Project 1

TRANSACTION FRAUD REPORT

Duyen Tran

MGTA 463 – Fraud Analytics

Table of content

Executive summary

Credit card fraud continues to be a significant challenge in the financial sector, with evolving methods of fraud necessitating advanced analytics to safeguard consumer transactions. Effective management of credit card transactions and fraud detection involves the systematic monitoring of transaction patterns and the identification of anomalies that could indicate fraudulent activities.

Fraud Analytics leverages a combination of statistical, machine learning, and artificial intelligence techniques to identify potentially fraudulent transactions. It uses historical data to train models that can predict and flag transactions as suspicious based on deviations from established spending patterns. Key techniques in fraud analytics include anomaly detection, predictive modeling, network analysis, and clustering algorithms, all aimed at identifying fraudulent transactions in near real-time.[1]

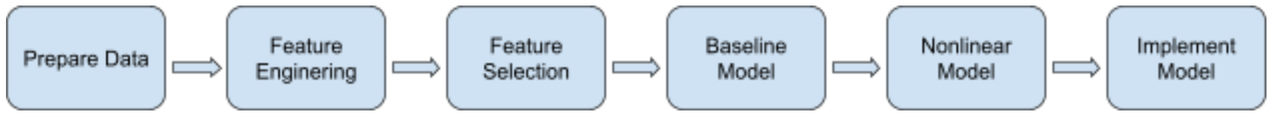


Figure 1: Workflow diagram of this report for predictive modeling

The figure 1 shows the workflow diagram throughout this project. In this project, we developed the machine learning model to identify the fraudulent transactions within credit card data from a synthetic dataset representing real U.S transactions over a year. By targeting a False Discovery Rate (FDR) of 3% on out-of-time (OOT) data, we estimate a significant reduction in financial looses due to fraud. This report outlines the data processing, feature engineering, model selection and final model performance, aiming to provide actionable insights with minimal technical jargon. The XGBoost model shows the best fraud detection result (FDR = 52,26%) for this particular dataset, which help predict the maximum possible savings is $48,444,000 at 5% cut-off.

I. Data description

1. Data overview

The dataset contains transaction records from credit card usage. It originates from a synthetic dataset representative of real U.S. credit card transactions over 10 years. The dataset comprises **97,852 records and 10 fields**, covering transactions from **01/01/2010 to 12/31/2010**. The fields are a mixture of numeric and categorical types, including identifiers, transaction details, merchant information, and a fraud indicator:

- Record: A unique identifier of each data record, from 1 to 97,852. This field also represents the time orders
- Date: The transaction's date, is formatted as a string (e.g., ‘1/1/10’).
- Cardnum: The account number for the transaction.
- Merchnum: A typically 12-digit merchant identification number involved in the transaction.
- Merch Description: A textual description of the merchant.
- Merch State: The state of the address for the merchant.
- Merch Zip: The state in which the merchant is located
- Transtype: A code denoting the type of transaction.
- Amount: The dollar amount of the transaction.
- Fraud: A binary indicator (0 or 1) denotes whether the transaction was fraudulent or not

2. Statistical Tables

Table 1 and 2 bellow show the summary information about all fields. There are Date and Amount are numeric type field whereas the other fields are categorical or text. There fields have some missing values: Merchnum, Merch state and Merch zip. It was noticed that the number of unique values of the Merch state field is 227, which is unexpected because the U.S has only 50 states. Some of values in field might be from toher countries.

Table 1: Numeric Fields Table

Field Name	**#Records Have Values**	**% Populated**	**# Zeros**	**Min**	**Max**	**Mean**	**Std Dev**	**Most Common Value**
Date	97,852	100.00%	0	1/1/10	12/31/10	NA	NA	2/28/10
Amount	97,852	100.00%	0	0.01	3,102,045.53	425.47	9949.8	3.62

Table 2: Categorical Fields Table

Field Name	**# Records with Values**	**% Populated**	**# Zeros**	**# Unique Values**	**Most Common Value**
Recnum	97,852	100.00%	0	97,852	1
Cardnum	97,852	100.00%	0	1,645	5142148452
Merchnum	94,455	96.52%	3,397	13,091	930090121224
Merch description	97,852	100.00%	0	13,126	GSA-FSS-ADV
Merch zip	92,149	95.19%	4,703	4,568	38118
Merch state	96,649	98.77%	1,203	227	TN
Transtype	97,852	100.00%	0	4	P
Fraud	97,852	100.00%	95,805	2	0

Bellow we show some further information about the data:

Figure 2 shows the transaction amount by class (Fraud and Non Fraud). For Fraud 0 (Bule), there are the most transactions are relatively small and densely populated near the lower end of the amount scale whereas Fraudulent transactions are visible throughout but are particularly noticeable at higher amounts, where they are more sporadic.

Figure 3 shows the total count of **Fraud = 0 is 95805**, and the total count of **Fraud = 1 is 2047.**

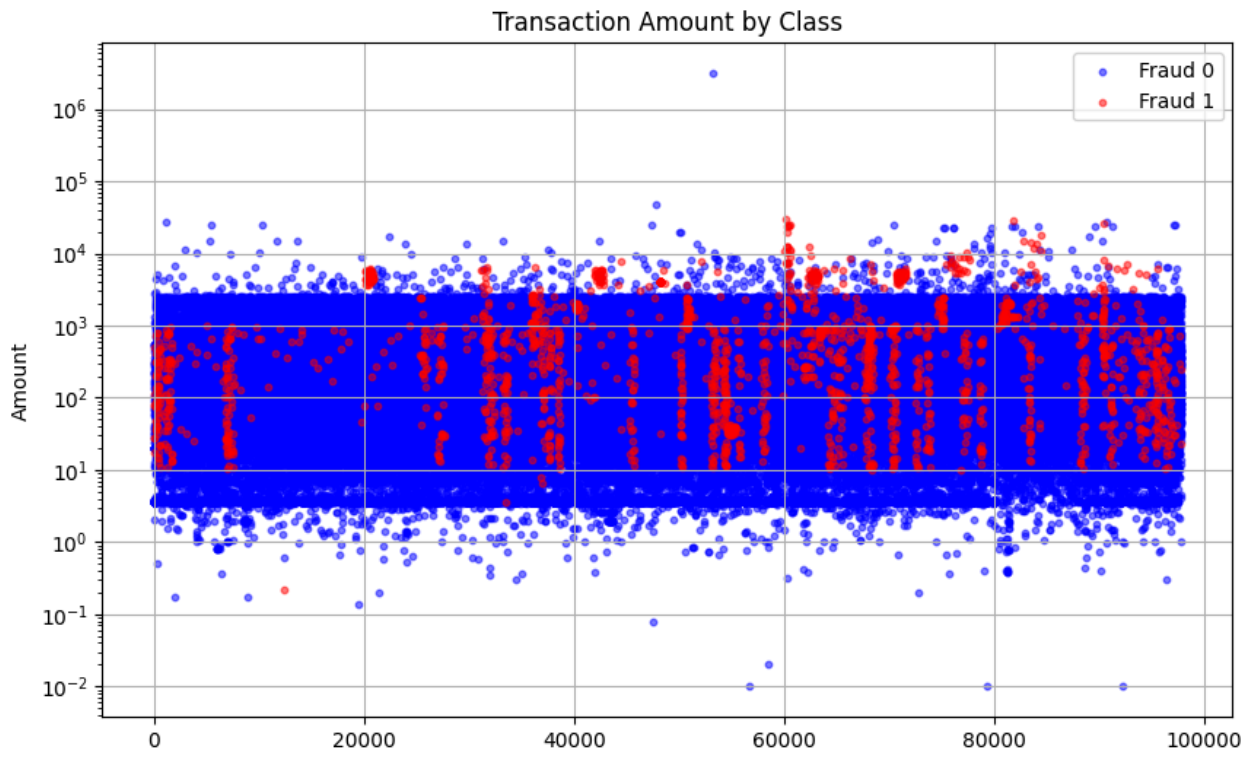


Figure 2: Data exploration based on two main classes of Data

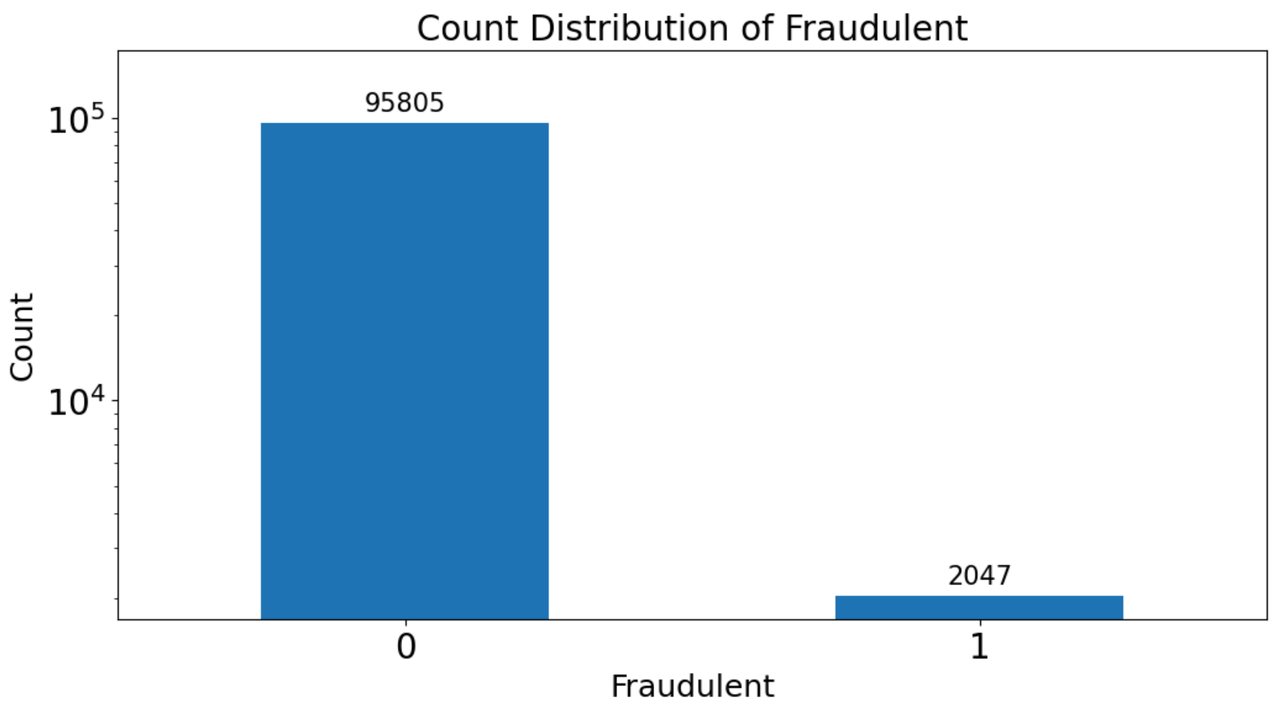


Figure 3: Data exploration bashed on the record number

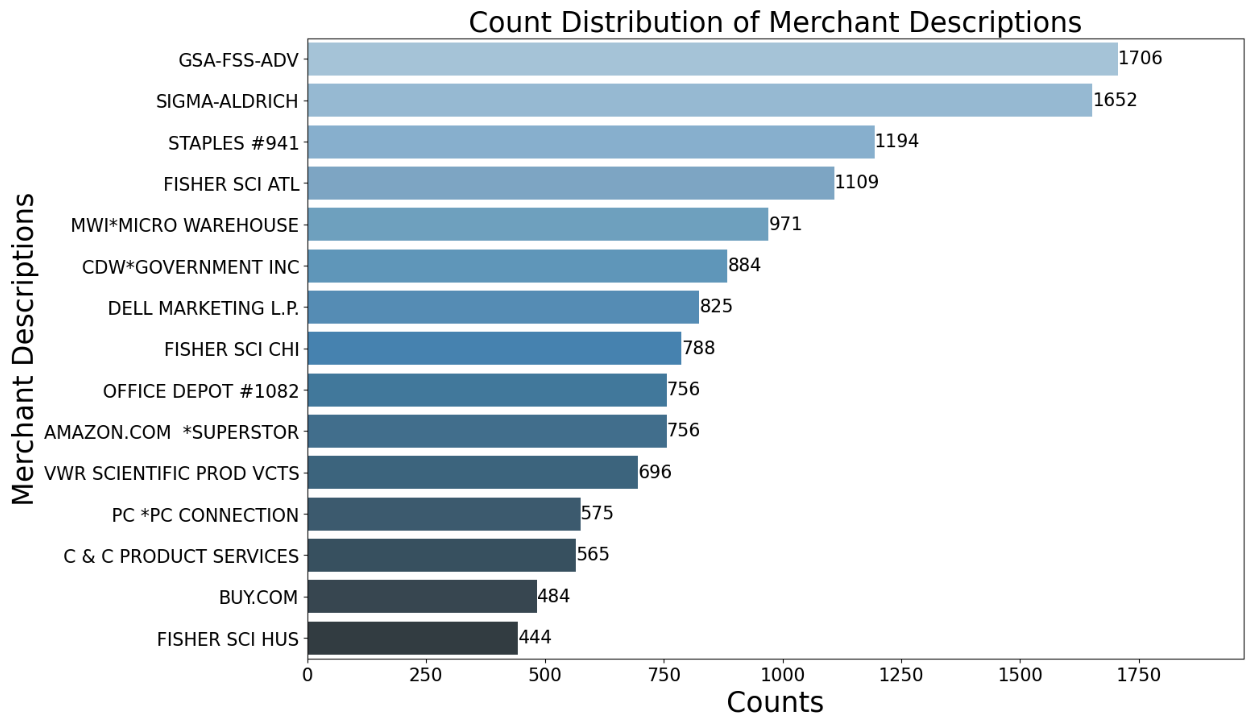


Figure 4: Most common merchant descriptions

Figure 4 shows the top 15 of the most transactions for different merchants based on the merchant descriptions. GSA-FSS-ADV is the most frequented merchant with 1,706 transactions, SIGMA-ALDRICH follows with 1,652 transactions.

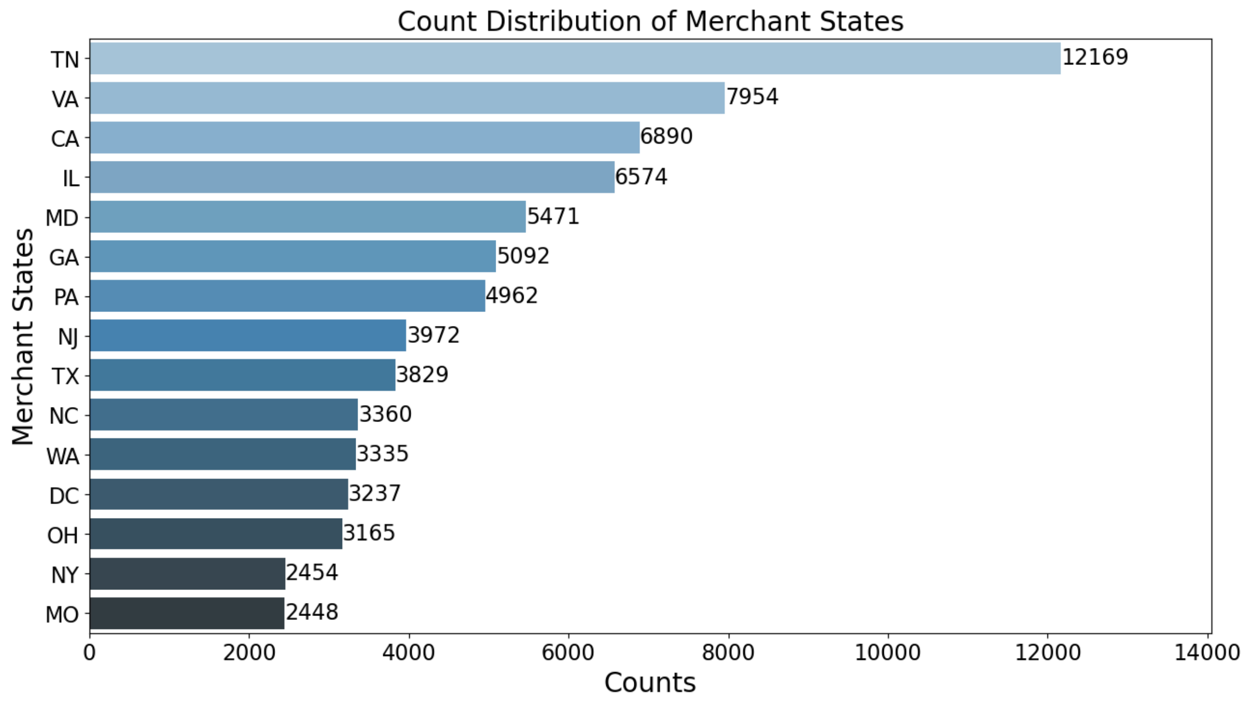


Figure 5: Most common States

Figure 5 shows the distribution of the top 15 of transactions across different merchant states. It presents the number of transactions that have occurred in each state, allowing us to observe where the most activity is taking place. Tennessee has the highest number of transactions with 12,169. Other states with high transactions counts include Virginia and California with counts of 7,954 and 6,890 respectively.

II. Data cleaning

The goald of this step is to clean the data and prepare it for the training phase of the classifier. In general, data in reality are noisy. Therefore, a cleaning step is necessary. In the contex of the cleaning process, the procedure is as follows:

1. Outlier Management:

Identification: During our initial data analysis, we identified an extreme outlier within the transaction amounts. This particular record showed a transaction amount exceeding $3,000,000. Such a value is highly unusual within the context of typical credit card transactions recorded in our dataset.

Action Taken:

After discovering this outlier, we consulted with the business manager to better understand its context. The decision to remove this record was made because it was determined to be non-representative of normal transactional behavior. This action was crucial to prevent the model from being skewed by this extreme value, ensuring that our predictive modeling efforts would be based on more typical transactional patterns, thus enhancing the overall accuracy and reliability of our analysis

2. Exclusions:

Overview of Transaction Types: The dataset included records of four distinct transaction types identified by the codes: P (Purchase), A (Adjustment), D (Deposit), and Y (Withdrawal). Each type represents a different kind of transactional activity.

Decision on Exclusions:

After a thorough review and consultations with business managers, it was decided to focus the analysis exclusively on transactions of type P (Purchase). This decision was informed by the managerial strategy that aimed to optimize fraud detection mechanisms specifically for purchase transactions, which are the most relevant for our current business objectives. Consequently, transactions coded as A, D, and Y were excluded from the dataset.

This exclusion helps streamline our data analysis and modeling efforts towards the transactions most susceptible to fraud, thereby aligning our resources and strategies with business priorities. The exclusion of non-purchase transactions ensures a focused analysis on the area of highest impact and relevance, improving the effectiveness of our fraud detection models.

3. Imputation

Table 3: Fields with missing values

Field	**Records with missing values**
Merchnum	3,397
Merch state	1,203
Merch zip	4,703

Table 3 shows the information about the fields with missing values. The three fields in question are closely linked to the "Merch description" field, indicating a strong association. However, it is important to note that identical "Merch description" entries may correspond to different values across these three fields. Additionally, these fields are interrelated, which justifies the use of the mode (the most frequently occurring value) of each respective field to impute the missing data.

a. Merchnum

Initial condition: There are 3279 records with missing **“Merchnum”**

Strategy:

- Merch Description Mapping: Use the Merch description field to fill in the most appropriate Merchnum for that Merch description. This resolved 1,164 missing records

- Specific Description: For entries with **“Merch description”** as “RETAIL CREDIT ADJUSTMENT” and “RETAIL DEBIT ADJUSTMENT”, assigned **“Merchnum” as “**unknown”, addressing 694 records
- Unique Assignment: For the remaining 1,421 records, each unique “Merch description” (515 in total) was assigned a unique **“Merchnum”,** ensuring all records now have a non-blank **“Merchnum”** field.

b. Merch state

Initial condition: There are 1,028 records with missing **“Merch state”**

Strategy:

- External Data Integration: Performed a left join with the USPS zip code database, matching **“Merch zip”** with **“zip”** in the database, resolving 57 records
- Zipcode Mapping: Used internal **“Merch zip”** to fill in the most appropriate **“Merch state”** for that **Merch zip,** filled 17 missing records
- Description of State mapping: Use mapping to fill in missing **“Merch state”** values based on the corresponding **“Merch description”**, resolving 667 records.
- Special Cases Handling:
  + Set Merch state to "unknown" for entries with "Retail Credit/Debit Adjustment," though it did not resolve any new cases.
  + Assigned "foreign" to non-US states, improving geographical data consistency.
- Default to Unknown: The remaining 287 records were set to “unknown”, ensuring no blanks in **“Merch state”**

c. Merch zip

Initial condition: There are 4,347 records with missing **“Merch zip”**

Strategy:

- External Data Integration: Performed a left join with the USPS zip code database, matching **“Merch state”** with **“state”** in the database, resolving 3,188 missing records

- Merch Description to zip mapping: For transactions with identifiable patterns in **"Merch description",** mapped to known **"Merch zip",** resolving 381 cases

- Handling Specific Descriptions: Directly set **"Merch zip"** to "unknown" for entries related to "RETAIL CREDIT ADJUSTMENT" and "RETAIL DEBIT ADJUSTMENT" where a specific ZIP code was not applicable.

- Merchnum to zip Mapping: Mapped **"Merchnum"** to corresponding **"Merch zip"** based on existing data patterns, filling in 302 more gaps.

- Utilization of Common ZIP Codes Based on State: For remaining cases, employ the most common zip codes corresponding to each **Merch state** to impute the missing **Merch zip**, fully addressing all missing **Merch zip**.

- Correction of Implausible Merch zip: Post-imputation, identified and corrected implausible ZIP codes (such as 1-digit ZIPs) by setting them to "unknown" to maintain data validity.

III. Variable creation

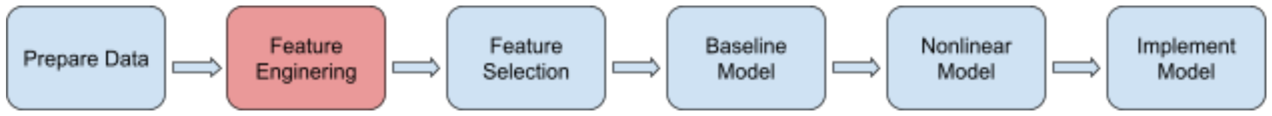


Figure 6: Workflow of diagram - Feature Engineering

The creation of a comprehensive set of variables is crucial for detecting potential fraudulent activities within credit card transactions. The variables are designed to capture different dimensions of transaction data such as temporal patterns, geographic consistency, transaction frequency, and monetary characteristics. These variables aim to highlight anomalies that might indicate fraudulent behavior.

1. Entities.

a. Importance of entites

In fraud detection, the creation of variables based on entities is essential for developing sophisticated and nuanced analytical capabilities. It enables deeper insights into individual and group behaviors, supports customized risk management, and enhances the overall strategic approach to preventing and detecting fraudulent activities. This entity-centric approach not only improves operational responses to fraud but also helps in adapting to evolving fraud tactics over time. Here is why entities are crucial in creating meaningful variables for fraud detection:

Behavioral Analysis: By examining transactions at the entity level, we can identify unusual patterns such as sudden increases in transaction frequency or size, which may indicate fraud.

Risk Segmentation: Entities allow for the segmentation of transaction data based on risk profiles. This segmentation aids in applying tailored fraud detection strategies that optimize resources and enhance detection accuracy.

Customized Controls: Using entity-specific variables supports the implementation of customized monitoring systems. High-risk entities can be subjected to stricter controls while maintaining a smoother experience for low-risk entities.

Collusive Fraud Detection: Entity analysis helps in identifying complex fraud schemes involving multiple parties, which might not be evident when viewing data in aggregate.

Improved Predictive Modeling: Entity-specific data enhances predictive models by allowing them to incorporate unique behavioral profiles, increasing their precision in distinguishing between fraudulent and legitimate transactions.

b. Entities list

There is a list of 23 entites:

'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'.

2. Variables creation

Table 4: Variables creation

Description	# Variables Created	# Cumulative Count
Original Fields The primitive fields in the dataset	10	10
Date of week target encoded Display the day of the week and its corresponding the average fraud percentage of that day	1	11
Geographic Variable Distance between consecutive transactions for each card	1	12
New Zip Count To track how often a new zip code appears for each entity which is if the zip code changed from the previous transaction and 'new zip count' counting these changes per day for each entity	23	35
Velocity Change Captures the rate of change in transaction amounts over successive transactions for each entity	23	58
Recency Days since the transaction by the entities was seen	23	81
Average Transaction Amount The average transaction amount over the last 7 days of entities	23	104
Daily Cumulative Amount Summing up the amount spent in a day can highlight days with unusually high activity	23	127
Daily Transaction Count of daily transactions for each entity	23	150
MoM Change in Avg Amount the change in average transaction amount over a month compared to the previous month for each entity	23	173

Transaction by Merchant Total number of transactions with the same merchant over the past 30 days	1	174
Transaction by day Total number of transactions on this day up to this transaction	1	175
Merchant Risk Scores Develop scores for merchants based on historical fraud rates	1	176
Frequency of transactions by Merchant Counting the number of transactions that occur at each merchant	1	177
Average Transaction Amount at Merchant The average transaction amount at each merchant	1	178
New Merchant Indicator A binary variable (0 and 1), denotes whether the transaction with a merchant occurred on the first recorded date or not	1	179
Merchant Return Frequency Calculate how often returns occur at each merchant	1	180
Location Consistency A binary variable checks whether most transactions at a merchant occur in a consistent geographic location or not	1	181
Card Centrality Measures the importance or influence of a particular card number within the entire network of transactions. Degree centrality counts the number of direct connections a node has, helping identify critical nodes that might be central to operational flows or potential fraud schemes	1	182
Small Amount Flag This feature identifies whether the transaction amount is unusually small compared to typical transactions seen for that card	1	183
Low-value Transaction Checking the frequency of low-value transactions	1	184

Cluster Label Identify natural groupings among transactions that might indicate different types of purchasing behavior, risk level, or fraudulent activity	1	185
PCA Represents the first/second principal component of the dataset derived through Principal Component Analysis. It helps identify underlying patterns in the transaction data that are not apparent in the raw data	2	187
Merch state_TE Captures the potential risk associated with transactions from different states	1	188
Merch zip_TE Capture the average rate for each zip code based on the training dataset	1	189
Dow_TE The average fraud rate for transactions that occurred on each particular day across the training data	1	190
Maximum Amount Variables: Maximum transaction amount for the given entity within (0,1,3,7,14,30,60) days for 23 entities	161	351
Median Amount Variables Median transaction amount for the given entity within 0,1,3,7,14,30,60 days for 23 entities	161	512
Total Amount Variables The sum of all transaction amounts for the given entity within (0,1,3,7,14,30,60 days) for 23 entities	161	673
Actual/Average Ratio The ratio of the current transaction amount to the average amount over ('7', '14', '30', '60' days)	161	834
Actual/Maximum Ratio The ratio of the current transaction amount to the maximum amount over ('7', '14', '30', '60' days)	161	995

Actual/Median Ratio The ratio of the current transaction amount to the median amount over ('7', '14', '30', '60' days)	161	1156
Actual/Total Ratio The ratio of the current transaction amount to the total amount over ('7', '14', '30', '60' days)	161	1317
Count by entity Represents the normalized ratio of the transaction count for day type 'd'(0 and 1) over a 1-day window, to the transaction over 'dd' day ('7', '14', '30', '60' days)	184	1501
Tota amount by an entity Represents the normalized ratio of the total transaction amount for day type 'd'(0 and 1) over a 1-day window, to the transaction over 'dd' day ('7', '14', '30', '60' days)	184	1685
Relative Velocity Represents the normalized velocity to days ratio, where the velocity is defined as the ratio of transaction counts for day type 'd'(0 and 1) over a 1-day window, to the transaction over 'dd' day ('7', '14', '30', '60' days)	184	1869
Average Transaction Variability Average difference in transaction amounts for the given entity over the past [0, 1, 3, 7, 14, 30] days.	138	2007
Maximum Transaction Variability Maximum difference in transaction amounts for the given entity over the past [0, 1, 3, 7, 14, 30] days.	138	2145
Median Transaction Variability Median difference in transaction amounts for the given entity over the past [0, 1, 3, 7, 14, 30] days.	138	2283
Amount_cat This variable splits transaction amounts into 5 bins with the same number of data points but potentially different ranges. The bins labeled from 1 to 5	1	2284
Foreign	1	2285

The binary variable (0 or 1) indicates whether that merchant is foreign or not.		

3. **Fraud detection conceptual**

- Temporal and Geographic Anomalies
  + Geographic Variable (Distance between consecutive transactions): Fraud often occurs when stolen card details are used in different locations rapidly, which this variable can flag by identifying transactions that are geographically too distant to be plausible within a short timeframe.
  + New Zip Count: Changing a zip code frequently can be indicative of a fraudster testing the card's validity across different merchant locations, often to avoid triggering fraud detection systems that monitor geographic consistency.
- Transaction Behavior Anomalies
  + Velocity Change: A sudden increase in the rate of transaction amounts can indicate card cloning or account takeover, where the fraudster is attempting to maximize the stolen card's limit.
  + Recency: If a card has not been used for a lengthy period, then suddenly has a flurry of activity, this might suggest it has been compromised and activated by fraudsters.
  + Daily Cumulative Amount & Daily Transaction Count: These variables help detect burst activity, which is a common fraud tactic where numerous transactions are processed in quick succession before the card is blocked.
- Merchant-Based Anomalies
  + Transaction by Merchant & Merchant Return Frequency: Repeated transactions or frequent returns at a single merchant might suggest collusion or refund fraud, where fraudsters manipulate transaction processes for financial gain.
  + Merchant Risk Scores: Merchants with a history of fraud incidents are more likely to be targets or participants in fraudulent schemes. This score helps prioritize scrutiny where it's most needed.
- Financial Deviations

+ Average/Median/Total/Maximum Amount Variables: Deviations from the norm in these financial variables can indicate manipulation or unauthorized usage, as fraudsters may attempt larger-than-usual transactions to maximize their gains.

+ Actual/Average/Maximum/Median/Total Ratios: These ratios highlight transactions that are out of character for the cardholder, which can be an indicator of fraud if the current transaction significantly deviates from historical patterns.

- Behavioral Indicators

+ Small Amount Flag & Low-value Transaction: Small amounts might be used initially to test the stolen card's validity before larger fraudulent transactions are made. This is a common first step in a broader fraudulent strategy.

+ Card Centrality: In a network of transactions, a central card may indicate a hub of fraudulent activity, especially if connected to other known fraud cases.

- Predictive and Analytical Insights

+ Cluster Label & PCA: These advanced analytical techniques help in segmenting transaction data into groups with similar patterns. Certain clusters may exhibit behaviors typically associated with fraud, such as similar amounts at similar times across multiple cards, suggesting coordinated fraud rings.

+ Foreign: Transactions that occur in foreign countries may be subject to less stringent checks and are often preferred by fraudsters looking to exploit these gaps.

- State and Zip Risk Levels

+ Merch state_TE and Merch zip_TE: Certain locations may have higher incidences of fraud based on historical data, and transactions from these states or zip codes might warrant additional scrutiny.

IV. Feature selection

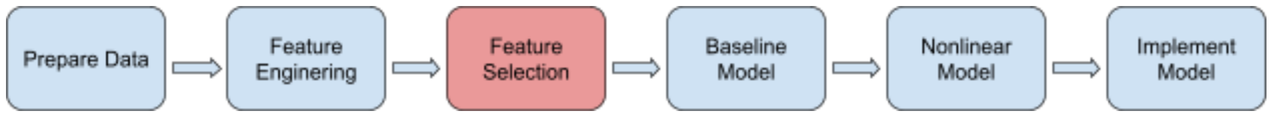


Figure 7: Workflow of diagram - Feature Selection

Feature selection in fraud detection is a critical process in which we identify the most relevant variables from our dataset to use in building predictive models. The goal is to enhance the model's performance by reducing overfitting, improving accuracy, and decreasing training time. Feature selection is particularly vital in fraud detection due to the complexity and high dimensionality of transaction data, which often includes irrelevant and redundant information that can obscure the significance of patterns of fraudulent behavior. [2]

In our endeavor to enhance the fraud detection capabilities of our predictive model, a critical step was the effective selection of most informative features from a dataset initially containing 2,285 variables. This section provides a detailed account of our feautre selection process, discussing the techniques used, the rationale of their selection and the implementation specifics that guided our approach from beginning to end.

In general, feature selection technique can be broadly classified into three categories:

- **Filter Methods:** These methods apply a statistical measure to assign a scoring to each feature. Features are ranked by the score and either selected to be kept or removed from the model. Common statistics include correlation coefficients, Chi-square test, mutual information, and Fisher's score.
- **Wrapper Methods:** These methods consider the selection of a set of features as a search problem, where different combinations are prepared, evaluated, and compared to other combinations. A predictive model is used to score each combination of features and determine which one performs best. Techniques include recursive feature elimination, sequential feature selection algorithms, and genetic algorithms.
- **Embedded Methods:** These methods perform feature selection during the model training process and are specific to given learning algorithms.

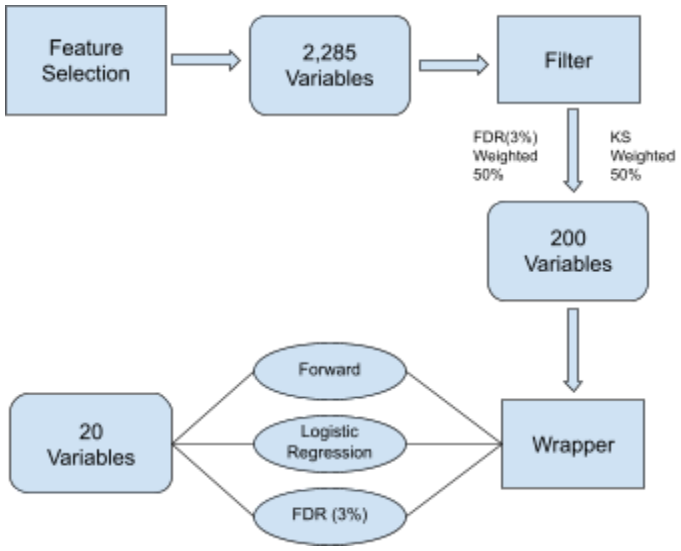


Figure 8 : Feature selection process

1. Description of the process

Initial Filtering:

The feature selection process began with the application of the Kolmogorov-Smirnov (KS) statistic, a powerful tool for determining the statistical significance between the distributions of fraudulent and non-fraudulent transactions across each feature. This stage aimed to reduce the vast array of features to a more manageable number by retaining only those with the highest ability to distinguish between the two classes.

Advanced Feature Selection:

Following the initial reduction, the process transitioned to more sophisticated feature selection techniques:

- Sequential Feature Forward Selection: Using a LightGBM model, this method involves starting with the most significant feature and iteratively adding features that offer the most substantial improvement to the model's performance.
- Wrapper Methods: These methods assess subsets of features based on their collective impact on the performance of the model, enhancing the predictive accuracy while avoiding redundancy.

2. Rationale.

Objective: To develop a robust fraud detection model by reducing the feature space from 2,285 to the most relevant 20 features without compromising the model's performance

The purpose of employing such rigorous feature selection methods was to enhance the model performance by:

- Reducing Overfitting: Fewer features reduce the complexity of the model, which help in minimizing overfitting and improving the generalizability of the model.
- Improving Acuracy: By focusing only on the most informative features, the model can achieve higher accuracy with a streamlined st of input variables.
- Enhancing Efficiency: Fewer features mean reduced computational resources are required, leading to faster training and prediction times, which is crucial for real-time fraud detection systems.
- Maintaining Model Interpretability: With fewer features, it is easier to understand and interpret the model's predictions, which is vital for gaining trust from stakeholders and for regulatory compliance.

3. Implementation

a. Filter phase: Feature selection using KS statistic

- *Data preprocessing*

Before feature scoring, the dataset was preprocessed to ensure its integrity for the selection process. Records that fell outside the project's frame were excluded, and a 'Random' feature was included to serve as a control

- *Balancing Data*

Given the typical imbalance between fraudulent and non-fraudulent transactions, a balancing step was incorporated, conditional on the balance parameter, to equalize the influence of both classes on the feature selection process

- *Feature scoring results*

Each feature was scored using KS statistic, resulting in the following initial filter scores. We identified a set of features with the highest scores, which indicates a stronger ability to distinguish between the transaction classes

Filter Outcome: Reducing the features from 2,285 to 200 features with top KS scores

b. Wrapper selection phase

Model: LightGBM Classifier configured with 10 estimators and 3 leaves

Feature selection Strategy: Sequential Forward Selection (SFS) method, focusing on maximizing the FDR at a 3% cutoff

Outcome: From 200 features, further reduced to the top 20 features which demonstrated the highest predictive power and contribution to the model performance.

Table 5: Key Features Identified

Wrapper Order	**Variable**	**Filter Score**
1	Cardnum_unique_count_for_card_state_1	0.476067
2	Card_Merchdesc_State_total_7	0.324668
3	Cardnum_count_1_by_30	0.428229
4	Cardnum_max_14	0.318826
5	Card_dow_vdratio_0by60	0.48648
6	Card_dow_vdratio_0by14	0.479086
7	Merchnum_desc_State_total_3	0.308586
8	Card_Merchdesc_total_7	0.324631
9	Card_dow_unique_count_for_merch_zip_7	0.418943

10	Cardnum_actual/toal_0	0.47955
11	Card_dow_vdratio_0by7	0.467961
12	Cardnum_vdratio_1by7	0.466766
13	Cardnum_unique_count_for_card_state_3	0.46641
14	Cardnum_unique_count_for_card_zip_3	0.464323
15	Merchnum_desc_Zip_total_3	0.305656
16	Cardnum_unique_count_for_Merchnum_3	0.460748
17	Cardnum_actual/toal_1	0.459715
18	Cardnum_unique_count_for_card_state_7	0.445967
19	Cardnum_actual/max_0	0.445726
20	Card_dow_unique_count_for_merch_state_1	0.447357

Implementation Details:

+ Feature importance Evaluation: Post selection, feature importance was revisited to insure the model leverages the most informative predictors.
+ Model stability: Evaluated through additional runs and tests to ensure consistent performance across various sets and conditions (see more in Apendix 3, 4)
+ Deployment considerations: Prepared for integration into existing transaction processing system, with considerations for real-time analysis and rapid scoring capabilities

V. Model exploration

This process details the outcomes of preliminary modeling efforts aimed at identifying optimal hyperparameter configurations for various machine learning models in the context of fraud detection. The objective was to minimize overfitting while maximizing out-of-time (OOT) performance to ensure robustness and reliability. The exploration included models such as Logistic Regression, Decision Trees, Random Forest, LGBM, Catboost, Neural Networks, and XGBoost, each tested across multiple iterations with varying settings (Table 6). Shortly, we will start with a baseline logistic regression and then a number of nonlinear statistical/machine.

Table 6: Model Exploration

Model	Parameters								Average FDR 3%			
Logistic Regression	**Itergration**	**Penalty**	**C**		**Solver**		**L1_ratio**	**max_iter**	**Train**	**Test**	**OOT**	**Note**
	1	l2	0.1		lbfgs		none	100	0.6796	0.6874	0.6874	
	2	l2	0.1		saga		none	200	0.6804	0.6831	0.4663	
	3	**l1**	**0.05**		**saga**		**none**	**400**	**0.6789**	**0.6859**	**0.4710**	Highest OOT
	4	**elasticnet**	**0.02**		**saga**		**0.5**	**300**	**0.6789**	**0.6859**	**0.4700**	Smallest Gap + High OOT
	5	elasticnet	0.5		saga		1	200	0.6811	0.6801	0.4680	
	6	l2	0.5		saga		none	200	0.6802	0.6805	0.4667	
	7	l2	1		lbfgs		none	1000	0.6818	0.6818	0.4663	
Decision Tree	**Itergration**	**max_depth**		**min_samples_split**		**min_samples_leaf**		**max_leaf_nodes**	**Train**	**Test**	**OOT**	**Note**
	1	None		2		1		10	0.6930	0.6784	0.4606	
	2	**7**		**100**		**50**		**40**	**0.7454**	**0.7267**	**0.4946**	
	3	30		100		50		50	0.7776	0.7351	0.5391	
	4	**10**		**120**		**60**		**60**	**0.7540**	**0.7194**	**0.5397**	Highest OOT
	5	7		1000		30		10	0.6761	0.6773	0.4640	
	6	300		60		500		1000	0.6826	0.6751	0.4798	
Random Forest	**Itergration**	**citerion**	**n_estimator**	**max_depth**	**min_samples_split**	**min_samples_leaf**	**max_features**	**bootstrap**	**Train**	**Test**	**OOT**	**Note**
	1	gibi	100	none	2	1	none	TRUE	1.0000	0.8142	0.5165	
	2	**entropy**	**200**	**10**	**40**	**2**	**sqrt**	**TRUE**	**0.8667**	**0.7870**	**0.5710**	Highest OOT
	3	gini	150	5	300	30	log2	FALSE	0.7216	0.7220	0.4838	
	4	entropy	300	15	6	1	0.5	TRUE	1.0000	0.8257	0.5620	
	5	gini	100	20	200	100	0.25	TRUE	0.7300	0.7270	0.4923	
	6	**gini**	**100**	**none**	**200**	**100**	**none**	**TRUE**	**0.7533**	**0.7402**	**0.5104**	Smallest Gap + High OOT
LBGM	**Itergration**	**num_leaves**	**n_estimators**	**max_depth**	**min_data_in_leaf**	**learning_rate**	**bagging_fraction**	**boosting_type**	**Train**	**Test**	**OOT**	**Note**
	1	40	300	3	10	0.1	0.8	gbdt	0.8712	0.7920	0.5222	
	2	64	200	10	50	0.05	0.7	dart	0.9011	0.8047	0.5327	
	3	256	500	20	100	0.005	0.9	goss	0.9121	0.8019	0.5562	
	4	**4**	**none**	**30**	**none**	**0.01**	**none**	**gbdt**	**0.7452**	**0.7332**	**0.5374**	**Smallest Gap + High OOT**
	5	10	30	5	30	0.02	0.6	gbdt	0.7331	0.7171	0.5135	
	6	20	40	7	70	0.01	0.9	goss	0.7655	0.7377	0.5596	Highest OOT
Catboost	**Itergration**	**l2_leaf_reg**	**n_estimators**	**max_depth**	**min_data_in_leaf**	**learning_rate**	**verbose**		**Train**	**Test**	**OOT**	**Note**
	1	3	100	6	60	0.1	0		0.9703	0.8211	0.5212	
	2	**6**	**500**	**8**	**60**	**0.02**	**0**		**0.7718**	**0.7532**	**0.5077**	Smallest Gap + High OOT
	3	3	200	6	30	0.05	0		0.9265	0.8203	0.5374	
	4	3	1000	4	20	0.03	0		0.8345	0.7937	0.5135	
	5	4	150	6	100	0.1	0		0.9629	0.8238	0.5222	
	6	**3**	**250**	**5**	**50**	**0.07**	**0**		**0.9245**	**0.8117**	**0.5263**	Highest OOT
Neural Network	**Itergration**	**solver**	**hidden_layer_sizes**	**activation**	**alpha**	**learning_rate**		**max_iter**	**Train**	**Test**	**OOT**	**Note**
	1	**adam**	**(10, 12)**	**relu**	**0.0001**	**constant**		**200**	**0.7400**	**0.7295**	**0.5040**	Smallest Gap + High OOT
	2	adam	(3,2,5)	relu	0.005	adaptive		300	0.7061	0.6972	0.4808	
	3	lbfgs	(10,15)	tanh	0.01	constant		40	0.7330	0.7184	0.4993	
	4	**lbfgs**	**(40,50)**	**tanh**	**0.0001**	**adaptive**		**100**	**0.8057**	**0.7642**	**0.5108**	Highest OOT
	5	sgd	(20,30)	logistic	0.005	constant		300	0.6399	0.6518	0.4367	
	6	adam	(10,12)	relu	0.001	constant		200	0.7372	0.7219	0.4909	
XGB	**Itergration**	**booster**		**max_depth**		**learning_rate**		**n_estimator**	**Train**	**Test**	**OOT**	**Note**
	1	**gbtree**		**3**		**0.1**		**100**	**0.7873**	**0.7710**	**0.5226**	Smallest Gap
	2	gbtree		**6**		**0.05**		**30**	**0.7839**	**0.7538**	**0.5431**	Highest OOT
	3	gbtree		4		0.005		200	0.7317	0.7158	0.4933	
	4	gbtree		5		0.05		500	0.9300	0.8198	0.5357	
	5	gbtree		8		0.01		20	0.7771	0.7336	0.5212	
	6	gbtree		3		0.02		1000	0.8137	0.8160	0.5064	

1. Methodology

a. Baseline Model

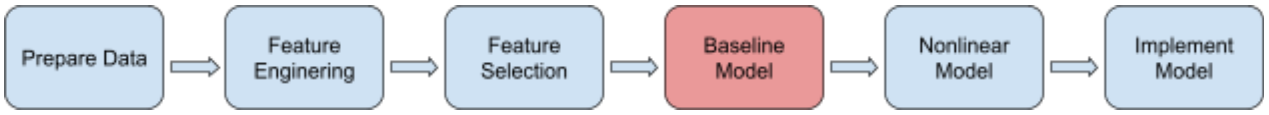


Figure 9: Workflow of diagram - Baseline Model

Logistic Regression

- Description: Logistic Regression is a statistical model that estimates the probability of a binary outcome. It predicts the probability that a given input belongs to a particular category. The model outputs probabilities by applying a logistic regression to a linear combination of input features. The logistic function ensures that the output value falls between 0 and 1, which makes it particularly suitable for modeling probability in binary classification tasks like fraud detection where outcomes are typically categorical (e.g., fraud or not fraud). The logistic function is defined as:

$$P = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\beta_3 X_3+...+\beta_n X_n)}}$$

Where:

P: The probability that the dependent variable y is 1 given predictors x.

β: The coefficients of the model.

x : Feature variables

In term of predicting model, logistic regression serves as excellent baseline model. Because it is straightforward to implement and interpret, it's often used as a starting point in predictive modeling. Comparing its performance with mode complex models can help validate the improvements offered by additional complexity.

Since fraud detection is typically a binary classification problem (fraud/no fraud), logistic regression is naturally suited for such analysis. It models the probability that each transaction is fraudulent, which is directly aligned with the goals of fraud detection.

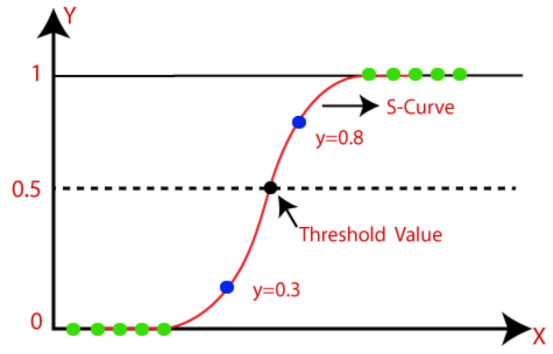
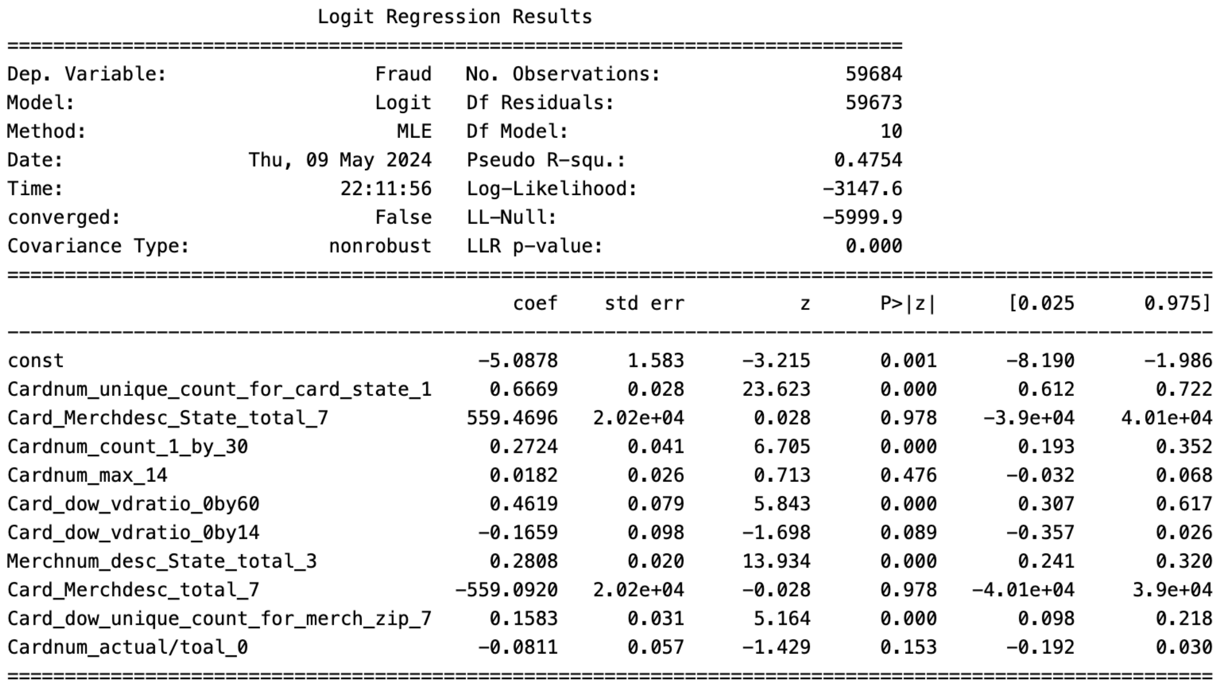


Figure 10: The concept of Logistic Regression Classification [3]

The figure 10 shows the fraud class takes in the value “1”, while the non-fraud class takes the value “0”. A threshold of 0.5 is used to differentiate between the two class. For this study, the first step is to build a baseline logistic regression using all variables from out feature selection process. Table 6 below is a summary of main parameters of this logistic regression model

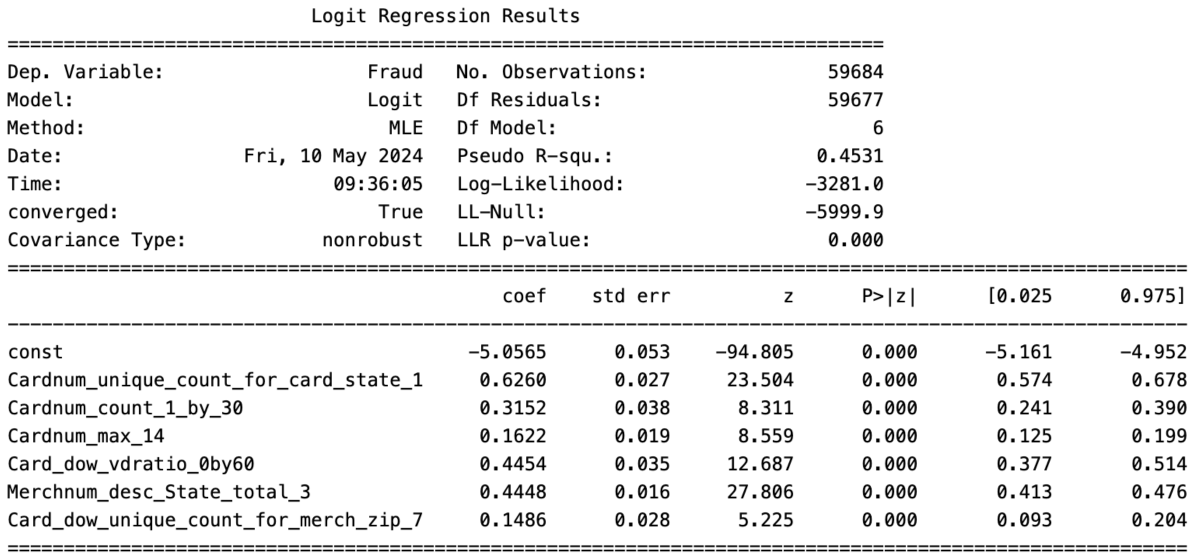
Table 6: Logistic regression output on the training data



```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Fraud   No. Observations:                59684
Model:                          Logit   Df Residuals:                    59673
Method:                           MLE   Df Model:                           10
Date:                Thu, 09 May 2024   Pseudo R-squ.:                  0.4754
Time:                        22:11:56   Log-Likelihood:                -3147.6
converged:                      False   LL-Null:                       -5999.9
Covariance Type:            nonrobust   LLR p-value:                     0.000
=========================================================================================================
                                          coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
const                                  -5.0878      1.583     -3.215      0.001      -8.190      -1.986
Cardnum_unique_count_for_card_state_1   0.6669      0.028     23.623      0.000       0.612       0.722
Card_Merchdesc_State_total_7          559.4696   2.02e+04      0.028      0.978    -3.9e+04    4.01e+04
Cardnum_count_1_by_30                   0.2724      0.041      6.705      0.000       0.193       0.352
Cardnum_max_14                          0.0182      0.026      0.713      0.476      -0.032       0.068
Card_dow_vdratio_0by60                  0.4619      0.079      5.843      0.000       0.307       0.617
Card_dow_vdratio_0by14                 -0.1659      0.098     -1.698      0.089      -0.357       0.026
Merchnum_desc_State_total_3             0.2808      0.020     13.934      0.000       0.241       0.320
Card_Merchdesc_total_7               -559.0920   2.02e+04     -0.028      0.978   -4.01e+04     3.9e+04
Card_dow_unique_count_for_merch_zip_7   0.1583      0.031      5.164      0.000       0.098       0.218
Cardnum_actual/toal_0                  -0.0811      0.057     -1.429      0.153      -0.192       0.030
=========================================================================================================
```

It is noticeable that "Card_Mechdesc_State_total_7", "Card_Merchdesc_total_7", "Cardnum_actual/toal_0" and "Card_dow_vdratio_0by14" have p values higher than 0.05, therefore, we excluded them out of the model since they are not statistically significant. The table 7 below is the output of final logistic regression

Table 7: Final Logistic Regression Output



```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Fraud   No. Observations:                59684
Model:                          Logit   Df Residuals:                    59677
Method:                           MLE   Df Model:                            6
Date:                Fri, 10 May 2024   Pseudo R-squ.:                  0.4531
Time:                        09:36:05   Log-Likelihood:                -3281.0
converged:                       True   LL-Null:                       -5999.9
Covariance Type:            nonrobust   LLR p-value:                     0.000
=========================================================================================================
                                          coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
const                                  -5.0565      0.053    -94.805      0.000      -5.161      -4.952
Cardnum_unique_count_for_card_state_1   0.6260      0.027     23.504      0.000       0.574       0.678
Cardnum_count_1_by_30                   0.3152      0.038      8.311      0.000       0.241       0.390
Cardnum_max_14                          0.1622      0.019      8.559      0.000       0.125       0.199
Card_dow_vdratio_0by60                  0.4454      0.035     12.687      0.000       0.377       0.514
Merchnum_desc_State_total_3             0.4448      0.016     27.806      0.000       0.413       0.476
Card_dow_unique_count_for_merch_zip_7   0.1486      0.028      5.225      0.000       0.093       0.204
=========================================================================================================
```

.

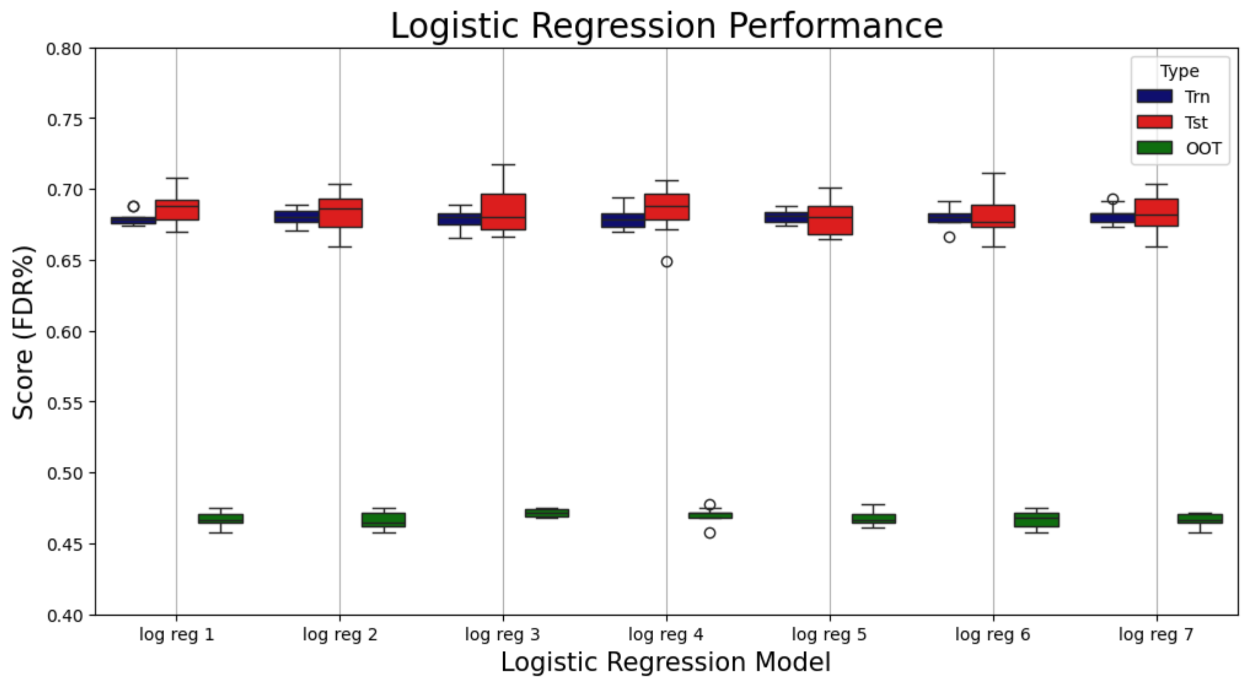


Figure 11: Performance of Logistic Regression Model

Figure 11 is a boxplot visualization taht shows the performance of servaral logistic regression model, identified as log reg 1 through log reg 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. Most models show relatively higher performance during training and slightly drops during test set, indicating how well the model generlizes to unseen data. It is noticable that setting 2 and setting 3 show more stability across different datasets (training, testing and OOT), which might make them preferable for deployment given their consistent performance

Table 8: Final choice of Logistic Regression Model

Hyperparamter			
Penalty	l1	L1_ratio	none
C	0.005	max iter	400
Solver	saga		

Table 8 shows the final hyperparameter choice of Logistic Regession. Here is the breakdown if each hyperparameter:

- Penalty (l1): This specifies the norm used in the penalization. The L1 norm encourages sparsity in the coefficients (many coefficients become zero). Using L1 penalty can help in feature selection by eliminating some features entirely, as it applies a cost proportional to the absolute value of the coefficients.

- C (0.005): This is the inverse of regularization strength. Smaller values specify stronger regularization. In logistic regression, C controls the amount of shrinkage: the larger the C, the less the model is regularized. A smaller C (such as 0.005) increases the regularization effect, which can help prevent overfitting but might lead to underfitting if too small.

- Solver (saga) : This solver is an extension of sag (Stochastic Average Gradient descent) that also supports the L1 penalty. It is often the solver of choice for large datasets as it is efficient and can handle both L1 and L2 penalties. The saga solver is effective in scenarios where there are many features or when data is sparse.

- L1_ratio (none): Typically, L1_ratio is used with elasticnet penalty, specifying the mix between L1 and L2 penalties. A ratio of 1 corresponds to an L1 penalty, 0 to L2, and values in between correspond to a combination of L1 and L2. The term none here might indicate that this parameter is not used or relevant unless elasticnet is specified as the penalty.

- Max Iter (400): This parameter defines the maximum number of iterations taken for the solvers to converge. A higher number of max_iter might be necessary for convergence in cases where the optimization algorithm requires more iterations to find the minimum of the loss function, especially for complex or very large datasets.

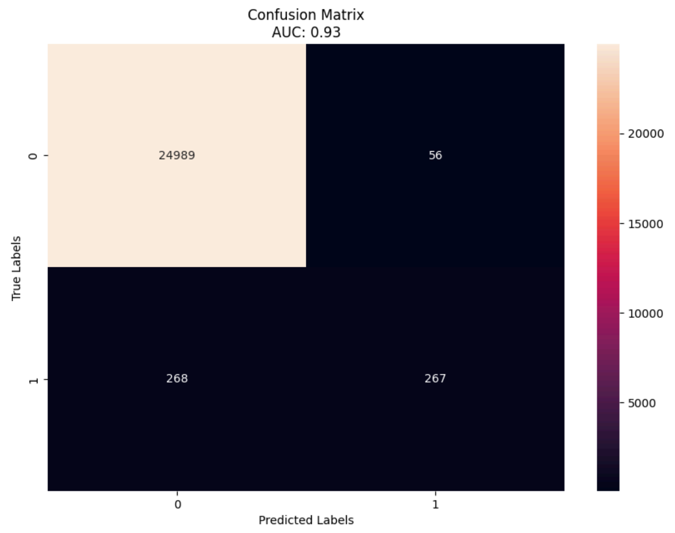


Figure 12: Confusion matrix of final choice logistic regression model

b. Nonlinear Model

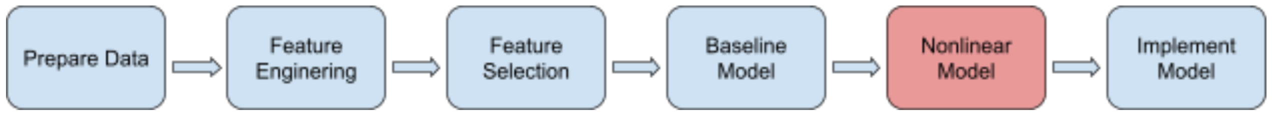
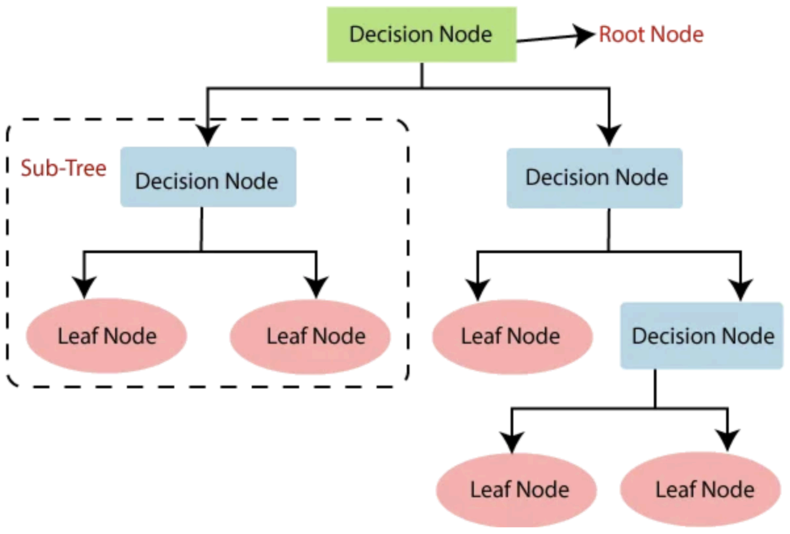


Figure 12: Workflow of diagram - Nonlinear Model

Decision Trees

A Decision tree is a flowchart-like structure in which each internal node represents a test on an attribute(e.g.,whether a transaction amount is above a certain threshold), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The path from root to leaf represents classification rules. A Decision Tree is a model that makes decisions by splitting data into branches based on feature values. Each branch ends in a leaf that represents a class or prediction, resembling a flowchart where each decision point is based on a feature.



-

Figure 13: Basic concept of Decision Tree Classification [4]

Why Use Decision Trees in Fraud Detection?

+ Handling of Complex Patterns: Decision trees can model complex relationships between features without requiring transformation of variables or assumption about the distribution of the data, unlike logistic regression or other parametric methods.

+ Feature Selection: In constructing a decision tree, the algorithm automatically selects the features that are most useful for classifying transactions. This inherent feature selection can help identify the most effective predictors of fraud.

+ Versatility in Data Types: They can handle both numerical and categorical data. Fraud detection datasets often contain a mix of data types, which decision trees can integrate without extensive preprocessing.

+ Robustness to Outliers: Decision trees are relatively robust to outliers in the dataset. Since each node focuses on a subset of data while splitting, outliers have less influence on the overall model, unless they are numerous enough to appear consistent in their behavior across splits.

+ Ease of Scaling and Updating: Trees can be easily updated with new data using algorithms designed for decision tree induction without rebuilding from scratch, making them suitable for dynamic environments like fraud detection where patterns can change over time.

+ Probabilistic Classification: While decision trees make hard classifications, they can also estimate class probabilities based on the proportion of samples of each class in the leaves, providing a measure of certainty in predictions which is useful for making decisions in ambiguous cases.

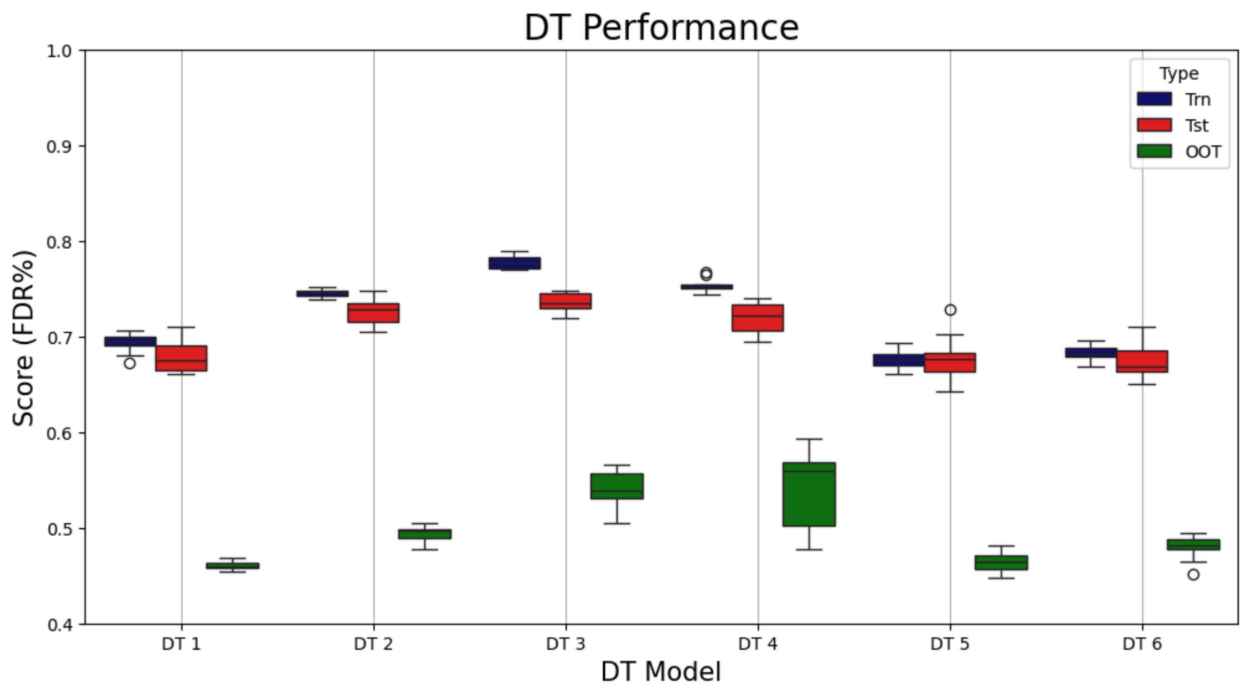


Figure 14: Performance of Decision Tree Model

Figure 14 is a boxplot visualization taht shows the performance of servaral Decision Tree model, identified as DT 1 through DT 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. Models that show closer performance across training and test set are generalizing. DT4 and DT5 seem to generalize better compared to others whereas DT2 and DT 4 have higher OOT compared to others.

Table 9: Final choice of Decision Tree Model

Hyperparamter			
max_depth	7	min_samples_leaf	50
min_samples_split	100	max_leaf_nodes	40
criterion	entropy	splitter	best

Table 9 shows the final hyperparameter choice of Decision Tree. Here is the breakdown if each hyperparameter:

- Max_depth (7): This hyperparameter controls the maximum depth of the tree. The depth of a tree is the length of the longest path from a root node down to the farthest leaf node. Setting max_depth to 7 means that the tree can have at most seven splits from the root to the deepest leaf. This limits the complexity of the model to prevent overfitting.

- Min_samples_leaf (50): This hyperparameter specifies the minimum number of samples that must be present in a leaf node after splitting. Setting it to 50 means that each leaf node must have at least 50 instances, encouraging the tree to be more generalized.

- Min_samples_split (100): This hyperparameter defines the minimum number of samples required to split an internal node.Setting this parameter to 100 implies that no node will split into two child nodes unless there are at least 100 instances in the node to split, promoting a less complex and more robust model.

- Max_leaf_nodes (40): Specifies the maximum number of leaf nodes that the tree can have. Having more leaves allows the model to capture more information about the data but can also lead to overfitting. Setting max_leaf_nodes to 40 restricts the tree to have at most 40 leaves, therefore controlling the model's complexity and potentially enhancing its ability to generalize.

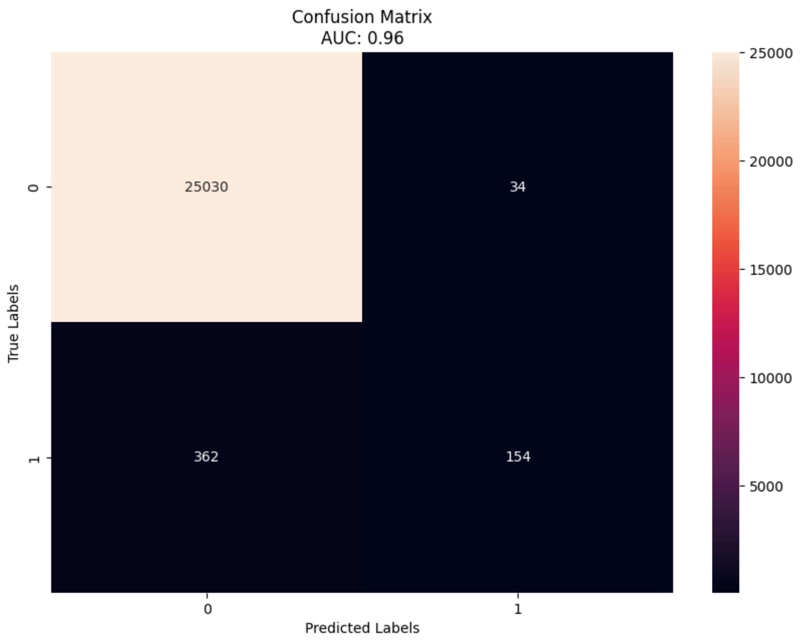


Figure 15: Confusion matrix of final choice decision tree model

Random Forest

Random Forest improves on the Decision Tree model by creating an ensemble of many trees and aggregating their predictions to decide the final output. This process is known as "bagging" or Bootstrap Aggregating.

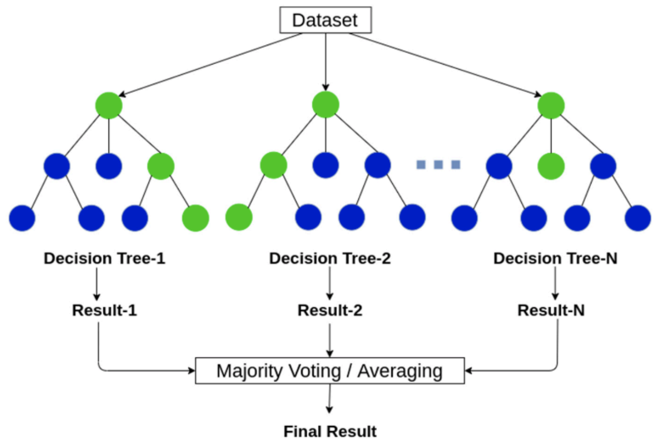


Figure 16: Random Forest Concept [5]

Based on figure 15, we can see how example is classifier where the final prediction is done by taking a vote from all n trees. The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class. Finally, the oob sample is then used for cross-validation, finalizing that prediction.

- Why use Random Forest in Fraud Detection:

  + Enhanced Accuracy and Stability: Random Forest inherently manages biases and variances by building multiple trees, making it significantly more accurate and stable than individual decision trees. This is crucial in fraud detection, where the cost of false predictions can be high.

+ Effective in Large and Complex Data Sets: Given the complex nature of fraud detection, which often involves large volumes of data and many input variables, Random Forest's capability to handle large and complex data sets makes it highly effective.

+ Feature Importance: It ranks the importance of different attributes for classification decisions. This feature importance score helps to identify significant predictors of fraud, contributing to more targeted and efficient investigative efforts.

+ Versatility in Data Handling: Random Forest can handle data with various types of variables (binary, categorical, or numerical). This versatility is particularly beneficial in fraud detection, where different types of data need to be analyzed together.

+ Minimizing Overfitting: While decision trees can overfit data, the ensemble nature of Random Forest minimizes this risk, making the model generalize better to new, unseen data.

+ Adaptive to Evolving Tactics: Fraudulent tactics evolve, and models need to adapt quickly. Random Forest models can be retrained with new data without starting the learning process from scratch, adapting more quickly to new patterns of fraud than many other algorithms.

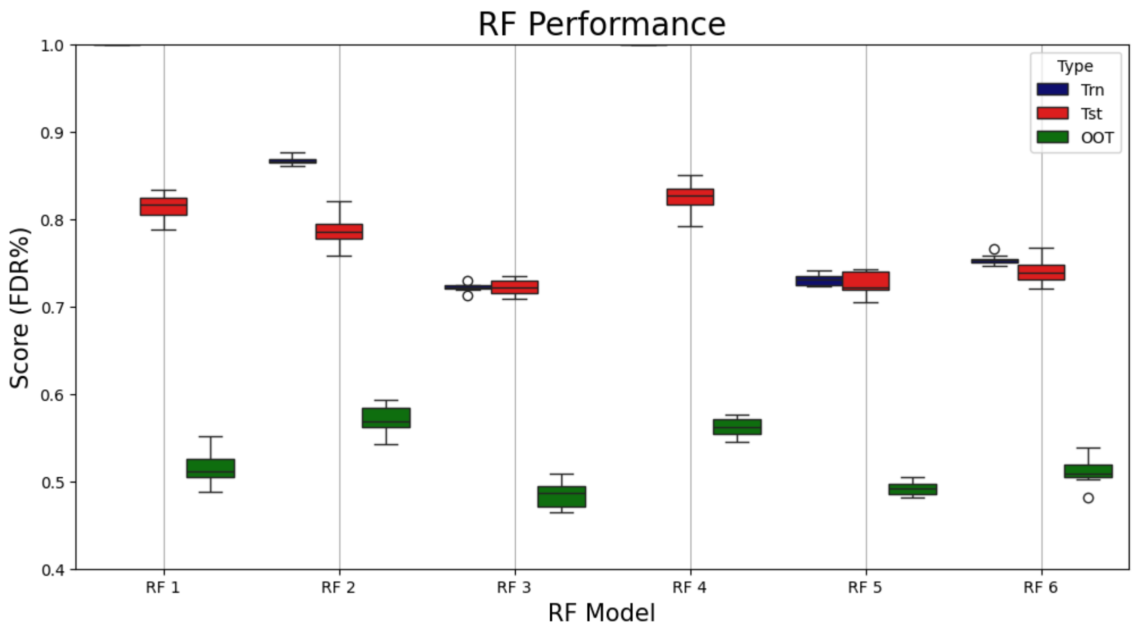


Figure 17: Performance of Random Forest Model

Figure 16 is a boxplot visualization taht shows the performance of servaral Random Forest model, identified as RF 1 through RF 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. It't noticeable that RF 5 and RF 6 seem to generalize better compared to others.

Table 10: Final choice of Random Forest Model

Hyperparamter			
n_estimators	100	min_samples_leaf	100
max_depth	none	min_samples_plit	200
criterion	gini	max_features	none
boostrap	True		

Table 10 shows the final hyperparameter choice of Random Forest Model. Here is the breakdown if each hyperparameter:

- N_estimators (100): This hyperparameter defines the number of trees in the forest. Increasing the number of trees increases the robustness of the model, as the ensemble's predictions are based on the average of the predictions from all trees. However, more trees also mean higher computational cost and diminishing returns after a certain point.
- Min_samples_leaf (100): Similar to a single Decision Tree, this parameter sets the minimum number of samples required to be at a leaf node. A larger min_samples_leaf results in a more significant constraint on the growth of each tree, reducing the risk of overfitting by ensuring that each leaf node generalizes well to new data.
- Max_depth (none): Specifies the maximum depth of each tree in the forest. If set to none, there is no limit on the depth of the trees, allowing them to grow until all leaves are pure or contain less than min_samples_split samples.
- Min_samples_split (200): This parameter determines the minimum number of samples required to split an internal node. Setting this parameter to a higher value can prevent the model from learning overly fine distinctions, thus reducing the model's variance but potentially increasing its bias.
- Criterion (gini): The function used to measure the quality of a split. gini refers to Gini impurity, a measure used for calculating the probability of a randomly chosen element being incorrectly classified.

- Max_features (none): The number of features to consider when looking for the best split. If none, then max_features=n_features is considered, meaning that all features are used to make each split in a tree. Setting this parameter can control how much each tree in the forest is randomized and how each tree is different from the others, which can influence the diversity within the forest.

- Bootstrap (True): Whether bootstrap samples are used when building trees. If True, each tree in the forest is built from a bootstrap sample (i.e., a sample taken with replacement) from the data. Using bootstrap sampling means that each tree in the forest is slightly different and can capture various aspects of the data, enhancing the model's generalization capability.

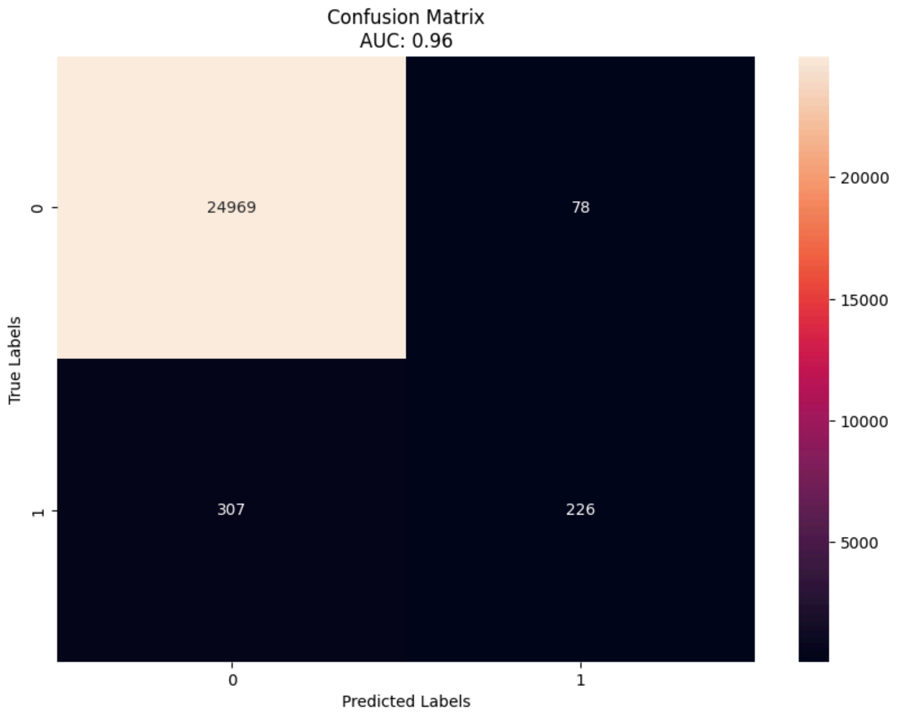


Figure 18: Confusion matrix of final choice Random Forest model

LGBM (Light Gradient Boosting Machine)

LGBM is an advanced type of Gradient Boosting algorithm that builds one tree at a time. Each new tree helps to correct errors made by the previous trees. It's designed to be efficient and faster, especially on large datasets. The "light" in LightGBM refers to its high speed and efficiency, which is achieved through techniques like Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), which reduce memory usage and increase the speed of computation.

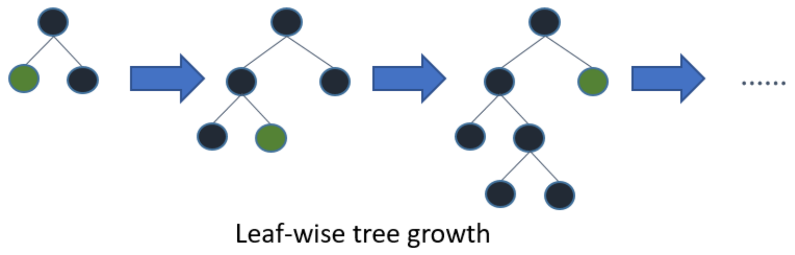


Figure 20: Tree growth workflow of LGBM model [6]

- Application in Fraud Detection

  + High Performance on Imbalanced Data: Fraud detection often deals with highly imbalanced datasets where fraudulent transactions are much rarer compared to non-fraudulent ones. LightGBM excels in scenarios with imbalanced data by focusing on harder cases that have larger gradients, thus improving model's ability to detect anomalies.

  + Efficient Processing of Large Datasets: In fraud detection, datasets often encompass millions of transactions. LightGBM's efficient use of system resources allows it to handle large volumes of data without the need for extensive hardware, making it a cost-effective solution.

  + Quick Iteration and Adaptability: Fraud patterns can change rapidly, requiring models that can be quickly retrained with new data. LightGBM's training efficiency enables faster retraining and adaptation to new fraud patterns as they develop.

  + Enhanced Predictive Accuracy: By leveraging gradient boosting, LightGBM can model complex nonlinear relationships within the data that are typical in fraud detection scenarios. This results in higher detection rates of fraudulent transactions.

  + Real-Time Fraud Detection: LightGBM's fast execution speed makes it suitable for real-time fraud detection systems, where decisions must be made quickly to prevent fraudulent transactions from being processed.

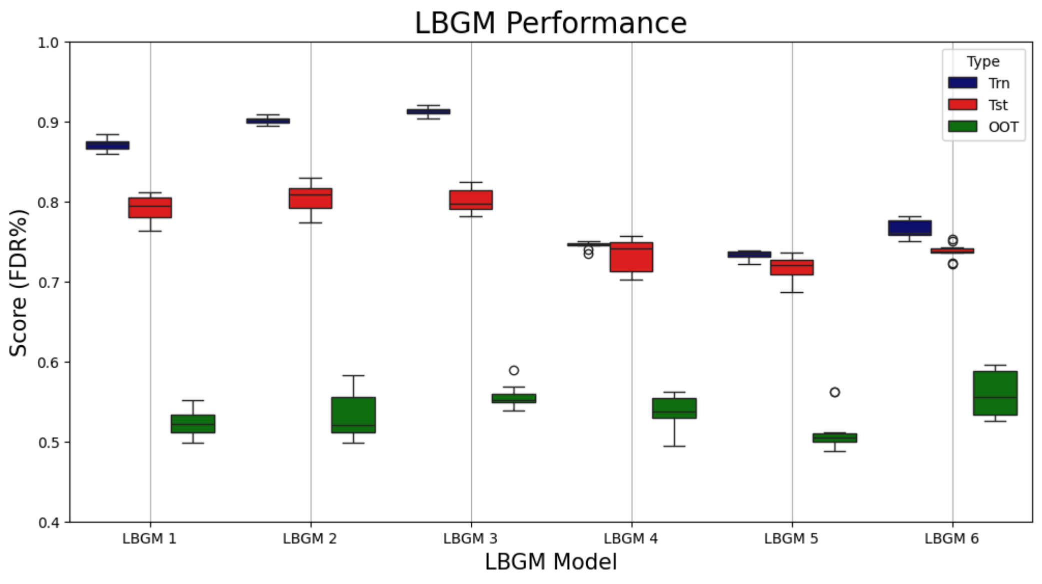


Figure 21: Performance of Light GBM Model

Figure 21 is a boxplot visualization taht shows the performance of servaral Light GBM model, identified as LGBM 1 through LGBM 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. It't noticeable that LGBM 5 and LGBM 4 seem to generalize better compared to others.

Table 11: Final choice of LGBM Model

Hyperparamter			
num_leaves	4	min_data_in_leaf	60
max_depth	30	learning_rate	0.02
boosting_type	gbdt		

Table 10 shows the final hyperparameter choice of LGBM Model. Here is the breakdown if each hyperparameter:

- Num_leaves (4): This hyperparameter sets the maximum number of leaves that can be formed in each tree. LightGBM constructs trees vertically, meaning that leaf-wise tree growth strategies are employed rather than level-wise tree growth strategies. A smaller number of leaves restricts the complexity of the model, reducing overfitting risks. It makes the model more robust by simplifying the learned structures.

- Min_data_in_leaf (60): This parameter is a threshold for the number of samples that must be present in a leaf. This is analogous to min_samples_leaf in scikit-learn's tree-based models. Setting this value higher can prevent the model from creating too fine-grained leaves, thereby controlling overfitting. It ensures that leaves have more than just a few data points, thus generalizing better.

- Max_depth (30): Specifies the maximum depth of trees being built. Despite LightGBM typically being leaf-wise, setting a max depth can help prevent overly deep and complex trees. A depth of 30 is quite high, which allows considerable model complexity. Depending on the dataset, this might need adjustment to avoid overfitting.

- Learning_rate (0.01): Also known as the shrinkage rate, this influences the rate at which the model learns. A lower learning rate requires more trees to be effective but can result in a more robust model as it gives more opportunity for the model to correct its mistakes in the previous trees. A learning rate of 0.01 is relatively low, leading to slower training but potentially better long-term accuracy and less risk of overfitting.

- Boosting_type (gbdt): Specifies the algorithm to use for boosting. gbdt stands for Gradient Boosted Decision Trees, a popular method that uses tree models and optimizes a differentiable loss function.This type is effective for a range of regression and classification problems, offering a good balance between speed and accuracy.

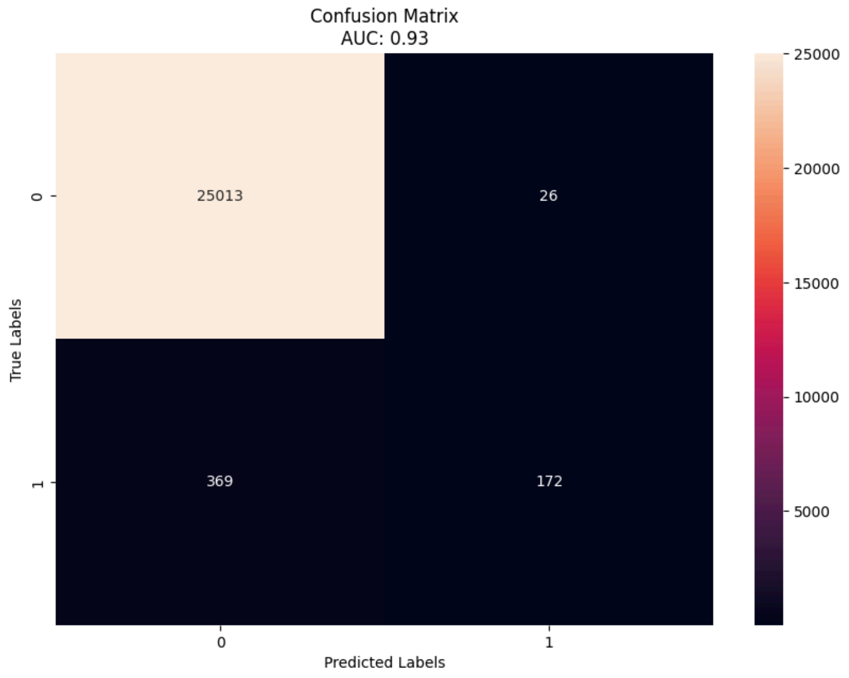


Figure 22: Confusion matrix of final choice LGBM model

Catboost

Catboost is a type of Gradient Boosting algorithm too but optimized to handle categorical variables directly. It transforms these variables in a way that allows the model to get more accurate results faster without extensive data preprocessing. The CatBoost algorithm is an improvement in the framework of the GBDT algorithm. It effectively solves the problem of gradient bias and prediction shift, avoids the occurrence of overfitting, and improves calculation accuracy and generalization ability (figure 23) The details are as follows.

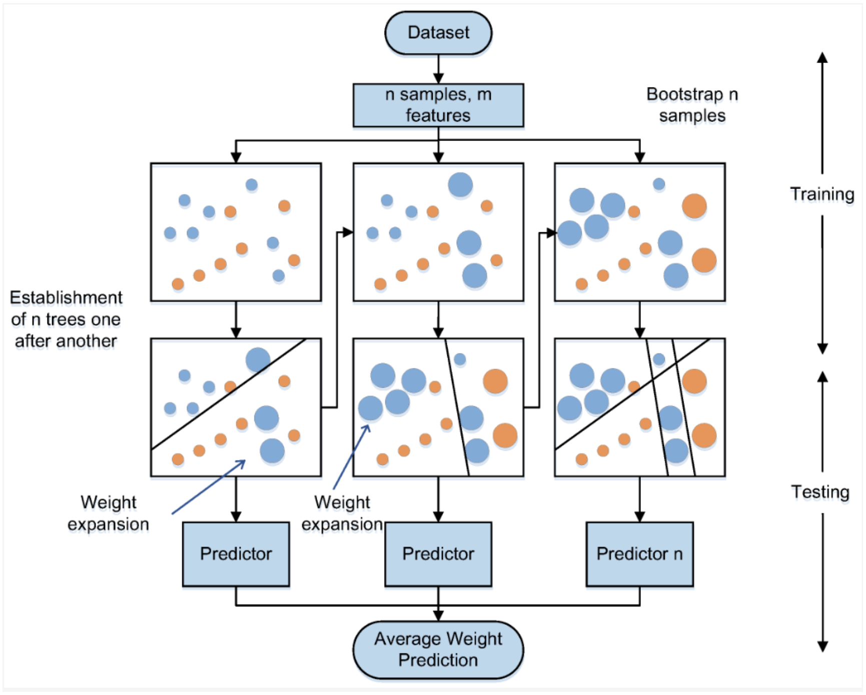


Figure 23: The structure of the Catboost algorithm [7]

- Application in Fraud Detection

  + Superior Handling of Categorical Data: Fraud detection systems often deal with various categorical features such as user IDs, merchant categories, and transaction locations. CatBoost's ability to handle categorical data directly is invaluable because it minimizes data leakage and preserves the integrity of the data's original structure.

  + Enhanced Predictive Performance: CatBoost achieves high accuracy and speed in training even on complex and noisy data, characteristics common in fraud detection datasets. Its sophisticated algorithms optimize the model's performance without extensive hyperparameter tuning.

  + Prevention of Overfitting: In fraud detection, where new, unseen data patterns frequently emerge, the robustness of CatBoost against overfitting ensures that models remain predictive over time and do not just memorize training data.

  + Adaptability to New and Emerging Fraud Patterns: CatBoost can be quickly updated with new data, allowing models to adapt to emerging fraud patterns. This adaptability is crucial for keeping the fraud detection system effective against the continually evolving tactics used by fraudsters.

  + Real-Time Prediction Capability: Given its efficiency and performance, CatBoost can be deployed in real-time environments where decisions need to be made rapidly, such as in online transaction processing.

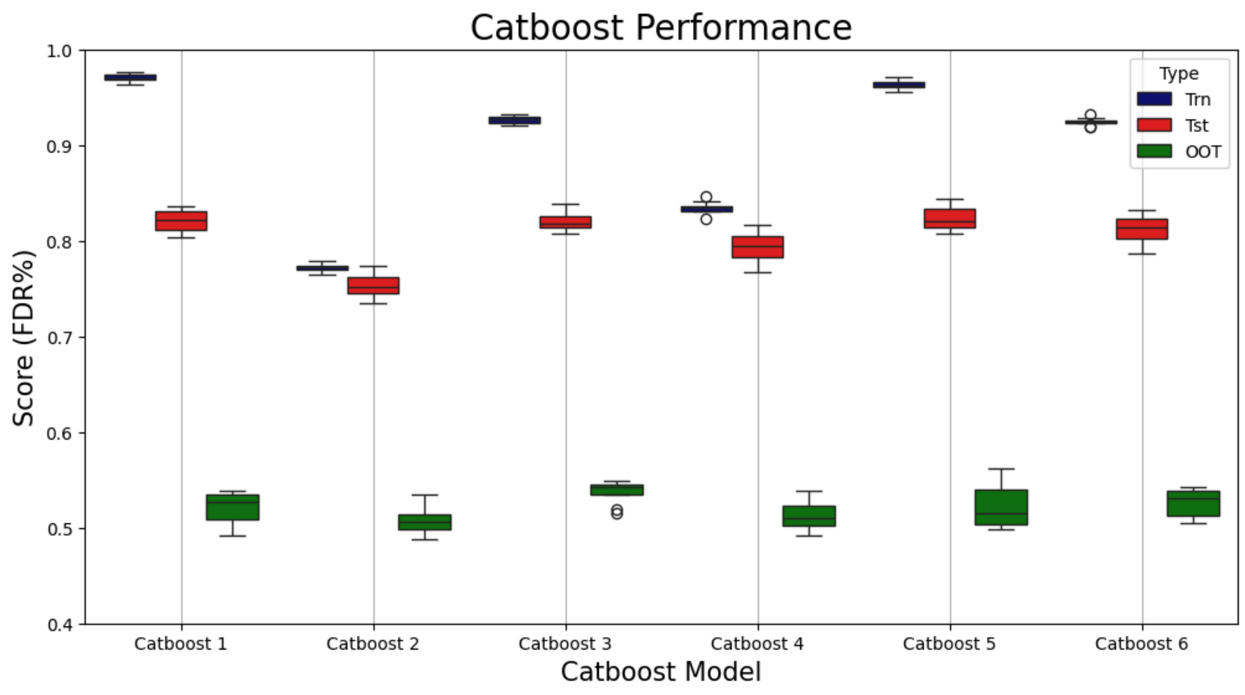


Figure 24: Performance of Catboost Model

Figure 24 is a boxplot visualization that shows the performance of servaral Catboost model, identified as Catboost 1 through Catboost 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. It't noticeable that Catboost 2 seem to generalize better compared to others.

Table 12: Final choice of Catboost Model

Hyperparamter			
l2_leaf_reg	6	min_data_in_leaf	60
n_estimators	500	learning_rate	0.02
max_depth	8	verbose	0

Table 12 shows the final hyperparameter choice of Catboost Model. Here is the breakdown if each hyperparameter:

- L2_leaf_reg (6): This parameter specifies the coefficient for L2 regularization on the weights. L2 regularization helps in reducing model complexity and fighting overfitting by penalizing large weights. Setting l2_leaf_reg to 6 means that the regularization term will effectively shrink the weights, thus preventing the model from fitting too closely to the training data.

- Min_data_in_leaf (60): Similar to other tree-based models, this parameter sets the minimum number of samples (data points) required to form a leaf. This helps in preventing the tree from growing too deep and complex, which can lead to overfitting. A minimum of 60 samples per leaf ensures that each decision made by the tree captures enough data to be statistically significant, enhancing the model's generalization capabilities.

- n_estimators (500): Defines the number of trees to be built in the model. More trees can improve the accuracy but also increase the risk of overfitting if not controlled by other parameters like learning_rate and max_depth. With 500 trees, the model is allowed to learn complex patterns from the data, but it requires careful tuning of other parameters to maintain a balance between bias and variance.

- learning_rate: 0.02: Determines the step size at each iteration while moving toward a minimum of a loss function. A lower value makes the boosting process more conservative, reducing the risk of overfitting by making the model more robust at the expense of needing more trees to converge. A learning rate of 0.02 helps ensure that each additional tree improves the model gradually, allowing for more fine-grained adjustments to the model.

- Max_depth (8): This parameter limits the depth of each tree. It's one of the most direct controls over the model complexity. A lower max_depth can significantly reduce the model's overfitting potential by limiting how detailed the learned patterns can be. A depth of 8 strikes a balance, allowing the model to learn complex patterns but not so deep as to fit to noise in the training data.

- Verbose (0): Controls the verbosity of the model's training process. A value of 0 means that the model will not output any training progress or potentially any other messages into the console, which is useful for keeping the output clean especially when automating the training process over many iterations. Setting verbose to 0 is particularly useful in production or when running multiple models in a batch process where output logs might clutter the process information.

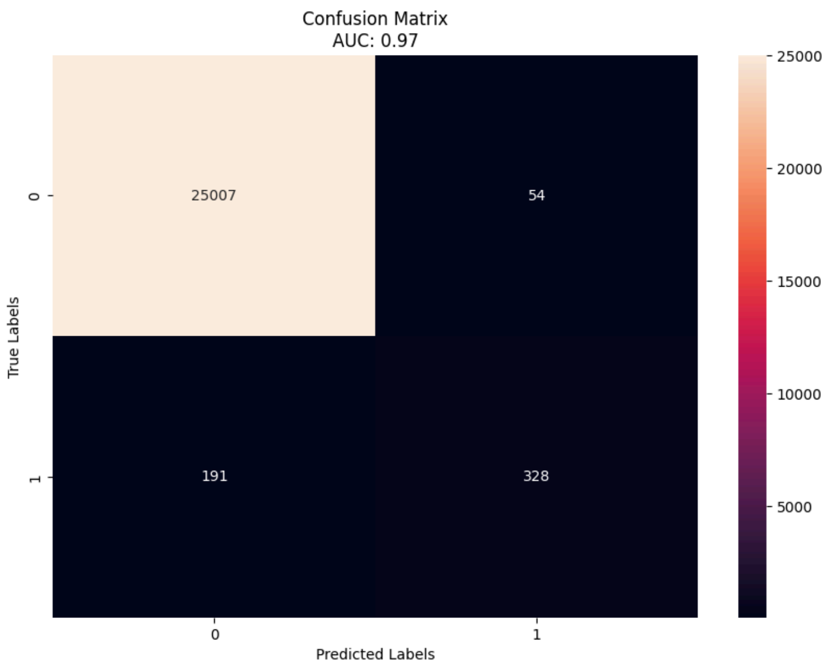


Figure 25: Confusion matrix of final choice Catboost model

Neural Networks

Neural Networks are inspired by the human brain and consist of layers of interconnected "neurons" that can learn complex patterns from data. They are highly flexible and can be used for a wide range of tasks from regression to classification and more advanced applications like image and speech recognition.

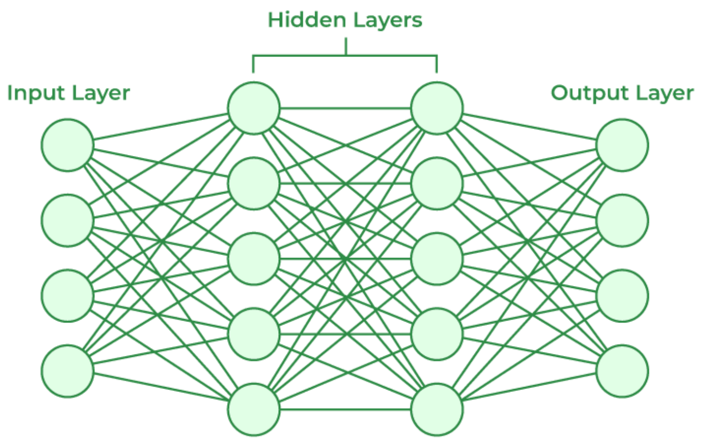


Figure 26: Neural networks architecture [8]

The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer. These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

- Why use it in fraud detection:
    + Handling High-Dimensional Data: Neural networks are particularly useful in fraud detection due to their ability to manage high-dimensional and diverse data from multiple sources. They can process and integrate various types of data such as transaction amounts, user behavior, and merchant information, all of which are critical in detecting fraudulent activities.
    + Learning Complex and Non-linear Relationships: The complexity of fraudulent transactions, which often involve subtle and non-linear interactions between variables, can be effectively modeled with neural networks. Their layered structure enables them to learn even the most intricate patterns that might indicate fraud.
    + Real-Time Processing: Neural networks, especially those designed for efficiency such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), can process information in real-time, making them ideal for applications that require immediate fraud detection, such as credit card transactions or online banking.
    + Continuous Learning and Adaptation: With capabilities to update their weights and biases in response to new data, neural networks can adapt over time, learning from new fraudulent tactics as they develop and thus continuously improving their predictive accuracy.
    + Anomaly Detection: They are particularly effective in anomaly detection, which is a core component of fraud detection. Neural networks can be trained to recognize 'normal' patterns and, subsequently, to flag deviations from these patterns as potential fraud

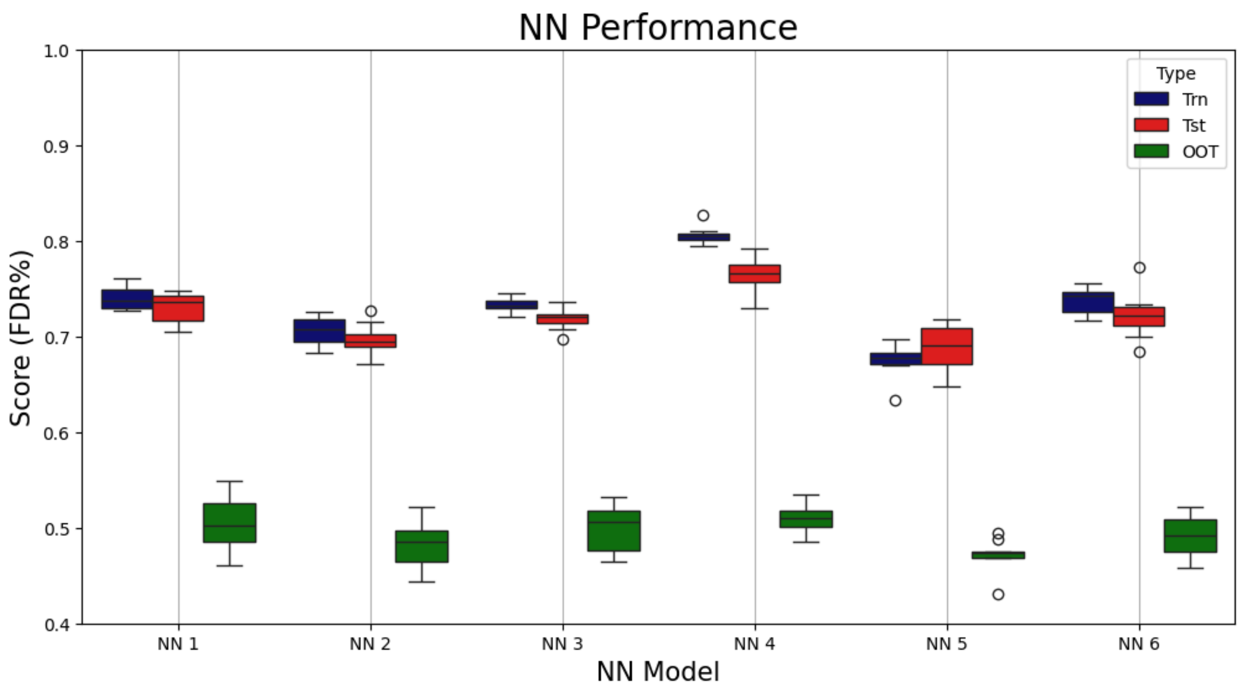


Figure 27: Performance of Neural Network Model

Figure 27 is a boxplot visualization taht shows the performance of servaral Neural Network model, identified as NN 1 through NN 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. It't noticeable that NN 1 seem to generalize better compared to others.

Table 13: Final choice of Neural Network Model

Hyperparamter			
hidden_layer_sizes	(10, 12)	max_iter	200
activation	relu	learning_rate	constant
alpha	0.0001	solver	adam

Table 13 shows the final hyperparameter choice of Neural Network Model. Here is the breakdown if each hyperparameter:

- hidden_layer_sizes: (10, 12)This setting specifies the structure of the hidden layers within the network. The notation (10, 12) indicates that the neural network has two hidden layers, where the first layer contains 10 neurons, and the second layer contains 12 neurons. The arrangement of hidden layers and their respective sizes significantly affects the network's capacity to model complex relationships in the data, influencing both the learning capabilities and computational demands.

- max_iter: 200. This parameter determines the maximum number of epochs, which are complete passes over the entire training dataset, that the optimizer will execute during training. A higher number of iterations allows more opportunities for the network to learn from the data, but setting it too high can lead to overfitting, especially if the training loss continues to decrease while validation performance deteriorates.

- activation: relu: This refers to the activation function used for the neurons in the hidden layers. The Rectified Linear Unit (ReLU) function is commonly used because it helps prevent the vanishing gradient problem and generally allows models to learn faster and perform better.

  + ReLU is defined as:

    $f(x)=\max(0,x)$

    $f(x)=\max(0,x)$ and is particularly effective at introducing non-linearity into the network, enabling it to capture more complex patterns.

- learning_rate: constant. This setting specifies that the learning rate during the training process remains constant, as opposed to being adaptive or employing a decay mechanism. Using a constant learning rate can simplify the training dynamics, but it may require careful tuning to strike a balance between convergence speed and stability.

- alpha: 0.0001. This is the L2 regularization parameter, which helps control overfitting by penalizing large weights in the model's parameters. A non-zero alpha encourages the weights to remain small, which can simplify the model and improve generalization. The specified value (0.0001) provides a moderate amount of regularization, helping to prevent overfitting without significantly detracting from the model's ability to fit the training data.

- solver: adam. The solver 'adam' refers to the optimization algorithm used for weight updates in the training process. Adam (Adaptive Moment Estimation) is an extension to stochastic gradient descent that has become popular for training deep neural networks. Adam combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp. It computes adaptive learning rates for each parameter by keeping track of the first and second moments of the gradients, facilitating efficient and effective convergence, especially on complex datasets and architectures.

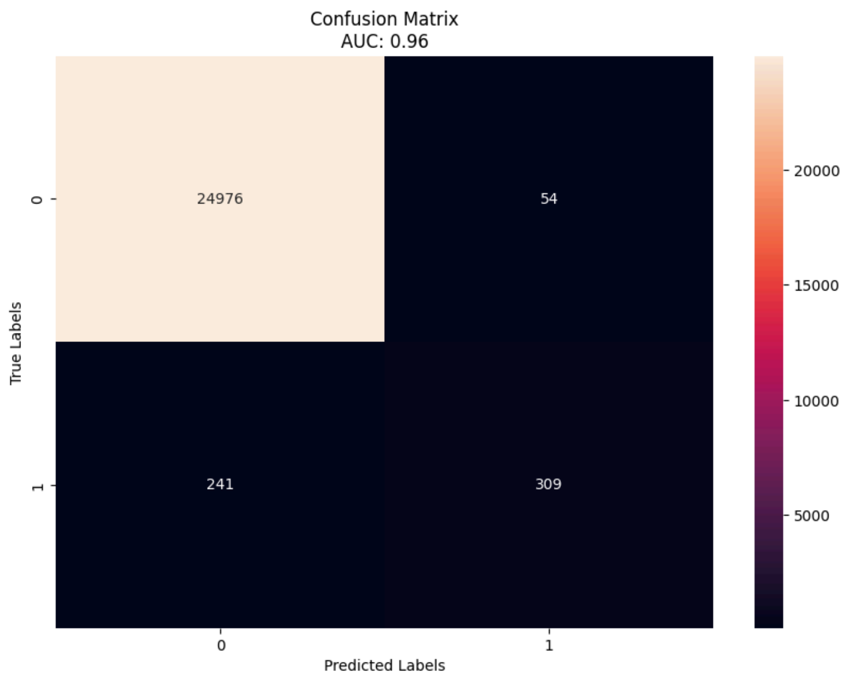


Figure 28: Confusion matrix of final choice Neural Network model

XGBoost (Extreme Gradient Boosting)

XGBoost stands for "Extreme Gradient Boosting" and is an advanced implementation of gradient boosting that has gained popularity due to its speed and performance. XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. At its core, XGBoost is an algorithm that sequentially builds an ensemble of shallow trees, with each new tree correcting errors made by the previously trained trees.

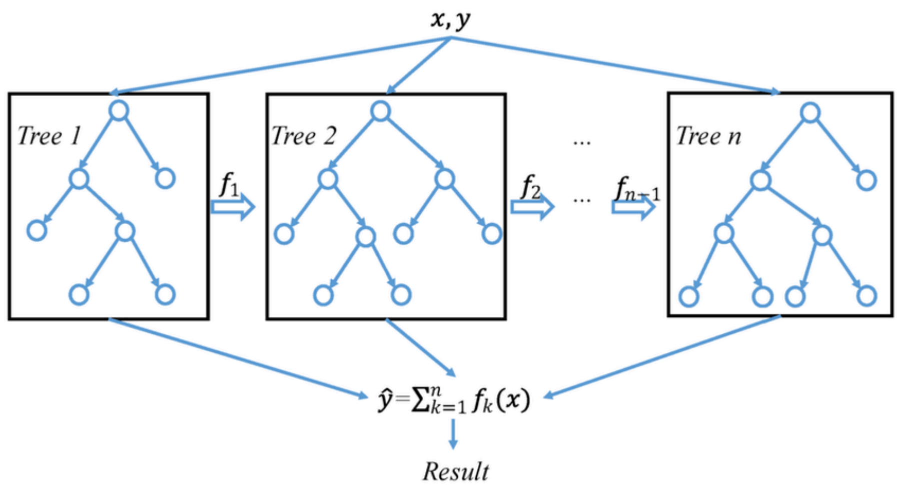


Figure 29: XGBoost Architecture [9]

Why Use It in Fraud Detection:

+ High Predictive Power: XGBoost is renowned for delivering highly accurate models, which can capture complex nonlinear patterns in data. This is essential in fraud detection, where fraudulent behaviors can be subtle and complex.

+ Scalability: Given the large volumes of data typically involved in fraud detection, XGBoost's scalability ensures that it can handle such datasets efficiently without compromising on speed or performance.

+ Feature Importance: XGBoost provides straightforward metrics to evaluate the importance of different features in making predictions. This insight is crucial in fraud detection to identify which variables are most predictive of fraudulent activity and to refine the models based on those insights.

+ Model Robustness: With its built-in regularization and cross-validation capabilities, XGBoost models are generally robust and less likely to overfit, making them reliable for deployment in critical applications like fraud detection.

+ Flexibility: XGBoost can be tuned with a wide range of hyperparameters that control the model's complexity and learning process, allowing fraud detection systems to be finely adjusted to optimize performance.

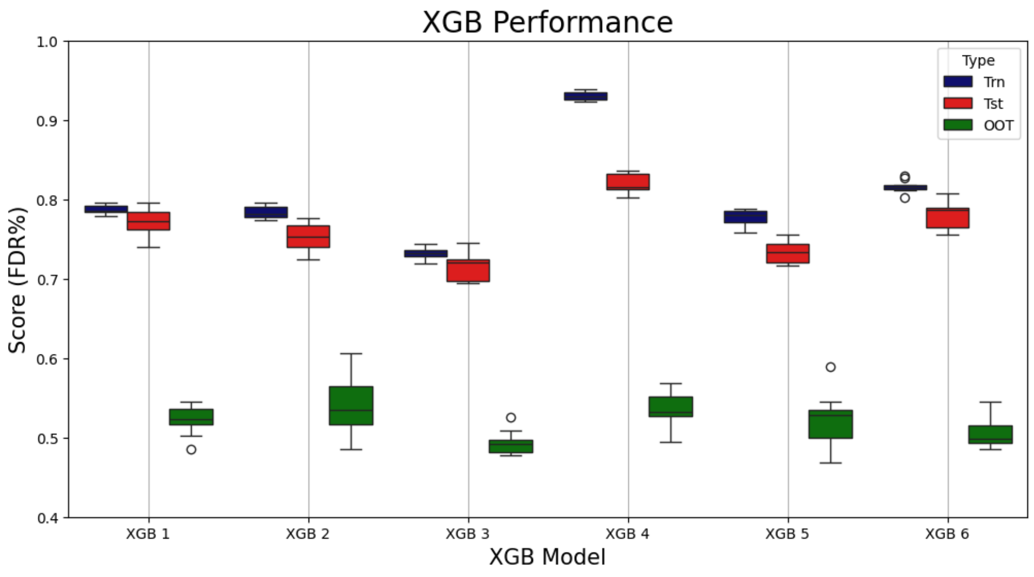


Figure 30: Performance of XGBoost Model

Figure 30 is a boxplot visualization taht shows the performance of servaral XGBoost model, identified as XGB 1 through XGB 7, which are output of hyperparameter settings in the table 5 above, using FDR score metric. It't noticeable that XGB 1 seem to generalize better compared to others.

Table 14: Final choice of XGBoost Model

Hyperparamter			
booster	gbtree	max_depth	3
n_estimatore	100	learning_rate	0.1

Table 14 shows the final hyperparameter choice of Neural Network Model. Here is the breakdown if each hyperparameter:

+ booster: gbtree. The booster parameter specifies the type of model to run at each iteration. The option gbtree indicates that the model uses tree-based models as base learners. This is the most commonly used and typically most powerful XGBoost model type, suitable for a variety of prediction tasks.Tree-based boosters are capable of modeling complex relationships in data by constructing decision trees in a sequential manner where each new tree corrects errors made by the previous ones.

+ max_depth: 3. This parameter sets the maximum depth of each tree. It is used to control overfitting as higher depth will allow the model to learn relations very specific to a particular sample. A max_depth of 3 means the trees are relatively shallow and can prevent the model from becoming overly complex and overfitting the training data. It limits the interaction of variables to simple combinations, which can be beneficial for generalization.

+ n_estimators: 100. This defines the number of boosting rounds or trees to build. Contrary to random forests, where a higher number of trees are always better, XGBoost will eventually start overfitting if trained with too many trees. Setting n_estimators to 100 provides a good balance, allowing the model to learn from the data sufficiently without excessive computation or risk of fitting too closely to the training data noise.

+ learning_rate: 0.1. Also known as the "shrinkage" or "eta", this parameter scales the contribution of each tree by a factor to reduce overfitting. A lower value makes the model robust to the specific structure of the tree and makes improvements robustly but slowly. A learning rate of 0.1 means that each tree's contribution is scaled down by 10%, helping to prevent overfitting and allowing more robust learning across many boosting rounds.

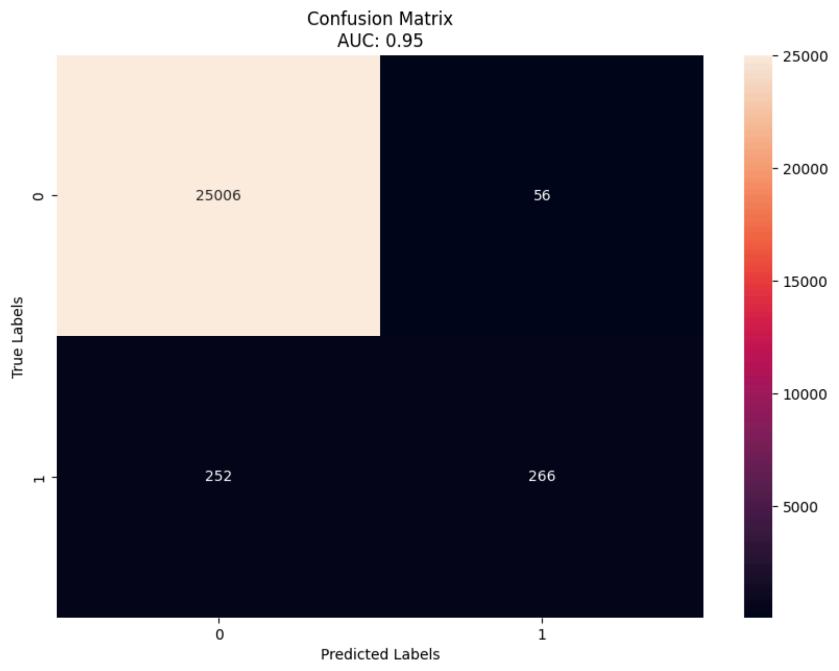


Figure 31: Confusion matrix of final choice XGBoost model

2. Model Result

Table 15: Model results on training, testing and OOT

Model	Train	Test	OOT
Logistic Regression	0.6789	0.6859	0.4710
Decision Tree	0.7454	0.7267	0.4946
Random Forest	0.7533	0.7402	0.5104
LGBM	0.7452	0.7332	0.5374
Neural Network	0.7400	0.7295	0.5040
Catboost	0.7718	0.7532	0.5077
XGBoost	0.7873	0.7710	0.5226

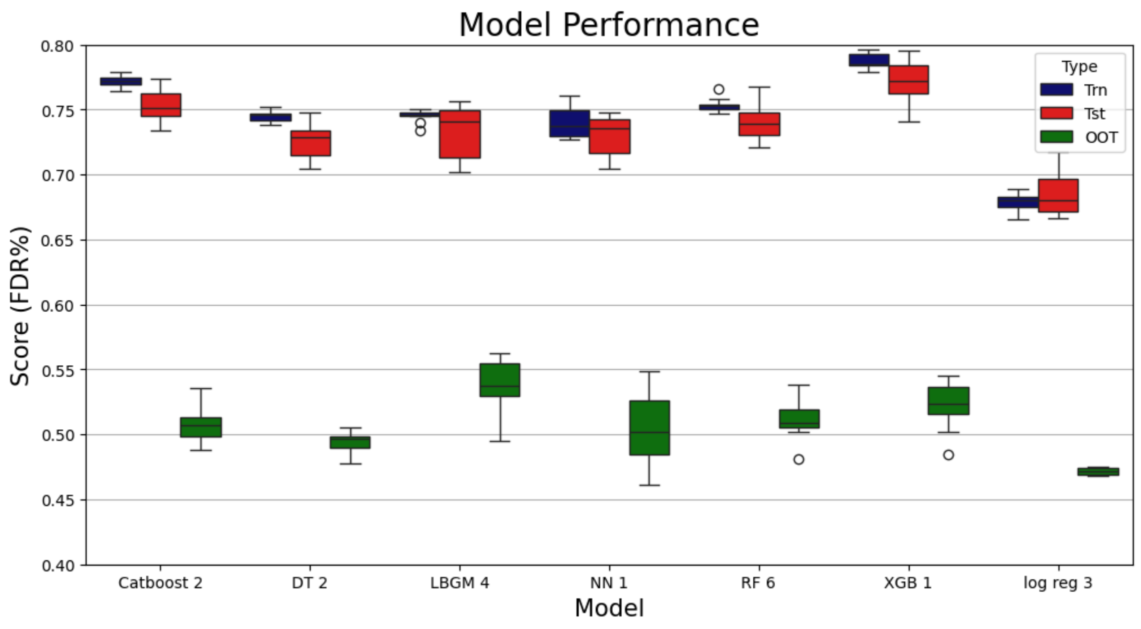


Figure 32: All final models performance

Table 15 and figure 32 showing the results of all final model performance. In the model exploration results, several models were assessed across training, testing, and out-of-time (OOT) data sets to evaluate their performance and suitability for deployment. The analysis revealed that while simpler models like Logistic Regression performed modestly, they struggled particularly with OOT data, suggesting limited capability to capture complex patterns. Decision Trees and Neural Networks showed moderate performance, but the standout models were ensemble and boosting techniques, such as Random Forest, LGBM, Catboost, and XGBoost. These models demonstrated higher consistency and robustness, with XGBoost displaying the best overall performance with the highest scores in training, testing, and crucially, OOT scenarios. Obviously, XGBoost should be the final choice for next step. This choice reflects a commitment to deploying a model that not only performs well on historical data but also maintains its effectiveness on future, unseen data.

Justification for Selecting XGBoost:

XGBoost has demonstrated high performance across training and testing phases, and most importantly, it has excelled in the OOT dataset, which is crucial for predicting future events accurately. This model's ability to handle large datasets efficiently and its faster training times compared to other ensemble methods make it particularly suitable for operational environments where quick decision-making is essential. Moreover, XGBoost is known for its ability to perform exceptionally well on structured data across various industries and problem types, including fraud detection.

VI. Final Model Performance

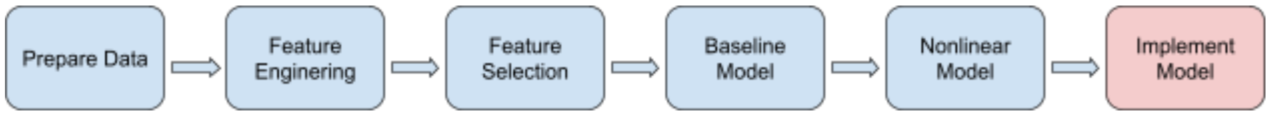


Figure 33: Workflow of diagram - Implement Model

These three tables below show the model performance statistic of three dataset (Table 16 of training dataset, Table 17 of testing dataset, Table 18 of OOT dataset), providing a detailed breakdown of results across various percentile bins. The OOT results, which are crucial for understanding how the model might perform in a real-world scenario, reveal that the FDR across the top bins (1-5) remains relatively high, suggesting that the model effectively prioritizes transactions with a higher likelihood of being fraudulent. The cumulative statistics indicate an increasing capture rate of fraudulent transactions as we move through the bins, which is a desirable property in a fraud detection system. The model exhibits stability across the training and testing datasets but shows variation in the OOT dataset, particularly in the lower bins where the fraud rate dips significantly. This could indicate potential overfitting or model degradation over time, necessitating continuous monitoring and recalibration. Specifically, the cumulative statistics for the OOT dataset reveal an FDR (Fraud Detection Rate) of 5%, indicating that setting the fraud score threshold at this level could potentially identify about 69.36% of all fraudulent transaction attempts. A 5% cutoff implies that the top 5% of transactions, ranked by their likelihood of fraud, would be declined. It's important to note that the FDR at 5% varies from the values presented in Table 15, which represents the average FDR from more than ten runs per dataset, whereas the FDR mentioned here is derived from a single run during the development of the XGBoost model.

Table 16: Training results

Training	# Record		# Good		# Bad		Fraud Rate					
	59,684		58,442		1,242		0.020809597					
Bin Statistic						Cumulative Statistics						
Population Bin %	# Recod	# Good	# Bad	% Good	% Bad	Total # Records	Cumulative Goods	Cumulative Bads	% Good	% Bad (FDR)	KS	FPR
1	597	48	549	8.04%	91.96%	597	48	549	0.08%	44.2%	44.121	0.087
2	597	262	335	43.89%	56.11%	1194	310	884	0.53%	71.18%	70.645	0.351
3	597	501	96	83.92%	16.08%	1791	811	980	1.39%	78.9%	77.517	0.828
4	596	547	49	91.78%	8.22%	2387	1358	1029	2.32%	82.85%	80.527	1.320
5	597	565	32	94.64%	5.36%	2984	1923	1061	3.29%	85.43%	82.136	1.812
6	597	575	22	96.31%	3.69%	3581	2498	1083	4.27%	87.2%	82.924	2.307
7	597	572	25	95.81%	4.19%	4178	3070	1108	5.25%	89.21%	83.958	2.771
8	597	587	10	98.32%	1.68%	4775	3657	1118	6.26%	90.02%	83.759	3.271
9	597	589	8	98.66%	1.34%	5372	4246	1126	7.27%	90.66%	83.395	3.771
10	596	585	11	98.15%	1.85%	5968	4831	1137	8.27%	91.55%	83.280	4.249
11	597	590	7	98.83%	1.17%	6565	5421	1144	9.28%	92.11%	82.834	4.739
12	597	589	8	98.66%	1.34%	7162	6010	1152	10.28%	92.75%	82.470	5.217
13	597	589	8	98.66%	1.34%	7759	6599	1160	11.29%	93.4%	82.106	5.689
14	597	594	3	99.5%	0.5%	8356	7193	1163	12.31%	93.64%	81.331	6.185
15	597	595	2	99.66%	0.34%	8953	7788	1165	13.33%	93.8%	80.474	6.685
16	596	594	2	99.66%	0.34%	9549	8382	1167	14.34%	93.96%	79.619	7.183
17	597	589	8	98.66%	1.34%	10146	8971	1175	15.35%	94.61%	79.255	7.635
18	597	594	3	99.5%	0.5%	10743	9565	1178	16.37%	94.85%	78.480	8.120
19	597	595	2	99.66%	0.34%	11340	10160	1180	17.38%	95.01%	77.623	8.610
20	597	596	1	99.83%	0.17%	11937	10756	1181	18.4%	95.09%	76.684	9.108

Table 17: Testing results

Testing	# Record		# Good		# Bad		Fraud Rate					
	25,580		25,072		508		0.019859265					
Bin Statistic						Cumulative Statistics						
Population Bin %	# Recod	# Good	# Bad	% Good	% Bad	Total # Records	Cumulative Goods	Cumulative Bads	% Good	% Bad (FDR)	KS	FPR
1	256	31	225	12.11%	87.89%	256	31	225	0.12%	44.29%	44.168	0.138
2	256	116	140	45.31%	54.69%	512	147	365	0.59%	71.85%	71.264	0.403
3	255	218	37	85.49%	14.51%	767	365	402	1.46%	79.13%	77.678	0.908
4	256	230	26	89.84%	10.16%	1023	595	428	2.37%	84.25%	81.879	1.390
5	256	245	11	95.7%	4.3%	1279	840	439	3.35%	86.42%	83.067	1.913
6	256	249	7	97.27%	2.73%	1535	1089	446	4.34%	87.8%	83.452	2.442
7	256	253	3	98.83%	1.17%	1791	1342	449	5.35%	88.39%	83.033	2.989
8	255	243	12	95.29%	4.71%	2046	1585	461	6.32%	90.75%	84.426	3.438
9	256	253	3	98.83%	1.17%	2302	1838	464	7.33%	91.34%	84.008	3.961
10	256	250	6	97.66%	2.34%	2558	2088	470	8.33%	92.52%	84.192	4.443
11	256	251	5	98.05%	1.95%	2814	2339	475	9.33%	93.5%	84.175	4.924
12	256	255	1	99.61%	0.39%	3070	2594	476	10.35%	93.7%	83.355	5.450
13	255	251	4	98.43%	1.57%	3325	2845	480	11.35%	94.49%	83.141	5.927
14	256	256	0	100%	0%	3581	3101	480	12.37%	94.49%	82.120	6.460
15	256	252	4	98.44%	1.56%	3837	3353	484	13.37%	95.28%	81.902	6.928
16	256	255	1	99.61%	0.39%	4093	3608	485	14.39%	95.47%	81.082	7.439
17	256	254	2	99.22%	0.78%	4349	3862	487	15.4%	95.87%	80.463	7.930
18	255	251	4	98.43%	1.57%	4604	4113	491	16.4%	96.65%	80.249	8.377
19	256	254	2	99.22%	0.78%	4860	4367	493	17.42%	97.05%	79.629	8.858
20	256	254	2	99.22%	0.78%	5116	4621	495	18.43%	97.44%	79.010	9.335

Table 18: OOT results

OOT	# Record		# Good		# Bad		Fraud Rate					
	12,110		11,804		270		0.022295623					
Bin Statistic						Cumulative Statistics						
Population Bin %	# Record	# Good	# Bad	% Good	% Bad	Total # Records	Cumulative Goods	Cumulative Bads	% Good	% Bad (FDR)	KS	FPR
1	122	23	99	18.85%	81.15%	122	23	99	0.19%	33.33%	33.14	0.23
2	123	93	30	75.61%	24.39%	245	116	129	0.97%	43.43%	42.46	0.90
3	122	95	27	77.87%	22.13%	367	211	156	1.77%	52.53%	50.76	1.35
4	122	98	24	80.33%	19.67%	489	309	180	2.59%	60.61%	58.02	1.72
5	123	97	26	78.86%	21.14%	612	406	206	3.4%	69.36%	65.96	1.97
6	122	110	12	90.16%	9.84%	734	516	218	4.32%	73.4%	69.08	2.37
7	122	105	17	86.07%	13.93%	856	621	235	5.2%	79.12%	73.92	2.64
8	123	118	5	95.93%	4.07%	979	739	240	6.19%	80.81%	74.62	3.08
9	122	114	8	93.44%	6.56%	1101	853	248	7.15%	83.5%	76.35	3.44
10	122	119	3	97.54%	2.46%	1223	972	251	8.14%	84.51%	76.37	3.87
11	123	120	3	97.56%	2.44%	1346	1092	254	9.15%	85.52%	76.37	4.30
12	122	118	4	96.72%	3.28%	1468	1210	258	10.14%	86.87%	76.73	4.69
13	122	121	1	99.18%	0.82%	1590	1331	259	11.15%	87.21%	76.05	5.14
14	122	121	1	99.18%	0.82%	1712	1452	260	12.17%	87.54%	75.38	5.58
15	123	122	1	99.19%	0.81%	1835	1574	261	13.19%	87.88%	74.69	6.03
16	122	121	1	99.18%	0.82%	1957	1695	262	14.2%	88.22%	74.01	6.47
17	122	121	1	99.18%	0.82%	2079	1816	263	15.22%	88.55%	73.34	6.90
18	123	123	0	100%	0%	2202	1939	263	16.25%	88.55%	72.31	7.37
19	122	122	0	100%	0%	2324	2061	263	17.27%	88.55%	71.28	7.84
20	122	122	0	100%	0%	2446	2183	263	18.29%	88.55%	70.26	8.30
21	123	123	0	100%	0%	2569	2306	263	19.32%	88.55%	69.23	8.77

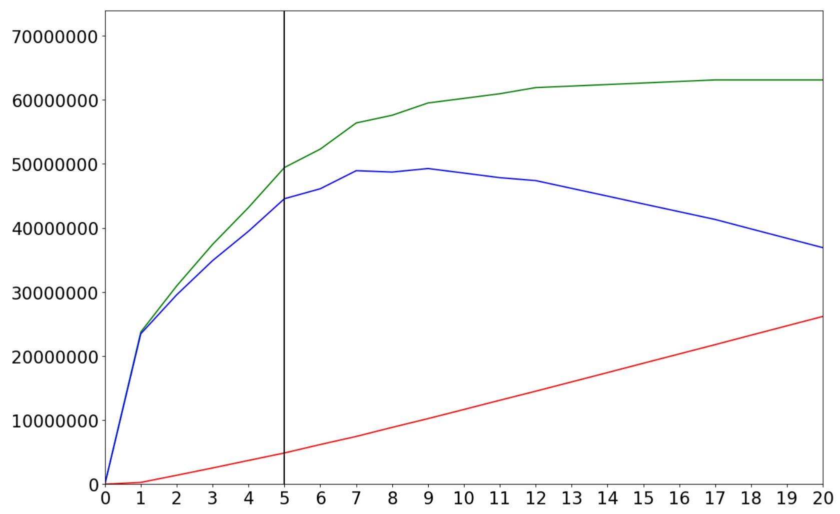


Figure 33: Financial implications in fraud detection system

Figure 33 presents the financial implication of different cutoff setting in a fraud detection system with three key metrics: The greenline represents cumulative fraud savings. This line shows the total amount saved by preventing fraud through the detection system up to each respective percentile cutoff; the redline represents cumulative losses due to false positives (FP), where legitimate transactions are incorrectly flagged as fraudulent, potentially leading to lost revenue or customer dissatisfaction; and finally the blueline represents the net savings, calculated as the difference between the fraud savings and the losses due to false positives. The green line's steep initial rise indicates that a significant amount of fraud is being caught and prevented in the lowest 5% of the scoring range, which demonstrates the effectiveness of the model in identifying high-risk transactions. This implies that the most fraudulent activities are effectively prioritized by the model within this segment. The red line shows a gradual increase, suggesting that the cost due to false positives is accumulating, but at a slower rate compared to the savings from fraud detection. This is expected as any effective fraud detection system will inevitably produce some false positives. Up until the 5% cutoff, the blue line increases, indicating that the overall savings from fraud prevention outweigh the losses from false positives. Beyond the 5% cutoff, the overall savings appear to plateau and even decrease slightly, suggesting diminishing returns from setting a more lenient cutoff.

Based on the data, the 5% cutoff is effective in maximizing the fraud savings while controlling losses from false positives. This cutoff ensures that a significant portion of the fraud is captured while keeping the cost of false positives manageable. The overal saving of the 5% cutoff is estimated to be $44,568,000.0 and the maximum possible saving of fraud detection using this model is $49,284,000.0

VII. Conclusions and Further Work

In this project, we developed a comprehensive fraud detection model using a methodical approach outlined in five primary stages: data preparation, feature engineering, feature selection, model evaluation and implementation. The culmination of this project in the deployment of an XGBoost-based fraud detection model represents a significant step forward in safeguarding our operations against fraudulent activities. The model’s performance, evidenced by substantial potential savings and a strategic 5% cutoff point, highlights its effectiveness in identifying high-risk transactions with precision.

This project’s success can be attributed to a methodical approach to the analytics lifecycle, from initial data handling to advanced model tuning, which explores the application of linear and nonlinear statistical and machine learning models. Notably, nonlinear models demonstrated superior capabilities compare to linear regression, indicating their effectiveness in handling complex and non-linear relationships inherent in our data. Random Forest, LGBM emerged as a strong contender with highest OOT performance which suggests their robustness in generlizing to unseen data, However, a significant gap between its training and testing performance raises concerns about potential overfitting. This discrepancy implies that while the model is highly accurate on known data, its complexity may cause it to perform inconsistently on new, unseen data sets.By choosing XGBoost, we leveraged an algorithm known for its high efficiency and scalability, which proved essential given the complexity and volume of our transaction data, it can detect about half of fraud attempts within only top 5% that was sorted as suspicious by the fraud algorithm score, estimated to be $44,568,000.00 overal saving. The optimized model not only enhances our preventive measures but also aligns with our commitment to continuous improvement in our risk management strategies

Furthermore, the insights gained from this project extend beyond immediate financial benefits. They reinforce the importance of advanced analytics in strategic decision-making, providing a blueprint for addressing similar challenges across different facets of our business. The model's ability to cut through noise and identify genuine threats allows us to allocate resources more effectively, ensuring both operational efficiency and customer trust.

Looking ahead, the journey of enhancing our fraud detection capabilities is far from complete. The dynamic nature of fraud, coupled with evolving technologies, necessitates ongoing adaptations and innovations in our approaches. Further explorations into machine learning and artificial intelligence could yield even more sophisticated modeling techniques. Additionally, incorporating feedback loops into the model's architecture can facilitate adaptive learning, ensuring the model remains effective as fraud tactics change.

Expanding the scope of our data integration to include more diverse and real-time data sources could also unveil new patterns and trends in fraud, providing earlier warnings and more nuanced understanding of risk factors. Collaborations with industry experts and participation in fintech innovations will further enhance our capabilities and keep us at the forefront of fraud prevention technologies.

By building on the foundations laid by this project and continually striving for excellence in our analytical capabilities, we can ensure the security and integrity of our transactions, safeguarding both our assets and our customers' trust.

Apendix 1: Data Quality Report

1. Data description

The dataset contains transaction records from credit card usage. It originates from a synthetic dataset representative of real U.S. credit card transactions over 10 years. The dataset comprises **97,852 records and 10 fields**, covering transactions from **01/01/2010 to 12/31/2010**. The fields are a mixture of numeric and categorical types, including identifiers, transaction details, merchant information, and a fraud indicator.

2. Summary Tables

Numeric Fields Table

Field Name	# Records Have Values	% Populated	# Zeros	Min	Max	Mean	Std Dev	Most Common Value
Date	97,852	100.00%	0	1/1/10	12/31/10	NA	NA	2/28/10
Amount	97,852	100.00%	0	0.01	3,102,045.53	425.47	9949.8	3.62

Categorical Fields Table

Field Name	# Records with Values	% Populated	# Zeros	# Unique Values	Most Common Value
Recnum	97,852	100.00%	0	97,852	1
Cardnum	97,852	100.00%	0	1,645	5142148452
Merchnum	94,455	96.52%	3,397	13,091	930090121224
Merch description	97,852	100.00%	0	13,126	GSA-FSS-ADV
Merch zip	92,149	95.19%	4,703	4,568	38118
Merch state	96,649	98.77%	1,203	227	TN
Transtype	97,852	100.00%	0	4	P
Fraud	97,852	100.00%	95,805	2	0

3. Visualization of Each Field

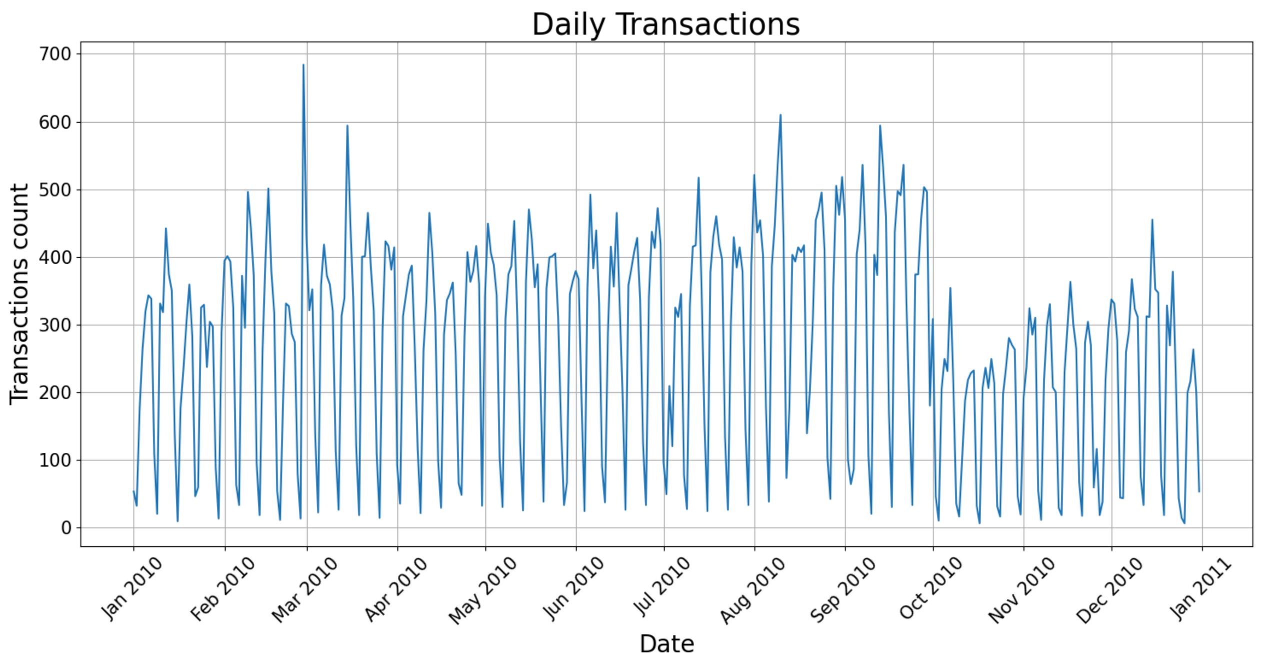
1) Recnum

Description: Ordinal unique positive integer for each transaction record, from 1 to 97,852.

2) Date

Description: The transaction's date, is formatted as a string (e.g., ‘1/1/10’).

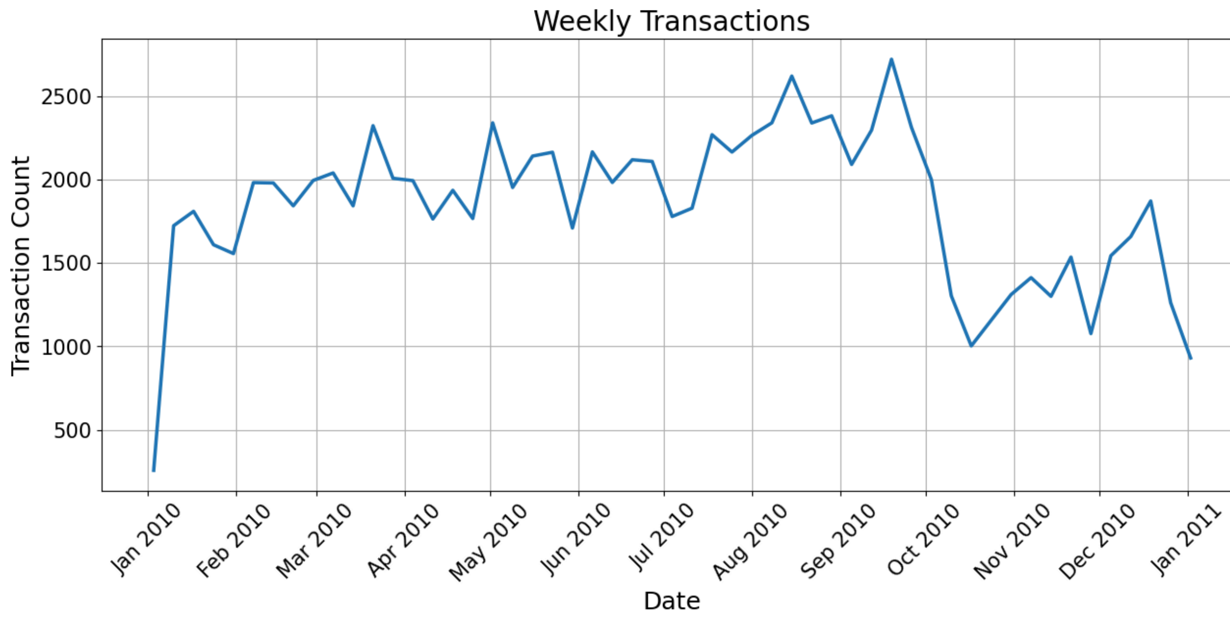
The first distribution shows the number of transactions per day over a year. The graphs display significant variability of transaction counts from day to day. While there are fluctuations, there does not appear to be a clear long-term upward or downward trend throughout the year. Moreover, the continuous nature of the line suggests that the transaction data is complete across the period with obvious gaps. This is a positive sign of data integrity in terms of completeness



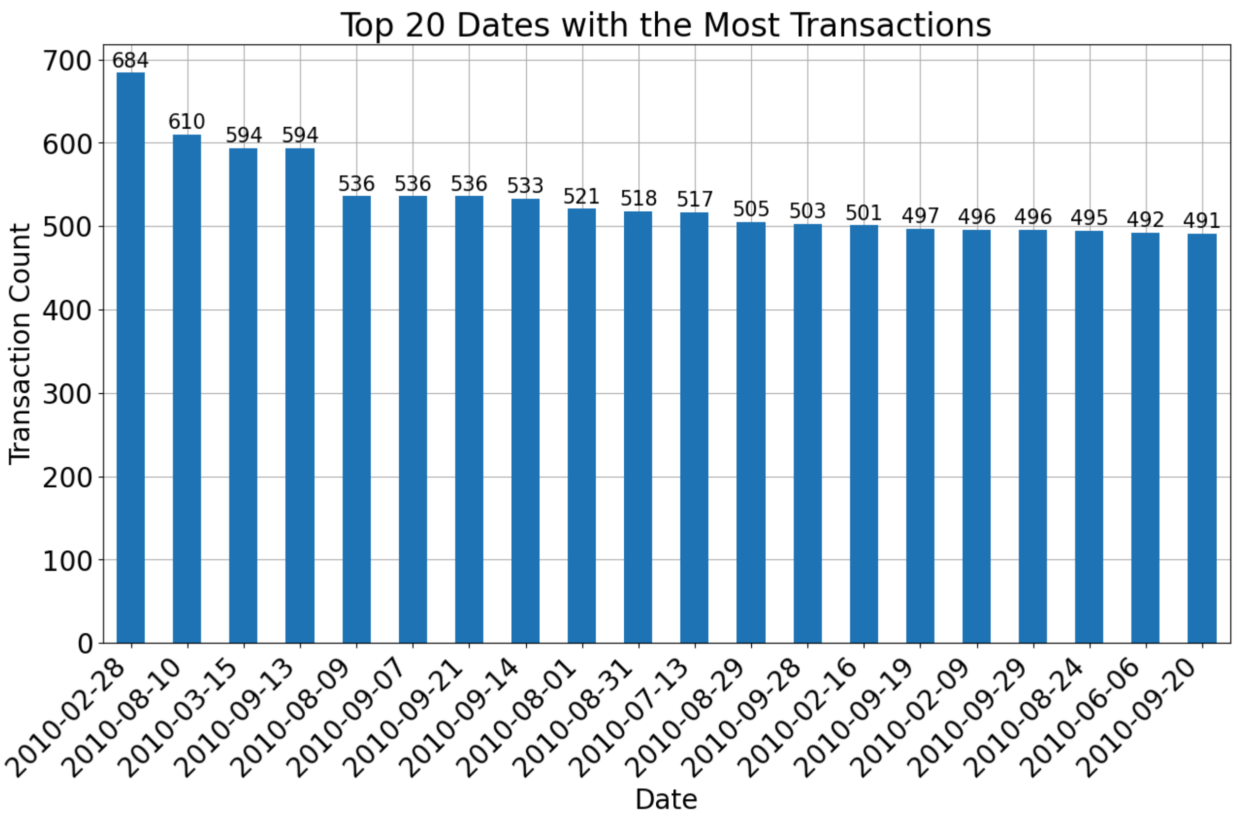
The second distribution shows the number of transactions recorded each week over the course of a year, from January 2010 through January 2011. There seems to be a pattern that could suggest seasonal trends, with transaction counts rising and falling at certain times of the there. For example, there are peaks that might correspond to seasonal events where transaction activity is higher.

There are a few sharp drops in transaction counts, such as a noticeable dip toward the end of the year. This could be due to various reasons like holidays when businesses are closed or it could indicate missing data or other data collection issues. The transaction count starts at a lower point, rises to a plateau with some fluctuations, and then shows a downward trend toward the end of the year. It would be important to determine if this is a typical pattern for the business or if there are external factors that may have influenced these trends

The consistency in weekly transactions, apart from the anomalies, suggests good data quality. However, the sharp increases or decreases warrant further investigation to confirm whether they represent true business activity or data issues

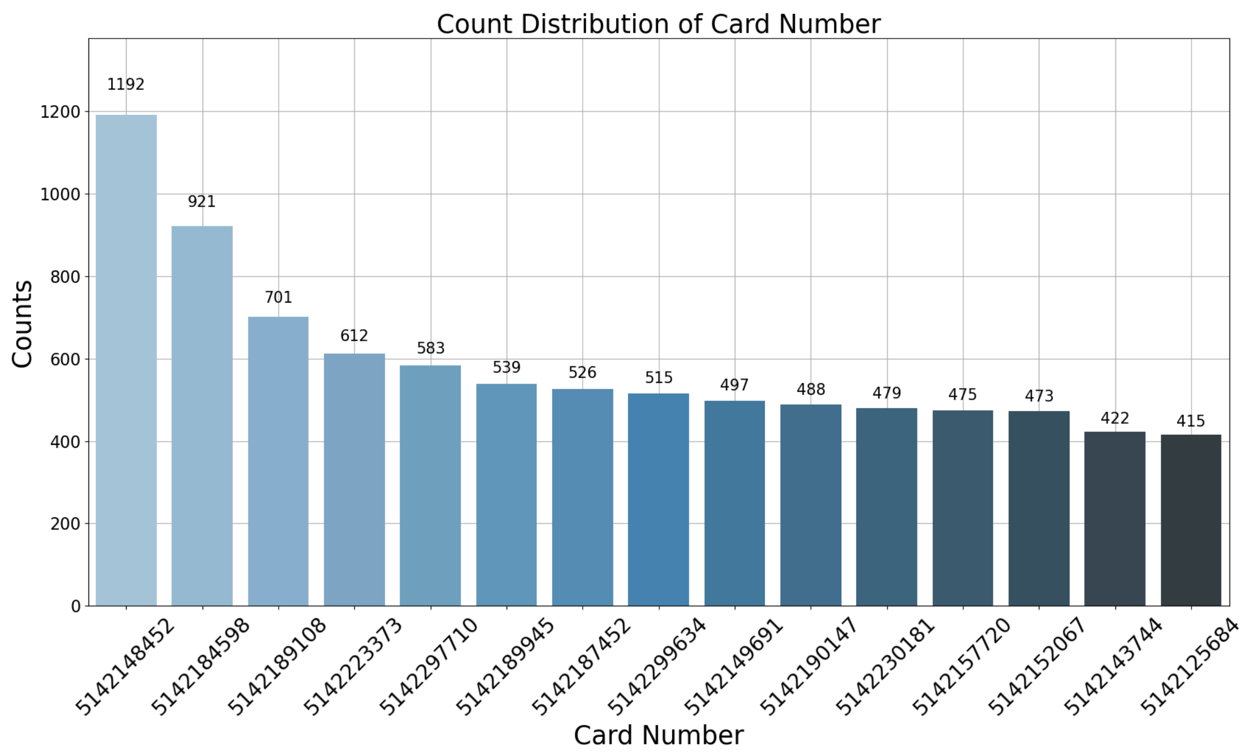


The third distribution shows the date with the top 20 transaction volume. There is a relatively small variation in the transaction counts among the top dates, ranging from around 490 to 680 transactions. This suggests a somewhat consistent high level of activity on these particular dates. The dates also are not in chronological order, indicating there are no immediate visible patterns or trends, like spikes on specific days or months. This could imply that high-transaction days are scattered throughout the year and may not be tied to common retail patterns like holidays or weekends.



3) Cardnum

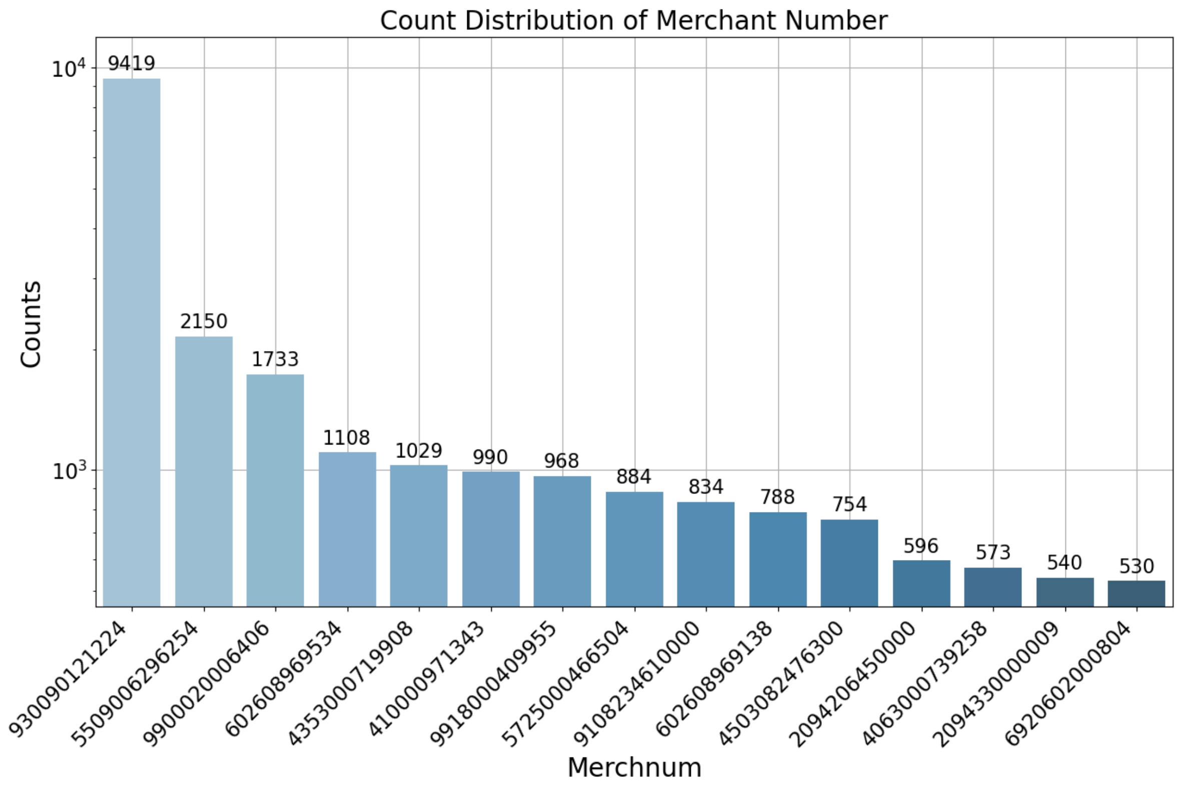
Description: Showed the frequency of transactions per card number. The distribution shows the top 20 field values of the Card Number. The most common card number is **515142148452** with a total of **1,192 transactions.** The rest of the card numbers have a much lower frequency of use. This suggests a variety of card numbers used less often, which might be expected in a large dataset representing individual transactions by different cardholders. The diversity in card number usage could validate the robustness of transaction data.



4) Merchnum

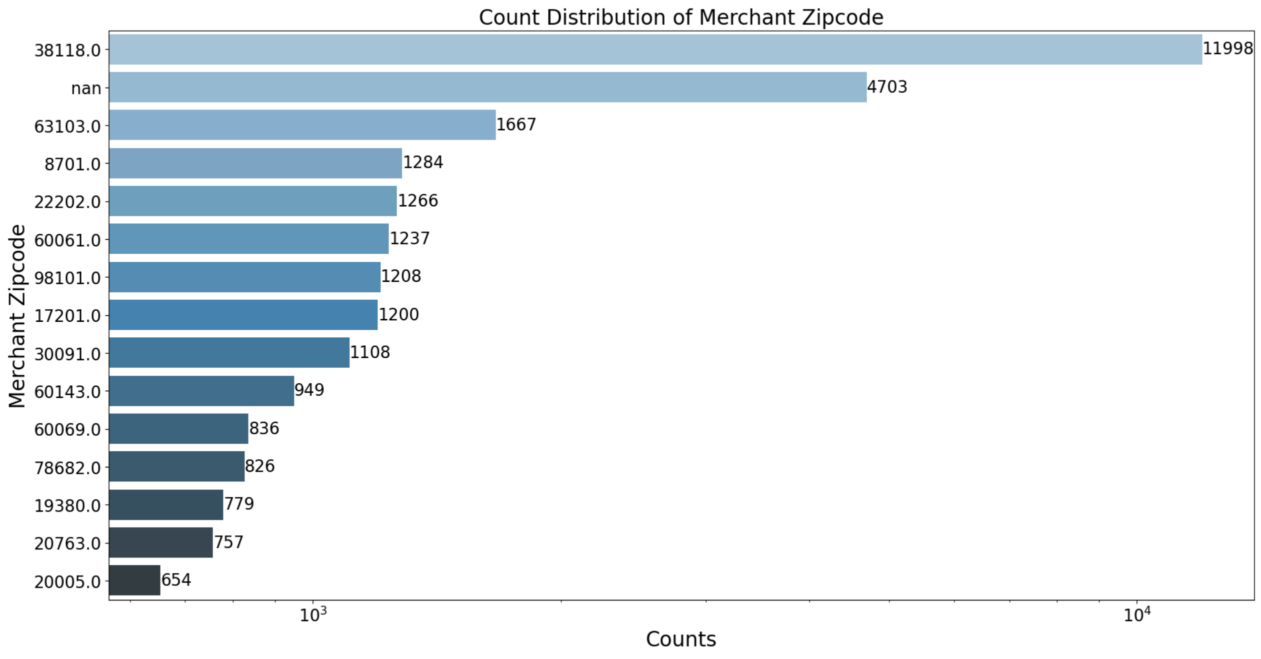
Description: A merchant number identifying the merchant involved in the transaction

The distribution shows the top 20 field values of Merchant Number. The most common merchant number is **930090121224** with a total of **9,419 transactions**



5) Merch zip

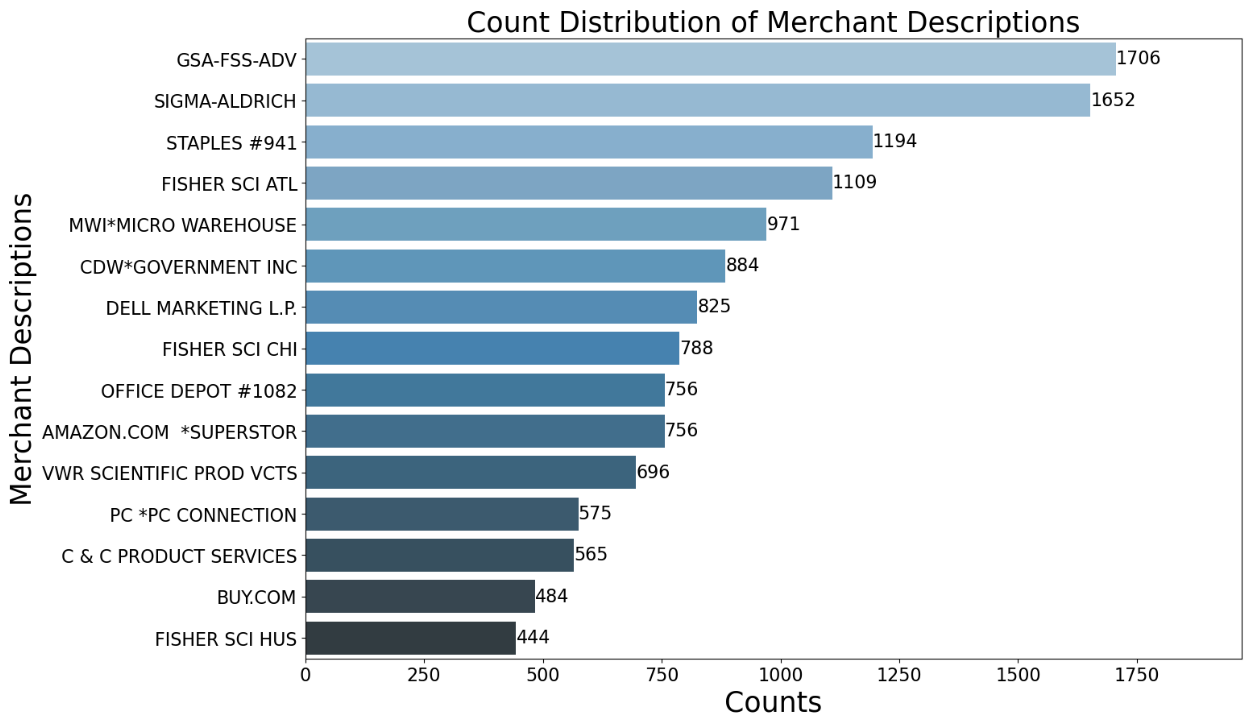
Description: The zip code of the merchant's location. The distribution shows the top 15 field values of the Merch zip. The most common value is **38118** with a total of **11,998 transactions**



6) Merch Description

Description: A textual description of the merchant. The distribution shows the top 15 field values of Merchant Descriptions. The most common value is **GSA-FSS-ADV** with a total of **1,706 transactions**, followed by 'SIGMA-ALDRICH' and 'STAPLES #941'.

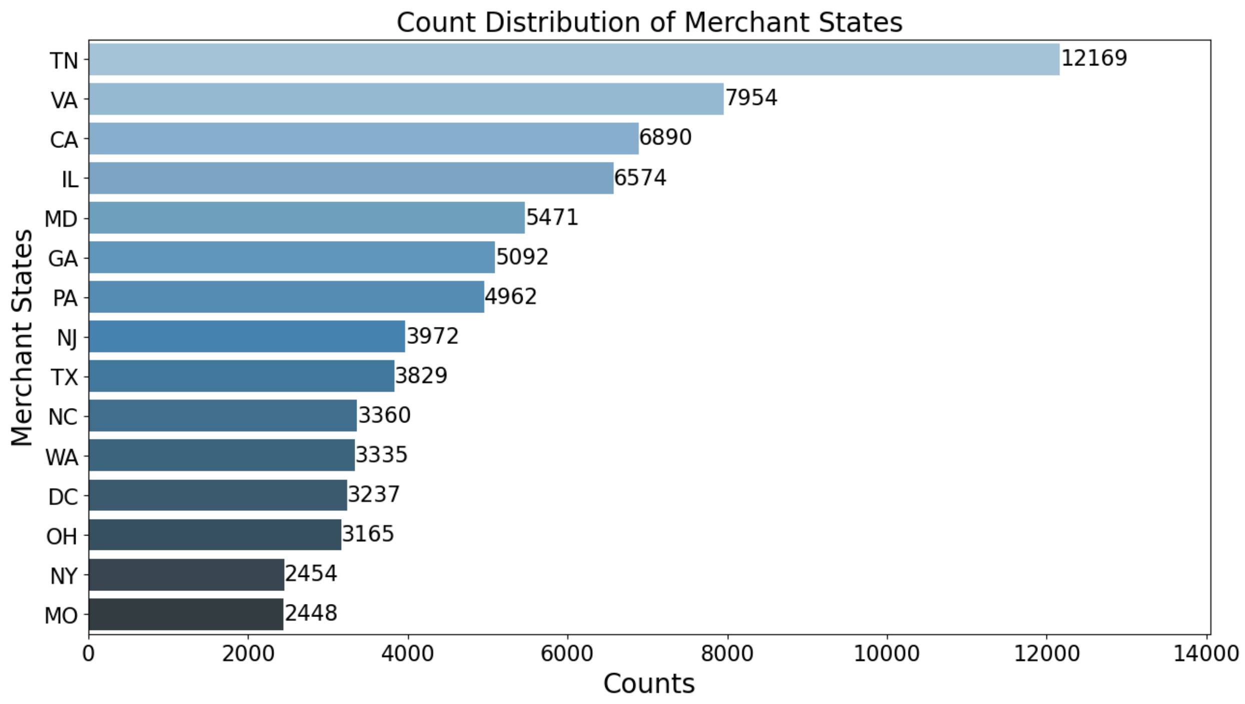
There is a visible concentration of transactions among a few merchants, this might suggest preferred vendors or could indicate areas where business is focused. The chart sheds light on vendor relationships, which vendors are most commonly used for transactions, and might point to contracts or purchasing agreements. It also appears that after the top few merchants, the frequency of transactions per merchant decreases, which is typical of a long-tail distribution. This could suggest a wide variety of less frequently used merchants.



7) Merch state

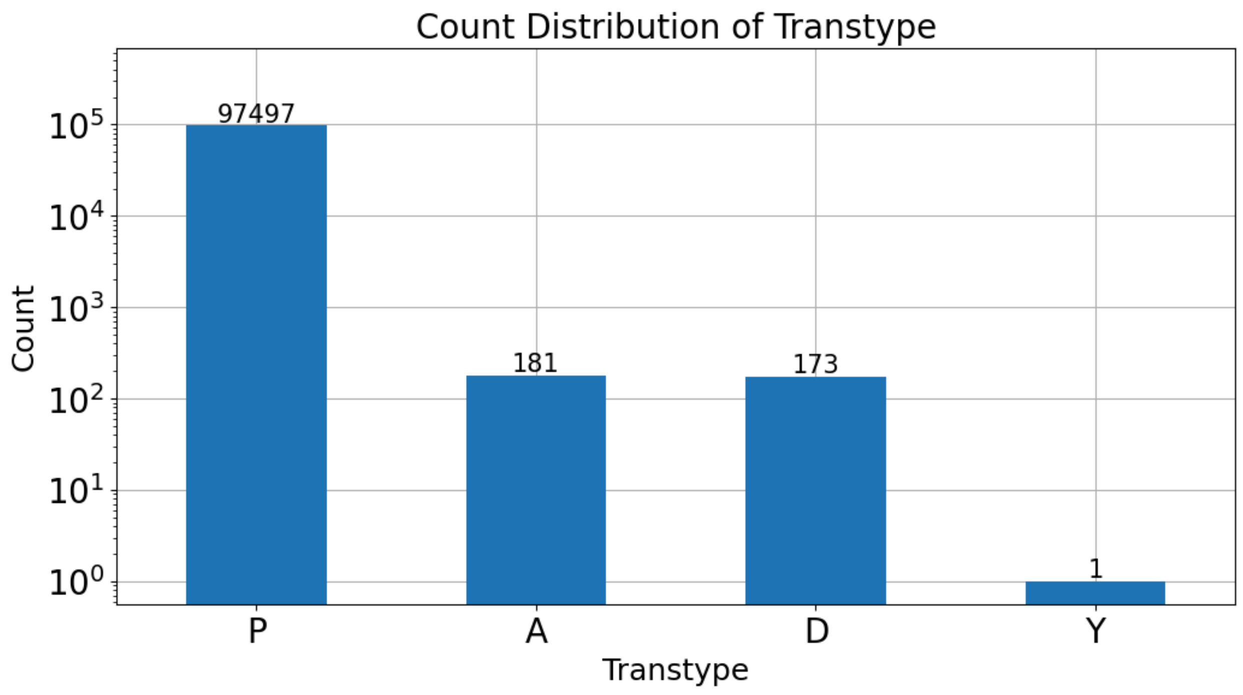
Description: The state in which the merchant is located. The distribution shows the top 15 field values of Merchant States. Tennessee (TN) has the highest count of transactions with **12169 counts**, followed by Virginia (VA) with **7,954 transactions**, and California(CA) with **6,890 transactions**

A significant proportion of the transactions take place in the top states. This could indicate that the business has a more substantial presence in the presence or customer base in these states. However, there is a noticeable drop-off in transaction counts for other states. This could be due to a variety of factors including population density, market penetration, or business focus.



8) Transtype

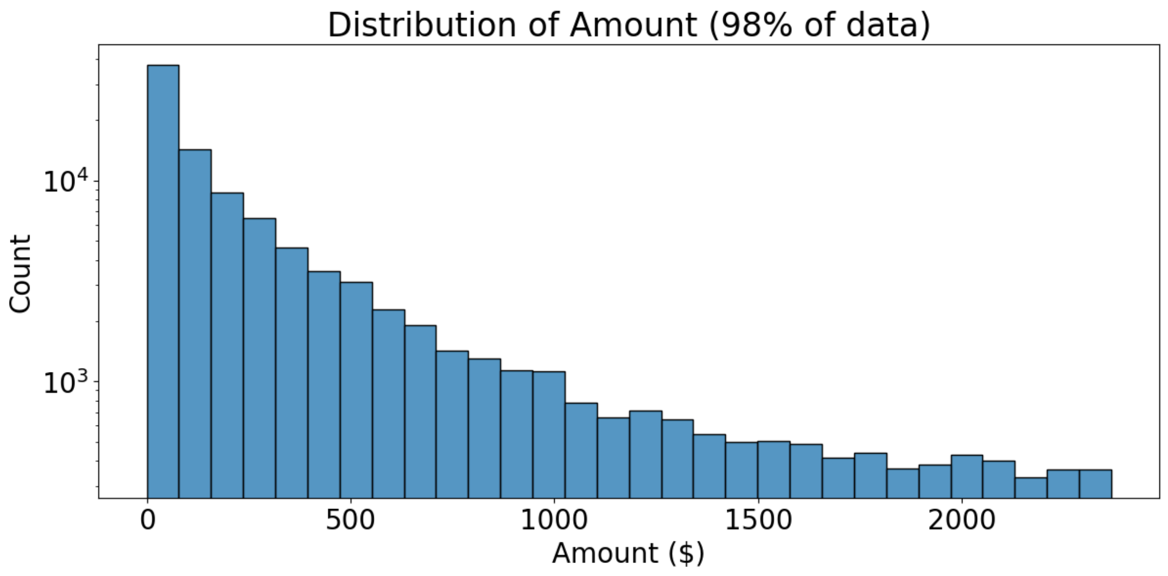
Description: The type of transaction, is represented as a single character. The pie chart shows the distribution of transaction type. “P” has dominated the highest number of transactions with a total of **97,497 transactions**, followed by “A” and “D”. The disproportion frequency of transaction types might warrant further analysis because “P” might be regular purchases whereas “Y” might represent less common or special types of transaction.



9) Amount

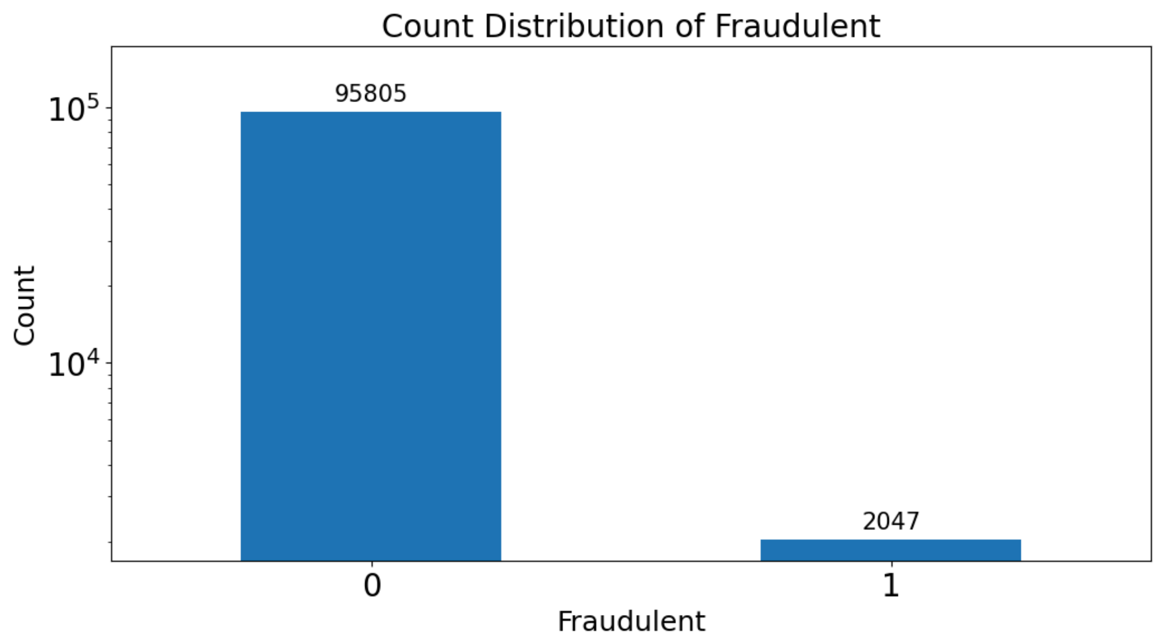
Description: The transaction amount in dollars.

The first histogram shows the distribution of the Amount field, displaying the data up to the 98th percentile. The tallest bar at the far left suggests a high frequency of transactions with a very small amount, likely indicating common low-value transactions typical in consumer behavior.

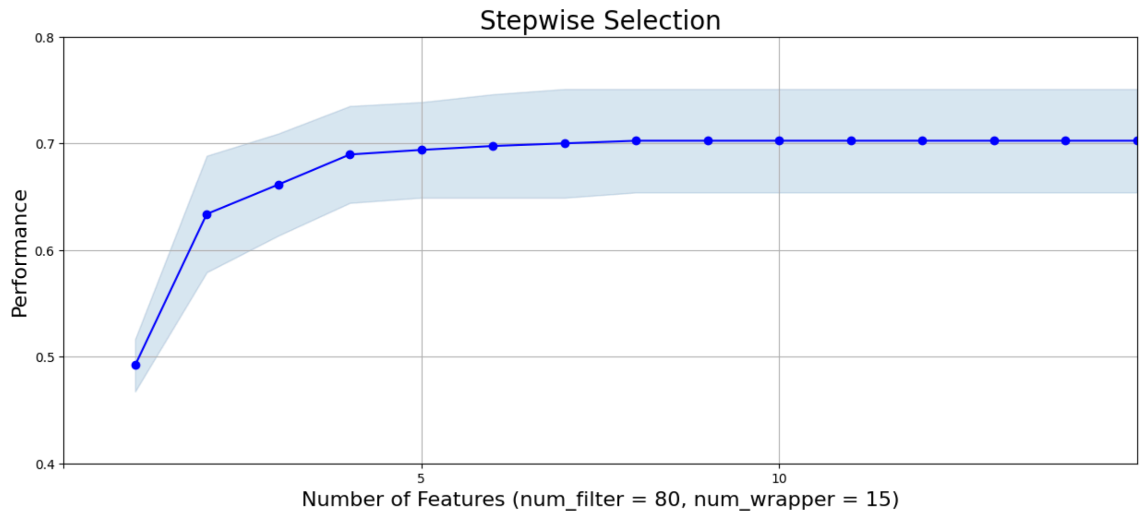


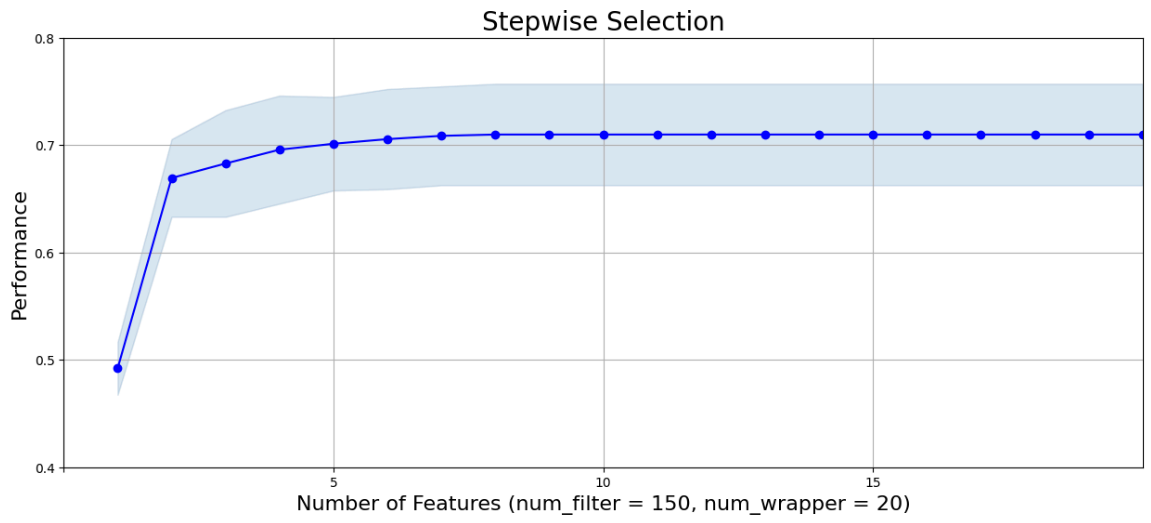
10) Fraud

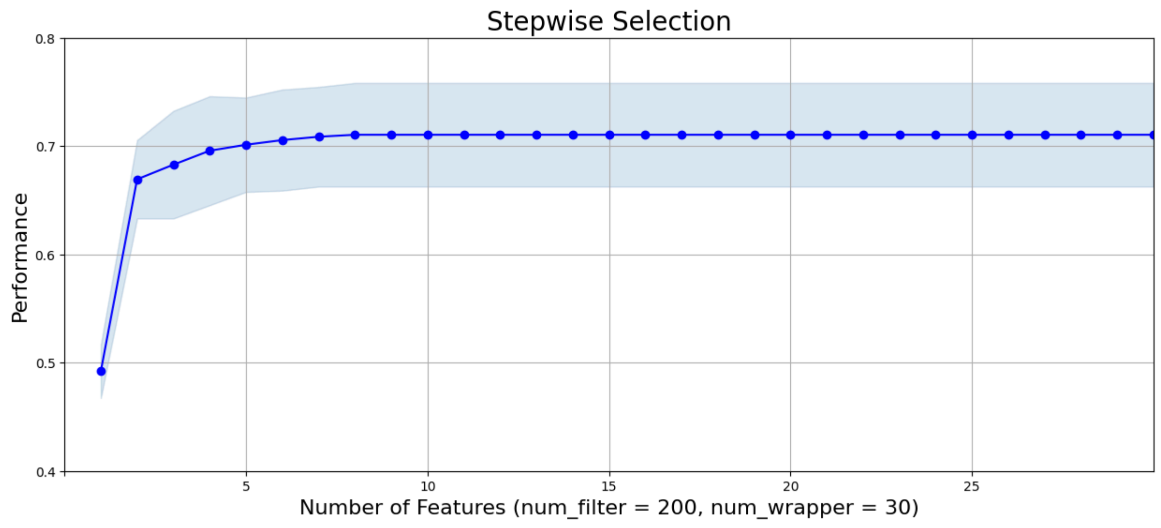
Description: A binary indicator (0 or 1) denotes whether the transaction was fraudulent or not. The total count of **Fraud = 0 is 95805**, and the total count of **Fraud = 1 is 2047.**



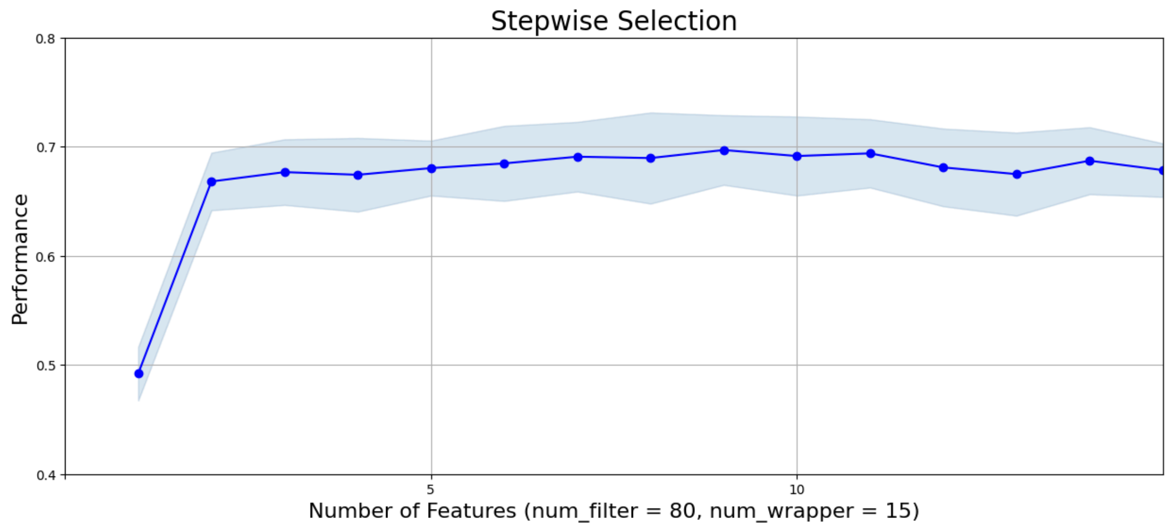
Appendix 2: Additional trials using LightGBM Forward

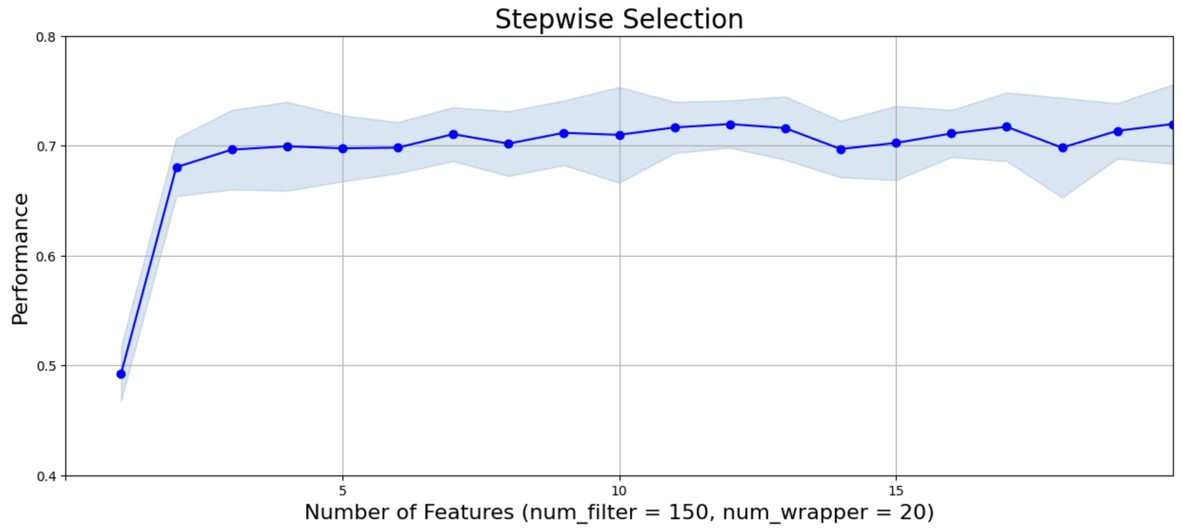


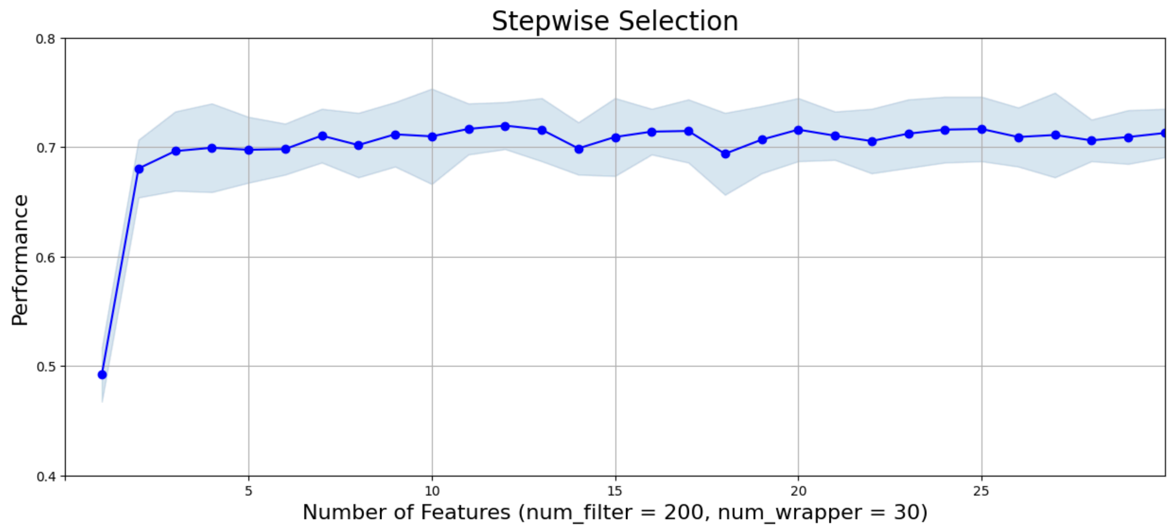




Appendix 3: Additional trials using Catboost Forward







References

[1], [2] ChatGPT, conversation with the author, May 10, 2024.

[3] Ansari, Y. (2023) *Understanding logistic regression: A beginner's guide*, *Medium*.

Available at:
https://medium.com/@novus_afk/understanding-logistic-regression-a-beginners-guide-73f148866910

[4] Tusyakdiah, H. (2020) *Klasifikasi decision tree Dengan R*, *Medium*.

Available at: https://halimatusyak.medium.com/klasifikasi-decision-tree-dengan-r-f12a0e48e060

[5] *Fraud detection in python* (2019) *Trenton McKinney*.

Available at: https://trenton3983.github.io/posts/fraud-detection-python/

[6] Mandot, P. (2018) *What is LIGHTGBM, how to implement it? how to fine tune the parameters?*, *Medium*. Available at:
https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

[7] Yang, H. *et al.* (2023) *CatBoost–Bayesian hybrid model adaptively coupled with modified theoretical equations for estimating the undrained shear strength of Clay*, *MDPI*. Available at:
https://www.mdpi.com/2076-3417/13/9/5418

[8] GeeksforGeeks (2023) *Artificial Neural Networks and its applications*, *GeeksforGeeks*.
Available at: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/

[9]*(PDF) a hybrid ensemble method for pulsar candidate classification*. Available at:
https://www.researchgate.net/publication/335483097_A_hybrid_ensemble_method_for_pulsar_candidate_classification\.