Fraud Analytics Report

Maehika Hegde

# Table of Contents

| **Sr. No.** | **Heading** | **Page** |
| --- | --- | --- |
| **1** | Executive Summary | 3 |
| **2** | Description of the Data | 5 |
| **3** | Data Cleaning | 11 |
| **4** | Variable Creation | 14 |
| **5** | Feature Selection | 21 |
| **6** | Preliminary Model Exploration | 40 |
| **7** | Final Model Performance | 55 |
| **8** | Financial Curves and Recommended Cutoff | 59 |
| **9** | Summary | 63 |
| **10** | Appendix | 67 |

# 1. Executive Summary

This project aims to leverage machine learning to enhance fraud detection in applications involving Personal Identifying Information (PII) such as names, addresses, Social Security Numbers (SSNs), and phone numbers. The business problem at the core of this project is the growing threat of identity fraud in high-value sectors, such as credit card and mobile phone applications, where fraudsters use stolen or fabricated identities to gain unauthorized access to products and services. As the volume of online transactions and applications continues to rise, so does the potential for fraudulent activity. Financial institutions and service providers are increasingly at risk of substantial losses due to fraudulent applications, with costs ranging from direct financial losses to the long-term reputational damage caused by failing to detect fraud.

In response to this challenge, the project focuses on building a robust fraud detection system that can effectively identify fraudulent applications in real-time while minimizing false positives that may result in unnecessary processing costs. Fraud detection is critical not only for preventing financial losses but also for ensuring compliance with regulations and safeguarding consumer trust. By accurately detecting fraud, businesses can mitigate these risks, protect their customers, and reduce the financial and operational burden of investigating and rectifying fraudulent activities.The dataset used in this analysis is a synthetic representation designed to closely mimic the statistical properties of real-world application data, offering a valuable tool for fraud detection analysis. This dataset includes approximately 1 million records and is primarily used to identify fraudulent applications for products like credit cards and mobile phones. The core challenge of the project was to clean and prepare the data while incorporating advanced machine learning models to optimize fraud detection. The project employed several machine learning techniques, including CatBoost and LightGBM, to create predictive models capable of detecting fraudulent applications with high accuracy. Data cleaning was a key component, with significant efforts put into resolving issues such as placeholder values, duplicate entries, and inconsistencies within the data. Key fields like SSNs, home phone numbers, and dates of birth were cleaned, and derived features such as "velocity count," "normalized velocity," and "cumulative velocity" were created to improve model performance.

After extensive experimentation with different models and configurations, a configuration using CatBoost emerged as the best performer, achieving a Fraud Detection Rate (FDR) of 3% for Out-of-Time (OOT) data. This high level of accuracy in detecting fraud was balanced by minimizing the costs associated with false positives. The model was optimized to ensure both economic efficiency and robustness, with an estimated savings potential of $3.1 billion by reducing fraud losses across large-scale applications. Additionally, financial curves, detection rate curves, and ROC curves were analyzed to determine the optimal cutoff, ensuring the model's operational efficiency by balancing fraud detection and minimizing false positives. The **0.68% cutoff** provides an optimal balance, offering a higher detection rate and slightly higher savings than the **0.3% cutoff**, making it the preferred choice for fraud detection in this scenario. This cutoff ensures the most effective fraud detection performance while maximizing the potential savings and minimizing false positive costs.

In conclusion, the project successfully delivered a powerful fraud detection solution that not only identifies fraudulent applications with high precision but also ensures cost-effective implementation. The final model’s generalization ability across different datasets, combined with its practical deployment potential, makes it a reliable tool for detecting and preventing fraud in real-world scenarios. By implementing this solution, businesses can reduce their exposure to fraudulent activities, improve their operational efficiency, and provide a more secure experience for their customers.

# 2. Description of the Data

This dataset is a synthetic version of real product application data designed to mimic the statistical properties of actual Personal Identifying Information (PII) commonly found in identity fraud detection tasks. This dataset includes fields such as names, Social Security Numbers (SSNs), addresses, dates of birth (dob), and phone numbers, which are the key PII elements used by identity fraud algorithms. The goal of the dataset is to detect fraud in applications, particularly for products like credit cards and cell phones. Although this is synthetic data, it is carefully crafted to reflect realistic patterns and relationships seen in billions of real U.S. applications over a decade, such as the frequency of names and SSNs, as well as linkages between fields (e.g., name-dob, SSN-phone). The dataset consists of around 1 million records and maintains distributions and relationships similar to actual data, providing a valuable tool for modeling and fraud detection analysis while treating most fields as categorical except for the two date fields.

## Summary Tables for Fields

### Numeric Fields

| **Field** | **# records with values** | **% populated** | **# zeros** | **# unique values** | **most common value** | **Min** | **Max** | **Median** | **Mean** | **Standard Deviation** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **date** | 1,000,000 | 100% | 0 | 365 | 2017-08-16 | 2017-01-01 | 2017-12-31 | 2017-07-02 | 20,170,668 | 344.99 |
| **dob** | 1,000,000 | 100% | 0 | 42,673 | 1907-06-26 | 1900-01-01 | 2016-10-31 | 1950-09-01 | 19,517,249 | 356,887 |

### Categorical Fields

| **Field** | **# records with values** | **% populated** | **# zeros** | **# unique values** | **most common value** | **Min** | **Max** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **firstname** | 1,000,000 | 100% | N/A | 78,136 | EAMSTRMT | N/A | N/A |
| **lastname** | 1,000,000 | 100% | N/A | 177,001 | ERJSAXA | N/A | N/A |
| **address** | 1,000,000 | 100% | N/A | 828,774 | 123 MAIN ST | N/A | N/A |
| **record** | 1,000,000 | 100% | 0 | 1,000,000 | 1 | 1 | 1,000,000 |
| **fraud\_label** | 1,000,000 | 100% | 985,607 | 2 | 0 | 0 | 1 |
| **ssn** | 1,000,000 | 100% | 0 | 835,819 | 999,999,999 | 36 | 999,999,999 |
| **zip5** | 1,000,000 | 100% | 0 | 26,370 | 68,138 | 2 | 99,999 |
| **homephone** | 1,000,000 | 100% | 0 | 28,244 | 9,999,999,999 | 593,799 | 9,999,999,999 |

## Description and Plots:

1. **Dob:**

**Description**: This field stores the date of birth of the applicants. It is used to verify the age of applicants, which can be important for eligibility and fraud detection purposes.

**Visualization**:

The bar plot shows the top 20 most common dates of birth in the dataset using a logarithmic scale. It highlights that certain dates, such as June 26, 1907, are far more frequent, potentially indicating default or placeholder values used in the data entry process.

The histogram with a KDE overlay visualizes the distribution of dates of birth, showing peaks and unusual concentrations that might warrant further investigation.

**A graph with numbers and a bar

Description automatically generated**

**A graph with a number of bars

Description automatically generated with medium confidence**

1. **Firstname:**

**Description**: This field contains the first names of applicants. It helps in identifying individuals and is often used in conjunction with other personal identifying information (PII) such as last name, date of birth, and SSN.

**Visualization**: The bar plot displays the top 20 most common first names in the dataset, showing the frequency distribution of these names.

**A graph with blue and green bars

Description automatically generated**

1. **Lastname:**

**Description**: This field contains the last names of applicants. Like the firstname field, it is essential for identifying individuals in the dataset and is part of the applicant's PII.

**Visualization**: The bar plot shows the top 20 most common last names, helping visualize the frequency of different surnames within the applicant data.

A graph of a number of names

Description automatically generated

1. **SSN:**

**Description**: This field contains the Social Security Numbers (SSNs) of applicants. SSNs are a critical piece of PII and are often used to identify individuals in the United States. This field is essential for detecting identity fraud when the same SSN is used in multiple applications or when unusual patterns are detected.

**Visualization**: The bar plot (log scale) displays the top 20 most common SSNs, highlighting instances where the same SSNs are repeatedly used, which could suggest fraudulent activity.

A graph with numbers and a bar

Description automatically generated

# 3. Data Cleaning

# 1. Identifying Categorical and Numeric Columns

The initial step in cleaning the data involved categorizing fields into either numerical or categorical fields. This process helps in understanding the type of data being dealt with, ensuring appropriate cleaning techniques are applied.  
  
While many fields, such as dates of birth (dob) and submission dates, are correctly identified as numerical, some fields such as Social Security Number (SSN), zip5, and homephone, though technically numeric, do not serve as true numerical values. These fields are better treated as categorical identifiers because their numeric properties are irrelevant to the analysis, and they act more like unique labels. They were manually switched to categorical fields to reflect their role accurately in identifying unique records.

# 2. Handling Placeholder and Frivolous Values

## i. SSN (Social Security Number) Placeholder Values

Problem:  
In the dataset, 16,935 records had a placeholder value of '999999999' for missing or unavailable Social Security Numbers (SSNs). SSNs are a critical piece of personal identifying information (PII) used in fraud detection algorithms. Leaving these placeholder values unaddressed would compromise the analysis, particularly when SSN is used as a unique identifier in models.  
  
Solution:  
To handle this, we flagged all placeholder SSNs and replaced them with unique identifiers taken from the 'record' field. This ensures that SSN placeholders no longer skew the results while still maintaining the integrity of the unique identity in each application.

A log-scaled bar chart was also generated to visualize the most frequent SSNs in the dataset. This helps highlight duplicates and placeholder entries that could indicate potential data quality issues.

Code handling the SSN cleanup:  
# Identify placeholder SSNs and replace them with corresponding record number  
len(data[data['ssn'] == 999999999]) # Count placeholder SSNs  
data.loc[data['ssn'] == 999999999, 'ssn'] = data.loc[data['ssn'] == 999999999, 'record']

## ii. Home Phone Placeholder Values

Problem:  
Similarly to SSNs, 78,512 records contained a placeholder value '9999999999' for missing or invalid home phone numbers. Phone numbers play an important role in verifying applicant identities, and placeholder values could lead to erroneous data analysis if not addressed.  
  
Solution:  
Like with SSNs, placeholder phone numbers were replaced with the corresponding 'record' values to ensure placeholders do not interfere with analysis. This approach allows for accurate representation of phone numbers without bias introduced by missing values.

To visualize this, a log-scaled bar chart was generated showing the frequency of phone numbers. This highlights common placeholder values and ensures that invalid phone numbers do not disproportionately affect the analysis.

Code handling the home phone cleanup:  
# Identify and replace placeholder phone numbers with record values  
len(data[data['homephone'] == 9999999999]) # Count placeholder phone numbers  
data.loc[data['homephone'] == 9999999999, 'homephone'] = data.loc[data['homephone'] == 9999999999, 'record']

# 3. Address and Zip Code Fields

Problem:  
The dataset contains both an 'address' column and a 'zip5' column, which often need to be combined for proper geographical analysis. Some addresses or zip codes are incomplete or incorrect, which can hinder fraud detection based on location data.  
  
Solution:  
To resolve this, a new 'fulladdress' column was created by concatenating the 'address' and 'zip5' fields. After generating this new column, the original 'address' and 'zip5' fields were dropped, simplifying the dataset and avoiding potential inconsistencies.

Code handling the address cleanup:  
# Combine address and zip5 into fulladdress  
data['fulladdress'] = data['address'] + ' ' + data['zip5'].astype('str')  
  
# Drop old address and zip5 columns  
data = data.drop(columns=['zip5', 'address'])

# 4. Date of Birth (DOB)

Problem:  
The 'dob' field is an important variable, particularly when analyzing the age of applicants for eligibility and fraud detection purposes. However, the dataset contained several placeholder or incorrect dates, such as '19070626', which was repeated in multiple records.  
  
Solution:  
To correct this issue, placeholder or incorrect dates were identified and replaced with corresponding record values. This ensures that the date of birth data is accurate and does not introduce bias into age-related analysis.

A bar chart with a log scale was also generated to show the frequency of common birthdates. This visualization helps in identifying suspicious or duplicated dates.

Code handling the DOB cleanup:  
# Identify and replace placeholder DOB with record numbers  
data.loc[data['dob'] == 19070626, 'dob'] = data.loc[data['dob'] == 19070626, 'record']

# 5. Visualizing Common Entries

To aid in the identification of placeholder values and duplicates, various visualizations were created. These included:  
- Bar charts (log scale) for SSN and home phone fields to detect duplicates and placeholder values.  
- Histograms for DOB to highlight unusual concentrations of birth dates that could indicate fraudulent entries.  
- Log-scaled bar charts for addresses to identify frequently repeated addresses, which may signal potential fraud.  
  
These visualizations were crucial in guiding the data cleaning process and ensuring no placeholder or invalid entries remained in the dataset.

Example of generating a log-scaled bar chart for SSN:  
ssn\_freq = data['ssn'].value\_counts().head(20)  
ssn\_freq.plot(kind='bar', logy=True)

# Conclusion

In conclusion, the systematic cleaning process addressed several critical data quality issues, including the removal of placeholder values in SSNs, home phone numbers, and dates of birth. Additionally, the combination of address and zip code fields into a single 'fulladdress' field further enhanced the dataset's usability for geographic analysis. By utilizing various visualizations, the data cleaning process was more effective in identifying and correcting issues, ensuring that the final dataset was free of frivolous entries. This cleaning process has prepared the data for further analysis, particularly for detecting fraud in personal identifying information.

# 4. Variable Creation

Fraud detection in applications involving Personal Identifying Information (PII) is crucial, as fraudsters typically use stolen or fabricated identities to gain unauthorized access to products and services, such as credit cards and mobile phones. Fraud often occurs through patterns such as repeated applications using similar or identical data, inconsistencies in identity information (e.g., a phone number or SSN appearing in multiple applications), or sudden spikes in application frequency, which can indicate suspicious behavior. Fraudulent applications are more likely to occur when fraudsters try to create multiple identities under different names or combine inconsistent data elements to bypass identity verification checks.

To identify such fraudulent activities, we created various new variables that help track velocity counts, relative velocity, and occurrences aggregated by different combinations of entities, as well as features to detect patterns such as unusual application frequency or sudden changes in behavior. These variables are designed to highlight anomalies and provide deeper insights into how fraudulent applications may manifest across time and different entities.

| **Variable Name** | **Description** | **# Variables Created** |
| --- | --- | --- |
| Original Variables | Original set of variables from the dataset. These are baseline features that describe various attributes of the entities involved in the analysis. | 24 |
| Velocity Count and Days Since | Measures velocity counts (activity or occurrences) and calculates days since last occurrence for 14 entities across a 7-day period.  Unique count of each entity within {0, 1, 3, 7, 14, 30, 60} days. | 98 |
| Relative Velocity | Captures the relative change in velocity (rate of occurrence or activity) for 16 entities over 7 different periods, helping identify trends or fluctuations. | 112 |
| Count by Entities | Count variables generated by aggregating occurrences across multiple combinations of entities, allowing granular analysis by entity and combination. | 1274 |
| Maximum Indicator | Indicators capturing the maximum occurrence or intensity of activity for 14 entities across four different time windows, identifying peak periods. | 56 |
| Age Indicator | Maximum, mean, and minimum ages of applicants at application time, grouped by age of each applicant. Tracks the age metric for each of the 14 entities, providing a measure of recency or seniority of the entities in the dataset. | 42 |

## Entities

| **Entity** | **Description** |
| --- | --- |
| ssn | Social Security Number |
| address | Full address of the applicant |
| zip5 | 5-digit zip code |
| dob | Date of Birth |
| homephone | Home phone number |
| name | Full name (first and last combined) |
| fulladdress | Full address with zip code |
| name\_dob | Combination of full name and date of birth |
| name\_fulladdress | Combination of full name and full address |
| name\_homephone | Combination of full name and home phone number |
| fulladdress\_dob | Combination of full address and date of birth |
| dob\_homephone | Combination of date of birth and home phone number |
| fulladdress\_homephone | Combination of full address and home phone number |
| homephone\_name\_dob | Combination of home phone number, full name, and date of birth |

**How Fraud Occurs**

Fraud in this context typically occurs when fraudsters attempt to use stolen or fabricated identities to gain unauthorized access to credit or services. Common patterns of fraudulent behavior include:

1. **Multiple Applications from the Same Location**: Fraudsters may submit applications from the same address but using different identities, creating multiple fraudulent accounts with different names and SSNs. This is often done to gain access to multiple lines of credit or services under separate accounts.
2. **Synthetic Identity Creation**: Fraudsters often create "synthetic" identities by combining real and fake information. For example, using a real SSN with a fabricated name and address can bypass traditional identity verification checks. Our models aim to detect these types of inconsistencies by comparing the activity of a particular SSN across different addresses or names.
3. **Multiple Applications Using the Same SSN**: Fraudsters may reuse a single SSN across several applications for different identities, a tactic known as synthetic identity fraud. This pattern can often go unnoticed unless the velocity and frequency of applications tied to the SSN are tracked over time.
4. **Unusual Spikes in Application Activity**: Fraudsters may try to exploit windows of opportunity by submitting a series of applications within a short time frame. These sudden spikes in activity can indicate attempts to open several fraudulent accounts quickly before the fraud is detected.

By tracking these patterns using the variables created above, we can better detect and prevent fraud, ensuring that businesses can protect their customers and minimize financial losses from fraudulent applications.

## New Suggested Variables

| **Variable Name** | **Description** | **# Variables Created** |
| --- | --- | --- |
| Application frequency ratio by address | Frequency of applications by address over {7, 30, 60} days. | 3 |
| Normalized velocity | Velocity counts normalized by average application frequency for each SSN over time windows. | 6 |
| Application day variance | Variance in application days within a month for each SSN. | 1 |
| Cumulative Velocity | Sum of velocity counts for each entity over the 7-day period, capturing total activity. | 14 |
| Standard Deviation of Counts | Standard deviation of counts for each entity across all periods, indicating consistency or fluctuation in activity levels. | 14 |

 **Application Frequency Ratio by Address**: Calculates application frequency for each address across rolling windows of 7, 30, and 60 days, allowing detection of frequency trends associated with specific addresses.

 **Normalized Velocity by SSN**: Measures application frequency by SSN over various time windows (0, 1, 3, 7, 14, 30 days) and normalizes it by the mean application frequency over a 30-day period. This normalization provides a standardized metric for comparing velocity across different SSNs, highlighting unusual frequency patterns.

 **Application Day Variance**: Computes the variance in application days within each month for each SSN, which can indicate irregular spikes in activity if variance is high.

 **Cumulative Velocity**: Sums the velocity counts for each entity over a 7-day period, providing insight into overall application activity within the timeframe.

 **Standard Deviation of Counts**: Calculates the standard deviation of application counts across all available periods for each entity, highlighting consistency or volatility in application activity.

## Code for New Suggested Variables

import pandas as pd

import numpy as np

data['date'] = pd.to\_datetime(data['date'], format='%Y%m%d')

data = data.sort\_values(by='date')

for days in [7, 30, 60]:

data[f'application\_frequency\_ratio\_address\_{days}d'] = (

data.groupby('address')['date']

.transform(lambda x: x.rolling(f'{days}D').count())

)

for days in [0, 1, 3, 7, 14, 30]:

data[f'velocity\_ssn\_{days}d'] = (

data.groupby('ssn')['date']

.transform(lambda x: x.rolling(f'{days}D').count())

)

data['mean\_application\_frequency\_ssn'] = data.groupby('ssn')['velocity\_ssn\_30d'].transform('mean')

for days in [0, 1, 3, 7, 14, 30]:

data[f'normalized\_velocity\_ssn\_{days}d'] = data[f'velocity\_ssn\_{days}d'] / data['mean\_application\_frequency\_ssn']

data = data.drop(columns=['mean\_application\_frequency\_ssn'])

data['month\_year'] = data['date'].dt.to\_period('M')

data['application\_day\_variance'] = (

data.groupby(['ssn', 'month\_year'])['date']

.transform(lambda x: x.dt.day.var())

)

data = data.drop(columns=['month\_year'])

data['cumulative\_velocity'] = data[[f'velocity\_day\_{i}' for i in range(1, 8)]].sum(axis=1)

data['std\_dev\_counts'] = data[[f'count\_period\_{i}' for i in range(1, 8)]].std(axis=1)

data.to\_csv("enhanced\_variables.csv", index=False)

data.head()

# 5. Feature Selection

Feature selection plays a critical role in the performance of any machine learning model, especially in tasks such as fraud detection. The goal of feature selection is to identify the most relevant features that help improve model performance while reducing the dimensionality of the dataset. By selecting the right features, we can make the model more interpretable, faster to train, and more effective at generalizing on unseen data.

**What is Feature Selection?**

Feature selection is the process of identifying and selecting the most important variables from a larger set of available features, aiming to improve the model’s performance by removing irrelevant or redundant features. In this project, we used different methods of feature selection to find the most significant features for detecting fraud in personal identifying information (PII). We applied both **forward selection** and **backward selection** techniques using **LightGBM** and **Random Forest** models to assess their impact on performance.

**Why is Feature Selection Important?**

Feature selection is important for several reasons:

1. **Reducing Overfitting**: By selecting only the most relevant features, we reduce the risk of overfitting, where the model learns noise in the training data rather than generalizable patterns.
2. **Improving Model Accuracy**: Selecting the right features can lead to better predictive performance, as irrelevant or redundant features may introduce noise and reduce the model's ability to generalize.
3. **Increasing Efficiency**: Fewer features mean that the model will be trained faster and be more computationally efficient.
4. **Improving Interpretability**: A simpler model with fewer features is easier to interpret and understand, making it easier for stakeholders to trust and act upon.

**How was Feature Selection Performed?**

We performed several experiments with different feature selection configurations using **LightGBM** and **Random Forest** models, with a focus on balancing exploration and computational efficiency.

**Experiment 1: LightGBM with Large Filter and Wrapper Sizes**

* **Configuration**: num\_filter = 150, num\_wrapper = 30
* **Model**: LightGBM, **Selection Type**: Forward
* **Observation**: This configuration allowed thorough exploration due to a high number of filters and wrappers, which might lead to overfitting. While this setup could find feature combinations that maximize performance, it risks noise inclusion and may lack generalizability.

A graph with blue dots

Description automatically generated

A screenshot of a computer

Description automatically generated

**Experiment 2: LightGBM with Balanced Filter and Wrapper Sizes (Optimal Configuration)**

* **Configuration**: num\_filter = 100, num\_wrapper = 20
* **Model**: LightGBM, **Selection Type**: Forward
* **Observation**: This configuration offered a balanced approach between exploration and efficiency, allowing the model to achieve stable performance with minimized overfitting risk. The forward selection process efficiently explored a reasonable feature space without excessive complexity. The model achieved consistent performance above 0.60, making it the most effective setup for maximizing LightGBM's potential with limited risk of overfitting.

A graph with a line

Description automatically generated

A screenshot of a computer

Description automatically generated

**Experiment 3: LightGBM with Small Filter and Wrapper Sizes**

* **Configuration**: num\_filter = 50, num\_wrapper = 20
* **Model**: LightGBM, **Selection Type**: Forward
* **Observation**: With a reduced number of filters, this configuration was quicker but lacked comprehensive exploration. As a result, it may miss significant feature interactions, leading to suboptimal model performance. A graph with numbers and a number of features

  Description automatically generated

A screenshot of a computer

Description automatically generated

**Experiment 4: LightGBM with Slightly Larger Filter Size**

* **Configuration**: num\_filter = 100, num\_wrapper = 10
* **Model**: LightGBM, **Selection Type**: Forward
* **Observation**: This setup provided reasonable feature exploration with a moderate wrapper size, yielding robust performance. However, the smaller wrapper size compared to Experiment 2 limited the model’s capacity to capture broader feature interactions, resulting in slightly lower performance stability.

A graph with a line going up

Description automatically generated

A screenshot of a program

Description automatically generated

**Experiment 5: LightGBM with Enhanced Filter Size**

* **Configuration**: num\_filter = 180, num\_wrapper = 20
* **Model**: LightGBM, **Selection Type**: Forward
* **Observation**: With a larger filter size, this configuration captured more features, providing extensive feature coverage. While it showed robust performance, the risk of overfitting increased with the larger number of filters, making this setup less efficient than Experiment 2.

A graph with a line

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Random Forest Models**

**Experiment 1: Random Forest with Moderate Filter and Wrapper Sizes**

* **Configuration**: num\_filter = 100, num\_wrapper = 10
* **Model**: Random Forest, **Selection Type**: Forward
* **Observation**: While Random Forest provided moderate performance, it lacked the consistency seen in LightGBM due to its sensitivity to selected features. This configuration highlighted the importance of model choice, as Random Forest showed less reliable performance compared to LightGBM with the same settings.

A graph with a line going up

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Experiment 2: Random Forest with Larger Wrapper**

**Case: 1**

* **Configuration**: num\_filter = 100, num\_wrapper = 20
* **Model**: Random Forest, **Selection Type**: Forward

**Goal**: Assessing stability and effectiveness of feature selection with Random Forest across two runs under identical configurations.

**Case: 1**

**Observation**: The model achieved a modest increase in performance with the larger wrapper size. However, the performance gains were marginal, and the Random Forest model exhibited variability across selected features. Despite the wrapper size allowing more feature combinations, the overall performance did not consistently improve. This suggests that Random Forest, while capturing some relevant features, struggles with consistency and is more prone to variance compared to LightGBM.

A graph showing a line of a number of features

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Case 2:**

**Observation**: In the second run, using the same num\_filter and num\_wrapper values, the model's performance fluctuated slightly from Case 1. This inconsistency in performance underscores Random Forest’s sensitivity to feature selection and data sampling. The model’s stochastic nature introduces variability across runs, making it harder to achieve reliable results. While the wrapper size enabled extensive feature exploration, the final model performance remained relatively unpredictable, indicating potential challenges in replicability.

A graph of a line

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Case 3:**

**Observation**: In **Case 3**, using the configuration of num\_filter = 100 and num\_wrapper = 20 with Random Forest, the performance initially rises quickly, peaking around 5-6 features before plateauing. This pattern suggests that additional features beyond the initial subset offer minimal improvement. The slight performance fluctuations visible in the shaded area indicate some variability across runs, underscoring Random Forest's sensitivity to feature selection and limited stability compared to LightGBM. Overall, while Random Forest achieves moderate performance with a smaller feature set, it struggles to leverage larger sets effectively, reinforcing the suitability of LightGBM for more consistent outcomes in feature selection tasks.

**A graph with a line going up

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Conclusion**

The three runs with identical configurations highlighted the following:

* **Stability**: Random Forest showed inherent instability in feature selection, resulting in variability across runs, which can make it challenging for consistent model performance.
* **Feature Selection Effectiveness**: Despite the increase in wrapper size, the marginal performance improvement suggests that Random Forest may not leverage the wrapper’s capacity as effectively as LightGBM. This reflects Random Forest’s limitations in achieving optimal performance through feature selection, making LightGBM a potentially more robust choice for this task

**Backward Selection with LightGBM**

**Experiment 1: LightGBM with Backward Selection**

* **Configuration**: num\_filter = 100, num\_wrapper = 10
* **Model**: LightGBM, **Selection Type**: Backward
* **Observation**: Backward selection identified a small set of critical features early in the process. However, it was less effective in exploring all feature interactions compared to forward selection, making it slightly less comprehensive.

**A graph showing a line

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Best Configuration & Conclusion**

**Best Configuration**: **Forward Selection in LightGBM (num\_filter = 100, num\_wrapper = 20)**

**Conclusion**: The optimal setup for this feature selection task is **Forward Selection with LightGBM using num\_filter = 100 and num\_wrapper = 20**. This configuration provides a balanced approach to feature exploration, allowing the model to consistently reach performance levels above 0.70 without excessive overfitting risks. It efficiently navigates the feature space, enabling LightGBM to maximize performance with limited computational overhead. This setup represents an ideal trade-off between model accuracy and feature selection efficiency, making it the recommended configuration for robust, generalized performance.

**Sekected Experiment 2: LightGBM with Balanced Filter and Wrapper Sizes (Optimal Configuration)**

* **Configuration**: num\_filter = 100, num\_wrapper = 20
* **Model**: LightGBM, **Selection Type**: Forward
* **Observation**: This configuration offered a balanced approach between exploration and efficiency, allowing the model to achieve stable performance with minimized overfitting risk. The forward selection process efficiently explored a reasonable feature space without excessive complexity. The model achieved consistent performance above 0.60, making it the most effective setup for maximizing LightGBM's potential with limited risk of overfitting.

A graph with a line

Description automatically generated

A screenshot of a computer

Description automatically generated

# 6. Preliminary Model Exploration

In this section, we provide a description of each machine learning model that was explored for fraud detection, including Logistic Regression, Decision Tree, Random Forest, LightGBM, Neural Network, and CatBoost. Each model was evaluated across different configurations, and the results were analyzed based on their Train, Test, and Out-of-Time (OOT) Fraud Detection Rate (FDR). These FDR scores give insight into each model’s performance, generalization, and effectiveness in detecting fraud.

**1. Logistic Regression**

* **Description**: Logistic Regression is a statistical model used for binary classification. It calculates the probability that an instance belongs to one of two classes by applying a logistic function to a linear combination of input features.
* **What It Does**: In fraud detection, logistic regression estimates the likelihood that an application is fraudulent based on features such as SSN, address, or other application-related attributes. It is simple to implement and interpret.
* **Why It Was Implemented**: Logistic regression is a good baseline model due to its simplicity and efficiency. It's particularly useful for understanding how individual features influence the prediction.
* **How It Was Used**: Different hyperparameters, such as regularization strength (C), solver type, and penalty (l1), were tested to control overfitting and regularization effects, with the primary goal being to identify a stable model with high generalization power.

**2. Decision Tree**

* **Description**: A Decision Tree is a non-linear classifier that splits the data into subsets based on feature values, aiming to create pure nodes with minimal class variance. It constructs a tree-like model of decisions and their possible consequences.
* **What It Does**: The decision tree algorithm recursively splits the dataset based on feature thresholds that maximize information gain (entropy) or minimize impurity (Gini index), ultimately creating branches that predict fraudulent applications.
* **Why It Was Implemented**: Decision Trees are easy to interpret and can capture non-linear relationships in the data. This is particularly useful when fraud patterns involve complex feature interactions.
* **How It Was Used**: Different hyperparameters such as max depth, minimum samples for splitting, and criterion (entropy) were explored to prevent overfitting and improve generalization while maintaining interpretability.

**3. Random Forest**

* **Description**: Random Forest is an ensemble method that combines multiple decision trees, each trained on random subsets of the data and features. It uses majority voting to make the final prediction.
* **What It Does**: Random Forest improves upon a single Decision Tree by averaging the predictions of many trees, reducing overfitting and increasing robustness against noise in the data.
* **Why It Was Implemented**: Random Forest is highly effective in handling high-dimensional datasets and capturing complex relationships, making it suitable for fraud detection tasks where interactions between multiple features may indicate fraudulent behavior.
* **How It Was Used**: Hyperparameters like the number of trees (n\_estimators), maximum depth, and minimum samples for leaf nodes were tuned to find the optimal configuration that balances model performance and computational efficiency.

**4. LightGBM**

* **Description**: LightGBM (Light Gradient Boosting Machine) is an advanced gradient boosting framework that uses decision tree learning algorithms. It is known for its efficiency and high performance on large datasets.
* **What It Does**: LightGBM builds multiple trees sequentially, where each tree attempts to correct the errors made by the previous tree. It uses histogram-based methods for faster training and better accuracy.
* **Why It Was Implemented**: LightGBM excels in high-dimensional datasets with complex relationships. It is especially useful for fraud detection where feature interactions are key to identifying fraudulent patterns.
* **How It Was Used**: Several hyperparameters such as subsample fraction, max depth, learning rate, and number of boosting rounds (n\_estimators) were tested. The model’s ability to handle large datasets and its speed made it a strong candidate.

**5. Neural Network**

* **Description**: A Neural Network is a computational model inspired by the structure of the human brain. It consists of layers of nodes (neurons) that transform input data into output through learned weights.
* **What It Does**: Neural Networks are capable of capturing complex, non-linear relationships in data. For fraud detection, they can learn intricate patterns in the features of legitimate vs. fraudulent applications.
* **Why It Was Implemented**: Neural Networks are powerful models that can learn deep representations of data, which can be crucial when detecting subtle fraud patterns in high-dimensional datasets.
* **How It Was Used**: The neural network configurations tested varied the activation function (ReLU, tanh), learning rate, number of hidden layers, and solver type (SGD, Adam) to find a configuration that balanced performance and minimized overfitting.

**6. CatBoost**

* **Description**: CatBoost (Categorical Boosting) is a gradient boosting algorithm that is particularly well-suited for datasets with categorical features. It handles categorical variables efficiently without the need for explicit encoding.
* **What It Does**: Like LightGBM, CatBoost builds trees sequentially to correct the errors made by previous trees. It introduces advanced techniques for handling categorical data, reducing preprocessing time and improving performance.
* **Why It Was Implemented**: CatBoost was chosen for its ability to handle categorical features natively, making it ideal for fraud detection tasks where many features (such as SSN, address, and phone numbers) are categorical. It also offers great performance and generalization with minimal overfitting.
* **How It Was Used**: Key hyperparameters such as subsample, max depth, learning rate, and number of trees (n\_estimators) were adjusted. CatBoost's built-in feature for handling categorical data without manual encoding made it efficient and robust for this task.

The below table presents configurations and results for various machine learning models, including Logistic Regression, Decision Tree, Random Forest, LightGBM, Neural Network, and CatBoost. For each model type, multiple iterations with different hyperparameter settings are shown, including their **Train FDR** (Fraud Detection Rate), **Test FDR**, and **Out-of-Time (OOT) FDR** scores. The FDR scores provide insight into each model’s performance across training, testing, and out-of-time validation datasets, allowing for an evaluation of the model's effectiveness and generalization ability.

Each model configuration is tuned by adjusting hyperparameters like max\_depth, min\_samples\_split, n\_estimators, and others, depending on the specific model type. The primary goal of varying these parameters is to balance model complexity, performance, and generalization.

**Part 1: Hyperparameter Tuning**

A table of numbers and numbers

Description automatically generated

**Model Configurations and Hyperparameters**

1. **Logistic Regression**
   * **Iterations and Hyperparameters**: Five configurations are presented, each adjusting the Penalty, C, Solver, and l1\_ratio hyperparameters.
   * **Penalty**: All configurations use an l1 penalty, which performs feature selection by forcing some coefficients to zero. This helps in reducing model complexity and improving interpretability by selecting the most influential features.
   * **C (Regularization Strength)**: Values of 0.01, 0.1, and 1 are explored. Lower values of C imply stronger regularization, preventing overfitting by discouraging large weights.
   * **Solver**: The solvers used are liblinear and saga. liblinear is efficient for small datasets, while saga is optimized for larger datasets and supports both l1 and l2 regularization.
   * **Performance Summary**: The Train, Test, and OOT FDR scores are fairly consistent across configurations, with minor differences. Regularization likely helped in achieving similar performance across datasets, which indicates good generalization.
2. **Decision Tree**
   * **Iterations and Hyperparameters**: Five configurations using different values for Criterion, Max Depth, Min Samples Split, Min Samples Leaf, and Splitter.
   * **Criterion**: Only entropy is used, which measures the information gain of splits. It tends to result in deeper trees that are more selective in feature splits, making it suitable for datasets where capturing subtle interactions is necessary.
   * **Max Depth**: All configurations are limited to a depth of 10, controlling the tree's size to avoid overfitting. Shallow trees generalize better but may underfit complex data.
   * **Min Samples Split** and **Min Samples Leaf**: These values (120 and 60 in some configurations) enforce minimum samples required to split a node or remain a leaf, reducing the risk of overfitting on small data splits.
   * **Splitter**: The random splitter adds randomness to the split selection, which can improve generalization by reducing variance.
   * **Performance Summary**: Train FDR scores are relatively high, but Test and OOT scores are lower, indicating slight overfitting. The chosen hyperparameters helped in maintaining manageable model complexity.
3. **Random Forest**
   * **Iterations and Hyperparameters**: Configurations vary Criterion, Max Depth, Min Samples Split, Min Samples Leaf, n\_estimators, and Bootstrap.
   * **Criterion**: Both gini and entropy are used. gini is faster computationally, while entropy might yield slightly more accurate splits.
   * **Max Depth**: Values of 8 and 12 constrain the depth, ensuring each tree is not too deep, which would increase variance and overfitting.
   * **Min Samples Split** and **Min Samples Leaf**: Used to prevent splitting on very small samples, these parameters control each tree’s complexity.
   * **n\_estimators**: Values like 100, 150, and 200 are explored. More trees (higher n\_estimators) improve accuracy but increase computational cost. This parameter balances model stability with runtime.
   * **Bootstrap**: Bootstrapping (sampling with replacement) is applied to reduce variance and improve model stability.
   * **Performance Summary**: Train FDR scores are generally higher than Test and OOT scores, suggesting some overfitting, but the use of bootstrap and min\_samples parameters helps manage this.
4. **LightGBM**
   * **Iterations and Hyperparameters**: Five configurations vary Subsample, Max Depth, Learning Rate, n\_estimators, and Metric.
   * **Subsample**: Values like 1, 0.9, and 0.6 indicate the fraction of data used in each iteration, which reduces overfitting by introducing randomness in data.
   * **Max Depth**: Constraining depth, with values from 10 to 25 or None, manages model complexity. Unlimited depth allows capturing more nuances in data but increases the risk of overfitting.
   * **Learning Rate**: Ranges from 0.01 to 0.1, controlling the step size for each update, with lower values providing more fine-grained control and stability in convergence.
   * **n\_estimators**: Specifies the number of boosting rounds. Higher values generally improve accuracy but increase computational cost.
   * **Metric**: Different metrics (logloss, binary\_error, auc) are used to guide model training, depending on which aspect of performance is prioritized.
   * **Performance Summary**: LightGBM models show competitive FDR scores, with minimal overfitting. Subsampling and learning rate control are effective in balancing model fit and generalization.
5. **Neural Network**
   * **Iterations and Hyperparameters**: Five configurations vary Activation, Alpha, Learning Rate, Hidden Layer Sizes, and Solver.
   * **Activation**: Functions like tanh and relu are used. tanh is well-suited for normalized data, while relu is faster and prevents vanishing gradients, making it popular for deep networks.
   * **Alpha**: This regularization parameter prevents overfitting by penalizing large weights.
   * **Learning Rate**: Lower values (0.001 to 0.01) are used for stable convergence. Neural networks benefit from small learning rates, especially when paired with iterative solvers.
   * **Hidden Layer Sizes**: Different architectures like (100, 50) and (150, 100) allow for exploration of model complexity. Larger hidden layers can capture more data complexity but risk overfitting.
   * **Solver**: sgd and adam solvers are chosen. adam is generally faster and adaptive, while sgd is stable for specific applications.
   * **Performance Summary**: The neural network models show good Train and Test FDR scores but some variability in OOT scores, suggesting sensitivity to hyperparameter choices.
6. **CatBoost**
   * **Iterations and Hyperparameters**: Configurations vary Subsample, Max Depth, Learning Rate, n\_estimators, and L2 Leaf Reg.
   * **Subsample**: Similar to LightGBM, subsample values (0.85, 0.9, and 1) introduce randomness, which enhances generalization.
   * **Max Depth**: Values range from 6 to 15. Higher depths capture more interactions but increase risk of overfitting.
   * **Learning Rate**: Smaller values (0.01 to 0.1) control how quickly the model learns, allowing for fine-tuning.
   * **n\_estimators**: Controls the number of trees, similar to boosting rounds in LightGBM. More trees generally improve performance.
   * **L2 Leaf Reg**: L2 regularization on leaves helps reduce overfitting by penalizing large values in leaf predictions.
   * **Performance Summary**: CatBoost models perform well, with competitive Train, Test, and OOT FDR scores. The combination of subsampling, learning rate, and leaf regularization provides balanced models with good generalization.

**Part 2: Comparison Plots**

A diagram of a graph

Description automatically generated with medium confidence

The plot displays the Train, Test, and OOT FDR scores for each model across various configurations. Each boxplot shows the range of FDR scores across iterations, with Train (blue), Test (red), and OOT (green) scores depicted for each model.

**1. Logistic Regression**

* **Performance Analysis**: Logistic Regression shows a relatively narrow spread in FDR scores across Train, Test, and OOT sets, with Train and Test scores clustering tightly around 0.70, while OOT scores hover slightly below.
* **Generalization**: The consistency between Train, Test, and OOT scores indicates that Logistic Regression has strong generalization capabilities and is less prone to overfitting compared to more complex models. The small spread suggests robustness across different splits, making it a reliable model for consistent performance.
* **Potential Drawbacks**: Despite its stability, Logistic Regression does not reach the highest FDR scores compared to other models. It achieves moderate FDR levels, which may be a limitation if the goal is to maximize detection power.

**2. Decision Tree**

* **Performance Analysis**: The Decision Tree model shows higher variability in FDR scores, particularly in the Test and OOT sets. The Train FDR scores are relatively high, indicating that the model fits the training data well, but the Test and OOT FDR scores are notably lower, with OOT scores showing more spread and some outliers.
* **Overfitting**: This significant difference between Train and Test/OOT FDR suggests that the Decision Tree model is overfitting, capturing noise in the training data that does not generalize well to new data.
* **Potential Drawbacks**: While Decision Trees can capture complex patterns, the high Train FDR paired with lower Test and OOT scores indicates poor generalization. This model may struggle with unseen data, making it less reliable for long-term performance.

**3. Random Forest**

* **Performance Analysis**: Random Forest displays a balanced spread of FDR scores, with a slight drop from Train to Test and OOT scores. Train scores are generally high, showing that the model captures training patterns effectively, while Test and OOT scores remain relatively close to each other, albeit slightly lower than Train.
* **Generalization**: The use of multiple trees in Random Forest helps improve generalization by reducing overfitting compared to a single Decision Tree. However, the decrease in OOT scores hints at a mild tendency toward overfitting, but to a lesser degree than the standalone Decision Tree model.
* **Potential Drawbacks**: The OOT scores are relatively stable but lower than the Test scores, suggesting that Random Forest might have limited predictive power on completely unseen data, especially if the data distribution shifts over time.

**4. LightGBM**

* **Performance Analysis**: LightGBM shows a noticeable separation between Train, Test, and OOT scores, with Train FDR reaching high values, Test FDR slightly lower, and OOT FDR showing a wider spread. LightGBM exhibits strong performance on training data but demonstrates some instability on the OOT set.
* **Overfitting**: The drop from Train to OOT scores is more pronounced in LightGBM, suggesting a risk of overfitting. This model is capturing intricate details in the training data but may be struggling to generalize to out-of-time samples.
* **Potential Drawbacks**: While LightGBM can provide high FDR scores, especially on Test data, its sensitivity to time-based data splits (as evidenced by the lower OOT FDR) raises concerns about its ability to maintain high performance over time.

**5. Neural Network**

* **Performance Analysis**: The Neural Network model exhibits high variability in FDR scores, with notable differences between Train, Test, and OOT sets. Train FDR scores are high, but Test and OOT scores are lower, with OOT scores showing the widest spread.
* **Overfitting**: The significant difference between Train and Test/OOT scores highlights overfitting in the Neural Network model. Neural networks tend to overfit when they capture complex patterns in the training data that may not be present in new data.
* **Potential Drawbacks**: Although Neural Networks are powerful and flexible, their tendency to overfit means they may not be the best choice for this task, especially if robustness to data shifts is required.

**6. CatBoost**

* **Performance Analysis**: CatBoost shows a narrower range of FDR scores for Train, Test, and OOT, with relatively consistent performance across different data splits. The Test and OOT scores for CatBoost are close, indicating strong generalization ability.
* **Generalization**: The relatively minor spread between Train, Test, and OOT scores demonstrates that CatBoost is capable of capturing patterns in the data without overfitting. This suggests a high level of stability and reliability.
* **Potential Drawbacks**: Although CatBoost’s FDR scores may not be the absolute highest in terms of Train FDR, its consistency across Test and OOT sets makes it a strong contender, especially for applications where stability is crucial.

**Part 3: Model Selection and Justification**

**Selected Model: CatBoost**

**Justification for Selection:**

1. **Generalization and Stability**:
   * **CatBoost** shows the most consistent performance across Train, Test, and Out-of-Time (OOT) FDR scores, with minimal drops and limited variability. This stability suggests that CatBoost is less prone to overfitting, making it a reliable choice for real-world applications where data distributions can change over time.
   * **Key Hyperparameters**:
     + **Subsample**: This parameter controls the fraction of data used in each boosting iteration. By setting subsample values below 1 (e.g., 0.85 or 0.9), CatBoost introduces randomness into each iteration, helping to prevent overfitting and improving generalization.
     + **L2 Leaf Regularization**: This regularization parameter penalizes overly complex leaf values, ensuring that each tree doesn’t capture too much noise from the training data. This helps reduce overfitting and contributes to CatBoost’s stable performance across different datasets.
2. **Balanced Performance**:
   * While some models, like LightGBM or Neural Networks, achieve higher Train FDR scores, they experience noticeable drops in Test and OOT scores, indicating potential overfitting. In contrast, CatBoost achieves competitive FDR scores without significant overfitting, balancing performance with generalization.
   * **Key Hyperparameters**:
     + **Max Depth**: Limiting the depth of trees (e.g., setting max\_depth to values like 6 to 15) helps control the model’s complexity. By setting this parameter appropriately, CatBoost avoids learning overly specific patterns in the training data, leading to better generalization on Test and OOT data.
     + **Learning Rate**: CatBoost’s learning\_rate controls the contribution of each tree to the overall model. Lower values (e.g., 0.01 or 0.05) allow the model to learn gradually, reducing the risk of overfitting and enhancing the model's ability to generalize across datasets.
3. **Robustness to Time-Based Shifts**:
   * The OOT FDR scores for CatBoost are very close to its Test FDR scores, suggesting that the model handles out-of-time data well. This robustness is particularly valuable in production settings, where the data seen by the model may not perfectly match the training data distribution.
   * **Key Hyperparameters**:
     + **n\_estimators**: This parameter controls the number of boosting rounds or trees in the model. More trees (e.g., 100 to 300) allow CatBoost to capture complex relationships, while the gradual learning with a fixed learning rate reduces sensitivity to data shifts over time.
     + **Bootstrap Type**: By selecting an appropriate bootstrap method, CatBoost can reduce variance. Poisson or Bayesian bootstrap methods, for instance, help create slight variations in each training subset, which can improve the model’s robustness to data shifts.
4. **Interpretability and Efficiency**:
   * Compared to complex models like Neural Networks, CatBoost provides a more interpretable structure, especially when using shallow trees or restricted depths. Additionally, CatBoost has efficient built-in handling for categorical features, which can simplify preprocessing and improve computational efficiency.
   * **Key Hyperparameters**:
     + **Categorical Feature Handling**: CatBoost natively supports categorical features without requiring manual encoding. By automatically handling these features, CatBoost reduces preprocessing time and helps maintain feature interpretability, making it easier to understand which categories contribute most to predictions.
     + **Depth and Leaf Estimation**: With optimized depth and leaf estimation, CatBoost controls tree complexity without compromising interpretability. This helps in balancing the model’s performance with its transparency, which can be valuable for understanding feature importance.

**Conclusion:**

Given its ability to generalize across different datasets, robust performance on the OOT set, and minimal overfitting, **CatBoost** is the most suitable model for this task. It combines a high FDR score with stability, making it well-suited for deployment in real-world scenarios where data may evolve over time. The hyperparameters used in CatBoost, such as subsample, L2 leaf regularization, max depth, learning rate, and its native handling of categorical features, ensure a balanced configuration that can capture complex patterns without compromising on predictive consistency across various data subsets.

By carefully tuning these hyperparameters, CatBoost achieves a model that is not only accurate but also interpretable, efficient, and robust to data shifts—qualities that make it an ideal choice for dynamic and production-grade environments.

**Part 4: Why CatBoost Over Neural Network?**

When choosing the best model for this project, both **CatBoost** and **Neural Network** were strong contenders. While Neural Networks are powerful and flexible, CatBoost was ultimately selected as the optimal model due to several key factors:

**1. Generalization and Stability Across Data Splits**

* **CatBoost** demonstrated consistent performance across Train, Test, and Out-of-Time (OOT) FDR scores, with minimal drops and a tight range. This suggests that CatBoost can generalize well to new data without overfitting to the training data.
* **Neural Network** showed a significant difference between Train and Test/OOT FDR scores, indicating overfitting. Neural Networks often excel at capturing complex relationships in the data, but without a very large dataset, they may capture patterns that don’t generalize well to new data. In this case, the Neural Network’s performance on OOT data was more variable, suggesting it may struggle in a real-world setting where data can shift over time.

**2. Data Requirements and Complexity**

* **CatBoost** is particularly well-suited for moderate-sized datasets and does not require extensive preprocessing. Its ability to handle categorical data without additional encoding simplifies data preparation and reduces the likelihood of overfitting.
* **Neural Network** models typically require larger datasets to achieve optimal performance, as they rely on learning complex patterns through layers of neurons. With a smaller or moderate dataset, Neural Networks are more likely to overfit, as evidenced in this case. Additionally, Neural Networks often require more preprocessing, including feature scaling and normalization, adding to their complexity.

**3. Interpretability and Practicality**

* **CatBoost** offers a reasonable level of interpretability, especially with feature importance scores and tools like **SHAP (SHapley Additive exPlanations)**. This allows us to understand which features are driving the model's predictions, which is valuable for applications requiring transparency, such as fraud detection or credit scoring.
* **Neural Network** models are inherently less interpretable due to their layered structure and complex weight configurations. While techniques like SHAP or LIME can be applied, interpreting a Neural Network's decision-making process remains challenging. This lack of interpretability can be a drawback in scenarios where understanding and explaining model decisions to stakeholders is important.

**4. Computational Efficiency and Ease of Tuning**

* **CatBoost** is optimized for efficiency, making it faster to train on tabular data. With **Ordered Boosting**, CatBoost avoids overfitting and can handle categorical features natively, reducing both computational cost and preprocessing time. CatBoost also offers relatively straightforward hyperparameter tuning compared to Neural Networks.
* **Neural Network** models are computationally intensive, especially when using deep architectures or large numbers of neurons. They often require GPU resources to train efficiently, which may not be practical in all environments. Additionally, Neural Networks have a large hyperparameter space (number of layers, neurons per layer, learning rate, activation functions, etc.), making tuning more complex and time-consuming.

**5. Handling of Time-Based Data (Out-of-Time Performance)**

* **CatBoost** showed robust performance on OOT data, with OOT FDR scores close to Test FDR scores. This suggests that CatBoost can handle slight shifts in data distribution over time, which is essential in real-world applications where data patterns may change.
* **Neural Network** models can be sensitive to temporal shifts unless specifically designed for time-series data (e.g., LSTM or GRU architectures). In this case, the Neural Network exhibited a wider spread in OOT FDR scores, indicating that it may not adapt as well to changes in data distribution over time.

**6. Ease of Deployment and Maintenance**

* **CatBoost** is widely used in industry due to its efficiency and ease of deployment. It can be deployed on standard infrastructure without requiring specialized hardware, and its model size is manageable compared to deep Neural Networks. Additionally, CatBoost’s structured nature makes it easier to monitor and maintain in a production environment.
* **Neural Network** models, particularly deeper networks, can be challenging to deploy due to high resource requirements and larger model sizes. They may require specialized hardware (like GPUs) for efficient inference, and maintaining complex Neural Network architectures in production can be demanding, especially for tasks where low-latency predictions are critical.

**Conclusion: Why CatBoost is the Best Model**

Based on the above comparison, **CatBoost** was chosen over Neural Network for this project due to its balanced performance, generalization ability, interpretability, and ease of deployment. CatBoost’s consistent Train, Test, and OOT FDR scores indicate strong generalization without significant overfitting, which is essential for a model intended to perform reliably in real-world scenarios where data distributions can change over time. Additionally, CatBoost’s efficiency, native handling of categorical data, and interpretability make it a practical choice for both development and deployment.

In summary, **CatBoost outperforms Neural Network** in this context because:

* It achieves high and stable FDR scores across all datasets, showing minimal overfitting.
* It’s computationally efficient, easy to tune, and can be deployed with standard infrastructure.
* It provides interpretability, which is valuable for applications requiring transparency.
* It handles out-of-time data well, making it robust to potential shifts in data distribution.

This combination of attributes makes CatBoost the best choice for a practical, high-performance solution, ensuring reliability and adaptability for future data variations.

# 7. Final Model Performance

**Final Model: CatBoost**

CatBoost, a gradient boosting algorithm designed for categorical feature handling, emerged as the best-performing model for fraud detection. Below is a description of the key hyperparameters that were fine-tuned to achieve optimal performance across the **Training**, **Testing**, and **Out-of-Time (OOT)** datasets:

1. **Subsample**:
   * **Description**: This parameter controls the fraction of data used for each boosting iteration. By setting subsample values between 0.85 and 1, the model introduces randomness, which enhances generalization by reducing overfitting.
   * **Optimal Value**: 0.85–0.9, balancing model fit with regularization.
2. **Max Depth**:
   * **Description**: Controls the depth of individual trees. Deeper trees capture more complex interactions but risk overfitting.
   * **Optimal Value**: Between 6 and 15, ensuring trees are complex enough to capture meaningful patterns without overfitting.
3. **Learning Rate**:
   * **Description**: Determines how quickly the model learns during each boosting round. Smaller values offer finer adjustments to the model.
   * **Optimal Value**: Ranges from 0.01 to 0.1, providing gradual, stable learning while preventing overshooting.
4. **n\_estimators**:
   * **Description**: Specifies the number of boosting rounds or trees in the model. More trees generally improve performance but at the cost of higher computational complexity.
   * **Optimal Value**: 100–200 trees, ensuring sufficient model complexity without excessive computation.
5. **L2 Leaf Regularization (L2 Leaf Reg)**:
   * **Description**: Regularization technique applied to leaf predictions, reducing overfitting by penalizing large values in the leaves.
   * **Optimal Value**: Values around 0.1–1 to strike the right balance between overfitting prevention and model complexity.

**Performance Across Datasets**

**1. Training Data Performance**

* **Fraud Detection Rate (FDR)**: The CatBoost model demonstrated strong performance in detecting fraudulent applications with an FDR consistently above 0.70.
* **Cumulative Statistics**: As the training dataset was processed, CatBoost achieved high cumulative fraud detection rates, with minimal overfitting shown in **KS** and **FPR** metrics.
* **Overall Savings**: The model resulted in substantial **Fraud Savings**, with **FP Loss** relatively low, indicating effective fraud detection without too many false positives.

**2. Testing Data Performance**

* **FDR Consistency**: When evaluated on the testing dataset, the CatBoost model continued to maintain high FDRs, although with slight drops compared to training data, indicating that it was generalizing well but not overfitting.
* **Cumulative Statistics**: The FDR, **KS** values, and **FP Loss** remained consistent, showing that the model was performing well across unseen data, further proving its stability and generalization ability.
* **Overall Savings**: The **Overall Savings** continued to be high, suggesting that fraud detection was efficient and cost-effective even in unseen testing data.

**3. Out-of-Time (OOT) Performance**

* **FDR in OOT**: The CatBoost model performed reasonably well on **OOT** data, with an FDR of **0.68**, indicating its robustness to shifts in the data distribution over time.
* **Cumulative Statistics**: The **KS** and **FPR** scores showed that, even with time-based data shifts, CatBoost was able to identify fraudulent applications effectively, with minimal loss in performance.
* **Overall Savings**: The **Overall Savings** for the OOT set remained substantial, demonstrating that CatBoost can still deliver significant fraud savings while controlling false positives, even with data not seen during training.

A table with numbers and a number of objects

Description automatically generated with medium confidence

A table with numbers and text

Description automatically generated

A screenshot of a spreadsheet

Description automatically generated

# 8. Financial Curves and Recommended Cutoff

**Financial Curves, Detection Rate Curve, and ROC Curve Explanation**

The following three curves provide insights into the performance and effectiveness of the fraud detection model. Each curve highlights different aspects of model evaluation, helping to find the optimal balance between fraud detection and minimizing costs.

1. **Financial Curves:**
   * The financial curves illustrate the trade-off between **fraud detection savings** and the **cost of false positives**.
   * **Green Curve (Fraud Detection Savings):** Shows the increasing savings as the cutoff for detecting fraud increases. Initially, at lower cutoffs, fewer fraud cases are detected, resulting in lower savings. As the cutoff rises, the model detects more fraud, leading to higher savings but at the risk of increasing false positives.
   * **Blue Curve (Cost of False Positives):** Illustrates the cost incurred when legitimate applications are incorrectly flagged as fraudulent. As the cutoff increases, false positives increase, and so does the cost.
   * **Red Curve (Total Cost):** Represents the total cost, which combines both the savings from fraud detection and the costs from false positives. The goal is to identify the optimal cutoff where fraud detection is maximized while keeping the false positive costs within an acceptable range. This balance ensures the model is both effective and economically efficient.
2. **Detection Rate Curve:**
   * This curve plots the **Detection Rate (DR)** against the **score cutoff percentage**.
   * **Interpretation:** As the cutoff increases, the detection rate rises sharply, meaning that the model identifies more fraudulent applications. However, this increase eventually levels off, indicating diminishing returns after a certain threshold.
   * **Use:** The Detection Rate curve helps identify the cutoff point at which the model achieves a satisfactory balance between identifying fraud and avoiding excessive false positives.
3. **ROC Curve (Receiver Operating Characteristic):**
   * The ROC curve plots the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)**.
   * **Interpretation:** The closer the curve is to the top-left corner, the better the model performs, indicating a higher TPR and a lower FPR. A higher TPR means the model correctly identifies more fraud, while a lower FPR means fewer legitimate applications are incorrectly flagged.
   * **Use:** The ROC curve is useful for evaluating the model's overall performance across different cutoffs and for comparing the model's ability to distinguish between fraudulent and legitimate applications.

**Overall Explanation:**

By analyzing these curves together, we can determine the optimal cutoff for the fraud detection model. The financial curves show the trade-off between savings and costs, the Detection Rate curve highlights the effectiveness of fraud detection at different cutoffs, and the ROC curve assesses the model's performance in distinguishing fraudulent applications. The goal is to select a cutoff that maximizes fraud detection savings while minimizing false positive costs and ensuring the model generalizes well across different data scenarios.

A graph of a line

Description automatically generated with medium confidence

Max possible savings: 3,146,760,000.0

A graph with a line

Description automatically generated

Based on the analysis of the **Financial Curves**, **Detection Rate (DR)** curve, and **ROC curve**, I recommend a **cutoff of 0.68%** for the **CatBoost** model.

A graph with a line

Description automatically generated

**Recommended Logic:**

The **0.68% cutoff** offers an optimal balance for **CatBoost** based on the following factors:

1. **Maximized Fraud Detection**: The higher cutoff leads to detecting a larger portion of fraudulent transactions, represented by the green curve. The **Fraud Savings** increase significantly as we detect more fraud.
2. **Minimized False Positives**: While increasing the cutoff further may increase **fraud detection savings**, it also significantly increases **False Positive Losses** (blue curve). At **0.68%**, the **False Positive Loss** is controlled, ensuring operational costs remain manageable.
3. **Maximizing Net Savings**: By balancing both **fraud savings** and **false positive loss**, **0.68%** minimizes the **overall cost**, maximizing net savings. The **red curve**, representing total cost, reaches its optimal point at this threshold.

**Summary:**

The **0.68% cutoff** is the most effective choice as it provides a good balance between detecting fraud (high DR) and minimizing false positive losses. It achieves high **overall savings** and demonstrates robust performance on **Train**, **Test**, and **OOT** datasets, making it the most suitable for fraud detection in this scenario.

# 9. Summary

This project focused on leveraging machine learning to detect fraud in applications involving Personal Identifying Information (PII) such as Social Security Numbers (SSNs), names, addresses, and phone numbers. The goal was to develop a robust fraud detection system capable of identifying fraudulent applications in real-time while minimizing false positives to avoid unnecessary processing costs.

**Steps Performed in the Project**

1. **Description of the Data**:
   * The dataset used in this project was synthetic but closely modeled after real-world data from applications for products like credit cards and mobile phones. It included fields such as names, SSNs, addresses, dates of birth (dob), and phone numbers, which are key elements used in fraud detection tasks. The dataset contained approximately 1 million records, mimicking the statistical properties of real application data.
2. **Data Cleaning**:
   * Data cleaning was a critical step to ensure data quality. Key issues addressed included:
     + **Placeholder values**: SSNs and home phone numbers with placeholder values (e.g., '999999999') were replaced with unique identifiers.
     + **Address and zip code**: Combined the 'address' and 'zip5' fields into a single 'fulladdress' field for better geographical analysis.
     + **Date of Birth (DOB)**: Corrected invalid or placeholder DOBs and removed duplicates.
     + **Visualizations**: Various plots (log scale bar charts for SSN, home phone, and DOB) helped identify placeholder and duplicate entries, improving the cleaning process.
3. **Variable Creation**:
   * New variables were created to enhance fraud detection, such as:
     + **Velocity count** and **normalized velocity**: Measures of activity for entities over different time windows (7, 30, 60 days).
     + **Cumulative velocity** and **standard deviation of counts**: To identify spikes in activity indicative of fraudulent behavior.
     + **Age indicators**: For detecting patterns related to age and identity discrepancies.
     + **Fraud detection patterns**: These new features helped detect repeated patterns, synthetic identities, and unusual spikes in application activity.
4. **Feature Selection**:
   * Various feature selection techniques were tested to identify the most significant features for fraud detection:
     + **Forward selection** and **backward selection** were used with models like LightGBM and Random Forest.
     + **Experiments** varied filter and wrapper sizes, with the optimal configuration identified as **num\_filter = 100, num\_wrapper = 20** for LightGBM, which resulted in consistent performance while avoiding overfitting.
5. **Preliminary Model Exploration**:
   * Multiple machine learning models were tested, including:
     + **Logistic Regression**: Provided a simple baseline with stable performance.
     + **Decision Tree**: Captured non-linear patterns but suffered from overfitting.
     + **Random Forest**: A robust ensemble method that performed well with multiple trees.
     + **LightGBM**: An advanced gradient boosting method that showed competitive results with high performance.
     + **Neural Network**: Powerful but prone to overfitting, requiring larger datasets to perform optimally.
     + **CatBoost**: The best-performing model with consistent FDR scores across Train, Test, and OOT datasets.
6. **Model Selection and Justification**:
   * **CatBoost** was selected as the final model due to:
     + **Generalization and stability**: Consistent performance across Train, Test, and OOT datasets.
     + **Handling categorical features**: Efficient native handling of categorical features.
     + **Interpretability**: Better understanding of feature importance.
     + **Robustness to data shifts**: Minimal performance drop in OOT data.
7. **Final Model Performance**:
   * **CatBoost** achieved a high **FDR (Fraud Detection Rate)** of **0.68%** in Out-of-Time (OOT) data, with robust performance across Train, Test, and OOT datasets.
   * The model showed significant **fraud savings** and relatively low **false positive loss**, making it cost-effective.
8. **Financial Curves and Recommended Cutoff**:
   * **Cutoff Recommendation**: After analyzing the **Financial Curves**, **Detection Rate (DR) curve**, and **ROC curve**, the optimal cutoff was determined to be **0.68%**. This cutoff provided an ideal balance between fraud detection and minimizing false positive costs, maximizing overall savings.
9. **Model Performance Statement**:
   * The **CatBoost** model demonstrated excellent performance with an **FDR of 0.68%** in OOT data, achieving high fraud detection rates while maintaining low false positive rates. The model is robust and generalizes well across different datasets, making it suitable for deployment in real-world applications where fraud detection is critical.

**Conclusion:**

The project successfully developed a robust fraud detection system using **CatBoost**, which provided excellent performance across training, testing, and out-of-time data. By carefully cleaning and preparing the data, creating new variables to capture fraud patterns, and selecting the best features, the model was able to detect fraud efficiently. The optimal cutoff of **0.68%** ensured a balance between fraud detection savings and the cost of false positives, resulting in significant financial savings.

**Max possible savings** achieved by implementing this solution amounted to **$3,146,760,000.0**, showcasing the substantial potential of the fraud detection model in minimizing fraud losses and improving operational efficiency.

# 10. Appendix

## Summary Tables for Fields

### Numeric Fields

| **Field** | **# records with values** | **% populated** | **# zeros** | **# unique values** | **most common value** | **Min** | **Max** | **Median** | **Mean** | **Standard Deviation** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **date** | 1,000,000 | 100% | 0 | 365 | 2017-08-16 | 2017-01-01 | 2017-12-31 | 2017-07-02 | 20,170,668 | 344.99 |
| **dob** | 1,000,000 | 100% | 0 | 42,673 | 1907-06-26 | 1900-01-01 | 2016-10-31 | 1950-09-01 | 19,517,249 | 356,887 |

### Categorical Fields

| **Field** | **# records with values** | **% populated** | **# zeros** | **# unique values** | **most common value** | **Min** | **Max** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **firstname** | 1,000,000 | 100% | N/A | 78,136 | EAMSTRMT | N/A | N/A |
| **lastname** | 1,000,000 | 100% | N/A | 177,001 | ERJSAXA | N/A | N/A |
| **address** | 1,000,000 | 100% | N/A | 828,774 | 123 MAIN ST | N/A | N/A |
| **record** | 1,000,000 | 100% | 0 | 1,000,000 | 1 | 1 | 1,000,000 |
| **fraud\_label** | 1,000,000 | 100% | 985,607 | 2 | 0 | 0 | 1 |
| **ssn** | 1,000,000 | 100% | 0 | 835,819 | 999,999,999 | 36 | 999,999,999 |
| **zip5** | 1,000,000 | 100% | 0 | 26,370 | 68,138 | 2 | 99,999 |
| **homephone** | 1,000,000 | 100% | 0 | 28,244 | 9,999,999,999 | 593,799 | 9,999,999,999 |

## Description and Plots:

1. **Date:**

**Description**: This field represents the application submission date. It is used to track when the application was submitted. The data is typically displayed on a daily, monthly, or yearly basis to identify trends in the volume of applications over time.

**Visualization**: The plot shows the fluctuation in the number of daily applications over the course of 2017, helping identify trends or seasonal patterns in the submission rate.

**A graph showing a number of applications

Description automatically generated with medium confidence**

1. **Dob:**

**Description**: This field stores the date of birth of the applicants. It is used to verify the age of applicants, which can be important for eligibility and fraud detection purposes.

**Visualization**:

The bar plot shows the top 20 most common dates of birth in the dataset using a logarithmic scale. It highlights that certain dates, such as June 26, 1907, are far more frequent, potentially indicating default or placeholder values used in the data entry process.

The histogram with a KDE overlay visualizes the distribution of dates of birth, showing peaks and unusual concentrations that might warrant further investigation.

**A graph with numbers and a bar

Description automatically generated**

**A graph with a number of bars

Description automatically generated with medium confidence**

1. **Firstname:**

**Description**: This field contains the first names of applicants. It helps in identifying individuals and is often used in conjunction with other personal identifying information (PII) such as last name, date of birth, and SSN.

**Visualization**: The bar plot displays the top 20 most common first names in the dataset, showing the frequency distribution of these names.

**A graph with blue and green bars

Description automatically generated**

1. **Lastname:**

**Description**: This field contains the last names of applicants. Like the firstname field, it is essential for identifying individuals in the dataset and is part of the applicant's PII.

**Visualization**: The bar plot shows the top 20 most common last names, helping visualize the frequency of different surnames within the applicant data.

A graph of a number of names

Description automatically generated

1. **Address:**

**Description**: This field stores the residential addresses of applicants. It helps in verifying the location of applicants and detecting potential fraud patterns, especially when the same address is associated with multiple applicants.

**Visualization**: The bar plot (log scale) highlights the top 20 most common addresses, showing how frequently specific addresses appear, which could indicate fraudulent activity if certain addresses are overrepresented.

A graph of address and address numbers

Description automatically generated

1. **fraud\_label:**

**Description**: This binary field indicates whether an application has been flagged as fraudulent. A value of 0 means no fraud, and a value of 1 means the application is suspected or confirmed as fraud.

**Visualization**: The bar plot shows the distribution of fraud\_label values, providing a clear view of the imbalance between fraudulent and non-fraudulent applications.

A graph with a bar and a bar chart

Description automatically generated with medium confidence

1. **SSN:**

**Description**: This field contains the Social Security Numbers (SSNs) of applicants. SSNs are a critical piece of PII and are often used to identify individuals in the United States. This field is essential for detecting identity fraud when the same SSN is used in multiple applications or when unusual patterns are detected.

**Visualization**: The bar plot (log scale) displays the top 20 most common SSNs, highlighting instances where the same SSNs are repeatedly used, which could suggest fraudulent activity.

A graph with numbers and a bar

Description automatically generated

1. **zip5:**

**Description**: This field contains the 5-digit ZIP codes of applicants' addresses. ZIP codes are useful for geographic analysis, identifying regions with high application volumes, or detecting location-based fraud trends.

**Visualization**: The bar plot (log scale) shows the top 20 most common ZIP codes, providing insights into the geographical concentration of applications.

A graph of a number of zip codes

Description automatically generated

1. **homephone:**

**Description**: This field contains the home phone numbers of applicants. Like address information, phone numbers are used to verify the applicant's identity and may indicate suspicious activity if the same phone number is used across multiple applications.

**Visualization**: The bar plot (log scale) visualizes the top 20 most common home phone numbers, helping identify frequently used numbers that could signal potential fraud.  
A graph of a number and a number

Description automatically generated