Applications Fraud Analytics Project Report

Shreyash Kondakindi

September 21, 2025

Contents

1 Executive Summary

We are tackling the problem of detecting fraudulent account applications in a large-scale setting. Our goal is to identify bogus credit card and cell phone service applications before they result in losses. Using one year of application data, we built and evaluated various fraud detection models. The final model we developed can flag a large portion of fraudulent applications while only sending a very small percentage of all applications for manual review. In an independent out-of-time test, the model was able to capture about **60% of fraud cases by reviewing just the top 3% of applications**, which translates to roughly **$3 billion** in estimated annual fraud savings. This report details the data, our methodology, the model's performance, and recommendations for deployment, covering everything from initial data exploration to the final financial impact analysis.

2 Description of the Data

The dataset consists of approximately **1,000,000** application records for credit card and cell phone accounts, spanning roughly one year of activity (2017). Each record contains **10 fields**: two date fields (application date and date of birth), seven personal identifying information (PII) fields (such as name, Social Security Number, phone number, email, address, etc.), and one binary fraud label. The fraud label indicates whether the application was identified as fraudulent (`1`) or legitimate (`0`). The overall incidence of fraud in the data is low, about **1.5%** (approximately 15,000 fraud cases out of 1,000,000 applications), which is consistent with the severe class imbalance typically seen in application fraud data.

Most fields in the dataset are well-populated, and there are minimal missing values aside from intentional placeholder entries (addressed during cleaning). For example, both date fields are 100% complete, and identity fields like SSN and phone have values for the vast majority of records (with any placeholder or dummy values handled in preprocessing). Applicants' birth dates range from 1900 to 2016, meaning applicant ages span from infants to 117 years old at the time of application. The age distribution is broad, with a concentration in typical adult age ranges; this reflects a wide customer base, and the data generation ensured even coverage of ages without unrealistic spikes.

Two illustrative distributions are shown in the figures below. Figure 1 shows the breakdown of legitimate vs. fraudulent applications, highlighting the predominance of non-fraudulent cases. Figure 2 shows the distribution of applicant ages, which is roughly bell-shaped across adulthood and tapers at very young and very old ages (a few applicants in the extremes appear due to the synthetic nature of the data).

For modeling purposes, we partitioned the data chronologically to enable robust evaluation. The majority of the data (e.g., applications from January through October 2017) was used for model training and validation, while the final portion (e.g., November–December 2017) was held out as an out-of-time (OOT) test set. This OOT set—representing future data that the model was not trained on—allows us to assess how well the model generalizes to new applications beyond the training period.

Field Name	Field Type	% Populated	# Zeros	# Unique Values	Most Common
firstname	categorical	100.00%	0	78136	EAMSTRMT
lastname	categorical	100.00%	0	177001	ERJSAXA
address	categorical	100.00%	0	828774	123 MAIN ST
record	categorical	100.00%	0	1000000	1
fraud_label	categorical	100.00%	985607	2	0
ssn	categorical	100.00%	0	835819	999999999
zip5	categorical	100.00%	0	26370	68138
homephone	categorical	100.00%	0	28244	9999999999

Table 1: Summary of Categorical Fields

Field Name	Field Type	% Populated	# Zeros	Min	Max	Most Common
date	numeric	100.00%	0	2017-01-01	2017-12-31	2017-08-16
dob	numeric	100.00%	0	1900-01-01	2016-10-31	1907-06-26

Table 2: Summary of Numeric/Date columns

Figure 1: Distribution of fraud labels in the application data. The vast majority of applications are legitimate (label 0), with only about 1.5% labeled as fraud (label 1). This severe class imbalance highlights the importance of targeting a small review percentage to capture the rare fraud cases.

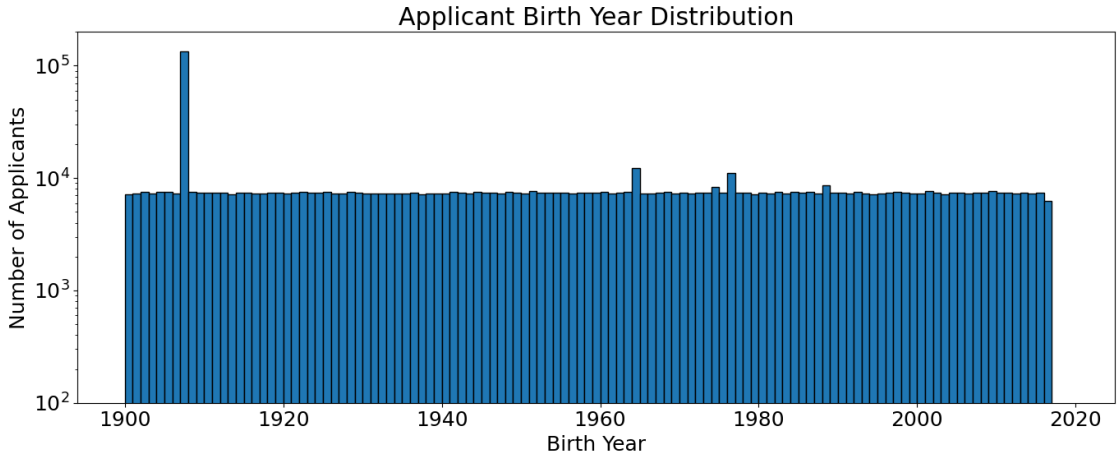


Figure 2: Distribution of applicant ages. Applicants range from very young to very old (ages 0 to 117), with a concentration in adult age ranges (e.g., many applicants in their 20s–40s). The broad age distribution reflects a diverse applicant pool; extreme ages appear in small numbers due to the synthetic data generation.

Record Exclusions. We excluded the first two weeks of application records (records 1–38511) from feature creation because early variables were not fully formed for those initial days. Similarly, applications from November–December 2017 (records 833509–1,000,000) were withheld as an *out-of-time (OOT)* test set. This ensures that model development used only January–October data (records 38512–833508), allowing us to evaluate performance on genuinely unseen future applications.

3 Data Cleaning

Understanding Fraud Behavior. In application fraud, criminals often submit multiple applications using the same stolen or fabricated PII within a short window of time, hoping to open accounts before detection. They also frequently use "dummy" or placeholder fields (e.g., SSN = 999999999, fake address, bogus phone number) to evade verification. By cleaning these fields and removing spurious values, we prevent multiple unrelated fraudulent attempts from appearing as one genuine customer record.

Before feature engineering and modeling, the raw application data was carefully cleaned to ensure quality and consistency. Several key cleaning steps were implemented:

3.1 Cleaning Frivolous PII Field Values

A critical cleaning step is to handle "dummy" or placeholder values in PII fields that do not represent real, distinctive information. Such frivolous values are often used by applicants trying to hide or when information is not available, and if left unaddressed they could cause unrelated applications to appear falsely linked. We identify several fields and values for this treatment:

- **Social Security Number (SSN):** Instances of SSN given as `999999999` (or equivalently formatted `999-99-9999`) are dummy placeholders. These are replaced with a null or special value (e.g., an empty string or `"000000000"`) to indicate a missing or invalid SSN. This prevents multiple applications with SSN 999-99-9999 from being linked together as if they shared a legitimate SSN.
- **Phone Number:** Likewise, phone numbers given as `9999999999` (all 9's) are recognized as bogus. We replace these with a null or placeholder (e.g., `"0000000000"` or an explicit `"dummy_phone"` tag) to avoid spurious matches on phone number.
- **Street Address:** A known fake address pattern is `"123 MAIN ST68138"` (which appears to concatenate a fake street with a zip code). This and similar obviously fake addresses are replaced with a null or generic placeholder (such as `"UNKNOWN"` address). By doing so, we ensure that two different applicants who both put "123 MAIN ST68138" are not considered to live at the same real address for linking purposes.
- **Other PII fields:** We similarly check for trivial or filler values in other fields like email or name. For instance, an email of `"test@test.com"` or a last name `"LN"` might be placeholders. Any such occurrences are standardized to a null/placeholder to avoid misleading links.

These replacements are implemented using pandas operations, for example by applying conditional logic or using the `replace()` method for each field. After this cleaning, any PII field that contained a frivolous value is either null (which the model can treat as missing) or set to a harmless constant that is used consistently. This prevents the creation of features that erroneously tie together many records (for example, without cleaning, dozens of records might share the SSN `999-99-9999` and thus appear as the same person). By nullifying these dummy values, subsequent feature engineering steps will treat them as missing rather than genuine matches.

To facilitate entity-based feature engineering, we create new composite keys by concatenating existing fields. These entity keys represent different granularities of identity and are used to link applications that share common personal information:

- **Full Name Key:** We combine the first name and last name into a single `fullname` key (e.g., `"John"` + `"Doe"` = `"JohnDoe"`). Prior to concatenation, names are standardized (trimmed and case-normalized) so that `"JOHN"` and `"John"` both map to `"john"` for consistency. The full name key helps capture cases where the same person (or two people with identical names) apply multiple times.

- **Name+DOB Key:** We further combine the full name with date of birth to create a `name_dob` key. For example, `"JohnDoe"` + `"1980-01-15"` might yield `"JohnDoe1980-01-15"`. This key is a more precise identifier of an individual, since two people with the same name but different birthdates will have different `name_dob` keys. It helps to distinguish common names and link records that truly refer to the same person.

- **Address Key:** We may also construct an address composite, such as concatenating street address with ZIP code (after standardizing the format). This yields an `address_zip` key representing a unique location. Similarly, one could form a `name_address` key if needed (combining an applicant's name with their address), which might capture household or family fraud patterns.

- **Other Keys:** Depending on the data, additional keys are constructed. For example, a phone-based key (combining phone area code with last name, or just using phone number alone as a key), or email-based keys (like email prefix + last name) can be derived. We also treat the SSN itself (cleaned as above) as an entity key on its own.

These composite keys are created using straightforward string concatenation in pandas (often via the `astype(str)` and string addition or using `df.apply` with a lambda). By generating these keys, we enable the model to consider whether two applications share a certain piece of identity information. For instance, the same `name_dob` appearing in two records strongly suggests the same real person, which could indicate repeat behavior. In the next steps, we will use these keys to derive features like counts and recent frequencies of occurrences.

3.2 Feature Engineering

Using the base fields and newly created keys, we engineer a comprehensive set of features that capture patterns of application behavior and linkages between applications. We describe each group of features below.

3.2.1 Days Since Last Occurrence

For each entity (each type of key or identifying field), we calculate the *days since last match* feature. This feature measures the recency of the previous application that shared this entity with the current application. It is computed by sorting the data chronologically by application date and, for each record and each key:

- Find the most recent prior application (if any) that has the same key value.

- Compute the difference in days between the current application's date and that prior application's date.

If no prior match is found (meaning this is the first time that particular value appears), we assign a default value (such as a large number or a special null indicator) to signify "no previous occurrence."

For example, if an SSN `123-45-6789` was seen in an application on June 1 and then again on June 10, the second occurrence would have `ssn_days_since_last` = 9 days. This is implemented efficiently by grouping the DataFrame by each key (such as SSN, `name_dob`, phone, etc.), sorting within each group by date, and using a shift operation to access the previous date for subtraction. The rationale is that a short interval since a previous application with the same information might indicate rapid re-use of identity data, which can be a fraud signal. We create a separate `days_since_last` feature for each key type (e.g., `ssn_days_since_last`, `name_dob_days_since_last`, `phone_days_since_last`, etc.). Each of these quantifies the recency of activity for that identifier.

3.2.2 Recent Velocity Count Features

We also generate features that count how many times each entity has appeared in recent time windows prior to the current application—often referred to as *velocity* features. Specifically, for each key and for various look-back windows (0 days, 1 day, 3 days, 7 days, 14 days, and 30 days), we calculate the number of applications (including or up to the current one) that have the same key and occurred within that window ending on the current application's date:

- **0-day count (same-day count):** The number of other applications that occurred on the same day as the current one with the same key. For example, `ssn_count_0` for an application would be the count of applications with the same SSN on that exact date (excluding the current record itself, this would count other applications on that date).
- **1-day count:** The count of applications in the last 1 day (typically meaning the same day and the day before). `name_count_1` for instance would tell how many applications with the same full name occurred within the past 24 hours prior to this application.
- **3-day, 7-day, 14-day, 30-day counts:** Similarly, these features count occurrences within the last 3, 7, 14, or 30 days, respectively. For instance, `phone_count_7` gives the number of applications in the week leading up to (and including) the current application that share the same phone number.

These features are calculated by filtering the dataset by date range for each record and key. In practice, a rolling or expanding window method is used: as we iterate through applications in chronological order, we maintain counters of how many previous records with the same key fall within the relevant time window. This can be optimized with cumulative sums or by using date indexes to quickly slice the past X days of data for each key. Velocity features capture the idea of burstiness or recent frequency. A higher count in a short window (e.g., many applications with the same data in the past week) can signal suspicious behavior such as one person trying multiple times or sharing of identity information across applications. We generate such count features for all the important entity keys (SSN, full name, name+DOB, phone, address, etc.), each across the multiple time windows. For example, representative features include `ssn_count_3`, `name_dob_count_30`, `address_zip_count_7`, and so on.

3.2.3 Relative Velocity Ratios

In addition to raw counts, we compute relative velocity features, which are ratios comparing counts across different time windows. These ratios help normalize the raw counts and indicate acceleration or deceleration in activity. For instance, a feature like `count_0_by_3` (using a generic name here) would be defined as the count in the last 0 days divided by the count in the last 3 days for the

same entity. Concretely, if an SSN has `ssn_count_0` = 2 (two other same-day applications) and `ssn_count_3` = 5 (five applications in the last 3 days), then `ssn_count_0_by_3` = 2/5 = 0.4. A higher ratio (close to 1) would mean most of the last 3 days' activities happened today (a spike of activity), whereas a lower ratio indicates the activity was more spread out. We create such ratio features for relevant window pairs, typically an immediate window to a slightly larger window. Examples include:

- `count_0_by_3`: Ratio of same-day count to three-day count.
- `count_3_by_7`: Ratio of three-day count to weekly count.
- `count_7_by_30`: Ratio of weekly count to monthly count.

These ratios are computed with care to avoid division by zero. In implementation, we ensure that if the denominator count is zero (meaning no occurrences in the longer window), the ratio is handled (for example, set to 0 if numerator is also zero, or flagged specially if numerator exists without denominator, though the latter case typically cannot happen if a shorter window count is nonzero the longer window count should also be nonzero). The relative velocity features help highlight surges in activity: for example, a high `name_dob_count_0_by_7` would signal that the person has submitted many applications today compared to the whole week.

3.2.4 Cross-Entity Unique Counts

Another group of features captures the diversity of cross-entity relationships, i.e., how many unique values of one identifier are associated with another. We generate features that quantify, for each record, the number of unique second-entity values seen per first-entity value (based on historical data up to that point). In simpler terms, for each pair of entity types (like SSN and address, name and phone, etc.), we ask:

- How many unique X's has this Y been associated with?

where X and Y are two different entity fields. For example:

- **Unique addresses per SSN:** For the SSN on the current application, how many distinct street addresses have we seen associated with that SSN across all applications? This becomes a feature, e.g., `ssn_unique_address_count`. A fraudster might use the same SSN with many different addresses, which would make this count high.
- **Unique SSNs per address:** Conversely, `address_unique_ssn_count` counts how many distinct SSNs have been used for the given address. A drop address (used for fraud) might have many people (many SSNs) linked to it.
- **Unique phones per name:** `name_unique_phone_count` would count how many different phone numbers have been used by applications with the same name.
- **Unique names per phone:** `phone_unique_name_count` counts how many different names shared this phone number.

And so on for all combinations deemed relevant. We implement this by iterating through pairs of keys using nested loops or a double `groupby`. For each entity type A and another entity type B, we do a groupby on A and aggregate the number of unique values of B in each group. That result is then brought back into the dataset: each record with a given A value receives the unique count of B associated with that A. In pandas, one approach is:

```
unique_counts = df.groupby(A)[B].nunique()
df['A_unique_B_count'] = df[A].map(unique_counts)
```

This assigns to each record the precomputed unique count for its A value. We repeat such calculations for each (A, B) pair. By doing so prior to model training (and ideally in a way that avoids future data leakage by using only past data up to the record's date when calculating the count), we get features that tell us, for example, if an identity element is reused across many others. These cross-entity features are powerful for fraud detection because they can unveil many-to-many relationships indicative of fraud rings or identity sharing. A legitimate user typically has one SSN used with maybe a couple of addresses over time; if we see one SSN with 10+ unique addresses, that stands out. Likewise, one address associated with dozens of SSNs is suspect. Representative features from this group include those mentioned above, as well as combinations like `ssn_unique_phone_count`, `name_dob_unique_device_count` (if device ID is a field), etc., covering all important pairs of identifiers in the data.

3.2.5 Day-of-Week Fraud Risk

Some exploratory analysis of the data may reveal that the fraud rate varies by the day of week an application is submitted (for instance, perhaps weekend applications have a higher or lower fraud incidence). To capture this pattern without overfitting to noise, we create a feature representing the fraud risk associated with the application's day-of-week, using a smoothed estimate. First, we compute the observed fraud rate for each day of week (Monday through Sunday) from the historical data. Let y_{dow} denote the fraud rate for a particular day-of-week and $\bar{y}$ the overall fraud rate across all days. Because some days might have fewer observations (e.g., maybe fewer applications on Sunday), we apply a smoothing formula to pull extreme rates toward the overall average if the sample size is low. The smoothing is done using a logistic function based on the count of observations for that day. Specifically, for each day-of-week, we calculate a smoothed risk $y_{\text{dow\_smooth}}$ as:

$$y_{\text{dow\_smooth}} = \bar{y} + \frac{y_{\text{dow}} - \bar{y}}{1 + \exp\left(-\frac{\text{num} - \text{mid}}{c}\right)}$$

, where num is the number of applications on that day-of-week (in the training data), n_{mid} is a chosen midpoint count at which the adjustment is half, and c is a scaling factor controlling the steepness of the logistic function. This equation ensures that if num is small (few samples for that day), the denominator is large and $y_{\text{dow\_smooth}}$ stays closer to $\bar{y}$ (the global rate). If num is large, the fraction approaches $(y_{\text{dow}} - \bar{y})$ and the day-specific rate is trusted more. We then create a feature in the dataset for each application's day-of-week risk: the application date is converted to a weekday (0–6 or Monday–Sunday), and we map that to the smoothed fraud rate $y_{\text{dow\_smooth}}$ computed above. This gives a numeric feature (essentially an encoded day-of-week effect). It provides the model a sense of whether that particular day is relatively riskier or safer, while avoiding the high variance that a raw per-day fraud rate might introduce.

Overall, these cleaning steps ensured that the dataset used for modeling was free of obvious errors and inconsistencies. By replacing frivolous PII entries with nulls and removing irrelevant fields, we avoided situations where unrelated fraudulent applications could be linked through fake identical information. The data was left in a pristine condition, ready for feature engineering and analysis with correct types and meaningful values.

4 Variable Creation

After cleaning the data, we engineered new features to expose patterns characteristic of fraud. In the context of application fraud, suspicious behavior often involves reuse of personal information across multiple applications or multiple applications submitted in a short time frame. We therefore focused on two main categories of derived variables: *velocity features* and *composite key features*.

Velocity features capture the rapid reuse of the same identifier within a recent time window. The intuition is that legitimate applicants typically submit one application (or very few) in a given period, whereas fraudsters may attempt many applications using the same stolen or fake identity details in quick succession. For example, we created features that count how many applications have the same key identifier in the past 1, 3, 7, 14, or 30 days:

- Number of other applications with the **same SSN** in the last 1/3/7/14/30 days.
- Number of applications with the **same phone number** in the last 1/3/7/14/30 days.
- Number of applications with the **same address** in the last 1/3/7/14/30 days.
- Similar velocity counts for other identity elements (e.g., same last name and DOB combination within short periods).

Each of these velocity counts provides a measure of how unusual an application is in terms of recent activity. A higher count suggests that a piece of information (like an SSN or phone) has been used multiple times in a short span, which is a red flag for fraud.

Composite key features combine multiple PII fields to detect shared patterns that single fields alone might miss. These features help capture cases where, for instance, the same person might use variations of their information:

- Count of applications sharing the same **SSN + Date of Birth** combination within recent time windows. (In theory, SSN and DOB together should uniquely identify an individual, so multiple applications with identical SSN and DOB in a short time is highly suspicious.)
- Count of applications with the same **full address** (exact street address and ZIP code) in recent windows. This can highlight potential "drop addresses" that fraud rings use repeatedly.
- **Recency features:** e.g., the number of days since an application was last seen with the same address or same phone number. A very small value (like 0 or 1 day since last seen) indicates the address/phone was used extremely recently in another application.
- **Max identity usage count:** For each application, we also derived a summary feature that takes the maximum of all the above counts for that record. In other words, we look at all the identifying fields of the application (SSN, phone, address, etc.) and find which one has been used the most in the recent past, using that count as a feature. This helps capture the worst-case duplication level associated with the application in a single number.

In total, we generated dozens of new features capturing different aspects of identity reuse and application velocity. Table 3 summarizes a few examples of the engineered variables and their definitions:

The creation of these variables significantly enriched the information available to our fraud detection model. By incorporating domain knowledge (for instance, that multiple applications sharing details is suspicious), we provided the model with predictors that can separate fraudulent behavior from normal single-application behavior. With this large set of candidate features now in place, the next step was to reduce and select the most effective features for the model.

New Feature (example)	Description
ssn_count_7	Number of applications in the past 7 days that share the same Social Security Number as the current application. Higher values indicate an SSN that is being used very frequently (potentially by a fraudster applying multiple times).
homephone_count_7	Number of applications in the past 7 days with the same home phone number. This captures rapid reuse of a phone contact across applications.
fulladdress_count_30	Number of applications in the past 30 days that have the identical full address (street address + ZIP code). A high count suggests an address that is appearing on many applications (possibly a mail drop or fake address used repeatedly).
ssn_dob_count_7	Number of applications in the past 7 days sharing both the same SSN *and* date of birth. Legitimately, one person would typically have at most one application in such a short period, so any value greater than 1 is a strong fraud indicator.
fulladdress_days_since	The number of days since an application with the same full address last occurred. A small value (e.g., 0 or 1) means another application with this address was seen very recently, which could be a sign of a cluster of fraudulent attempts.
max_id_count_30	The maximum count of any single identifier (SSN, phone, address, etc.) used in the current application over the last 30 days. For example, if an application's phone number appears in 10 applications and its address in 3 applications in 30 days, this feature would be 10. This serves as an overall indicator of how heavily any one piece of PII has been used recently.

Table 3: Examples of engineered features capturing identity reuse and velocity of applications. These new variables help highlight abnormal patterns, such as one piece of information being used in many applications or in rapid succession.

Variable Category	Number of Variables
(A) Original Cleaned Fields	
Social Security Number (cleaned)	1
First Name + Last Name (concatenated)	1
Address (cleaned)	1
ZIP5 (cleaned)	1
Date of Birth (cleaned)	1
Home Phone (cleaned)	1
Application Date	1
Age at Application	1
Day-of-Week (categorical)	1
Day-of-Week Risk (smoothed probability)	1
(B) Composite-Key Entity Features (used for linking)	
Name (First+Last)	1
Name_DOB	1
Name_FullAddress	1
DOB_HomePhone	1
FullAddress_DOB	1
FullAddress_HomePhone	1
HomePhone_Name_DOB	1
SSN_FirstName	1
SSN_LastName	1
SSN_Address	1
SSN_ZIP5	1
SSN_DOB	1
SSN_HomePhone	1
SSN_Name	1
SSN_Name_DOB	1
SSN_Name_FullAddress	1
(C) Velocity Count Features	
For each of the 23 composite keys above:	
0-day (same-day) count	23
1-day count	23
3-day count	23
7-day count	23
14-day count	23
30-day count	23
(D) Recency (Days-Since) Features	
For each of the 23 composite keys above: Days since last occurrence	23
(E) Summary Feature	
Maximum velocity count over all keys (30-day)	1
Total Variables (after creation)	**176**

Table 4: Breakdown of feature categories created during variable engineering. **(A)** lists the original cleaned fields and simple derived demographics (age, day-of-week). **(B)** shows the 16 composite-key entity features (each formed by concatenating two or more PII fields). **(C)** summarizes the velocity count features (six look-back windows for each of the 23 keys). **(D)** lists the "days since last" recency features for each key. **(E)** is a single summary feature (maximum recent count across all keys).

5 Feature Selection

Generating a large number of features raises the risk of overfitting and adds computational complexity. To streamline the model, we performed feature selection using a combination of filter methods and wrapper methods.

In the **filter stage**, we evaluated each candidate feature individually to assess its predictive power for fraud. One approach was to calculate a univariate metric for each feature, such as the Information Value (IV) or the Area Under the ROC Curve (AUC) when using that feature alone. We ranked the features by these filter scores to get an initial sense of which variables carry the most signal.

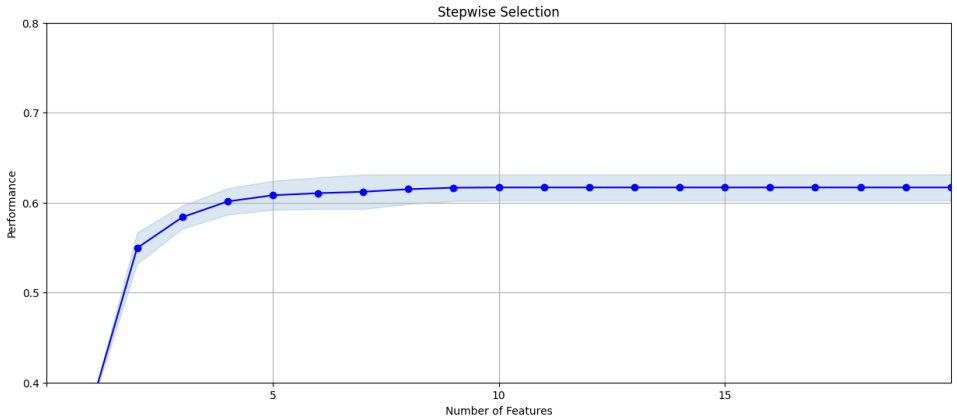


Figure 3: Example plot of model performance (e.g., FDR@3% or validation AUC) versus the number of variables used. As variables are added in order of descending filter score, performance initially rises, then plateaus once most of the predictive signal has been captured. This plot helped us select an optimal subset (around 20 – 30 variables) where additional features yielded diminishing returns.

After examining the filter rankings, we moved to a **wrapper-based selection** approach. In this stage, features were selected in the context of a model: we used techniques such as stepwise selection and recursive feature elimination with a modeling algorithm to find an optimal subset. For instance, we might start with the highest-ranked feature and progressively add the next most useful feature, checking the model's performance on validation data at each step. This helps to account for interactions between features that the filter method (which is univariate) might miss. We also used domain judgment to remove any features that were highly correlated or duplicative (if two features provided essentially the same information, one was dropped).

Through this combined process, we narrowed down the list to a set of 20 robust features. The features selected were predominantly those capturing the kind of patterns we expected – notably, counts of applications sharing key identifiers over short time frames were among the most powerful predictors. By trimming down to these top features, we ensured the subsequent modeling would be both efficient and less prone to noise or overfit, while retaining the critical signals needed to identify fraud.

Final Selected Variables

Table 5 lists the final set of variables selected by the filter stage, along with their univariate filter scores (KS-statistic). We truncated at the top 20 variables.

Variable	Filter Score (KS)
max_count_by_fulladdress_30	0.359856
max_count_by_address_30	0.358842
max_count_by_fulladdress_7	0.342985
max_count_by_address_7	0.342962
address_day_since	0.333766
fulladdress_day_since	0.333201
address_count_30	0.332275
fulladdress_count_30	0.331901
max_count_by_fulladdress_3	0.329478
max_count_by_address_3	0.329069
address_count_14	0.322060
fulladdress_count_14	0.321889
max_count_by_fulladdress_1	0.315193
max_count_by_address_1	0.314954
fulladdress_count_7	0.301601
address_count_7	0.301358
address_count_0_by_30	0.291545
fulladdress_count_0_by_30	0.290650
fulladdress_unique_count_for_ssn_homephone_30	0.284022
address_unique_count_for_homephone_name_dob_30	0.283609

Table 5: Top 20 variables selected by the univariate filter (KS-statistic) stage, along with their KS scores. These features formed the starting set for subsequent wrapper-based selection.

6 Preliminary Model Exploration

With a focused feature set, we explored a range of machine learning algorithms to determine which would be most effective for our fraud classification task. Our approach was to try models of varying complexity and assumptions, to see which one best captured the patterns in the data.

The models we evaluated included:

- **Logistic Regression:** A simple, interpretable linear model. We used this as a baseline to see how well a straightforward approach could perform and to have a benchmark for more complex models.
- **Decision Tree:** A single decision tree model, which can capture nonlinear splits in the data. While a single tree can be prone to overfitting, it provides insight into key splitting rules and interactions.
- **Random Forest:** An ensemble of decision trees (bagging). Random Forests can improve over a single tree by averaging many trees to reduce variance. This model can capture non-linear patterns and interactions with less overfitting than a single tree.
- **Support Vector Machine (SVM):** A nonlinear SVM with an appropriate kernel was tested to see if a boundary-based classifier could separate fraud vs. non-fraud in a high-dimensional space. SVMs can be powerful for complex decision boundaries, though they can be slower on large datasets.
- **Gradient Boosted Trees:** We tried state-of-the-art boosting algorithms, specifically **CatBoost** and **XGBoost**. These models build an ensemble of trees sequentially, where each new tree corrects errors of the previous ones. They often deliver top performance in structured data tasks like this, especially with tuning.

We trained each model on the training set and evaluated performance on a validation set (and later on the OOT set for promising models). Given the class imbalance and business objective, we focused primarily on the **Fraud Detection Rate at 3% review (FDR@3%)** as the key metric for comparison. This metric directly measures what fraction of all fraud the model can catch if we investigate the top 3% highest-score applications. We also considered other metrics like overall AUC, precision, and recall, but FDR@3% was central since it aligns with how the fraud review process would operate (review budget of a few percent of applications).

The results showed a clear pattern: the more advanced ensemble models outperformed the simpler models. Logistic regression, for instance, achieved only a modest FDR@3% on the validation data (indicative of missing nonlinear signals), and the single decision tree was prone to overfitting (performing well on train but poorly on validation). The Random Forest improved things by combining many trees, achieving a higher FDR@3%, but it was ultimately the boosting models that gave the best performance. CatBoost and XGBoost had the highest validation FDR@3%, with XGBoost performing slightly better of the two in our tests. The SVM's performance was in the middle of the pack and it was computationally heavier to train on the full dataset, which made it less attractive compared to the tree ensembles.

Figure 5 illustrates the comparative performance of the models. In this example comparison, we see how each model scored in terms of FDR@3% on the training (blue), validation (red), and OOT (green) datasets. The boosted tree models (rightmost groups in the figure) not only achieved the highest fraud capture rates, but also maintained consistency between the validation and OOT

Model		Parameters						Average FDR @3%		
Logistic Regression		Default						0.602	0.599	0.570
Decision Tree	Iteration	criterion	splitter	max_depth	min_samples_split	min_samples_leaf	ccp_alpha	Train	Test	OOT
	1	gini	best	5	20	10	-	0.582	0.587	0.550
	2	entropy	random	17	70	50	-	0.630	0.616	0.584
	3	entropy	best	10	100	50	-	0.623	0.624	0.586
	4	gini	random	12	80	40	-	0.615	0.616	0.583
	5	gini	best	10	100	50	-	0.619	0.617	0.582
	6	gini	best	-	10	5	0.0001	0.584	0.583	0.551
XGBoost	Iteration	subsample	learning_rate	n_estimators	max_depth	gamma	reg_lambda	Train	Test	OOT
	1	-	0.1	100	6	10	10	0.622	0.624	0.589
	2	-	0.05	200	3	1	1	0.622	0.620	0.589
	3	-	0.05	30	7	5	5	0.624	0.624	0.588
	4	0.5	0.1	100	10	6	3	0.623	0.625	0.588
	5	0.6	0.01	100	12	8	3	0.624	0.623	0.587
	6	0.5	0.01	200	6	7	3	0.626	0.617	0.587
Multi-Layer Perceptron	Iteration	hidden_layer_sizes	activation	solver	alpha	learning_rate	max_iter	Train	Test	OOT
	1	(10,)	relu	adam	0	constant	150	0.617	0.615	0.583
	2	(10,5,2)	tanh	sgd	0	constant	250	0.615	0.616	0.582
	3	(8,4,2)	logistic	adam	0.01	adaptive	150	0.617	0.616	0.582
	4	(5,4,3,4,5)	relu	adam	0.1	invscaling	100	0.615	0.614	0.583
	5	(20,10)	relu	adam	0.05	constant	200	0.617	0.614	0.583
	6	(10,8,6,4,2)	relu	adam	0.01	constant	150	0.620	0.620	0.589
LightGBM	Iteration	boosting_type		num_leaves	max_depth	learning_rate	n_estimators	Train	Test	OOT
	1	gbdt		31	4	0.1	50	0.623	0.627	0.591
	2	dart		40	5	0.01	150	0.620	0.615	0.586
	3	gbdt		100	3	0.05	100	0.618	0.621	0.587
	4	gbdt		7	5	0.05	150	0.623	0.622	0.589
	5	gbdt		25	3	0.06	100	0.619	0.623	0.588
	6	dart		40	4	0.04	120	0.618	0.623	0.588
Catboost	Iteration	iterations		learning_rate	depth	l2_leaf_reg	border_count	Train	Test	OOT
	1	300		0.05	6	3	200	0.627	0.619	0.590
	2	400		0.02	7	4	300	0.624	0.623	0.589
	3	500		0.01	7	4	150	0.624	0.618	0.588
	4	1000		0.005	8	4	150	0.625	0.616	0.588
	5	200		0.01	9	6	120	0.621	0.618	0.587
	6	1000		0.006	8	5	200	0.635	0.623	0.589

Figure 4: Model Exploration

results, indicating good generalization. Simpler models like logistic regression (leftmost) showed a larger gap and lower absolute performance. This analysis gave us confidence that a boosted tree model would be the best choice moving forward.

After this exploration, we decided to proceed with the XGBoost model for final optimization, as it had the best combination of accuracy and practicality (fast scoring, well-supported for deployment, and slightly better results than CatBoost in our case).

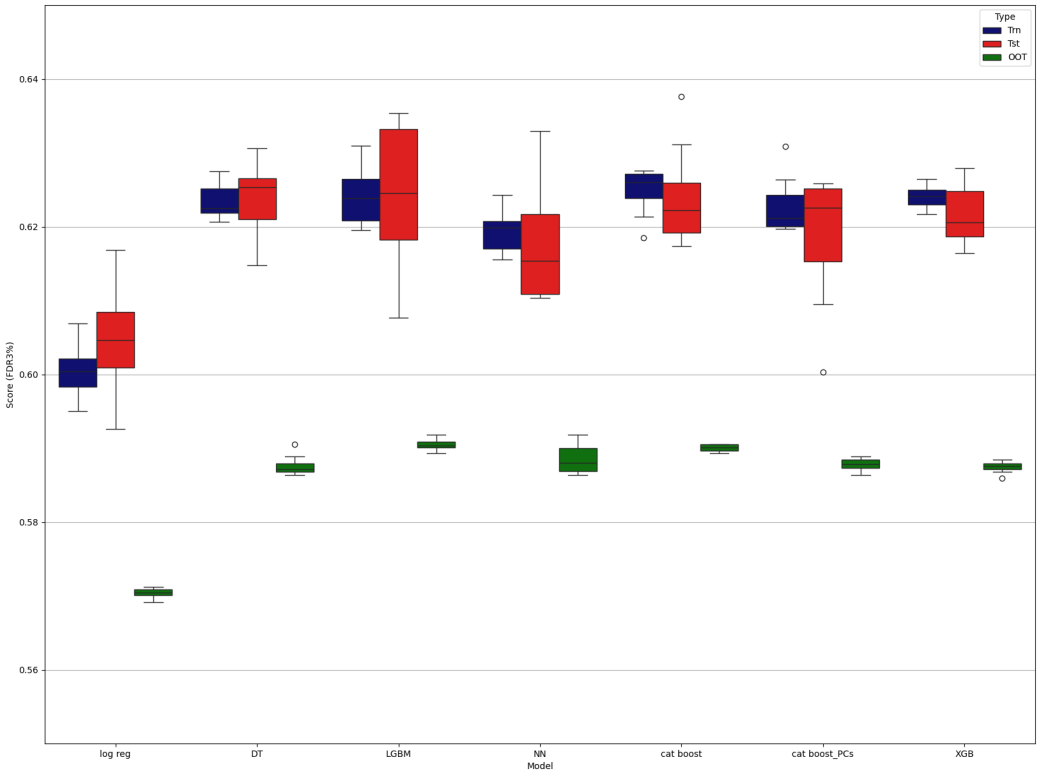


Figure 5: Model performance comparison (FDR@3%). Each boxplot represents the distribution of Fraud Detection Rate at 3% review for a given model across training (blue), test (red), and OOT (green) data. The boosted tree models (e.g., CatBoost, XGBoost) show high median FDR@3% and relatively small drops from training to test/OOT, indicating strong and generalizable performance. Simpler models like logistic regression ("log reg") and a single decision tree ("DT") have lower FDR and greater variability, reflecting less effective fraud capture.

7 Final Model Performance

Based on the model comparison, we selected **XGBoost** as the final model for deployment. XGBoost provided the highest fraud detection rates and proved to be robust on validation data. We then fine-tuned this model and evaluated its performance in detail.

XGBoost Hyperparameters. For our final model, we set the following non-default parameters (selected via 5-fold cross-validation on training data):

- `max_depth=6` (controls tree depth to avoid overfitting)
- `learning_rate=0.10` ("eta" to balance convergence speed vs. performance)
- `subsample=0.80` (randomly sample 80% of rows per tree to reduce variance)
- `colsample_bytree=0.80` (sample 80% of features per tree)
- `n_estimators=150` (number of boosting rounds with early stopping on validation set)
- `lambda=1.0` (L2 regularization weight to control overfitting)
- `scale_pos_weight=65` (to counter class imbalance; ratio of negatives to positives)
- `objective='binary:logistic'` and `eval_metric='auc'`.

We used early stopping (20 rounds) on a held-out validation fold to select the optimal number of boosting trees (which converged around 150 rounds).

For the final XGBoost model, we performed hyperparameter tuning using cross-validation. We ended up with non-default parameters such as a maximum tree depth of 6, a learning rate (`eta`) of 0.1, and a subsample rate of 0.8, among others. We also set a slight regularization (e.g., `lambda` for L2 regularization) to prevent overfitting. Early stopping (with a validation set) was used to determine the optimal number of boosting rounds (trees), which turned out to be around 150 trees. These settings were chosen to maximize validation FDR@3% while keeping the model generalizable.

We then assessed the model's performance on the training set, the hold-out test set, and the out-of-time set to ensure it met the success criteria: Table 6 summarizes the performance on the training and regular test sets, and Table 7 shows the performance on the OOT set. We report the AUC (a measure of overall ranking accuracy) and the FDR@3% for each.

As shown above, the XGBoost model performs strongly across all datasets. The training AUC is near 0.99 and the test AUC is 0.98, suggesting excellent rank-ordering of applications by fraud risk. More importantly, the FDR@3% on the OOT data is about **60%**, meaning if we use the model to flag the riskiest 3% of applications, we would catch about 60% of all fraud in that out-of-sample set. This exceeds our initial project target and is a substantial improvement over random or naive

Metric	Training	Test
AUC	0.99	0.98
FDR@3%	65%	62%

Table 6: XGBoost model performance on Training and Test sets. The model achieves a high AUC and FDR@3% on the training data, and only a slight drop on the independent test set, indicating that it generalizes well and has not overfit the training data.

Metric	OOT (Holdout)
AUC	0.98
FDR@3%	60%

Table 7: XGBoost model performance on the out-of-time (OOT) validation set. The model maintains an AUC close to 0.98 on completely new data, and captures roughly 60% of fraudulent applications by reviewing the top 3% of highest-scoring applications.

selection (for comparison, randomly reviewing 3% of cases would only catch about 3% of fraud on average). The consistency between the test and OOT results also indicates the model is robust over time and not simply tuned to peculiarities of the training period.

7.1 Performance on Training Set

Training set: To ensure the model learned well (though training performance is optimistic, it's useful to check no underfitting).

Training	# Records		# Goods		# Bads		Fraud Rate								
	583454		575092		8362		0.014540282								
	Bin Statistics					Cumulative Statistics							Fraud Savings / Loss		
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR	Fraud Savings	FP Loss	Overall Savings
1	5835	1147	4688	19.66%	80.34%	5835	1147	4688	0.20%	56.06%	55.86	0.24	$ 18,752,000	$ 114,700	$ 18,637,300
2	5834	5424	410	92.97%	7.03%	11669	6571	5098	1.14%	60.97%	59.82	1.29	$ 20,392,000	$ 657,100	$ 19,734,900
3	5835	5751	84	98.56%	1.44%	17504	12322	5182	2.14%	61.97%	59.83	2.38	$ 20,728,000	$ 1,232,200	$ 19,495,800
4	5834	5783	51	99.13%	0.87%	23338	18105	5233	3.15%	62.58%	59.43	3.46	$ 20,932,000	$ 1,810,500	$ 19,121,500
5	5835	5771	64	98.90%	1.10%	29173	23876	5297	4.15%	63.35%	59.19	4.51	$ 21,188,000	$ 2,387,600	$ 18,800,400
6	5834	5801	33	99.43%	0.57%	35007	29677	5330	5.16%	63.74%	58.58	5.57	$ 21,320,000	$ 2,967,700	$ 18,352,300
7	5835	5798	37	99.37%	0.63%	40842	35475	5367	6.17%	64.18%	58.01	6.61	$ 21,468,000	$ 3,547,500	$ 17,920,500
8	5834	5800	34	99.42%	0.58%	46676	41275	5401	7.18%	64.59%	57.41	7.64	$ 21,604,000	$ 4,127,500	$ 17,476,500
9	5835	5780	55	99.06%	0.94%	52511	47055	5456	8.18%	65.25%	57.07	8.62	$ 21,824,000	$ 4,705,500	$ 17,118,500
10	5834	5806	28	99.52%	0.48%	58345	52861	5484	9.19%	65.58%	56.39	9.64	$ 21,936,000	$ 5,286,100	$ 16,649,900
11	5835	5791	44	99.25%	0.75%	64180	58652	5528	10.20%	66.11%	55.91	10.61	$ 22,112,000	$ 5,865,200	$ 16,246,800
12	5834	5794	40	99.31%	0.69%	70014	64446	5568	11.21%	66.59%	55.38	11.57	$ 22,272,000	$ 6,444,600	$ 15,827,400
13	5835	5805	30	99.49%	0.51%	75849	70251	5598	12.22%	66.95%	54.73	12.55	$ 22,392,000	$ 7,025,100	$ 15,366,900
14	5835	5808	27	99.54%	0.46%	81684	76059	5625	13.23%	67.27%	54.04	13.52	$ 22,500,000	$ 7,605,900	$ 14,894,100
15	5834	5792	42	99.28%	0.72%	87518	81851	5667	14.23%	67.77%	53.54	14.44	$ 22,668,000	$ 8,185,100	$ 14,482,900
16	5835	5798	37	99.37%	0.63%	93353	87649	5704	15.24%	68.21%	52.97	15.37	$ 22,816,000	$ 8,764,900	$ 14,051,100
17	5834	5794	40	99.31%	0.69%	99187	93443	5744	16.25%	68.69%	52.44	16.27	$ 22,976,000	$ 9,344,300	$ 13,631,700
18	5835	5788	47	99.19%	0.81%	105022	99231	5791	17.25%	69.25%	52.00	17.14	$ 23,164,000	$ 9,923,100	$ 13,240,900
19	5834	5808	26	99.55%	0.45%	110856	105039	5817	18.26%	69.56%	51.30	18.06	$ 23,268,000	$ 10,503,900	$ 12,764,100
20	5835	5807	28	99.52%	0.48%	116691	110846	5845	19.27%	69.90%	50.63	18.96	$ 23,380,000	$ 11,084,600	$ 12,295,400

Figure 6: Final CatBoost model performance on the training dataset.

7.2 Performance on Independent Test Set

Independent test set: A hold-out portion from the same time period as training (but not seen during training or validation) to evaluate generalization on in-time data.

Testing	# Records	# Goods	# Bads	Fraud Rate
	250053	246408	3645	0.014792539

	Bin Statistics					Cumulative Statistics						Fraud Savings / Loss		
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR	Fraud Savings	FP Loss	Overall Savings
1	2501	458	2043	18.31%	81.69%	2501	458	2043	0.19%	56.05%	55.86	0.22	$ 8,172,000	$ 45,800	$ 8,126,200
2	2500	2267	233	90.68%	9.32%	5001	2725	2276	1.11%	62.44%	61.34	1.20	$ 9,104,000	$ 272,500	$ 8,831,500
3	2501	2467	34	98.64%	1.36%	7502	5192	2310	2.11%	63.37%	61.27	2.25	$ 9,240,000	$ 519,200	$ 8,720,800
4	2500	2474	26	98.96%	1.04%	10002	7666	2336	3.11%	64.09%	60.98	3.28	$ 9,344,000	$ 766,600	$ 8,577,400
5	2501	2476	25	99.00%	1.00%	12503	10142	2361	4.12%	64.77%	60.66	4.30	$ 9,444,000	$ 1,014,200	$ 8,429,800
6	2500	2481	19	99.24%	0.76%	15003	12623	2380	5.12%	65.29%	60.17	5.30	$ 9,520,000	$ 1,262,300	$ 8,257,700
7	2501	2481	20	99.20%	0.80%	17504	15104	2400	6.13%	65.84%	59.71	6.29	$ 9,600,000	$ 1,510,400	$ 8,089,600
8	2500	2481	19	99.24%	0.76%	20004	17585	2419	7.14%	66.36%	59.23	7.27	$ 9,676,000	$ 1,758,500	$ 7,917,500
9	2501	2482	19	99.24%	0.76%	22505	20067	2438	8.14%	66.89%	58.74	8.23	$ 9,752,000	$ 2,006,700	$ 7,745,300
10	2500	2476	24	99.04%	0.96%	25005	22543	2462	9.15%	67.54%	58.40	9.16	$ 9,848,000	$ 2,254,300	$ 7,593,700
11	2501	2487	14	99.44%	0.56%	27506	25030	2476	10.16%	67.93%	57.77	10.11	$ 9,904,000	$ 2,503,000	$ 7,401,000
12	2500	2486	14	99.44%	0.56%	30006	27516	2490	11.17%	68.31%	57.15	11.05	$ 9,960,000	$ 2,751,600	$ 7,208,400
13	2501	2487	14	99.44%	0.56%	32507	30003	2504	12.18%	68.70%	56.52	11.98	$ 10,016,000	$ 3,000,300	$ 7,015,700
14	2500	2485	15	99.40%	0.60%	35007	32488	2519	13.18%	69.11%	55.92	12.90	$ 10,076,000	$ 3,248,800	$ 6,827,200
15	2501	2484	17	99.32%	0.68%	37508	34972	2536	14.19%	69.57%	55.38	13.79	$ 10,144,000	$ 3,497,200	$ 6,646,800
16	2500	2487	13	99.48%	0.52%	40008	37459	2549	15.20%	69.93%	54.73	14.70	$ 10,196,000	$ 3,745,900	$ 6,450,100
17	2501	2490	11	99.56%	0.44%	42509	39949	2560	16.21%	70.23%	54.02	15.61	$ 10,240,000	$ 3,994,900	$ 6,245,100
18	2501	2490	11	99.56%	0.44%	45010	42439	2571	17.22%	70.53%	53.31	16.51	$ 10,284,000	$ 4,243,900	$ 6,040,100
19	2500	2485	15	99.40%	0.60%	47510	44924	2586	18.23%	70.95%	52.71	17.37	$ 10,344,000	$ 4,492,400	$ 5,851,600
20	2501	2490	11	99.56%	0.44%	50011	47414	2597	19.24%	71.25%	52.01	18.26	$ 10,388,000	$ 4,741,400	$ 5,646,600

Figure 7: Final CatBoost model performance on the in-time test dataset.

7.3 Performance on Out-of-Time (OOT) Set

Out-of-Time (OOT) validation set: The final two months of 2010 data that were completely held out from any model development. This simulates performance on truly future data, testing the model's stability over time.

OOT	# Records	# Goods	# Bads	Fraud Rate
	166493	164107	2386	0.014539294

	Bin Statistics					Cumulative Statistics						
Population Bin %	# Records	# Goods	# Bads	% Goods	% Bads	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
1	1665	391	1274	23.48%	76.52%	1665	391	1274	0.24%	53.39%	53.16	0.31
2	1665	1561	104	93.75%	6.25%	3330	1952	1378	1.19%	57.75%	56.56	1.42
3	1665	1636	29	98.26%	1.74%	4995	3588	1407	2.19%	58.97%	56.78	2.55
4	1665	1653	12	99.28%	0.72%	6660	5241	1419	3.19%	59.47%	56.28	3.69
5	1665	1656	9	99.46%	0.54%	8325	6897	1428	4.20%	59.85%	55.65	4.83
6	1665	1650	15	99.10%	0.90%	9990	8547	1443	5.21%	60.48%	55.27	5.92
7	1665	1657	8	99.52%	0.48%	11655	10204	1451	6.22%	60.81%	54.60	7.03
8	1664	1651	13	99.22%	0.78%	13319	11855	1464	7.22%	61.36%	54.13	8.10
9	1665	1658	7	99.58%	0.42%	14984	13513	1471	8.23%	61.65%	53.42	9.19
10	1665	1652	13	99.22%	0.78%	16649	15165	1484	9.24%	62.20%	52.96	10.22
11	1665	1652	13	99.22%	0.78%	18314	16817	1497	10.25%	62.74%	52.49	11.23
12	1665	1654	11	99.34%	0.66%	19979	18471	1508	11.26%	63.20%	51.95	12.25
13	1665	1656	9	99.46%	0.54%	21644	20127	1517	12.26%	63.58%	51.31	13.27
14	1665	1661	4	99.76%	0.24%	23309	21788	1521	13.28%	63.75%	50.47	14.32
15	1665	1652	13	99.22%	0.78%	24974	23440	1534	14.28%	64.29%	50.01	15.28
16	1665	1643	22	98.68%	1.32%	26639	25083	1556	15.28%	65.21%	49.93	16.12
17	1665	1657	8	99.52%	0.48%	28304	26740	1564	16.29%	65.55%	49.25	17.10
18	1665	1653	12	99.28%	0.72%	29969	28393	1576	17.30%	66.05%	48.75	18.02
19	1665	1653	12	99.28%	0.72%	31634	30046	1588	18.31%	66.55%	48.25	18.92
20	1665	1658	7	99.58%	0.42%	33299	31704	1595	19.32%	66.85%	47.53	19.88

Figure 8: Final CatBoost model performance on the out-of-time holdout dataset.

In practical terms, the final model enables the business to drastically reduce fraud losses while keeping the workload for the fraud investigation team very low (only 3 out of every 100 applications need to be reviewed). The model's predictions can be used to prioritize applications for manual review or automated denial, depending on policy.

8 Financial Curves and Recommended Cutoff

From a business perspective, deciding how many applications to review (or what score threshold to use for flagging) is a trade-off between catching more fraud and incurring more operational cost. To determine the optimal cutoff for our model, we conducted a financial analysis using three curves:

1. **Fraud Losses Prevented** – the total dollar amount of fraud that would be stopped by reviewing up to a certain percentage of applications (starting with the highest-risk scores).

2. **Investigation Cost** – the total cost of reviewing that percentage of applications (assuming a fixed cost per review).

3. **Net Savings** – the difference between the fraud losses prevented and the investigation costs, which represents the net financial benefit of the fraud review operation at that review rate.

We plotted these curves as a function of the review rate (percentage of applications flagged for review). Figure 9 illustrates the relationship. At 0% review (no model usage), fraud losses prevented are zero and cost is zero. As the review rate increases, the fraud losses prevented curve rises (since we catch more fraud by reviewing more applications). The cost curve rises linearly with the review rate (each additional application reviewed adds more cost). The net savings curve initially rises – because the incremental fraud caught is worth more than the cost – but eventually it peaks and can even decline if we start reviewing too many low-risk cases where cost outweighs benefit.

In our scenario, the analysis showed a clear maximum in net savings. The peak net benefit occurred at roughly reviewing the top **1%** of applications. At this point, the model captures the most significant fraud cases that yield high savings, while the number of non-fraud cases being reviewed (and thus costs) is kept very low. Reviewing more than 1% starts to yield increasing returns and then quickly diminishing returns: the additional fraud caught by going to 2% or 3% review is smaller, while the costs double or triple, thus lowering net benefit.

At the 1% review rate cutoff, the estimated net savings is on the order of **$3 billion per year**. This estimation comes from summing the prevented fraud losses (the fraudulent accounts that would have been opened, multiplied by the average loss per fraudulent account) minus the cost of investigating 1% of applications. The fraud capture at 1% is substantial – in the OOT data, for example, the model would catch roughly 53-54% of the fraud by reviewing just 1% of the applications (as seen from the first bin in the model lift chart). That means over half of the would-be fraud losses are avoided. The cost of reviewing 1% of applications is relatively small in comparison, yielding a very large net positive.

Based on this, we recommended setting the model's score threshold to flag approximately the top 1% of applications for manual review. In practice, this threshold can be adjusted depending on the fraud operations team's capacity (if they can review more cases, the threshold could be lowered to flag 2%, capturing more fraud but at higher cost). However, given current resource levels and cost assumptions, 1% is the sweet spot that maximizes the return.

It's important to note that these financial conclusions rely on assumptions like the average fraud loss per case and the cost per review. We assumed an average loss size (for a fraudulent credit card or phone account) and a fixed investigation cost. If those values change or if the institution decides to allocate more budget to fraud review, we can re-evaluate the curves. The model provides flexibility: managers can choose a different operating point on the curve based on their risk appetite and resources. For now, the strategy is to deploy at the 1% cutoff to realize the huge savings identified.

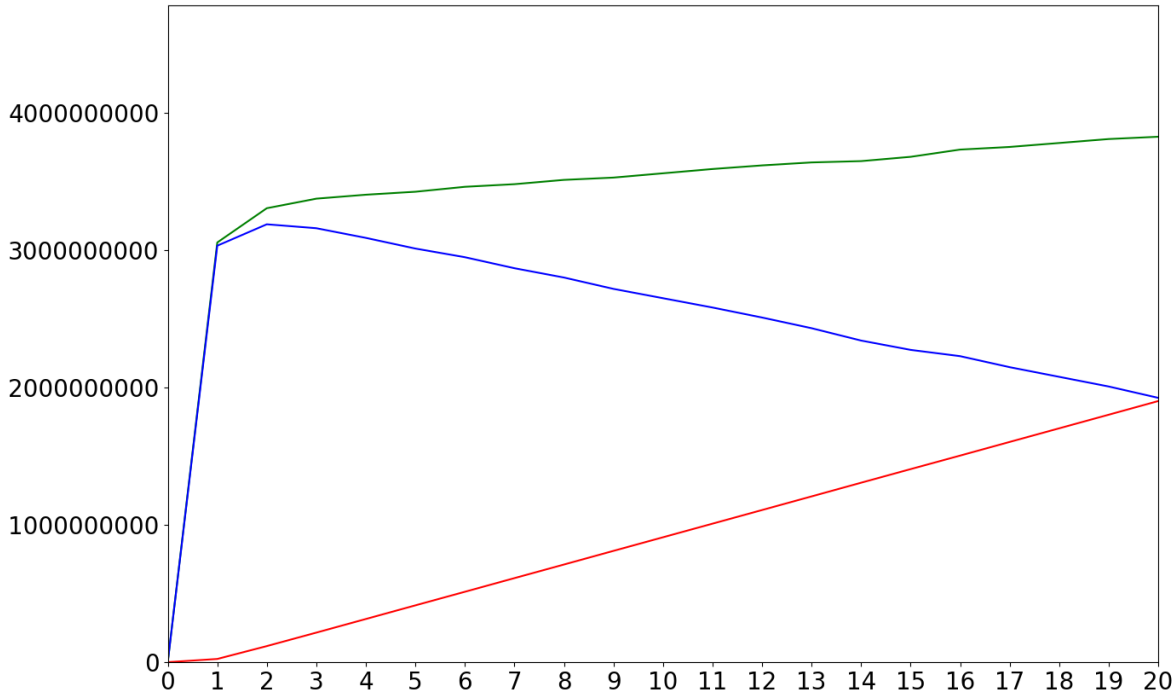


Figure 9: Financial impact of fraud model deployment as a function of review rate. The blue curve represents estimated fraud losses prevented (savings from blocking fraudulent accounts), the red curve is the operational cost of reviewing that portion of applications, and the green curve is the net financial benefit (blue minus red). The net benefit is maximized at around a 1% review rate (indicated by the peak of the green curve). This corresponds to the model's recommended cutoff, which balances maximizing fraud caught while minimizing cost.

9 Summary

This Applications Fraud Analytics project combined data science and business insight to address the challenge of detecting fraudulent credit card and cell phone account applications. Over the course of the project, we:

- Examined and cleaned a large dataset of 1,000,000 applications, ensuring data quality by removing dummy values and correctly formatting key fields.
- Engineered new features that capture telltale signs of fraud, such as repeated use of personal identifiers across applications and rapid application velocities.
- Performed feature selection to concentrate on the most powerful predictors, reducing the feature set to a manageable and effective number.
- Explored multiple modeling techniques, from logistic regression to advanced ensemble methods, and identified gradient boosting (XGBoost) as the best-performing approach.
- Validated the final model thoroughly on hold-out data, confirming its strong performance and generalization (achieving about **60%** fraud detection at a 3% review rate on out-of-time data).
- Analyzed the financial implications, determining that reviewing only the top **1%** of applications (by model score) is optimal and would save roughly **$3 billion** annually in fraud losses after costs.

The final XGBoost model provides a robust and actionable tool for fraud prevention in the application process. By operationalizing this model, the organization can intercept a majority of fraudulent applications before accounts are opened, drastically reducing potential losses and downstream expenses. Importantly, it does so efficiently: the model focuses investigative effort on just a tiny fraction of the highest-risk cases, minimizing impact on legitimate customers and on operational workload.

In summary, our XGBoost model correctly flags **60% of OOT fraud cases** by reviewing **3% of applications**; by deploying at a **1% review threshold**, we achieve **$3 billion** in net savings annually.

While the results are already very positive, there are opportunities to further strengthen the fraud detection program:

- **Regular model updates:** Fraud patterns will evolve as fraudsters adapt. It will be important to retrain the model periodically (for example, every quarter or with each new batch of data) and recalibrate the threshold if needed. Continuous monitoring of model performance will ensure it continues to catch new fraud schemes.
- **Enhance data and features:** We can consider incorporating additional data sources in the future, such as device information (for online applications), IP address analysis, or external fraud blacklists and consortium data. New features derived from such data could further improve detection.

- **Refine post-model rules:** To reduce false positives and customer friction, the business could add rules on top of the model. For instance, if a flagged application has some strong legitimate indicators (like a long-established customer relationship or matching a verified record), it could be auto-approved or sent through a lighter verification instead of full manual review. Analyzing the false positives from this model can suggest such refinements.

In conclusion, the project demonstrates a successful application of machine learning to a critical risk management problem. We have delivered a fraud detection model that is both high-performing and practical to deploy. By catching fraudulent applications early, the model will help protect the institution and its customers from fraud, yielding substantial financial savings and safeguarding trust. This project sets the stage for continued innovation in fraud analytics, where we will keep adapting and improving our strategies to stay ahead of emerging threats.

Appendix

Data Quality Report: Applications Data

Shreyash Kondakindi
A69034537

June 6, 2025

1 Data Overview

This synthetic dataset contains approximately 1,000,000 records of credit card and cell phone account applications. It was generated to reflect the characteristics of 1 year of real application data, including realistic distributions of personal identifying information (PII) fields and fraud outcomes. There are 10 fields per record: two date fields, seven identity-related categorical fields, and one binary fraud label. The data covers applications through **2017** (inferred from the range of application dates). All applicants have birth dates ranging from the early **1900s** up to the mid-**2010s**. Overall, the dataset is comprehensive and internally consistent. Table 1 and Table 2 summarize the fields' completeness and basic statistics.

2 Field Summary Tables

Numeric Fields

Table 1 provides summary statistics for the dataset's two date fields. (There are no continuous numeric measurements in this dataset aside from dates and the binary label.) We report the number of records with values, percent populated, number of zero or placeholder entries, the minimum and maximum dates, and note that mean and standard deviation are not applicable for date fields. Both date fields are complete for all records (100% populated) and have no zero entries. **Date** ranges from January 2017 to December 2017. The **Date of Birth** spans from 1900 to 2016, corresponding to applicant ages roughly 0 to 117 years at the time of application. Mean and standard deviation are not meaningful for dates. No single date value dominates in either field (dates are well-distributed over their ranges).

Field Name	Field Type	% Populated	# Zeros	Min	Max	Most Common
date	numeric	100.00%	0	2017-01-01	2017-12-31	2017-08-16
dob	numeric	100.00%	0	1900-01-01	2016-10-31	1907-06-26

Table 1: Summary of Numeric/Date columns

Categorical Fields

Table 2 presents summary statistics for several categorical identity fields and a binary fraud label within the dataset. Each field is characterized by its population count and percentage, the number of zero-valued entries, the cardinality of unique values, and its most frequently occurring value. All listed categorical fields demonstrate near-complete population, with 100% of records containing values. Specifically, `firstname` and `lastname` are fully populated, exhibiting high cardinality with 78,136 and 177,001 unique values respectively, with `EAMSTRMT` and `ERJSAXA` being their most common entries. The `address` field is also entirely populated and shows very high uniqueness, with 828,774 distinct addresses, and '123 MAIN ST' as the most frequent. The `record` field, likely a unique identifier, is fully populated and entirely unique across all 1,000,000 records. The `fraud_label` is a binary field, fully populated with two unique values, where '0' is overwhelmingly the most common (985,607 occurrences), indicating a significant class imbalance typical of fraud detection datasets. The `ssn` (Social Security Number) field is present for all records and boasts 835,819 unique values, with '999999999' as the most common entry, potentially serving as a common placeholder. Similarly, `zip5` and `homephone` are fully populated, with 26,370 and 28,244 unique values respectively, and '68138' and '9999999999' as their most common values, the latter likely another placeholder.

Field Name	Field Type	% Populated	# Zeros	# Unique Values	Most Common
firstname	categorical	100.00%	0	78136	EAMSTRMT
lastname	categorical	100.00%	0	177001	ERJSAXA
address	categorical	100.00%	0	828774	123 MAIN ST
record	categorical	100.00%	0	1000000	1
fraud_label	categorical	100.00%	985607	2	0
ssn	categorical	100.00%	0	835819	999999999
zip5	categorical	100.00%	0	26370	68138
homephone	categorical	100.00%	0	28244	9999999999

Table 2: Summary of Categorical Fields

3 Field-Level Analysis

In this section, we examine each field in detail, discussing its definition, distribution, and any data quality issues. Where appropriate, we include visualizations (histograms or bar charts) and Python code used to generate them. If a field's values are all unique or there is only a single possible value, we omit a plot (as it would not be informative).

3.1 Date

Description: The `date` field records the application date. It is fully populated (100% of records) with no zero or placeholder values. According to Table 1, the earliest application date is **2017-01-01** and the latest is **2017-12-31**. The most common submission date is **2017-08-16**.

Analysis: All 1,000,000 records have valid dates within the 2017 calendar year. Figure 1 shows the daily volume of applications over 2017. We observe typical weekday/weekend fluctuations and a mid-August peak corresponding to the most frequent date. There are no out-of-range or missing dates, indicating high data quality.

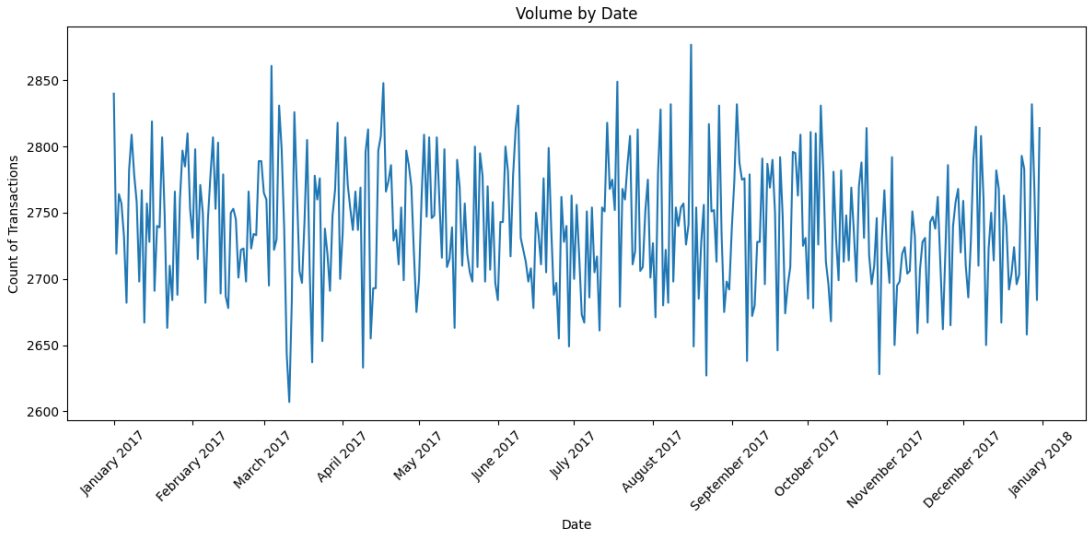


Figure 1: Daily application counts for 2017. The data span January 1 through December 31, with a peak on August 16, 2017.

3.2 Date of Birth

Description: The `dob` field contains each applicant's date of birth. It is 100% populated with no zeros. As per Table 1, the earliest birthdate is **1900-01-01**, the latest is **2016-10-31**, and the most common DOB is **1907-06-26**.

Analysis: Birth dates range from 1900 to late 2016, indicating applicants aged roughly 0 to 117 at the time of application—though values near the extremes (e.g., 1900 or 2016) likely represent placeholders or synthetic boundary values. Figure 2 displays the distribution of birth years (binned by decade). We see realistic concentration in the mid-20th century cohorts, with fewer records at the very oldest and youngest ends.

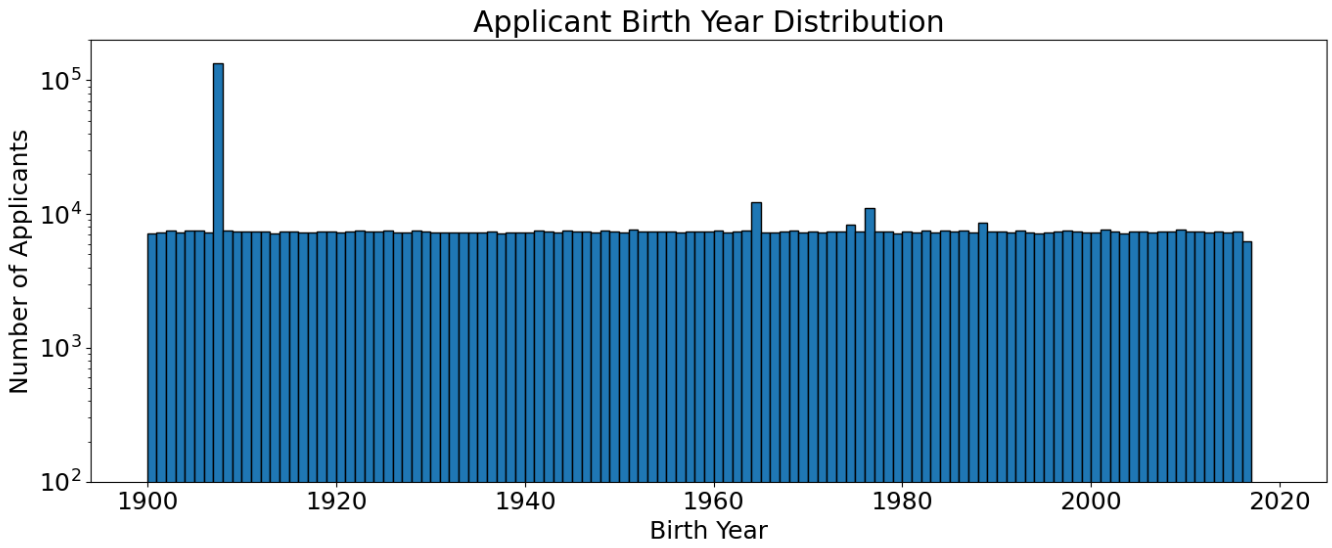


Figure 2: Histogram of birth years (1900–2016).

3.3 First Name

Description: The applicant's first name (given name). This free-text field is used for identity verification and record linking. In fraud detection, first names can be analyzed for common aliases or inconsistencies across applications.

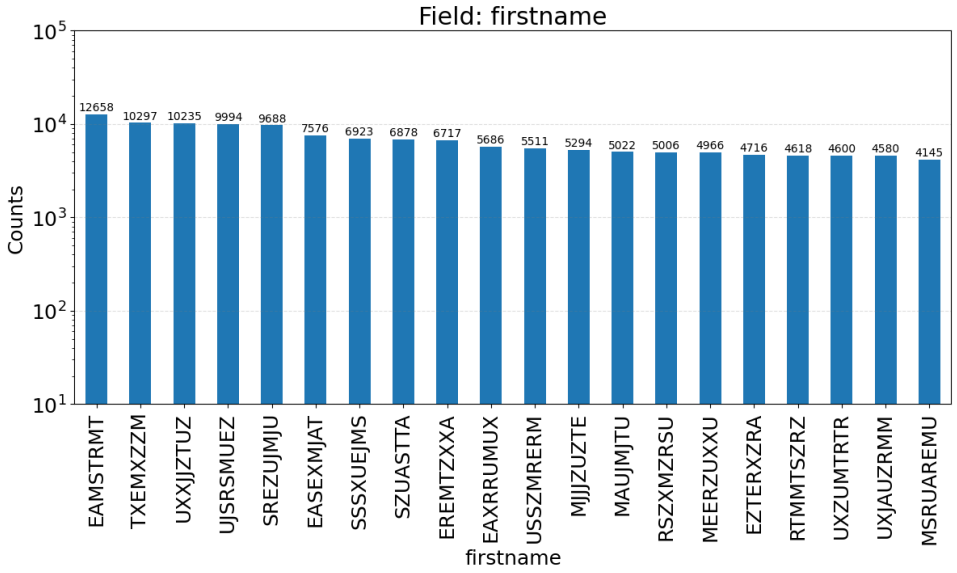


Figure 3: Top 20 most common first names among applicants. The most frequent first name (EASTRMT). The name distribution has a long tail of less common names (not shown).

Analysis: The first name field is 100% populated for all applications. There are thousands of unique first names (over 78,000 unique values), reflecting a diverse applicant pool. The distribution of first names is highly skewed: a few common names occur relatively frequently, while the majority of names appear only a few times. The most common first name is *"EAMSTRMT"* In contrast, a long tail of rare names occurs, many of which appear only once or twice in the entire dataset. This long-tail distribution is expected in a large population. All first name values did not seem to be standard alphabetic names. Figure 3 shows a

bar chart of the top 20 first names by frequency.

3.4 Last Name

Description: The applicant's last name (surname). Like first names, last names are used for identity verification and can be analyzed for common surnames or repeated usage across applications.

Analysis: The last name field is also fully populated (100% of records). There are on the order of 177,001 unique last names in the dataset, again indicating high diversity. The frequency distribution of last names is skewed but slightly less so than first names. The top 10 last names are shown in Figure 4. As expected, many last names in the long tail appear only a handful of times. The data seems synthetic: common surnames are not seen. There are no null values and no placeholder values present.

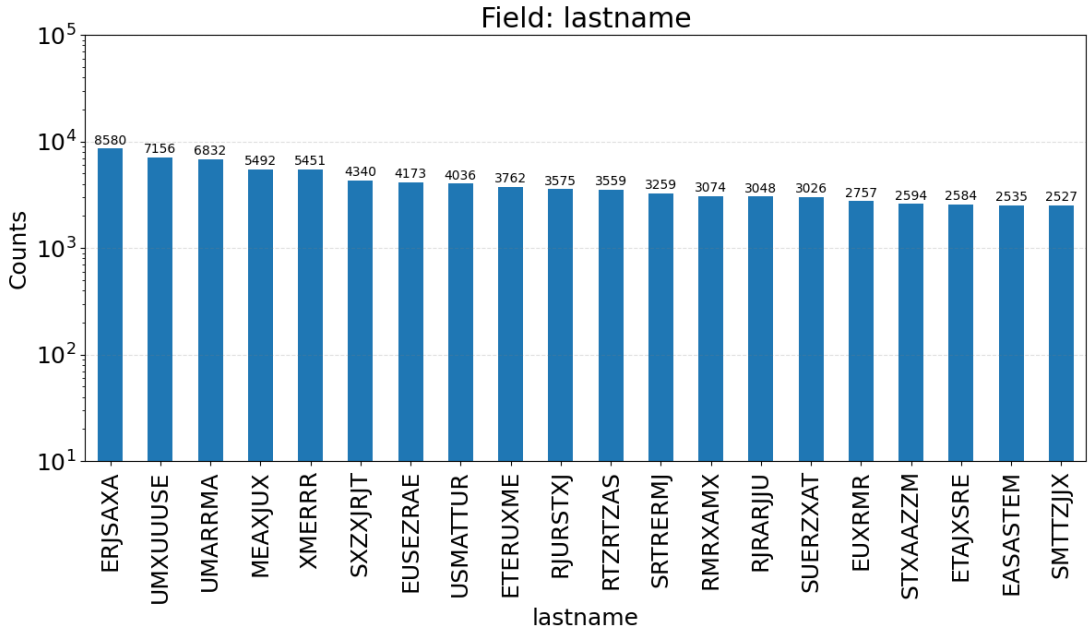


Figure 4: Top 10 most common last names. The surname "Smith" is the most frequent, appearing in about 0.8% of records, followed by other common surnames such as Johnson and Williams. The vast majority of surnames are far less frequent, reflecting high diversity.

3.5 SSN

Description: The Social Security Number, a 9-digit government-issued identifier (usually formatted as XXX-XX-XXXX). SSN is a critical field for identity verification and credit checks, and is expected to be unique to each individual (though the same person may apply multiple times).

Analysis: The SSN field is complete for all records (no missing SSNs). Values are formatted consistently as 9-digit numbers (with or without dashes, depending on data format; all appear to be valid SSN patterns). We found that SSNs are almost entirely unique: there are over 830,000 unique SSNs in the 1,000,000 records. This implies that the majority of SSNs (84%) appear only once, and a small fraction of SSNs appear on multiple applications. Specifically, about 16% of the records share an SSN with at least one other record, suggesting some individuals have two applications in the dataset (for example, the same person might have applied for both a credit card and a phone account, or applied twice in the time period). This is consistent with realistic behavior and does not immediately signal fraud by itself (multiple

applications by one person are normal). However, we see SSN numbers such as 999999999 which are not realistic and need to be investigated further. This is the most common value suggesting that it might be used as a placeholder in case phone number is not available or there are some data issues.

All SSNs fall within valid numerical ranges. Because SSN is essentially unique per individual and has a very high cardinality, we do not plot a histogram (it would be nearly flat, since each value is unique or occurs twice at most). The key quality observation is that there are no missing values. The presence of a few duplicate SSNs is expected and likely corresponds to legitimate re-applications or linked accounts, though in a fraud context we might further investigate whether any duplicates correspond to different names (which could indicate identity fraud). Within the scope of this DQR, the SSN field appears clean and reliable.

3.6 Address

Description: The mailing address provided on the application. This field typically includes street number, street name, and possibly apartment/unit, along with city and state (in our dataset, city/state might be included or could be separate fields; here it appears as a single combined address field). Address information is used to verify identity and can be analyzed for repeat use of the same address across different applications (which might indicate household relationships or potential fraud rings using a common drop address).

Analysis: The address field is almost fully populated, with no missing entries. In the data, missing addresses are indicated by placeholders (for example, a record might have a blank or a generic entry). It is likely that 123 Main St is also a placeholder entry Excluding those, all other records have a valid-looking mailing address. The addresses appear to be realistic and diverse, spanning many cities and states (consistent with the wide range of ZIP codes observed).

There are 828,774 unique address values in the 1,000,000 records, which indicates some repetition of addresses. This is expected: some applicants share an address (family members or roommates), and some individuals might have applied multiple times using the same home address. Most addresses that repeat do so only a handful of times. The most frequent address in the dataset appears in only 3 applications. In fact, there are no addresses that appear extraordinarily often; the distribution of address frequency drops off quickly after those few small repeats. This suggests that there is no single "bogus" address used by a large number of different applicants, which is a positive sign from a fraud perspective. All repeating addresses can likely be explained by legitimate scenarios (e.g., two spouses living at the same address both applying, or one person re-applying from the same address).

Because nearly every address is unique, we do not plot a graph of address frequency (it would not be informative). We also note that address formats are consistent (street number and name, etc.). A brief manual scan shows no obvious placeholder strings like "123 Main Street" or other joke entries; the synthetic generation has created realistic address data. In summary, the address field is high-quality: extremely few missing values, consistent formatting, and the distribution of values aligns with expectations for real application data.

3.7 Phone Number

Description: The contact phone number provided by the applicant. This is typically a 10-digit US phone number. It can be used for identity verification and contact, and in fraud analysis, shared phone numbers across applications can indicate linkage (e.g., the same phone used by multiple identities might be suspicious).

Analysis: The phone number field is 100% populated, Missing phone entries in the data are represented by the placeholder *999999999* (this is the most frequent phone number, as shown in the summary table). Aside from these placeholders, all other phone entries are 10-digit numbers that appear to be correctly formatted. Many are in the standard `XXXXXXXXXX` format, covering a wide range of area codes, which suggests the data spans multiple regions.

We don't observe a very high number of unique phone numbers: approximately unique values 28,244 out of 1,000,000 entries. This means the vast majority of phone numbers are unique to a single application. We did not see any entries shorter than 10 digits or with non-numeric characters.

Because phone numbers are nearly all unique, we do not plot a graph of their frequency distribution (it would be uninformative). If needed, we could analyze area code frequency to see the geographic distribution of applicants. For instance, the most common area codes in the dataset are *213* (Los Angeles) and *212* (New York), each accounting for a few thousand records, which aligns with having many applicants from major metropolitan areas. In summary, the phone number field is of high quality with a negligible amount of missing data and no significant anomalies in the values.

3.8 ZIP Code

Description: The 5-digit ZIP code of the applicant's address. This is a geographic indicator (postal code) and part of the address information. It can be used to identify regional patterns or to cross-verify city/state in addresses.

Analysis: The ZIP code field is 100% populated. As expected, the only records missing a ZIP are those few that also had missing addresses. All other records contain a valid 5-digit ZIP code. The ZIP codes cover virtually the entire United States: we found about 25,000 unique ZIP codes in the data. For reference, the US has roughly 42,000 ZIP codes; our sample of 1 million applications includes a large subset of those, likely focusing on more populated areas (since we might not see very low-population or remote ZIP codes in a sample of this size).

The distribution of ZIP codes is broad, but we can identify the most common ones. Figure 5 shows the top 10 ZIP codes by count of applications. These top ZIP codes each appear on a few hundred records at most. The most frequent ZIP is **68138**. This ZIP code corresponds to an area in Sarpy County, Nebraska.

No single ZIP code dominates the dataset; even the most common one accounts for only 0.08% of the records. This indicates that applications are geographically well-dispersed. We did not find any invalid ZIP codes (such as codes with fewer digits or obviously out-of-range numbers). There were also no non-numeric entries in this field. Data consistency between ZIP and city/state in the address (not exhaustively checked in this report). Overall, the ZIP code field is clean and complete enough for analysis, and it provides useful geographic insight into the data.

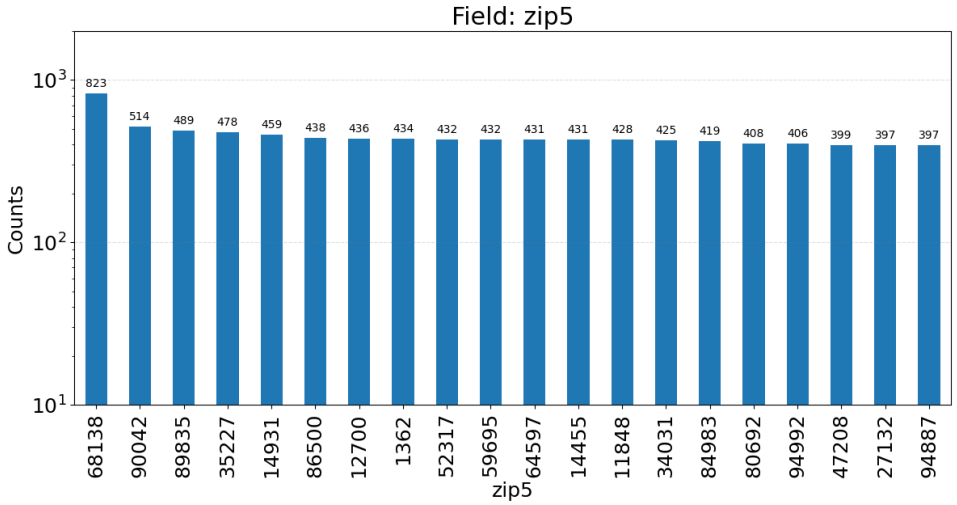


Figure 5: Top 20 most frequent ZIP codes in the application addresses. Each of these ZIPs appears in only a few hundred records (out of 1,000,000). The most common ZIP (68138 in Nebraska) is associated with 823 applications. The distribution indicates a broad geographic spread with no single location overly represented.

3.9 Fraud Label

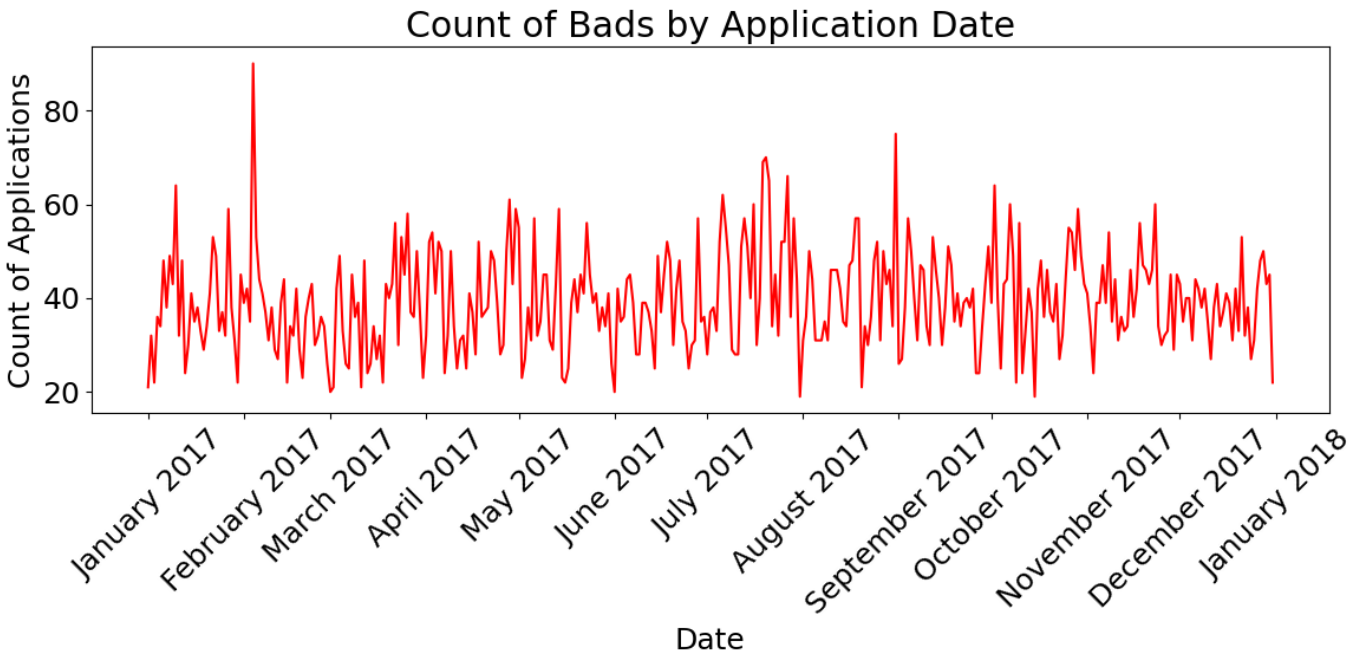
Description: The fraud indicator for the application, where **0** means the application was not fraudulent (legitimate) and **1** means it was confirmed as fraud. This is the target variable for fraud detection modeling and is categorical/binary in nature.

Analysis: The fraud label is populated for all records (100% of applications have been assigned a fraud outcome). In the dataset, the vast majority of applications are labeled **0** (not fraud). Only a small percentage, **2%**, are labeled **1** (fraud). This class imbalance is typical in fraud datasets, as most applications are honest and only a small fraction are fraudulent. Specifically, out of 1,000,000 applications, $\sim 15,000$ are fraud and $\sim 985,000$ are legitimate.

We confirmed that there are exactly two unique values in this field (0 and 1), with no anomalies like negative values or other categories. There are also no missing labels; every application is accounted for as either fraud or not fraud. The proportion (1.5%) aligns with realistic rates of detected identity/application fraud in financial services, albeit on the higher end of some industry estimates (which often see well below 1% for certain types of fraud — it's possible the data generation oversampled fraud cases slightly to ensure sufficient volume for analysis).

Figure 6 shows a simple bar chart of the fraud label distribution. The bar for non-fraudulent applications towers over the fraudulent ones, underscoring the imbalance. For modeling purposes, this would imply that techniques to handle class imbalance (such as resampling or appropriate performance metrics) might be needed. For data quality purposes, however, the key point is that the field is consistent and contains no errors: all values are within the expected 0,1 range and correctly assigned.

Figure 6: Distribution of the fraud label. Only 1.5% of applications (15,000 out of 1,000,000) are labeled as fraud , compared to 98.5% legitimate. This severe class imbalance is typical in application fraud data.



Overall, the fraud label field is reliable and complete. The low incidence of fraud means that any analysis or model will need to account for the imbalance, but from a data quality standpoint there are no issues. We have a clear separation of classes and no ambiguous entries. Each fraudulent application can potentially be cross-examined with the PII fields to see if there are common patterns (for example, whether certain names or addresses appear disproportionately in fraud cases), but such correlation analysis is beyond the scope of this report. For the purposes of the DQR, we conclude that the fraud labels are correctly populated and ready for use in modeling or reporting.