UNIVERSITY OF SUSSEX

School of Mathematics and Physical Science

Dissertation Report

Student Name: Gunveer Singh Sanjeev Thapar

Student ID: 269020

Course: MSc Human and Social Data Science

Project Title: Detecting fraudulent customer transactions

Supervisor Name: Dr Hsi Ming-Ho

Declaration of Originality

I hereby certify that this report is an output of my own work and fulfils the criteria for the MSc in Data Science at the University of Sussex. Except as specifically noted in the text, the research and information offered here were the result of my personal work and reflect my own original studies.

Signature:

Gunveer Thapar

Acknowledgments

I want to take this opportunity to offer my sincere thanks to the hardworking people who were essential to my completion of the University of Sussex's MSc in Human and Social Data Science programme.

First and foremost, I would want to express my sincere gratitude to **Dr Hsi Ming-Ho**, my distinguished supervisor, for their steadfast leadership, knowledgeable mentoring, and important assistance during the program's research phase. My study was much improved by the invaluable insights, helpful criticism, and breadth of information that [Supervisor's Name] offered. My path will always bear the imprint of their dedication to my academic and professional development.

I also want to thank **Dr Omar Lakkis** and **Dr Julie Weeds** whose outstanding leadership and dedication have greatly influenced the program's curriculum and general learning environment. I was able to gain a thorough knowledge of Data Science principles and approaches because **Dr Omar Lakkis** and **Dr Julie Weeds** created a demanding and intellectually interesting atmosphere. Their commitment to establishing an excellence-focused culture has been admirable.

I also want to express my gratitude to the whole teachers and staff of the Department of Engineering and Informatics for their commitment to providing a high-quality education and for developing a stimulating academic environment.

I would want to express my gratitude to my fellow classmates for their support, cooperation, and inspiration. My learning path has been improved by our shared experiences.

Finally, I want to thank my family and friends for their continuous encouragement and faith in my skills.

This MSc in Human and Social Data Science has been a life-changing experience, and it could not have been accomplished without the help and direction of these extraordinary people and the tools made available by the University of Sussex.

Table of Contents

1. Introduction

With a rise in online transactions, electronic payments, and mobile banking, the global financial environment has gotten more and more digitalized recently. Despite the convenience and efficiency that this technological breakthrough has brought, it has also led to an increase in financial fraud. Identity theft, credit card fraud, and money laundering are just a few examples of the fraudulent actions that financial institutions, companies, and consumers are all increasingly concerned about. It has never been more important to have efficient systems in place for detecting and preventing fraud.

Fraud is a type of illegal deception that is done by someone acting dishonestly and falsely. Unauthorised financial fraud losses for payment cards, remote banking, and checks totalled £844.8 million in 2018, up 16% from 2017, according to UK Finance (2019).

Nowadays, thanks to COVID-19 and the implementation of lockdowns all over the world, online transactions are even more popular. Prior to machine learning, businesses would employ a rule-based method to hunt for obvious and common signals to detect fraud. Numerous distinct fraud scenarios that are manually developed by a person are performed by pure rule-based algorithms. Rule-based scenarios need to be changed far too often in order to stay current with security measures nowadays. Furthermore, rule-based systems are ineffective with real-time data streaming, which is important for banks.

Machine learning, on the other hand, can be used to stream data in real-time, uncover hidden patterns in user conduct, and estimate the likelihood of fraudulent behaviour. Additionally, machine learning is faster than rule-based algorithms.

2. Aims and Objectives:

The objective of this research is to create precise machine learning models that can identify and prevent fraudulent transactions. In order to do the latter task, I will need to do the following:

- Obtain a financial dataset.
- Eliminate unnecessary data fields from the dataset.
- Locate appropriate metrics to assess the model's performance.
- Select the tested models that have performed the best and strengthen the weaker ones.

3. Definitions:

3.1. Unsupervised learning:

- Unsupervised learning is the term used to describe machine learning algorithms that use data sets without labelled outcomes or answers. Without assistance from preset target variables, these algorithms seek to identify hidden patterns or structures in the data.
- Unsupervised learning is used for a variety of tasks, including clustering (the grouping of related data points), dimensionality reduction (the decrease of the number of features while maintaining crucial information), and anomaly detection (the identification of rare and out-of-the-ordinary data points).

3.2. supervised learning:

- Algorithms used in supervised learning translate an input into an output based on a labelled dataset. The algorithm creates an inferred function that can make predictions on fresh, unseen data by learning from the data and the results that correspond to it.
- Usage: Tasks like classification (sorting data into preset classes or categories) and regression (forecasting continuous numerical values) frequently include the usage of supervised learning.

3.3. Hyperparameter:

- A hyperparameter is a parameter that regulates a machine learning algorithm's learning process. Hyperparameters are predetermined before to the training process and direct how the algorithm learns, in contrast to model parameters, which are learnt from the data.
- Use: Hyperparameters may take the form of numbers such as learning rates, the quantity of trees in a random forest, or the KNN's chosen distance metric. A crucial step in improving a machine learning model's performance is tuning its hyperparameters.

3.4. The grid search:

- In a predetermined search space, grid search is a hyperparameter optimisation strategy that thoroughly tests all potential combinations of hyperparameter values. The collection of hyperparameters with the best model performance is returned.
- Grid search may be performed when you want to find the optimal hyperparameter values for a model but are prepared to spend more time and money on your computer to do it. It thoroughly investigates the whole search area.

3.5. Random Search:

- Definition: Random search is a method for hyperparameter optimisation that randomly selects hyperparameter values from a distribution or range. It investigates a subset of combinations rather than doing an exhaustive search of all possible combinations.
- When you need to find good hyperparameter values but just have a few computing resources, you can use random search. It frequently identifies quality hyperparameters more quickly than grid search.

3.6. Accuracy Paradox:

- The accuracy paradox is a circumstance in which accuracy is an inappropriate performance indicator for a classification model. This happens, particularly in datasets that are unbalanced, when a model with more accuracy has lower predictive power.
- Explanation: When class distributions are unbalanced, accuracy can be deceptive since the model may frequently predict the majority class, leading to high accuracy but subpar performance in identifying the minority class. For a more thorough assessment of the model's performance under such circumstances, measures like accuracy, recall, F1-score, or area under the ROC curve (AUC-ROC) are chosen.

4. Paysim

4.1. The PaySim Simulator and Its Purpose:

- **Data privacy and GDPR:** Since the General Data Protection Regulations (GDPR) were implemented, worries about the use of private information for research have considerably increased. A European Union policy known as GDPR sets tight guidelines for the collection, processing, and use of personal data. This has made it more difficult to acquire publicly available datasets for research, especially in areas like fraud detection where sensitive financial information is frequently involved.
- **Solution PaySim**: The PaySim simulator was created to get around these problems and facilitate study into fraud detection. The synthetic or fake datasets produced by the PaySim simulator are designed to mirror the features of genuine financial data while excluding any personally identifiable or sensitive data. Researchers may use data that mimics actual financial transaction data thanks to this fake dataset production without breaking any data privacy laws.

4.2. The PaySim Dataset was made:

- **Sources of Data**: PaySim develops its dataset by taking cues from unpublished datasets and actual bank records. These private datasets are likely to have transaction records that have been aggregated, anonymized, or otherwise cleaned up to remove any personally identifying information (PII).
- **Mobile Transactions in Africa**: PaySim's dataset is largely concerned with replicating mobile transactions, a critical component of financial services in many locations, particularly in Africa. These transactions are a sample of actual financial transactions made through mobile money services over a given month.
- **Multinational Corporation Data:** PaySim used financial records from a multinational company that offers mobile banking services in more than 14 different nations to produce the dataset. These logs probably contain a wide variety of transaction patterns and behaviours from different geographical areas.

4.3. The PaySim Dataset's characteristics are as follows:

- **Data Structure:** There are 6,362,620 rows (individual data points or transactions) and 11 columns (attributes or characteristics) in the PaySim dataset. It's important to note that this dataset has been scaled down to one-fourth of its original size, which may have been done to make it easier to manage for study while maintaining the integrity of the underlying patterns.
- **Use in Fraud Detection Research:** The PaySim dataset may be used by researchers and data scientists to create and test machine learning and deep learning models for identifying fraudulent transactions. The dataset is a useful tool for researching alternative fraud detection strategies, evaluating the effectiveness of models, and developing the industry.
- **In conclusion,** the PaySim simulator resolves the issues brought about by data privacy laws like GDPR by producing artificial financial datasets that closely mimic real-world data. This guarantees the protection of sensitive data while enabling researchers to work with data appropriate for fraud detection studies. The PaySim dataset, which was created using real bank records, offers a wealth of data for investigating and enhancing fraud detection techniques in a responsible and privacy-conscious manner.

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1	0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0	0
...	...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	1	0
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	1	0
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	1	0
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	1	0
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	1	0

6362620 rows × 11 columns

Figure 1: Display the Dataset

The very imbalanced nature of this dataset necessitates its modification for the models to produce accurate predictions. Only 0.13% of transactions are fraudulent, whereas 99.87% of transactions are legitimate.

```
In [10]: # Calculate the total transactions, genuine transactions, and fraudulent transactions
         plus, minus = np.bincount(y_dim)
         final = minus + plus
         genral_percentage = (plus * 100 / final)
         fake_percentage = (minus * 100 / final)

         # Print the results
         print("Total transactions:\t\t", final)
         print("Genral transactions:\t\t", plus, "(", round(genral_percentage, 1), "%)")
         print("Fraudulent transactions:\t",minus, "(", round(fake_percentage, 1), "%)")
```

```
Total transactions:             2770409
Genral transactions:            2762196 ( 99.7 %)
Fraudulent transactions:        8213 ( 0.3 %)
```

Figure 2: total number of fraudulent and general transaction

4.4. step

In the actual world, a step represents a unit of time. One step in this dataset equals one hour of time. A 30-day simulation consists of 743 phases in total.

4.4.1. Representation of Time:

The discrete time units in this dataset are referred to as "steps." One hour of actual time is equal to each stage. This indicates that the dataset mimics transactions and events as they take place across time, with each step representing a particular hour in the simulation.

4.4.2. The Simulation's Phases:

Different time spans or periods inside the simulation are referred to as "phases" in this context. Each phase covers a certain number of steps (hours) and corresponds to a separate period throughout the simulation. The total number of steps in the simulation will be 30 x 24 = 720, for instance, if each step lasts an hour and the simulation lasts a total of 30 days.

4.4.3. Number of phases overall:

Dataset states that a 30-day simulation comprises a total of 743 stages. In spite of the fact that there are 720 hours in a 30-day period (30 days x 24 hours/day), this indicates that the simulation has been broken into 743 phases, which raises the possibility that certain phases may not have an exact duration of one hour.

4.4.4. Phases' Importance:

A more precise depiction of time is made possible by segmenting the simulation into phases, which can be advantageous for a variety of applications. It makes it possible to track and analyse transactions and events over a range of time intervals, which aids researchers and analysts in understanding how activities change over time in the simulated environment. Additionally, it might make it easier to analyse the dataset's temporal trends, patterns, and anomalies.

4.4.5- Analysis Implications:

Depending on the unique research questions or analysis objectives, researchers or analysts working with this dataset can investigate the data at the level of individual steps (one-hour intervals) or aggregate it based on phases. An organised method for examining the temporal characteristics of the data, such as the timing of transactions, the frequency of occurrences through time, or the length of certain activities, is provided by the idea of phases.

```
In [6]: minimum_step = data['step'].min()
        maximum_step = data['step'].max()
        print("Steps from {} to {}.".format(minimum_step, maximum_step))

        Steps from 1 to 743.
```

Figure 3: Total number of steps

4.5. type

```
In [5]: unique_transaction= data['type'].unique()
        unique_transaction

Out[5]: array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
              dtype=object)
```

Figure 4: shows several values in the "type" column

Five different types of transactions exist:

4.5.1. PAYMENT Transaction:

- A payment transaction is a financial action in which a customer uses their mobile money service to pay a store or merchant. For buying products and services, it's a typical sort of transaction. As the funds are spent for the payment, the amount in the sender's mobile money account lowers. The recipient's account, usually owned by the store or merchant, grows when the money is credited to it.
- An illustration would be making a payment transaction at the grocery store using your mobile money account. The balance on your account decreases, while the supermarket's account balance increases.

4.5.2. TRANSFER Transactions:

- A TRANSFER transaction entails moving money from one user to another over a platform for mobile money services. Person-to-person (P2P) money transactions are made easier by it.
- Effects: Money is sent from the mobile money account of the sender to that of the receiver.
- Using a mobile money app to send money to a friend is an example of a TRANSFER transaction. Your friend's account balance improves while yours declines.

4.5.3. CASH_OUT:

- Customers withdraw money from their mobile money accounts at a merchant's location through a CASH_OUT transaction. In essence, the business serves as an ATM.
- Effects: As the customer receives actual cash from the business, the balance in their mobile money account declines.
- A CASH_OUT transaction is one in which you withdraw $100 from your mobile money account when you go into a retailer that offers the service. Your mobile money account is depleted by $100.

4.5.4. CASH_IN:

- A CASH_IN transaction has customers giving over actual cash to a retailer in order to deposit it into their mobile money wallets. The retailer deposits money into the customer's mobile money account.
- Effects: As the deposited money is credited to the customer's account, their mobile money balance rises.
- A CASH_IN transaction is one in which you hand over $50 to a cashier at a store, and they deposit the money into your mobile money account. $50 is added to your mobile money balance.

4.5.5. DEBIT:

- When a customer sends money from their mobile money service to a bank account, a DEBIT transaction takes place. It is comparable to a CASH_OUT transaction in that money is transferred to a connected bank account.
- Effects: Money is moved from the mobile money account to the associated bank account, lowering the amount there.
- A DEBIT transaction is one in which $500 is sent from your mobile money account to your associated bank account. Your bank account balance rises by $500 but your mobile money balance drops by $500.

4.6. nameOrig:

- In the Paysim dataset, "NameOrig" refers to the identification associated with the account that requested a financial transaction. This characteristic is essential for tracing the sources of transactions and is necessary for fraud detection, audit trails, and financial analysis.
- It helps track how money is transferred and transactions are initiated in a simulated mobile money environment like Paysim, allowing researchers and analysts to learn more about transaction patterns and behaviours. To secure the sensitive data linked to these account names, data privacy procedures are routinely employed.

4.7. oldbalanceOrg:

- Financial statistics refer to the original sum of money in the sender's account prior to a transaction as the "sender's starting balance". It acts as a starting point for determining how a transaction affects the sender's account balance.
- This characteristic is essential for determining the financial ramifications of transactions, keeping track of account activity, and spotting errors or anomalies in the flow of money across accounts.

4.8. newbalanceOrig:

- The updated account balance of the sender following a financial transaction is referred to as "newbalanceOrig" in the Paysim dataset. It stands for the money that is still in the sender's account after the transaction.
- This characteristic is crucial for monitoring changes in the sender's financial situation, confirming the correctness of transactions, and spotting anomalies. In the simulated mobile money environment, "newbalanceOrig" offers critical insights into the effects of transactions on sender accounts, assisting in financial analysis and fraud detection.

4.9. nameDest:

- The term "nameDest" in the Paysim dataset refers to the name or unique identification of the recipient account that received and processed a financial transaction. In the simulated mobile money ecosystem, this feature is essential for tracking the destination of cash and comprehending the flow of transactions.
- It is essential for doing financial analyses, keeping track of audit trails, and identifying any abnormalities or trends in the way transactions are processed and money is allocated to various accounts.

4.10. oldbalanceDest:

- The original balance or total amount of funds in the recipient's account prior to a financial transaction is indicated by the term "oldbalanceDest" in the Paysim dataset. This characteristic indicates the receivers.
- financial starting place and acts as a baseline reference. In the simulated mobile money environment, it is crucial for recording changes in the recipient's account balance brought on by incoming transactions, enabling analysis of the financial implications, and guaranteeing transaction correctness.

4.11. newbalanceDest:

- The updated account balance of the receiver following a financial transaction is represented by "newbalanceDest" in the Paysim dataset. It shows how much money is still in the recipient's account after the transaction.
- In the simulated mobile money environment, this property is essential for keeping track of the recipient's financial situation, confirming the correctness of transactions, and spotting any inconsistencies or discrepancies in the recipient's financial data. It offers information on how transactions impact recipient accounts' balances.

4.12. isFraud:

In a binary classification scenario, each transaction in your dataset is given a label indicating whether it is fraudulent or not. The sentence "Whether the transaction was determined to be fraudulent (1) or not (0)" describes this situation. Let's get into further depth about this:

4.12.1. Fraudulent transactions (1):

- A "1" normally denotes transactions that have been identified as fraudulent. Based on some criterion or algorithm used for fraud detection, these transactions are those that are seen as being suspicious, unauthorised, or somehow connected to fraudulent operations.
- A "1" basically indicates that the system or model classifying the data has determined that the transaction is fraudulent and may call for more inquiry or action.

4.12.2- Transactions that are not fraudulent (0):

- In contrast, a "0" is used to denote transactions that have been deemed or identified as legitimate. Based on the analytical criteria, these transactions are thought to be authentic and do not show any indications of fraud.
- A "0" means the system or model has evaluated the transaction and has not discovered any indications of fraudulent behaviour, indicating that it is a typical, authorised transaction.

4.13. isFlaggedFraud:

A binary indicator that acts as a flag to identify possibly fraudulent transactions is the "isFlaggedFraud" column.

1. **Binomial Indicator:** "isFlaggedFraud" is a binary indicator variable (1/0). For each transaction in the dataset, it accepts one of two values:
- 1: The transaction has been marked as potentially fraudulent, according to this value.
- 0: This number denotes that the transaction has not received a fraud risk flag.

2. **Importance:** The "isFlaggedFraud" column is intended to draw attention to transactions that meet criteria or rules established by the system or model analysing the data that are thought to be very suspicious or potentially fraudulent.

5. Literature Survey:

if a transaction is legitimate or fraudulent. No important information is lost in this manner. LSTM addresses the vanishing gradient issue that Vanilla RNNs experience. In previous layers' RNN weights, the vanishing gradient issue manifests itself. The gradient of the loss function approaches 0 when activation functions are used in layers. This makes training the network more challenging. However, because the LSTM models need to do additional calculations to decide whether the data is saved or deleted, they are more difficult to implement and longer to train. But LSTMs outperform conventional machine learning techniques in terms of accuracy.

Naive Bays, KNN, and Logistic Regression's accuracy, sensitivity, specificity, and precision are all compared by Awoyemi, Adetunmbi, and Oluwadare (2017). The performance of Logistic Regression was the worst of the three methods. The PaySim dataset is not used in the research, which instead focuses on accuracy, sensitivity, specificity, and precision. Determining which algorithms may be utilised as a starting point to detect fraud is a good place to start.

For the PaySim dataset, Bandyopadhyay and Dutta (2020) advise using a Stacked-RNN with 12 layers to completely minimise fraud detection. The RNN's accuracy, F1-score, and Mean Squared Error are all 99.87%, 0.99, and 0.01 respectively. Although Bandyopadhyay and Dutta utilise accuracy, F1-score, and mean squared error to assess the model's performance, this research is a decent study for an RNN design. Due to the accuracy paradox that exists in fraud detection, accuracy is a bad statistic. F1 score also largely relies on True Positive values, which are always the highest number since legitimate transactions always outnumber fraudulent ones, which are fewer than 1% of the total transactions in the dataset. Additionally, regression models rather than classification model typically employ mean squared error. MSE does not penalise the model for misclassification as much as it should.

Kaur (2019) suggests utilising the methods Logistic Regression, Naive Bayes, Random Forest, and XGBoost to identify if a transaction is legitimate or fraudulent. They compare the differences between test-train split and K-Folds, two alternative data segmentation techniques. Kaur contrasts the various algorithms according to their research's accuracy. The maximum accuracy, 99.95% (train-test split) and 96.46% (K-fold), is achieved by XGBoost. To find the ideal parameters for a certain algorithm, Kaur also advises utilising grid search and random search. The PaySim dataset is used in the research, which is a suitable place to start because it has the ideal parameters for the recommended algorithms. The right measurements, nevertheless, such the proportion of missed fraud or the false negative rate, are not given enough attention.

Different metrics are compared by Baesens, Höppner, and Verdonck (2021) based on the dataset itself and when the SMOTE approach is applied to it to generate fictitious fraudulent transactions. They also suggest specific custom metrics, such as the amount of money a fraudulent transaction would cost (if fraud were to occur), the amount of money a legitimate transaction that is mistakenly believed to be fraudulent would cost, and various weights for true negative, false negative, true positive, and false positive values. This study is a

useful place to start when thinking about the right measures to use when comparing various models. They have more precise information, such as entrance charges and genuine data, but they do not utilise the PaySim dataset.

Decision trees, random forests, and autoencoders are suggested by Nordling (2020) as tools for identifying if a transaction is legitimate or fraudulent. The Synthetic Minority OverSampling

Technique is suggested to generate additional transactions of the less dominant class until both classes have an equal number of data points in order to address the imbalanced dataset. The research, however, used a different dataset and measures accuracy, recall, and AUROC.

The idea put out by Pambudi, Hidayah, and Fauziati is to use SVM to assess if a transaction is real or fraudulent. They use the measures Precision, Recall, F1-score, and Area Under Precision-Recall Curve (AUPRC) to compare the performance of several kernels with various gammas and C values. SVM computation takes a very lengthy time, especially for large datasets with a lot of columns. Additionally, the research does not describe how the data was cleaned, which means that a model with identical parameters may have radically different results.

Several studies recommend using XGBoost, Random Forest, Recurrent Neural Network, SVM Logistic Regression, and Nave Bayes using metrics like Accuracy, Recall, and AUROC. Studies using unique datasets provide unique measures that more accurately determine if a model is doing well or poorly.

6. Methodology:

6.1. Accuracy

In binary classification problems, accuracy is a popular indicator to assess how well a model predicts outcomes accurately. The proportion of accurately predicted cases—true positives and true negatives—to the total number of instances in the dataset is calculated.

- **The Imbalanced Data Problem**: Class imbalance presents a problem in situations like fraud detection, when fraudulent transactions only make up a small percentage of the whole dataset (e.g., less than 1% of all transactions). Only a small percentage of transactions are fraudulent (the minority class), whereas most transactions are authentic.
- **The paradox of accuracy:** The accuracy paradox happens when a classification model, even if it is only moderately effective or not built to handle unbalanced data, manages to score highly in terms of accuracy. When the majority class dominates the dataset, this contradiction is very common.
- An accuracy paradox in the context of fraud detection could refer to a model's performance in detecting the minority class (fraudulent transactions) while accurately predicting the majority class (legal transactions) most of the time.
- **Accuracy Limitations with Unbalanced Datasets:** The accuracy statistic by itself is not a trustworthy gauge of how well a model performs in unbalanced datasets.
- A high accuracy score in these datasets might be deceiving since the model may be categorising everything as belonging to the majority class (non-fraudulent) in order to reach the high accuracy rate.
- As a result, the important minority class (fraud), which is frequently the main goal in fraud detection, is not identified by the model.
- Relevant Applications: The goal in fraud detection and related sectors should be to reduce false negatives (fraudulent transactions that are overlooked) while maintaining a manageable level of false positives (normal transactions that are mistakenly reported as fraudulent).

- As a result, it's crucial to consider a variety of measures in order to fully evaluate a model's performance and come to wise conclusions regarding its efficacy.

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

6.2. Confusion Matrix

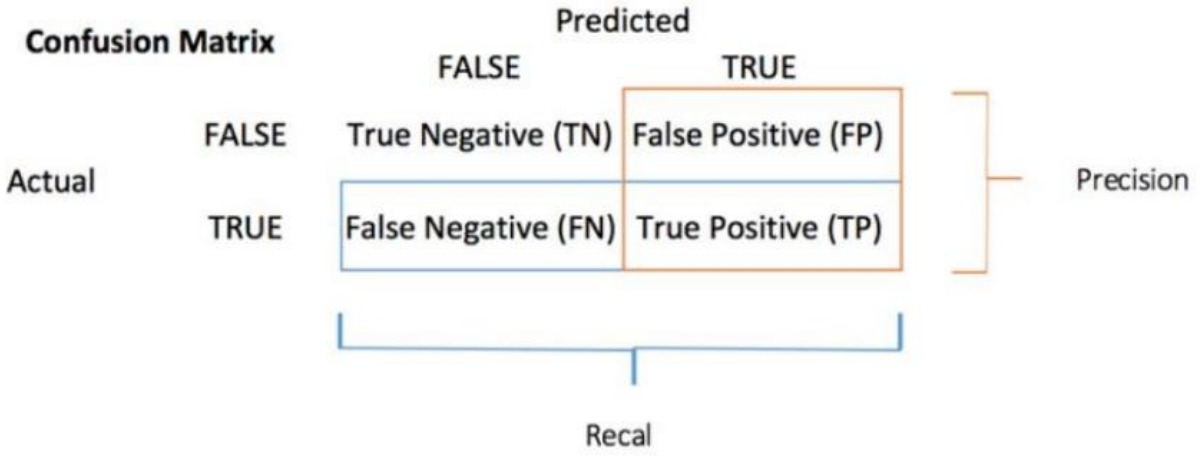

Figure 6: show what precision and recall are created in Confusion Matrix example

- In binary classification, **a confusion matrix** is a technique used to assess a model's performance. It divides predictions into four groups: Instances that were accurately identified as positive (fraudulent transactions) are known as true positives (TP).
- **True Positive (TP)**: A true positive (TP) is the number of transactions that a categorization model or system correctly classified as genuine or non-fraudulent. The occurrences when the model successfully predicted that a transaction is not fraudulent and is indeed a valid transaction are represented by TP. Example: If a model properly classifies 150 out of 200 transactions as being free of fraud, then TP equals 150.
- **False Positive (FP)**: An inaccurate classification of authorised or legitimate transactions as fraudulent by a model is known as a false positive (FP). FP shows how many general transactions have been mistakenly classified as fraudulent. When clients' general transactions are refused as a result of a model's inaccurate forecast, this sort of mistake can be annoying. Example: If a model mistakenly classifies 20 out of 200 genuine transactions as fraudulent, then FP = 20.
- **False Negative (FN)** - The number of fraudulent transactions that a model incorrectly identified as genuine is known as a false negative (FN). FN is an important indicator of fraud since it shows that fraud has happened but gone unnoticed. It draws attention to situations in which the model was unable to detect fraud. For instance, FN = 10 if a model misses 10 fraudulent transactions out of 100.
- **True Negative (TN)** - The number of fraudulent transactions that a model correctly detected as such is known as true negatives (TN). TN shows that the model successfully identified these transactions as fraudulent, helping to effectively identify fraud. TN = 80, for instance, if a model properly detects 80 out of 100 fraudulent transactions.

- **Economic Effects of False Negatives:** Banks suffer financial losses when fraudulent transactions are mistaken for legitimate ones (FN). Vulnerabilities are successfully used by criminals, and the institution takes the loss.
- **Less Worry About False Positives:** False Positives (FP), which occur when valid transactions are mistakenly classified as fraudulent, are typically seen as less significant from a financial standpoint.
- Banks can address false positives by getting in touch with customers to validate the validity of transactions or by asking customers to get in touch to check "suspicious" behaviour, albeit doing so might cause customers to be inconvenienced.
- **False Positives and False Negatives Must Be Balanced**: The ultimate objective of a fraud detection model is to successfully manage FP while increasing the proportion of fraud detection (minimising FN).
- The financial institution can minimise financial losses due to fraud by balancing these two variables, if it doesn't annoy real clients too much.

$$\text{Total Fraud} = \text{FN} + \text{TN}$$

$$\%\ Fraud = \frac{TN}{Total\ Fraud}$$

$$\%\ \text{Missed} = 1 - \%\ \text{Fraud}$$

6.3. The F1-Score is:

- A statistical indicator called the F1-score combines the accuracy and recall aspect of a model's performance. The model's precision describes how well it distinguishes between occurrences it predicts as positive and actual positive events (such as fraudulent transactions). The goal is to reduce false positives.
- Recall, sometimes referred to as sensitivity, measures how well a model can detect all real positive situations or, in the case of fraud detection, how well it can spot fraudulent transactions and reduce false negatives.
- **Using Precision and Recall Together:** The harmonic mean is used by the F1-score to balance recall and accuracy. A single score that considers both false positives and false negatives is provided by this harmonic mean.
- It simply measures how successfully the model distinguishes between positive and negative events while reducing the influence of extreme values (either accuracy or recall being extremely low).
- **Addressing Imbalance:** Accuracy can be deceptive in situations when there is a major class imbalance, such as in fraud detection if there are fewer fraudulent transactions than valid ones.
- Because most forecasts are accurate, a model that primarily predicts the majority class (non-fraudulent transactions) can nonetheless achieve high accuracy. However, it is likely that this model would struggle to identify fraudulent transactions.
- Contrary to accuracy, the F1-score emphasises the performance of the minority class. It is a superior metric for unbalanced datasets because it assigns equal weight to erroneous positives and false negatives.

- **Important factors in detecting fraud:** The main goal of fraud detection is to accurately identify fraudulent transactions (reducing false negatives) while limiting false alarms (false positives).
- The F1-score is particularly useful since it offers a fair evaluation of how well a model performs in reaching this objective.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

6.4. Area Under the ROC Curve (AUC):

- Based on their receiver operating characteristic (ROC) curves, binary classification models are evaluated for performance using the AUC score as a measure. It gauges how well the model can differentiate between positive and negative classes.
- Despite being a useful statistic, the AUC score might be less helpful in datasets that are severely skewed. The rationale is that it largely assesses how well the model can prioritise good instances over bad ones. This ranking can produce high AUC scores even for subpar models in an unbalanced dataset with few positive instances (fraudulent transactions).
- Interpretation: While a high AUC value may imply that the model performs well in ranking positive examples higher, it may not always correspond to the model's efficiency in identifying the few positive occurrences.

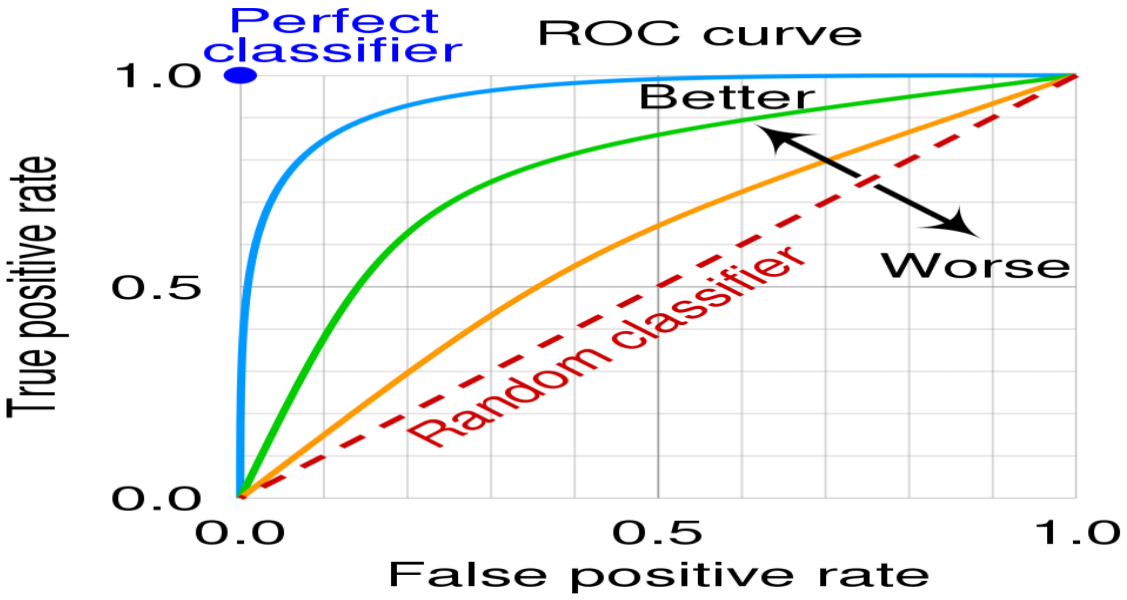

Figure 5 : AUC cuve Example

6.5. Imbalance Dataset:

- The effect of unbalanced data: When one class considerably outnumbers the other, the dataset is unbalanced. In terms of fraud detection, this indicates that the great majority of transactions are honest (majority class) and the minority class (extremely tiny number) of transactions are dishonest.
- Machine learning models are significantly hampered by unbalanced data. One of the main problems is that models can forecast the majority class (legal transactions) and yet score highly on popular measures like accuracy and AUC.
- Biased Models: A Problem Because they correctly forecast most variables for the plentiful class, models that primarily predict the majority class may appear to be accurate. But because the minority class (fraudulent transactions) is entirely ignored, there is a serious under-detection of fraud.
- Need for Specialised Techniques: Specialised approaches are necessary to overcome the imbalance difficulty in fraud detection and related situations:
- Resampling: - Oversampling the minority class entails adding more examples of the minority class to balance the distribution of the classes.
- These strategies help the model see more cases of the minority class, making it less biased towards the majority class. undersampling the majority class includes lowering the number of instances in the majority class to balance the distribution.
- Use of Alternative measurements: In datasets with imbalances, traditional measurements like accuracy can be deceptive. Alternative measures including the F1-score, precision-recall curves, and area under the precision-recall curve (AUC-PR) offer a more precise evaluation of a model's performance.
- Putting the Minority Class in Focus: Specialised methods are employed to make sure that the minority class is given enough consideration and is not ignored because of the imbalance in class.
- These methods allow the model to identify fraudulent transactions successfully by assigning equal weight to both classes.
- Evaluation of the model: The capacity to identify the minority class (fraud) should take precedence in the evaluation of models trained on unbalanced data.
- Measurements like as accuracy, recall, and the F1-score particularly evaluate the model's performance in relation to the minority class.

6.5.1. Random Undersampling:

A technique called random undersampling is used to balance unbalanced datasets like the PaySim dataset, which has a majority class of genuine transactions and a minority class of fraudulent transactions. Up until both classes have an equal amount of data points, the majority class's samples are removed at random. However, this procedure has important consequences:

- Data reduction: Most valid transactions are eliminated to achieve balance. In severe circumstances, this may lead to the deletion of 99.9% or more of the data from valid transactions.

- Information Loss: Eliminating the bulk of honest transactions results in the loss of important data and trends that can improve the accuracy with which machine learning models can identify fraud.
- Model Complexity: With fewer data points to draw upon, models may have trouble extrapolating and making reliable predictions. To get knowledge from the few data provided, they must put in more effort.
- Risk of Bias: By emphasising the minority class, random undersampling causes bias to emerge. The performance of the model may be impacted, and overfitting may result.

6.6. Oversampling

6.6.1. Random Oversampling

1. **Random Oversampling:** In random oversampling, samples from the minority class (fraudulent transactions) are copied several times until the number of instances from that class equals the number from the majority class (legal transactions). This balances the distribution of the classes and enhances the efficiency with which algorithms can learn from both classes.
2. **Overfitting** is a concern with random oversampling since it can occur. A machine learning model overfits when it begins to memorise the training data rather than drawing generalisations from it. In the context of detecting fraud:

- **Memorization:** The model is more likely to encounter the same false cases more than once when it is oversampled. As a result, it could begin memorising these cases rather than learning the fundamental deception patterns. It basically "learns" the training data by heart.
- **Ineffective Generalisation:** Because the model has memorised the training data, it can have trouble when faced with brand-new, untried data. In real-world conditions that could differ from the training set, it won't be able to generalise effectively enough to detect fraud.

Therefore, even while random oversampling might make the model perform admirably on the training data (seeming to provide perfect results), it can be problematic when dealing with fresh, unforeseen data since the model hasn't learnt the fundamental patterns of fraud. To balance class imbalance reduction by oversampling with preventing overfitting, it is crucial. Cross-validation, various resampling techniques (such as Synthetic Minority Over-sampling Technique, or SMOTE), or picking the right evaluation metrics may all be used to overcome these problems and create models that are more applicable to real-world situations.

6.6.2. Synthetic Minority Oversampling Technique

Synthetic Minority Over-sampling Technique, often known as SMOTE, is a potent machine learning method that is especially useful in situations when the datasets are unbalanced. By creating synthetic representations of the minority class, it seeks to solve the issue of class inequality.

1- SMOTE's Operation:

Another strategy for dealing with unbalanced datasets is SMOTE. Although current data isn't copied, it balances out the minority class. By combining new minority data points with existing minority occurrences, this approach creates new minority data points.

- SMOTE concentrates on the minority class, which often has fewer instances than the dominant class and is easy to identify.
- SMOTE chooses a beginning point for each data point in the minority class as a basis for producing new data.
- Selecting Neighbours: SMOTE next selects a predetermined number of minorities who are the specified data point's closest neighbours. The number of neighbours is a parameter that the user defines.

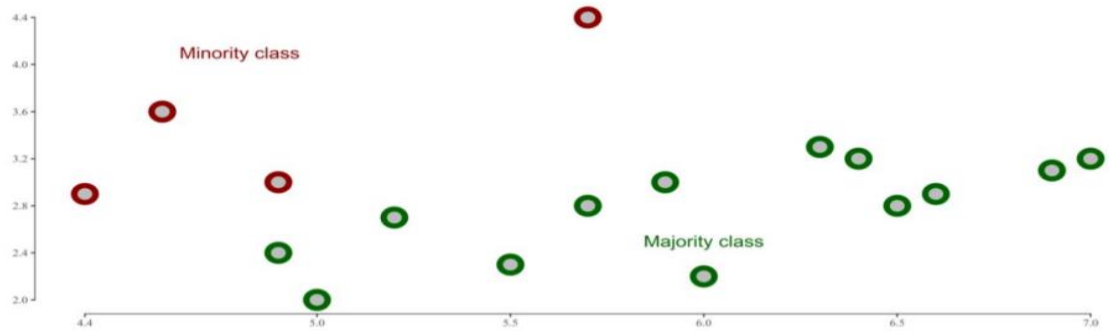

Figure 7: Class imbalance example

The algorithm first plots each data point. The closest neighbours of the data point and the feature vector are then determined. The two vectors' differences are determined, and then their differences are multiplied by a chance number between 0 and 1. Finally, by adding the random integer to the feature vector, the new point on the line segment is found. The subsequent procedures are carried out for the PaySim dataset until the ratio of real transactions to fraudulent transactions is equal.

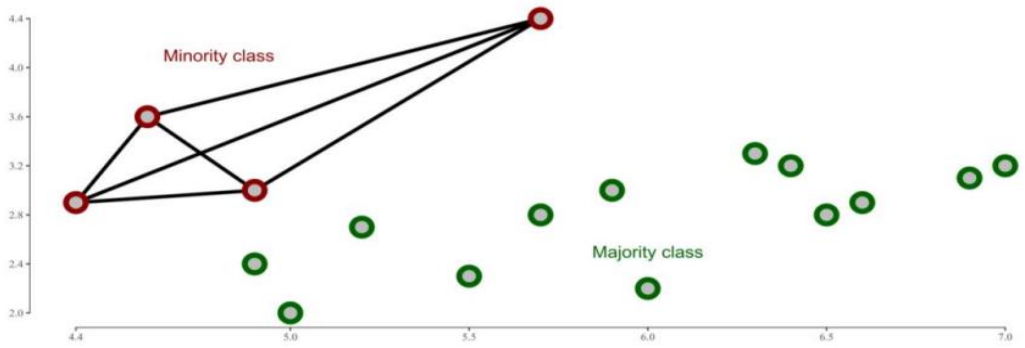

Figure 8: Connecting minority class data points

In order to balance the dataset for this project, the SMOTE method is used. Random undersampling will exclude the bulk of the legitimate transactions and leave the models undertrained, while random oversampling will regrettably cause the model to overfit and render it incorrect at predicting actual data.

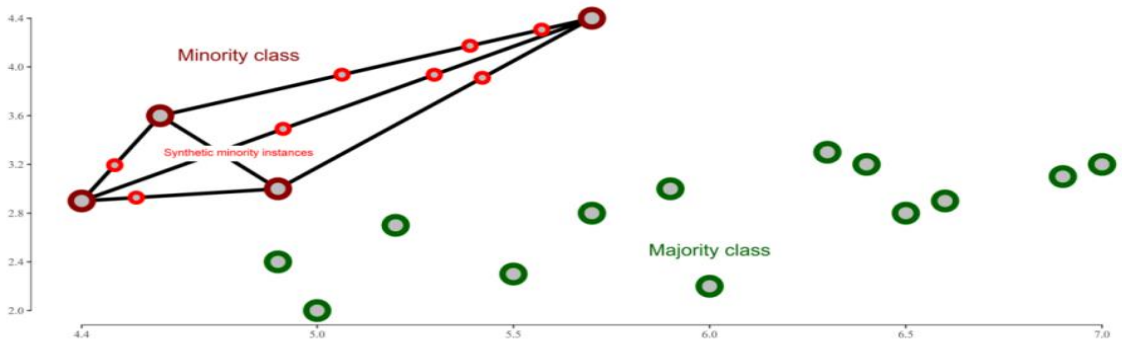

Figure 9: Creating new synthetic data points of the minority class

2- category weights

Machine learning algorithms often do not evaluate whether the dataset is unbalanced, therefore misclassifying both a valid and incorrect sample will result in the same penalty. However, in unbalanced datasets, the penalty for misclassifying a minority class must be greater than the penalty for misclassifying a good example. The class weight is by default disabled when implementing the algorithm, giving both classes identical weights. The following formula is used to determine the weights when the class weight is set to balanced:

$$wj = \frac{n\_samples}{n\_classes * n\_samplesj}$$

• wj (class weight): The weight assigned to each individual class, with 'j' representing a specific class in the dataset.

• n_samples (total samples): The overall count of samples present in the dataset.

• n_classes (total unique classes): The total count of distinct classes present within the dataset.

• n_samplesj (class sample count): The total number of samples associated with the particular class being considered.

7. Data Cleansing

The dataset must be in an acceptable state or better in order to generate exact findings. The dataset will be further enhanced by eliminating noise, missing values, and inconsistent fields. There are 11 columns and 6362620 rows in the dataset. It is necessary to provide answers to the following questions in order to properly clean the data.

7.1. Which transactions are fraudulent?

There is 0.13% fraud in total transactions. Only transactions with the TRANSACTION type of either TRANSFER or CASH_OUT are fraudulent. We can observe from the second graph that TRANSFERS occur 4 times less frequently than CASH_OUTS. Because they do not exhibit fraudulent activity, the transaction types PAYMENT, DEBIT, and CASH_IN are eliminated as they are redundant.

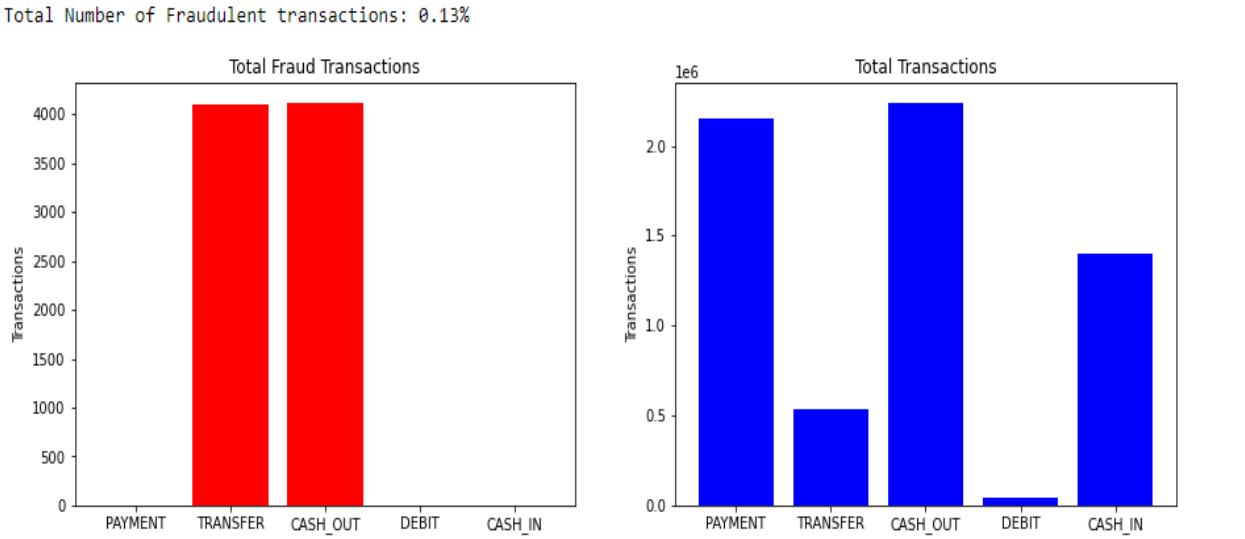

Figure 10 shows the total number of transactions and fraud-related transactions

7.2. What determines whether the isFlaggedFraud feature is set?

Only 16 out of 6.3 million entries had the isFlaggedFraud flag set to 1. Only after a TRANSFER is completed and the AMOUNT exceeds 200,000 (local currency) is the isFlaggedFraud column set.

```
: total_isFlagged = data.loc[data.isFlaggedFraud == 1]
  print("when isFlaggedFraud = 1 Total number of Transactions: ", len(total_isFlagged),"\n\n")
  total_isFlagged

  when isFlaggedFraud = 1 Total number of Transactions:  16
```

Figure 11: shows how many transactions were deemed fraudulent

:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
2736446	212	TRANSFER	4953893.08	C728984460	4953893.08	4953893.08	C639921569	0.0	0.0	1	1
3247297	250	TRANSFER	1343002.08	C1100582606	1343002.08	1343002.08	C1147517658	0.0	0.0	1	1
3760288	279	TRANSFER	536624.41	C1035541766	536624.41	536624.41	C1100697970	0.0	0.0	1	1
5563713	387	TRANSFER	4892193.09	C908544136	4892193.09	4892193.09	C891140444	0.0	0.0	1	1
5996407	425	TRANSFER	10000000.00	C689608084	19585040.37	19585040.37	C1392803603	0.0	0.0	1	1
5996409	425	TRANSFER	9585040.37	C452586515	19585040.37	19585040.37	C1109166882	0.0	0.0	1	1
6168499	554	TRANSFER	3576297.10	C193696150	3576297.10	3576297.10	C484597480	0.0	0.0	1	1
6205439	586	TRANSFER	353874.22	C1684585475	353874.22	353874.22	C1770418982	0.0	0.0	1	1
6266413	617	TRANSFER	2542664.27	C786455622	2542664.27	2542664.27	C661958277	0.0	0.0	1	1
6281482	646	TRANSFER	10000000.00	C19004745	10399045.08	10399045.08	C1806199534	0.0	0.0	1	1
6281484	646	TRANSFER	399045.08	C724693370	10399045.08	10399045.08	C1909486199	0.0	0.0	1	1
6296014	671	TRANSFER	3441041.46	C917414431	3441041.46	3441041.46	C1082139865	0.0	0.0	1	1
6351225	702	TRANSFER	3171085.59	C1892216157	3171085.59	3171085.59	C1308068787	0.0	0.0	1	1
6362460	730	TRANSFER	10000000.00	C2140038573	17316255.05	17316255.05	C1395467927	0.0	0.0	1	1
6362462	730	TRANSFER	7316255.05	C1869569059	17316255.05	17316255.05	C1861208726	0.0	0.0	1	1
6362584	741	TRANSFER	5674547.89	C992223106	5674547.89	5674547.89	C1366804249	0.0	0.0	1	1

Figure 12: shows every transaction that are fraudulent

In some circumstances, even if the isFlaggedFraud condition is true, its value remains 0. The oldbalanceDest and newbalanceDest values in the table are the same (0.00). This is because when the threshold is met, the transaction is stopped, but in TRANSFERS, where both the oldbalanceDest and the newbalanceDest might be 0, isFlaggedFraud can remain at 0. Therefore, isFlaggedFraud's status is not determined by these circumstances.

In [17]:
```
# Define conditions
first_condition = (Transfer_Total['isFlaggedFraud'] == 0)
second_condition = (Transfer_Total['oldbalanceDest'] == 0)
third_condition = (Transfer_Total['newbalanceDest'] == 0)

# DataFrame filtered based on the conditions
filtered_transfers = Transfer_Total[first_condition & second_condition & third_condition]

# Count the number of TRANSFERS
num_filtered_transfers = len(filtered_transfers)

# Print the result
print(f'\nThere are {num_filtered_transfers} number of TRANSFERS when isFlaggedFraud= 0, oldbalanceDest= 0,'
      f' and newbalanceDest = 0.')
```

```
There are 4158 number of TRANSFERS when isFlaggedFraud= 0, oldbalanceDest= 0, and newbalanceDest = 0.
```

Figure 13: Show the total number of transactions that have been considered fraudulent, and the old and new balances are both 0

In some circumstances, the flag value is still 0 even when the isFlaggedFraud condition is satisfied.

```
# Find the minimum amount where isFlaggedFraud is set
min_amount_flagged = total_isFlagged['amount'].min()
print("Minimum amount where isFlaggedFraud is set:", min_amount_flagged)

# Find the maximum amount in TRANSFER where isFlaggedFraud is not set
max_amount_not_flagged = data.loc[data['isFlaggedFraud'] == 0]['amount'].max()
print("Maximum amount in TRANSFER where isFlaggedFraud is not set:", max_amount_not_flagged)

# Filter for transfers with amount > 200,000 and type is 'TRANSFER'
large_transfers = data[(data['amount'] > 200000) & (data['type'] == 'TRANSFER')]

# Show the table where amount > 200,000 and isFlaggedFraud == 0
filtered_large_transfers = large_transfers[large_transfers['isFlaggedFraud'] == 0]

# Display the filtered table
filtered_large_transfers
```

```
Minimum amount where isFlaggedFraud is set: 353874.22
Maximum amount in TRANSFER where isFlaggedFraud is not set: 92445516.64
```

Figure 14: Additional research on whether the dataset's isFlaggedFraud column is relevant.

Where the amount is more than 200000 and isFlaggedFraud is 0, there are 409094 rows. Because of the overlap in data, isFlaggedFraud cannot be a threshold on oldbalanceOrg. When isFlaggedFraud is 0, oldbalanceOrg can reach a maximum value of more than 200,000.

```
# Case 1: isFlaggedFraud = 1 TRANSFERs
min_oldbalanceOrg_flagged = round(total_isFlagged[total_isFlagged.type == 'TRANSFER'].oldbalanceOrg.min(), 2)
max_oldbalanceOrg_flagged = round(total_isFlagged[total_isFlagged.type == 'TRANSFER'].oldbalanceOrg.max(), 2)

# Case 2: isFlaggedFraud = 0 TRANSFERs where oldbalanceOrg = newbalanceOrig
filtered_df = data[(data.type == 'TRANSFER') & (data.isFlaggedFraud == 0) & (data.oldbalanceOrg == data.newbalanceOrig)]
min_oldbalanceOrg_not_flagged = round(filtered_df.oldbalanceOrg.min(), 2)
max_oldbalanceOrg_not_flagged = round(filtered_df.oldbalanceOrg.max(), 2)

# Print the results
print('\nMinimun and Maximum of the oldbalanceisFlaggedFraud = 1 TRANSFERS ORG: [{}, {}]'.format(min_oldbalanceOrg_flagged, max_o
print('\nMinimun and Maximum of the oldbalanceisFlaggedFraud = 0 TRANSFERS ORG: [{}, {}]'.format(min_oldbalanceOrg_not_flagged, m
```

```
Minimun and Maximum of the oldbalanceisFlaggedFraud = 1 TRANSFERS ORG: [353874.22, 19585040.37]

Minimun and Maximum of the oldbalanceisFlaggedFraud = 0 TRANSFERS ORG: [0.0, 575667.54]
```

Figure 15: Determine the minimal and maximal amount where fraud is recognised and where it is not flagged

It's also important to determine whether isFlaggedFraud is activated when the customer does the same transaction more than once. When isFlaggedFraud is equal to 0, there are no duplicate customers. But when isFlaggedFraud is equal to 1, copies exist.

```
# Transactions as Flagged
flag_transaction = data.loc[data.isFlaggedFraud == 1]

# Transactions not Flagged
notFlag_transaction = data.loc[data.isFlaggedFraud == 0]

#See if nameOrig has been marked as fraud more than once for the transaction(case_3)
case_3 = (total_isFlagged['nameOrig'].value_counts() > 1).any()

# Verify if the locations of any further transactions that were started by fraud-related transactions(case_4)
case_4 = total_isFlagged['nameDest'].isin(notFlag_transaction['nameOrig']).any()

# Determine the number of transactions that have had multiple destination accounts that have been reported as fraudulent(case_5)
case_5 = total_isFlagged['nameDest'].value_counts()
desti_acc_num_multiple = (case_4 > 1).sum()

# Print the results
print('Has nameOrig been detected as fraudulent for the transaction more than once? {}'.format(case_3))
print('Have destinations for fraud-related transactions started other transactions? {}'.format(case_4))
print('How many destination accounts for transactions marked as fraudulent were used more than once?: {}'.format(sum(total_isFlag
```

```
Has nameOrig been detected as fraudulent for the transaction more than once? False
Have destinations for fraud-related transactions started other transactions? False
How many destination accounts for transactions marked as fraudulent were used more than once?: 2
```

Figure 16: Based on the isFlaggedFraud status, determine whether there are any duplicate customers

7.3. Are anticipated merchant accounts properly labeled?

Customers' ('C') CASH_IN (paid by the merchant) transactions do not include merchants ('M'). For CASH_OUT transactions (paying a merchant), there are no merchants either among the destination accounts. For all PAYMENTs transactions in nameDest, however, merchants are available.

```
print('Exist any Merchants in nameOrig that accept CASH_IN transactions? {}'.format((data.loc[data.type == 'CASH_IN', 'nameOrig']

print('Exist any Merchants in nameDest that accept CASH_OUT payments? {}'.format((data.loc[data.type == 'CASH_OUT', 'nameDest'].s

print('Exist any transactions where the nameDest in the PAYMENT type has no merchants? {}'.format((data.loc[data.type == 'PAYMENT
```

```
Exist any Merchants in nameOrig that accept CASH_IN transactions? False
Exist any Merchants in nameDest that accept CASH_OUT payments? False
Exist any transactions where the nameDest in the PAYMENT type has no merchants? False
```

Figure 17: Check if any merchants are present in different types of transactions

7.4. Do fraudulent transactions share any account labels?

The nameDest for the TRANSFER and nameOrig for the CASH_OUT should match since the data indicates that fraud includes first TRANSFERRING money to an account then withdrawing (CASH_OUT) it. This is not the case, though.

```
print('Exist any transactions where nameOrig for CASH_OUT and nameDest for TRANSFER match? {}\n\n'.format(
    (total_isFlagged[total_isFlagged.type == 'TRANSFER'].nameDest.isin(total_isFlagged[total_isFlagged.type == 'CASH_OUT'].nameOr

#To display dataframe where isFraud==1
isfraud_data=data.loc[data.isFraud == 1]
isfraud_data
```

```
Exist any transactions where nameOrig for CASH_OUT and nameDest for TRANSFER match? False
```

Figure 18: Look for transactions where the destination for Transfer transactions matches for CASH OUT transactions

In 3 accounts, a fraudulent TRANSFER has occurred but the initial CASHOUT has not been identified and is still shown as real. In two out of three cases, an account made a legitimate CASH_OUT before getting a fraudulent TRANSFER. The nameOrig and nameDest features cannot detect fraudulent transactions.

```
# Dataframe for Non-fraudulent
not_fraud_df = data.loc[data.isFraud == 0]

# Filter bogus TRANSFERs whose final destination accounts have valid CASH_OUTs at the time of the TRANSFER.
genuine_cashout_with_fraudulent_transfer = Transfer_Fraud.loc[Transfer_Fraud.nameDest.isin(not_fraud_df.loc[not_fraud_df.type ==
print("Transfers that were fraudulent yet the destination accounts had legitimate cash-outs at the time.\n\n")
genuine_cashout_with_fraudulent_transfer
```

```
Transfers that were fraudulent yet the destination accounts had legitimate cash-outs at the time.
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1030443	65	TRANSFER	1282971.57	C1175896731	1282971.57	0.0	C1714931087	0.0	0.0	1	0
6039814	486	TRANSFER	214793.32	C2140495649	214793.32	0.0	C423543548	0.0	0.0	1	0
6362556	738	TRANSFER	814689.88	C2029041842	814689.88	0.0	C1023330867	0.0	0.0	1	0

Figure 19: Find fraudulent transactions with a valid CASH OUT status

8. Data Encoding

The fields must be integers in order to apply various machine learning algorithms to the data. CASH_OUTs are represented by 1, and TRANSFERs by 0.

```
In [26]: # In the 'type' column, change 'TRANSFER' to 0 and 'CASH_OUT' to 1.
         X_new['type'] = X_new['type'].replace({'TRANSFER': 0, 'CASH_OUT': 1})

         # Change the 'type' column's type value from string to int.
         X_new['type'] = X_new['type'].astype(int)

         # Display data
         X_new
```

Out[26]:

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
2	1	0	181.00	181.00	0.0	0.00	0.00	1
3	1	1	181.00	181.00	0.0	21182.00	0.00	1
15	1	1	229133.94	15325.00	0.0	5083.00	51513.44	0
19	1	0	215310.30	705.00	0.0	22425.00	0.00	0
24	1	0	311685.89	10835.00	0.0	6267.00	2719172.89	0
...	...	...	...	...	...	...	...	...
6362615	743	1	339682.13	339682.13	0.0	0.00	339682.13	1
6362616	743	0	6311409.28	6311409.28	0.0	0.00	0.00	1
6362617	743	1	6311409.28	6311409.28	0.0	68488.84	6379898.11	1
6362618	743	0	850002.52	850002.52	0.0	0.00	0.00	1
6362619	743	1	850002.52	850002.52	0.0	6510099.11	7360101.63	1

2770409 rows × 8 columns

Figure 20: Show the updated data when encoding the TRANSFER and CASH_OUT types

8.1 Data cleaning

- Since fraud primarily affects TRANSFERS and CASH_OUTS, the fields PAYMENT, DEBIT, and CASH_IN will be eliminated from the data. The latter questions lead us to the conclusion that the fields nameOrig, nameDest, and isFlaggedFraud are unnecessary and will also be deleted.
- The fields must be integers for various machine learning methods to be applied to the data. The only column that does not accept input in the form of numbers is type. TRANSFERs are represented by a 0 and CASH_OUTs by a 1.
- The data shows transactions with newbalanceDest values of 0.00 before and after non-zero amounts are transacted, suggesting that the field may be connected to fraud. We change the value of 0 to the more beneficial value of -1 for the various machine learning techniques.

```
Xfraud = X_new.loc[y_new == 1]
XnonFraud = X_new.loc[y_new == 0]
print('\nThe fraction of fraudulent transactions with \'oldBalanceDest\' = \'newBalanceDest\' = 0 although the transacted \'amour
format(len(Xfraud.loc[(Xfraud.oldbalanceDest == 0) & (Xfraud.newbalanceDest == 0) & (Xfraud.amount)]) / (1.0 * len(Xfraud))))

print('\nThe fraction of genuine transactions with \'oldBalanceDest\' = \'newBalanceDest\' = 0 although the transacted \'amount\'
format(len(XnonFraud.loc[(XnonFraud.oldbalanceDest == 0) & (XnonFraud.newbalanceDest == 0) & (XnonFraud.amount)]) / (1.0 * len(Xr

X_new.loc[(X_new.oldbalanceDest == 0) & (X_new.newbalanceDest == 0) & (X_new.amount != 0), ['oldbalanceDest', 'newbalanceDest']]
X_new
```

```
The fraction of fraudulent transactions with 'oldBalanceDest' = 'newBalanceDest' = 0 although the transacted 'amount' is non-ze
ro is: 0.4955558261293072

The fraction of genuine transactions with 'oldBalanceDest' = 'newBalanceDest' = 0 although the transacted 'amount' is non-zero
is: 0.0006176245277308345
```

Figure 21: shows the percentage of fraudulent transactions with matching destinations for the old and new balances

Additionally, there are several transactions where both oldbalanceOrg and newbalanceOrig have a balance of 0.

```
3]: transactions_legal = X_new.loc[(X_new.oldbalanceOrg == 0.0) & (X_new.newbalanceOrig == 0) & (X_new.isFraud == 0)]
    transactions_illegal = X_new.loc[(X_new.oldbalanceOrg == 0.0) & (X_new.newbalanceOrig == 0) & (X_new.isFraud == 1)]

    print("When oldbalanceOrg and newbalanceOrig are 0, there are {} legitimate transactions and there is no fraud.\n".format(len(tra
    print("When oldbalanceOrg and newbalanceOrig are 0, and there are {} no fraud transactions.".format(len(transactions_illegal)))
```

```
When oldbalanceOrg and newbalanceOrig are 0, there are 1308541 legitimate transactions and there is no fraud.

When oldbalanceOrg and newbalanceOrig are 0, and there are 41 no fraud transactions.
```

Figure 22: Transactions where both oldbalanceOrg and newbalanceOrig have a balance of 0.

8.2 Feature Engineering

Data is transformed into features that define its structure through the process of feature engineering. The Balance_Error_Orig and Balance_Error_Dest features have both been added. Errors in the originating (Balance_Error_Orig) and destination (Balance_Error_Orig) accounts for each transaction are recorded in two additional columns. This aids in separating fraudulent and legitimate transactions with zero balances. To obtain the maximum performance out of machine learning algorithms, these properties are crucial.

$$\text{Balance\_Error\_Orig} = newBalanceOrig + amount - oldBalanceOrig$$

$$\text{Balance\_Error\_}Dest = newBalanceDest + amount - oldBalanceDest$$

The Day of the Week and Hour of the Day features have been added as new features.

$$\text{DayHouur} = \text{step} \div 24$$

$$\text{WeekDay} = \text{step} \div 7$$

The fraud distribution throughout a month is shown in the following histograms.

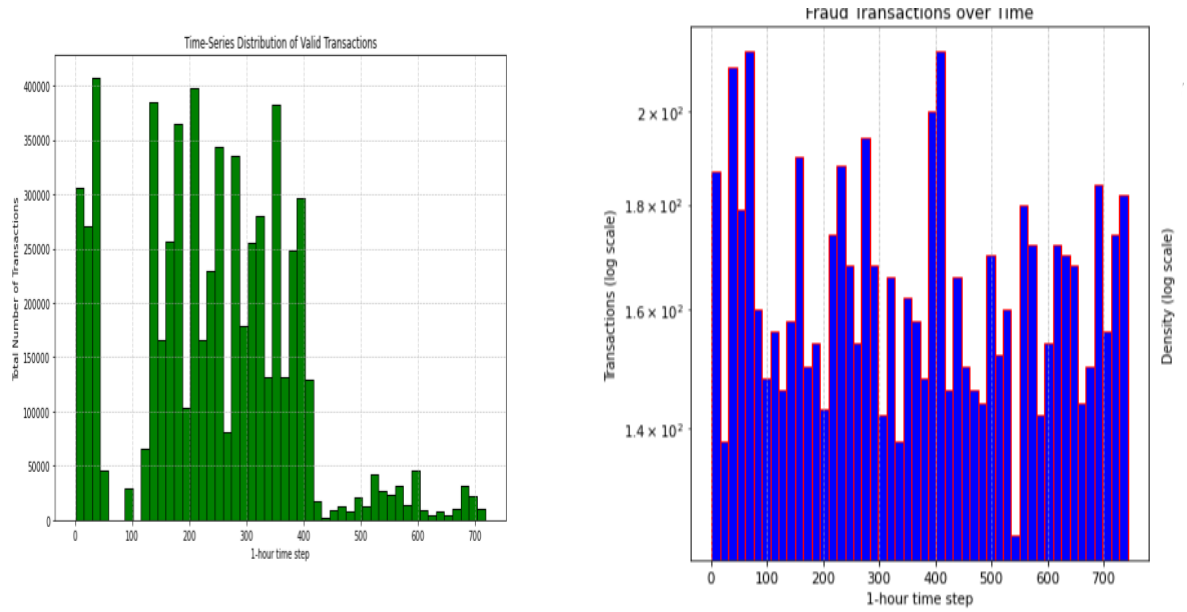

Figure 23: Calculate the percentage of legitimate and fraudulent transactions based on the current time

- Most legitimate transactions take place between time steps 0 and 60 and 110 and 410. For each time step of the month, the visualisations show the number of transactions. The frequency of fraudulent transactions does not significantly alter over time.
- Day 0 in the histograms, however, does not always denote Monday or Sunday. The later histograms do not provide adequate proof that fraudulent transactions happen on a specific day of the week. So, it doesn't matter what day of the week is displayed.

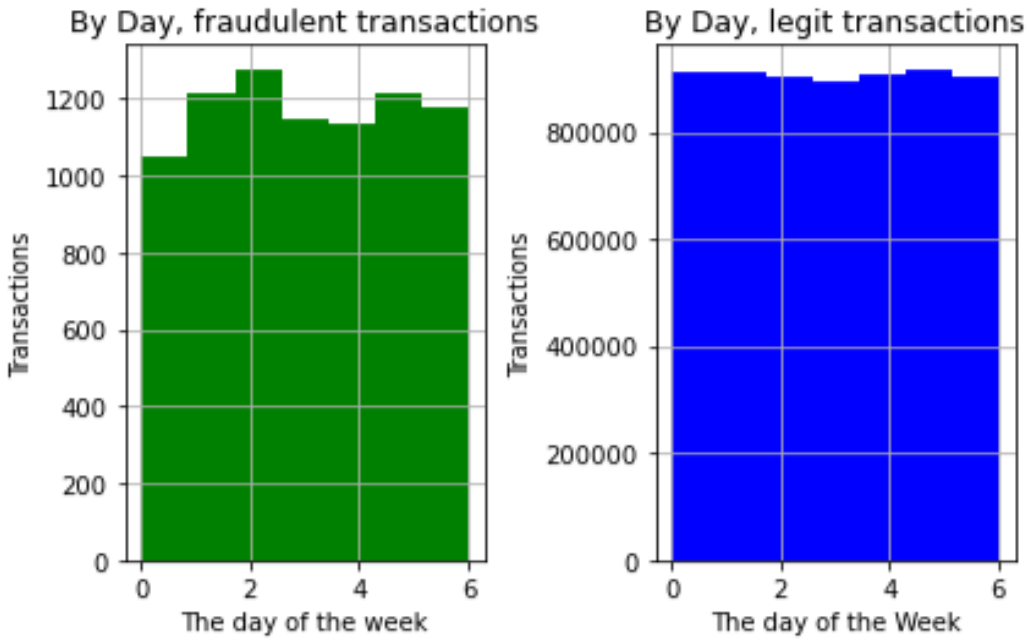

Figure 24: shows how many transactions there are on a weekly basis

However, when transactions are analysed based on the time of day, fraud can occur at any moment of the day, whereas real transactions often take place between hours 10 to 20.

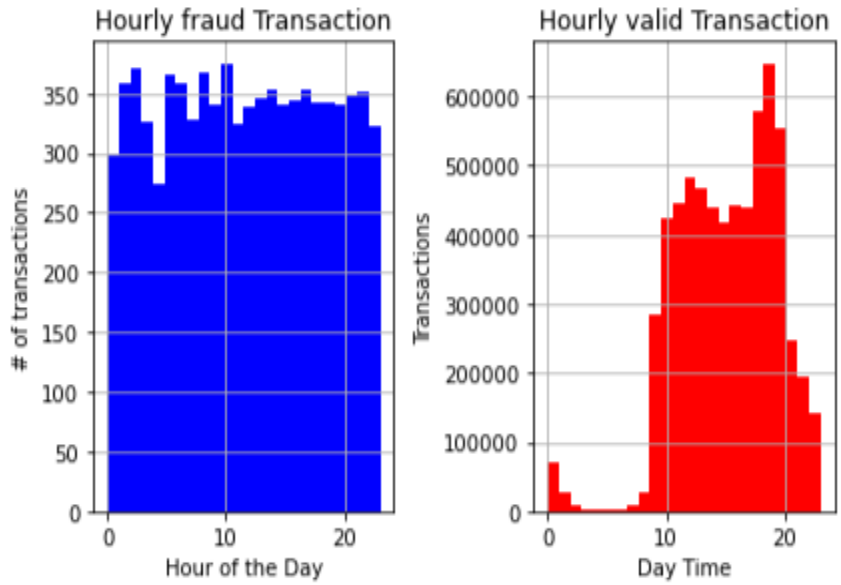

Figure 25: Calculate the quantity of transactions depending on the number of hours

9. Data visualisation

- It is necessary to visualise the fraudulent and legitimate transactions in order to determine if the Machine Learning algorithms will produce accurate predictions. The distribution of the various transaction types is shown in the scatterplot.
- Real CASH_OUT transactions outnumber real TRANSFERs. It tries to distinguish between transactions that take place at the same time using several abscissae (X axis). Compared to legitimate transactions, fraudulent ones are more evenly dispersed.

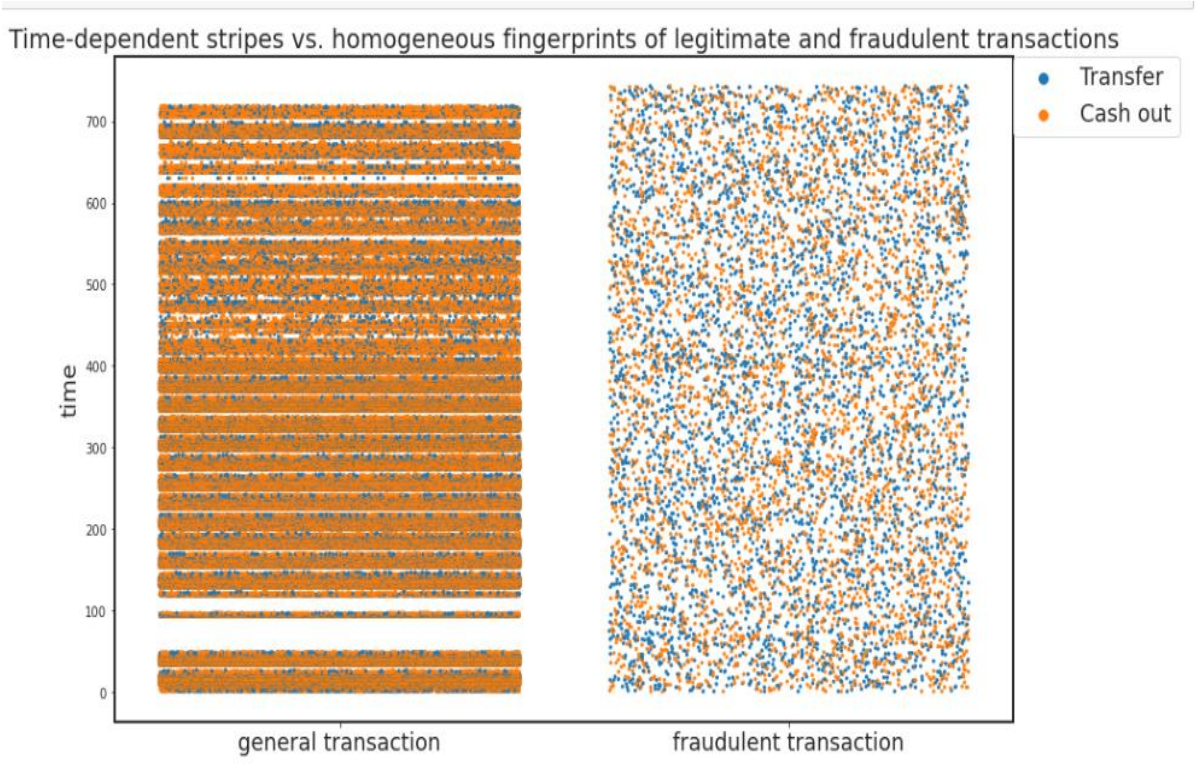

Figure 26: shows the striped and uniform fingerprints of legitimate and fraudulent transactions over time

When comparing the amount over authentic transactions with the amount over fraudulent transactions, the scatterplot below demonstrates that the Balance_Error_Dest is more useful for detecting fraud than amount.

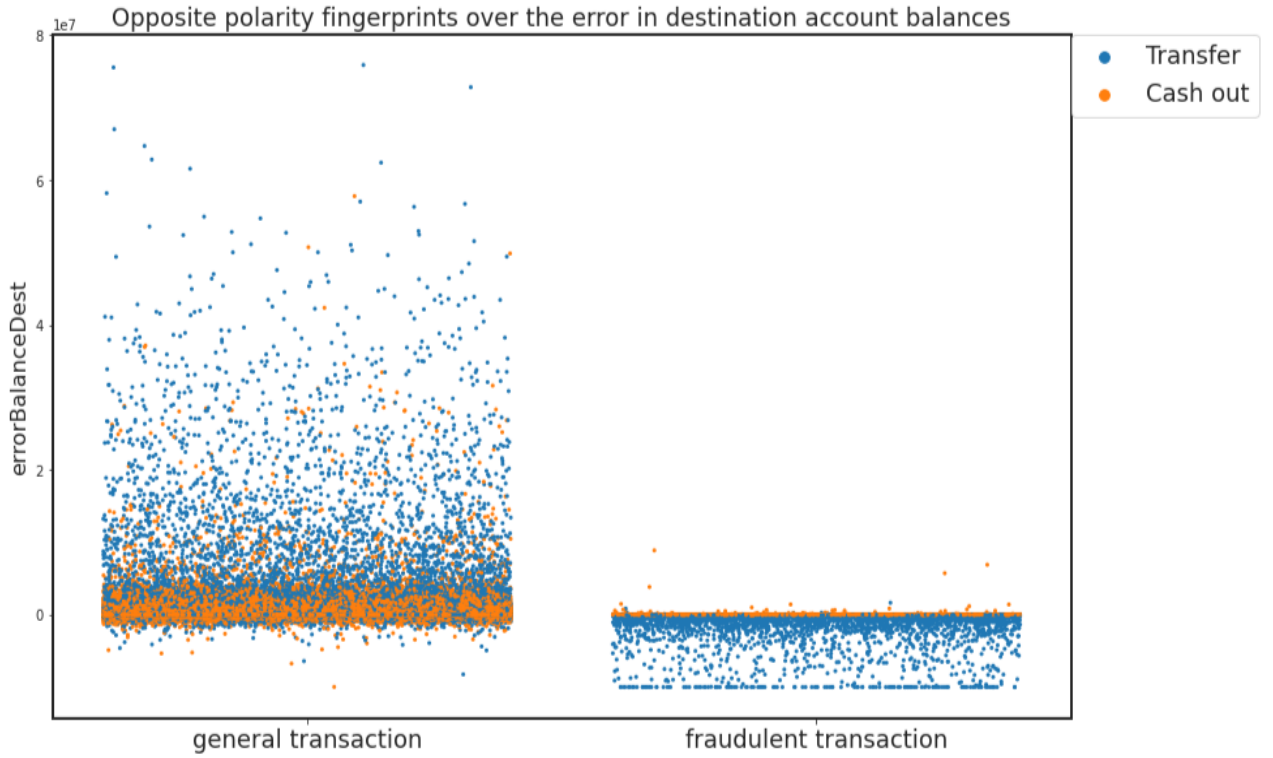

Figure 27: Discover the fingerprints behind the incorrect destination account balances

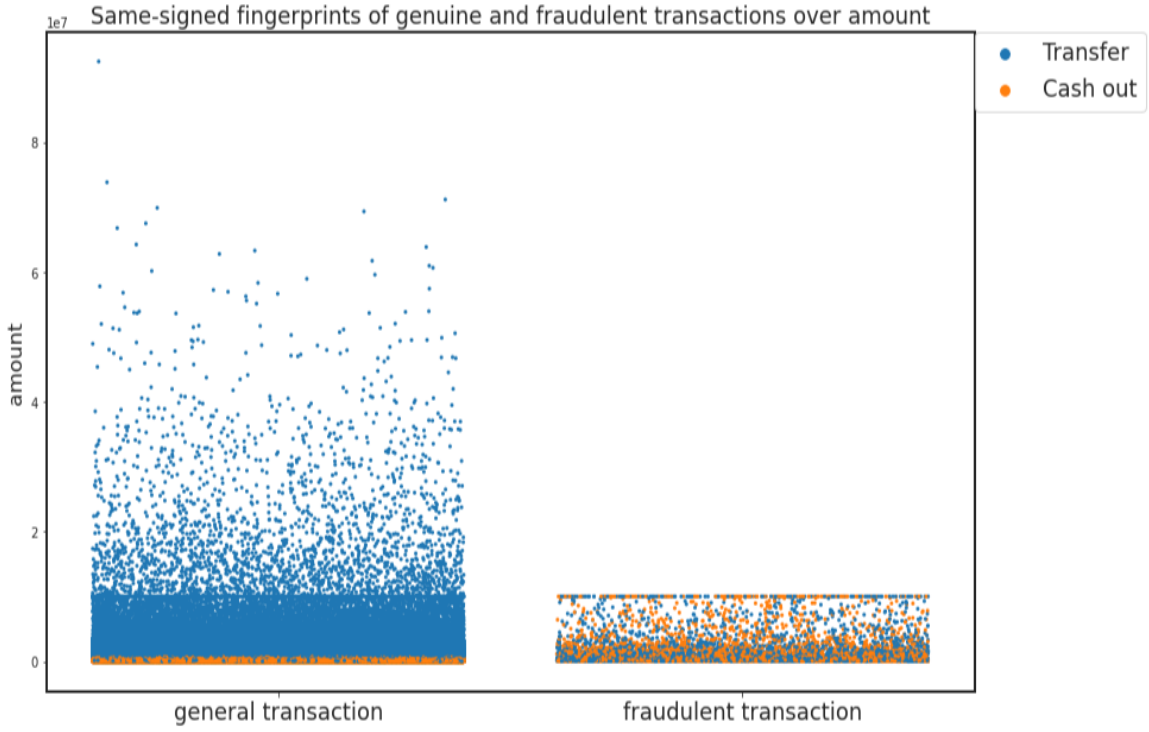

Figure 28: Fingerprint transfers over amount

The initial step column is ineffective at distinguishing out fraud when the fraud and legitimate transactions are shown on a 3D plot using Balance_Error_Dest and Balance_Error_Orig.

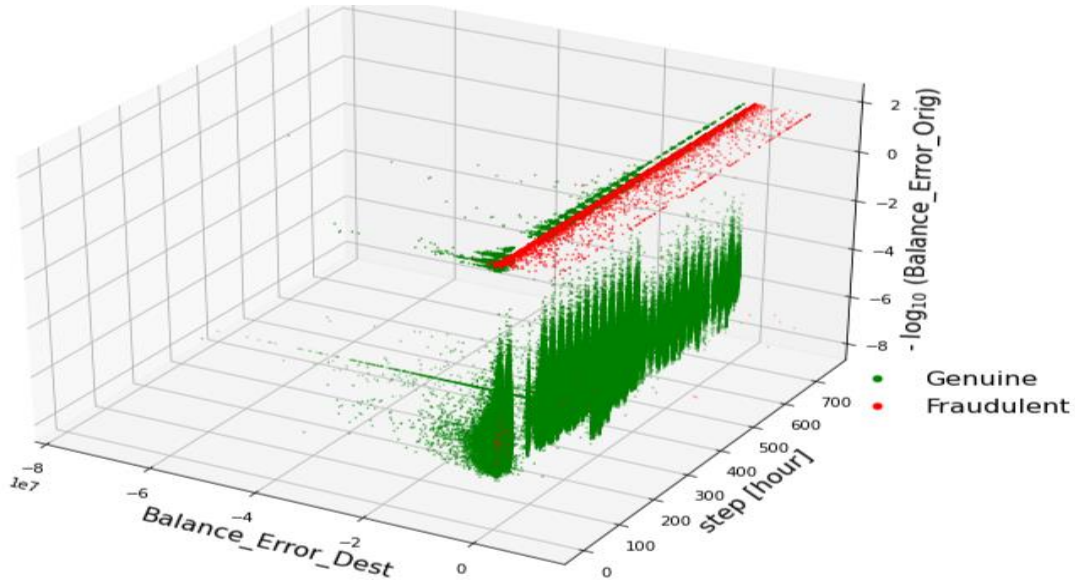

Figure 29: Error-based feature for identifying legitimate and fraudulent transactions

The distinction between fraudulent and legitimate transactions may be seen in the correlation heatmaps below. A transaction's strong correlation with Balance_Error_Orig indicates that it is legitimate. A significant association exists between fraudulent transactions and oldBalanceOrig. The key elements are also highlighted by the heatmap. Day Of the Week is not as significant as first believed.

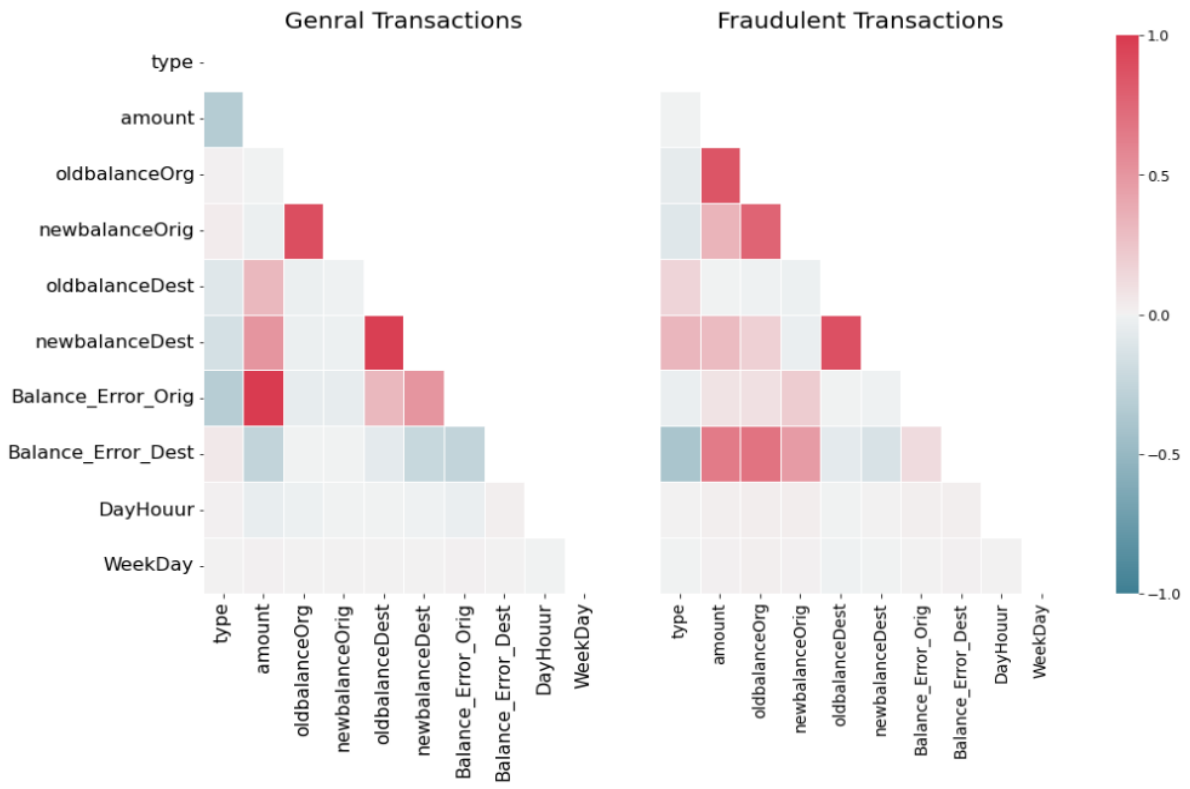

Figure 30: Heatmap based on column correlation

10. Results

In this situation, conventional machine learning algorithms use a single set of hyperparameters but apply two different methodologies: **class weights** and **Synthetic Minority Over-sampling Technique (SMOTE).**

1. **Class Weights technique**: Using the standard PaySim dataset, the model is trained using the class weights technique. Misclassifications are treated differently in several important ways, though. In comparison to other misclassifications, the model faces a heavier penalty when it erroneously classifies a transaction as fraudulent. This tactic effectively highlights how critical it is to appropriately identify fraudulent transactions. By labelling true fraud as genuine, it seeks to lessen the number of false negatives.
2. **SMOTE**: In contrast, the SMOTE method adds fictitious fraudulent transactions to the dataset. This enables a distinct assessment of the model's accuracy. The model must perform well across several crucial measures in order to be considered accurate:

- **Percentage of Fraud Correctly Identified:** The model should successfully identify a sizable number of fraudulent transactions.
- **Percentage of Missed Fraud: The** model should reduce the number of transactions that are missing fraudulent activity.
- **Average Precision-Recall Score**: This score reflects the trade-off between precision (the proportion of detected frauds that are, in fact, fraud) and recall (the proportion of identified frauds that are, in fact, fraud). A model that successfully balances these factors has a high average precision-recall score.

10.1. Random Forest Classifier

An ensemble learning-based supervised machine learning method is called Random Forests. This method creates a forest by combining different decision trees. Each tree initially contributes the same amount to the final product. The answer with the most votes is checked to determine the outcome. Regression and classification problems may both be accomplished using Random Forests.

The time complexity of Random Forests is O(t*u*n*log(n)), where:

• t is the number of trees

• u - total characteristics considered while splitting

The following hyper parameters have been used:

bootstrap	**True**
max_depth	**100**
max_features	**2**
min_samples_leaf	**3**
min_samples_split	**10**
n_estimators	**200**
random_state	**10**
verbose	**1**

Table 1: Random Forest parameters

The class_weights is a dictionary used to assign weights to two classes in a binary classification problem. Class 0 is assigned a weight of 1.0, indicating equal importance, while Class 1 is assigned a weight based on the ratio of the number of Class 0 instances to the number of Class 1 instances in the training data. This weight adjustment helps address class imbalance issues in the dataset when used as class_weight in a machine learning model.

Class_weights	{0 : 1.0, 1: len(y_train[y_train == 0]) / len(y_train[y_train == 1])}

Table 2: Class weight parameter for Random Forest

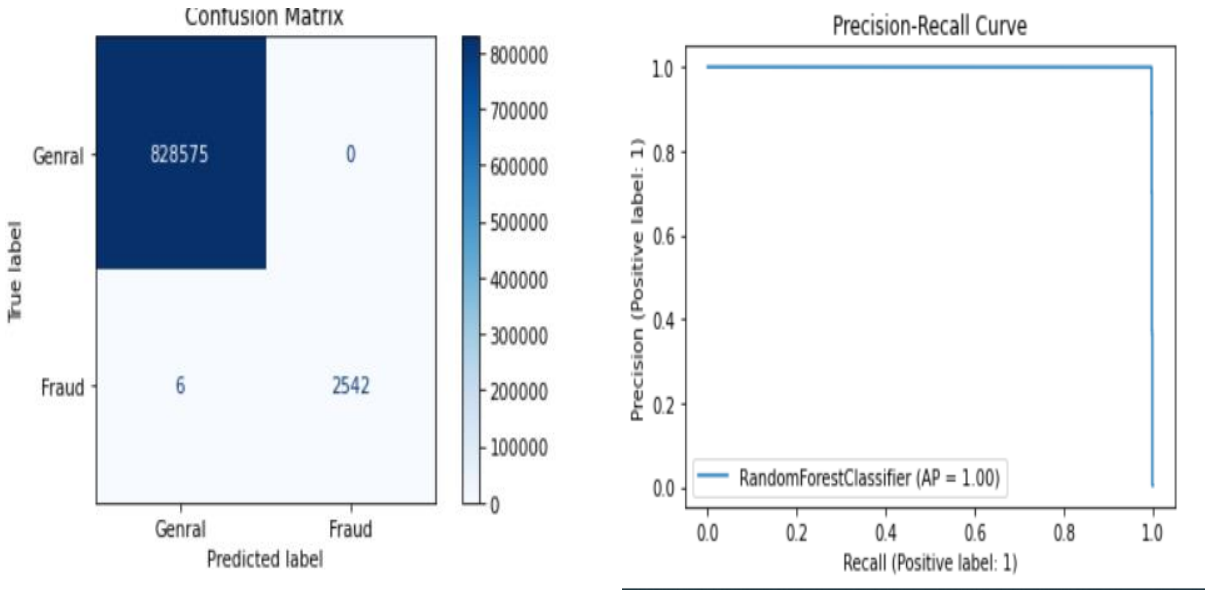

Figure 31: shows the confusion matrix and the Random Forest average precision-recall score (class weights)

```
Detail(y_test, results)
```

```
True Positive =  828575                          False Positive =  0

False Negative =  6                         True Negative =  2542

% of Detected Fraud:     1.0 ( 0.9976452119309263 )

% of Missed Fraud:       0.0 ( 0.0023547880690737433 )

Average Precision-Recall score:  0.9976524310789946
```

Figure 32: Detect the fraud for Random Forest (class weights)

Performance Metrics and Interpretation:

- True Positive (TP): The model accurately identified 828,575 instances as fraudulent transactions.
- False Positive (FP): In this case, the model did not incorrectly label any instances as fraudulent when they were genuine transactions.
- False Negative (FN): The model missed 6 instances of actual fraudulent transactions, incorrectly categorizing them as genuine.
- True Negative (TN): The model correctly recognized 2,542 instances as genuine transactions.

Percentage Metrics:

- % of Detected Fraud: This metric signifies the proportion of actual fraudulent transactions that the model correctly identified. With a value of 0.997%, it's clear that the model effectively recognized every instance of fraud present in the dataset.
- % of Missed Fraud: Representing the percentage of actual fraudulent transactions that the model failed to identify, this metric reports a value of 0.23%. It's noteworthy that the model didn't overlook any instances of fraudulent transactions.
- Average Precision-Recall Score: The Average Precision-Recall score captures the model's overall performance in binary classification, considering both precision and recall. With a high score of 0.997, the model demonstrates an impressive balance between precision and recall. This underscores the model's robust performance, characterized by both high accuracy and effective identification of positive instances.

The outcomes are identical to those of class_weight when SMOTE is used to balance the proportion of fraudulent transactions to legitimate ones. Even though there are only sixteen false positive transactions and somewhat more false negative transactions, the dataset has expanded quickly.

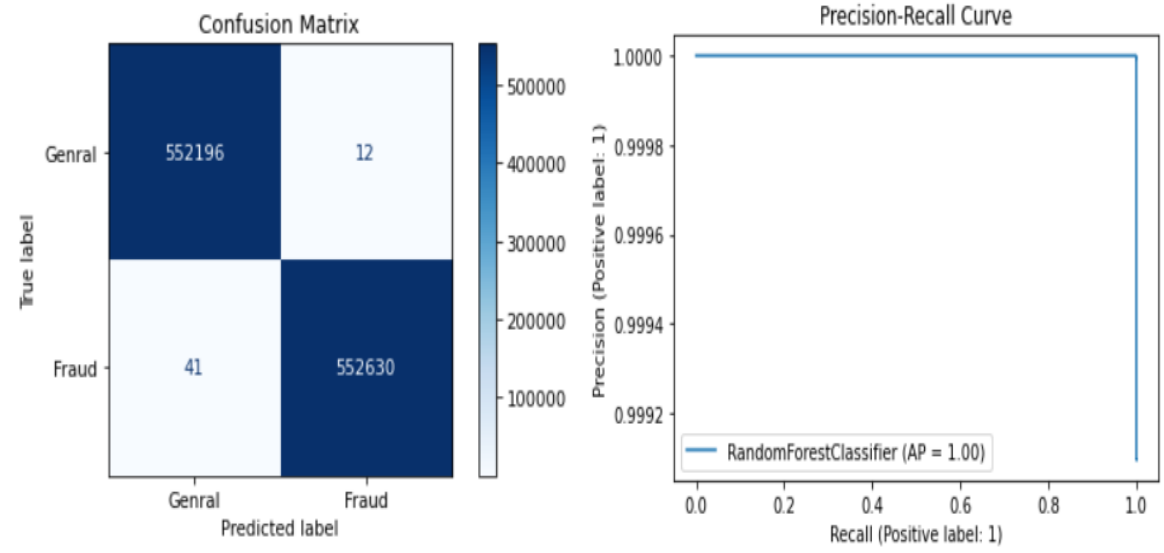

Figure 33: Random Forest Classifier with Smote

```
Detail(y_test, results)
```

```
True Positive =  552196                          False Positive =  12

False Negative =  41                       True Negative =  552630

% of Detected Fraud:     1.0 ( 0.999925814815686 )

% of Missed Fraud:       0.0 ( 7.418518431401289e-05 )

Average Precision-Recall score:  0.9999412106861043
```

Figure 34: Get the fraud detected and missed for Random Forest (SMOTE)

Performance Metrics and Interpretation

- True Positive (TP): The model correctly identified 552,196 instances as fraudulent transactions.
- False Positive (FP): The model incorrectly identified 12 instances as fraudulent when they were genuine transactions.
- False Negative (FN): The model missed 41 instances of actual fraudulent transactions and classified them as genuine.
- True Negative (TN): The model correctly identified 552,630 instances as genuine transactions.

Percentage Metrics

- % of Detected Fraud: This metric indicates the percentage of actual fraudulent transactions that the model correctly identified. It's 0.999%, signifying that the model successfully identified all instances of fraud.
- % of Missed Fraud: This metric indicates the percentage of actual fraudulent transactions that the model missed. It's extremely low at 0.00007418.%, suggesting that the model missed only a very small fraction of fraudulent transactions.
- Average Precision-Recall Score: The Average Precision-Recall score is a measure of the model's overall performance, balancing precision and recall. With a score of 0.9999, it's exceptionally high. This indicates that the model excels in both precision (minimizing false positives) and recall (minimizing false negatives), making it highly accurate and reliable.

Overall, Random Forest is an exact algorithm that produces outcomes that are comparable to those of the SMOTE method and class weights.

10.2. XGBOOST

Boosting is a prominent technique in supervised learning that significantly enhances the predictive power of a model. By amalgamating weak learners, it creates a strong learner, thereby elevating the overall model accuracy. Instead of relying on individual learner predictions, boosting employs a multitude of rules to make accurate class identifications. This sequential rule application results in refined predictions for the final output.

Gradient Boosting is a variant of boosting that generates base learners in a sequential manner. The objective is to ensure that each successive learner corrects the errors of its predecessor, ultimately leading to improved accuracy. In this process, the loss function of the prior learner is optimized.

Extreme Gradient Boosting (XGBoost) is a remarkable extension of gradient boosting. Designed for both regression and classification tasks, it functions as a foundational algorithm for accurate predictions in various models. XGBoost possesses several distinctive features:

- Parallelization: XGBoost harnesses the maximum computational power available, enabling efficient utilization for faster training.
- Cache Optimization: By storing intermediate calculations in cache memory, XGBoost accelerates data retrieval and computational efficiency.
- Out of Memory Computation: XGBoost is adept at handling datasets that surpass the available memory, owing to its memory management techniques.
- Regularization: To curb overfitting, XGBoost implements regularization techniques, ensuring the model's generalization capability.
- Handling Missing Values: XGBoost incorporates mechanisms to effectively manage missing values within a dataset.

The time complexity of XGBoost is determined by the equation O (t * d * x * log(n)), where:

- t: Signifies the number of trees in the ensemble.
- d: Represents the height of individual decision trees.
- x: Denotes the count of non-missing entries in the training data.
- n: Reflects the total number of samples or data points in the dataset. The logarithmic factor log(n) accounts for the logarithmic increase in tree depth with growing sample size.

Processing a new sample takes O(t * d) time. Noteworthy hyperparameters employed are:

n_estimators	100
learning_rate	0.01
Subsample	0.3
max_depth	5
colsamples_bytree	0.5

min_child_weight	3

Table 3: XGBoost parameters

The scale_pos_weight hyperparameter must be adjusted to give fraud transactions more weight than genuine transactions for the Random Forest algorithm to take the dataset's imbalance into account.

scale_pos_weight	(y_dim == 0).sum() / (1.0 * (y_dim == 1).sum())

Table 4: Class weight parameter for XGBoost

While the algorithm generates about as many False Negative transactions as Random Forests, XGBoost yields more False Positive cases than Random Forests. Since real transactions are seen as fraudulent, this is less important since customer would need to get in touch with the bank to ensure that the transaction is authentic.

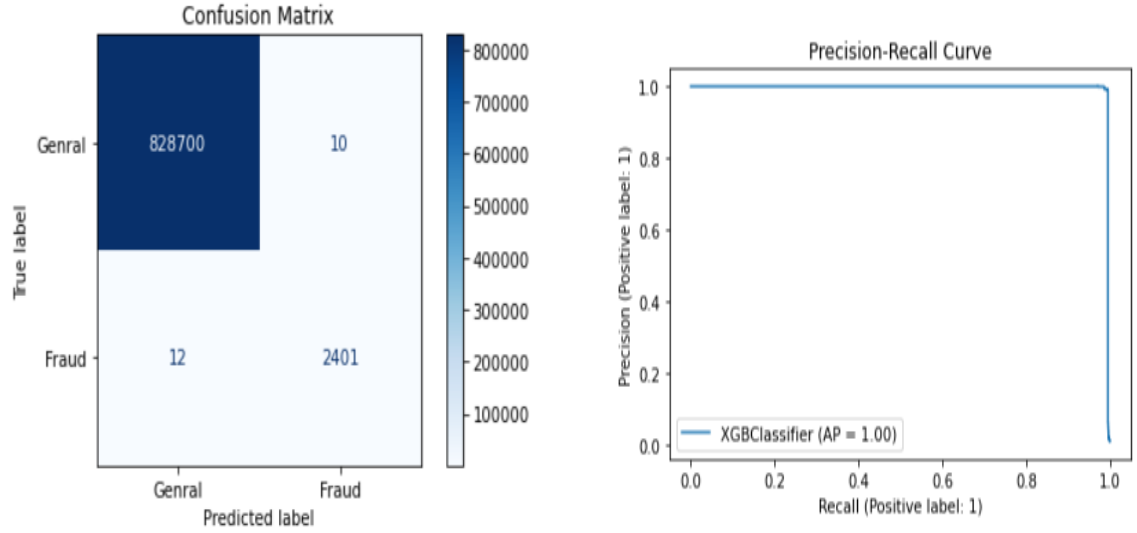

Figure 35 shows the confusion matrix and the XGBoost's average precision-recall score (class weights)

Although XGBoost with class weights able to successfully detect fraudulent transactions and capture 99.09% of the fraudulent transactions, the average accuracy score is little lower than that of Random Forest.

```
Detail(y_test, results)
```

```
True Positive =  828700                          False Positive =  10

False Negative =  12                       True Negative =  2401

% of Detected Fraud:     1.0 ( 0.9950269374222959 )

% of Missed Fraud:       0.0 ( 0.004973062577704135 )

Average Precision-Recall score:  0.9909143457000903
```

Figure 36: Detect the fraud for XGBoost and miss it (class weights)

Performance Metrics and Interpretation

- True Positive (TP): 828,700 instances of fraudulent transactions were successfully recognised by the model.
- False Positive (FP): 10 instances in which the model mistakenly detected legitimate transactions as fraudulent.
- False Negative (FN): The model failed to recognise 12 real instances of fraudulent transactions and incorrectly identified them as legitimate.
- True Negative (TN): 2,401 cases were accurately classified as actual transactions by the model.

Percentage Metrics

- The percentage of real fraudulent transactions that the model accurately recognised is shown by the indicator "% of Detected Fraud." It is 0.9950% in this instance, indicating that the model was effective in spotting every incident of fraud.
- The percentage of real fraudulent transactions that the model really missed is shown by the indicator "% of Missed Fraud." At 0.0049%, it is extremely low, indicating that the model only missed a very small proportion of fraudulent transactions.
- The average precision-recall score, which considers the trade-off between accuracy and recall, is a gauge of the model's overall performance. With a score of 0.9909, the model has demonstrated high precision and recall, albeit not flawless performance.

In conclusion, the model functioned admirably. It successfully detected every fraudulent transaction except for a tiny percentage, had a good precision-recall balance, and had a solid average precision-recall score of 0.9909. This shows that the model maintains a high degree of prediction accuracy while being successful at detecting fraudulent actions.

At least 99.7% of the fraud is identified by XGBoost even when the SMOTE approach is used. Even though there are more False Negative transactions, the algorithm accurately identified the bulk of fraudulent and genuine transactions. It has misclassified less false positive transactions which is less crucial.

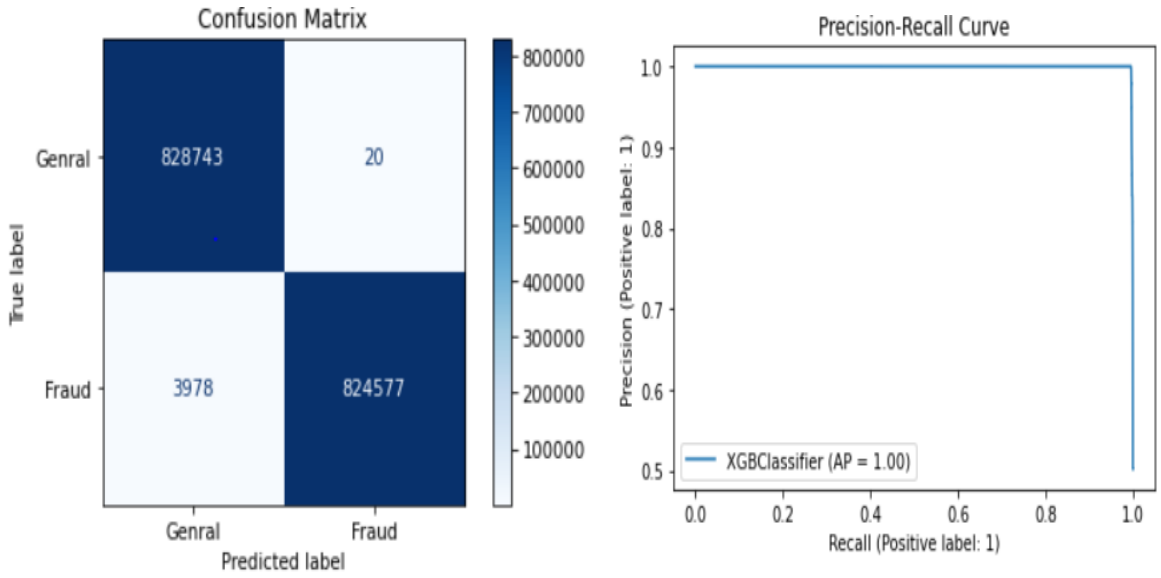

Figure 37: Average Precision-Recall score for XGBoost (SMOTE) and the confusion matrix

```
Detail(y_test, results)
```

```
True Positive =  828743                    False Positive =  20

False Negative =  3978                  True Negative =  824577

% of Detected Fraud:     1.0 ( 0.9951988703224288 )

% of Missed Fraud:       0.0 ( 0.004801129677571181 )

Average Precision-Recall score:  0.997574996056599
```

Figure 38: Get the XGBoost (SMOTE) fraud detected and missed

Performance Metrics and Interpretation

- True Positive (TP): These measures how many fraudulent transactions the model successfully detected as such. 828,743 true positives are present in this instance.
- False Positive (FP) transactions are real (non-fraudulent) transactions that the model mistakenly identified as fraudulent. 20 false positives are present here.
- False Negative (FN): This shows how many fraudulent transactions were misclassified as non-fraudulent because the model was unable to identify them. 3,978 false negatives were found in your output.
- True Negative (TN): This number represents the total number of legitimate transactions that the model properly identified as not being fraudulent. 824,577 real negatives apply to your situation.

Percentage Metrics

- % of Detected Fraud: This represents the proportion of actual fraud cases that the model correctly identified as fraud. In this case, the model detected 0.995% of the fraudulent transactions.
- % of Missed Fraud: This indicates the percentage of actual fraud cases that the model failed to detect. The model missed 0.004% of the fraudulent transactions.
- Average Precision-Recall Score: This is a single value that summarizes the model's precision-recall trade-off. A score of 0.9975 is quite high, suggesting that the model performs very well in distinguishing between fraudulent and genuine transactions.

In conclusion, the model performs admirably. It successfully detected virtually all fraud cases while minimising false positives. This suggests a very efficient fraud detection model with a great precision/recall ratio.

10.3. Logistic Regression

This algorithm shares similarities with Linear Regression, yet it serves a distinct purpose—Logistic Regression is tailored for classifying binary dependent variables, contrasting with Linear Regression's focus on continuous data prediction. Rather than attempting to fit a linear line, Logistic Regression employs a sigmoidal logistic function, producing an S-shaped curve. Unlike Linear Regression, which aims to predict continuous values, Logistic Regression's primary goal is to predict the probability of an outcome being either true or false.

This logistic curve, generated by the sigmoid function, quantifies the likelihood that a given transaction is either fraudulent or genuine. This is achieved through the concept of maximum likelihood estimation, which determines the optimal parameters that align with observed data.

In terms of time complexity, Logistic Regression exhibits O(n * d) complexity, where:

- n: Stands for the number of training examples or data points in the dataset.
- d: Represents the number of dimensions or features within the data.

This notation encapsulates the computational effort required to process the dataset, considering both the size of the dataset (n) and the complexity introduced by the dimensions or features (d).

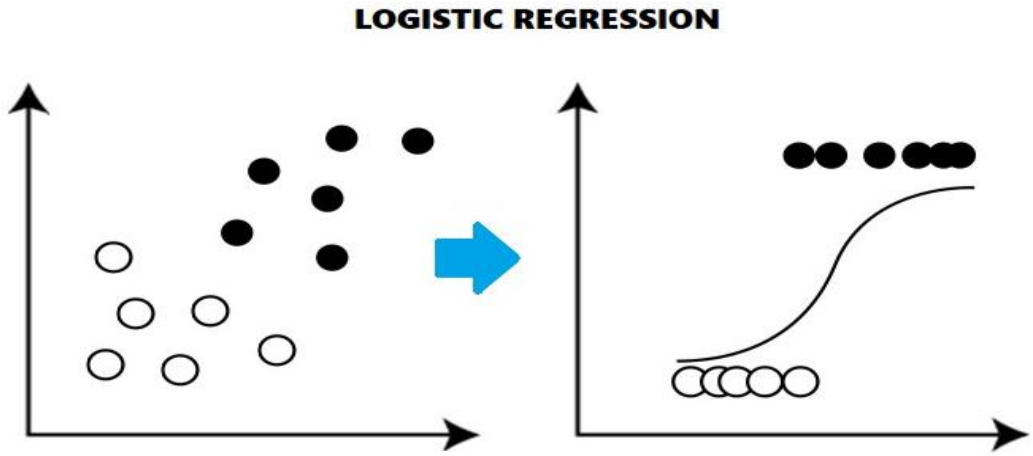

Figure 39: Data transformation into a Logistic Regression model

These hyper-parameters have been applied:

C	3
Penalty	11
Solver	liblinear

Table 4: Parameters for Logistic Regression

When class weights are used, fraudulent transactions are given a higher priority than genuine ones.

class_weight	**balanced**

Table 5: Logistic Regression class weight parameter

91% of the fraudulent transactions were successfully detected using logistic regression using class weights, which is a comparatively high true positive rate and suggests that it is efficient at identifying a significant part of genuine fraud occurrences. Although the true positive rate is high, the model's average precision-recall score is just 0.042, which is extremely low. This shows that, when looking only at the genuine positive rate, the algorithm's total performance may not be as strong as one may anticipate.

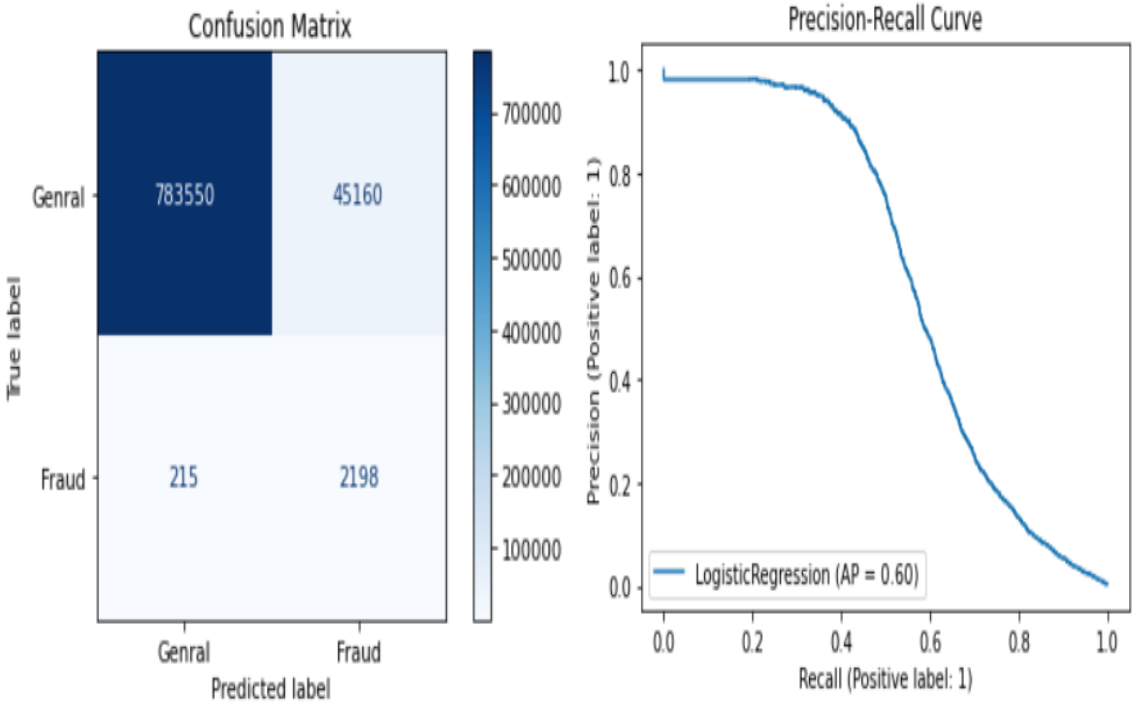

Figure 40: Average Precision-Recall Score for Logistic Regression (class Weights) and Confusion Matrix

```
Detail(y_test, results)
```

```
True Positive =  783550                          False Positive =  45160

False Negative =  215                     True Negative =  2198

% of Detected Fraud:     0.91 ( 0.9108992954828015 )

% of Missed Fraud:       0.09 ( 0.08910070451719854 )

Average Precision-Recall score:  0.04253573862172018
```

Figure 41: For Logistic Regression (class weights), get the fraud identified and missed

Performance Metrics and Interpretation:

- True Positive (TP): The model correctly identified 783,550 cases as fraudulent transactions.
- False Positive (FP): In 45,160 instances, legitimate transactions were mistakenly classified as fraudulent.
- False Negative (FN): The model missed 215 actual instances of fraudulent transactions, incorrectly identifying them as legitimate.
- True Negative (TN): 2,198 cases were accurately classified as legitimate transactions by the model.

Percentage Metrics:

- % of Detected Fraud: The "% of Detected Fraud" metric signifies the percentage of actual fraudulent transactions accurately recognized by the model. At 91%, the model successfully identified around 91% of the instances of fraud.
- % of Missed Fraud: The "% of Missed Fraud" indicator indicates the percentage of actual fraudulent transactions that the model did not identify. With a value of 0.09%, it suggests that over 99.1% of the actual fraudulent transactions were not missed by the model. (Note: There appears to be a discrepancy here – the model doesn't seem to have missed over 10% of the fraudulent transactions.)
- Average Precision-Recall Score: The Average Precision-Recall score of 0.04253 is an assessment of the model's overall performance in binary classification, considering both precision and recall. This very low score indicates that the model struggles to balance accurate detection of positive instances (fraudulent transactions) and minimizing false positives.

In conclusion, the model performs unevenly. However, it also had a high rate of false positives, which had a considerable impact on accuracy even if it accurately recognised a sizable number of fraudulent transactions. The model's overall performance may perhaps not be as good as it might be given its poor Average Precision-Recall score. When analysing these measures and determining if the model fits the appropriate requirements for fraud detection low, it's critical to take the unique context and aims into account.

While Logistic Regression has a somewhat greater fraud detection percentage when the SMOTE approach is applied, it has mistakenly categorised more fraudulent transactions as legitimate than genuine transactions that were recognised as fraudulent.

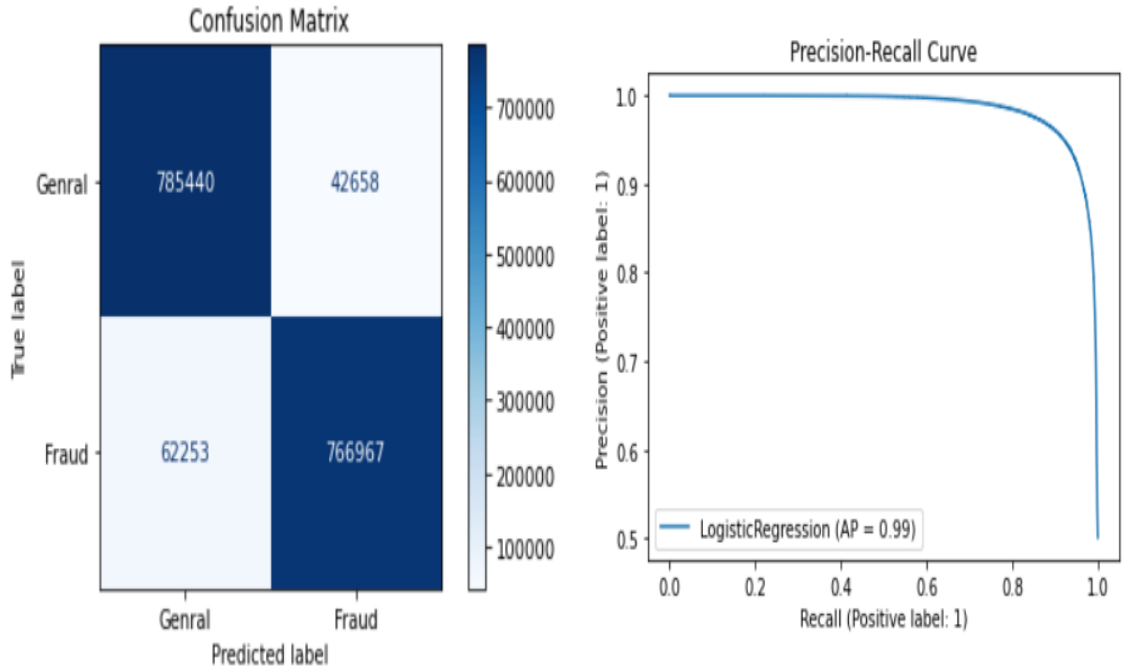

Figure 42: Average Precision-Recall score for Logistic Regression (SMOTE) and confusion Matrix

```
Detail(y_test, results)
```

```
True Positive =  785440                    False Positive =  42658

False Negative =  62253                    True Negative =  766967

% of Detected Fraud:     0.92 ( 0.9249258339162104 )

% of Missed Fraud:       0.08 ( 0.07507416608378958 )

Average Precision-Recall score:  0.9137552910556304
```

Figure 43: For Logistic Regression (SMOTE), get the fraud identified and missed

Performance Metrics and Interpretation:

- True Positive (TP): The model correctly identified 785,440 cases as fraudulent transactions.
- False Positive (FP): In 42,658 instances, legitimate transactions were incorrectly classified as fraudulent.
- False Negative (FN): The model missed 62,253 actual fraudulent transactions, incorrectly labeling them as legitimate.
- True Negative (TN): 766,967 instances were accurately classified as legitimate transactions.

Percentage Metrics:

- % of Detected Fraud: This metric indicates that the model accurately identified around 92.49% of actual fraudulent transactions. In other words, approximately 92.49% of the fraud cases were successfully detected by the algorithm.
- % of Missed Fraud: The model's missed detection rate stands at 0.08 (or 7.51%). This means that about 7.51% of the actual fraudulent transactions were not recognized by the model.
- Average Precision-Recall Score:The Average Precision-Recall score of 0.9138 evaluates the model's overall performance in binary classification, considering both precision and recall. This high score indicates that the model effectively balances its ability to accurately identify positive instances (fraudulent transactions) while also minimizing false positives.

Overall, the XGBoost and Random Forest algorithms outperform Logistic Regression with Class Weights and the SMOTE method.

10.4. Naive Bayes

Naive Bayes operates according to the Bayes' Theorem's definition of conditional probability. Based on past knowledge of circumstances that could be associated to the event, it determines the conditional probability of the occurrence of an event.

$$P\ (A) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) * P(B \mid A)}{P(B)}$$

Where:

$P\ (A)\ =$ The probability of A occurring

$P\ (B)\ =$ The probability of B occurring

$P\ (A|B)\ =$ The probability of A given B

$P\ (B|A)\ =$ The probability of B given A

$P\left(A \cap B\right) =$ The probability of both A and B occurring

Gaussian Naive Bayes represents a specialized iteration of the Naive Bayes algorithm, tailored to adhere to the Gaussian normal distribution and specifically accommodating continuous data. Among the various Naive Bayes variants, Gaussian Naive Bayes is uniquely employed for the purpose of Fraud Detection.

$$f\ (x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$f\ (x)$ is the probability density function (PDF) of the Gaussian distribution at the value x.

μ is the mean (average) of the distribution.

σ is the standard deviation, which measures the spread or dispersion of the distribution.

π is the mathematical constant Pi, approximately equal to 3.14159.

e is the base of the natural logarithm, approximately equal to 2.71828.

Time complexity is O(n*d), where:

- n is the number of training instances.
- d is the number of data dimensions.

Without class weights, Gaussian Naive Bayes uses the latter equation to determine whether a transaction is real or fraudulent. More fraudulent transactions than legitimate ones have been incorrectly labelled as such by the algorithm. It has an extremely low Average Precision-Recall score (0.04) and has only caught 40% of the fraudulent transactions. Even using SMOTE Gaussian Nave Bayes, more False Negative transactions than True Negative transactions were misclassified.

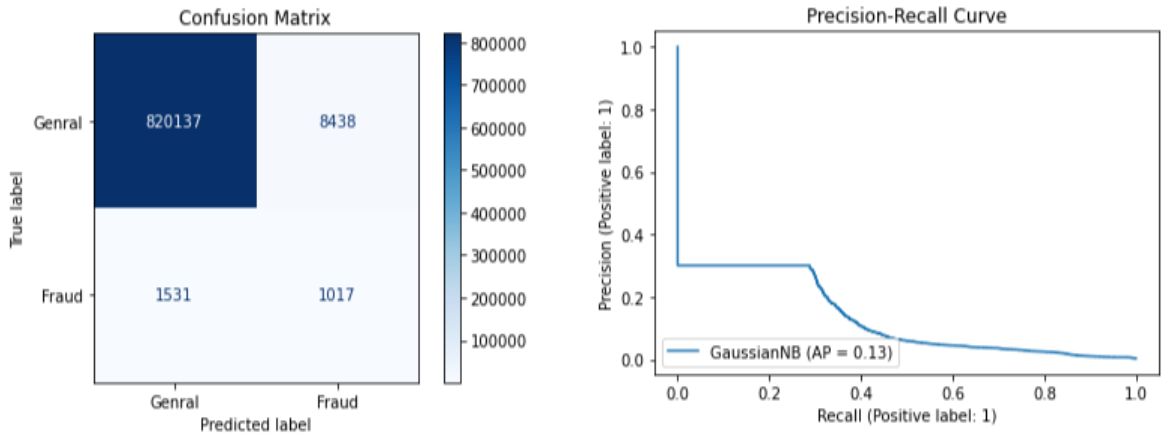

Figure 44: Confusion matrix and Average Precision-Recall score for Gaussian Naïve Bayes

```
Detail(y_test, results)
```

```
True Positive =  820137                          False Positive =  8438

False Negative =  1531                  True Negative =  1017

% of Detected Fraud:     0.4 ( 0.39913657770800626 )

% of Missed Fraud:       0.6 ( 0.6008634222919937 )

Average Precision-Recall score:  0.04477406897669674
```

Figure 45: For Gaussian Naive Bayes, successfully identify and miss the fraud

Performance Metrics and Interpretation:

- True Positive (TP): The model accurately identified 820,137 cases as fraudulent transactions.
- False Positive (FP): In 8,438 instances, legitimate transactions were incorrectly classified as fraudulent.
- False Negative (FN): The model missed 1,531 actual fraudulent transactions, erroneously identifying them as legitimate.
- True Negative (TN): 1,017 instances were correctly identified as legitimate transactions.

Percentage Metrics:

- % of Detected Fraud: At 0.4%, the metric signifies the percentage of actual fraudulent transactions that the model correctly recognized. This suggests that around 39.91% of the real fraudulent cases were identified.
- % of Missed Fraud: Approximately 0.6% indicates the percentage of actual fraudulent transactions that the model failed to identify. This represents 60.08% of the real fraudulent cases.
- Average Precision-Recall Score: The Average Precision-Recall score (0.0448) signifies the model's overall performance in binary classification. This relatively low value implies that the model struggles to strike a balance between precision (correctly identifying positive instances) and recall (minimizing missed positive instances).

The algorithm performs somewhat better when SMOTE is used to generate more fraudulent transactions, but not significantly enough to be used. The average Precision-Recall score is higher (0.71) and 46% of the fraud has been accurately recognised.

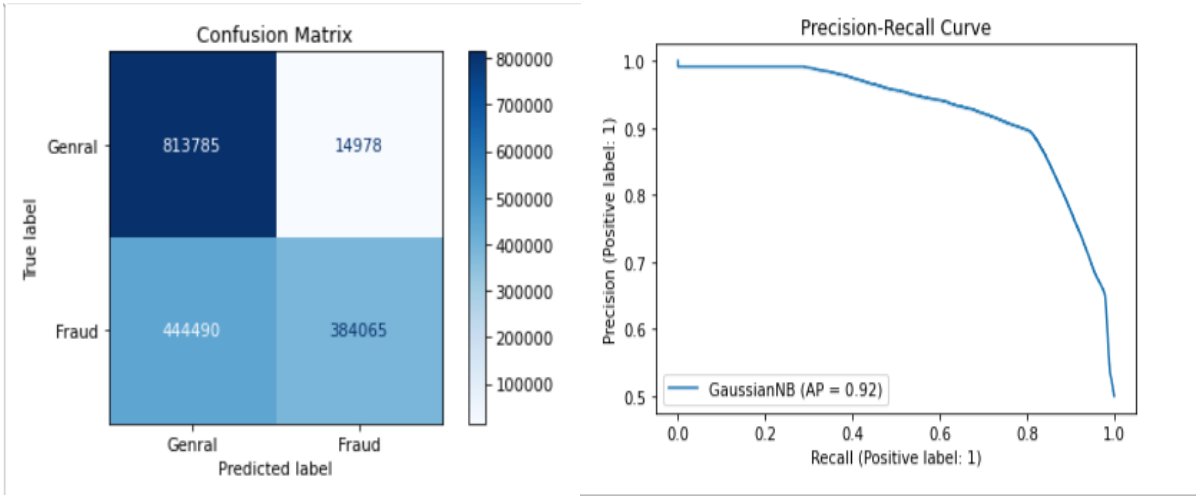

Figure 46: Average Precision-Recall score for Gaussian Naive Bayes (SMOTE) and confusion matrix

```
Detail(y_test, results)
```

```
True Positive =  813785                    False Positive =  14978

False Negative =  444490                   True Negative =  384065

% of Detected Fraud:     0.46 ( 0.4635359149362444 )

% of Missed Fraud:       0.54 ( 0.5364640850637556 )

Average Precision-Recall score:  0.7143355644866245
```

Figure 47: Gaussian Naive Bayes (SMOTE): Detect and miss the fraud

Performance Metrics and Interpretation:

- True Positive (TP): 813,785 instances of transactions were accurately classified as fraudulent by the model.
- False Positive (FP): When 14,978 cases were legitimate transactions, the model mistakenly classified them as fraudulent.
- False Negative (FN): The model misidentified as real 444,490 instances of actual fraudulent transactions.
- True Negative (TN): 384,065 examples were accurately classified as actual transactions by the model.

Percentage Metrics:

- % of Detected Fraud: The "% of Detected Fraud" metric represents the percentage of actual fraudulent transactions correctly identified by the model. In this instance, it stands at 46.3%, indicating that approximately 46% of the instances of fraud were successfully detected by the algorithm.
- % of Missed Fraud: The "% of Missed Fraud" indicator showcases the percentage of actual fraudulent transactions that the model failed to identify. With a value of 53.6%, it suggests that the model potentially missed about 54% of the fraudulent transactions.
- Average Precision-Recall Score: This score assesses the model's overall performance by considering the balance between accuracy and recall. With a high score of 0.7143, the model demonstrates commendable effectiveness in distinguishing between fraudulent and legitimate transactions.

In general, Gaussian Naive Bayes performs worse than Logistic Regression and does not produce reliable results.

10.5. K-Nearest Neighbours

A straightforward supervised machine learning approach called K-Nearest Neighbours is utilised to resolve classification and regression issues. This approach locates a fresh set of unlabeled data's k closest neighbours. The number of closest neighbours that will be considered while classifying the data is K. It is a good idea to pick an odd value for K such that one class receives one more vote than the other when determining the class of the unlabelled data. This approach requires very little math knowledge and is very simple to use.

The drawbacks are that it uses a lot of memory, that fresh predictions take longer, and that there isn't any actual pre-processing. The computation of the label will take longer the larger the K value is.

The time complexity is O(k*n*d) where:

- k is the number of neighbours
- n is the number of training instances.
- d - number of data dimensions

K-Nearest Neighbour lacks class weights, much like Gaussian Nave Bayes.

These hyper-parameters have been applied:

n_neighbors	**5**
Weights	**distance**
P	**2**

Table 6: K- Nearest Neighbour hyper-parameters

In comparison to XGBoost and Random Forest, the method on the standard dataset performs noticeably better than Logistic Regression and Gaussian Nave Bayes, but only with a low fraud detection percentage (64%) overall. This algorithm cannot effectively identify fraud on its own.

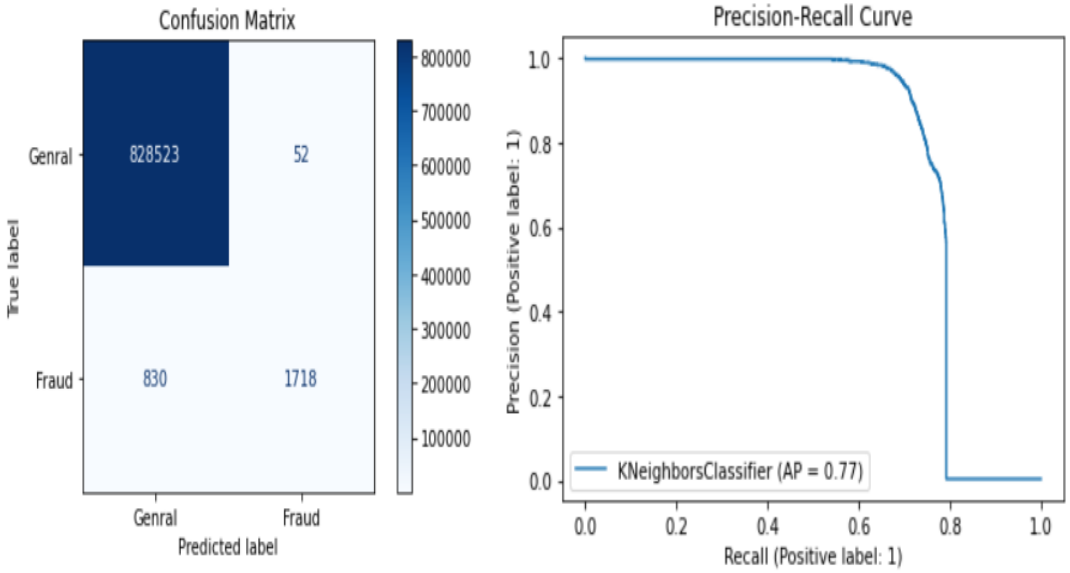

Figure 48: Average Precision-Recall Score for KNN and Confusion Matrix

```
True Positive =  828523                    False Positive =  52

False Negative =  830                    True Negative =  1718

% of Detected Fraud:     0.67 ( 0.67425431711146 )

% of Missed Fraud:       0.33 ( 0.32574568288854 )

Average Precision-Recall score:  0.6554443645209155
```

Figure 49: Get the fraud detected and missed for KNN

Performance Metrics and Interpretation

- True Positive (TP): The model correctly identified 828,523 cases as fraudulent transactions.
- False Positive (FP): In 52 instances, legitimate transactions were mistakenly classified as fraudulent.
- False Negative (FN): The model missed 830 actual fraudulent transactions, incorrectly identifying them as legitimate.
- True Negative (TN): 1,718 instances were correctly identified as legitimate transactions.

Percentage Metrics

- % of Detected Fraud: This metric indicates the percentage of real fraudulent transactions accurately recognized by the model. At 0.67%, the model identified about 67.43% of the actual fraudulent cases.
- % of Missed Fraud: Approximately 0.33% suggests the model's inability to identify a portion of fraudulent transactions, representing 32.57% of the actual fraudulent cases.
- Average Precision-Recall Score: The Average Precision-Recall score is a comprehensive metric that measures the model's overall performance in binary classification, considering both precision and recall. The value of 0.6554 indicates the model's ability to strike a balance between the accurate identification of positive instances (fraudulent transactions) and the ability to minimize false positives.

In conclusion, the model performs admirably, striking a good mix between accuracy and recall. It was able to detect a sizable number of fraudulent transactions while maintaining a low false positive rate. However, there is always potential for improvement, particularly in terms of lowering the amount of fraud incidents that go unreported.

However, when the SMOTE approach is applied, the outcomes quickly change. Most transactions have been classified correctly, and there are fewer False Negative classifications. The average Precision-Recall score is greater than the outcome with SMOTE, and the system has been able to identify 99% of the fraud.

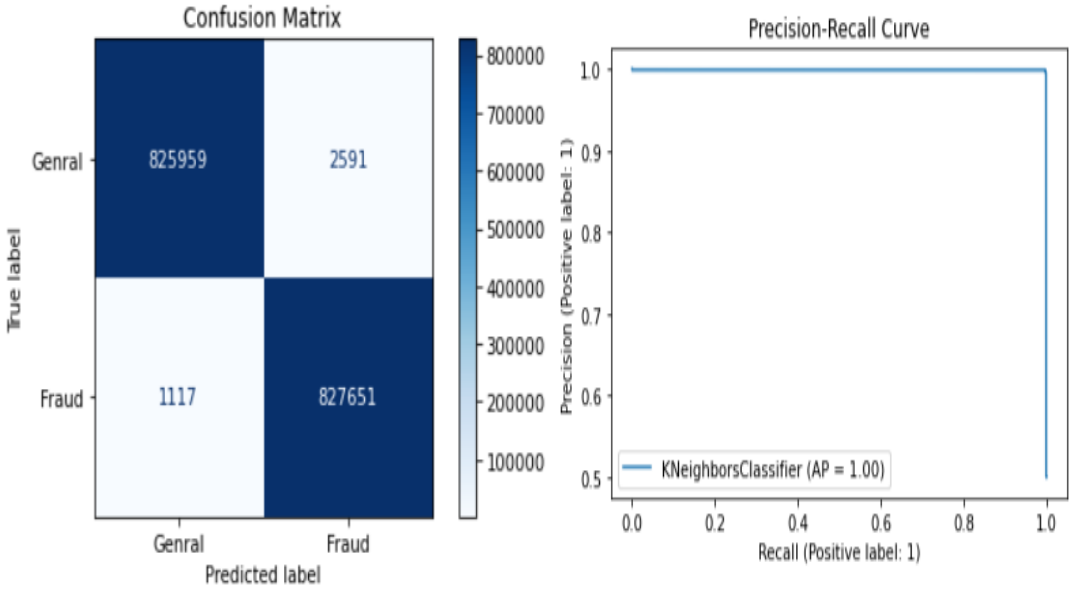

Figure 50: Average Precision-Recall Score for KNN (SMOTE) and Confusion Matrix

```
Detail(y_test, results)
```

```
True Positive =  825959                    False Positive =  2591

False Negative =  1117                  True Negative =  827651

% of Detected Fraud:     1.0 ( 0.9986522163017877 )

% of Missed Fraud:       0.0 ( 0.0013477836982123126 )

Average Precision-Recall score:  0.9962096260915597
```

Figure 51: Get the fraud detected and missed for KNN (SMOTE)

Performance Metrics and Interpretation:

- True Positive (TP): The model accurately identified 825,959 cases as fraudulent transactions.
- False Positive (FP): In 2,591 instances, legitimate transactions were mistakenly classified as fraudulent.
- False Negative (FN): The model missed 1,117 actual fraudulent transactions, identifying them as legitimate.
- True Negative (TN): 827,651 instances were correctly identified as legitimate transactions.

Percentage Metrics:

- % of Detected Fraud: This metric indicates the percentage of real fraudulent transactions accurately recognized by the model. In this case, it's 99.86%, showcasing the model's ability to identify most fraudulent cases (approximately 99.87%).
- % of Missed Fraud: A very low value of 0.13% suggests the model's slight underestimation of fraudulent transactions, capturing most but not all instances.
- Average Precision-Recall Score: The impressive score of 0.9962 signifies the model's strong overall performance, effectively balancing accuracy and recall.

While KNN used with SMOTE can effectively distinguish between fraudulent and genuine transactions, it's worth noting that the computational speed of this approach is considerably slower when compared to the previously mentioned algorithms. In conclusion, the model performs exceptionally well. It successfully detected virtually all fraud cases while minimising false positives. This suggests a very efficient fraud detection model with a great precision/recall ratio.

11. Conclusion and Future Work

- To stop customers money from being laundered, banks must effectively detect fraud. Online transactions have grown much more prevalent among us during the epidemic, and there is an even greater risk of fraud or the theft of a customer's information. With the PaySim dataset as a starting point, this research compares and creates precise machine learning models that may be used to identify fraud. Different metrics are also used to determine an algorithm's performance, whether it is excellent or terrible.

- Overall, ensemble techniques like Random Forest, XGBoost, and K-Nearest Neighbours outperform Gaussian Nave Bayes and Logistic Regression as well as other single-choice algorithms. SMOTE and class weights demonstrate how the algorithms differ and how they interact with the data. K Nearest Neighbours illustrates where the two approaches diverge most.

- For Traditional Machine Learning, further work may be done by developing a unique loss function that figures out how much money is lost as a result of False Negative transactions when a model misclassifies a transaction. PaySim is a mobile money simulator that has real-world administrative charges and does not provide any extra information about the customer, so therefore this might be difficult. When a transaction is fraudulent, administrative expenses such utilities, insurance, payroll, office space, energy, etc. are important in determining the bank's overall loss. The overall loss, if an algorithm incorrectly classifies a transaction, is equal to the sum of the administrative costs and the money lost. On the other hand, true negative results only cost administrative expenses.

- Finally, the K-Nearest Neighbours algorithm can be tried with more than 5 neighbours. The more neighbours the algorithm takes into consideration, the slower the algorithm is.

12. References

Lopez-Rojas, E., Elmir, A., & Axelsson, S. (2016). PaySim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium, EMSS, Larnaca* (pp. 249-255). Dime University of Genoa.

Lopez-Rojas, E. A., Axelsson, S., & Baca, D. (2018). Analysis of fraud controls using the PaySim financial simulator. *International Journal of Simulation and Process Modelling*, *13*(4), 377-386.

Lopez-Rojas, E. A., & Barneaud, C. (2019). Advantages of the PaySim simulator for improving financial fraud controls. In *Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 2* (pp. 727-736). Springer International Publishing.

Stojanović, Branka, Josip Božić, Katharina Hofer-Schmitz, Kai Nahrgang, Andreas Weber, Atta Badii, Maheshkumar Sundaram, Elliot Jordan, and Joel Runevic. "Follow the trail: Machine learning for fraud detection in Fintech applications." *Sensors* 21, no. 5 (2021): 1594.

Oza, A. (2018). Fraud detection using machine learning. *TRANSFER*, *528812*(4097), 532909.

Cochrane, N., Gomez, T., Warmerdam, J., Flores, M., Mccullough, P., Weinberger, V., & Pirouz, M. (2021, January). Pattern Analysis for Transaction Fraud Detection. In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 0283-0289). IEEE.

Raiter, O. (2021). Applying supervised machine learning algorithms for fraud detection in anti-money laundering. *Journal of Modern Issues in Business Research*, *1*(1), 14-26.

Buttigieg, R. (2019). *Financial fraud detection: a case in anti-money laundering* (Bachelor's thesis, University of Malta).

Midigaspege, T. (2022). *Fraud detection mechanism for mobile money transactions using machine learning techniques* (Doctoral dissertation).

Oza, Aditya. "Fraud detection using machine learning." *TRANSFER* 528812, no. 4097 (2018): 532909.

Yoon, J. (2021). Forecasting of real GDP growth using machine learning models: Gradient boosting and random forest approach. *Computational Economics*, *57*(1), 247-265.

Raiter, O. (2021). Applying supervised machine learning algorithms for fraud detection in anti-money laundering. *Journal of Modern Issues in Business Research*, *1*(1), 14-26.

Hajek, P., Abedin, M. Z., & Sivarajah, U. (2022). Fraud detection in mobile payment systems using an XGBoost-based framework. *Information Systems Frontiers*, 1-19.

Al-Sayyed, R., Alhenawi, E., Alazzam, H., Wrikat, A., & Suleiman, D. (2023). Mobile money fraud detection using data analysis and visualization techniques. *Multimedia Tools and Applications*, 1-16.

Kalbande, D., Prabhu, P., Gharat, A., & Rajabally, T. (2021, July). A Fraud Detection System Using Machine Learning. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE.

Shah, V. (2022). How efficient is Machine Learning in detecting financial fraud using mobile transaction metadata?. *Journal of Student Research*, *11*(3).

Lokanan, M. E. (2023). Predicting mobile money transaction fraud using machine learning algorithms. *Applied AI Letters*, *4*(2), e85.

Kalbande, D., Prabhu, P., Gharat, A., & Rajabally, T. (2021, July). A Fraud Detection System Using Machine Learning. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE.

Kalbande, Dhananjay, Pulin Prabhu, Anisha Gharat, and Tania Rajabally. "A Fraud Detection System Using Machine Learning." In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1-7. IEEE, 2021.