A black background with a black square

Description automatically generated with medium confidence

~ ST1508 Practical AI ~

~ **SP-buy Fraud Detection** ~

DAAA/FT/2A/23

|  |  |  |
| --- | --- | --- |
| Name | Admin No | Role |
| Goh Pin Pin Isaac | P2317623 | Scrum Master | Data Translator |
| Adam Bin Roslan | P2317425 | Data Engineer |
| Choy Jee Hung Caleb | P2341475 | Data Scientist |
| Thor Hui Qin | P2309219 | Software Developer |

Introduction

|  |
| --- |
| **This project focuses on building a fraud detection model for SP Buys, an e-commerce company while navigating a heavily imbalanced dataset. Through advanced data processing, model optimization, and experiment tracking with MLflow, we aim to develop a high-recall model that accurately detects fraudulent transactions. Dask was used for efficient large-scale processing, ensuring scalability. The final model will be deployed as a graphical user interface (GUI) for the operations team to monitor and prevent fraud in real time.** |

Table of Contents

[1. Objectives 3](#_Toc190397355)

[2. Advanced Data Processing 3](#_Toc190397356)

[i. Further Exploratory Data Analysis 3](#_Toc190397357)

[ii. Outlier Detection Techniques 4](#_Toc190397358)

[iii. Imbalanced Classes Techniques 5](#_Toc190397359)

[3. Model Development Process 6](#_Toc190397360)

[i. Creation of Pipeline 6](#_Toc190397361)

[ii. Model Selection 7](#_Toc190397362)

[iii. Dask Client 7](#_Toc190397363)

[iv. Tracking and Monitoring ML Models using MLFlow 8](#_Toc190397364)

[v. Remarks on Model Performance 8](#_Toc190397365)

[vi. Model Optimization 9](#_Toc190397366)

[vii. Final Model Comparison 10](#_Toc190397367)

[viii. Feature Importance 12](#_Toc190397368)

[4. User Interface - Tkinter 13](#_Toc190397369)

[I. Wireframes 13](#_Toc190397370)

[II. First Draft 13](#_Toc190397371)

[III. Second Draft (Final) 14](#_Toc190397372)

[IV. Error Handling 14](#_Toc190397373)

[V. Deployment 15](#_Toc190397374)

[5. Conclusion 15](#_Toc190397375)

[6. Roles and Contributions 15](#_Toc190397376)

[7. Appendix 16](#_Toc190397377)

[i. Dataset 16](#_Toc190397378)

# Objectives

Our team has established two key objectives for model development to maintain a clear focus and ensure a robust process:

1. Optimize Accuracy and Balance Errors – We aim to achieve high accuracy while minimizing both false positives and false negatives as much as possible, with a focus on maximizing the F1 Score to ensure a well-balanced model.
2. Prioritize Minimizing False Negatives – Our model is designed to prioritize minimizing false negatives, particularly in cases where misclassifying a fraudulent order as legitimate could lead to significant financial or security risks. In this case, we emphasize Recall to reduce the chances of overlooking actual fraud.

**A black text on a white background

AI-generated content may be incorrect.**

Figure 1. Formular for *recall*

# Advanced Data Processing

To begin our analysis, we first merged three key datasets (customers\_df, orders\_df, and order\_labels\_df) to create a unified dataset, which we named "SP-Buy-Dataset". This consolidated dataset serves as the foundation for our analysis, combining essential customer, transaction, and order classification information to provide a holistic view of the data. Once the datasets were merged, we performed feature engineering to enhance the dataset for more meaningful analysis. One crucial step involved grouping different payment methods into seven distinct categories, ensuring a more structured and interpretable representation of the available payment options. This categorization allows us to analyze payment behaviors and their relationship with fraudulent activities more effectively. We then did Exploratory Data Analysis, plotting Correlation Matrix and PairPlot to get insights which we might have missed from our Cleaned Dataset in CA1.

### Further Exploratory Data Analysis

|  |  |
| --- | --- |
| Figure 2.1: Heat Map of Columns | The correlation values between num\_orders\_last\_50days, num\_cancelled\_orders\_last\_50days, and num\_refund\_orders\_last\_50days [0.40,0.92,0.45][0.40, 0.92, 0.45] indicate a strong mathematical relationship between order frequency and cancellation/refund patterns. This suggests that customers with more orders tend to have a higher likelihood of cancellations and refunds.  Additionally, there is a strong correlation [0.81][0.81] between num\_orders\_last\_50days and total\_payment\_last\_50days, reinforcing the connection between order volume and total payment value. The relationship between order\_value and num\_items\_ordered [0.42][0.42] shows that larger orders typically contain more items.  Apart from these key relationships, most other correlations are relatively weak, suggesting that variables such as num\_associated\_customers and refund\_value do not have a strong direct impact on other features. |
| **Figure 2.2: Pairplot of Numerical Features with Fraud Indicator** | The pair plot highlights right-skewed distributions in transaction features, with most values low but some significantly high. Strong correlations exist between order frequency and total payment, as well as order size and value, indicating higher spending with frequent transactions. Refund values show little correlation with other metrics.  Fraudulent transactions cluster around high order values, large payments, and frequent recent orders, suggesting these as fraud indicators. Some fraud cases also appear in refund metrics, hinting at refund abuse. However, fraud remains a minority among total transactions. |
| **Figure 2.3: Pairplot of Numerical Features with Mobile Verified** | The pair plot shows right-skewed transaction distributions, with most users making low-value purchases while a few spend significantly more. Strong correlations exist between order frequency and total payment, as well as order size and value, indicating higher spending with more orders. Refund values remain independent of other metrics.  Mobile-verified users (orange) dominate, while unverified users (blue) are fewer and cluster in lower transaction volumes. High-value transactions are mostly from verified users, suggesting a link between verification, trust, and spending. Outliers in order value and payments may indicate fraud or large-scale purchases. |
| **Figure 2.4: Pairplot of Numerical Features with Country Code** | The pair plot reveals distinct transaction patterns across different countries, with most numerical features exhibiting right-skewed distributions. This indicates that while the majority of transactions are of lower value, a few high-value outliers exist. A strong positive correlation is observed between the number of orders and total payments, as well as between order size and order value, suggesting that customers who place more orders tend to spend more. Refund values, however, do not show strong correlations with other features, indicating that refunds occur independently of transaction frequency or size. The country-wise distribution highlights noticeable clustering, suggesting regional differences in spending behavior, with some countries exhibiting higher transaction values than others. Additionally, the presence of extreme outliers, especially in total payments and order value, points to the possibility of bulk purchases or unusual spending patterns. |
| **Figure 2.5: Pairplot of Numerical Features with Payment Method** | The pair plot reveals that most transaction-related features are right-skewed, with a few high-value outliers. A strong positive correlation exists between order frequency and total payment, as well as between order size and value, while refunds appear independent of other variables. Different payment methods show distinct clustering patterns, with PayPal, credit cards, and invoices having broad distributions. Some methods are associated with higher-value transactions, suggesting customer preferences for larger purchases. Overall, the plot highlights transaction behaviors, customer spending patterns, and potential areas for further analysis. |

The projected pair plots revealed intriguing shapes and clusters, indicating notable trends and correlations within the transaction data. Different payment methods and country codes displayed distinct spending behaviors, while mobile-verified users tended to make higher-value transactions. Fraudulent transactions, represented by orange data points, clustered around high order values and frequent purchases, suggesting that fraud detection models could benefit from focusing on these patterns. Additionally, strong correlations between order frequency, total payments, and order values suggest that certain variables play a crucial role in transaction behavior. Notably, refund values appeared largely independent of other metrics, hinting that refunds may not be a strong fraud indicator. Outliers in order values and total payments suggest potential fraud or bulk purchases, requiring careful handling to improve model accuracy. By identifying and possibly removing these extreme values, models can better generalize and detect genuine spending patterns while reducing the risk of overfitting. Furthermore, ensuring balanced training data could enhance predictive accuracy, making fraud detection and customer behavior analysis more effective. Overall, the pair plots revealed meaningful relationships within the dataset, emphasizing key spending patterns and risk factors. Addressing outliers and refining model inputs could lead to improved classification accuracy, helping to distinguish normal transactions from potentially fraudulent ones.

### Outlier Detection Techniques

Outliers in a dataset are data points that significantly deviate from the majority of the observations. These uncommon values can distort statistical analysis, affect model performance, and violate key assumptions in data modeling. Outliers can arise due to various reasons, such as measurement errors, data corruption, or rare but valid occurrences in real-world scenarios. If not handled properly, outliers can lead to biased model training, misleading insights, and poor predictive performance. To effectively manage outliers, we employed three distinct anomaly detection techniques, Local Outlier Factor (LOF), Isolation Forest (IF) and One-Class SVM. Each of these methods detects anomalies using a different approach, leading to variations in the identified outliers. A visualization of the detected outliers is presented below, highlighting the differences in detection patterns among these techniques.

A graph of a data set

AI-generated content may be incorrect.

Figure 2.6. Anomaly Detection Visualization of Outliers

From the results, the above three methods are Isolation Forest, One-Class SVM, and Cluster-based LOF. Among these, Isolation Forest is considered the best choice for large-scale fraud detection datasets due to its high scalability and efficiency, making it suitable for big data, it’s ability to handle high-dimensional datasets effectively and lastly, its approach of detecting anomalies based on their separation from the normal data distribution. The fact that it does not require labeled data for training is also valuable. Therefore, we will use Isolation Forest as our preferred outlier detection method in this scenario.

### Imbalanced Classes Techniques

Machine learning models typically perform best when both target classes are evenly distributed. However, in our dataset, we have an imbalanced class distribution, with a 5:1 ratio of 'is 'Is\_Fraud' to 'Not\_Fraud' (as shown in Fig. 1). This imbalance poses a challenge since models tend to favor the majority class ('Is\_Fraud'), leading to poor predictive performance on the minority class ('Dangerous Trips'). If left unaddressed, this issue can result in biased predictions and overfitting to the majority class. To mitigate this problem, we applied various data-level, algorithm-level, and hybrid methods to balance the dataset and improve model performance.

A graph with a red and blue bar

AI-generated content may be incorrect.

Figure 2.7. Imbalanced Classes Visualizations

To address the class imbalance in our dataset, we applied a combination of Data-Level, Algorithm-Level, and Hybrid Methods to improve model performance. Data-Level Methods modify the dataset to balance class distribution. Oversampling techniques, such as SMOTE and ADASYN, generate synthetic minority class samples, while Random Oversampling duplicates existing samples. Undersampling techniques, including Random Undersampling and Tomek Links, reduce the majority class to refine decision boundaries and remove noise. Algorithm-Level Methods adjust the model to better handle imbalanced data. Cost-Sensitive Learning assigns higher misclassification costs to the minority class to improve detection. Additionally, ensemble techniques like XGBoost with Scale-Pos-Weight adjust training weights to enhance minority class recognition. Class Prior Modification further refines decision thresholds to favor the minority class. Hybrid Methods combine multiple approaches for improved performance. SMOTEENN, which merges SMOTE and Edited Nearest Neighbors (ENN), provides both oversampling and data cleaning, ensuring better generalization. By applying these techniques, we effectively balanced the dataset, reducing bias and improving accuracy in detecting fraudulent transactions ('Is\_Fraud').

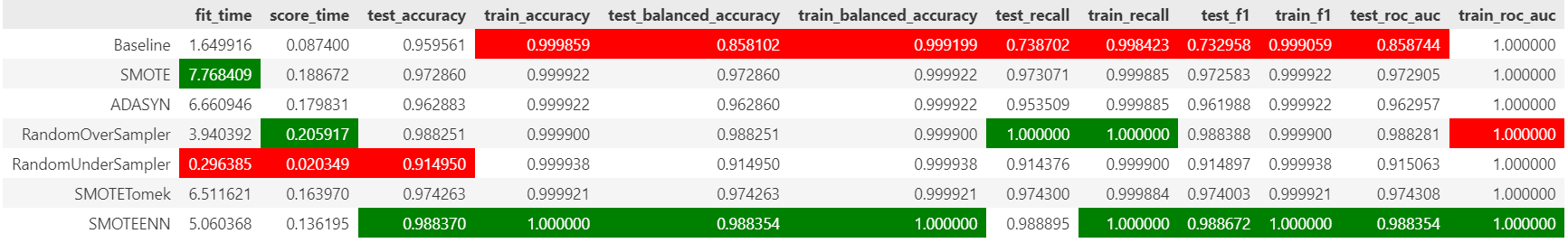


Figure 2.8: Comparison of Class Imbalance Methods

Among all techniques, SMOTEENN demonstrated the best overall performance, achieving perfect training accuracy (1) and recall (1), the highest test balanced accuracy (0.988354), and excellent ROC AUC scores (train: 1, test: 0.988354), making it the most reliable choice for our model. In terms of efficiency, RandomUnderSampler was the fastest (fit time: 0.296385s) but had lower test accuracy (0.914950) and test recall (0.914376), making it less effective. RandomOverSampler achieved perfect recall (1) and F1 score (0.999900) for training, but this suggests potential overfitting. SMOTE and SMOTETomek provided strong, balanced results, with test balanced accuracy scores of 0.972860 and 0.974263, respectively, and high ROC AUC scores (0.972905 and 0.974308, respectively). However, SMOTE had the longest processing time (7.768409s), while SMOTEENN maintained a strong balance with a reasonable processing time (5.060368s). Considering all factors, we selected SMOTEENN for training our final model, as it provides the most balanced performance across all metrics, generalizes well, achieves top scores in key evaluation criteria, and maintains a reasonable processing time. This ensures that our model is both accurate and efficient in detecting fraudulent transactions while avoiding biases toward the majority class.

# Model Development Process

### Creation of Pipeline

A computer code with text

AI-generated content may be incorrect.

Figure 3.1. Preprocessing Pipeline

Building a pipeline helps streamline the data preprocessing process by ensuring consistency and automating repetitive tasks. In our pipeline, we used a ColumnTransformer to handle different feature types efficiently. StandardScaler was applied to numerical features to normalize the data, preventing model bias, while OneHotEncoder (with drop='first') was used to convert categorical features into numerical format for compatibility with machine learning models. This structured approach ensures that all features are processed appropriately before training, improving model performance and reliability.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 3.2. Model Evaluation Pipeline

Building a model evaluation pipeline helps streamline the process of comparing multiple machine learning models by ensuring consistency and automating repetitive evaluation tasks. In our pipeline, we defined a list of classifiers, including Decision Tree, Random Forest, Logistic Regression, Gaussian Naïve Bayes, and Gradient Boosting, allowing us to systematically test and compare different models. To ensure comprehensive evaluation, we incorporated multiple scoring metrics, such as accuracy, balanced accuracy, recall, F1 score, and ROC AUC, which provide insights into overall model performance, class balance handling, and predictive capabilities. The evaluation was performed using the evaluate\_models() function, which trains each model on the resampled dataset (X\_resampled, y\_resampled) and assesses its performance across all defined metrics. The results were stored in two formats: styled results, presented in a formatted table for easy interpretation, and raw results, containing the numerical performance metrics. By structuring our model evaluation process into a pipeline, we ensure consistent performance comparison across multiple models, making it easier to select the best-performing algorithm based on objective evaluation criteria.

### Model Selection

In our model selection process, we trained and evaluated five distinct binary classification models: Decision Tree, Random Forest, Logistic Regression, Gaussian Naive Bayes, and Gradient Boosting. We used metrics such as accuracy, balanced accuracy, recall, F1 score, and ROC AUC to evaluate the models. While all metrics were considered, we prioritized recall as our primary evaluation criterion, as it measures the model's ability to correctly identify fraudulent orders, which is critical for fraud detection. A high recall ensures that fewer fraudulent cases are missed, even if it results in a higher number of false positives.

After training each model, we logged the average cross-validation results, including key metrics and model parameters, into MLflow for tracking and comparison. Additionally, we generated graphical analyses, such as learning curves, to provide deeper insights into model performance and facilitate easier interpretation of results. This approach allowed us to identify the Random Forest model as the best-performing model, achieving the highest recall and overall balanced performance.

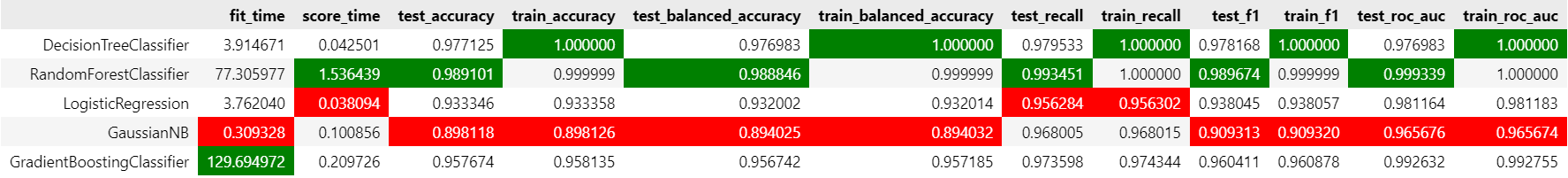


Figure 3.3. Model Selection in Notebook

### Dask Client

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 3.4. Dask Client set up

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 3.5. Dask dataframe details

In our work, we will utilized Dask to improve memory efficiency and scalability, particularly for handling large datasets. By converting the Pandas DataFrame into a Dask DataFrame partitioned into smaller chunks (npartitions=2), Dask processes data in parallel and out-of-core, reducing memory usage by loading only relevant partitions during operations. After splitting the dataset into training and testing sets, Dask DataFrames further partition these sets (npartitions=4), avoiding the need to load the entire dataset into memory at once. This helps to enhance scalability for large-scale fraud detection tasks.

### Tracking and Monitoring ML Models using MLFlow

For this assignment, we will use MLFlow, an open-source library that includes numerous features such as Experiment Tracking, Model Registry, Model Deployment, and so on. Experiment Tacking will be used largely to track the results of our models and compare which has the best result. After each model has been trained and assessed, it is recorded as a run, and its model attributes and outcomes are preserved. This offers a centralized spot where we can easily glance and compare, saving us time while analyzing each model.

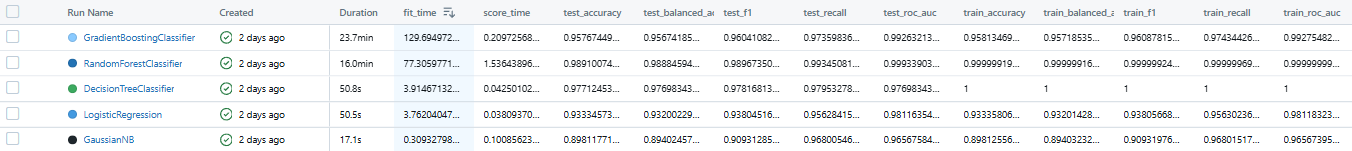


Figure 3.4. Model Selection experiments in MLFlow

As seen above, we are able to log each experiment into MLFlow. Next, we will interpret and give remarks for each model’s evaluation scores and graphical analysis

### Remarks on Model Performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Fit Time (s) | Test Recall | Test F1 Score | Remarks |
| DecisionTreeClassifier | 3.9 | 0.979 | 0.978 | Fast fitting time. Train scores are 1, indicating perfect fit and potential overfitting. Needs regularization if deploying to production |
| RandomForestClassifier | 77.3 | 0.993 | 0.989 | Excellent performance across all metrics. Possibly the best balance and a good choice of model. Train scores are near-perfect, watch out for overfitting. |
| LogisticRegression | 3.8 | 0.956 | 0.938 | Lowest performance compared to other models. |
| GaussianNB | 0.3 | 0.968 | 0.909 | Fast fitting time, but poor performance. |
| GradientBoostingClassifier | 129.7 | 0.973 | 0.960 | Good performance, similar to Random Forest but takes considerably more time to fit. Check if time is a deal-breaker or if you're ok with a longer turn around for potentially better results |

|  |  |
| --- | --- |
| Graphical Analysis (Learning Plot) | |
| A graph of a training classifier  AI-generated content may be incorrect. | A graph with green and red dots  AI-generated content may be incorrect. |

We determine our best two models to be the DecisionTreeClassifier and RandomForestClassifier. While both models demonstrate strong performance on the test set, they exhibit signs of overfitting, as indicated by perfect (1.0) train accuracy scores for DecisionTreeClassifier and near-perfect train accuracy(0.999) for RandomForestClassifier, compared to test accuracies of 0.977 and 0.989 respectively. Despite this, they achieved high F1 scores on the test set compared to the other models. To address the overfitting and further improve their performance, we will now attempt to find the best set of hyperparameters. Hyperparameter optimization can be done through methods such as GridSearchCV or RandomizedSearchCV. For this assignment, we will make use of RandomizedSearchCV from Dask\_ML to save time and speed up the process.

### Model Optimization

After selecting the best models, *Random Forest Classifier* and *Decision Tree Classifier*. We note that tree-based models are very prone to overfitting. We will need to optimize the models to resolve the issue with the tree-based models.

Due to computational and time complexity of the RandomForestClassifer, we will use a faster but less accurate approach to find the best parameters. We will be using the RandomizedSearchCV from dask.ml\_selection to run through the grid of parameters to see which model parameters will give the best f1 score. Note that the RandomizedSearchCV will randomly choose a few parameters to test and choose the best score from the test. The issue with this is that the parameters that are selected will not guarantee the best score.

|  |  |  |
| --- | --- | --- |
| Model | Hyperparameter Tuning | Reason |
| Random Forest Classifier | max\_depth | The DecisionTreeClassifier and RandomForestClassifier have perfect/near-perfect training accuracy, indicating potential overfitting. Limiting max\_depth prevents the trees from growing too complex and memorizing noise in the training data. |
| min\_samples\_split and min\_samples\_leaf | These parameters control the stopping criteria for tree growth. Tuning them helps balance bias and variance. Given the perfect training accuracy, increasing these values will act as regularization, preventing the model from creating overly specific branches and leaf nodes that only fit the training data. |
| n\_estimators | While more trees in the RandomForestClassifier generally improve performance, there's a point of diminishing returns. Tuning n\_estimators prevents the model from becoming unnecessarily complex, especially given the long fit time (77.3 seconds) already observed. This is important to find an appropriate balance between performance and computational cost. |
| max\_features | This parameter controls the number of features considered for each split in the RandomForestClassifier. With a dataset that presumably has many columns, tuning max\_features allows for careful selection of the most relevant features. Randomly selecting a subset of features helps to reduce correlation between trees and prevent overfitting, ultimately improving model performance. |
| bootstrap | Bootstrapping helps Random Forest generalize better by training each tree on a different subset of the data. Ensuring bootstrap is set to True adds randomness and reduces variance, making the model more stable and less prone to overfitting, especially given the high training accuracy observed. If set to false, the data set will be trained on the original set of data. |
| Decision Tree Classifier | max\_depth | Limiting the depth of the decision tree helps prevent overfitting. Without a depth limit, the tree may grow too complex and memorize the training data, leading to high variance and poor generalization to unseen data. By tuning max\_depth, we control the trade-off between model complexity and performance. |
| min\_samples\_split | This parameter determines the minimum number of samples required to split an internal node. A lower value allows the tree to grow deeper, potentially leading to overfitting, while a higher value restricts tree growth, promoting regularization and reducing variance. Given a range from 2 to 20, tuning min\_samples\_split helps balance bias and variance. |
| min\_samples\_leaf | This defines the minimum number of samples required to be in a leaf node. Smaller values allow the model to create more fine-grained splits, increasing complexity, while larger values encourage simpler, more generalized trees. Tuning this parameter helps prevent overfitting by ensuring that leaf nodes have sufficient data to generalize patterns rather than capturing noise. |
| max\_features | This controls the number of features considered for splitting at each node. Using sqrt or log2 reduces correlation between splits, improving generalization. When set to None, all features are considered, which can lead to overfitting in high-dimensional datasets. Tuning max\_features helps optimize model performance by balancing feature selection and model complexity. |
| criterion | This determines the function used to measure the quality of a split. gini focuses on maximizing class purity, while entropy considers information gain. Tuning this parameter helps select the best metric for the given dataset, ensuring optimal splits that lead to better model performance. |



Finally, we will need to select the best model out of the models that have been hypertuned. We see that there are notable differences between the models. We have compared and found out that RandomForestClassifier is indeed better at predicting fraudulent orders but DecisionTreeClassifier is not far behind. However, as the model that will be deployed on a GUI, speed of the model is very important. If the model inference time is very long, the system administrator will be stuck processing a lot of data. Therefore even though RandomForestClassifier is a strong model, we need to consider the inference time and therefore we will be selecting the more lightweight model of DecisionTreeClassifier.

A blue squares with numbers and labels

AI-generated content may be incorrect.

Figure 3.5. Confusion Matrix of Hyper tuned DecisionTreeClassifier

Although our hyper tuned models have very high recall, because of our imbalanced test set, we can see that its has severe class imbalance, predicting all items as non-fraud. This could be because the matric we use is recall.

### Final Model Comparison

|  |  |  |
| --- | --- | --- |
| Raw Data Model | Isolation Forest Model | Final Model |
|  |  |  |
| Accuacy: 0.9281  Recall: 0.6835 | Accuracy: 0.9282  Recall: 0.6639 | Accuracy: 0.802  Recall: 0.92 |
| **Raw Data**   * This confusion matrix represents the performance of our Decision Tree model trained on the unprocessed, imbalanced dataset. * The model struggles with distinguishing fraudulent transactions (1) from non-fraudulent ones (0). * False positives (upper-right) and false negatives (lower-left) indicate misclassifications. * The imbalance in class distribution likely skews predictions toward the majority class (non-fraud), reducing recall for fraud detection.   **Key Observations:**   * **True Negatives (TN):** 386,117 (correctly predicted non-fraud cases) * **False Positives (FP):** 16,748 (non-fraud cases misclassified as fraud) * **False Negatives (FN):** 15,783 (fraud cases misclassified as non-fraud) * **True Positives (TP):** 34,088 (correctly predicted fraud cases)   The model struggles with fraud detection due to the inherent class imbalance, leading to a higher number of false negatives. | **Isolation Forest**   * The Isolation Forest method was applied for outlier detection before training the Decision Tree model. * Compared to the raw data, the Decision Tree model’s performance has improved slightly in detecting fraudulent transactions. * False negatives (missed fraud cases) have decreased slightly, suggesting better fraud identification. * However, some false positives (legitimate transactions misclassified as fraud) still persist.   **Key Observations:**   * **TN:** 369,195 (slightly lower due to anomaly removal) * **FP:** 15,701 (similar to before, indicating slight reduction in misclassified non-fraud cases) * **FN:** 15,192 (a small decrease, showing better fraud detection) * **TP:** 30,012 (slightly lower, possibly due to some fraud cases being removed as anomalies)   The slight decrease in total fraud cases detected may be due to some fraudulent transactions being classified as anomalies and removed from the dataset. | **Final Model**   * The final model incorporated imbalance class and Isolation Forest  to handle class imbalance and improve fraud detection. * The increase in correctly classified fraudulent transactions (bottom-right cell) suggests improved recall and precision. * The significant increase in true positive (fraud) cases indicates that the model generalizes better and identifies more fraud attempts. * False positives have increased, but this is expected as the model prioritizes detecting fraud while maintaining a high recall.   **Key Observations:**   * **TN:** 475,922 (significantly higher due to more non-fraud samples) * **FP:** 1,063,660 (a massive increase, showing that many non-fraud cases are misclassified) * **FN:** 56,036 (higher than before, indicating fraud detection worsened) * **TP:** 124,780 (increase in correctly detected fraud cases)   This suggests that while the model improved in detecting fraud (higher TP), it also misclassified many non-fraud cases (high FP), leading to a trade-off between precision and recall. |

### Feature Importance

A graph with blue lines

AI-generated content may be incorrect.

Figure 3.6 Feature Importance with Final Model

*Refunds are the biggest fraud indicator*

* refund\_value and total\_payment\_last\_50days are the top features.
* Fraud is highly linked to refund behavior.

*Country & Payment Methods Matter*

* country\_code and payment\_group show fraud varies by region and payment type.

*Order Behavior Plays a Role*

* num\_orders\_last\_50days, num\_items\_ordered, and order\_value affect fraud risk.
* num\_associated\_customers has low impact.

*Collect Type is Not Important*

* Minimal influence, suggesting order collection method isn't a key fraud factor.

# User Interface - Tkinter

## Wireframes

A screenshot of a computer

AI-generated content may be incorrect.

Image 1: Wireframes made with Figma

Creating wireframes helps us plan our user interface, providing a clear vision of how user input and results should be structured. We intend to have two tabs:

1. **Batch Mode** – Allows bulk input, with results displayed in a table.
2. **Real-Time Mode** – Allows manual input, displaying results one at a time.

This approach ensures an organized and efficient user experience.

## First Draft

|  |  |  |
| --- | --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect. | A screenshot of a computer  AI-generated content may be incorrect. | A screenshot of a computer  AI-generated content may be incorrect. |
| Bulk Insert Prediction (Batch Mode) | Table of results for Batch Mode | Manual Insert prediction (Real-time Mode) + result |

We created our first draft to closely match our wireframe. However, since Tkinter does not support gradients, we opted for a blue background instead. Instead of displaying the real-time mode result in a separate window, we chose to show it in a message box for simplicity. This approach is more efficient, as we don’t need a large space just to indicate whether a transaction is fraudulent or not. We also added pagination for table page to cater to larger csv/xlsx file upload.

## Second Draft (Final)

|  |  |  |
| --- | --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect. |  | A screenshot of a computer  AI-generated content may be incorrect. |
| Bulk Insert Prediction (Batch Mode) | Table of results for Batch Mode | Manual Insert prediction (Real-time Mode) + result |

We refined our first draft and selected this as the final version. This time, we added error handling and included a title on the table page. We also removed the blue background, as we preferred a cleaner, plain design. Additionally, we changed the title from "Suspicious Order Detector" to "Fraudulent Order Detector" for a more professional tone. We also enlarged the input box for better usability and added “Export Results” button in the table page so SP-Buy staffs can save the results on their device.

## Error Handling

Error handling is an essential part of building robust applications that can manage user inputs effectively. It ensures that users are guided through correct data entry and prevents the program from failing due to invalid or improperly formatted inputs.

Here are the types of error handling we did:

|  |  |  |
| --- | --- | --- |
| A screenshot of a computer error  AI-generated content may be incorrect. | A screenshot of a computer error  AI-generated content may be incorrect. |  |
| Empty inputs | Wrong format for Customer ID | Wrong format for Order ID |
| A screenshot of a computer error  AI-generated content may be incorrect. | A screenshot of a computer error message  AI-generated content may be incorrect. | A screenshot of a computer error  AI-generated content may be incorrect. |
| Incorrect inputs (only can accept numerical for “number of cancelled orders last 50 days”, “number of refund orders last 50 days”, “number of orders last 50 days”, “num\_associated\_customers”, “num\_items\_ordered”, “Total payment last 50 days”, “order value”, refund value”) | Incorrect inputs (no float for “number of cancelled orders last 50 days”, “number of refund orders last 50 days”, “number of orders last 50 days”) | Incorrect inputs (no more than 2 decimal place for “total payment last 50 days”, “order value” and “refund value” as they are currency) |

## Deployment

Finally, we deploy the Python script as a standalone executable using PyInstaller. This process packages all the necessary dependencies into a single folder, which contains the executable file, making it easy for SP-Buy staffs to run the fraudulent detector without needing to set up the environment or install dependencies themselves.

These are the steps we took:

* 1. Run ```python -m pip install pyinstaller```in anaconda prompt.
  2. Then, run ```python -O -m PyInstaller --hidden-import=sklearn fraudDetector.py``` in anaconda prompt.
  3. Go to “Dist” folder in the same directory as fraudDetector.py
  4. Click on “fraudDetector” folder.
  5. Copy our model pickle file and paste it inside “fraudDetector” folder
  6. Double click fraudDetector.exe
  7. Command prompt popped up, waited for a while for the application to load.

# Conclusion

|  |
| --- |
| **To sum up, this task required the use of sophisticated pre-processing methods, such as the SMOTEENN to address class imbalance in the dataset and the Isolation Forest to identify and eliminate outliers. The top two models were then determined using the model selection procedure, and they were further improved by hyperparameter tuning. Because it performed the best, Decision Tree Classifier was ultimately chosen as the best model. We then deployed our model with Tkinter through PyInstaller. Now, SP-Buy can detect fraudulent orders with the help of our application.** |

# Roles and Contributions

|  |  |
| --- | --- |
| Member | Contributions |
| Goh Pin Pin Isaac | * Advance Pre-processing (Class Imbalance) * Scrum Deliverables * Model Optimization |
| Adam Bin Roslan | * Model Development Process   + Data Pre-Processing   + Model Selection |
| Choy Jee Hung Caleb | * Advance Pre-processing (Outlier Detection) * Background Research (EDA) * Model Optimization |
| Thor Hui Qin | * User Interface (Tkinter) * Deployment * Wireframes |

# Appendix

### Dataset

* **Customer Features** (2195916 rows, 9 columns)

|  |  |  |
| --- | --- | --- |
| **Column name** | **Definitions** | **Data Type** |
| customer\_id | Unique identifier of customers | object |
| mobile\_verified | Whether the customer have verified their mobile number or not | bool |
| num\_orders\_last\_50days | Number of orders the customer ordered last 50 days | int64 |
| num\_cancelled\_orders\_last\_50days | Number of orders the customer cancelled last 50 days | int64 |
| total\_payment\_last\_50days | Total amount a customer spent last 50 days | float64 |
| num\_associated\_customers | Number of accounts related to the customer | int64 |
| num\_refund\_orders\_last\_50days | Number of orders the customer refunded last 50 days | int64 |
| first\_order\_datetime | Date and time of the first order of a customer | object |
| country\_code | Two letters code of a country | object |

* **Order Features** (2414179 rows, 8 columns)

|  |  |  |
| --- | --- | --- |
| **Column name** | **Definitions** | **Data Type** |
| order\_id | Unique identifier of orders | object |
| collect\_type | Collection method: Pick-up or Delivery | object |
| payment\_method | The payment method the customer use to pay for their order | object |
| order\_value | The value of the order the customer ordered | float64 |
| num\_items\_ordered | Number of items the customer ordered | float64 |
| refund\_value | The value of refund of the order | float64 |
| order\_date | Date in which the customer places an order | object |
| country\_code | Two letters code of a country | object |

* **Order labels** (2414179 rows, 4 columns)

|  |  |  |
| --- | --- | --- |
| **Column name** | **Definitions** | **Data Type** |
| order\_id | Unique identifier of orders | object |
| customer\_id | Collection method: Pick-up or Delivery | object |
| is\_fraud | Whether the order is a fraud or not | int64 |
| country\_code | Two letters code of a country | object |