Credit Card Fraud Transaction Detection
CIS*4020

Gariel Mahwastu
Garien Rahadi
Thomas Martin
Simonas Tamkevicius
Vedant Kanabar

Introduction

Credit card transactions have become a vital part of financial systems worldwide. With the advancement of technology, which has led to the rise of online shopping, digital banking, and cashless transactions, credit card usage has grown significantly. However, this has also led to an increase in fraudulent activities, costing billions of dollars every year. Detecting and preventing credit card fraud has become critical for protecting assets and maintaining trust between consumers and financial institutions. This project aims to leverage machine learning techniques to analyze and detect fraudulent credit card transactions.

Problem Statement:

Our key research question is:

"How can we effectively detect credit card fraud based only on transaction information?"

We chose this initial question to help us explore the best models for data science with simplicity in data processing. Typically, the data generated by the current transaction is the most accessible and efficient to utilize in real time, which is why we are choosing to only consider the current transaction information.

If time and data allow, we would further process this question to not only consider the current transaction information but also the user's previous transaction information. We will do this by examining different models with varying transaction history length (e.g. comparing n=2,5,10,... transactions back). This comes with time and data restrictions, which we will evaluate for further development.

Background:

Credit card fraud is one of the biggest forms of financial crime in the world, resulting in billions of dollars worth of losses each year for banks, retailers, and customers. Additionally, as the online e-commerce space grows, so does the rate of attempted credit card fraud. However, this type of fraud can take many forms, including stolen credit card information, identity theft, counterfeit cards, and transaction manipulations, making it hard to identify (Department of Justice Canada, 2022).

With the growing problem of credit card fraud, a lot of financial institutions have turned to utilizing machine learning to detect fraudulent transactions. Machine learning models are able to learn complex patterns from vast amounts of transaction data and classify it using different techniques. These include supervised learning, which classifies transactions based on whether they are likely to be fraudulent or not (Afriyie et al., 2023) and unsupervised learning (anomaly detection), which identifies outlier transactions that deviate from the norm (Rai & Dwivedi, 2020). Machine learning can also be used to detect fraudulent transactions in real time, eliminating the need for manual monitoring by human interaction, which sometimes can miss a detection. It can also learn and adapt to new information so that a change of tactics in fraudulent transactions can still be detected (Stripe, 2025).

Detailed Literature review:

In the study *A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions,* (Afriyie et al., 2023) considered three different machine learning models, including logistic regression, random forest, and decision trees, were considered to determine which model was the best performing. They used a synthetic dataset from Kaggle, since real financial information is very hard to get your hands on, and balanced it to prevent overfitting, as there are far more legitimate transactions than illegitimate ones. After training and testing all three models, they determined that out of logistic regression, random forest, and decision trees, the model they found that performed the best was the random forest with a 96% accuracy. This is compared to the 92% from the decision tree and logistic regression models. The paper made conclusions like most fraud occurs between 22:00GMT and 4:00GMT or most fraud victims are over 60 years old (Afriyie et al., 2023), which are insights we also hope to capture with our research.

Fraud Detection in Credit Card Data using Unsupervised Machine Learning Based Schemes (Rai & Dwivedi, 2020) explored the use of unsupervised learning, more specifically, a Neural Network technique to identify fraudulent credit card transactions. The authors of this paper also used a Kaggle credit card fraud dataset, which was a set of PCA-normalized vectors with no meaning but based on real customer data. Although it differs from our goals since the vectors have no meaning, it does tell us which models work well. Though complete implementation details are not written about, a comparison is made between multiple unsupervised learning approaches, including Neural Network (NN), Auto Encoder (AE), Isolation Forest (IF), Local Outlier Factor (LOF), and K-Means clustering. The results gathered by the authors show that the best-performing unsupervised learning algorithm was the Neural Network, followed by K-Means Clustering. Since there were no meaningful vectors in the data (after PCA normalization) for anonymity, the data concerns were not covered.

In the paper *Data Leakage and Deceptive Performance: A Critical Examination of Credit Card Fraud Detection Methodologies* (Hayat & Magnier, 2025), the authors critically analyzed common pitfalls in credit card fraud detection research, focusing on the impact of data leakage and improper evaluation techniques. They examined multiple machine learning approaches, including logistic regression, random forest, and neural networks, across widely used datasets. The study highlights that reported high accuracies in many prior works can be misleading due to subtle forms of data leakage, such as improper train-test splits or feature selection based on future information. The authors emphasize the need for rigorous experimental design and propose best practices for evaluation to ensure that fraud detection models provide genuinely reliable performance estimates. This paper will guide us in choosing the right methodologies and data sampling techniques as we avoid the pitfalls of other papers.

In the paper *Credit card fraud detection using asexual reproduction optimization* (Farhang Ghahfarokhi et al., 2021) introduced an innovative approach to credit card fraud detection by employing the Asexual Reproduction Optimization (ARO) algorithm. ARO is a supervised method that enhances classification performance by focusing on the majority class during training, effectively addressing class imbalance – a common challenge in fraud detection tasks. The authors benchmarked ARO against traditional methods, demonstrating superior recall and reduced training time, making it a promising candidate for real-time fraud detection systems. It guides us on new approaches being developed in the credit card fraud detection space and provides us with a new data set for validation.

(Chung & Lee, 2023) proposed a hybrid machine learning model combining K-Nearest Neighbours (KNN), Linear Discriminant Analysis (LDA), and Linear Regression (LR) to enhance recall in credit card fraud detection. Their approach addresses the critical need for high recall in fraud detection by integrating these algorithms, which individually excel in capturing different aspects of the data. The study's results underscore the effectiveness of this combined strategy in improving recall without compromising other performance metrics. One important aspect of this study that we are incorporating in our project is combining datasets to reduce overfitting on any single dataset or simulator, since readily available real data is very rare when dealing with credit card fraud.

Materials and Methods:

Dataset details:

As discussed earlier, in the world of banking, we do not have readily available access to datasets due to the privacy of customer data; instead, we have to rely on some datasets based on the real world and simulator data.

The first dataset we are using is by (Shenoy, 2020), which is data created by a simulator which mimics transactions of a normal credit card user based on factors such as a user's age, demographic, occupation, etc. The data set is very large and includes a variety of user data, with 1,852,394 rows, 23 columns of data and a fraud case proportion of 0.52%. This highlights one of the problems we will face in our methodology of class imbalance. The data is also for transactions between Jan 1, 2019 and Dec 31, 2020, giving us data across the year since different months will have different transaction and/or fraud volumes.

Another dataset we are using is (Choudhury, 2024), which is a dataset consisting of credit card transactions in the western United States. The data is real-life and contains 14,446 rows, 15 columns of data and a fraud case proportion of 12.7%. There are some common features with the previous dataset that make the database combinations a lot easier. This adds a layer of real-life data we can use for both training and validation.

Looking at the datasets, when we start processing, we have to consider some issues. Firstly, the simulator data is overly present in the data from one simulator (Shenoy, 2020). Unfortunately, there are no other simulator datasets present on that scale (with 1,852,394 data points), and in banking in general, real-life data is very hard or impossible to get, as it is private to the user and the bank. This is a common theme across research papers (Marazqah Btoush et al., 2023), as there is a lack of real-life data with well-defined features. This would very easily violate a user's data, hence why it's hard to find such real-life data sets.

Another major issue seen is class imbalances (fraudulent vs non-fraudulent transactions). Overall, a good approach to overcome this is oversampling the fraudulent transactions or undersampling the non-fraudulent transactions in model training (Makki et al., 2019). These approaches will be experimented with, and the best sampling method will be chosen according to what works best with our data.

Finally, another issue is how we plan on combining the data sets. Some features are common across data sets, like gender or transaction time, or transaction amount, but there are some features present in certain datasets and not the others. In such a situation, do we ignore those features and only focus on the common features? Or, individually evaluate the models and train them data-set-wise. These are some of the questions we will answer through experimentation in our methodology because, ideally, we want a model to be able to perform well across datasets and be applicable in real life.

Data Processing:

Cleaning Data:

Because we will be working with two different datasets, there are some things we will need to do before we can combine them.

First, we will need to make sure that feature names are consistent across the two datasets. As a part of this step, we might need to rename some columns to make sure identical variables share the same name.

Next, we will need to align the schemas so that each feature column exists in both datasets, creating new feature columns if needed. We need both datasets to have the same structure so that it is easier to combine them (this will be examined in the next section).

Finally, we will enforce data type consistency. This will be done by verifying that each column follows the same structure and making sure numeric values are stored as the appropriate type (e.g. float or int). This will help avoid data mismatches between the two datasets.

One additional step we will need to take is to convert any string label fields into numerical values. There might be some minimal differences in the two datasets in terms of label names, e.g. Doctor vs doctor, which we will convert before encoding the labels. We will do this using the label encoder from sklearn.preprocessing (scikit-learn developers, 2025) after aligning the data correctly.

As part of the cleanup process, we will also be removing unused or identifier columns from the dataset. These columns will contain information that will not be relevant for predicting fraudulent transactions, such as street names, zip codes, first and last names, and transaction numbers. Removing these columns ensures that our model will not learn from these identifiers and unused columns, making our model actually learn patterns of true transaction fraud.

Combining datasets:

To combine the datasets, we are using the pandas concat function, which combines the two datasets by vertically stacking them. By doing so, if one dataset has columns that don't exist in the other dataset, it will keep all columns from both datasets and fill the missing values with NaN in the rows.

We then standardized several formatting issues, such as converting date and time fields into the same format to maintain consistency across all entries. For categorical variables, we applied a label encoder, which transforms non-numeric categories into integer values that machine learning algorithms can interpret.

Finally, to address the missing or undefined values, we used the SimpleImputer with the median, a tool that automatically fills these missing or undefined values by using the median. An important note is that we run it only on training data to avoid any data leaks. This helps keep the data consistent and prevents problems when training the models, especially when dealing with the impact of extreme outliers.

Model Training:

Fixing Class Imbalance:

As discussed earlier, we have a great class imbalance problem that has been an issue covered in great detail (Makki et al., 2019) for credit card fraud models. To overcome this issue, we want to try different sampling methods with the **Random Forest Classifier** (see below for model details) and try to identify which sampling method will be the best for our actual model training. The data will be split with a Stratified testing-training split. The following methods will be considered:

A stratified **Random Sample** will be the first sampling method we will consider. This will give us a baseline to compare the other sampling methods with.

Random Over-Sampling is where we over-sample the minority class(es) by picking samples at random with replacement to ensure class sizes match (Lemaitre & Aridas). Although this seems very extreme, it is a method we want to experiment with.

Synthetic Minority Over-Sampling Technique (SMOTE) is another suggested method for such class imbalances (Makki et al., 2019). In SMOTE, we first find the nearest neighbours in the minority class, then create artificial data points between x_i and x_nn (where x_nn is one of the nearest neighbours) until there is no class imbalance (Maklin, 2022). This is one of the most common ways to fix class imbalances.

Adaptive Synthetic Sampling (ADASYN) is an extension of SMOTE, which focuses on generating samples next to the original samples which are wrongly classified using a k-Nearest Neighbours classifier (*2. over-sampling#*). ADASYN tries to create more synthetic points near the complex boundary region of the class compared to SMOTE (Tang, 2023).

Random Under-Sampling is where we under-sample the majority class(es) by picking samples at random with replacement to ensure class sizes match (Lemaitre & Aridas). This gives us a base reading on any undersampling techniques.

Near-Miss Under-Sampling is another method to deal with class imbalances. It works by randomly undersampling from the majority class based on the version of the algorithm until there is a class balance (Lemaitre & Aridas). Version 1 of NearMiss involves randomly picking points close to the minority class, version 2 chooses points furthest away from the minority class, while version 3 is a hybrid of both approaches (Globaldee, 2023).

To evaluate the model (the Random Forest Classifier for all) and which sampling methods fit well for our data, we use the evaluation techniques detailed in the Model Evaluation section. This will be the sampling method we use to train the final model.

Training models:

Since credit card fraud detection involves identifying a very small number of fraudulent transactions among a very large number of legitimate transactions, some models perform much better than others due to the highly imbalanced and nonlinear nature of the data.

Random Forest is one of the most effective models, as it combines multiple decision trees to capture complex relationships, handle noisy data, and reduce overfitting. It provides interpretable feature importance scores, allowing us to identify which features contribute most to distinguishing fraud vs. legitimacy, and it can be adjusted to account for class imbalance through weighted sampling in addition to our resampling method.

AdaBoost is another strong option for this. Since it works by combining a series of simple models, with each new model focusing on the transactions the previous models misclassified, it is better able to detect rare and hard-to-identify fraudulent transactions. However, it can be sensitive to noisy or mislabeled data, which may reduce its accuracy.

Support Vector Machines (SVMs) can also be effective for this. Since they find the best boundary that separates fraudulent from legitimate transactions and can use kernel functions to handle complex patterns, they can capture nonlinear relationships in the data. However, they can be computationally intensive on large datasets and require careful handling of class imbalance.

In addition, **Logistic Regression** is a simple and interpretable baseline model. Since it estimates the probability of fraud based on a linear combination of features, it is able to provide clear insights into how each variable affects the overall outcome. However, it assumes a linear relationship and often struggles to capture the complex patterns typically found in actual fraud, and it requires adjustments to handle imbalanced data.

Finally, a single **Decision Tree** or **Lasso Regression** can be used for initial analysis or feature selection. Decision Trees are interpretable but prone to overfitting, while Lasso can help identify important features, it also assumes linear relationships and may not handle nonlinear fraud patterns effectively.

Overall, ensemble tree-based models such as **Random Forest** and **AdaBoost** will be the most effective for our scenario of identifying credit card fraud. This is due to their ability to model complex patterns, handle class imbalance, and achieve high recall and precision for uncommon fraudulent transactions.

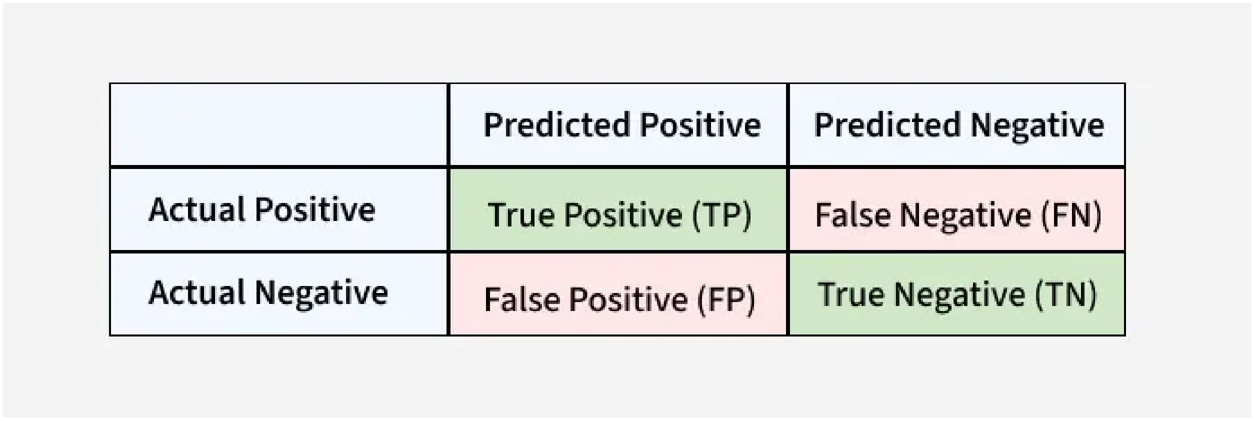
During the training process of these models, we will also perform hyperparameter optimization. Hyperparameter optimization will be an important part of our model training that will enable us to find the best hyperparameter settings for each model in order to maximize its performance. To perform hyperparameter optimization, we will be using **Random Search** as our primary method (due to the computational cost and time requirement associated with Grid Search), as **RandomizedSearchCV** from scikit-learn. RandomizedSearchCV evaluates a fixed number of randomly sampled hyperparameter combinations, making it much more efficient than GridSearchCV, particularly when models involve large or complex hyperparameters (scikit-learn developers, 2025; Chauhan, 2023).

Model evaluation:

Model evaluation will be one of the important aspects of our project. We have to ensure that the model we pick is the most reliable and accurate, especially with datasets that are highly imbalanced. In this context, model evaluation involves testing and analyzing the model's predictive performance on unseen data to ensure it generalizes well beyond the training set. Since we have imbalanced data, we will need to perform resampling by performing oversampling or undersampling to ensure that our dataset can be balanced during evaluation. Since our data is imbalanced, we will also need to take into account other performance metrics other than accuracy, like the Confusion Matrix, Precision, F1 Score, and Recall (Singh, 2025).

Confusion Matrix

Confusion Matrix works by taking the classification result and groups them into four categories: True Positive (model predicted positive correctly), True Negative (model predicted negative correctly), False Positive (model predicted positive while it's actually negative), False Negative (model predicted negative while it's actually positive). The numbers for each category are useful to calculate metrics such as Precision, Recall, and F1 Score (GeeksforGeeks, 2025).



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Precision

Precision indicates the ratio of True Positives and all Positives. This calculates how many correct Positive predictions were made by the model (GeeksforGeeks, 2025). The formula for Precision is:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall

Recall, also called True Positive Rate, measures the proportion of actual positives that were correctly predicted (GeeksforGeeks, 2025). Recall answers the question, "Out of all the actual positive examples, how many did the model correctly identify as positive?" (Kashyap, 2024).
The formula for Recall is:

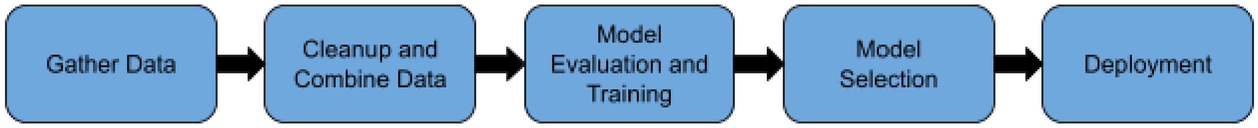
$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

F1 Score

F1 Score is a performance metric that combines both Precision and Recall using the harmonic mean (GeeksforGeeks, 2025). F1 Score uses the harmonic mean to balance out the importance of both Precision and Recall into a single value (Kashyap, 2024). The formula for F1 Score is:

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Method Diagram:



This is a diagram that shows our development process for this project. For the Gather Data part, refer to the Dataset details section. For our second step, we have also discussed our method in the data processing section above. For model evaluation and training, we have 3 steps: finding a resampling method, training models, and evaluating models, as discussed above in the model training section. For model selection, we will be taking the result of model evaluation and picking a model that performs the best. Lastly, deployment is the part where we would deploy the model to actually perform predictions on new and unseen data.

Results:

Resampling experiment:

As described in the fixing class imbalance section, to fix the issue present in our data, we applied different resampling methods to try and find the best one for our data. We ran all these on a **Random Forest Classifier** model. After running our code, below is the summary of the results:

Property	Random Sample	Random Oversampling	SMOTE	ADASYN	Random Under Sampling	Near Miss 1	Near Miss 3
Accuracy	99.77%	99.78%	99.69%	99.71%	99.77%	96.34%	96.79%
Precision	92.35%	89.13%	77.89%	82.47%	92.35%	15.70%	17.48%
Recall	74.22%	78.45%	78.56%	74.49%	74.22%	94.60%	94.17%
F1-Score	82.30%	83.45%	78.22%	78.28%	82.30%	26.94%	29.49%
True Positive	74.22%	78.45%	78.56%	74.49%	74.22%	94.60%	94.17%
True Negative	99.96%	99.93%	99.84%	99.89%	99.96%	96.35%	96.81%
False Positive	0.04%	0.07%	0.16%	0.11%	0.04%	3.65%	3.19%
False Negative	25.78%	21.55%	21.44%	25.51%	25.78%	5.40%	5.83%

Our objective with the resampling method choice was the lowest False Negative rate, which was achieved by the **Near-Miss-1 Under-Sampling**. This meant, however, that we had one of the highest False Positive rates of 3.65%. Although we had a high Recall score, we had the lowest Precision score of 15.70%. We gained a lot in terms of identifying the correct True Positives, but the caveat is the way higher number of false positives. In a banking sense, this means a lot of things in terms of dealing with a very large number of wrongly identified frauds, with a lot of customer complaints and failed transactions that are not ideal. But in the banking world today, there are more possibilities like asking the customer to insert the credit card if it's a machine transaction or asking for an OTP (One Time Password) confirmation for a transaction. Yes, it might be more inconvenient for the customers, but it significantly reduces the number of frauds. This was our mindset when selecting the best models as well.

One thing to note here is that we were unable to run Near-Miss-2 or Near-Miss version 2 due to space restrictions. The way version 2 has been implemented in the `imbalanced-learn` library makes it calculate all pairwise distances between each minority class data point and each majority class data point which for versions 1 & 3, we only calculate them for the k nearest neighbors (default value of 3) making it almost impossible for us to run it on our machines since 1 million + majority class data points * 10,000+ minority class data points are already needing multiple terabytes of application storage for the process to run.

Hyperparameter Optimization:

The tables below display the results of running a hyperparameter tuning script using Randomized Search via RandomizedSearchCV from scikit-learn to tune the KNN, AdaBoost, Decision Tree, Random Forest, and Linear SVM models introduced earlier. Naive Bayes and Logistic Regression were not included in the tuning process, as these models either have few or no meaningful hyperparameters to optimize in this context.

KNN:

Hyperparameter	Description	Range	Optimal Value
n_neighbors	Number of nearest neighbours used to classify each sample	randint(3, 10)	9
weights	Determines whether all neighbours contribute equally or if closer neighbours have more influence	[“uniform”, “distance”]	“distance”
p	Power parameter for Minkowski distance (1=Manhattan, 2=Euclidean)	[1, 2]	1

AdaBoost:

Hyperparameter	Description	Range	Optimal Value
n_estimators	Number of boosting stages in the ensemble	randint(100, 500)	357
learning_rate	Scales the contribution of each weak learner (higher values boost)	uniform(0.001, 2)	np.float64(1.7009102740624293)

Decision Tree:

Hyperparameter	Description	Range	Optimal Value
max_depth	Number of nearest neighbours used to classify each sample	[None, 20, 30, 40, 50]	None
min_samples_leaf	Minimum samples required to form a leaf node	randint(1, 5)	4
min_samples_split	Minimum samples required to split the internal node	randint(2, 10)	9

Random Forest:

Hyperparameter	Description	Range	Optimal Value
bootstrap	Number of nearest neighbours used to classify each sample	randint(3, 10)	False
max_depth	Maximum depth of each tree	[None, 20, 30, 40, 50]	None
min_samples_leaf	Minimum samples required to form a leaf node	randint(1, 5)	1
min_samples_split	Minimum samples required to split the internal node	randint(2, 10)	5
n_estimators	Whether sampling with replacement is used when building trees	randint(100, 500)	353

Linear SVM:

Hyperparameter	Description	Range	Optimal Value
C	Controls the balance between maximizing margin and minimizing classification errors	uniform(0.001, 2)	np.float64(9.897248354111614)

Model Training and Evaluation:

The results table below is produced from a model training and evaluation script that applies preprocessing, NearMiss undersampling, and feature scaling before training the models listed earlier using optimized hyperparameters. Each model is fitted on the resampled and preprocessed training data, then used to generate predictions on the test set. The script calculates accuracy, precision, recall, F1 score, and normalized confusion matrix values for each model and saves the outputs along with confusion matrices and feature importance plots.

Metric	Logistic Regression	Decision Tree	Random Forest	SVM Linear	KNN	Naive Bayes	AdaBoost
Accuracy	86.21%	95.87%	95.74%	86.57%	61.22%	81.68%	93.04%
Precision	11.34%	13.55%	15.75%	12.66%	1.09%	7.28%	8.48%
Recall	76.74%	96.15%	95.13%	76.95%	64.60%	69.73%	93.32%
F1 Score	19.76%	23.75%	27.02%	21.75%	2.14%	13.18%	15.55%
True Positive	76.74%	96.15%	95.13%	76.95%	64.60%	69.73%	93.32%
True Negative	95.69%	95.59%	96.34%	96.19%	57.83%	93.62%	92.77%
False Positive	4.31%	4.41%	3.66%	3.81%	42.17%	6.38%	7.23%
False Negative	23.26%	3.85%	4.87%	23.05%	35.40%	30.27%	6.68%

One of our key objectives with the best model selection is the lowest False Negative rate, as that reduces the actual number of fraudulent transactions that slip through. Looking at this data and the resampling data, it is clear that it comes at the cost of precision. Our current best model is a Decision Tree.

The results indicate that nonlinear, tree-based models such as Decision Trees, Random Forest, and AdaBoost are the most effective for this dataset, achieving high accuracy and recall as well as balanced error rates. Models such as KNN and the linear classifiers perform worse, reflecting the lack of clear linear separation or neighbourhood structure in the data. This supports the hypothesis that clustering methods would not be effective, as fraudulent vs. non-fraudulent transactions do not form distinct groups in feature space. Overall, the models that can learn complex, feature-specific boundaries perform best and are most suitable for our data.

Model with History:

For the model with history, we first needed to create historical features so the model could use the data of previous transactions from the same card. We took the features that changed between transactions, like the merchant, the merchant latitude and longitude, the category, the amount, and whether or not there were fraudulent transactions previously. With this, we also calculated a few new features, such as, time between each transaction, the average amount spent across all transactions on the given account, and the total amount of transactions within the last 24 hours on the card.

To create these historical features, we needed some sort of identifier to link transactions from the same account. We found that the dataset from (Choudhury, 2024) did not have this type of identifier; however, the dataset by (Shenoy, 2020) did. This meant that the models with history were only trained on this single dataset. The (Shenoy, 2020) dataset had a credit card number (cc_num) column, which we used to group the transactions and create the historical features from there.

Because we wanted to find what the optimal transaction history length for the model was, we experimented with varying lengths. We decided to create a model with 1, 3, 5, and 7 historical transactions and tested each with our models. This also tells us how the models change when more history is sent down and passed to the model, and how they behave in terms of the patterns we see with an increase in history. All of the results we have in terms of the confusion matrices and top feature plots can be found in Appendix C. The numerical results have been summarized below in the tables:

Model with a history of 1 transaction			
Metric	**Random Forest**	**AdaBoost**	**Decision Tree**
Accuracy	98.99%	98.70%	99.17%
Precision	39.59%	29.23%	37.01%
Recall	98.87%	98.80%	99.33%
F1 Score	56.54%	45.12%	53.92%
True Positive	98.87%	98.80%	99.33%
True Negative	99.12%	98.61%	99.02%
False Positive	0.88%	1.39%	0.98%
False Negative	1.13%	1.20%	0.67%

Model with a history of 3 transactions			
Metric	**Random Forest**	**AdaBoost**	**Decision Tree**
Accuracy	98.56%	98.63%	99.14%
Precision	39.06%	25.51%	38.52%
Recall	98.00%	98.93%	99.20%
F1 Score	55.86%	40.56%	55.49%
True Positive	98.00%	98.93%	99.20%
True Negative	99.11%	98.32%	99.08%
False Positive	0.89%	1.68%	0.92%
False Negative	2.00%	1.07%	0.80%

Model with a history of 5 transactions			
Metric	**Random Forest**	**AdaBoost**	**Decision Tree**
Accuracy	98.18%	98.51%	99.11%
Precision	36.53%	25.94%	38.82%
Recall	97.34%	98.67%	99.13%
F1 Score	53.13%	41.08%	55.79%
True Positive	97.34%	98.67%	99.13%
True Negative	99.02%	98.36%	99.09%
False Positive	0.98%	1.64%	0.91%
False Negative	2.66%	1.33%	0.87%

Model with a history of 7 transactions			
Metric	**Random Forest**	**AdaBoost**	**Decision Tree**
Accuracy	98.04%	98.34%	99.07%
Precision	34.89%	25.71%	42.30%
Recall	97.14%	98.33%	98.93%
F1 Score	51.34%	40.76%	59.26%
True Positive	97.14%	98.33%	98.93%
True Negative	98.94%	98.35%	99.21%
False Positive	1.06%	1.65%	0.79%
False Negative	2.86%	1.67%	1.07%

We can see from the tables above that both the Random Forest and Decision Tree models consistently performed well while examining the outcomes from the models trained with varying transaction history lengths. However, the Decision Tree model performed better than the others for the models utilizing 5 and 7 previous transactions in all significant measures, including Accuracy, Precision, Recall, and F1 Score.

We took a closer look at the top-performing models, which used 5 and 7 historical transactions. The 7-transaction Decision Tree model demonstrated significant increases in Precision and F1 Score, but did not outperform the 5-transaction version in every metric. Because Precision and F1 Score show improved overall classification quality and fewer false positives, these two metrics are essential. As a result, the Decision Tree model that uses 7 past transactions performs better in categorization and yields the most dependable outcomes overall. Due to our process of undersampling, we expected a reduction in precision, but our Decision Tree model was the best in mitigating this loss

In contrast, looking at models with 5 and 7 historical transactions for Random Forest and Ada Boost, the precision is reduced, and overall, the extra history is acting like noise. Thus, something to consider before adding more history is ensuring the right model is chosen for the purpose.

Another important note, looking at Appendix C, is that the merchant is still a top feature of importance in the models with history. This makes sense because a fraudulent transaction from a merchant in the past is a good indicator of a future fraudulent transaction from the same merchant. Another important feature seen in the model with history is whether a previous transaction was a fraudulent one. Repeated fraudulent transactions are a pattern learnt by the model, which makes sense as that is the most common behaviour in real life as well.

Conclusion:

Best Model:

Looking at our data, the Appendix, and the results, we can see that overall our best model is the **Decision Tree** model. Initially, in the model without history, its performance is on par with Random Forest and Ada Boost models, but when provided with history, it does the best job at coping with the cost of undersampling and increasing its precision by almost 3X from 15% to 42%. Clearly, providing more history to the model made it better, so we believe that with more refined historical features and/or more history, we can overcome the precision loss for a more balanced model.

Other models, like KNN, tracking the nearest neighbours, were not suitable for our data as there is no specific separation between the fraudulent and non-fraudulent data points. This tells us that clustering would not work with our data, which supports one of our initial hypotheses.

Additionally, Linear Regression or SVM models also do not perform too well; this is expected as there is no strong separation between the fraudulent and non-fraudulent data points. This is another reason why the decision tree models performed better, as there are more individual feature boundaries based on merchant or amount compared to actual boundaries in an n-dimensional space.

Feature Importance:

One of the things that is very prevalent in all of our models is the importance of the merchant feature. All models seem to learn that the merchant determines whether a transaction is fraud or not the most. This makes sense when we look at it in real life, too, as any large banking company is going to have some sort of merchant blacklist or greylist for fraudulent transactions.

However, this brings out one of the big limitations of our study: we are relying a lot on the simulated data. Although we combined two datasets, the simulated data has a million-plus data points, which overshadows the smaller real-life dataset. Overall, it is very possible that the simulator data created fraudulent transactions with certain merchants only, which the model picked up on and relied on too much.

This is a common limitation in credit card fraud detection, as we have to rely on simulated data a lot more than real-life data. Real-life credit card transaction data is very private to a person and legally protected in all situations. In a sense, it is also a business secret, as such data can help other companies improve their fraud detection. This data is hard to access for researchers, hence the reliance on simulated data hinders our conclusions on feature importance overall. This is a limitation that is hard to fix, as it's hard to get the needed data for a proper study.

What we could do differently:

After the study, we did see that although undersampling helped with lowering the False Negative rate, which we were looking for, it did also cost a lot in terms of the model's precision. Going down that path, we just chose lowering False Negatives as the objective to enable detecting actual fraudulent transactions better, while ignoring the loss of precision and instead improving the model to get that precision back. However, even with our best attempt, we only got a precision of 42% after undersampling. In real life, the cost of precision has a lot greater implications

As a credit card provider, if my fraud detection system is not precise and instead flags a lot of normal transactions as fraud, there are customer implications there. The pain of calling your bank to confirm a transaction often is very inconvenient, and as a company, that means a lot of customer complaints and in the worst case, loss of market share as customers shift to other companies without those issues.

There are a lot more business decisions at play when choosing the best model that we have definitely not considered, and as for now, our current model choices are infeasible for real life. Considering these decisions would have broadened the scope of our project by a lot more than what is intended for the class; hence, we chose not to consider them for now, but it is a great viewpoint to consider in these studies.

Future Expansion:

1. Explore Oversampling: It is clear from our study and overall work that we have mainly considered the undersampling approach when working with a class imbalance. Working with oversampling instead would be a different side of the coin and have its own costs and considerations to make. Maybe even with history, it would be a lot better at getting a more balanced model, which might not have a very high Recall but still be balanced. This would be a great area to explore.

2. Refined Historical features: In our model with history, we mainly give it past transactions and a few refined historical features like average transaction amount or time since last transaction. However, in real life, it is never fast to give actual old transactions, and instead, there are more refined historical features present, like a running alpha mean of previous transactions, or normal geographical location of transactions, and those are easier to calculate, store and update.

3. Explore models without merchant information: Although merchant was a strong feature in our study, we have determined that this could just be due to the simulator data limitations. So running the models without it could warrant us some interesting results with patterns in data for other features we haven't seen. In real life, most merchants would be vetted by the company, so we would already consider a merchant safe; hence, dropping the merchant feature might mimic real life more closely.

Appendix

Appendix A - Resampling:

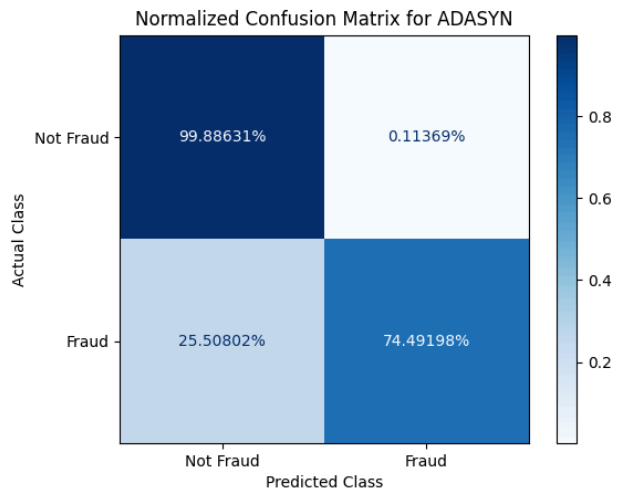


Figure A1. ADASYN Confusion Matrix

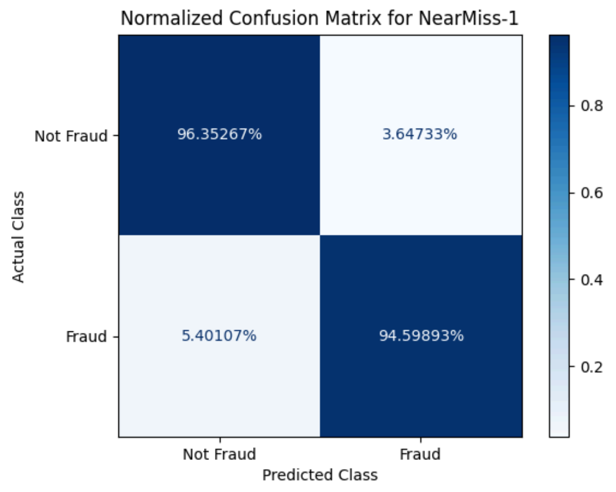


Figure A2. NearMiss-1 Confusion Matrix

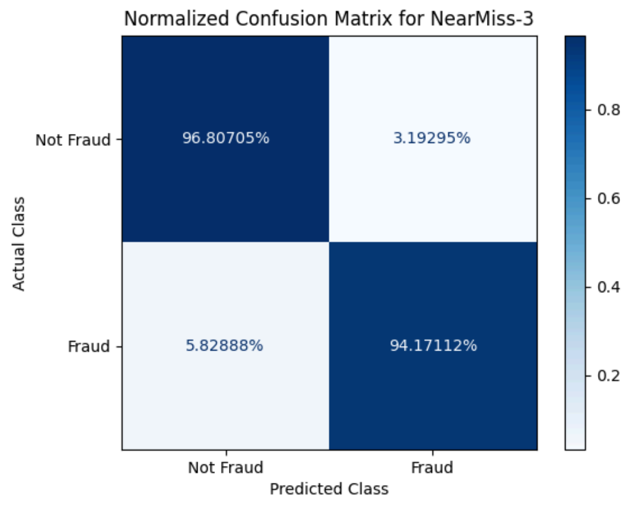


Figure A3. NearMiss-3 Confusion Matrix

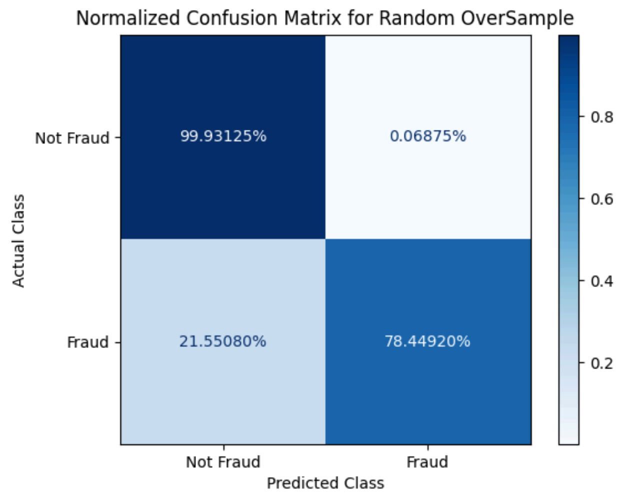


Figure A4. Random OverSample Confusion Matrix

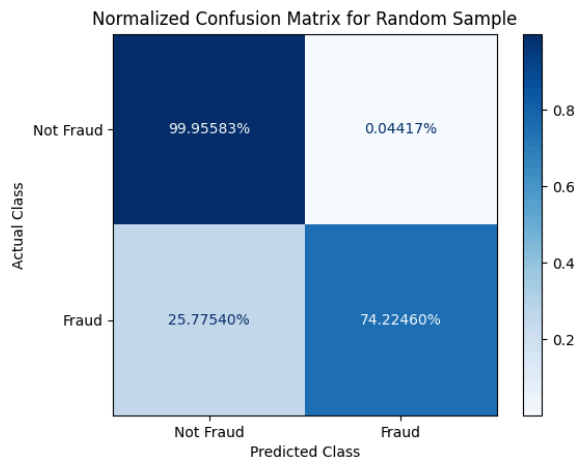


Figure A5. Random Sample Confusion Matrix

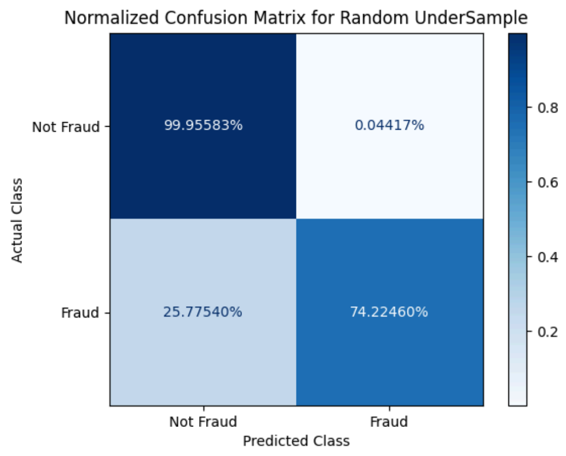


Figure A6. Random UnderSample Confusion Matrix

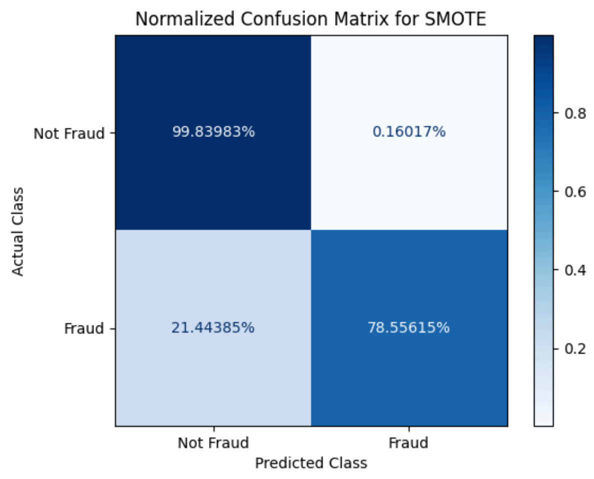


Figure A7. SMOTE Confusion Matrix

Appendix B - Model Training and Evaluation:

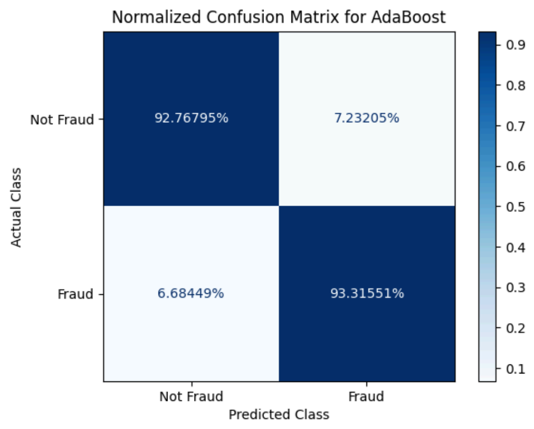


Figure B1. AdaBoost Confusion Matrix

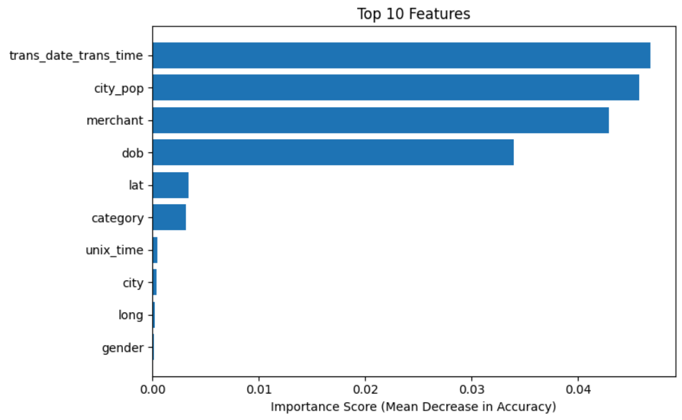


Figure B2. AdaBoost Feature Importance

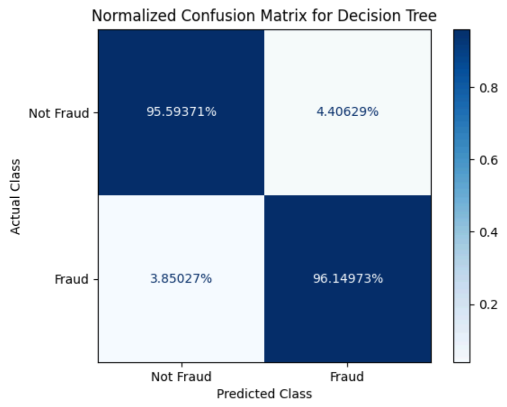


Figure B3. Decision Tree Confusion Matrix

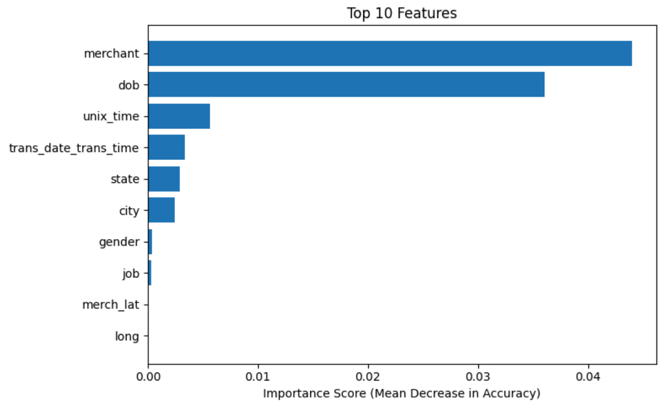


Figure B4. Decision Tree Feature Importance

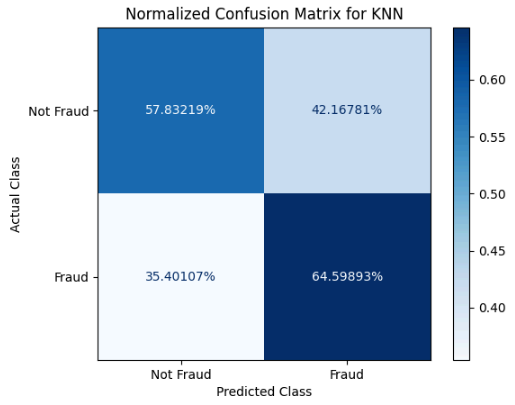


Figure B5. KNN Confusion Matrix

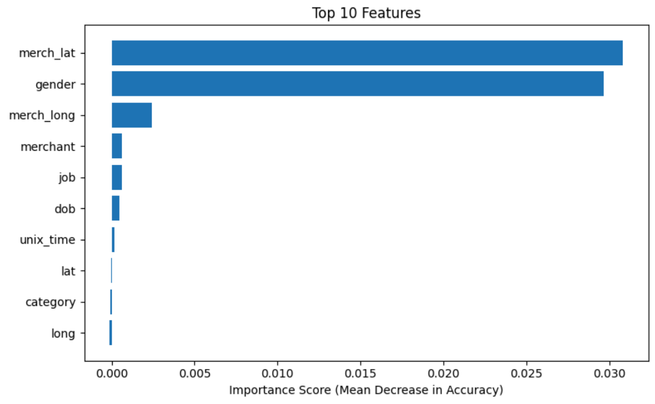


Figure B6. KNN Feature Importance

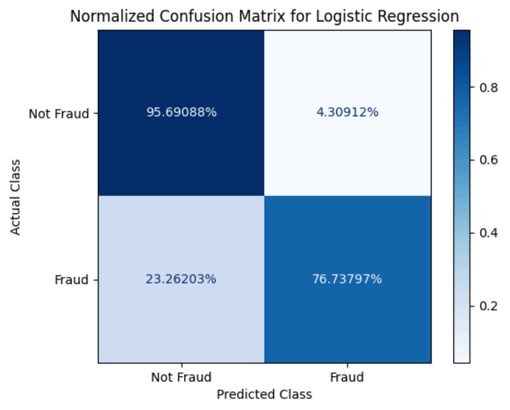


Figure B7. Logistic Regression Confusion Matrix

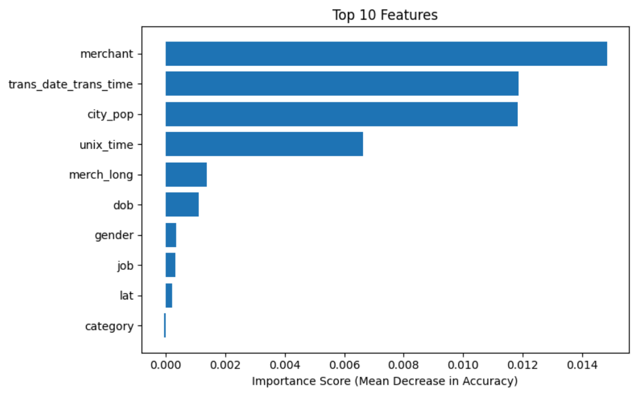


Figure B8. Logistic Regression Feature Importance

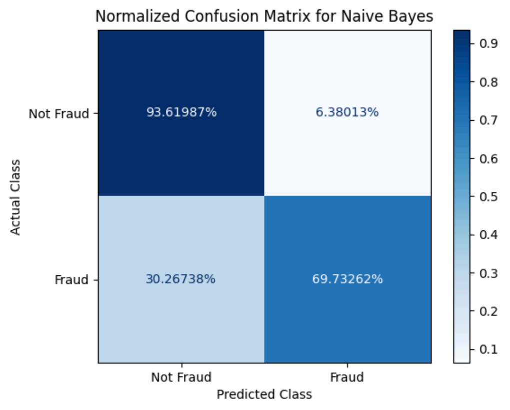


Figure B9. Naive Bayes Confusion Matrix

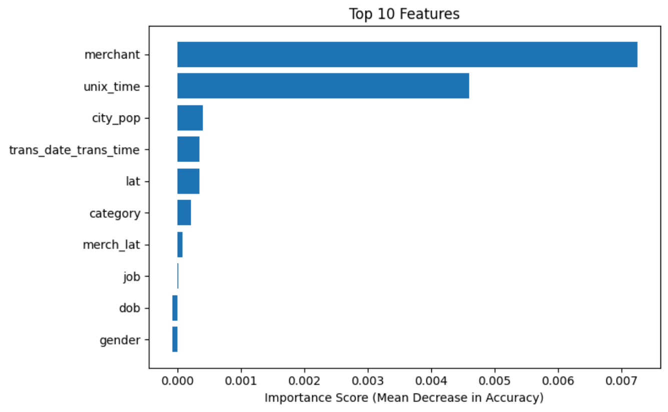


Figure B10. Naive Bayes Feature Importance

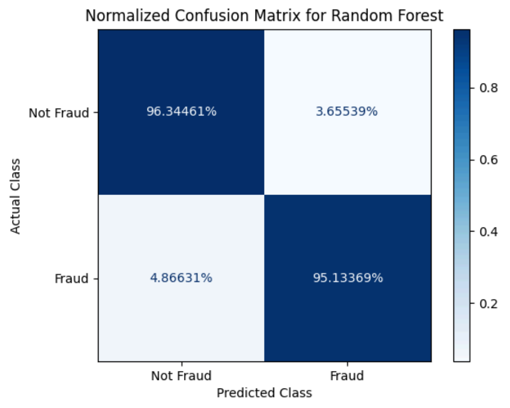


Figure B11. Random Forest Confusion Matrix

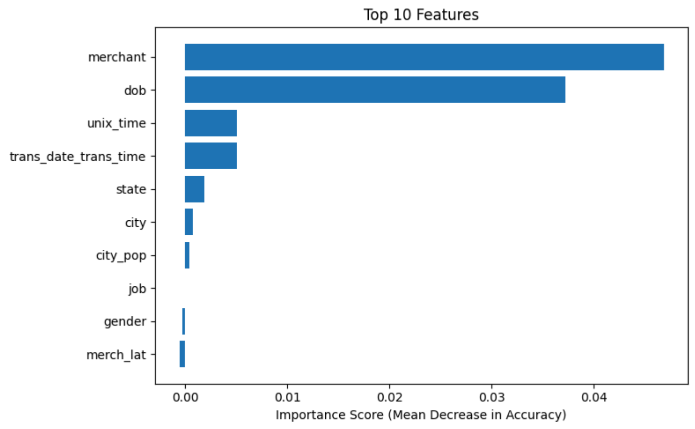


Figure B12. Random Forest Feature Importance

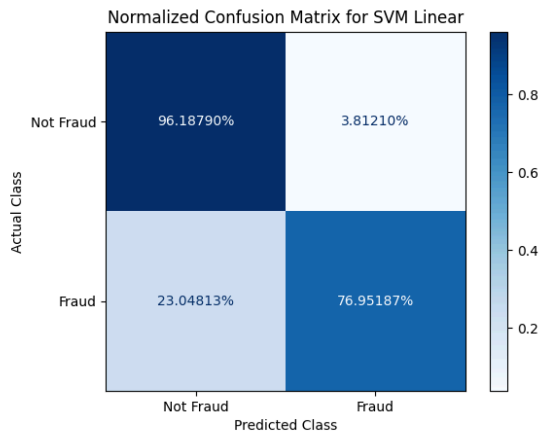


Figure B13. SVM Linear Confusion Matrix

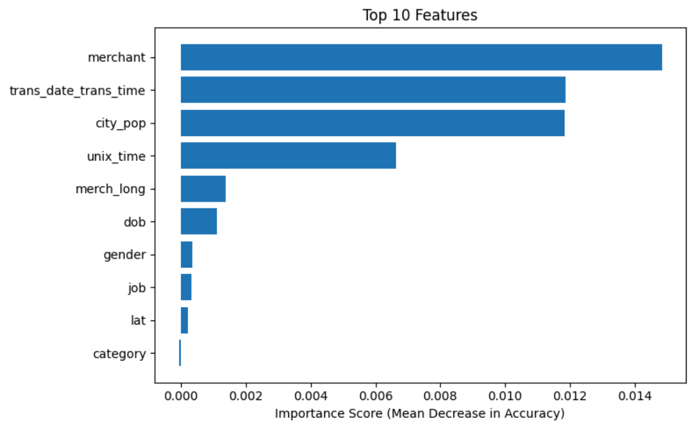


Figure B14. SVM Linear Feature Importance

Appendix C - Models with History:

Models with 1 Historical Transaction

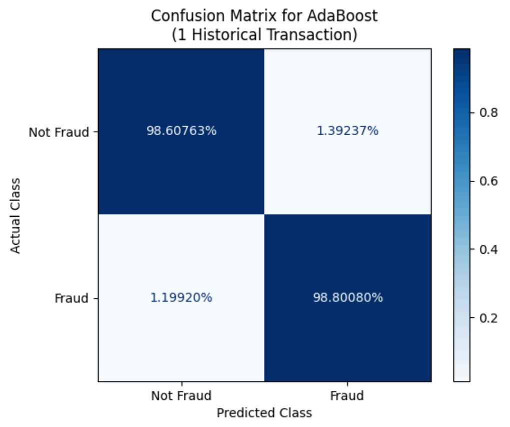


Figure C1. AdaBoost Confusion Matrix

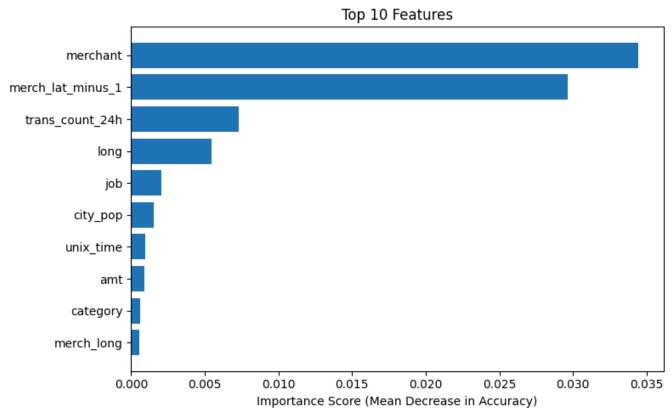


Figure C2. AdaBoost Feature Importance

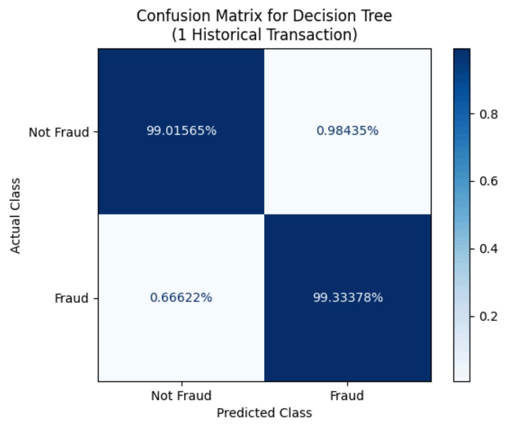


Figure C3. Decision Tree Confusion Matrix

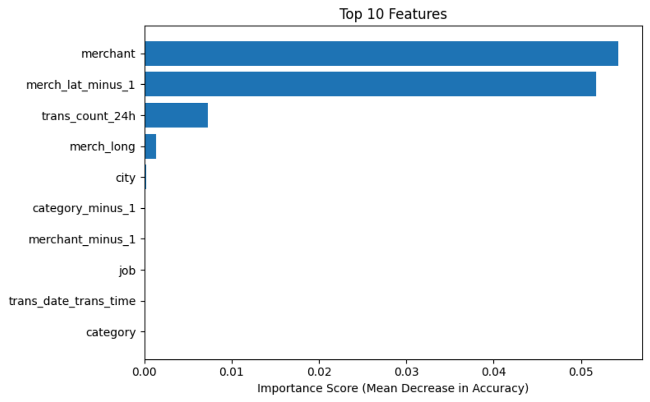


Figure C4. Decision Tree Feature Importance

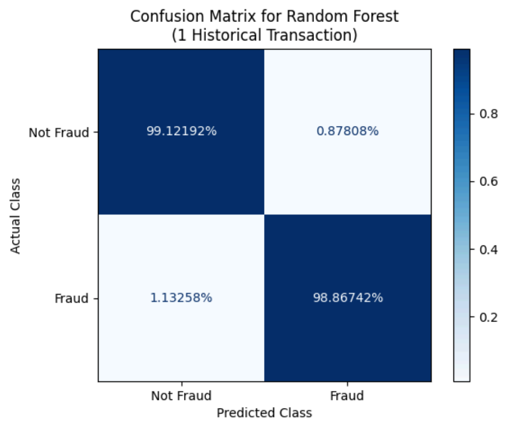


Figure C5. Random Forest Confusion Matrix

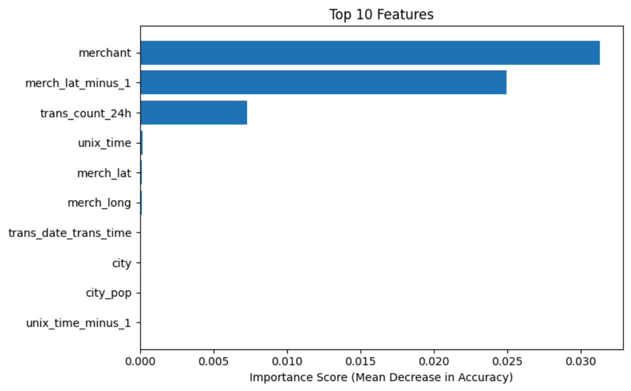


Figure C6. Random Forest Feature Importance

Models with 3 Historical Transaction

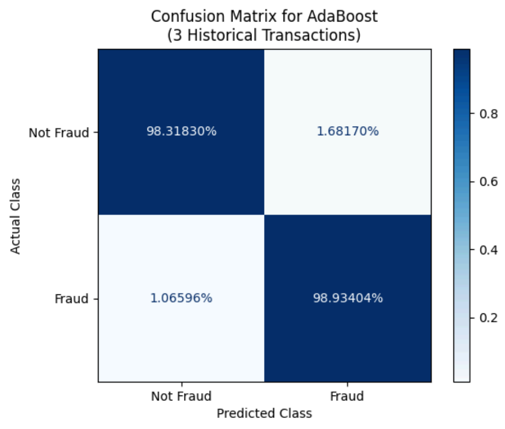


Figure C7. AdaBoost Confusion Matrix

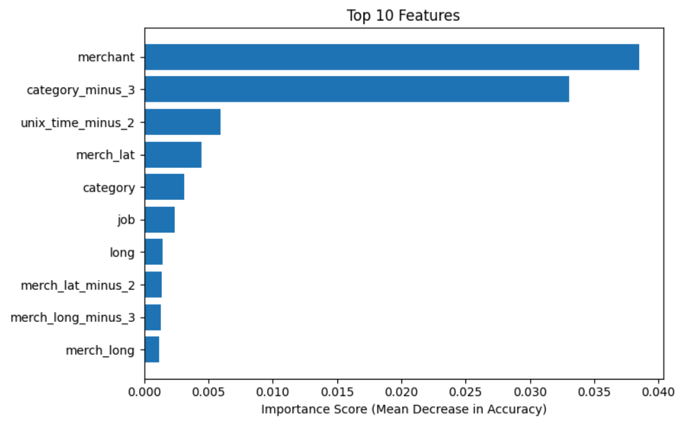


Figure C8. AdaBoost Feature Importance

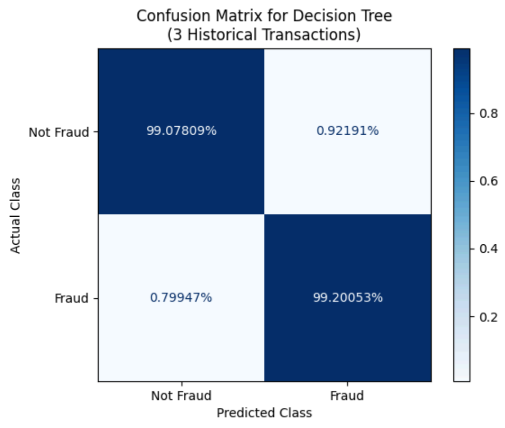


Figure C9. Decision Tree Confusion Matrix

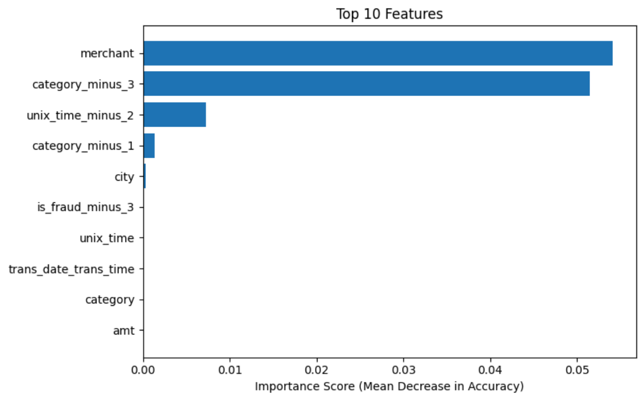


Figure C10. Decision Tree Feature Importance

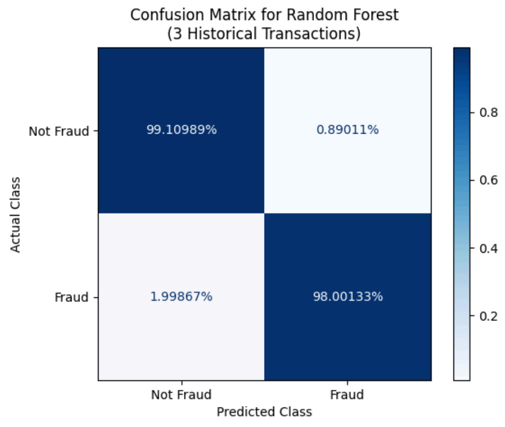


Figure C11. Random Forest Confusion Matrix

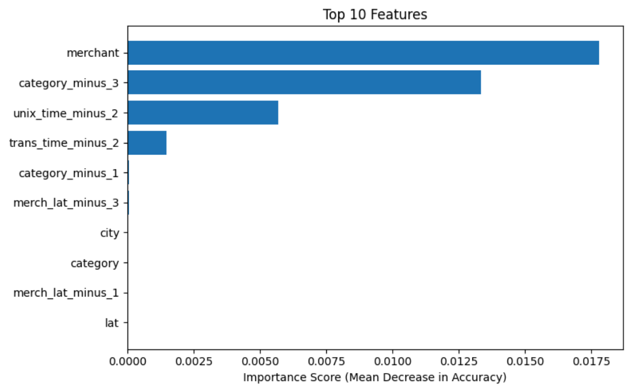


Figure C12. Random Forest Feature Importance

Models with 5 Historical Transactions

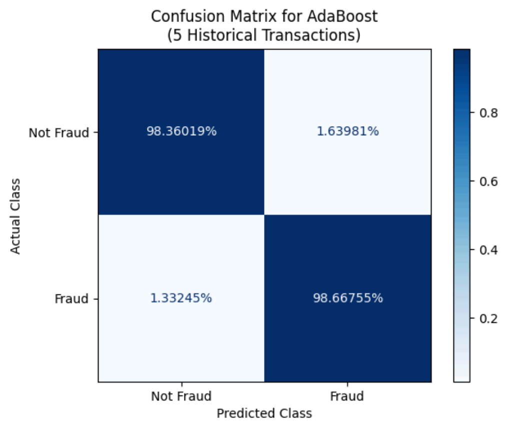


Figure C13. AdaBoost Confusion Matrix

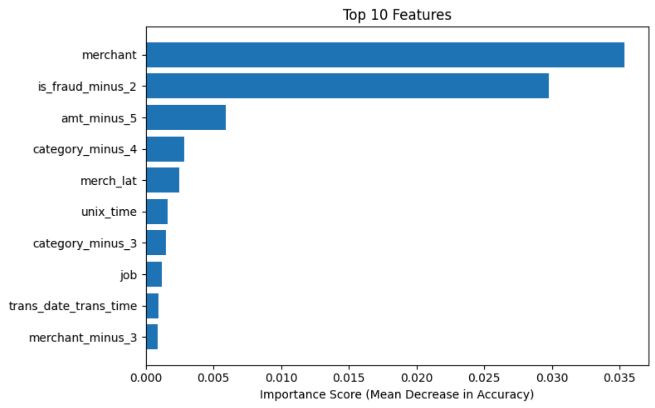


Figure C14. AdaBoost Feature Importance

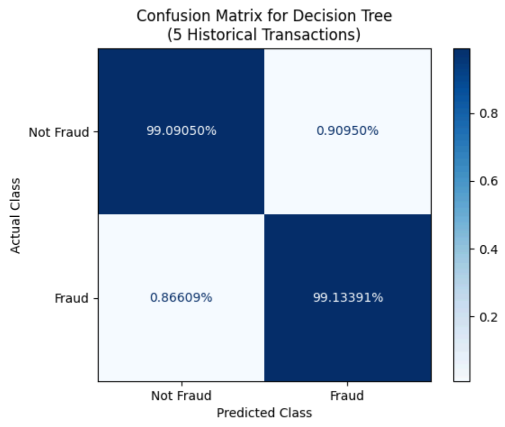


Figure C15. Decision Tree Confusion Matrix

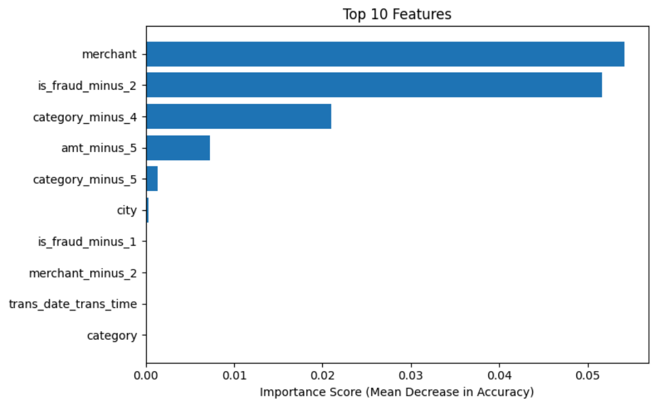


Figure C16. Decision Tree Feature Importance

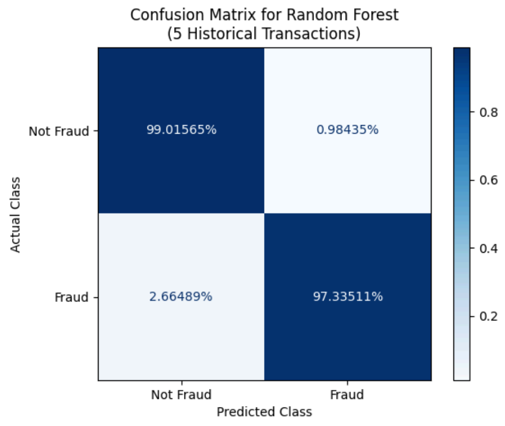


Figure C17. Random Forest Confusion Matrix

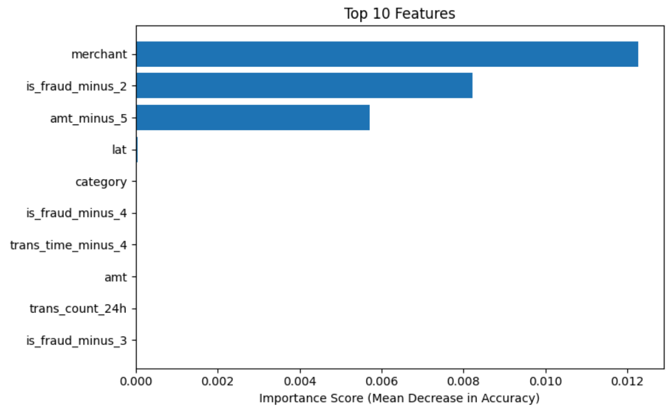


Figure C18. Random Forest Feature Importance

Models with 7 Historical Transactions

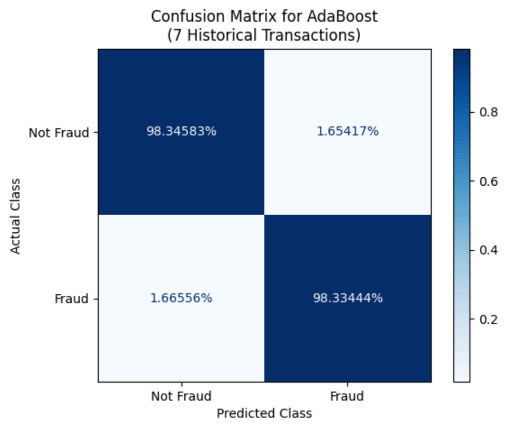


Figure C19. AdaBoost Confusion Matrix

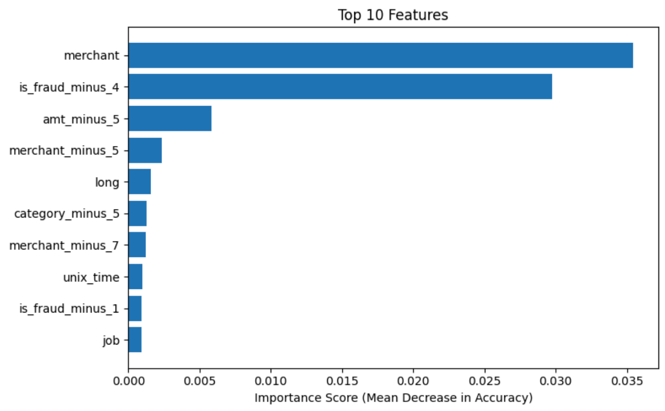


Figure C20. AdaBoost Feature Importance

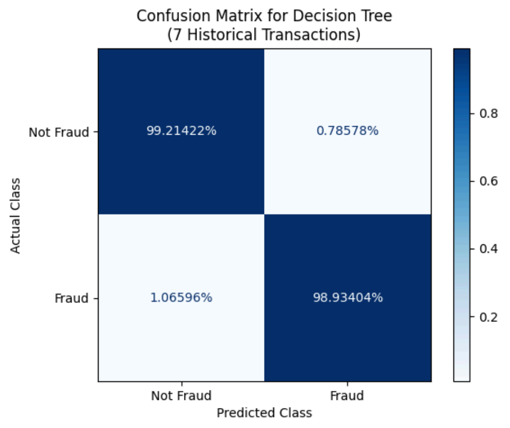


Figure C21. Decision Tree Confusion Matrix

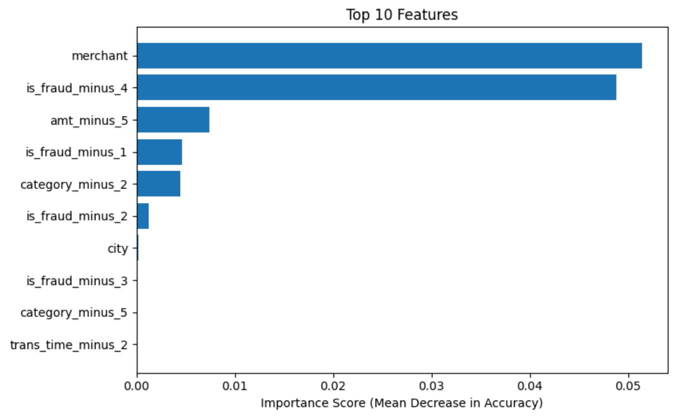


Figure C22. Decision Tree Feature Importance

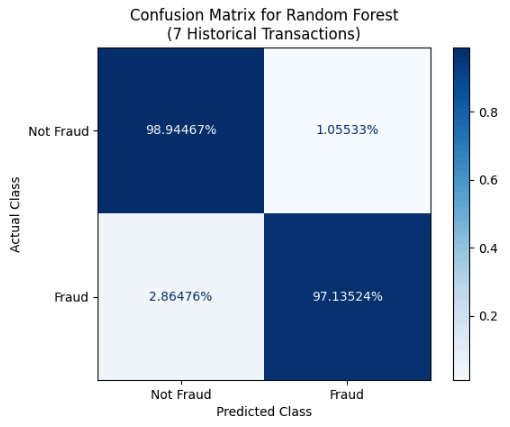


Figure C23. Random Forest Confusion Matrix

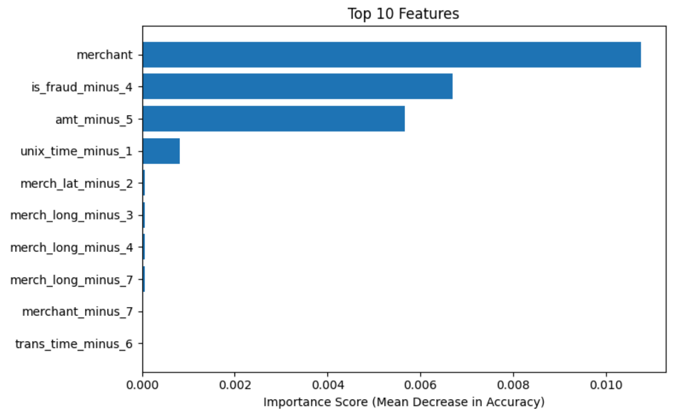


Figure C24. Random Forest Feature Importance

References:

Afriyie, J. K., Tawiah, K., Pels, W. A., Addai-Henne, S., Dwamena, H. A., Owiredu, E. O., Ayeh, S. A., & Eshun, J. (2023). A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions. *Decision Analytics Journal*, *6*, 100163. https://doi.org/10.1016/j.dajour.2023.100163

Choudhury, N. R. (2024, July 30). *Credit Card Fraud Data*. Kaggle. https://www.kaggle.com/datasets/neharoychoudhury/credit-card-fraud-data

Chung, J., & Lee, K. (2023). Credit Card Fraud Detection: An improved strategy for high recall using KNN, LDA, and linear regression. *Sensors*, *23*(18), 7788. https://doi.org/10.3390/s23187788

Farhang Ghahfarokhi, A., Mansouri, T., Sadeghi Moghaddam, M. R., Bahrambeik, N., Yavari, R., & Fani Sani, M. (2021). Credit card fraud detection using asexual reproduction optimization. *Kybernetes*, *51*(9), 2852–2876. https://doi.org/10.1108/k-04-2021-0324

Government of Canada, D. of J. (2022, August 17). *A typology of profit-driven crimes*. 3. Detailed Analysis of Selected Cases. https://www.justice.gc.ca/eng/rp-pr/csj-sjc/crime/rr02_3/p3.html

Hayat, K., & Magnier, B. (2025). Data leakage and deceptive performance: A critical examination of credit card fraud detection methodologies. Mathematics, 13(16), 2563. https://doi.org/10.3390/math13162563

Globaldee. (2023, September 3). *Nearmiss: A powerful undersampling technique for imbalanced data*. The Content Farm Blog. https://thecontentfarm.net/nearmiss-an-undersampling-technique-for-imbalanced-data/

Imbalanced Learn. (n.d.). *2. over-sampling#*. 2. Over-sampling - Version 0.14.0. https://imbalanced-learn.org/stable/over_sampling.html#smote-adasyn

Lemaitre, G., & Aridas, C. (n.d.-a). *Nearmiss*. NearMiss - Version 0.14.0. https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.NearMiss.html

Lemaitre, G., & Aridas, C. (n.d.). Random Oversampler. RandomOverSampler - Version 0.14.0. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html

Lemaitre, G., & Aridas, C. (n.d.-c). *Randomundersampler#*. RandomUnderSampler - Version 0.14.0. https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html

Makki, S., Assaghir, Z., Taher, Y., Haque, R., Hacid, M.-S., & Zeineddine, H. (2019). An experimental study with imbalanced classification approaches for credit card fraud detection. *IEEE Access*, *7*, 93010–93022. https://doi.org/10.1109/access.2019.2927266

Maklin, C. (2022, May 14). *Synthetic minority over-sampling technique (smote)*. Medium. https://medium.com/@corymaklin/synthetic-minority-over-sampling-technique-smote-7d419696b88c

Marazqah Btoush, E. A., Zhou, X., Gururajan, R., Chan, K. C., Genrich, R., & Sankaran, P. (2023). A systematic review of literature on credit card cyber fraud detection using machine and Deep Learning. *PeerJ Computer Science*, *9*. https://doi.org/10.7717/peerj-cs.1278

Rai, A. K., & Dwivedi, R. K. (2020). Fraud detection in credit card data using unsupervised machine learning based scheme. *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 421–426. https://doi.org/10.1109/icesc48915.2020.9155615

Shenoy, K. (2020, August 5). *Credit Card Transactions Fraud Detection Dataset*. Kaggle. https://www.kaggle.com/datasets/kartik2112/fraud-detection

Stripe. (2025, January 23). *Fraud detection using machine learning: What to know*. Stripe. https://stripe.com/en-ca/resources/more/how-machine-learning-works-for-payment-fraud-detection-and-prevention

Scikit-learn developer. (2025). sklearn.preprocessing – scikit-learn 1.7.2 documentation. Retrieved October 24, 2025, from https://scikit-learn.org/stable/api/sklearn.preprocessing.html?utm_source=chatgpt.com

Nguyen, T., Khadka, R., Phan, N., Yazidi, A., Halvorsen, P., & Riegler, M. A. (2022). Combining datasets to increase the number of samples and improve model fitting (arXiv preprint arXiv:2210.05165). https://arxiv.org/pdf/2210.05165.pdf

Chauhan, Y. S. (2023). *RANDOMIZEDSEARCHCV: A powerful hyperparameter tuning technique 💪💯*. Kaggle. https://www.kaggle.com/discussions/getting-started/467822

Singh, H. (2025, May 1). *10 techniques to solve imbalanced classes in machine learning (updated 2025)*. Analytics Vidhya.

https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/

Scikit-learn developer. (2025). RandomizedSearchCV. Scikit-learn 1.7.2 Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html#sklearn.model_selection.RandomizedSearchCV

Scikit-learn developer. (2025). GridSearchCV. Scikit-learn 1.7.2 Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#

Tang, T. (2023, April 25). *Class imbalance strategies-A visual guide with code*. Medium. https://medium.com/data-science/class-imbalance-strategies-a-visual-guide-with-code-8bc8fae71e1a

Kashyap, P. (2024, December 2). Understanding precision, recall, and F1 score metrics. Medium. https://medium.com/@piyushkashyap045/understanding-precision-recall-and-f1-score-metrics-ea219b908093

GeeksforGeeks. (2025, July 23). *F1 score in Machine Learning*. https://www.geeksforgeeks.org/machine-learning/f1-score-in-machine-learning/

GeeksforGeeks. (2025, May 30). *Understanding the Confusion Matrix in Machine Learning*. https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/

GeeksforGeeks. (2025, August 2). *Precision and Recall in Machine Learning*. https://www.geeksforgeeks.org/machine-learning/precision-and-recall-in-machine-learning/