 2010. The second distribution shows the number of transactions per week during 2010.





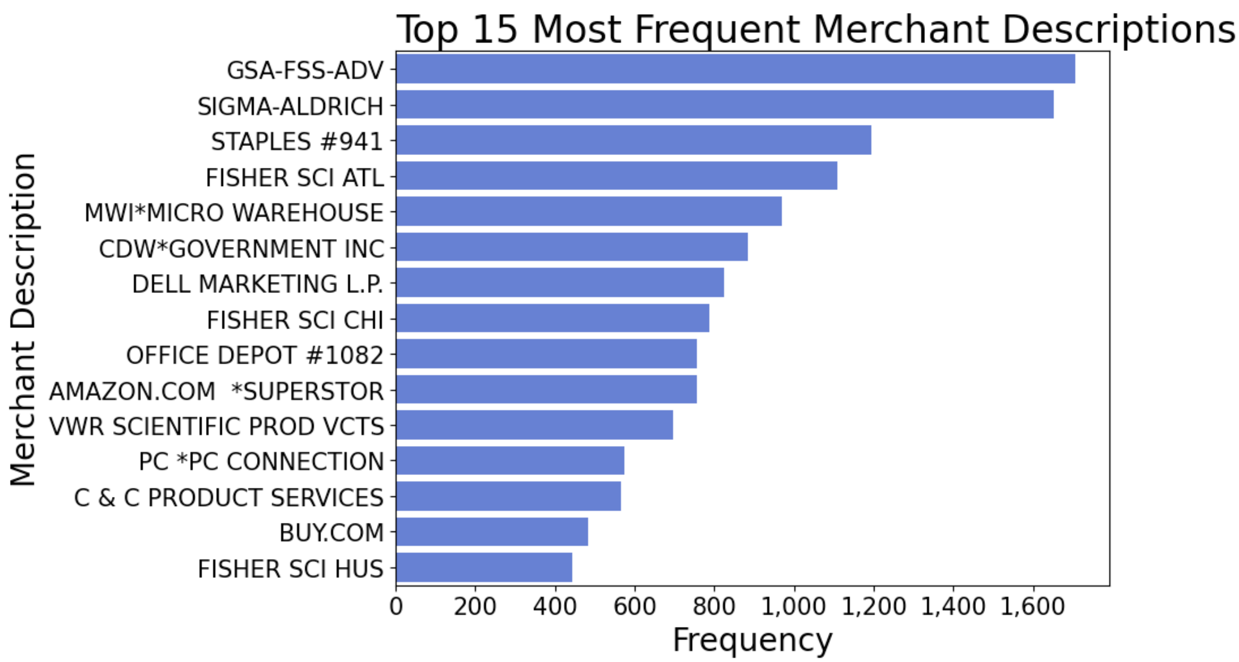
4) Field Name: Merchnum

Description: Identification number of the merchant. The distribution shows the top 15 field values of merchant's identification number. The most common merchant is the one with identification number 930090121224, with over 9'419 transactions in 2010.



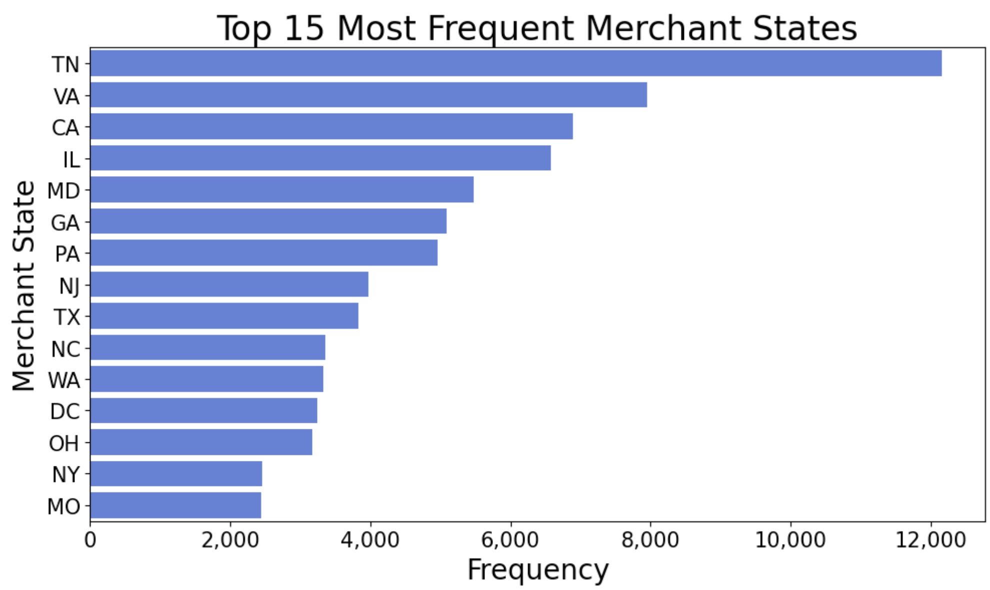
5) **Field Name: Merch description**

Description: Merchant's description. The distribution shows the top 15 field values of merchant's description. The most common description is 'GSA-FSS-ADV', with a total of 1'706 transactions.



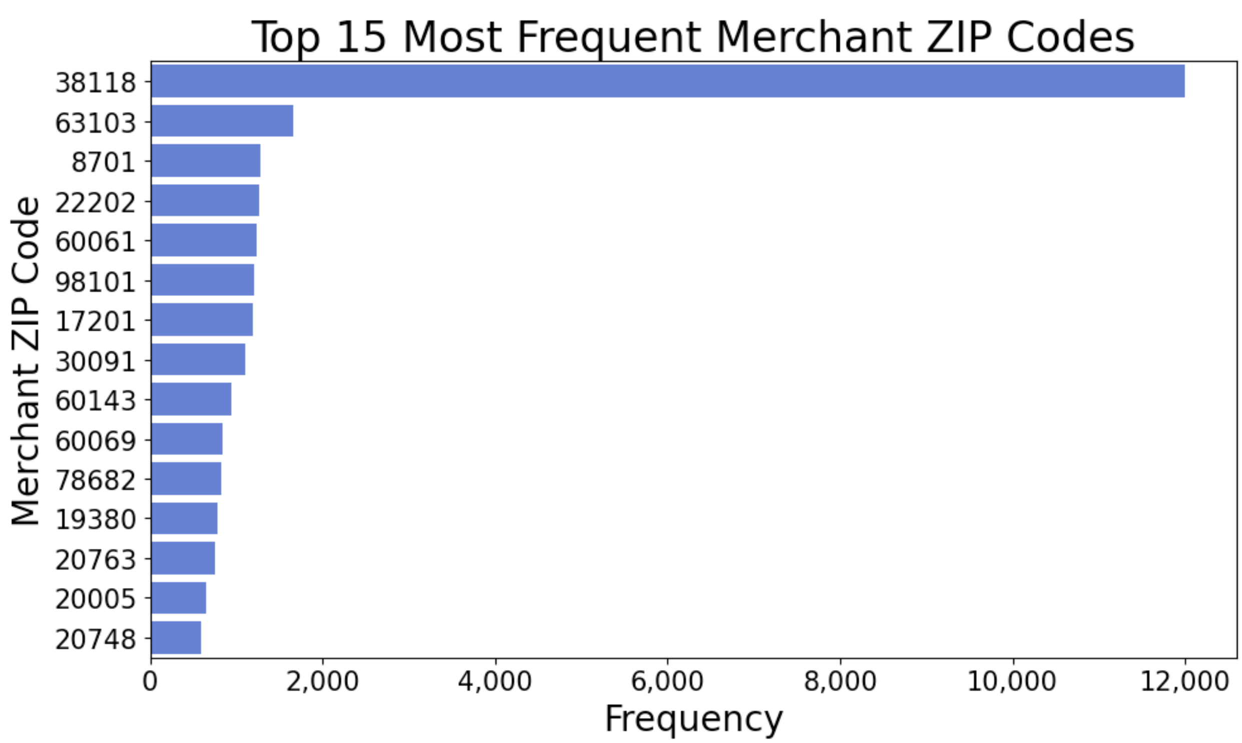
6) **Field Name: Merch state**

Description: State in which the merchant operates. The distribution shows the top 15 field values of merchant's state. The most common state is Tennessee, with a total count of 12'169 transactions.

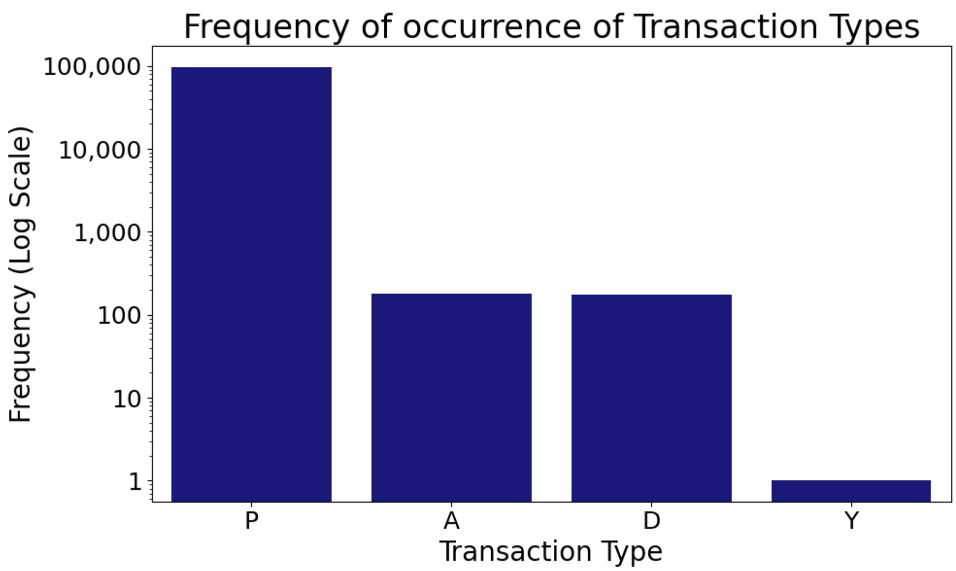


7) **Field Name: Merch Zip**

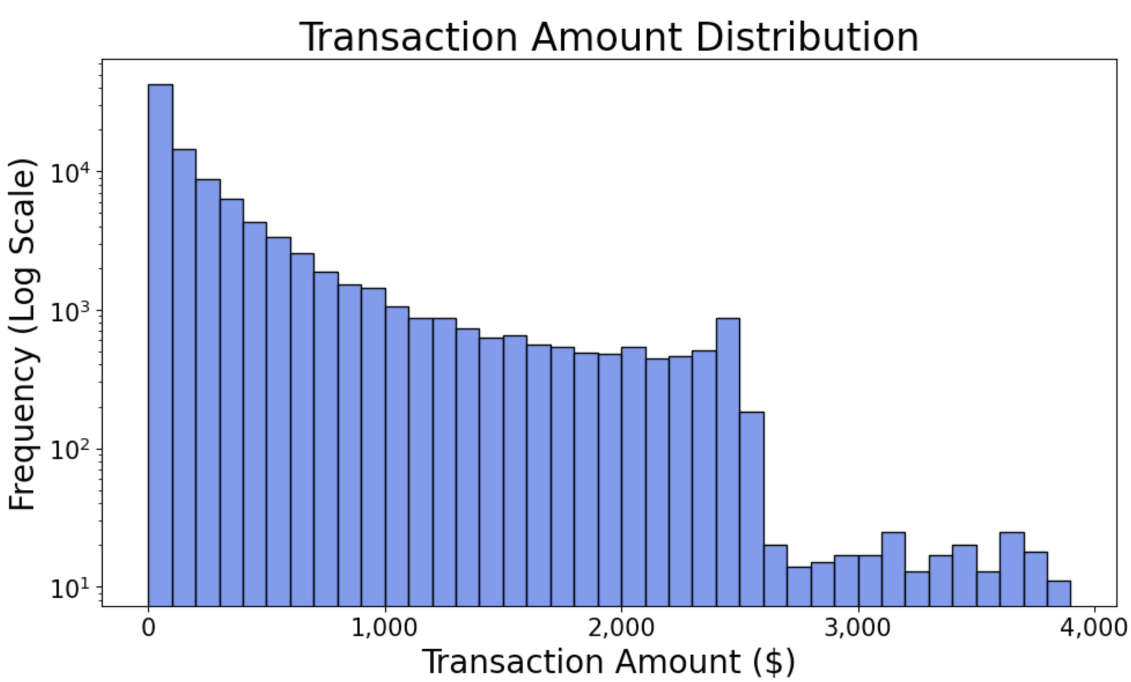
Description: Merchant's zip code. The distribution shows the top 15 field values of merchant's zip code. The most common zip code is 38118, with a total count of 11'998.



8) **Field Name: Transtype**
Description: Type of transaction. The distribution shows the distributions of transaction types. The most common transaction is “P”, with a total count of 97’497.



9) **Field Name: Amount**
Description: Amount of the transaction. The distribution shows the distribution of the transaction amounts. We can see that the distribution is moderate right-skewed, meaning that larger transactions are less frequent.



10) Field Name: fraud

Description: Fraud identification label. Fraud = 0 (Not fraudulent), Fraud =1 (Fraud identified). The total count of fraud_label = 0 is 95’805. The total count of fraud_label = 1 is 2’047.