A · P · U

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

CAPSTONE PROJECT

CT085-5-M

STUDENT NAME:	SITI NURMAISARAH BINTI MOHD SOBRI
TP NUMBER	TP062943
INTAKE CODE:	APUMF2105DBSA (DE)(PR)
MODULE CODE:	CT085-5-M
SUPERVISOR NAME	DR VAZEERUDEEN ABDUL
TITLE	CREDIT CARD FRAUD DETECTION UNDER EXTREME IMBALANCED DATA

Acknowledgment

First and foremost, I would like to thank my capstone supervisor Dr Vazeerudeen Abdul and Dr Muhammad Ehsan Rana for supervising and supporting this project as it has greatly benefit me in terms of gaining more knowledge on machine learning using the python program. He has been very helpful in giving guidance and feedback on my work to keep me on track with my project.

Not to forget my classmates who were very helpful in guiding and helping me throughout my project of credit card fraud detection under extreme imbalanced data.

Abstract

As we experience a series of technological advancements over time, we will also undergo a transition in consumer behaviour. With the swift advancement of e-commerce, incidences of credit card fraud appear to be a global issue (Kumari & Mishra, 2019). Albeit much research have been conducted to use machine learning models to address this issue, the core cause of why it persists to remain unsolved is the uneven nature of the data itself. This necessitates the development of automated Fraud Detection Systems (FDS) capable of accurately detecting fraudulent transactions and dealing with the diversity of fraudster activity. Typically, the focus of attention in such situations is the minority class. Due to the small number of cases of one class, the machine learning model struggles to generalise the behaviour of the minority class, resulting in poor predicted accuracy (Japkowicz & Stephen, 1998). This has hampered the ability of numerous models to function to their full capacity in tackling this long-standing issue. As a result, multiple resampling approaches will be used in this project, including random under-sampling, Synthetic Modified Oversampling Technique (SMOTE), and Modified Synthetic Modified Oversampling Technique (MSMOTE), which we have defined to be the Support Vector Machine Synthetic Modified Oversampling Technique SVMSMOTE. The outcome of these machine learning models have been assessed by their sensitivity, specificity, accuracy, precision, recall, and F1-Score, which will be used to classify the resampled datasets using classification techniques such as Logistic Regression, Random Forest, and XGBoost. Upon completeing the analysis, we have found that the best model that outperforms the rest is the random forest with SMOTE with precision of 88% and recall of approximately 85%. Morever, the model achieved an overall accuracy of approximately 100%. The project's key contribution will be the development of optimal machine learning algorithms and the most optimal resampling approaches for dealing with class unbalanced difficulties in credit card fraud datasets.

TABLE OF CONTENTS

Chapter 1: Introduction

In recent times, we have witnessed businesses all over the world progressively adopt online operations and transactions in response to technological advancements. Unquestionably, these progressions are what keep those businesses active in their respective markets. Presently, a wide range of payment cards, including credit, charge, debit, and prepaid cards, are readily accessible. As a result, the adoption of payment cards in society has become a necessity, as an increasing number of people choose to conduct transactions without using cash. In 2017, the worldwide e-commerce market was worth 1.85 trillion pounds (GBP), with an estimate of 4.9 trillion by 2021 (Vakulenko et al., 2019). In Malaysia, an analysis by GlobalData on Malaysia's E-commerce Statistics predicted a 24.7% increase in 2020 as a result of the Covid-19 outbreak. The market is anticipated to reach MYR51.6 billion by 2024 (Nikhil, 2020). For that reason, numerous different industries, especially the financial industry such as banks have adopted the Internet as their main approach to conducting business as a result of the advent of e-commerce.

Payment cards enable individuals to make large-scale payments without having to carry significant amounts of cash. They have altered the way people make cashless transactions and made any type of payment more accessible for consumers. This, however, comes at a price that will have a significant impact on financial agencies and their clients (Hordri et al., 2018). The incidence of fraudulent transactions has steadily increased in tandem with the growing number of payment card users. According to a report published by KPMG's Global Banking Fraud from late 2018 to early 2019, roughly 61% of banks surveyed indicated an increase in fraud (KPMG International, 2019). Despite the use of numerous verification measures, the number of fraudulent transaction instances has not decreased much (Somasundaram & Reddy, 2019). Fraudsters tend to use the internet because it allows them to hide their location and profile.

Due to the ever-increasing volume of data, it is becoming increasingly difficult for a human expert to discover meaningful patterns from transaction data. It involves deciding which transactions are false among millions of daily transactions. Data-driven initiatives are increasingly being utilised to help with online transactions, user authentication, credit card details verification, and detecting and blocking fraudulent transactions (Farooq et al., 2019). As a response, banks must engage in innovative technology such as real-time checking, machine learning, and behavioral biometrics to combat fraud.

1.1 Types of Fraud

Fraud has a heavy financial impact both internationally and locally in Malaysia. Fraud is defined as a deliberate act of deception carried out for monetary advantage. It is an unjust practice that is becoming more common by the day. Since a huge number of similar transactions occur at the same time, it is easy to lose track of each one individually and hence detect fraud. Consequently, a specific individual's payment card details might be illegally acquired and used for fraudulent transactions. Unfortunately for the bank institutions, they lose roughly $67 billion on an annual basis as a result of this crime (Makki et al., 2019a).

Insurance fraud, credit card fraud, statement fraud, securities fraud, and many other frauds exist today, where credit card fraud poses to be the most common. We can classify credit card frauds into a few types based on the behaviour of the fraudulent activities.

- **Simple theft**: The most basic form of credit card fraud is a stolen card. It is also the quickest to recognize.
- **Application fraud**: Occurs when people use fake personal credentials to access new credit cards.
- **Bankruptcy fraud**: This involves using a credit card while making a loss and purchasing things with the knowledge that they will be unable to pay. Credit scoring procedures can help avoid this from occurring.
- **Internal fraud**: When bank workers hack card information to utilise it remotely.
- **Counterfeit fraud**: The cardholder does not need to be present when transactions are conducted remotely. All that is required are the information of a valid credit card. Skimming or shoulder surfing can reveal the card's details. This form of credit card fraud can take a long time to discover, and it necessitates the use of sophisticated technologies to recognize transaction patterns.

Janbandhu (2020) estimates that 10,000 credit card transactions occur every second around the world and in Malaysia, every minute, there are 1,000 card transactions. Credit cards have been the most common targets of fraud due to their high transaction volume. Credit card firms have been combating fraud since the Diners Club introduced its first credit card in 1950 (Mittal & Tyagi, 2020). We will concentrate on counterfeit fraud in this paper because it is significantly more difficult to detect. The acceptance or rejection of a transaction occurs in a

very short time frame, ranging from microseconds to milliseconds. Transaction data contains far more genuine transactions than fraudulent ones: in a real-world dataset, the percentage of fraudulent transactions is often considerably under 1% (Dal Pozzolo, Johnson, et al., 2014). As a result, the method used to detect a fraudulent transaction must be exceedingly fast and efficient. Learning from unbalanced data is challenging because most learning algorithms struggle with big class disparities. Class imbalance necessitates the adoption of additional learning procedures such as sampling, which is referred to as unbalanced learning. To solve this problem, certain machine learning algorithms and resampling techniques can be used to tackle this issue of imbalanced data.

1.2 Detection System for Credit Card Fraud

As shown in Figure 1, a credit card fraud detection system (CCFDS) typically consists of five levels of monitoring.

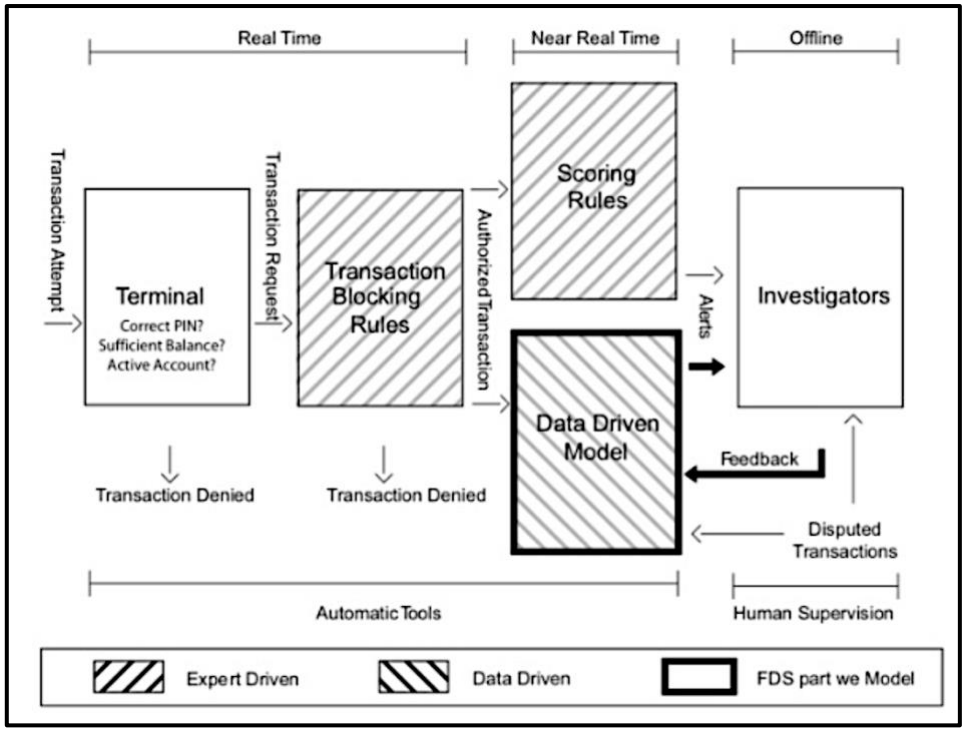


Figure 1: Fraud detection system (Dal Pozzolo et al., 2018)

Real-time execution is used for the first two layers, which are the Terminal and Transaction Blocking Rules. The next two layers, the Scoring Rules and Data-Driven Model (DDM) operate in near real-time to restrict and prohibit further fraud. The Investigators layer is the only one which necessitates human intervention and is conducted offline security procedures include checking the PIN, the number of attempts, the card status, the amount available, and the spending limit. Requests that violate any of these inspections are denied, while the remainder is processed as transaction attempts by the second layer of security. On the other hand, if-then-else statements are used in transaction-blocking rules to detect fraudulent transaction requests (Dal Pozzolo et al., 2018). These recommendations use the information present now the payment is requested, rather than evaluating previous data or cardholder profiles. "IF internet transactions AND unsecured website THEN refuse the transaction," for example, could be a blocking rule (Bontempi, 2021). Similar to the transaction blocking rules, scoring rules also utilise the if-then-else statements (Carcillo et al., 2018). These, on the other hand, use feature vectors to allocate a score to each allowed transaction: the higher the score, the higher the likelihood of the transaction being fraudulent. However, scoring rules can only uncover fraudulent tactics that investigators have already found and that have trends comprising only a few feature vector elements. Furthermore, scoring standards are highly subjective, as different experts create different sets of guidelines. Finally, they can be unfulfilled and difficult to keep up with over time. The data-driven model stage is entirely data-driven, using a classifier or another statistical model to calculate the likelihood of each feature vector being a fraud. Investigators cannot interpret or directly modify the data-driven model because it is developed on a set of labelled transactions. As a result, the DDM is required to uncover frauds using rules that are outside the scope of investigator expertise and do not always correlate to interpretable rules (Pozzolo, 2015). Investigators are experts who are skilful in analysing credit card transactions and are in charge of the FDS's expert-driven tiers. Investigators, in particular, develop transaction-blocking and scoring criteria. Investigators must also determine if the alerts reported by the scoring standards and the DDM are false alarms or frauds (Pozzolo, 2015). They exhibit every potential fraud in a case management interface that includes all essential information, such as assigned scores/probabilities that illustrate how expensive each transaction is in practice.

1.3 Imbalanced Data

The imbalanced learning challenge refers to the efficiency of machine learning models in the situation of unbalanced data and severe class distribution skews. Learning from imbalanced data sets needs innovative solutions, algorithms, and applications for quickly translating large volumes of raw data into information and knowledge discovery due to their inherent complexity. The majority of cases in such scenarios are divided into one of two classes, with the latter being the more crucial of the two. Over the years, several data and algorithmic solutions to the class-imbalance problem have already been developed. Various resampling procedures, such as random under-sampling, random oversampling, oversampling with intelligent synthesis of new samples, and combinations of the above approaches, are used at the data level (Chawla et al., 2002). On the other hand, adjusting the costs of different classes to fix the class imbalance, updating the probabilistic prediction at the tree leaf, changing the decision threshold, and learning based on recognition rather than discrimination are other algorithmic strategies (Provost & Fawcett, 2001). The following variables contribute to the poor performance of classifiers generated by typical machine learning methods on an imbalanced dataset:

1. Accuracy: The minority class provides insufficiently in terms of the accuracy of the algorithm, which is important in determining the minimization of the overall error.
2. Error costs: The existing classifiers presume that errors from various classes have the same costs.

However, careful examination of the data reveals that the actual data sets do not take into account the aforementioned criteria. By allocating all data to the majority class for a data set with 98 majority examples and only two minority cases, a 98% accuracy is attained. Assuming that the minority class reflects a rare condition and that the domain expert is interested in it, the classifier is very unhelpful in this real-world scenario. Consider the following scenarios: cancer vs. non-cancer, fraud vs. legitimate, and system OK vs. system breakdown. The correct threshold can be simply calculated if the error costs and class distribution are specified. However, error costs are difficult to quantify even by human experts in the field, and as a result, these costs are rarely known. It is also worth noting that when errors from distinct classes have varying but unknown costs, even balanced data causes difficulty for classifiers.

The following is a breakdown of the paper's structure. Chapter 2 goes over past work that has been done using various resampling strategies and machine learning models on similar datasets by earlier researchers. Section 3 then goes through the methodology that will be used to carry out the experiment. The dataset, pre-processing methods, proposed resampling approaches, machine learning algorithms, and a variety of assessment metrics were all reviewed in this part.

Chapter 2: Literature Review Data

We discuss a scenario of credit card fraud in which someone uses somebody else's credit card without the cardholder's permission. To benefit from fraudulent conduct, it is very usual to hack someone's credit card PIN or account. Worse yet, the card does not have to be physically present during certain transactions, which raises concerns about the problem. Furthermore, retailers who are victims of credit card fraud will be responsible for all costs, including card issuer fees, charges, and administrative fees (Quah & Sriganesh, 2008). Customers' trust in cashless banking is compromised as fraud rates rise (Nath, 2014). Many researchers have tried to solve this problem by implementing various machine learning approaches, but the outcomes are still insufficient today (Sailusha et al., 2020a). The majority of the classification algorithm's optimization phases try to accurately classify the dominant class while disregarding the others. However, the most difficult element is balancing the data before passing it to the classifiers (Makki et al., 2019a). Since most learning algorithms are not always designed to deal with large variations in the number of examples in each class, learning from imbalanced datasets is difficult. Figure 2 depicts a problem of class imbalance discovered in one of the author's investigations. This section reviews some literature relating to credit card fraud detection and their technique in dealing with the class imbalanced challenges, as this project will be addressing this issue.

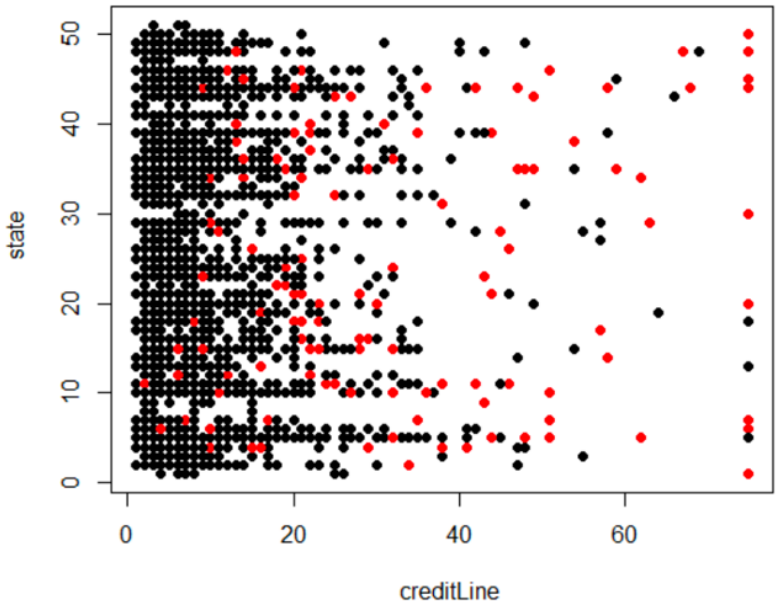


Figure 2: Imbalanced dataset (Makki et al., 2019a)

According to Makki et al., (2019), one of the most significant obstacles in fraud detection is class inequality. This issue is described as a significantly skewed and uneven data distribution. They attempted to find shortcomings in current methods to fix class imbalance datasets, particularly through machine learning models. To begin, the author analysed eight machine learning approaches for fraud detection and classification, including C5.0, SVM, ANN, LR, NB, BBN, KNN, and NSA. The most effective approach was chosen based on three performance indicators. Subsequently, the same algorithms were compared after incorporating class imbalance approaches to them. According to their findings, all algorithms with varying sensitivity and AUPRC values achieved an accuracy of greater than 90%. They found that out of the nine algorithms tested, C5.0, SVM, and ANN were the most suitable for evaluating the performance of imbalance classification, as they all obtained 96% accuracy and high sensitivity of more than 39%, in comparison with other models. Even though NB and NSA have the highest sensitivities, the sensitivity is achieved by increasing the number of false alarms with an accuracy of less than 94%. According to the readings of the AUPRC, LR was found to be the most efficient approach. However, LR was not considered since it required variables to have assumptions and conditions, which was not practical in their situation. The author developed a standard strategy of fraud dataset for C5.0 and compared it to RO C5.0 and CS C5.0 to explore class unbalanced approaches. The conventional SVM was then compared to the OCC SVM and the CS SVM as a binary classification technique. ANN was employed and compared to Auto-Associative.

Mînăstireanu & Meșniță, (2020) investigated a public dataset found on Kaggle, which contained data on transactions done by European owners of credit cards in September 2013. They used three strategies to deal with the unbalanced data concern: resampling, cost-sensitive learning, and tree algorithms, which they implemented to the datasets. The dataset was under-sampled and over-sampled during the resampling technique. The recall score, accuracy score, F_ score, and AUC precision-recall curve were used to evaluate the performance of three models, which were decision tree, random forest, and Naive Bayes loaded from Scikit-learn. Among the three methods, the classifier that uses oversampling with Synthetic Minority The classification model that implements oversampling with the Synthetic Minority Oversampling Technique (SMOTE) technique appeared as the most ideal way of managing an imbalanced dataset, with the best performance metrics measured (precision = 97%), and it was also the preferred method in another literature (Abdellatif et al., 2018). The Random Forest Classifier was the second-best model, with a 94% accuracy in identifying frauds. Even though the

SMOTE method has some significant advantages, such as being independent of the underlying classifier and being very simple to construct, the authors argue that it still consumes a lot of time due to the increased computational cost and overfitting. The author also stated that the overall performance of the classifier should be based on the AUC PR curve findings rather than the AUC of ROC because the former does not reflect the genuine output in high imbalance ratios. Since the false positive rate (FPR) does not fall dramatically when the total number of actual negative cases is large, the ROC curve would not be an effective visual representation of extremely unbalanced data. The authors of another paper by Swamidass et al., (2010) pointed out that the ROC curve might produce inaccurate findings and that it requires extra consideration when the dataset is severely uneven.

Fang et al., (2019) introduced a new technique for identifying credit card fraudulent transactions based on a LightGBM model (Light Gradient Boosting Machine) and the Synthetic Minority Oversampling Technique (SMOTE) algorithm. The LightGBM model was then compared to the Random Forest and Gradient Boosting Machine algorithms in the experiment to verify the new model's superiority in identifying credit card fraudulence. By averaging the results of five distinct training groups, a five-fold cross-validation method was utilised to minimize variance. The results showed that the LightGBM model performed successfully, with an AUC score of 99% in a real dataset and a quick response of 46.63 seconds, compared to Random Forest's AUC score of 98% at 127.29 seconds, demonstrating the new model's effectiveness in identifying credit card fraud. Despite the slight difference in the AUC score, Random Forest (RF) is not suitable for real-time detection since the classifier's training pace is poor due to the employment of a large number of trees.

Itoo et al., (2020) investigated the use of random under-sampling (RU) to address the concern of class imbalance. For establishing the most appropriate resampling ratio, the authors created three distinct ratios of random under-sampling (50:50, 34:66, and 25:75 for fraud and non-fraud) before analysing them using Logistic Regression (LR), Naive Bayes (NB), and k-nearest neighbours (KNN) models. The accuracy, sensitivity, specificity, precision, F-measure, and area under the curve (AUC) were used to evaluate the algorithms' efficiency. Based on these findings, Logistic Regression was proven to be superior to other prediction models, with better outcomes obtained by using under-sampling techniques on the data before generating the prediction mode. The greatest accuracy of the LR was 95%, the NB was 91%, and the KNN was 75%. In addition, when compared to NB and KNN techniques, the LR approach has greater

sensitivity, specificity, accuracy, and F-Measure. In terms of accuracy and AUC values, the 25:75 (fraud: non-fraud) ratio outperformed all other RUS data proportions.

In another study, Khatri et al., (2020) investigated a publicly available imbalanced dataset to examine and assess it with various machine learning algorithms, which were Decision Tree, Logistic Regression, k-nearest neighbour (kNN), Naïve Bayes and Random Forest to discover which one is the most efficient at identifying fraudulent transactions The authors employ a variety of parameters to evaluate the performance of the classifier algorithms, including sensitivity, precision, and the time required to train the model and forecast the test data. Because of its insensitivity to severely imbalanced data, accuracy as a metric was not utilized to assess overall classification performance, according to the author, and would not produce a compelling conclusion. The kNN model has the highest sensitivity, followed by Decision Tree, Random Forest, and Logistic Regression, with values of 81.19%, 79.21%, 78.22%, and 69.31% respectively, based on their experimental results. However, because the time required to forecast the test data for the kNN model is too long (462 seconds), the authors concluded that Decision Tree is the better model than kNN because the predicting time is considerably shorter, around 0 seconds. Even though Naïve Bayes outperformed Decision Tree with a sensitivity of 85.15%, it still has the lowest precision (6.52%) of any supervised machine learning model. The authors claimed that by using resampling approaches to reduce the class imbalance ratio, the model's sensitivity can be greatly enhanced, resulting in more favourable classification results.

By analysing the performance of machine learning models, Mrozek et al., (2020) also evaluated the effects of applying resampling strategies to handle the class unbalanced issue. The authors investigated three types of resampling strategies, including random under-sampling, SMOTE resampling, and the original data without any resampling treatments, to establish the best technique for dealing with the extremely unbalanced dataset. When these three sampling techniques were tested using Logistic Regression, Random Forest, K-Nearest Neighbours, and Stochastic Gradient Descent models, the authors discovered that Random Forest with the random under-sampling technique was the most optimal resampling technique, with a 100% recall and an AUC score of 0.98, followed by Logistic Regression model with random under-sampling and SMOTE resampling methods. While the SMOTE model is praised for its capacity to randomly duplicate the minority class to balance the dataset, it has also been

linked to overfitting the classifier, according to the author. As a result, the random under-sampling method has outperformed oversampling.

In another work, Dhankhad et al., (2018) showed that random under-sampling is among the best techniques for tackling the class unbalanced issue in credit card fraud detection. The accuracy, recall, precision, TPR, FPR, specificity, and G-mean of a variety of machine learning classifiers, including Logistic Regression, Random Forest, Decision Tree, Nave Bayes, Support Vector Machine (SVM), XGBoost, Gradient Boosted Tree, Multilayer Perceptron, Stacking Classifier, and K-Nearest Neighbours, were used to measure the processed data. The stacking classifier, with an accuracy of 95.27%, is the most favourable for predicting fraud transactions, according to the authors, followed by Random Forest and XGB classifiers, both with 94.59% accuracy. For the three models listed, the precision, recall, and F-1 score were all similar.

Recent work by Hordri et al., (2018) applied three commonly used resampling methods, including random under-sampling, random over-sampling, and Synthetic Minority Oversampling Technique (SMOTE), before assessing their efficiency using four classifiers, including Nave Bayes, Logistic Regression, Random Forest, and Multilayer Perceptron. The sensitivity, specificity, accuracy, F-measure, and Area Under Curve (AUC) of the four classifiers were compared. The training dataset was divided into 30:70 and 50:50 (fraud: non-fraud) proportions. All of the classifiers performed admirably, with an accuracy of higher than 90%. For both ratios of each resampling method, Random Forest was found to lead with higher accuracy than other classification approaches. When compared to Random Forest with alternative resampling approaches, the combination of Random Forest with random oversampling produced outstanding results in both ratios.

A study by Dal Pozzolo, Caelen, et al., (2014) emphasized that in a detection task such as fraud detection, it is vital to have the highest precision reading. The goal is to deliver the right status of the minority classes, rather than forecasting each class accurately. Investigators must examine each fraud warning generated by the detection system before moving forward with actions in detection teams. Only a small number of alerts may be verified due to the restricted number of investigators available (Carcillo et al., 2021).

Meng et al., (2020) reported that any feature engineering was not needed as the PCA transformation has made it impossible to create new features from the current ones. Moreover, the author also reported that there were no missing values in the dataset, thus there was no need for imputing missing values. Feature correlation was also not a concern as principle components by construction are orthogonal, thus, uncorrelated with one another. The paper worked on the XGBoost algorithm to detect fraudulent activities in the highly imbalanced dataset. As detecting as many instances of fraud was more important, the author emphasised more on the recall and ROC curve results. It was found that SMOTE was more successful in balancing the data when compared to under-sampling and oversampling as it created a more stable and generalised model of the XGBoost algorithm.

In Varmedja et al., (2019) the author proposed the use of the SMOTE technique to handle the highly imbalanced data. Next, feature selection was also performed by using the feature selector tool by Will Koehrsen to remove irrelevant data, which can reduce overfitting and simultaneously improve the accuracy reading. After performing this method, it was found that 27 out of 31 features were selected for modelling. The models used in this experiment were Random Forest, Logistic Regression, Naïve Bayes and Multilayer Perceptron. It was found that the Random Forest model gave the best output in terms of recall, accuracy and precision with an output of 81.63%, 99.96% and 96.38% respectively.

According to Hu et al., (2009) MSMOTE surpasses SMOTEBoost in balancing the minority class. SMOTE produces synthetic instances by constructing synthetic instances from each minority class example, but it overlooks the spread of minority classes and latent noises in the data set, according to the author. The improved method, MSMOTE, was introduced to optimize the functionality of SMOTE. By evaluating the distances between all of the samples, the enhanced technique splits minority class samples into three groups: security samples, border samples, and latent noise samples. The author carried out the experiment using MSMOTE with j48, SMOTE with j48, MSMOTEBoost and SMOTEBoost. It was found that the MSMOTE applied with both the classifiers shows improved values for precision, recall and f-values of the minority class.

Table 1 summarised the literature review for the study, highlighting the resampling approaches used, the machine learning models selected, and the study's findings. A gap analysis and a possible research opportunity were also presented.

2.1 Summary of Literature Review

Table 1: Summary of the Literature Reviews

Work	Resampling Technique	Machine Learning Model	Results	Gap Analysis/ Research Opportunity
Makki et al. (2019) (Makki et al., 2019b)	Random Oversampling ~Also use Cost-Sensitive Model and One-Class Classification	▪ SVM ▪ ANN ▪ Decision Tree	Imbalanced classification approaches are not that effective, producing a notable number of false positives, hence dropping the accuracy of AUC-PR values.	Gap: Only tested one resampling technique and compare it with an ensemble and classic approaches.
Mînăstireanu & Meșniță (2020) (Mînăstireanu & Meșniță, 2020)	▪ Random Under-sampling ▪ Random Oversampling ▪ SMOTE	▪ Decision Tree ▪ Random Forest ▪ Naïve Bayes	Oversampling with SMOTE emerged as the most optimal way of handling an unbalanced dataset with the best performance metrics recorded (precision = 97%)	
Fang et al. (2019) (Fang et al., 2019)	▪ SMOTE	Light Gradient Boosting Machine	LightGBM model with SMOTE improved training efficiency and demonstrated the highest AUC score with 99%, followed by RF and GBM with 98% and 84%, respectively. The model also exhibits the shortest training time compared to that of RF and GBM models with 46.62, 127.29 and 198.59 seconds, respectively.	Opportunity: The use of Gradient Boosting can accelerate the training speed without damaging the accuracy.
Itoo et al. (2020) (Itoo et al., 2020)	Random Under-sampling	▪ Logistic Regression ▪ Naïve Bayes ▪ k-nearest neighbours	LR possessed the most optimal performance for all RU data proportions, surpassing NB and KNN with the accuracy of 95%, 91% and 75%, respectively. Random under-sampling of 25:75 (fraud: non-fraud) is the best-performed ratio in terms of Accuracy and AUC values	Gap: Only implemented random under-sampling in dealing with the class imbalance issue.
Khatri et al. (2020) (Khatri et al., 2020b)	No resampling used	▪ Decision Tree ▪ Logistic Regression ▪ k-nearest neighbour ▪ Naïve Bayes ▪ Random Forest	Decision Tree is the most preferred model over kNN due to the much lower predicting time (approximately 0 seconds) even though kNN has the highest sensitivity over Decision Tree with 81.19% and 79.21%, respectively.	Gap: Not implementing any resampling techniques causing low sensitivity and accuracy for all machine learning models (less than 82%)
Mrozek et al. (2020) (Mrozek et al., 2020)	▪ Random Under-sampling ▪ SMOTE ▪ Original imbalanced dataset	▪ Logistic Regression ▪ Random Forest ▪ k-nearest neighbours ▪ Stochastic Gradient Descent	Random Forest with random under-sampling emerged as the most ideal resampling technique with an AUC of 0.98 and 100% of recall.	

Dhankhad et al. (2018) (Dhankhad et al., 2018)	Random under-sampling	▪ Logistic Regression ▪ Random Forest ▪ Decision Tree ▪ Naïve Bayes ▪ SVM ▪ XGBoost ▪ k-nearest neighbours	Overall results show that random forest and XGB classifiers with random under-sampling are the most promising classifier for predicting fraudulent transactions based on their accuracy (0.95), recall (0.95), precision(0.95), TPR (0.96) and specificity (0.96).	Opportunity: Can implement the most optimum proportion with other resampling techniques
Hordri et al. (2018) (Hordri et al., 2018)	▪ Random Under-sampling ▪ Random Oversampling ▪ SMOTE	▪ Naïve Bayes ▪ Logistic Regression ▪ Random Forest ▪ Multilayer Perceptron	RF showed a robust performance in three resampling methods, surpassing NB, MLP and LR. In addition, random oversampling appeared as the most optimum resampling technique compared to that of SMOTE and random under-sampling.	Opportunity: Need to address SMOTE limitation.

2.2 Gaps in Literature

Following the previously discussed literature, determining which resampling strategy is the most effective in overcoming this problem remains a mystery to be solved. That being said, many researchers have recently adopted the SMOTE methodology as their ideal way of resampling. It is said to be incredibly efficient and has exceeded many other resampling approaches numerous times. Nevertheless, (Hordri et al., 2018) found that this approach was still vulnerable to oversampling on both minority and majority data, which is one of its known flaws (Figure 3). Thus, our research sees an opportunity to tackle this problem by implementing various resampling strategies to close the research gaps about the current class imbalance problem. Furthermore, this study proposes the use of another resampling technique, the Modified Synthetic Minority Oversampling Technique, which has never been utilised before to balance the fraud dataset.

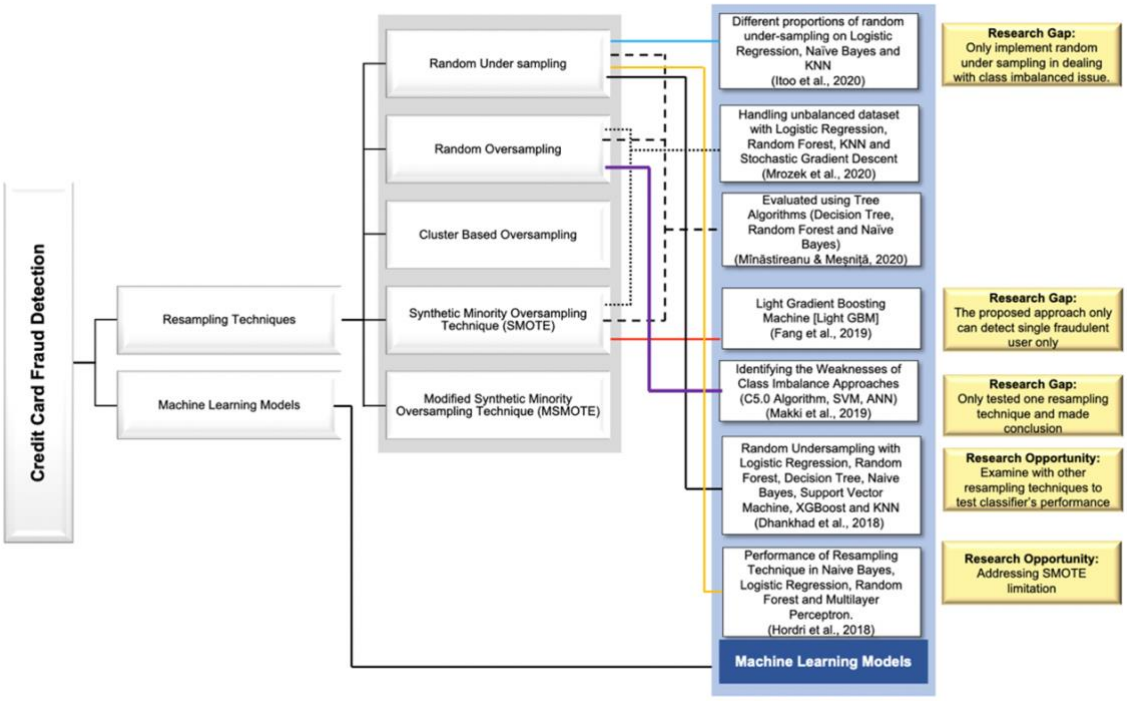


Figure 3: Summary of previous works and the gap analysis

2.3 Problem Statement

The unbalanced learning problem has piqued the interest of academics, industry, and government funding organisations in recent years. The primary issue with the imbalanced learning problem is that unbalanced data can decrease the performance of most machine learning approaches significantly. Most conventional algorithms assume or expect balanced class variations As a consequence when faced with huge imbalanced data sets, these algorithms fail to capture the data's distributive qualities sufficiently, leading to poor accuracies across data classes. Even though a lot of emphasis has gone into using machine learning techniques to avoid and identify credit card fraud, addressing the imbalance dataset remains a mystery to researchers all over the world (Alenzi & Aljehane, 2020). To date, no specific resampling approach has been proclaimed the best strategy for preventing credit card fraud (Makki et al., 2019a). While SMOTE is the most popular and widely used approach, it has the disadvantage of oversampling the data. As the SMOTE tries to learn from heavily skewed data, the majority class frequently has an impact on the classifiers, resulting in higher false-negative rates. Furthermore, SMOTE is believed to ignore the minority class's distribution as well as the noise present in the data (Hu et al., 2009). As a result, there is a pressing need to establish the most practical machine learning method that is both optimal and practicable for resolving the class imbalance problem using appropriate resampling strategies.

2.4 Research Questions

RQ1	What is the most appropriate resampling approach to efficiently tackle class imbalance data related to credit card fraud transactions?
RQ2	What are the best combination of resampling approach and machine learning algorithm for producing the best performance evaluation metrics?
RQ3	How can the suggested resampling technique help machine learning algorithms perform better?

2.5 Aim & Objectives

1. To discover and construct numerous resampling strategies that can be used to address the credit card fraud detection class imbalanced challenge.
2. To assess the implementation of the proposed resampling technique in combination with machine learning algorithms in predicting credit card fraud transactions using various resampling techniques.

2.6 Scope of Research

We hope to present a survey of current concepts of the imbalanced learning problem as well as the latest solutions to this problem in this work. This research proposes the use of the Modified SMOTE (MSMOTE) approach to address the SMOTE constraint of oversampling and latent noise in the data. By comparing each resampling technique with each other, this research will demonstrate the advantages and drawbacks of each. Furthermore, we will discuss how the proposed resampling strategy might increase the performance of classifier algorithms. Eventually, we will present the most optimum technique of the resampling technique and the machine learning algorithm for dealing with credit card fraud.

2.7 Significance of Research

In light of modern digitalization, using credit cards to conduct financial transactions at banks or other financial institutions is a common practice. Companies intend to utilize the latest technologies to offer excellent services to clients to take benefit of advanced technologies. Numerous companies and consumers profit from online payments in terms of ease, speed, and versatility in accomplishing daily tasks (Deufel et al., 2019). Switching from a manual process to a completely automated system, such as those seen in smart cities, is not without its risk. Credit card fraud is still a problem in the financial business, therefore it must be handled as soon as possible. The problem has been exacerbated by attackers' dangerously advanced capabilities, as these individuals can abuse security flaws to gain personal consumer data or their credit information to carry out criminal acts like a fraud. Every year, the banking industry has to face financial damage, and customers' dissatisfaction with their security protocols has had a significant impact. As a result, we must address the problem by addressing the core cause, which is imbalanced datasets. An unbalanced dataset is known to considerably impede the operation of many machine learning models, as the data is significantly skewed and the models are unable to operate to their maximum potential. Although several resampling approaches, including the SMOTE strategy, have been identified to alleviate this problem, it is still prone to oversampling and noise (Hu et al., 2009). As a consequence, the goal of this study is to investigate different resampling strategies, such as Modified SMOTE (MSMOTE), that could outperform previously utilised resampling techniques. Furthermore, the goal of this research is to find the ideal combination of resampling techniques and a machine learning model to be used in a credit card fraud detection framework. The study's findings will significantly benefit the community in terms of avoiding fraudulent credit card transactions and thereby decreasing damages by offering new knowledge about the class imbalance concern that credit card fraud detection system developers must solve. This research will also be used to design more effective credit card fraud detection systems in the future.

CHAPTER 3: Methodology for Handling Imbalanced Domains

3.1 Overview of Research Design

When it comes to constructing predictive models, unbalanced domains pose considerable obstacles. Due to the scarcity of representation of the most relevant cases, models prefer to concentrate on common examples, ignoring unusual events. Several solutions, mostly in a classification setting, have been proposed to overcome this issue. Figure 4 displays the proposed research flow for credit card fraud detection. The Kaggle credit card fraud dataset will be used for this research. To construct the model, the dataset will be split up into two: a training set and a testing set, with the training set going through some pre-processing and resampling before being used as an input to machine learning models. To identify the most optimum resampling technique, 3 types of resampling approaches are proposed, namely the random under-sampling, Synthetic Modified Oversampling Technique (SMOTE) and Modified Synthetic Modified Oversampling Technique (MSMOTE). Subsequently, the resampled training data will be employed in the Logistic Regression, Random Forest and XGBoost models. In the development of a machine learning-based fraud detection system, there are two steps to consider The first stage is to create a prediction model using a set of labelled historical data. This approach is known as supervised learning because the label of the transactions (legitimate or fraudulent) is provided. In the second stage, the prediction model generated through the supervised learning process is used to predict the category of new transactions. Finally, the performance of all the classifier algorithms will be evaluated using classification evaluation metrics to determine which model performs well. The model with the top performance, that is, the lowest loss on the validation set is chosen at the end of the operation and used to make predictions on future transactions.

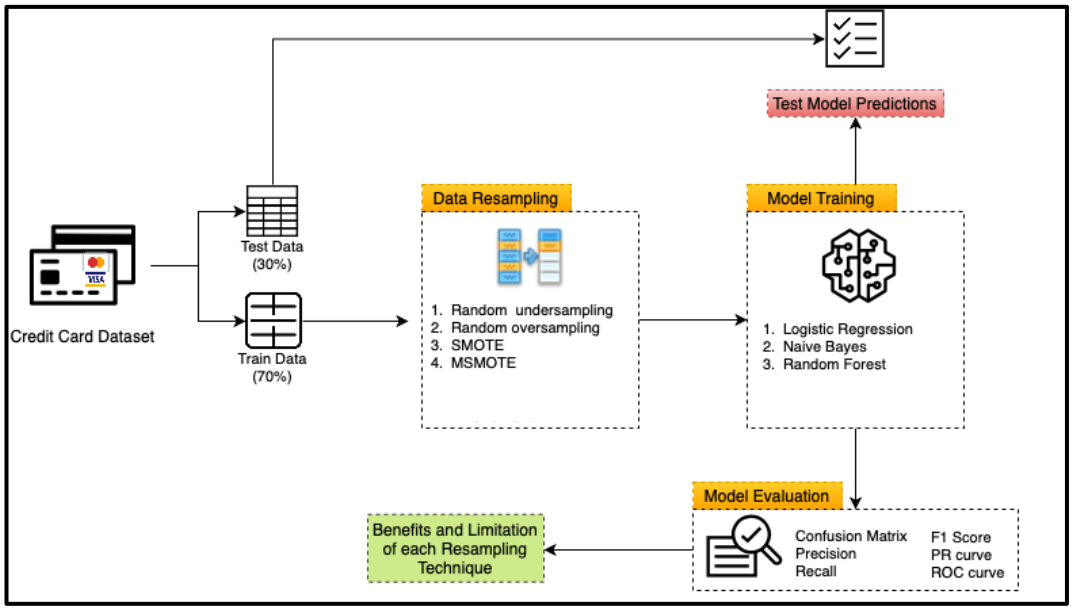


Figure 4: Research flow for credit card fraud detection

3.2 Data Source and Type

This project makes use of a standard dataset that is freely available on Kaggle, where it comprises over 284,807 records of credit card transactions of European cardholders that occurred within two days in September 2013. The dataset is highly imbalanced where it displays only 492 fraudulent transactions out of 284,807 transactions (Figure 5). The ratio is calculated simply by dividing the majority class's sample size (N_{maj}) by the minority class's sample size (N_{min}).

$$\mathrm{IR} = \frac{N_{maj}}{N_{min}}$$

The dataset is evenly distributed between the two groups if the IR value is 1. However, the dataset is unbalanced if the IR value is greater than 1 (Zhu et al., 2020). In the CCF dataset utilised in this study, the minority class accounted for only 0.172 % of all transactions.

Figure 5: Bar chart display the class imbalance (Kaur, 2021)

From the 31 columns, 28 numerical variables were named V1 to V28, and the remaining three are time, amount and class. As some of the variables carry financial information, PCA transformation was performed to keep these data private. The class variable is divided into two, where 1 is a positive class, which is fraud and 0 is the negative class, which is non-fraud. Missing values are not found in the dataset.

Data used in credit card fraud detection is generally transaction data acquired by a payment service or a bank. There are three types of transaction records (Adewumi & Akinyelu, 2017). Account-related information comprises the account number, the date the account was opened, the card limit, the card expiration date, and so on. The transaction-related information is the transaction reference number, time of the transaction, the amount of transaction, and terminal number. Lastly, the customer-related features consist of the background of the customer and the customer number. A card payment transaction, in its most basic form, is any amount paid to a merchant by a consumer at a specific moment.
Link to dataset: [https://www.kaggle.com/mlg-ulb/creditcardfraud/version/3].

3.3 Data Pre-processing

Pre-processing the data is an important step in model development. Data pre-processing is a technique for processing raw data so that a system can interpret it successfully. The real-world data is often inadequate, with missing variables. As a result, pre-processing is crucial for fixing these concerns and cleansing the data. Data pre-processing is done to increase the accuracy and quality of the data. Figure 6 depicts the stages of data pre-processing, which comprise data cleansing, data integration, data transformation, and data reduction.

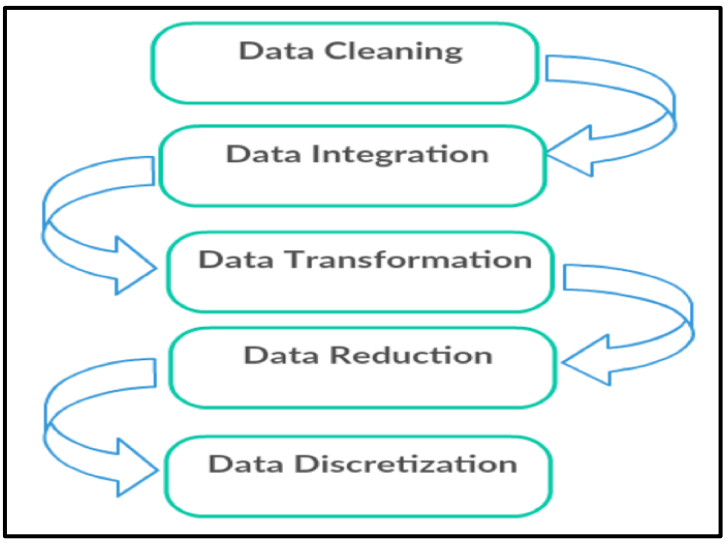


Figure 6: Stages of data pre-processing (Kaur, 2021)

Data cleaning assists in the replacement of missing values, smoothing down the noise and detecting outliers. The next step is data integration, which entails merging information from several sources into a single, uniform representation. Data transformation on the other hand comprises techniques for extracting data, evaluating it, comprehending it, and converting it into something you can analyse. Lastly, data reduction is a technique for reducing the amount of space required to store data to increase storage capacity while lowering costs.

Resampling the data to improve the dataset's data distribution, causing the model to focus on the least-represented examples, is also one of the pre-processing strategies (Branco et al., 2015). Resampling is the process of constructing a newly modified version of the training dataset with a distinctive class distribution for the selected samples to achieve a more symmetrical data distribution. Rather than using a machine learning technique straight to the given training data, we should first pre-process the data to achieve our goal of data balance.

3.3.1 Proposed Resampling Techniques

Due to the class imbalance, detecting the characteristics of fraudulent activities and establishing fraud trends is particularly challenging. The skewness of existing data sets, which are typically imbalanced, hinders the output of all machine learning datasets. This corresponds to low recall of the minority class, which is usually the class of interest (Pozzolo et al., 2015). There are numerous approaches to this issue, and we may distinguish between data-driven and algorithm-driven approaches (Chawla et al., 2004). Before implementing any method, resampling approaches are used as a pre-processing step on the data level to rebalance the two classes. These approaches need not take any particular information into account when eliminating or adding data from one class, but they are simple to use and understand (Dal Pozzolo, Caelen, et al., 2014). We will look into resampling techniques, which are a subset of data-level methods. Thus, the resampling techniques proposed in this work are over-sampling, under-sampling, SMOTE and MSMOTE. Both random under-sampling and over-sampling are called "naive resampling" approaches because they make no assumptions about the data and don't employ any heuristics. This makes them easy to design and quick to run, which is ideal for huge, complex datasets. However, due to their drawbacks, which will be discussed in the next section, One of the most prevalent and powerful oversampling techniques, SMOTE, is introduced. The SMOTE is composed of three parts: generating synthetic instances, randomising minority class examples, and determining k nearest neighbours for each minority class instance (Chawla et al., 2002).

3.3.1.1 Random Under-sampling

A common strategy for dealing with unbalanced classification issues is to under-sample the majority class in the training set before developing a classifier (Akbani et al., 2004). The random under-sampling strategy, which is simple and efficient, has been used in certain earlier research. The concept behind this technique is that there are many redundant data in the majority class and that eliminating some of them at random does not affect the approximation of the within-class distribution. It functions by eliminating certain samples from the dominant class to ensure that the minority class has the same number of samples as the majority class. Its advantages comprise reduced run time and capacity affiliated issues. However, it has the drawback of necessarily losing some crucial information that could decide the decision border between the classes.

Figure 7: Random under-sampling (Mohammed et al., 2020)

3.3.1.2 Random Oversampling

Oversampling involves randomly upsizing the minority class to reduce class imbalance (Drummond & Holte, 2003). Random oversampling requires replicating minority class instances at random and incorporating them into the training dataset. Instances are being chosen at random with replacements from the training dataset. This implies that instances belonging to the minority class will be chosen from the original dataset and added to the newly balanced dataset and will be returned or replaced in the original dataset, enabling them to be selected again.

The limitation with ROS is that biassing the model towards the minority class increases the risk of over-fitting (Pozzolo et al., 2015). To have a better understanding of the method's effects, it is a good idea to track the performance of both the training and testing datasets after resampling and compare the results before resampling.

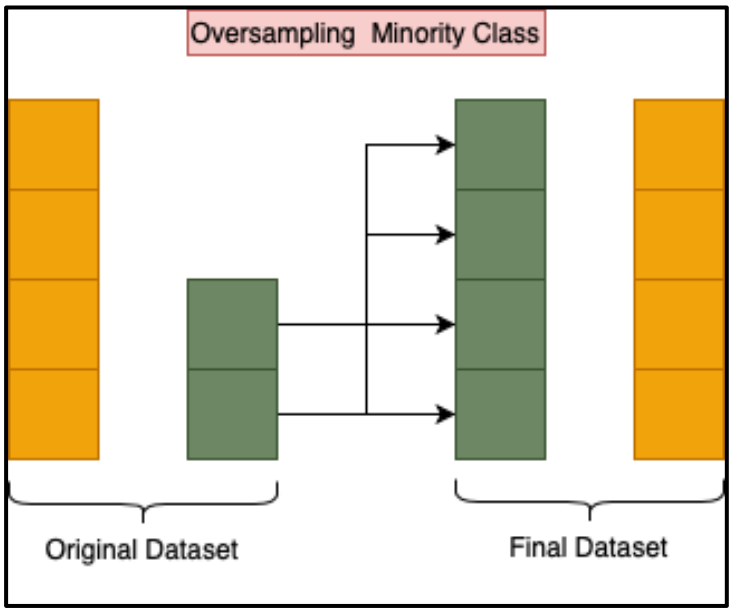


Figure 8: Random oversampling (Mohammed et al., 2020)

3.3.1.3 Synthetic Minority Oversampling Technique (SMOTE)

Another important technique for managing the imbalance problem during the pre-processing phase is to create new synthetic data. As opposed to random under-sampling, which removes the dominating class, the Synthetic Minority Over-Sampling Technique (SMOTE) selectively oversamples it.

This is accomplished via linear interpolation of a randomly selected minority observation and one of its neighbouring minorities, as shown in Figure 9. To be more exact, there are three steps to creating synthetic samples with this method. The first action is to choose a random minority observation, denoted by $\vec{a}$. The next step is to choose an instance $\vec{b}$ from among its k closest minority class neighbours. The final part is to randomly interpolate the sample to generate a new sample $\vec{x}$: $\vec{x} = \vec{a} + w \times (\vec{b} - \vec{a})$, where w is a random weight in the range [0,1].

It has been discovered that the SMOTE approach works best in a lower-dimensional area. Synthesising new data offers several well-known benefits, including reducing the possibility of overfitting, which occurs when replicas of the examples are added into the training set, and improving generalisation ability, which was hindered by over-sampling techniques (Chawla et al., 2002).

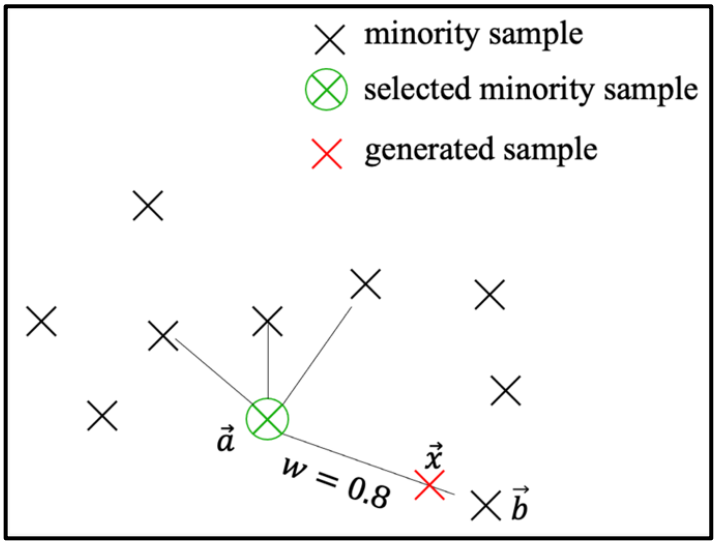


Figure 9: SMOTE technique to generate minority instances (Last et al., 2017)

3.3.1.4 Mo Technique (MSMOTE)

MSMOTE (Modified Synthetic Minority Oversampling Technique) is the improved form of SMOTE. SMOTE ignores the underlying distribution of the minority class, as well as hidden noises in the dataset. Security or safe samples, border samples, and latent nose samples are the three types of minority class samples in MSMOTE.

With the help of the security samples, the classifier can perform more effectively. Noise, on the other hand, is data that can harm a classifier's performance. Border samples are difficult to place in one of the two categories.

MSMOTE's primary operation is similar to SMOTE's. What makes it distinguishable, though, is the approach to identifying the nearest neighbours. Suppose that the sample's label in the minority class is comparable to the labels of the k nearest neighbours in MSMOTE, and thus it is included in the security sample. However, if their labels are completely different, in which case they are classified as noise. If they are neither of the preceding two, the sample is a border sample, with labels from both classes among the sample's k nearest neighbours. For this work, we have defined the MSMOTE model to be the SVMSMOTE. It is a variant of SMOTE algorithm which uses an SVM algorithm to detect samples to use for generating new synthetic samples.

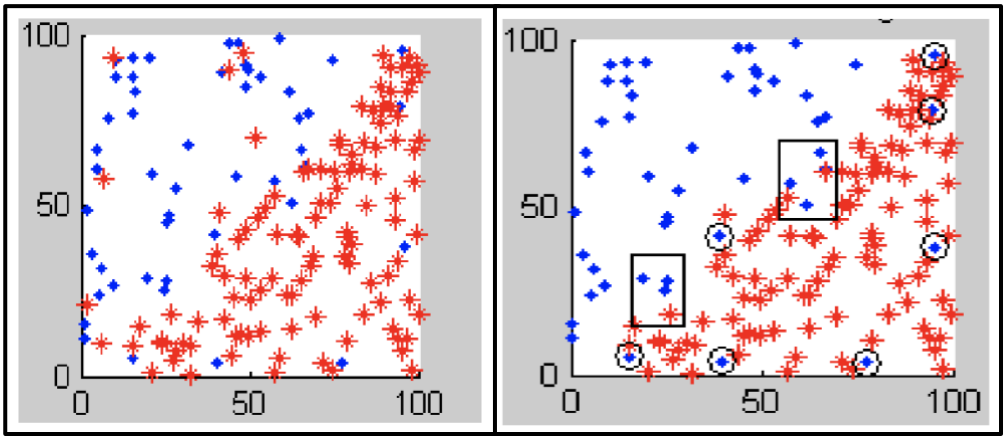


Figure 10: 1) Distribution of imbalanced data 2) Noise samples being deleted in the majority class (Hu et al., 2009)

Figure 10 shows that the blue points (minority class) in the rectangle are recognized as border samples, whilst those in the circle are recognized as noise.

3.4 Proposed Machine Learning Models

Machine Learning (ML) is the application of Artificial Intelligence principles that allow machines to learn on their own (Dhankhad et al., 2018). Statistics, Pattern Recognition, and

Data Mining are all strongly related to machine learning. Simultaneously, it appears as a discipline of computer science and artificial intelligence that focuses on the algorithmic aspects of knowledge extraction. Without being directly instructed, the machine can simply make use of the previously acquired data and examine it further. In the subject of fraud detection, where data extraction from big datasets is essential, machine learning approaches are becoming widely used (Sailusha et al., 2020b). In the recent decade, ML approaches' capacity to efficiently solve the difficulties highlighted by CCFD has resulted in a substantial and increasing corpus of research. Thousands of publications on this issue were published between 2010 and 2020, as seen in Figure 11, with around 1500 papers released in 2020 alone.

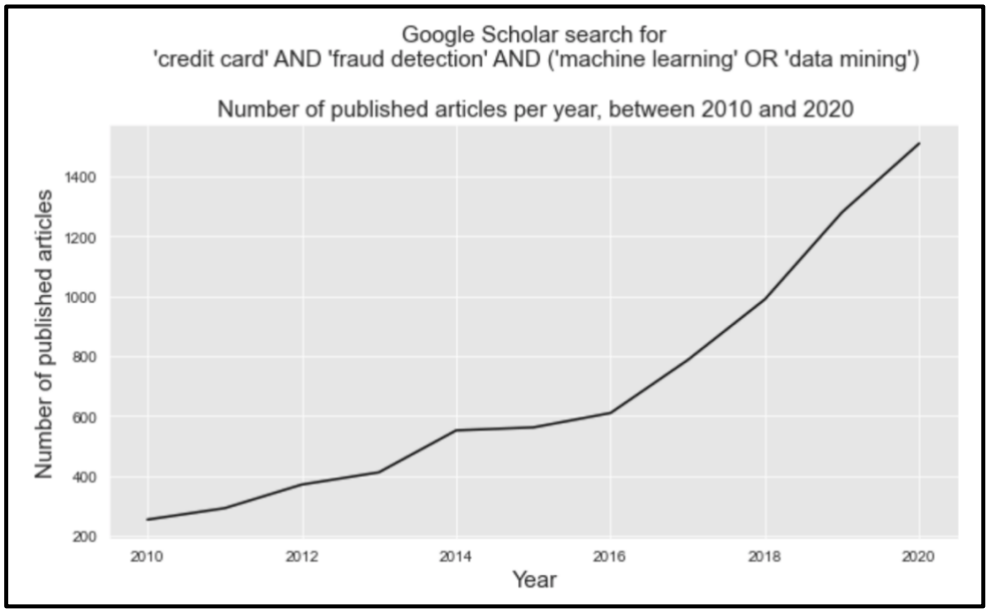


Figure 11: Amount of published articles on credit card fraud using machine learning between the years 2010 and 2020 (*Borgne*, 2012)

It is possible to train the system to recognise fraudulent and non-fraudulent transactions. As a result, machine learning algorithms can be used advantageously in the banking sector to tackle potentially risky transactions (He et al., 2018). Many strategies have been devised and are still being designed. Various detecting approaches are being developed in an attempt to solve this problem as quickly as possible (Ngai et al., 2011). The machine learning algorithms proposed for this work to identify credit card fraud are briefly described in this chapter.

3.4.1 Logistic Regression

In comparison to linear regression, logistic regression is more advance. The reason being when the data crosses, the algorithm is unable to separate it into two distinct classes. Logistic regression overcomes this problem. For the goal of illustrating this shortcoming, Fig.

7 shows a graphical depiction of the linear regression and logistic regression models. The logistic regression was extensively employed by statisticians to describe the properties of population expansion in ecology, and it was also referred to as the sigmoid function (Bowlee, 2016). It has an S-shaped curve that can accept any real-valued integer and assign it to 0 and 1 values (Figure 8). If the outcome of the functions is greater than 0.5, it is classified as '1'. If the result is less than 0.5, however, it is categorised as '0.' The linear regression equation can be used to derive the logistic regression formula. The straight-line equation is expressed as:

$$y = a_o + a_1 \times x_1 + a_2 \times x_2 + \ldots a_k \times x_k \quad (1)$$

The value of y in logistic regression ranges between 0 and 1. The equation above can be divided by (1 y):

$$\frac{y}{1-y} \; |0 \; for \; y = 0 \; and \; \infty \; for \; y = 1 \quad (2)$$

Thus, the logistic regression equation is written as:

$$\log \frac{y}{1-y} = a_0 + a_1 \times x_1 + a_2 \times x_2 + \cdots a_k \times x_k \quad (3)$$

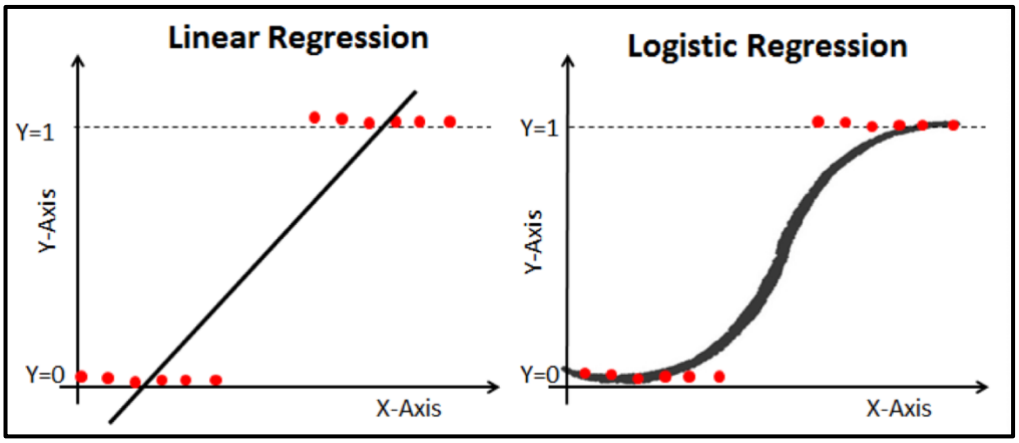


Figure 12: Logistic Regression (Navlani, 2017)

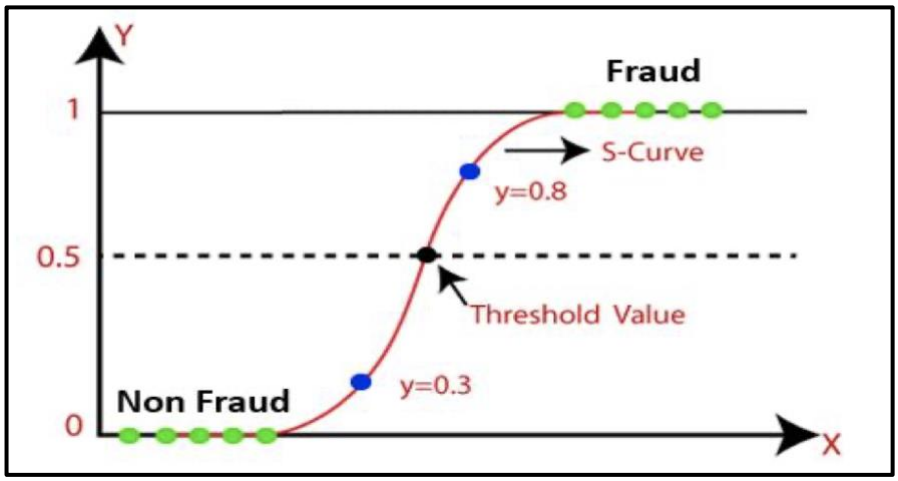


Figure 13: The concept of logistic regression classification (Java Point, 2018)

The following are some of the benefits of logistic regression:

1) Logistic regression is both simpler to use and more efficient to train than linear regression.
2) Logistic regression does not require any assumptions on how classes are distributed in the feature space.
3) Logistic regression is simpler to adapt to several classes.

3.4.2 Random Forest

Decision tree models have a strong reputation. Quinlan, (1986) attributes their mathematical simplicity and versatility in processing diverse types of data to their success in data mining. However, single-tree models, on the other hand, are susceptible to overfitting and may be sensitive to certain training data (Bhattacharyya et al., 2011). Single classifiers are seen to be less dependable than ensemble techniques, which combine several individual judgments in a specific way to solve similar difficulties (Dietterich & Oregon, 1996). Leo Breiman created random forests in the 2000s as a way of constructing a predictive ensemble with several decision trees that grow in randomly chosen subspaces of data (Biau, 2012). A random forest's performance is governed not just by the strength of individual trees, but also by their association. The higher the strength of a single tree and the lower the correlation between multiple trees, the greater the random forest's performance. When used as a classifier, the Random Forest uses the majority vote to determine the predicted class. In a tree-based model, the given dataset is successively partitioned into two divisions based on a specific set of criteria until a predefined stopping condition is satisfied. Each tree is trained by randomly sampling a

subset of the training data (Niveditha et al., 2019). Each tree in the collection is created using Breiman's approach, which involves selecting a small number of variables to split at random at each node and then determining the best split based on these features in the training set (Breiman et al., 1984). The CART technique is used to grow the tree to maximum size with no pruning.

The Random Forest output is non-biased because it is built on several decision trees, meaning that the results are more reliable. Instead of using the original sample, the separate trees are based on bootstrap samples. This is known as bootstrap aggregating, or simply bagging, and it is used to minimize overfitting in decision tree models. They're easy to use, produce high-quality predictions, and can handle a large number of input variables.

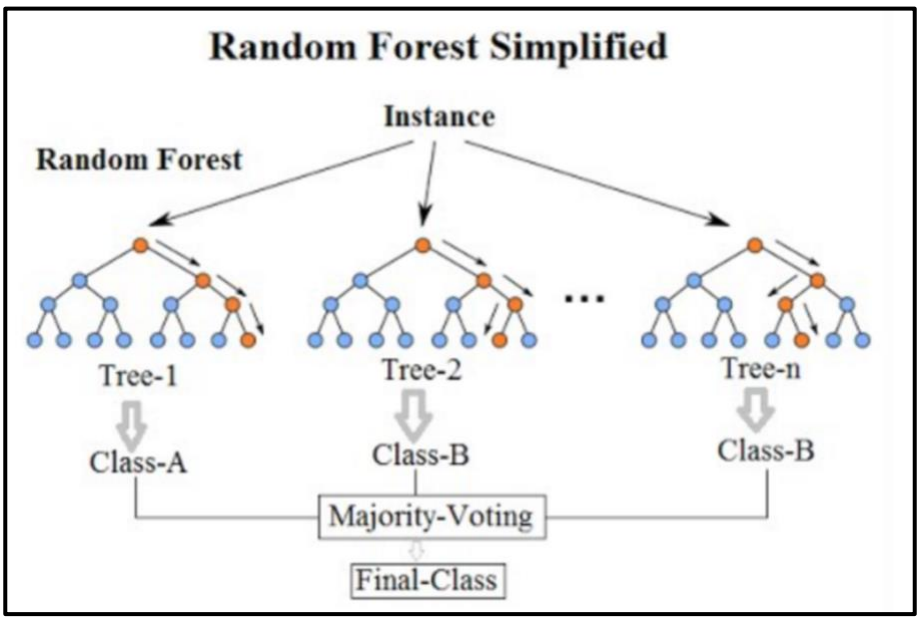


Figure 14: Random Forest (Koehrsen, 2017)

3.4.3 XGBoost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for e**X**treme **G**radient **B**oost. The beauty of this powerful algorithm lies in its scalability, which drives fast learning through parallel and distributed computing and offers efficient memory usage. XGBoost is an ensemble learning method. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models. The models that form the ensemble, also known as base learners, could be either from the same learning algorithm or different learning algorithms. Bagging and boosting are two widely used ensemble learners. While these two techniques can be used with several statistical models, the most predominant usage has been with decision trees. While decision trees are one of the most easily interpretable models, they exhibit highly variable behaviour. Bagging or boosting aggregation helps to reduce the variance in any learner. Several decision trees which are generated in parallel, form the base learners of the bagging technique. The final prediction is the averaged output from all the learners. In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

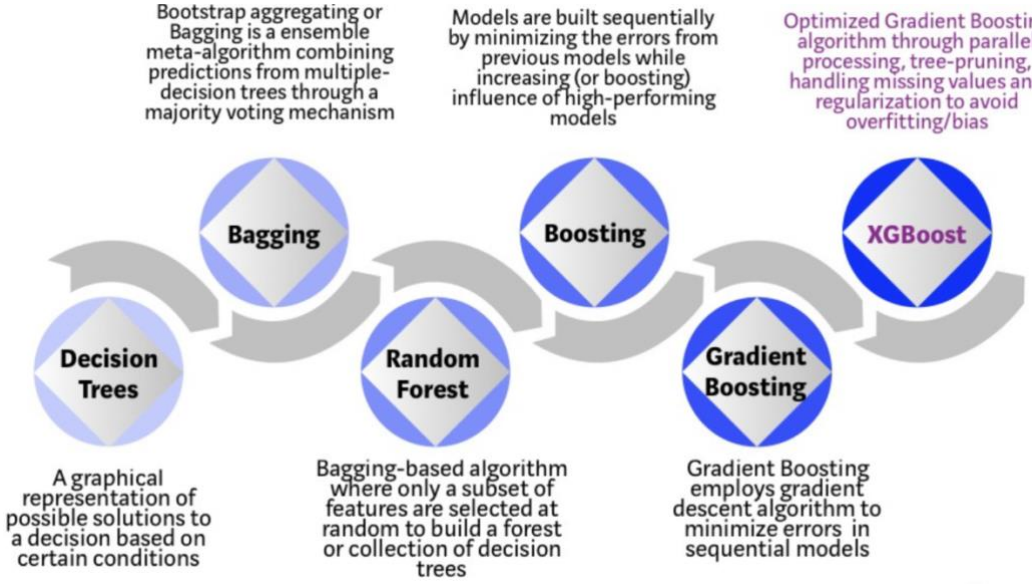


Figure 15: Evolution of XGBoost model from Decision Tree

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

3.5 Proposed Evaluation Metrics For Model Validation

The evaluation of a model is a crucial aspect of its development. Appropriate metrics should not only allow users to evaluate models based on their preferences but should also be utilized to guide model learning. Accuracy' has been the most commonly used metric. This will indicate how each model performed and allow us to determine which algorithms produce satisfactory or unsatisfactory outcomes. High accuracy does not imply that the model is more accurate in all circumstances. It is not always seen to be accurate, as it can be deceiving in some cases, such as with imbalanced class datasets. Thus, several assessment metrics such as sensitivity, precision, recall, and F1- score, associated with classification problems will be included, all of which are listed below as ways to assess a model's performance. The metrics of each model will be presented based on how well they executed with our original, under-sampling, and oversampling datasets, and then we will provide a comparative analysis to see which of our models performed best at predicting credit card fraud.

3.5.1 Confusion Matrix

In imbalanced classification scenarios, the majority class is frequently referred to as the negative outcome, whereas the minority class is frequently referred to as the positive outcome. A confusion matrix has been suggested as an appropriate measure for evaluating the performance of a classifier system in evaluating the validity of credit card transactions in numerous studies. It shows which classes are correctly categorized, which classes are incorrectly classified, and what types of errors are committed. A confusion matrix or a two-class problem is depicted in Figure 11 as an example (Draelos, 2019).

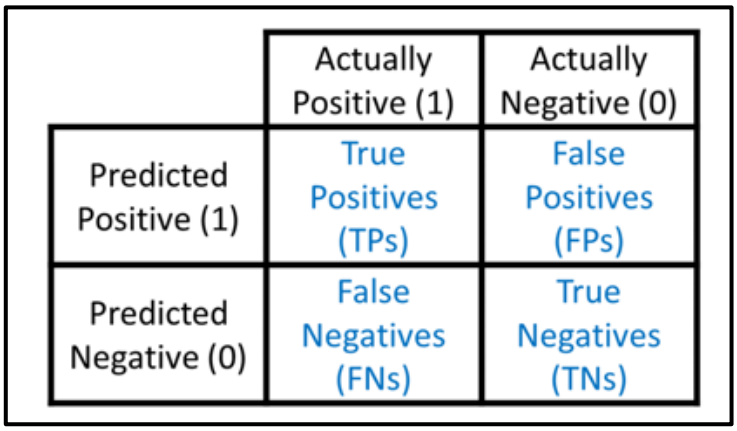


Figure 16: Overview of the confusion matrix (Draelos, 2019)

A confusion matrix is comprised of these four elements:

True Positive (TP): The model classified the observation as positive because it is positive. The transaction is expected to be fraudulent, and it is. **True Negative (TN)**: The model identified the observation as negative since it is negative. The transaction is anticipated to be non-fraudulent, and it is not.

False Positive (FP): The model identified the observation as positive even though it is negative. The transaction is projected to be fraudulent, yet it is not.

False Negative (FN): The model classified the observation as negative even though it is positive. It is projected that the transaction will be non-fraudulent, but it is a fraud.

3.5.2 Accuracy

Represents the rate percentage of correctly classified samples. It emphasizes the true positive and true negative results. When examining a bias towards the minority (positive) class examples, accuracy should not be the only metric focused on because the influence of the least represented class is extremely less when compared to the majority class. The accuracy of a two-class problem can be characterised as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \text{ x } 100 \quad (6)$$

3.5.3 Sensitivity

Measures the amount of actual positive cases that are predicted as positive or true positive. Sometimes referred to as Recall. The more false negatives the model predicts, the lower the recall. A high recall means that the model can correctly identify relevant outcomes without incorrectly identifying them as irrelevant. The total number of true positives and false negatives is divided by the number of true positives as shown below.

$$Sensitivity\ (Recall) = \frac{TP}{TP + FN} \quad (7)$$

3.5.4 Specificity

Interpreted as the accuracy of negative cases that are classified correctly which is the accuracy on predicting legitimate transactions classifications. Also known as True Negative Rate (TNR).

$$Specificity = \frac{TN}{FP + TN} \quad (8)$$

3.5.5 Precision

Precision represents how precise the model is in calculating the number of correct positive results over the number of positive classifications predicted by the model. High precision means a large number of relevant results are produced with a small number of irrelevant ones.

$$Precision = \frac{TP}{TP+FP} \tag{9}$$

3.5.6 F1- Score

The F1 score is mostly used to assess the accuracy of the model. It allows for a report on how accurate and powerful the classification is. If a result has high precision but a low recall, it implies we have exceptionally high accuracy, but keep in mind that it may miss a large number of difficult-to-classify options. A high value of F1-Score indicates that the model is reliable and has good performance. Mathematically, it can be expressed as:

$$F1 - Score = \frac{2*(Precision*Recall)}{Precision+Recall} \tag{10}$$

3.5.7 Precision-Recall (PR) curve

As the PR curve is more sensitive than the ROC curve, it is a better measure of performance. Precision-recall curves (PR curves) are recommended for highly skewed circumstances where ROC curves may provide an overly positive perception of performance (Davis & Goadrich, 2006). Precision and recall are both concerned with the positive class (the minority class) and ignore the negatives (the majority class) (Brownlee, 2020). The precision rate is plotted against the recall rate to get this curve. The use of precision rather than FPR (as with the ROC curve) permits the method to detect the impact of major negative samples on its performance. Although generally employed for classifiers, ROC can conceal weak results in an imbalanced classification. The proximity of the curve to the upper right-hand corner determines the quality of the model in the PR plot. The Area within the PR curve can be used to determine this (AUPRC).

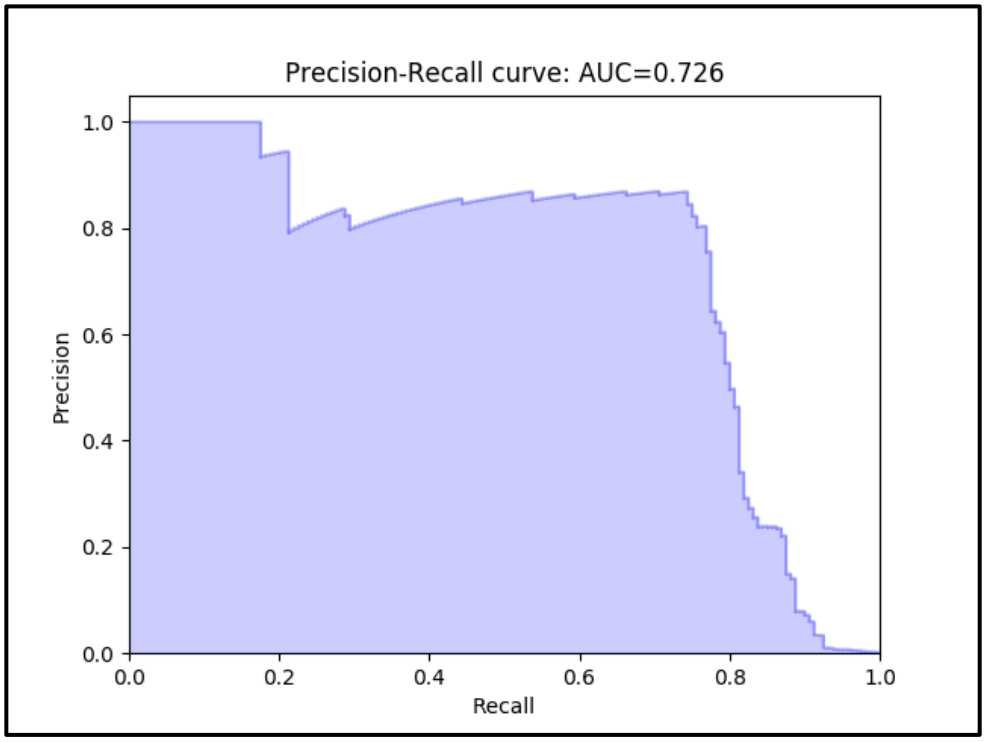


Figure 17: Example of a PR curve

3.6 Exploratory Data Analysis (EDA)

3.6.1 Importing the Libraries

This project has employed a variety of libraries for various purposes. To import the data into a Dataframe object, we have to utilise the pandas package. Numpy, on the other hand, is a general-purpose array-preprocessing package that includes a high-performance multidimensional array object as well as features for managing them. Plotting was accomplished using the matplotlib and seaborn libraries. Some data preprocessing, model development and model evaluation was done with the sklearn library. Finally, when working with classification involving imbalanced classes, the imblearn library was implemented.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from skimpy import skim
import time
import missingno as msno
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from datetime import datetime
from sklearn import metrics
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import SVMSMOTE
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint


import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Figure 18: Libraries imported for this work

3.6.2 Performing Exploratory Data Analysis

It is important to study the data we are working with before using it to train the three machine learning algorithms. Exploratory data analysis refers to the process of finding the structure of a dataset, such as the number of rows and columns, recognizing the types of data objects in each column, detecting missing values, evaluating correlation values, and so on. Figure 19 depicts the descriptive statistics for fraudulent and non-fraudulent transactions. Skimpy is a lightweight programme that also offers summary statistics for datagram variables. We will begin by reviewing the data before beginning EDA operations. There are 284807 rows and 31 columns, as we can see.

skimpy summary

Data Summary

dataframe	Values
Number of rows	284807
Number of columns	31

Data Types

Column Type	Count
float64	30
int64	1

number

	missing	complete rate	mean	sd	p0	p25	p75	p100	hist
Time	0	1	95000	47000	0	54000	140000	170000	
V1	0	1	3.9e-15	2	-56	-0.92	1.3	2.5	
V2	0	1	5.7e-16	1.7	-73	-0.6	0.8	22	
V3	0	1	-8.8e-15	1.5	-48	-0.89	1	9.4	
V4	0	1	2.8e-15	1.4	-5.7	-0.85	0.74	17	
V5	0	1	-1.6e-15	1.4	-110	-0.69	0.61	35	
V6	0	1	2e-15	1.3	-26	-0.77	0.4	73	
V7	0	1	-1.7e-15	1.2	-44	-0.55	0.57	120	
V8	0	1	-1.9e-16	1.2	-73	-0.21	0.33	20	
V9	0	1	-3.1e-15	1.1	-13	-0.64	0.6	16	
V10	0	1	1.8e-15	1.1	-25	-0.54	0.45	24	
V11	0	1	9.3e-16	1	-4.8	-0.76	0.74	12	
V12	0	1	-1.8e-15	1	-19	-0.41	0.62	7.8	
V13	0	1	1.7e-15	1	-5.8	-0.65	0.66	7.1	
V14	0	1	1.5e-15	0.96	-19	-0.43	0.49	11	
V15	0	1	3.5e-15	0.92	-4.5	-0.58	0.65	8.9	
V16	0	1	1.4e-15	0.88	-14	-0.47	0.52	17	
V17	0	1	-7.5e-16	0.85	-25	-0.48	0.4	9.3	
V18	0	1	4.3e-16	0.84	-9.5	-0.5	0.5	5	
V19	0	1	9e-16	0.81	-7.2	-0.46	0.46	5.6	
V20	0	1	5.1e-16	0.77	-54	-0.21	0.13	39	
V21	0	1	1.5e-16	0.73	-35	-0.23	0.19	27	
V22	0	1	8e-16	0.73	-11	-0.54	0.53	11	
V23	0	1	5.3e-16	0.62	-45	-0.16	0.15	23	
V24	0	1	4.5e-15	0.61	-2.8	-0.35	0.44	4.6	
V25	0	1	1.4e-15	0.52	-10	-0.32	0.35	7.5	
V26	0	1	1.7e-15	0.48	-2.6	-0.33	0.24	3.5	
V27	0	1	-3.7e-16	0.4	-23	-0.071	0.091	32	
V28	0	1	-1.2e-16	0.33	-15	-0.053	0.078	34	
Amount	0	1	88	250	0	5.6	77	26000	
Class	0	1	0.0017	0.042	0	0	0	1	

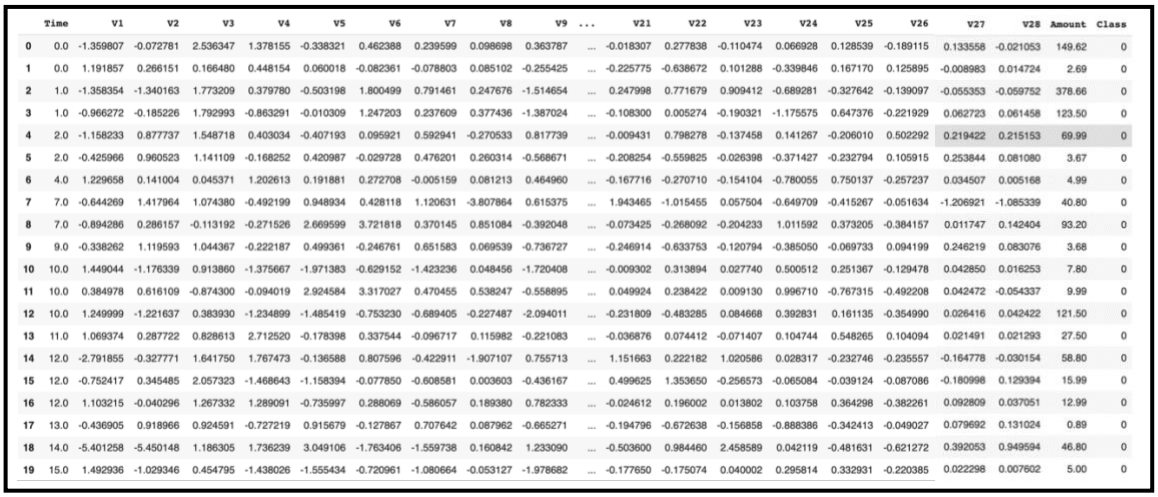
End

Figure 19: Skimpy summary of the dataset

By default, the head() function in Python shows the first five rows of the data frame. This parameter can be used to display n rows in our dataset. When 'n' is set to 20, the first 20 rows of data are returned. We can see that the features 'Time' and 'Amount' differ significantly from the rest, thus we'll need to standardise them before generating the model during data pre-processing.

```
# Loading the data
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/CreditCardFraud/creditcard.csv')
df.head(20)
```

Figure 20: Code to load the data and get an overview of the dataset



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0
10	10.0	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236	0.048456	-1.720408	...	-0.009302	0.313894	0.027740	0.500512	0.251367	-0.129478	0.042850	0.016253	7.80	0
11	10.0	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455	0.538247	-0.558895	...	0.049924	0.238422	0.009130	0.996710	-0.767315	-0.492208	0.042472	-0.054337	9.99	0
12	10.0	1.249999	-1.221637	0.383930	-1.234899	-1.485419	-0.753230	-0.689405	-0.227487	-2.094011	...	-0.231809	-0.483285	0.084668	0.392831	0.161135	-0.354990	0.026416	0.042422	121.50	0
13	11.0	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717	0.115982	-0.221083	...	-0.036876	0.074412	-0.071407	0.104744	0.548265	0.104094	0.021491	0.021293	27.50	0
14	12.0	-2.791855	-0.327771	1.641750	1.767473	-0.136588	0.807596	-0.422911	-1.907107	0.755713	...	1.151663	0.222182	1.020586	0.028317	-0.232746	-0.235557	-0.164778	-0.030154	58.80	0
15	12.0	-0.752417	0.345485	2.057323	-1.468643	-1.158394	-0.077850	-0.608581	0.003603	-0.436167	...	0.499625	1.353650	-0.256573	-0.065084	-0.039124	-0.087086	-0.180998	0.129394	15.99	0
16	12.0	1.103215	-0.040296	1.267332	1.289091	-0.735997	0.288069	-0.586057	0.189380	0.782333	...	-0.024612	0.196002	0.013802	0.103758	0.364298	-0.382261	0.092809	0.037051	12.99	0
17	13.0	-0.436905	0.918966	0.924591	-0.727219	0.915679	-0.127867	0.707642	0.087962	-0.665271	...	-0.194796	-0.672638	-0.156858	-0.888386	-0.342413	-0.049027	0.079692	0.131024	0.89	0
18	14.0	-5.401258	-5.450148	1.186305	1.736239	3.049106	-1.763406	-1.559738	0.160842	1.233090	...	-0.503600	0.984460	2.458589	0.042119	-0.481631	-0.621272	0.392053	0.949594	46.80	0
19	15.0	1.492936	-1.029346	0.454795	-1.438026	-1.555434	-0.720961	-1.080664	-0.053127	-1.978682	...	-0.177650	-0.175074	0.040002	0.295814	0.332931	-0.220385	0.022298	0.007602	5.00	0

Figure 21: First 20 rows of the dataset

In most datasets, missing values are expressed by Nan, null, or None. df.isnull(). The sum function could also be used to find missing values in a data set. This will give us the labels of the columns as well as the amount of non-values in each column. We discovered that there were no null data in the data after executing the algorithm below.

```
print(df.isnull().sum())
```

Figure 22: Code to find missing values

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Figure 23: Missing values output

The missingno package, which provides an excellent means to visualise the frequency of NaN values using a bar chart, is the second way of determining if we have null values in the data. The bar graph shows how much missing data are there in each column. We found no missing values once again.

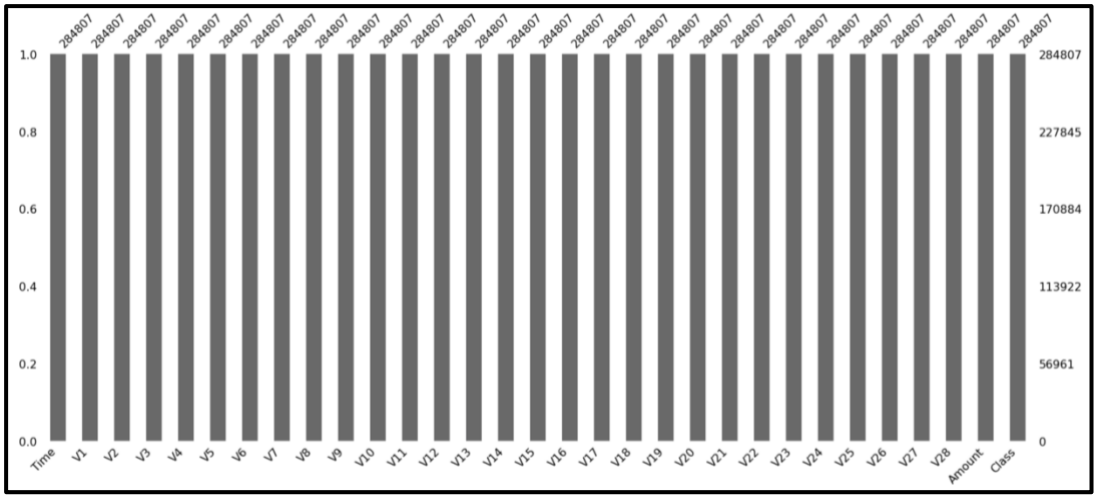


Figure 24: Visualization of the missing values in dataset

Before proceeding into with the deployment of predictive analysis, it is advisable to view the data beforehand. The data can be visualised in terms of correlation between variables, where correlation corresponds to the variables' mutualistic relationship. During the training stage, variables with a stronger correlation with the target variable have a greater effect. Figure 25 depicts a correlation matrix that explains the pairwise association between each variable. According to the correlation matrix, none of the V1 through V28 principal components are related to each other. However, if we look closer, we can see that the 'Amount' variable has a -0.53 negative correlation with the principal component 'V2.'

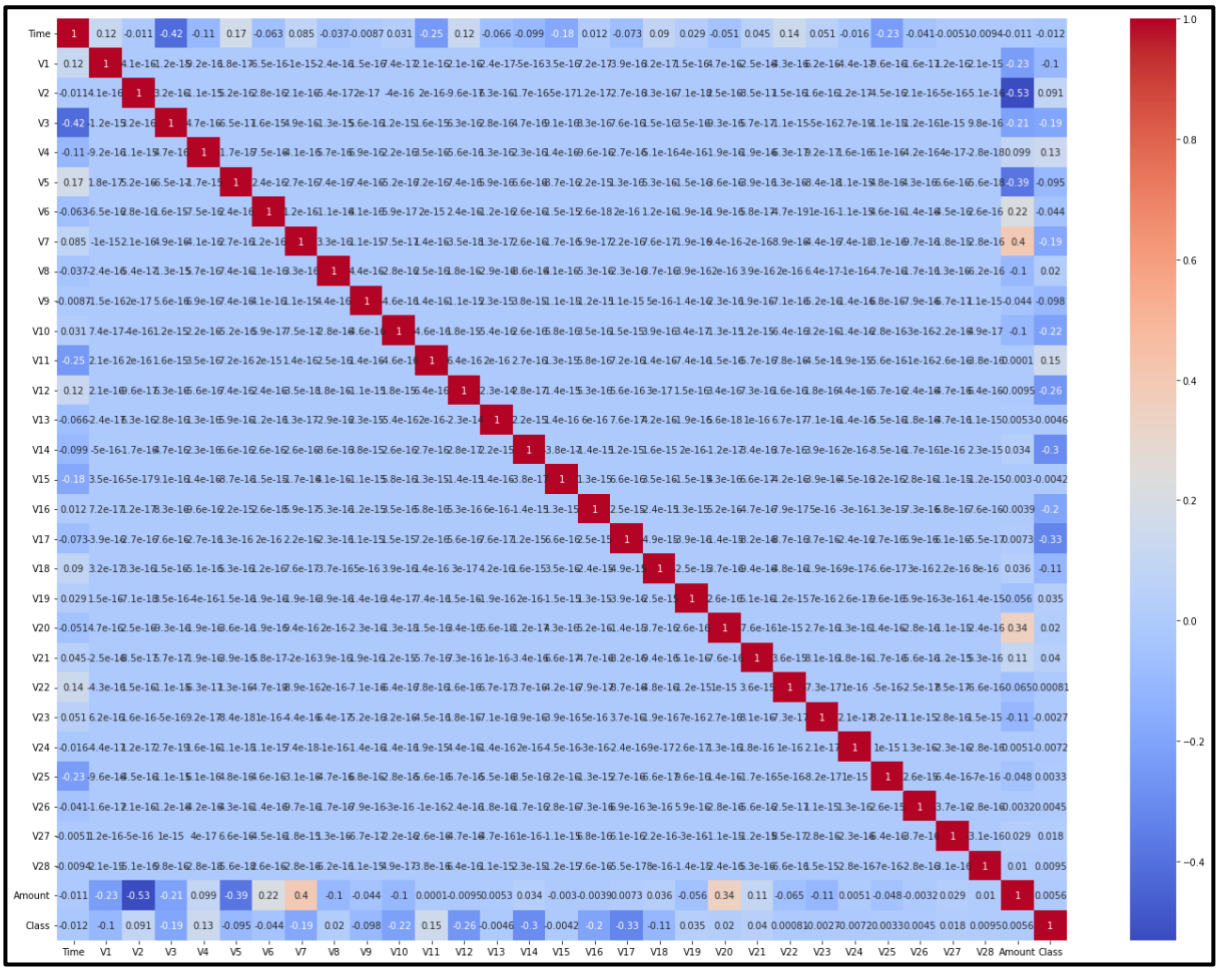


Figure 25: Correlation plot

The dataset seems to be skewed, with fraudulent transactions accounting for only 0.17% of the transactions. Unbalanced datasets can cause bias in machine learning models, hence they should be treated as such.

```
# Checking the class distribution of the target variable in percentage
print((df.groupby('Class')['Class'].count()/df['Class'].count()) *100)
((df.groupby('Class')['Class'].count()/df['Class'].count()) *100).plot.pie()
```

Figure 26: Code to check the class distribution of the target class in percentage

```
Class
0    99.827251
1     0.172749
Name: Class, dtype: float64
<matplotlib.axes._subplots.AxesSubplot at 0x7f168c362090>
```

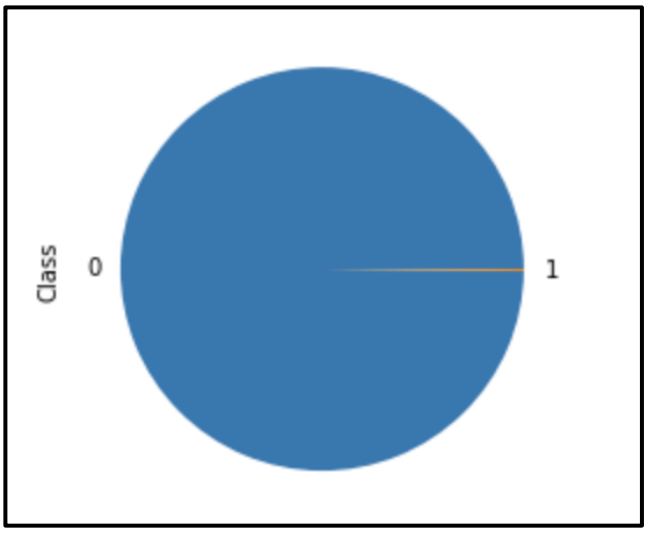


Figure 27: Imbalanced class distribution

3.7 Data Pre-processing

3.7.1 Remove duplicates

The dataset contains no null values by default. However, it contained 1081 duplicate values. We were able to eliminate the duplicate values during data pre-processing.

```
[ ] df.duplicated(keep='first').sum()

    1081

[ ] df.drop_duplicates(keep='first', inplace=True)

[ ] df.duplicated().sum()

    0
```

Figure 28: Code to find and drop duplicates in the dataset

Using the skim function to remove the duplicated values, we discovered that the dataset now has 283726 rows (figure 29). We also discovered that the non-fraud class now has 283253 rows while the fraud class had 473 rows (figure 31).

Data Summary

dataframe	Values
Number of rows	283726
Number of columns	31

Figure 29: Data summary after dropping duplicates

```
nonfraud = df[df.Class==0]
fraud = df[df.Class==1]

print("Amount stats for non fraud class")
print(nonfraud.Amount.describe())

print("\nAmount stats for fraud class")
print(fraud.Amount.describe())
```

Figure 30: Code to obtain the descriptive analysis of the target class

```
Amount stats for non fraud class
count    283253.000000
mean         -0.000236
std           0.999920
min          -0.353327
25%          -0.330683
50%          -0.265467
75%          -0.043981
max         102.247564
Name: Amount, dtype: float64

Amount stats for fraud class
count    473.000000
mean       0.141371
std        1.039186
min       -0.353327
25%       -0.349333
50%       -0.314109
75%        0.069558
max        8.136603
Name: Amount, dtype: float64
```

Figure 31: Descriptive analysis of the target variable

3.7.2 Standardization

A basic criterion for many machine learning algorithms is dataset standardisation. Rescaling the range of data so that the mean of observed data is 0 and the standard deviation is 1 is the process of standardising a dataset. This is similar to removing the mean value or centring the data. Our observations should fit a Gaussian distribution with a well-behaved mean and standard deviation, according to standardisation. The StandardScaler function is provided by the preprocessing module, and it is a fast and simple way to conduct the following operation on an array-like dataset. The equation for standardization is shown below:

y = (x-mean) / standard_deviation
where the mean = sum(x) / count(x) and
standard_deviation = sqrt(sum((x-mean)^2) / count(x))

The fit transform() method alters the data points while also fitting and transforming the input data. It calculates the mean and standard deviation at the same time as it changes the variable's data points. Because their values differed substantially from the others, we just need to scale the 'Time' and 'Amount' variables.

```
# Values of amount variable varies alot, thus we normalise it
sc = StandardScaler()

amt = df['Amount'].values

df['Amount'] = sc.fit_transform(amt.reshape(-1,1))

df['Amount']
```

Figure 32: Code to scale the 'Amount' variable

```
0         0.244200
1        -0.342584
2         1.158900
3         0.139886
4        -0.073813
            ...
284802   -0.350252
284803   -0.254325
284804   -0.082239
284805   -0.313391
284806    0.513290
Name: Amount, Length: 283726, dtype: float64
```

Figure 33: Output of the scaled 'Amount' variable

```
time = df['Time'].values

df['Time'] = sc.fit_transform(time.reshape(-1,1))

df['Time']
```

Figure 34: Code to scale the 'Time' variable

```
0        -1.996823
1        -1.996823
2        -1.996802
3        -1.996802
4        -1.996781
            ...
284802    1.642235
284803    1.642257
284804    1.642278
284805    1.642278
284806    1.642362
Name: Time, Length: 283726, dtype: float64
```

Figure 35: Output of the scaled 'Time' variable

3.7.3 Feature Engineering

The 'Time' feature, on the other hand, indicates the number of seconds that already have elapsed between each transaction and the dataset's initial transaction. The time column gives very little valuable insight because we do not have a launch time for the first transaction. As a result, we have added a new column called 'Time Hour' to the dataset and removed the previous 'Time' feature.

```
# As time is given in relative fashion, we are using pandas.Timedelta which Represents a duration,
#the difference between two times or dates.
Delta_Time = pd.to_timedelta(df['Time'], unit='s')

#Create derived columns Mins and hours
df['Time_Day'] = (Delta_Time.dt.components.days).astype(int)
df['Time_Hour'] = (Delta_Time.dt.components.hours).astype(int)
df['Time_Min'] = (Delta_Time.dt.components.minutes).astype(int)

# Drop unnecessary columns
# We will drop Time,as we have derived the Day/Hour/Minutes from the time column
df.drop('Time', axis = 1, inplace= True)
# We will keep only derived column hour, as day/minutes might not be very useful
df.drop(['Time_Day', 'Time_Min'], axis = 1, inplace= True)
```

Figure 36: Code to transform the 'Time' variable and drop unnecessary columns

The approach must be unbiased before evaluating our model's predicted performance. We have split the dataset into 80% training and 20% testing using train test split() from the scikit-learn module to reduce the risk of skew in our assessment and validation process. The training set is used to train and fit our models, while the test set is used to evaluate the final model objectively. The parameter that determines randomization during splitting is the 'random state' feature. To ensure that our tests are repeatable, the random split must produce the same results each time the programme is run. As a result, we have set the random state to 100. The data is initially separated into the x and y predictors. x contains 283726 data records with 30 features each while y contains 283726 data records with one column, which is the target class.

3.7.4 Data Splitting

In this section, we have given x the task of carrying all the independent variables and y the task of carrying only the target variable, 'Class.' The data was then split into x train, x test, y train, and y test using the train_test_split function, with the training set receiving 80% of the data and the test set receiving the remainder.

```
# Splitting the dataset into X and y
y= df['Class']
x = df.drop(['Class'], axis=1)
```

Figure 37: Code to split the data into x and y

```
# Splitting the dataset using train test split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=100, test_size=0.20)
```

Figure 38: Code to split the data into train and test

3.7.5 Data Resampling

Since this data is significantly skewed, if we apply it to train our algorithm, the model will be biased toward valid transactions, resulting in poor simulation results when evaluated. We employed resampling approaches such as random under-sampling, random over-sampling, SMOTE, and SVMSMOTE to solve this problem.

3.7.5.1 Random Oversampling

Random Oversampling (ROS) generally increases the amount of minority class samples until they reflect a balanced number of instances in the training data as compared to the majority class samples. We can see that the minority class has 226620 samples, while the majority class has 226620.

```
# define oversampling strategy
oversample = RandomOverSampler(sampling_strategy='minority', random_state=10)

x_over, y_over = oversample.fit_resample(x_train, y_train)
print("RANDOM OVERSAMPLING RESULTS:")
y_over.value_counts()

RANDOM OVERSAMPLING RESULTS:
0    226620
1    226620
Name: Class, dtype: int64
```

Figure 39: Code to run random oversampling

3.7.5.2 Random Under-sampling

Random under-sampling is a method for removing the bulk of observations from training data. It was discovered that by reducing the size of the majority class, we can decrease the amount of training computation time needed to develop the model. We can see that the majority class now has 360 samples, which is similar to the minority class.

```
# define undersample strategy
undersample = RandomUnderSampler(sampling_strategy='majority', random_state=20)

# fit and apply the transform
x_under, y_under = undersample.fit_resample(x_train, y_train)
print("RANDOM UNDERSAMPLING RESULTS:")
y_under.value_counts()

RANDOM UNDERSAMPLING RESULTS:
0    360
1    360
Name: Class, dtype: int64
```

Figure 40: Code to run random undersampling

3.7.5.3 SMOTE

SMOTE develops a new instance mostly by interpolating preexisting minority samples that are nearby each other. It chooses one or more k nearest neighbours of any initial sample xi arbitrarily and conducts interpolation of the original sample and its neighbour to generate a new sample. The minority class has the same amount of samples as the majority class, as can be shown below.

```
# define SMOTE strategy
SMOTE = SMOTE(random_state=30)

# fit and apply the transform
x_SMOTE, y_SMOTE = SMOTE.fit_resample(x_train, y_train)
print("SMOTE RESULTS:")
y_SMOTE.value_counts()

SMOTE RESULTS:
0    226620
1    226620
Name: Class, dtype: int64
```

Figure 41: Code to run SMOTE

3.7.5.4 SVMSMOTE

Another over-sampling strategy is SVMSMOTE, which prioritises on producing additional minority class samples near borderlines using SVM to define a class boundary. We can see that the minority group has the same amount of samples as the majority group, which is 226620.

```
SVMSMOTE = SVMSMOTE(random_state=42)

# fit and apply the transform
x_SVM_SMOTE, y_SVM_SMOTE = SVMSMOTE.fit_resample(x_train, y_train)
print("SVM_SMOTE RESULTS:")
y_SVM_SMOTE.value_counts()

SVM_SMOTE RESULTS:
0    226620
1    226620
Name: Class, dtype: int64
```

Figure 42: Code to run SVMSMOTE

3.8 Training and Testing Phase

3.8.1 Without Sampling

3.8.1.1 Logistic Regression

```
start = datetime.now()
LR_before = LogisticRegression(random_state=20)
# training
LR_before.fit(x_train, y_train)
# testing
y_pred_LR_before = LR_before.predict(x_test)
print(classification_report(y_test, y_pred_LR_before))

recall_sensitivity = metrics.recall_score(y_test, y_pred_LR_before, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_LR_before, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_LR_before)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 43: Code to run logistic regression without sampling

The code above creates a LogisticRegression model and attaches its references to the variable 'LR before,' which is the model's name. The random number generator was set at 20. We used the.fit() function to fit the model once it was constructed. After we've defined the model, we can use the.predict() function to acquire the actual predictions, which delivers the expected target value as a one-dimensional array. We utilised the classification report() method to acquire a more detailed report on the classification. The actual and anticipated outputs are sent as parameters to this function. For specialised uses, the sklearn.metrics function implements methods of quantifying prediction error. To obtain the sensitivity score we used the metrics.recall_score with pos_label=1 (recall of the positive class), while to obtain the specificity score, we used the metrics.recall_score with pos_label=0 (recall of the negative class). Furthermore, to obtain the accuracy score of the overall model, we called the metrics.accuracy_score function. To get the total time taken by the model to run, we have used the datetime.now() function.

3.8.1.2 Random Forest

```
start = datetime.now()
RF_before = RandomForestClassifier(random_state=10)
# training
RF_before.fit(x_train, y_train)
# testing
y_pred_RF_before = RF_before.predict(x_test)
print(classification_report(y_test, y_pred_RF_before))

recall_sensitivity = metrics.recall_score(y_test, y_pred_RF_before, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_RF_before, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_RF_before)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 44: Code to run random forest without sampling

3.8.1.3 XGBoost

```
start = datetime.now()
XGB_before = XGBClassifier(n_jobs=-1, random_state=30)
# training
XGB_before.fit(x_train, y_train)
# testing
y_pred_XGB_before = XGB_before.predict(x_test)
print(classification_report(y_test, y_pred_XGB_before))

recall_sensitivity = metrics.recall_score(y_test, y_pred_XGB_before, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_XGB_before, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_XGB_before)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 45: Code to run XGBoost without sampling

3.8.2 Random Under- sampling

3.8.2.1 Logistic Regression

```
start = datetime.now()
# training
LR_after_under = LogisticRegression(random_state=10)
LR_after_under.fit(x_under, y_under)
# testing
y_pred_LR_after_under = LR_after_under.predict(x_test)
print(classification_report(y_test, y_pred_LR_after_under))

recall_sensitivity = metrics.recall_score(y_test, y_pred_LR_after_under, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_LR_after_under, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_LR_after_under)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 46: Code to run a logistic regression with random under-sampling

3.8.2.2 Random Forest

```
start = datetime.now()
# training
RF_after_under = RandomForestClassifier(n_jobs=-1,random_state=30)
RF_after_under.fit(x_under, y_under)
# testing
y_pred_RF_after_under = RF_after_under.predict(x_test)
print(classification_report(y_test, y_pred_RF_after_under))

recall_sensitivity = metrics.recall_score(y_test, y_pred_RF_after_under, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_RF_after_under, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_RF_after_under)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 47: Code to run random forest with random under-sampling

3.8.2.3 XGBoost

```
start = datetime.now()
# training
XGB_after_under = XGBClassifier(n_jobs=-1,random_state=70)
XGB_after_under.fit(x_under, y_under)
# testing
y_pred_XGB_after_under = XGB_after_under.predict(x_test)
print(classification_report(y_test, y_pred_XGB_after_under))

recall_sensitivity = metrics.recall_score(y_test, y_pred_XGB_after_under, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_XGB_after_under, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_XGB_after_under)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 48: Code to run XGBoost with random under-sampling

3.8.3 Random Oversampling

3.8.3.1 Logistic Regression

```
start = datetime.now()
# training
LR_after_over = LogisticRegression(random_state=40)
LR_after_over.fit(x_over, y_over)
# testing
y_pred_LR_after_over = LR_after_over.predict(x_test)
print(classification_report(y_test, y_pred_LR_after_over))

recall_sensitivity = metrics.recall_score(y_test, y_pred_LR_after_over, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_LR_after_over, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_LR_after_over)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 49: Code to run a logistic regression with random oversampling

3.8.3.2 Random Forest

```
start = datetime.now()
# training
RF_after_over = RandomForestClassifier(n_jobs=-1,random_state=40)
RF_after_over.fit(x_over, y_over)
# testing
y_pred_RF_after_over = RF_after_over.predict(x_test)
print(classification_report(y_test, y_pred_RF_after_over))

recall_sensitivity = metrics.recall_score(y_test, y_pred_RF_after_over, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_RF_after_over, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_RF_after_over)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 50: Code to run random forest with random oversampling

3.8.3.3 XGBoost

```
start = datetime.now()
# training
XGB_after_over = XGBClassifier(n_jobs=-1, random_state=80)
XGB_after_over.fit(x_over, y_over)
# testing
y_pred_XGB_after_over = XGB_after_over.predict(x_test)
print(classification_report(y_test, y_pred_XGB_after_over))

recall_sensitivity = metrics.recall_score(y_test, y_pred_XGB_after_over, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_XGB_after_over, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_XGB_after_over)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 51: Code to run XGBoost with random oversampling

3.8.4 SMOTE

3.8.4.1 Logistic Regression

```
start = datetime.now()
# training
LR_after_SMOTE = LogisticRegression(random_state=30)
LR_after_SMOTE.fit(x_SMOTE, y_SMOTE)
# testing
y_pred_LR_after_SMOTE = LR_after_SMOTE.predict(x_test)
print(classification_report(y_test, y_pred_LR_after_SMOTE))

recall_sensitivity = metrics.recall_score(y_test, y_pred_LR_after_SMOTE, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_LR_after_SMOTE, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_LR_after_SMOTE)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 52: Code to run a logistic regression with SMOTE

3.8.4.2 Random Forest

```
start = datetime.now()
# training
RF_after_SMOTE = RandomForestClassifier(n_jobs=-1,random_state=50)
RF_after_SMOTE.fit(x_SMOTE, y_SMOTE)
# testing
y_pred_RF_after_SMOTE = RF_after_SMOTE.predict(x_test)
print(classification_report(y_test, y_pred_RF_after_SMOTE))

recall_sensitivity = metrics.recall_score(y_test, y_pred_RF_after_SMOTE, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_RF_after_SMOTE, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_RF_after_SMOTE)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 53: Code to run random forest with SMOTE

3.8.4.3 XGBoost

```
start = datetime.now()
# training
XGB_after_SMOTE = XGBClassifier(n_jobs=-1,random_state=90)
XGB_after_SMOTE.fit(x_SMOTE, y_SMOTE)
# testing
y_pred_XGB_after_SMOTE = XGB_after_SMOTE.predict(x_test)
print(classification_report(y_test, y_pred_XGB_after_SMOTE))

recall_sensitivity = metrics.recall_score(y_test, y_pred_XGB_after_SMOTE, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_XGB_after_SMOTE, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_XGB_after_SMOTE)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 54: Code to run XGBoost with SMOTE

3.8.5 SVMSMOTE

3.8.5.1 Logistic Regression

```
start = datetime.now()
# training
LR_after_SVMSMOTE = LogisticRegression(random_state=20)
LR_after_SVMSMOTE.fit(x_SVM_SMOTE, y_SVM_SMOTE)
# testing
y_pred_LR_after_SVMSMOTE = LR_after_SVMSMOTE.predict(x_test)
print(classification_report(y_test, y_pred_LR_after_SVMSMOTE))

recall_sensitivity = metrics.recall_score(y_test, y_pred_LR_after_SVMSMOTE, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_LR_after_SVMSMOTE, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_LR_after_SVMSMOTE)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 55: Code to run a logistic regression with SVMSMOTE

3.8.5.2 Random Forest

```
start = datetime.now()
# training
RF_after_SVMSMOTE = RandomForestClassifier(n_jobs=-1,random_state=60)
RF_after_SVMSMOTE.fit(x_SVM_SMOTE, y_SVM_SMOTE)
# testing
y_pred_RF_after_SVMSMOTE = RF_after_SVMSMOTE.predict(x_test)
print(classification_report(y_test, y_pred_RF_after_SVMSMOTE))

recall_sensitivity = metrics.recall_score(y_test, y_pred_RF_after_SVMSMOTE, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_RF_after_SVMSMOTE, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_RF_after_SVMSMOTE)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 56: Code to run random forest with SVMSMOTE

3.8.5.3 XGBoost

```
start = datetime.now()
# training
XGB_after_SVMSMOTE = XGBClassifier(n_jobs=-1, random_state=15)
XGB_after_SVMSMOTE.fit(x_SVM_SMOTE, y_SVM_SMOTE)
# testing
y_pred_XGB_after_SVMSMOTE = XGB_after_SVMSMOTE.predict(x_test)
print(classification_report(y_test, y_pred_XGB_after_SVMSMOTE))

recall_sensitivity = metrics.recall_score(y_test, y_pred_XGB_after_SVMSMOTE, pos_label=1)
recall_specificity = metrics.recall_score(y_test, y_pred_XGB_after_SVMSMOTE, pos_label=0)
print('Sensitivity:',recall_sensitivity)
print('Specificity:',recall_specificity)

accuracy = metrics.accuracy_score(y_test, y_pred_XGB_after_SVMSMOTE)
print('Accuracy:',accuracy)

end = datetime.now()
time_taken = end - start
print('Time: ',time_taken)
```

Figure 57: Code to run XGBoost with SVMSMOTE

3.9 Hyperparameter Tuning

```
print('Parameters currently in use:\n')
print(RF_after_over.get_params())
```

```
estimator = RandomForestClassifier(n_jobs=-1, random_state=35)

n_estimators = [int(x) for x in np.linspace(start = 1, stop = 20, num = 20)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)]
min_samples_split = [2, 6, 10]
min_samples_leaf = [1, 3, 4]
bootstrap = [True, False]
```

```
random_grid = {'n_estimators': n_estimators,

'max_features': max_features,

'max_depth': max_depth,

'min_samples_split': min_samples_split,

'min_samples_leaf': min_samples_leaf,

'bootstrap': bootstrap}
```

```
rf_random = RandomizedSearchCV(estimator = estimator,param_distributions = random_grid,n_iter = 100, cv = 5, verbose=
2, random_state=35, n_jobs = -1)
```

```
rf_random.fit(x,y)
```

```
# this prints the contents of the parameters in the random grid
print ('Random grid: ', random_grid, '\n')

# print the best parameters
print ('Best Parameters: ', rf_random.best_params_, ' \n')
```

Figure 58: Codes to run hyperparameter tuning for random forest

We tuned the hyperparameters for the top three models that executed well in this section. Only a few random parameter combinations, such as max features, max depth, min samples split, min sample leaf, and bootstrap, were chosen using the RandomSearchCV. We have set the n iter= 100 and cv= 5, which indicates that any 100 random combinations will be run five times. The overall accuracy for each combination of hyperparameters is the average of these five iterations after partitioning the data into five parts and choosing one as testing and the other four as training data.

CHAPTER 4: Results and Analysis

In this section, we discuss the results of our experiments. We conducted experiments with 3 machine learning models and compared them before and after resampling methods were applied. The result of the experiments is summarized in table 2 and table 3.

4.1 Before Resampling

4.1.1 Logistic Regression

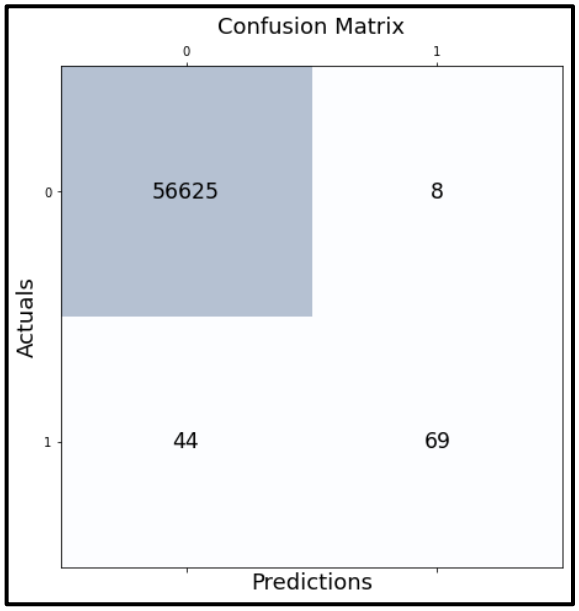


Figure 59: Confusion matrix of logistic regression before resampling

From figure 59 we can observe, out of 56746 total test samples, 56625 are true positives (correctly classified), 8 are false positives (misclassified), 44 are false negatives (misclassified), and 69 are true negatives (correctly classified).

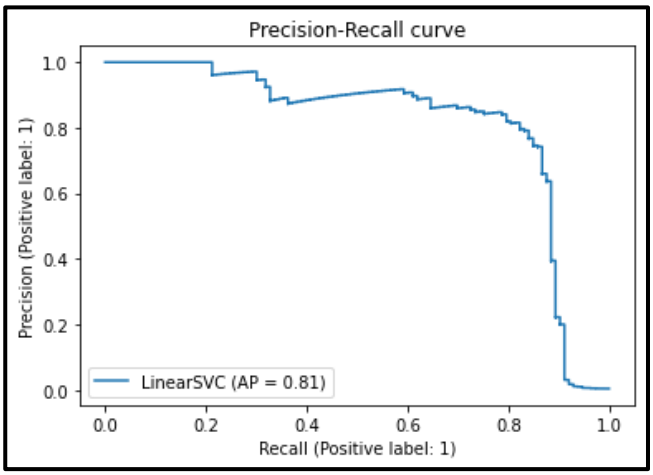


Figure 60: Precision-Recall curve of logistic regression before resampling

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.90	0.61	0.73	113
accuracy			1.00	56746
macro avg	0.95	0.81	0.86	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.6106194690265486
Specificity: 0.9998587396041178
Accuracy: 0.9990836358509851
Time: 0:00:09.823724

Figure 61: Classification report of logistic regression before resampling

Reviewing both precision and recall is useful in cases where there is an imbalance in the dataset between the two classes. The reason for this is that typically a large number of class 0 examples means we are less interested in the skill of the model at predicting class 0 correctly (true negatives). As observed from the classification report, we have a high specificity (true negative) of almost 100%. In this case, we are only concerned with the correct prediction of the minority class, which is class 1. One way to calculate the Precision-Recall curve is to find the average precision (AP) value. AP summarizes a precision-recall curve as the weighted mean of precision achieved at each threshold, with the increase in recall from the previous threshold used as the weight. From figure 60, we can deduce that our AP value is at 81%, which is moderate. At thresholds with very high precision, the recall is very low as observed in figure 60 and figure 61. A high precision score relates to a low false-positive rate, while a low recall relates to a high false-negative rate. A precision value of 90% indicates that when the model predicted fraud cases, it was correct 90% of the time, while a recall value of 61% indicates that the model was able to identify 61% of all positive cases correctly, which is rather low.

Furthermore, the overall accuracy of the model is almost 100%, however, since the dataset is imbalanced, we are not encouraged to consider the accuracy score to access the model. Lastly, the time taken for the model to run is 9s.

4.1.2 Random Forest

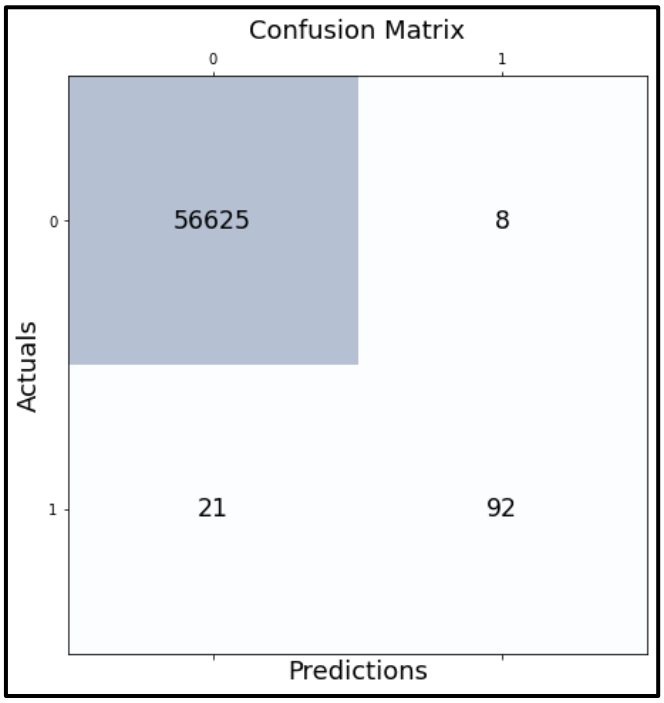


Figure 62: Confusion matrix of the random forest before resampling

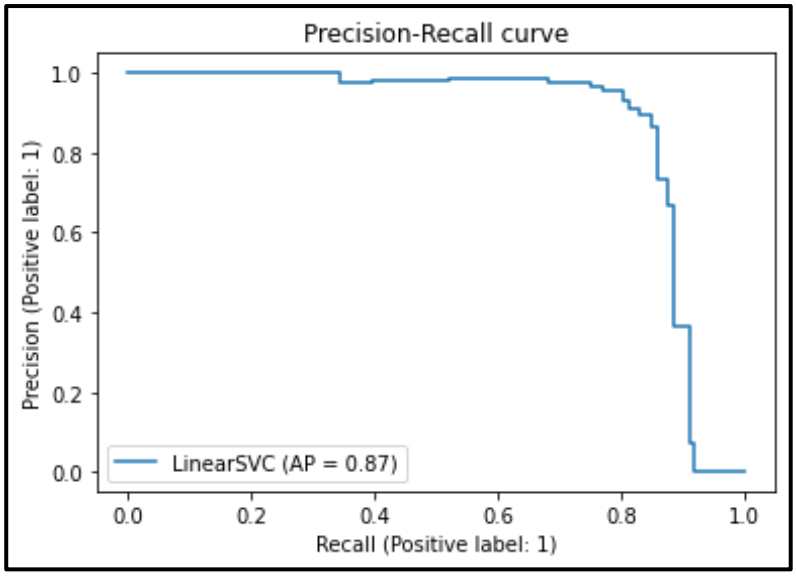
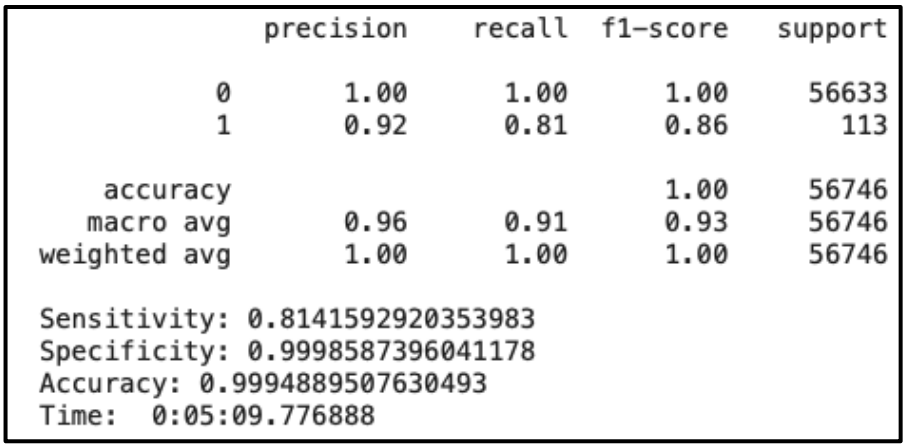


Figure 63: Precision-Recall curve of the random forest before resampling



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.92	0.81	0.86	113
accuracy			1.00	56746
macro avg	0.96	0.91	0.93	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.8141592920353983
Specificity: 0.9998587396041178
Accuracy: 0.9994889507630493
Time: 0:05:09.776888

Figure 64: Classification report of the random forest before resampling

From figure 62 we can observe, out of 56746 total test samples, 56625 are true positives (correctly classified), 8 are false positives (misclassified), 21 are false negatives (misclassified), and 92 are true negatives (correctly classified).

From figure 63, we can deduce that our AP value is 87%, which is quite high. At thresholds with very high precision, the recall is at a moderate value as observed in figure 63 and figure 64. A precision value of 92% indicates that when the model predicted fraud cases, it was correct 92% of the time, while a recall value of 81% indicates that the model was able to identify 81% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run is 5m 9s.

4.1.3 XGBoost

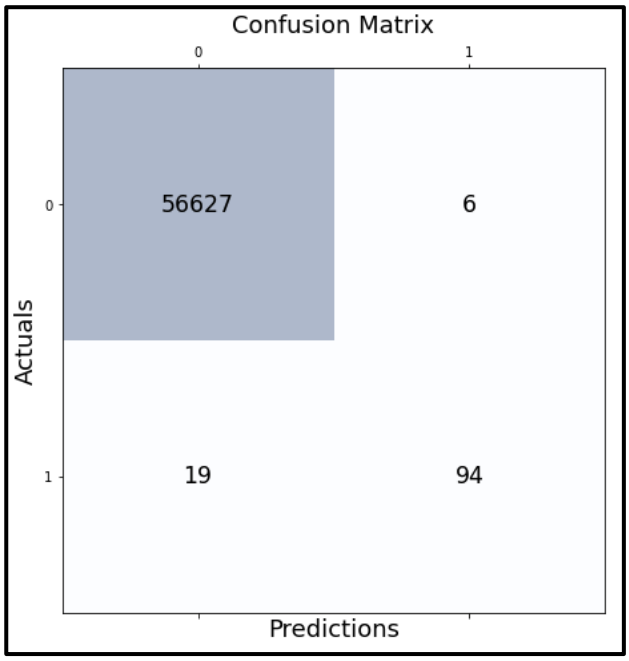


Figure 65: Confusion matrix of XGBoost without sampling

From figure 65 we can observe, out of 56746 total test samples, 56627 are true positives (correctly classified), 6 are false positives (misclassified), 19 are false negatives (misclassified), and 94 are true negatives (correctly classified).

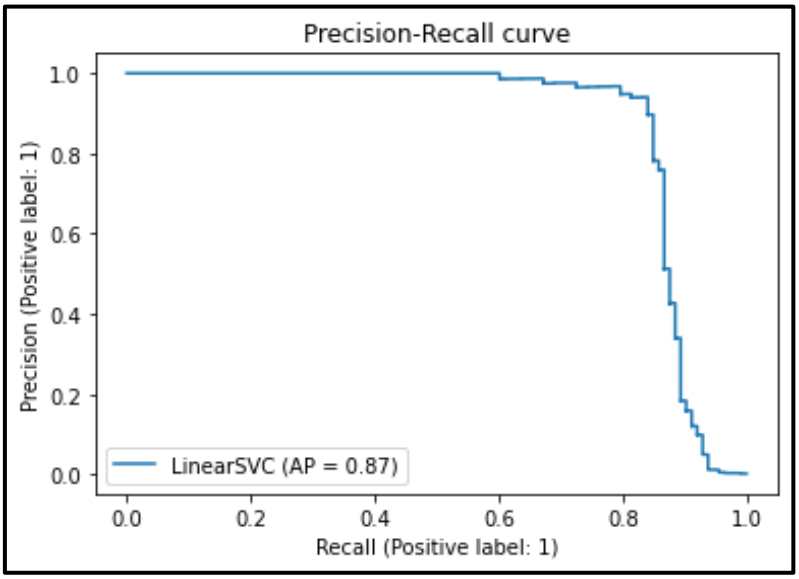
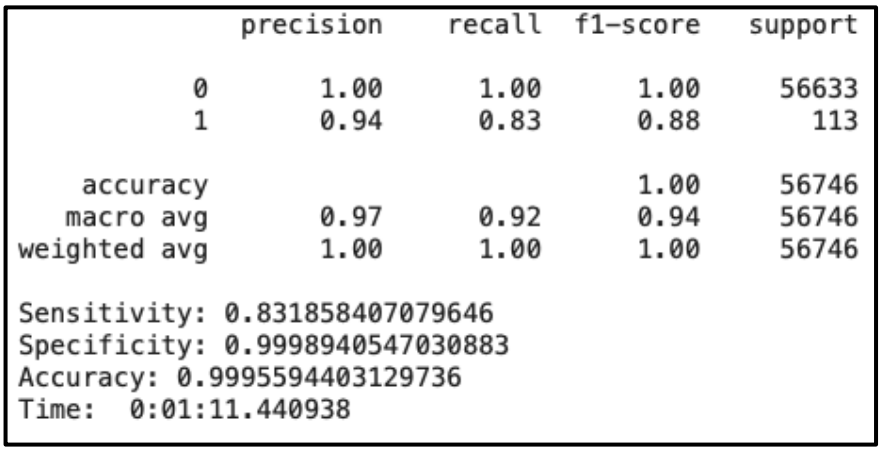


Figure 66: Precision-Recall curve of XGBoost without sampling



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.94	0.83	0.88	113
accuracy			1.00	56746
macro avg	0.97	0.92	0.94	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.831858407079646
Specificity: 0.9998940547030883
Accuracy: 0.9995594403129736
Time: 0:01:11.440938

Figure 67: Classification report of XGBoost without sampling

From figure 65, we can deduce that our AP value is 87%, which is quite high. At thresholds with very high precision, the recall is at a moderate value as seen in figure 66 and figure 67. A precision value of 94% indicates that when the model predicted fraud cases, it was correct 94% of the time, while a recall value of 31% indicates that the model was able to identify 83% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run is 1m 11s.

4.2 Random Oversampling

4.2.1 Logistic Regression

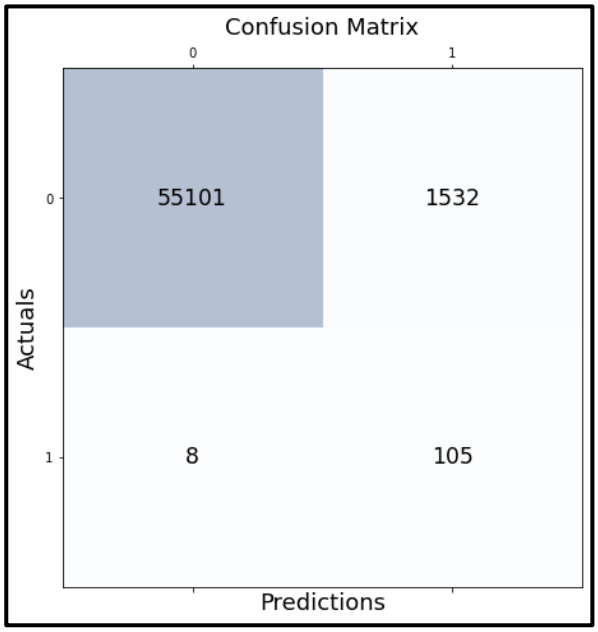


Figure 68: Confusion matrix of logistic regression after random oversampling

From figure 68 we can observe, out of 56746 total test samples, 55101 are true positives (correctly classified), 1532 are false positives (misclassified), 8 are false negatives (misclassified), and 105 are true negatives (correctly classified).

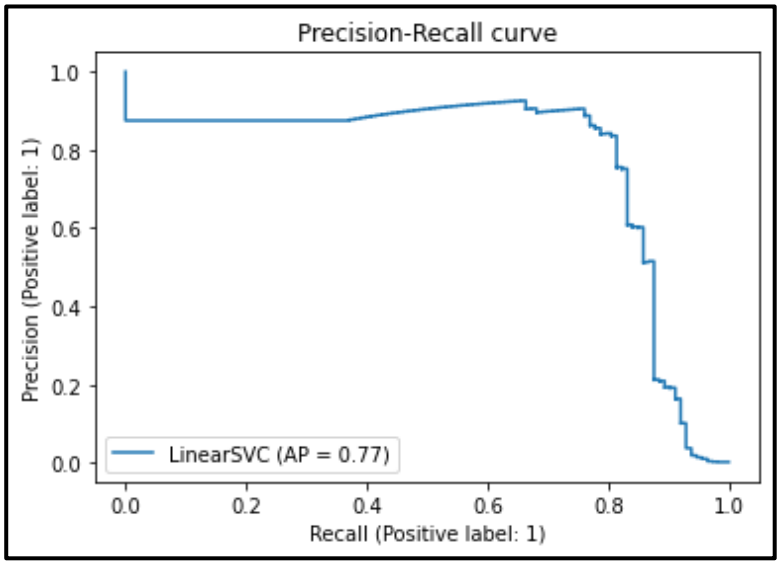
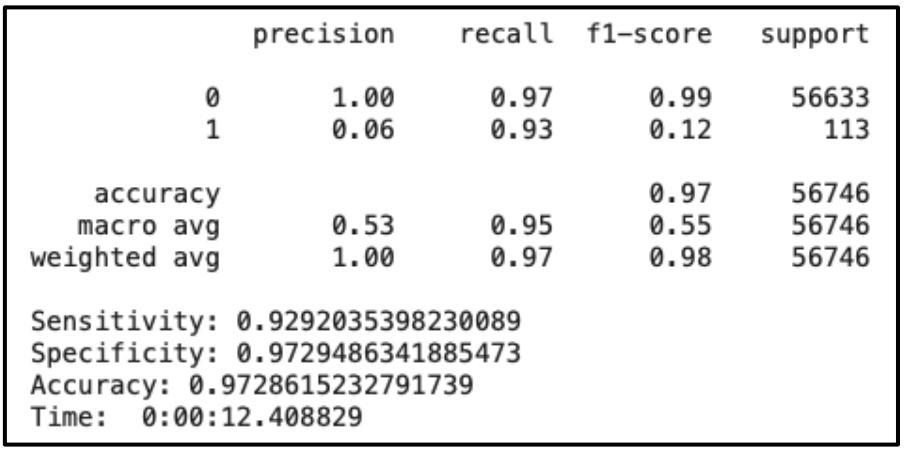


Figure 69: Precision-Recall curve of logistic regression after random oversampling



	precision	recall	f1-score	support
0	1.00	0.97	0.99	56633
1	0.06	0.93	0.12	113
accuracy			0.97	56746
macro avg	0.53	0.95	0.55	56746
weighted avg	1.00	0.97	0.98	56746

Sensitivity: 0.9292035398230089
Specificity: 0.9729486341885473
Accuracy: 0.9728615232791739
Time: 0:00:12.408829

Figure 70: Classification report of logistic regression after random oversampling

From figure 69, we can deduce that our AP value is at 77%, which is rather low. At thresholds with extremely low precision, the recall is at a very high value as seen in figure 69 and figure 70. A precision value of only 6% indicates that when the model predicted fraud cases, it was correct 6% of the time, while a recall value of 93% indicates that the model was able to identify 93% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run is 12s.

4.2.2 Random Forest

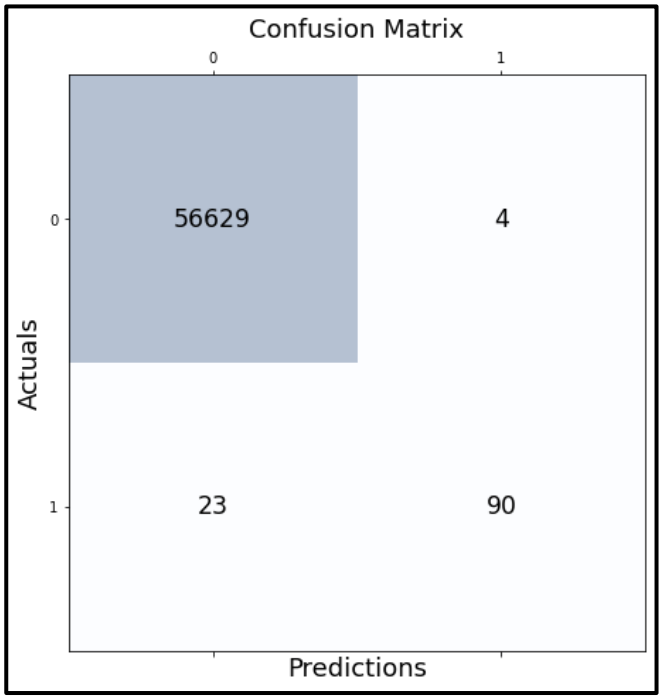


Figure 71: Confusion matrix of the random forest after random oversampling

From figure 71 we can observe, out of 56746 total test samples, 56629 are true positives (correctly classified), 4 are false positives (misclassified), 23 are false negatives (misclassified), and 90 are true negatives (correctly classified).

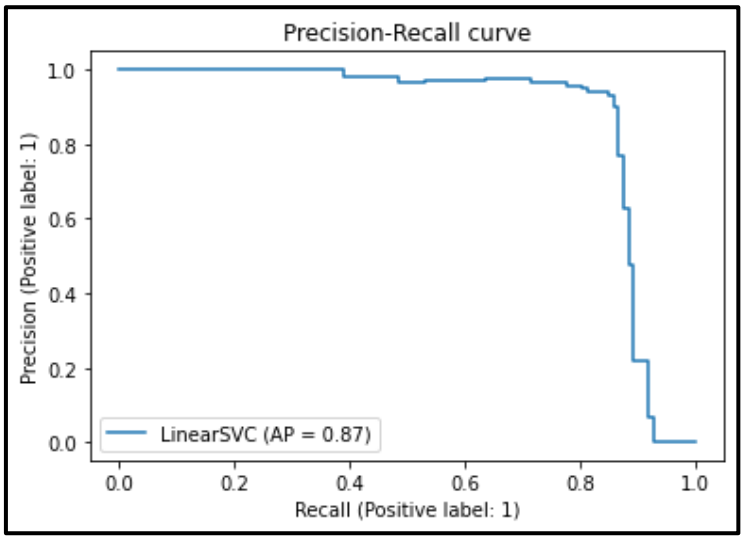
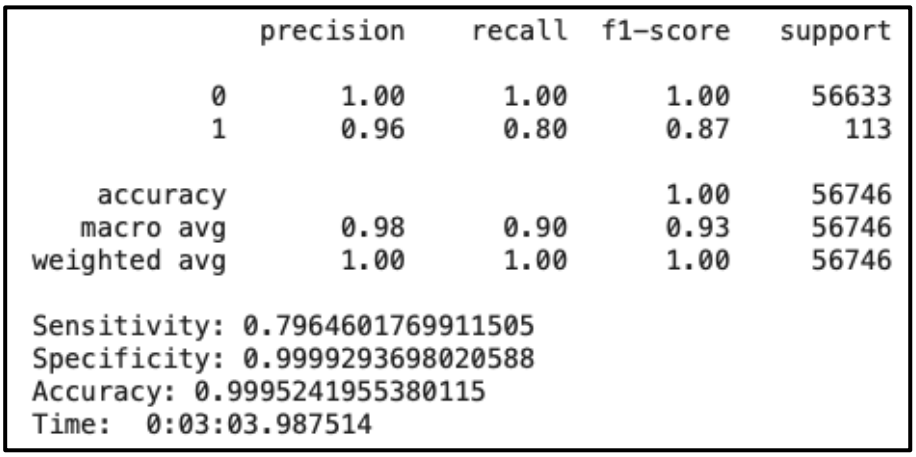


Figure 72: Precision-Recall curve of the random forest after random oversampling



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.96	0.80	0.87	113
accuracy			1.00	56746
macro avg	0.98	0.90	0.93	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.7964601769911505
Specificity: 0.9999293698020588
Accuracy: 0.9995241955380115
Time: 0:03:03.987514

Figure 73: Classification report of the random forest after random oversampling

From figure 72, we can deduce that our AP value is at 87%, which is high. At thresholds with very high precision, the recall is at a moderately high value as seen in figure 72 and figure 73. A precision value of 96% indicates that when the model predicted fraud cases, it was correct 96% of the time, while a recall value of 80% indicates that the model was able to identify 80% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run is 3m 3s.

4.2.3 XGBoost

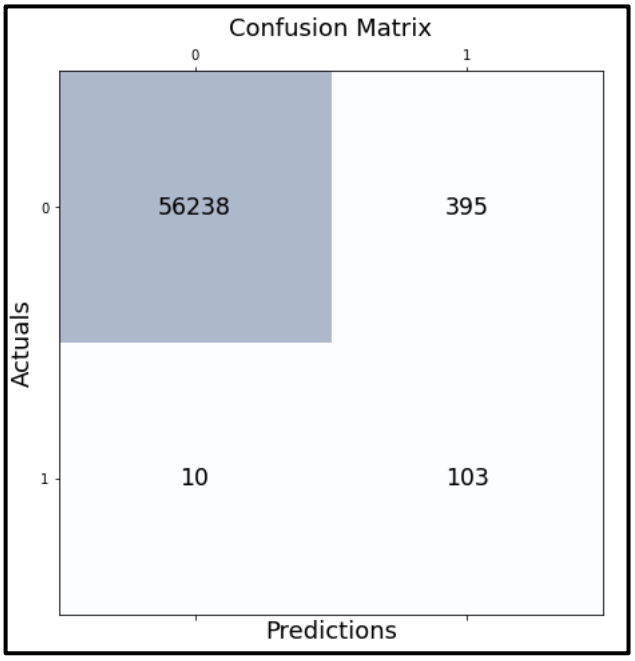


Figure 74: Confusion matrix of XGBoost after random oversampling

From figure 74 we can observe, out of 56746 total test samples, 56238 are true positives (correctly classified), 395 are false positives (misclassified), 10 are false negatives (misclassified), and 103 are true negatives (correctly classified).

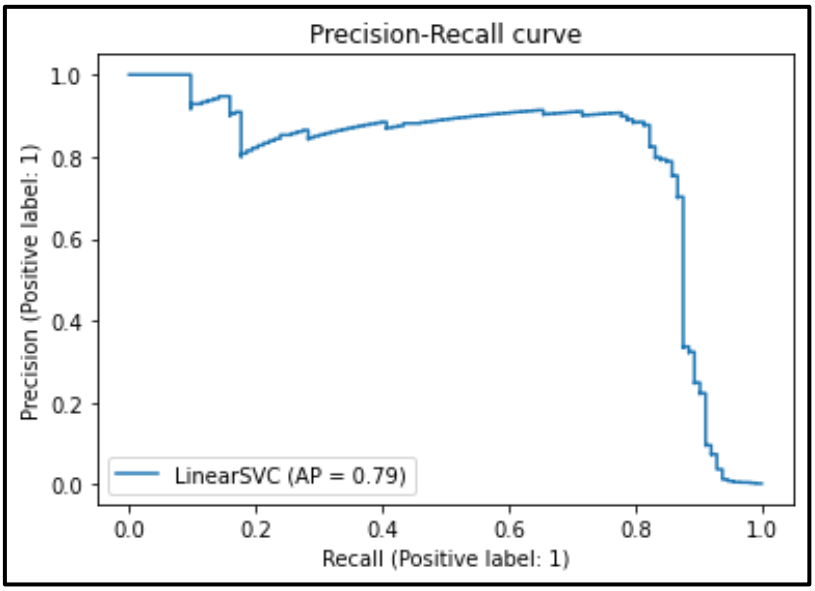
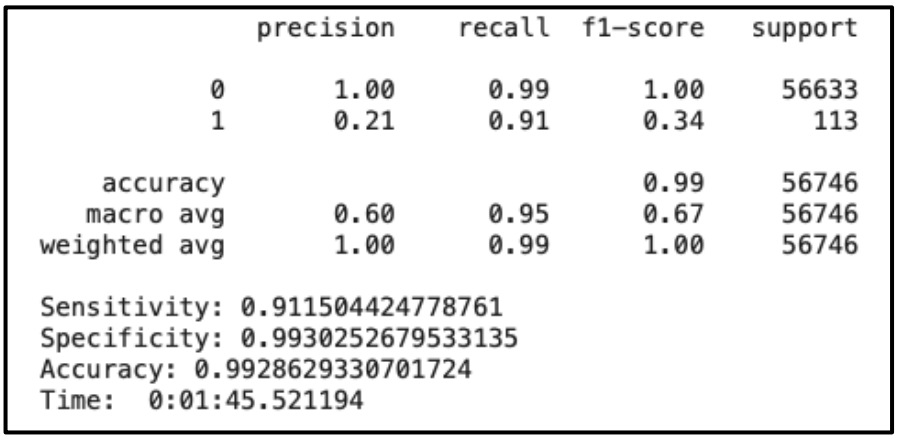


Figure 75: Precision-Recall curve of XGBoost after random oversampling



	precision	recall	f1-score	support
0	1.00	0.99	1.00	56633
1	0.21	0.91	0.34	113
accuracy			0.99	56746
macro avg	0.60	0.95	0.67	56746
weighted avg	1.00	0.99	1.00	56746

Sensitivity: 0.911504424778761
Specificity: 0.9930252679533135
Accuracy: 0.9928629330701724
Time: 0:01:45.521194

Figure 76: Classification report of XGBoost after random oversampling

From figure 75, we can deduce that our AP value is at 79%, which is rather low. At thresholds with very low precision, the recall is at a high value as seen in figure 75 and figure 76. A precision value of only 21% indicates that when the model predicted fraud cases, it was correct 21% of the time, while a recall value of 91% indicates that the model was able to identify 91% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run is 1m 45s.

4.3 Random Under-sampling

4.3.1 Logistic Regression

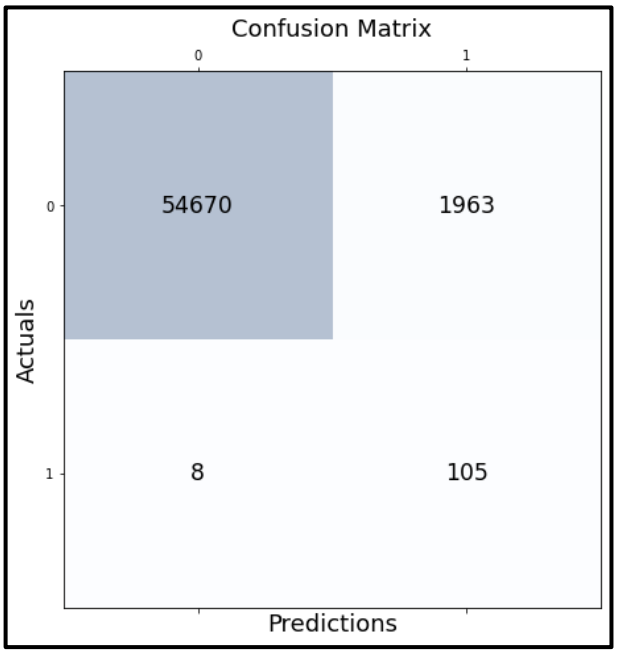


Figure 77: Confusion matrix of logistic regression after random under-sampling

From figure 77 we can observe, out of 56746 total test samples, 54670 are true positives (correctly classified), 1963 are false positives (misclassified), 8 are false negatives (misclassified), and 105 are true negatives (correctly classified).

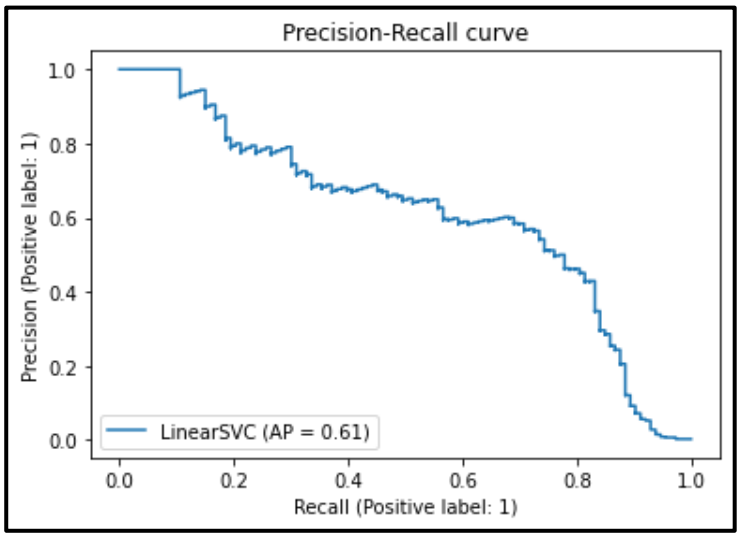
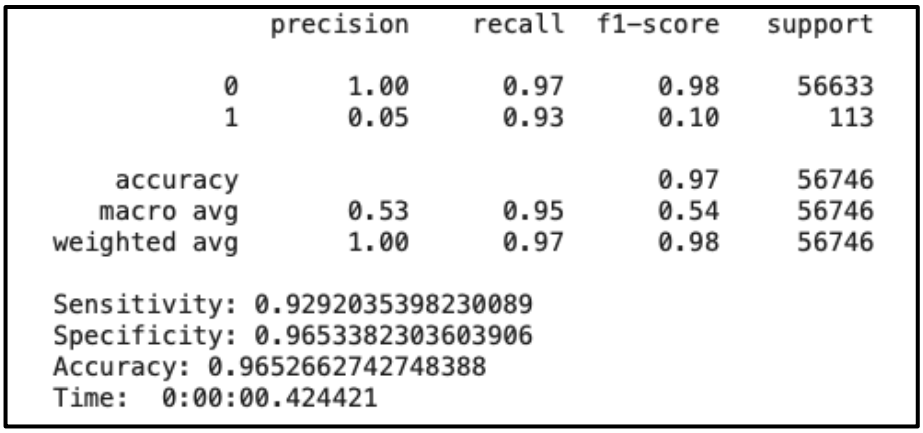


Figure 78: Precision-Recall curve of logistic regression after random under-sampling



	precision	recall	f1-score	support
0	1.00	0.97	0.98	56633
1	0.05	0.93	0.10	113
accuracy			0.97	56746
macro avg	0.53	0.95	0.54	56746
weighted avg	1.00	0.97	0.98	56746

Sensitivity: 0.9292035398230089
Specificity: 0.9653382303603906
Accuracy: 0.9652662742748388
Time: 0:00:00.424421

Figure 79: Classification report of logistic regression after random under-sampling

From figure 78, we can deduce that our AP value is at 61%, which is very low. At thresholds with very low precision, the recall is at a high value as seen in figure 78 and figure 79. A precision value of only 5% indicates that when the model predicted fraud cases, it was correct 5% of the time, while a recall value of 93% indicates that the model was able to identify 93% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 97%. Lastly, the time taken for the model to run was 42ms.

4.3.2 Random Forest

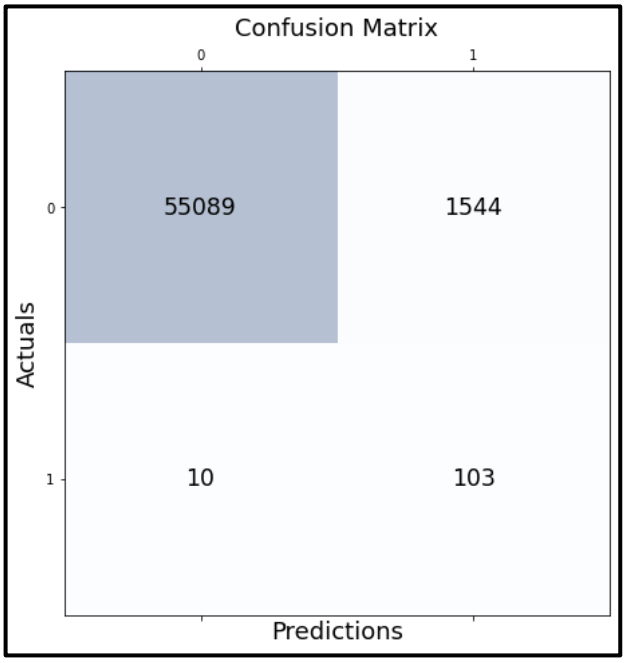


Figure 80: Confusion matrix of the random forest after random under-sampling

From figure 80 we can observe, out of 56746 total test samples, 55089 are true positives (correctly classified), 1544 are false positives (misclassified), 10 are false negatives (misclassified), and 103 are true negatives (correctly classified).

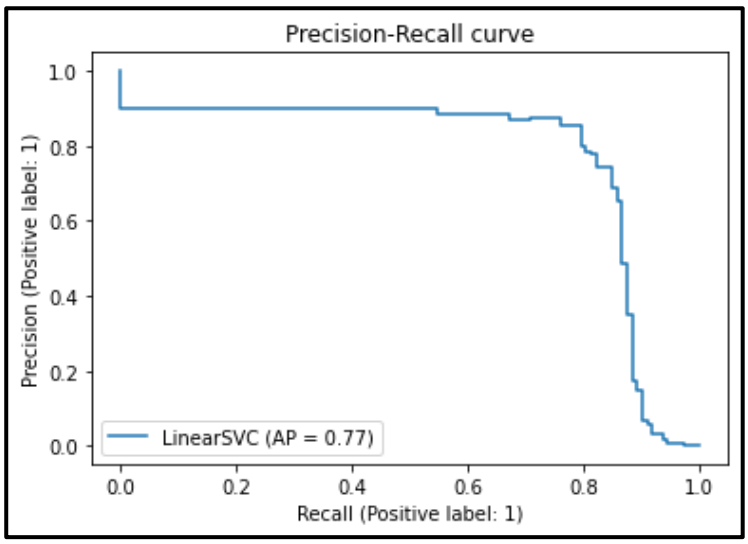
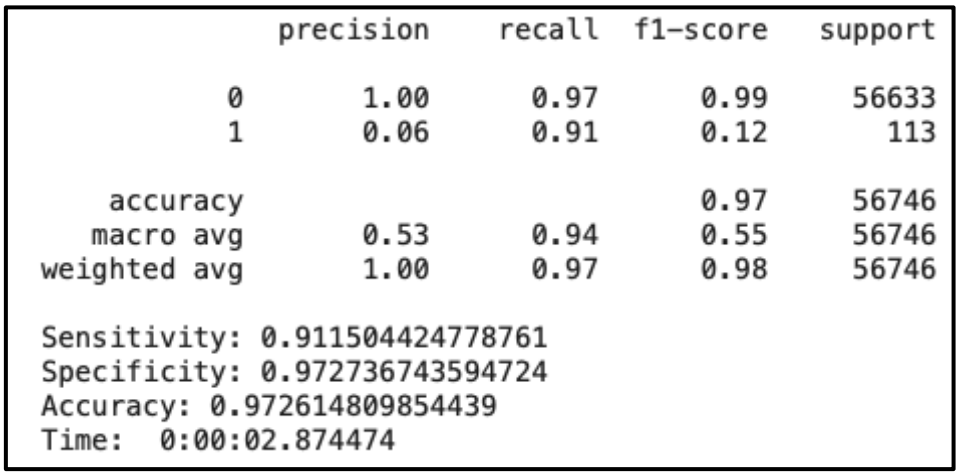


Figure 81: Precision-Recall curve of the random forest after random under-sampling



	precision	recall	f1-score	support
0	1.00	0.97	0.99	56633
1	0.06	0.91	0.12	113
accuracy			0.97	56746
macro avg	0.53	0.94	0.55	56746
weighted avg	1.00	0.97	0.98	56746

Sensitivity: 0.911504424778761
Specificity: 0.972736743594724
Accuracy: 0.972614809854439
Time: 0:00:02.874474

Figure 82: Classification report of the random forest after random under-sampling

From figure 81, we can deduce that our AP value is at 77%, which is rather low. At thresholds with very low precision, the recall is at a high value as seen in figure 81 and figure 82. A precision value of only 6% indicates that when the model predicted fraud cases, it was correct 6% of the time, while a recall value of 91% indicates that the model was able to identify 91% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is 97%. Lastly, the time taken for the model to run was 2s.

4.3.3 XGBoost

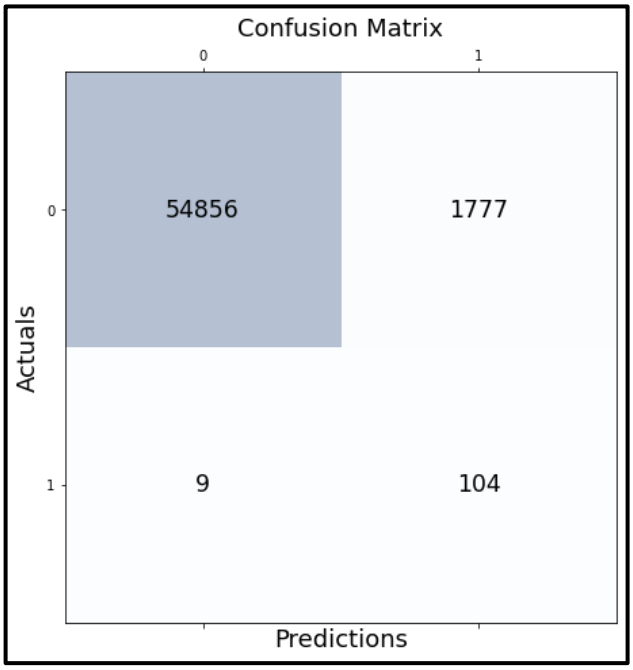


Figure 83: Confusion matrix of XGBoost after random under-sampling

From figure 83 we can observe, out of 56746 total test samples, 54856 are true positives (correctly classified), 1777 are false positives (misclassified), 9 are false negatives (misclassified), and 104 are true negatives (correctly classified).

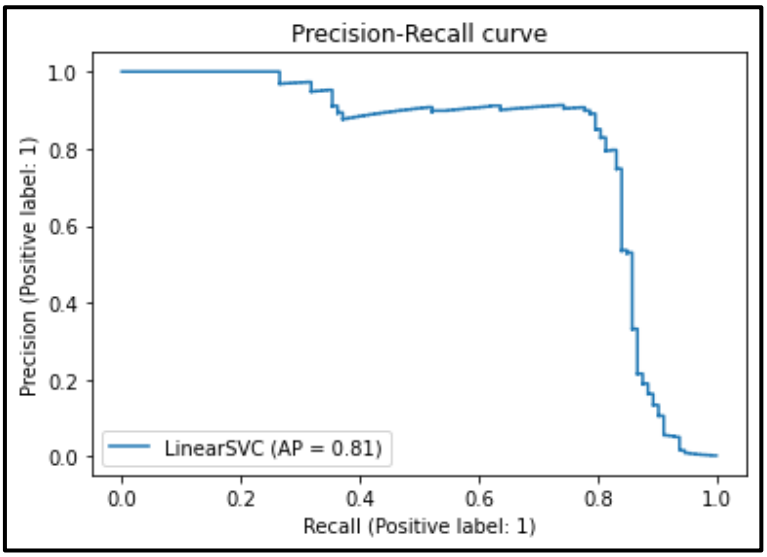
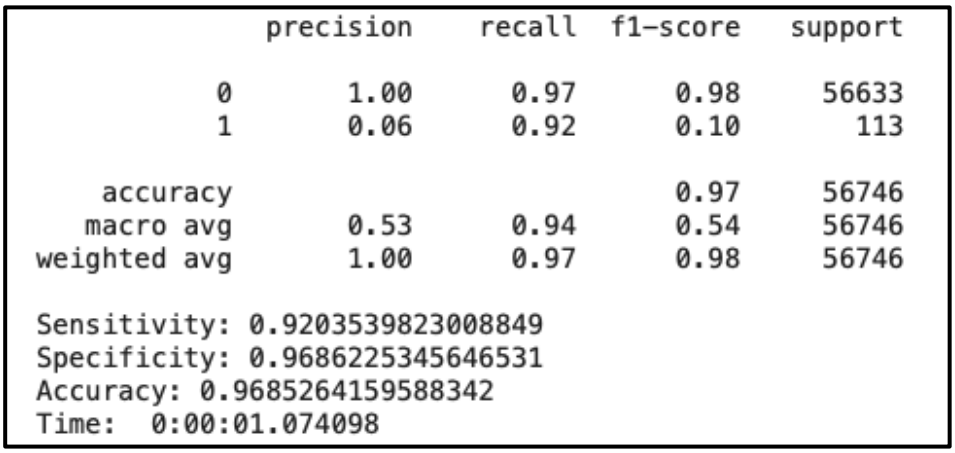


Figure 84: Precision-Recall curve of XGBoost after random under-sampling



	precision	recall	f1-score	support
0	1.00	0.97	0.98	56633
1	0.06	0.92	0.10	113
accuracy			0.97	56746
macro avg	0.53	0.94	0.54	56746
weighted avg	1.00	0.97	0.98	56746

Sensitivity: 0.9203539823008849
Specificity: 0.9686225345646531
Accuracy: 0.9685264159588342
Time: 0:00:01.074098

Figure 85: Classification report of XGBoost after random under-sampling

From figure 84, we can deduce that our AP value is at 81%, which is moderate. At thresholds with very low precision, the recall is at a high value as seen in figure 84 and figure 85. A precision value of only 6% indicates that when the model predicted fraud cases, it was correct 6% of the time, while a recall value of 92% indicates that the model was able to identify 92% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 97%. Lastly, the time taken for the model to run was 1s 7ms.

4.4 SMOTE

4.4.1 Logistic Regression

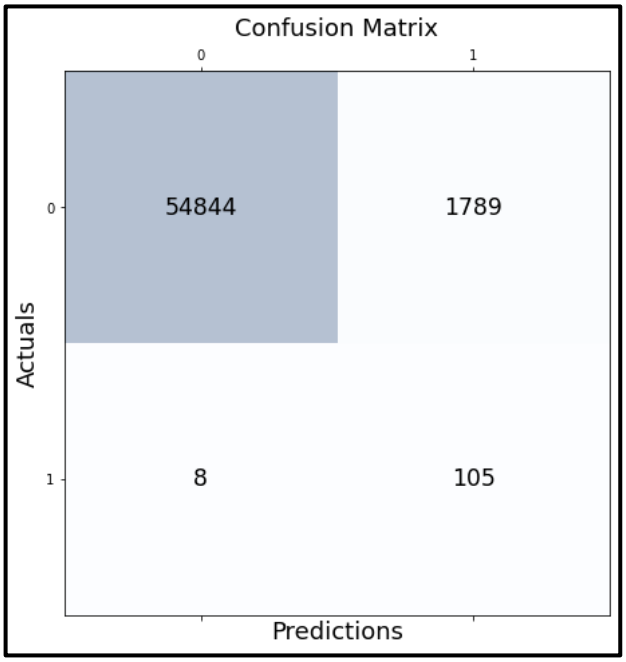


Figure 86: Confusion matrix of logistic regression after SMOTE

From figure 86 we can observe, out of 56746 total test samples, 54844 are true positives (correctly classified), 1789 are false positives (misclassified), 8 are false negatives (misclassified), and 105 are true negatives (correctly classified).

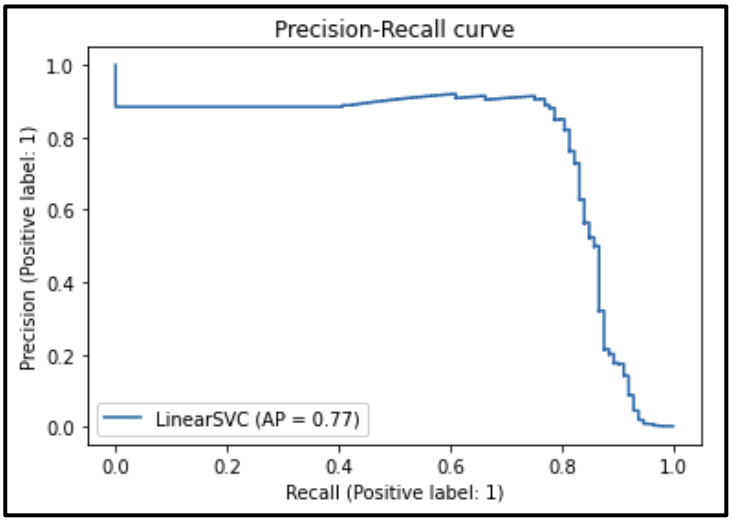
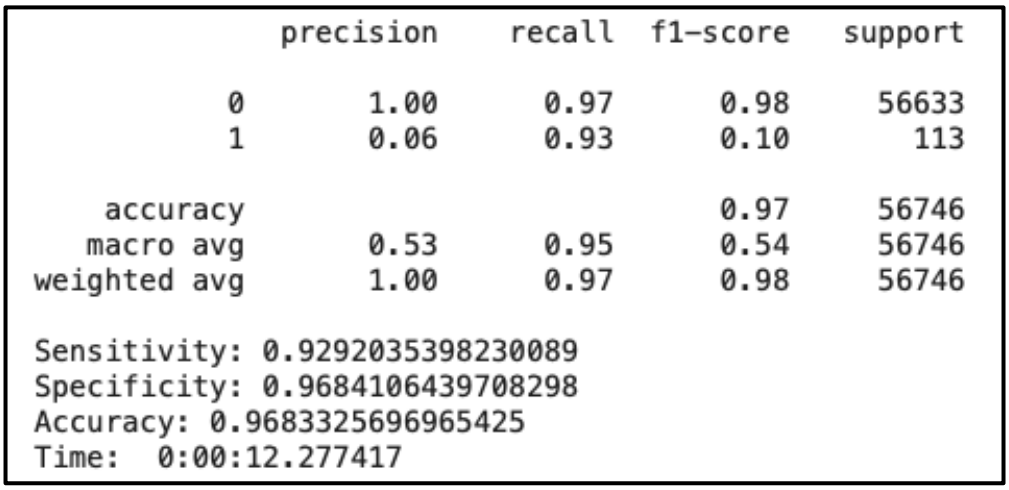


Figure 87: Precision-Recall curve of logistic regression after SMOTE



	precision	recall	f1-score	support
0	1.00	0.97	0.98	56633
1	0.06	0.93	0.10	113
accuracy			0.97	56746
macro avg	0.53	0.95	0.54	56746
weighted avg	1.00	0.97	0.98	56746

Sensitivity: 0.9292035398230089
Specificity: 0.9684106439708298
Accuracy: 0.9683325696965425
Time: 0:00:12.277417

Figure 88: Classification report of logistic regression after SMOTE

From figure 87, we can deduce that our AP value is at 77%, which is rather low. At thresholds with very low precision, the recall is at a high value as seen in figure 87 and figure 88. A precision value of only 6% indicates that when the model predicted fraud cases, it was correct 6% of the time, while a recall value of 93% indicates that the model was able to identify 93% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 97%. Lastly, the time taken for the model to run was 12s.

4.4.2 Random Forest

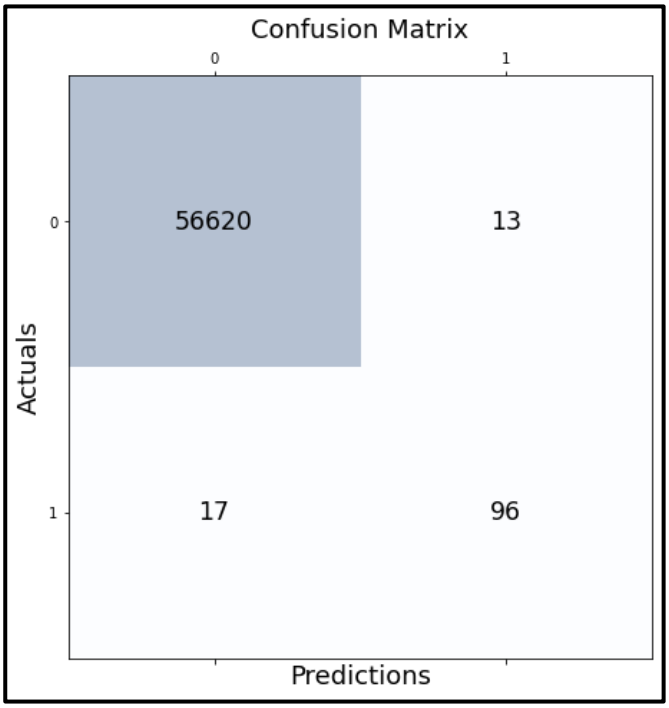


Figure 89: Confusion matrix of the random forest after SMOTE

From figure 89 we can observe, out of 56746 total test samples, 56620 are true positives (correctly classified), 13 are false positives (misclassified), 17 are false negatives (misclassified), and 96 are true negatives (correctly classified).

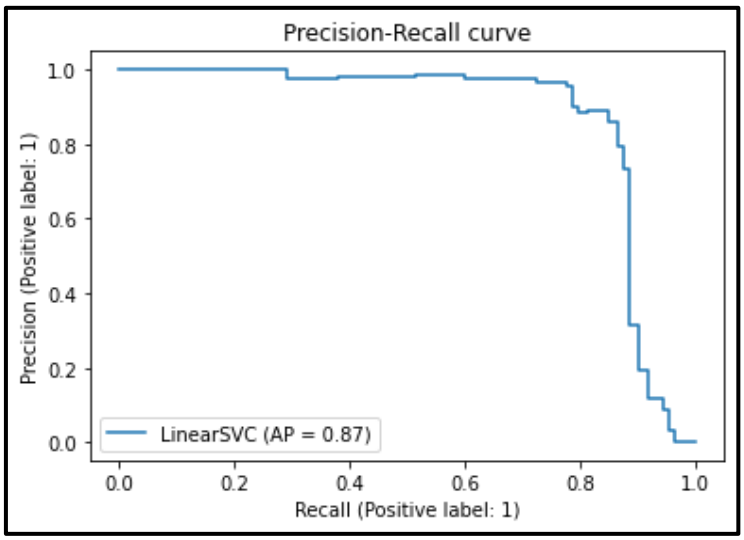
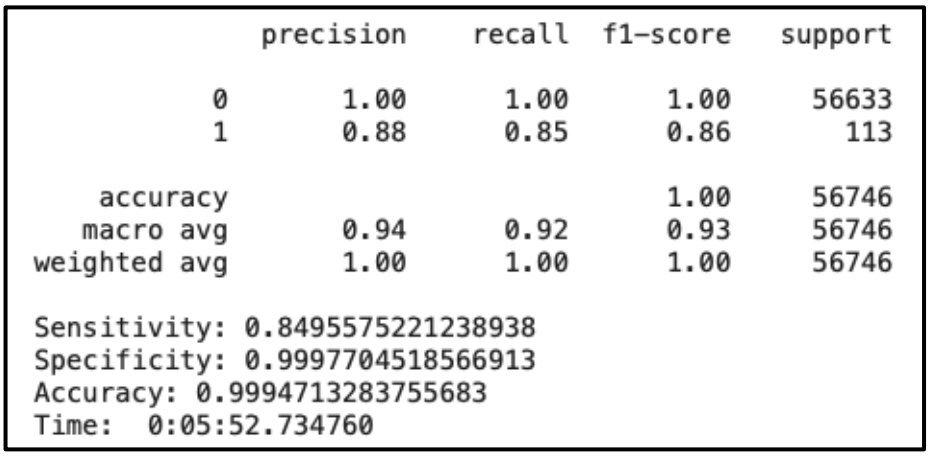


Figure 90: Precision-Recall curve of the random forest after SMOTE



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.88	0.85	0.86	113
accuracy			1.00	56746
macro avg	0.94	0.92	0.93	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.8495575221238938
Specificity: 0.9997704518566913
Accuracy: 0.9994713283755683
Time: 0:05:52.734760

Figure 91: Classification report of the random forest after SMOTE

From figure 90, we can deduce that our AP value is at 87%, which is high. At thresholds with high precision, the recall is also at a high value as seen in figure 90 and figure 91. A precision value of 88% indicates that when the model predicted fraud cases, it was correct 88% of the time, while a recall value of 85% indicates that the model was able to identify 85% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run was 5m 52s.

4.4.3 XGBoost

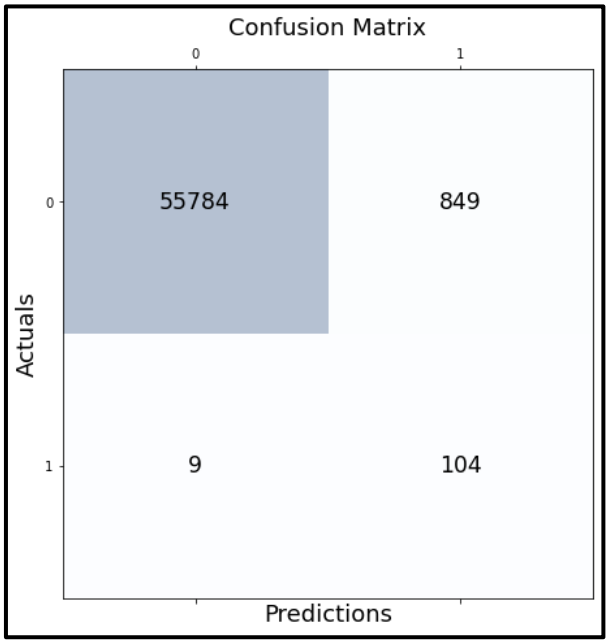


Figure 92: Confusion matrix of XGBoost after SMOTE

From figure 92 we can observe, out of 56746 total test samples, 55784 are true positives (correctly classified), 849 are false positives (misclassified), 9 are false negatives (misclassified), and 104 are true negatives (correctly classified).

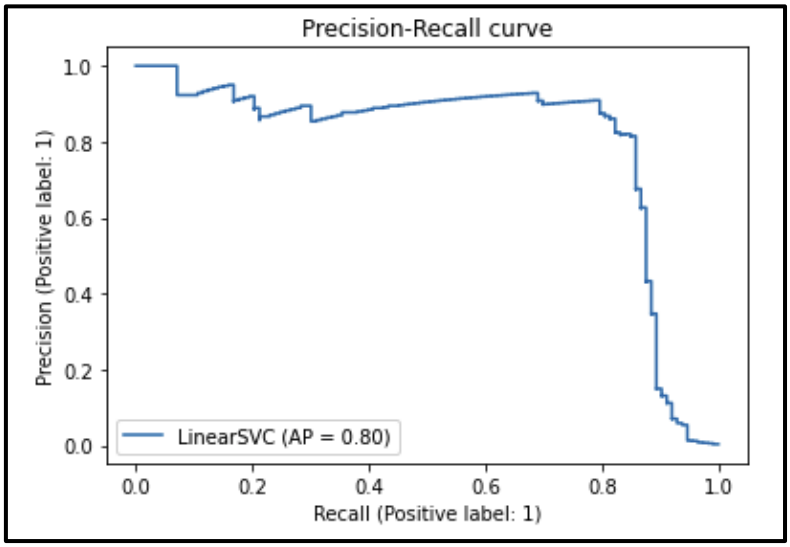
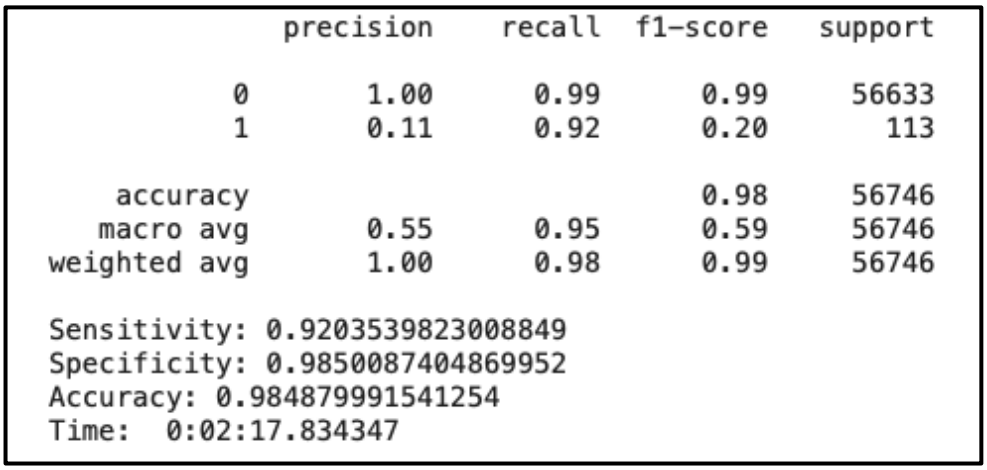


Figure 93: Precision-Recall curve of XGBoost after SMOTE



	precision	recall	f1-score	support
0	1.00	0.99	0.99	56633
1	0.11	0.92	0.20	113
accuracy			0.98	56746
macro avg	0.55	0.95	0.59	56746
weighted avg	1.00	0.98	0.99	56746

Sensitivity: 0.9203539823008849
Specificity: 0.9850087404869952
Accuracy: 0.984879991541254
Time: 0:02:17.834347

Figure 94: Classification report of XGBoost after SMOTE

From figure 93, we can deduce that our AP value is at 80%, which is moderate. At thresholds with very low precision, the recall is also at a high value as seen in figure 93 and figure 94. A precision value of only 11% indicates that when the model predicted fraud cases, it was correct 11% of the time, while a recall value of 92% indicates that the model was able to identify 92% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is 98%. Lastly, the time taken for the model to run was 2m 17s.

4.5 SVMSMOTE

4.5.1 Logistic Regression

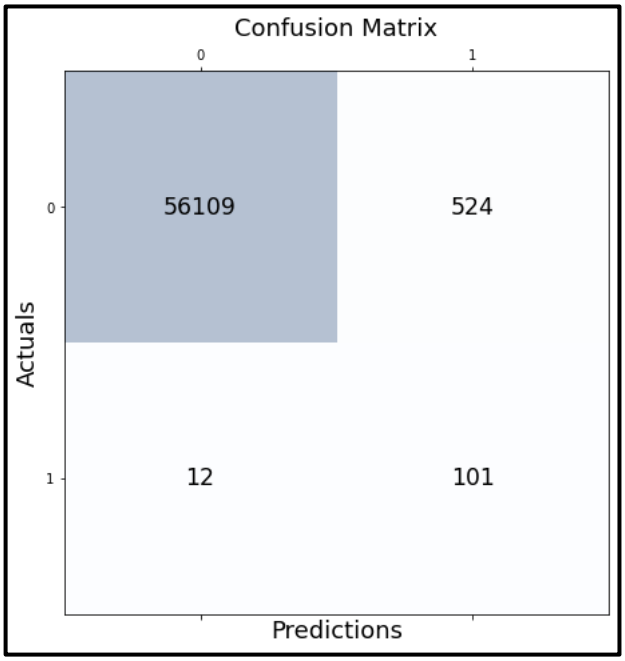


Figure 95: Confusion matric of logistic regression after SVSMOTE

From figure 95 we can observe, out of 56746 total test samples, 56109 are true positives (correctly classified), 524 are false positives (misclassified), 12 are false negatives (misclassified), and 101 are true negatives (correctly classified).

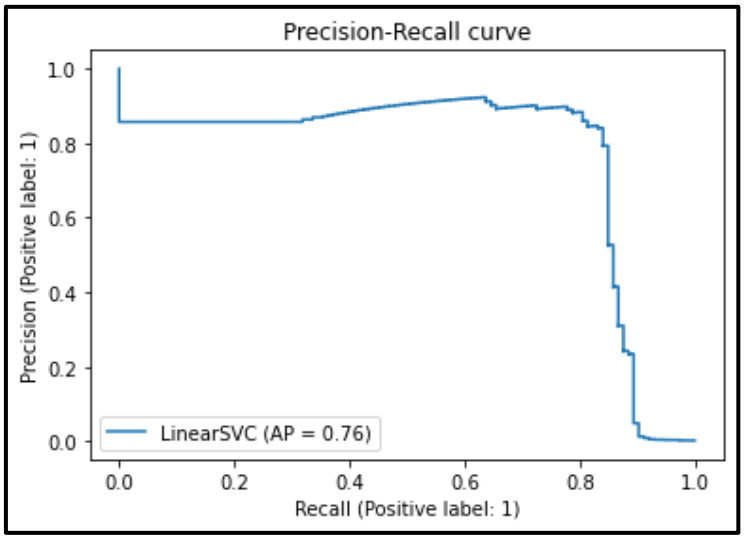
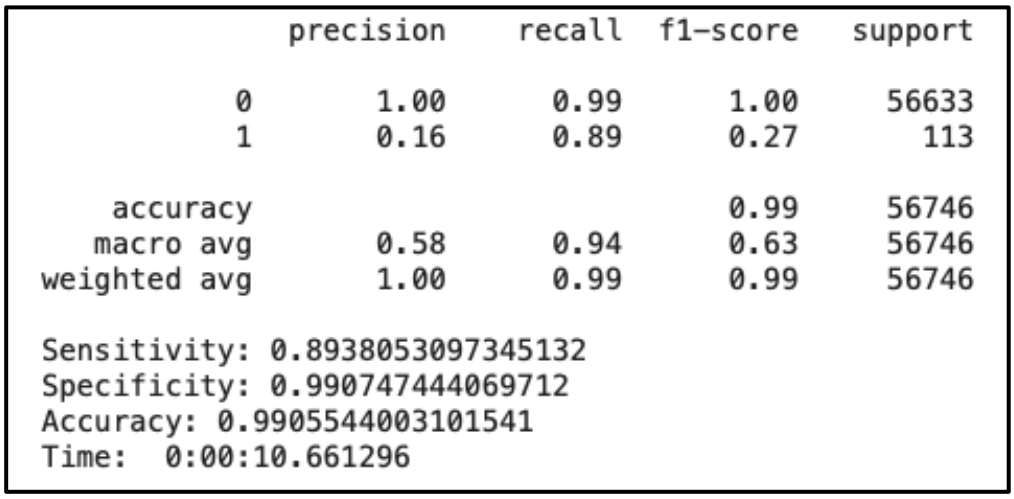


Figure 96: Precision-Recall curve of logistic regression after SVSMOTE



	precision	recall	f1-score	support
0	1.00	0.99	1.00	56633
1	0.16	0.89	0.27	113
accuracy			0.99	56746
macro avg	0.58	0.94	0.63	56746
weighted avg	1.00	0.99	0.99	56746

Sensitivity: 0.8938053097345132
Specificity: 0.990747444069712
Accuracy: 0.9905544003101541
Time: 0:00:10.661296

Figure 97: Classification report of logistic regression after SVSMOTE

From figure 96, we can deduce that our AP value is at 76%, which is rather low. At thresholds with very low precision, the recall is also at a high value as seen in figure 93 and figure 94. A precision value of only 16% indicates that when the model predicted fraud cases, it was correct 16% of the time, while a recall value of 89% indicates that the model was able to identify 89% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is 99%. Lastly, the time taken for the model to run was 10.66s.

4.5.2 Random Forest

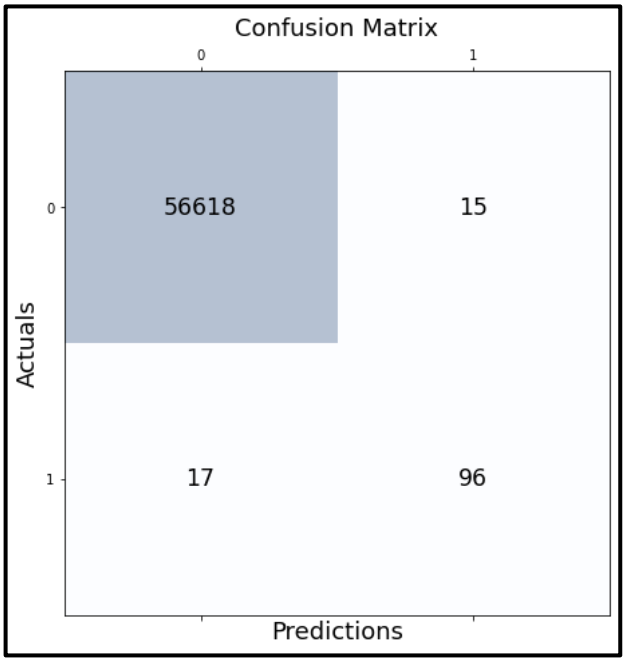


Figure 98: Confusion matrix of the random forest after SVMSMOTE

From figure 98 we can observe, out of 56746 total test samples, 56618 are true positives (correctly classified), 15 are false positives (misclassified), 17 are false negatives (misclassified), and 96 are true negatives (correctly classified).

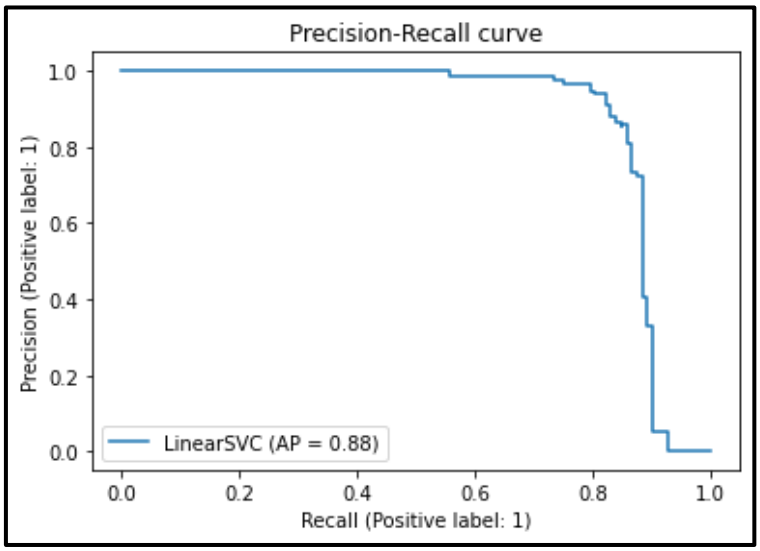
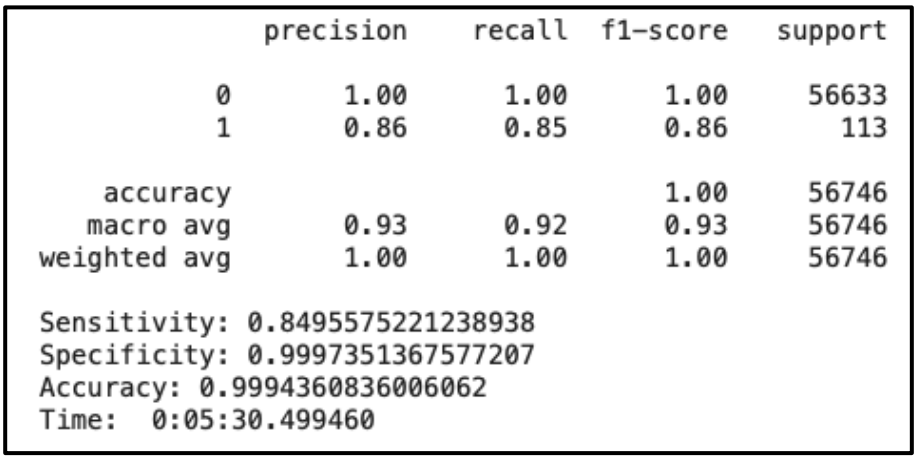


Figure 99: Precision-Recall curve of the random forest after SVMSMOTE



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.86	0.85	0.86	113
accuracy			1.00	56746
macro avg	0.93	0.92	0.93	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.8495575221238938
Specificity: 0.9997351367577207
Accuracy: 0.9994360836006062
Time: 0:05:30.499460

Figure 100: Classification report of the random forest after SVMSMOTE

From figure 99, we can deduce that our AP value is at 88%, which is high. At thresholds with high precision, the recall is also at a high value as seen in figure 99 and figure 100. A precision value of 86% indicates that when the model predicted fraud cases, it was correct to 86% of the time, while a recall value of 85% indicates that the model was able to identify 85% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run was 5m 30s.

4.5.3 XGBoost

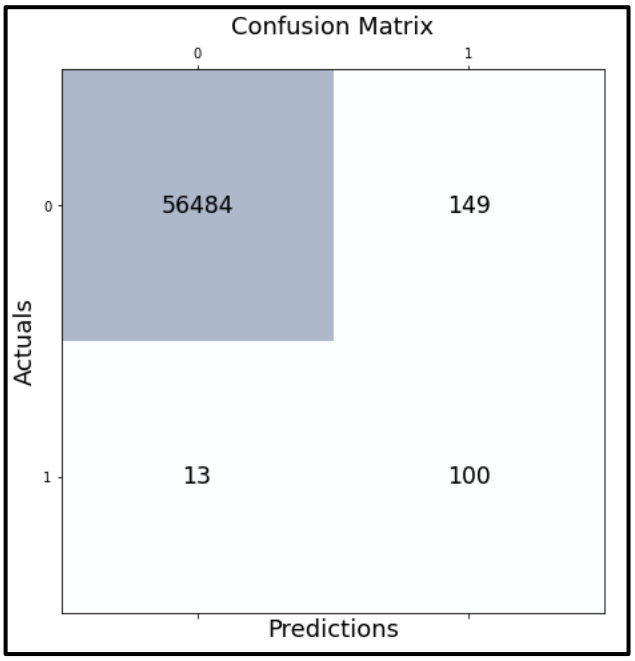


Figure 101: Confusion matrix of XGBoost after SVSMOTE

From figure 101 we can observe, out of 56746 total test samples, 56484 are true positives (correctly classified), 149 are false positives (misclassified), 13 are false negatives (misclassified), and 100 are true negatives (correctly classified).

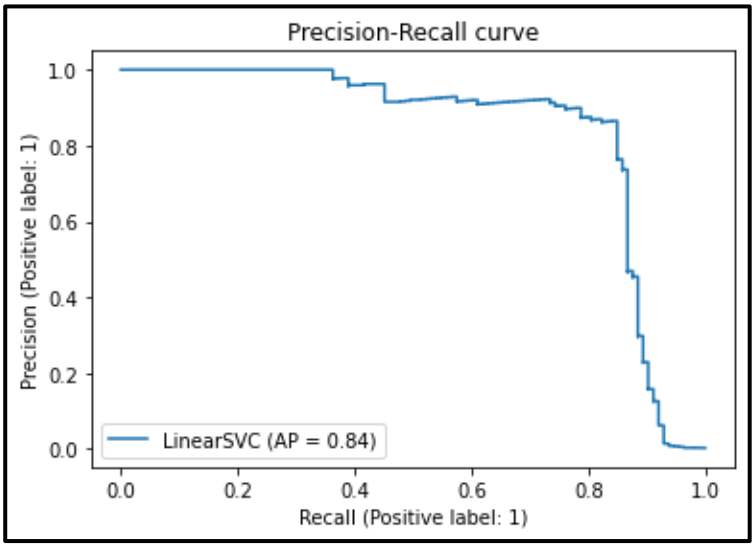
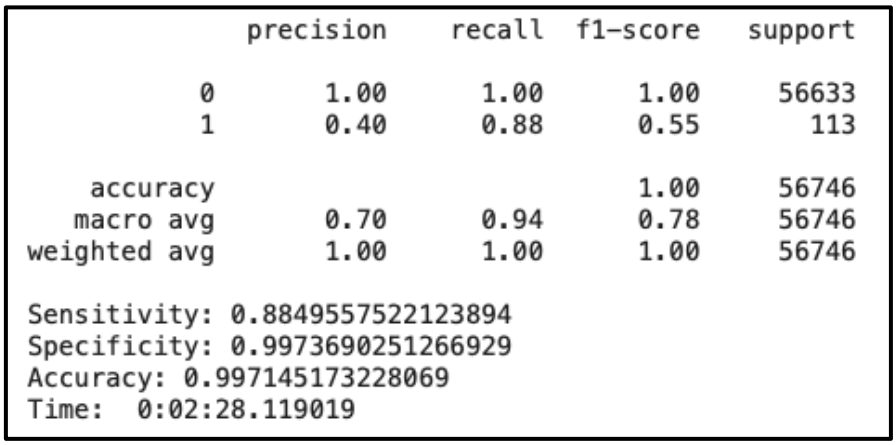


Figure 102: Precision-Recall curve of XGBoost after SVSMOTE



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.40	0.88	0.55	113
accuracy			1.00	56746
macro avg	0.70	0.94	0.78	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.8849557522123894
Specificity: 0.9973690251266929
Accuracy: 0.997145173228069
Time: 0:02:28.119019

Figure 103: Classification report of XGBoost after SVSMOTE

From figure 102, we can deduce that our AP value is at 84%, which is moderate. At thresholds with very low precision, the recall is at a high value as seen in figure 102 and figure 103. A precision value of only 40% indicates that when the model predicted fraud cases, it was correct 40% of the time, while a recall value of 88% indicates that the model was able to identify 88% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run was 2m 28s.

4.6 Hyperparameter Tuning

4.6.1 Random Forest with Random Oversampling

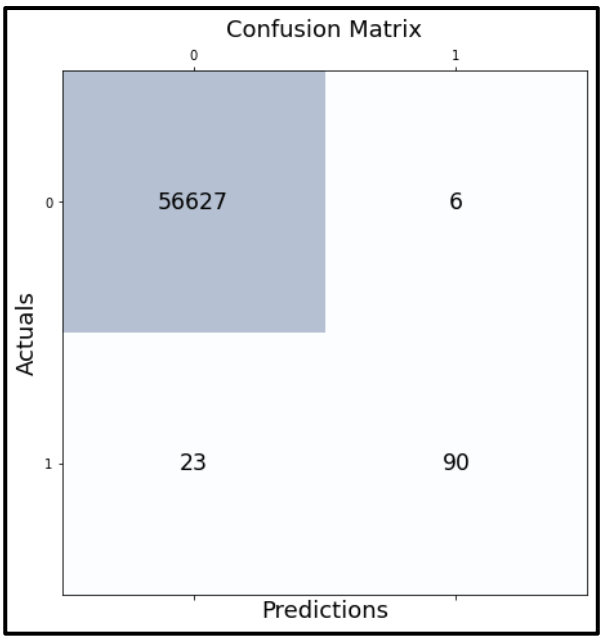


Figure 104: Confusion matrix of random forest with random oversampling after tuning

From figure 104 we can observe, out of 56746 total test samples, 56627 are true positives (correctly classified), 6 are false positives (misclassified), 23 are false negatives (misclassified), and 90 are true negatives (correctly classified).

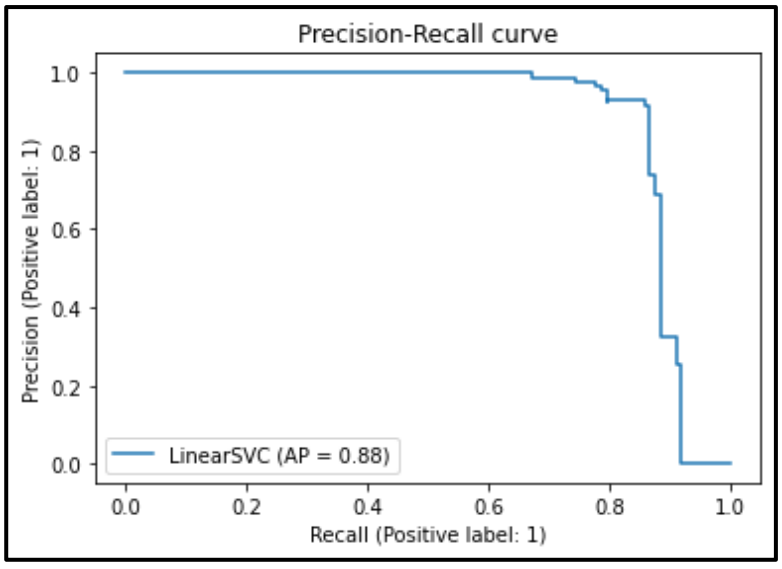
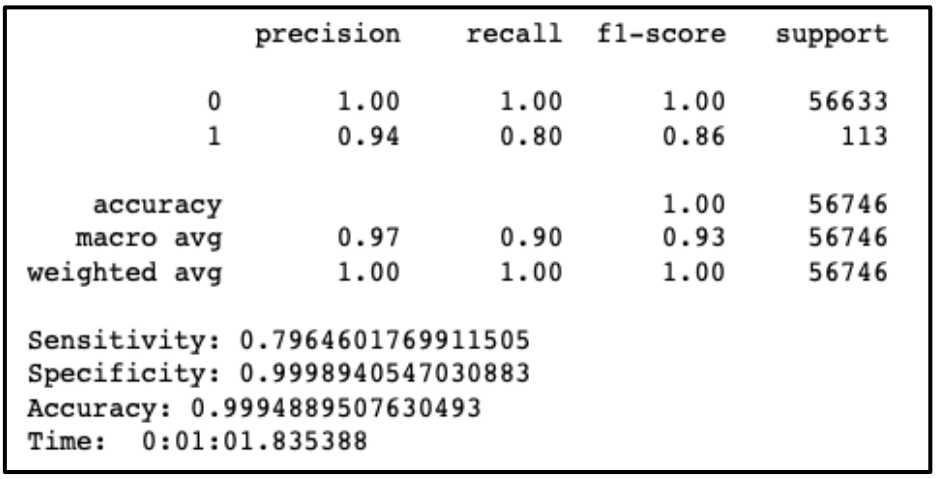


Figure 105: Precision-Recall curve of random forest with random oversampling after tuning



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.94	0.80	0.86	113
accuracy			1.00	56746
macro avg	0.97	0.90	0.93	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.7964601769911505
Specificity: 0.9998940547030883
Accuracy: 0.9994889507630493
Time: 0:01:01.835388

Figure 106: Classification report of random forest with random oversampling after tuning

From figure 105, we can deduce that our AP value is 88%, which is high. At thresholds with very high precision, the recall is at a high value as seen in figure 105 and figure 106. A precision value of 94% indicates that when the model predicted fraud cases, it was correct 94% of the time, while a recall value of 80% indicates that the model was able to identify 80% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run was 1m 1s.

4.6.2 Random Forest with SMOTE

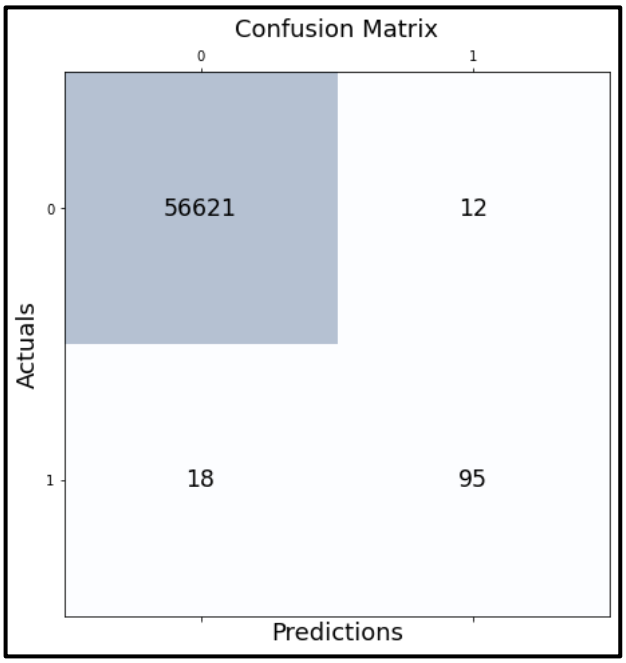


Figure 107: Confusion matrix of random forest with SMOTE after tuning

From figure 107 we can observe, out of 56746 total test samples, 56621 are true positives (correctly classified), 12 are false positives (misclassified), 18 are false negatives (misclassified), and 95 are true negatives (correctly classified).

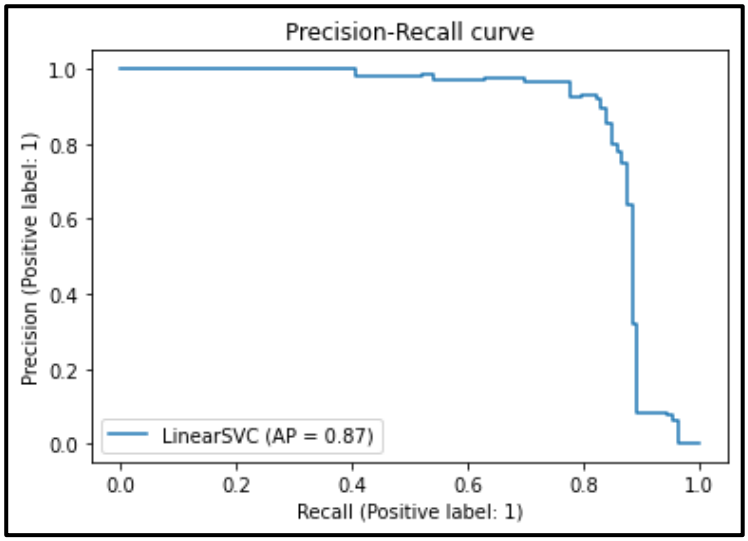
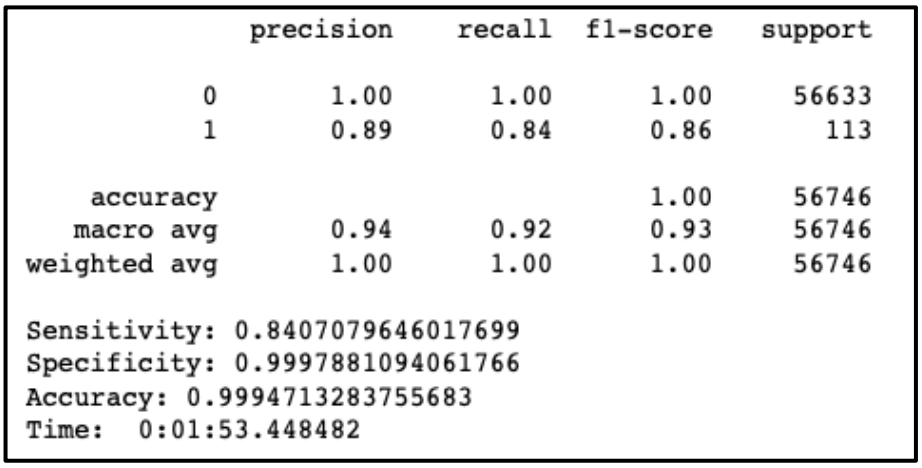


Figure 108: Precision-Recall curve of random forest with SMOTE after tuning



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.89	0.84	0.86	113
accuracy			1.00	56746
macro avg	0.94	0.92	0.93	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.8407079646017699
Specificity: 0.9997881094061766
Accuracy: 0.9994713283755683
Time: 0:01:53.448482

Figure 109: Classification report of random forest with SMOTE after tuning

From figure 108, we can deduce that our AP value is at 87%, which is high. At thresholds with high precision, the recall is at a moderate value as seen in figure 108 and figure 109. A precision value of 89% indicates that when the model predicted fraud cases, it was correct 89% of the time, while a recall value of 84% indicates that the model was able to identify 84% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run was 1m 53s.

4.6.3 Random Forest with SVMSMOTE

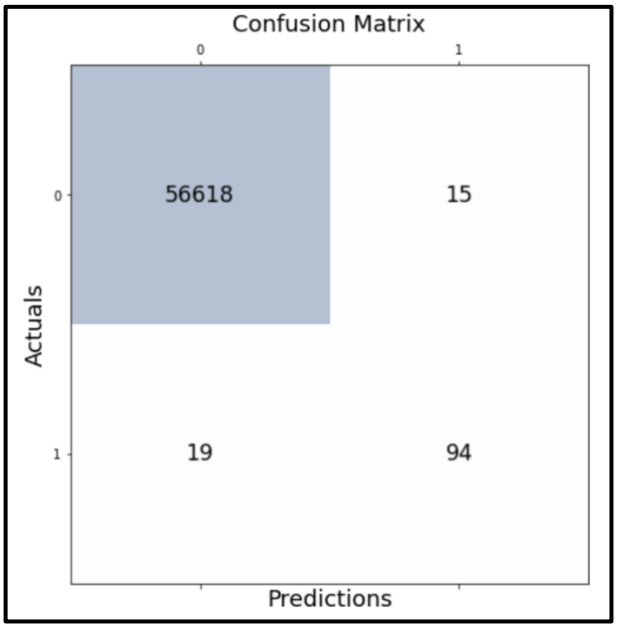


Figure 110: Confusion matrix of random forest with SVMSMOTE after tuning

From figure 110 we can observe, out of 56746 total test samples, 56618 are true positives (correctly classified), 15 are false positives (misclassified), 19 are false negatives (misclassified), and 94 are true negatives (correctly classified).

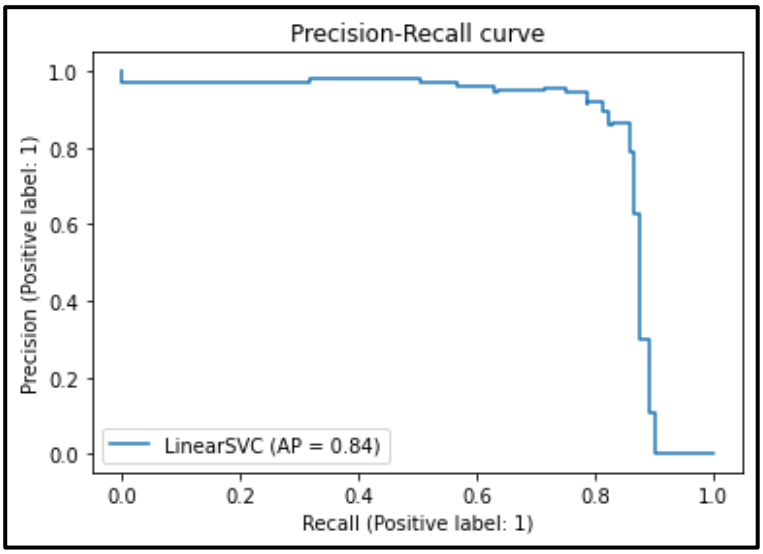
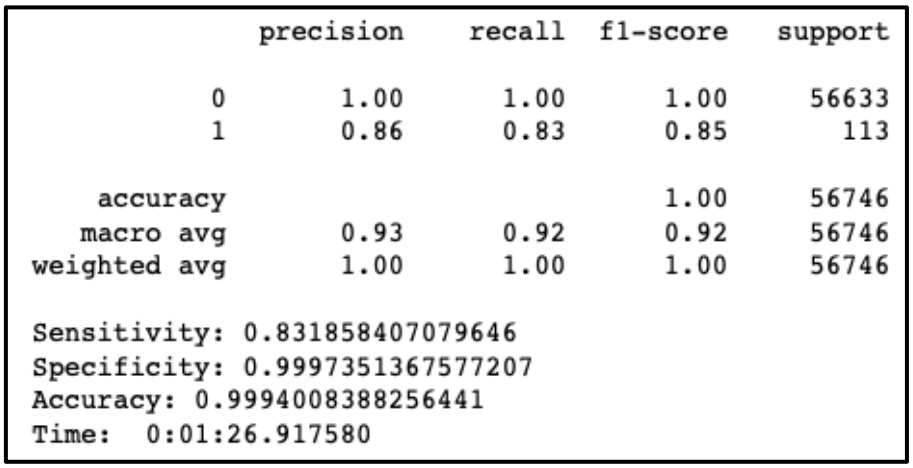


Figure 111: Precision-Recall curve of random forest with SVMSMOTE after tuning



	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.86	0.83	0.85	113
accuracy			1.00	56746
macro avg	0.93	0.92	0.92	56746
weighted avg	1.00	1.00	1.00	56746

Sensitivity: 0.831858407079646
Specificity: 0.9997351367577207
Accuracy: 0.9994008388256441
Time: 0:01:26.917580

Figure 112: Classification report of random forest with SVMSMOTE after tuning

From figure 111, we can deduce that our AP value is at 84%, which is moderate. At thresholds with high precision, the recall is at a moderate value as seen in figure 111 and figure 112. A precision value of 86% indicates that when the model predicted fraud cases, it was correct to 86% of the time, while a recall value of 83% indicates that the model was able to identify 83% of all positive cases correctly, which is rather moderate. Furthermore, the overall accuracy of the model is almost 100%. Lastly, the time taken for the model to run was 1m 26s.

Table 2: Result summary of all the models

Without Resampling							
Techniques	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**Accuracy**	**AP Value**	**Time**
Logistic Regression	61.06%	99.96%	90.00%	73.00%	99.91%	81.00%	9s 82ms
Random Forest	81.42%	99.99%	92.00%	86.00%	99.95%	87.00%	5m 9s
XGBoost	83.19%	99.99%	94.00%	88.00%	99.96%	87.00%	1m 11s
Resampling Method: Random Under-sampling							
Logistic Regression	92.92%	96.53%	5.00%	10.00%	96.53%	61.00%	42ms
Random Forest	91.15%	97.27%	6.00%	12.00%	97.27%	77.00%	2.87ms
XGBoost	92.04%	96.86%	6.00%	10.00%	96.85%	81.00%	1s 07ms
Resampling Method: Random Oversampling							
Logistic Regression	92.92%	97.29%	6.00%	12.00%	97.29%	77.00%	12ms
Random Forest	80.00%	100.00%	96.00%	87.00%	99.95%	87.00%	3m 3s
XGBoost	91.15%	99.30%	21.00%	34.00%	99.29%	79.00%	1m 45 ms
Resampling Method: SMOTE							
Logistic Regression	92.92%	96.84%	6.00%	10.00%	96.83%	77.00%	12ms
Random Forest	84.95%	99.98%	88.00%	86.00%	99.95%	87.00%	5m 52s
XGBoost	92.06%	98.50%	11.00%	20.00%	98.49%	80.00%	2m 17ms
Resampling Method: SVMSMOTE							
Logistic Regression	89.38%	99.07%	16.00%	27.00%	99.06%	76.00%	10ms
Random Forest	84.96%	99.97%	86.00%	86.00%	99.94%	88.00%	5m 30ms
XGBoost	88.50%	99.74%	40.00%	55.00%	99.71%	84.00%	2m 28ms

Table 3: Result summary of the tuned top 3 models

Resampling Method: Random Oversampling							
Techniques	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**Accuracy**	**AP Value**	**Time**
Random Forest	80%%	100.00%	94.00%	86.00%	100.00%	88.00%	1m 1s
Resampling Method: SMOTE							
Random Forest	84.07%	100.00%	89.00%	86.00%	99.95%	87.00%	1m 53s
Resampling Method: SVMSMOTE							
Random Forest	83.12%	100.00%	86.00%	85.00%	99.94$	84.00%	1m 26s

4.7 Summary of Analysis

After analysing all the models, we can conclude that the model that outperforms the rest is the random forest with SMOTE. In real-time scenarios, we cannot have both precision and recall high. If we increase the precision, it will simultaneously reduce recall and vice versa. This is known as the precision-recall trade-off. From table 2, we can see that the random forest model after resampling with SMOTE has the best-balanced trade-off with a precision of 88% and recall of approximately 85%. Precision refers to the percentage of our results being positive, and recall refers to the percentage of total positive results correctly classified by the model. Although the random forest with a random oversampling model has the highest precision rate of 96%, its recall rate is only 80%, which gives a bad trade-off between the two metrics. This tells us that the model misses a lot of the actual fraud cases, even when it manages to identify large numbers of the fraud cases. Furthermore, we can also see that the second-best model is the random forest with SVMSMOTE with a slightly lower precision rate of 86% and recall rate of almost 85%. This model also gives us a good trade-off between the precision and recall metrics. From both the models, we can clearly say that their accuracy is approximately 100%. Our best model was able to identify 56620 fraud cases out of 56746 total test samples, while our second-best model was able to detect 56618 out of 56746 total test samples. The time taken for both models to run was approximately 5 minutes, which is considered as fast. For this work, we did not consider the tuned models as the outputs for precision and recall were compromised when tested for our top three models.

Conclusion

Several advancements were influenced by the change in technology. Cybercrime, such as credit card or debit card fraud, is on the rise as the world becomes more digitalized. One of the most significant issues confronting today's financial institutions and organisations is credit card fraud. Credit card fraud detection (CCFD) is a tricky issue that necessitates the analysis of enormous amounts of transaction data to find fraud trends. Human experts are unable to effectively handle this problem due to the vast amounts of data and fraudsters' shifting strategies. Thus, fraud detection has been used in a variety of areas, including banking, commerce, financial sectors, and healthcare. Fraud detection, as we all know, is a series of efforts aimed at preventing the use of illicit ways to obtain money or property under false pretences. Detecting internet fraud has always been difficult due to the infinite and expanding number of methods fraudsters conduct fraud.

Machine learning techniques have been discovered to assist fraud analysts to locate fraudulent transactions and improving the efficiency of fraud detection systems. While different efforts have been made to address this problem, we must begin with the source of the problem, which is the imbalance dataset. The majority of credit card transactions are legitimate, while only a small percentage are fraudulent. From our work, we were able to conclude that the random forest with SMOTE model outperforms the rest with the best-balanced trade-off with a precision of 88% and recall of approximately 85%. Furthermore, we can also see that the second-best model is the random forest with SVMSMOTE with a slightly lower precision rate of 86% and recall rate of almost 85%. This model also gives us a good trade-off between the precision and recall metrics. From both the models, we can clearly say that their accuracy is approximately 100%. Our best model was able to identify 56620 fraud cases out of 56746 total test samples, while our second-best model was able to detect 56618 out of 56746 total test samples. The time taken for both models to run were approximately 5 minutes, which is considered as fast. There was not much difference between the SVMSMOTE and SMOTE approach as the SVMSMOTE is just another variant of the SMOTE technique and both approach are under the oversampling method. For future work, we propose to explore other resampling methods such as the BorderlineSMOTE and the KMeansSMOTE techniques to see if we can further improve the precision and recall scores.

References

Abdellatif, S., Ben Hassine, M. A., Ben Yahia, S., & Bouzeghoub, A. (2018). ARCID: A new approach to deal with imbalanced datasets classificationAbdellatif, S., Ben Hassine, M. A., Ben Yahia, S., & Bouzeghoub, A. (2018). ARCID: A new approach to deal with imbalanced datasets classification. Lecture Notes in Computer Science (Incl. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10706 LNCS*(February 2019), 569–580. https://doi.org/10.1007/978-3-319-73117-9_40

Adewumi, A. O., & Akinyelu, A. A. (2017). A survey of machine-learning and nature-inspired based credit card fraud detection techniques. *International Journal of Systems Assurance Engineering and Management*, *8*, 937–953. https://doi.org/10.1007/s13198-016-0551-y

Akbani, R., Kwek, S., & Japkowicz, N. (2004). to Imbalanced Datasets. *European Conference on Machine Learning*, 39–50.

Alenzi, H. Z., & Aljehane, N. O. (2020). Fraud Detection in Credit Cards using Logistic Regression. *International Journal of Advanced Computer Science and Applications*, *11*(12), 540–551. https://doi.org/10.14569/IJACSA.2020.0111265

Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision Support Systems*, *50*(3), 602–613. https://doi.org/10.1016/j.dss.2010.08.008

Biau, G. (2012). Analysis of a random forests model. *Journal of Machine Learning Research*, *13*, 1063–1095.

Bontempi, G. (2021). Foreword — Machine Learning for Credit Card Fraud detection - Practical handbook. In *Machine Learning for Credit Card Fraud detection - Practical handbook* (Issue May, pp. 11–16). https://fraud-detection-handbook.github.io/fraud-detection-handbook/Foreword.html

Bowlee, J. (2016). Logistic Regression for Machine Learning. In *Machine Learning Mastery* (pp. 1–20). https://machinelearningmastery.com/logistic-regression-for-machine-learning/

Branco, P., Torgo, L., & Ribeiro, R. (2015). *A Survey of Predictive Modelling under*

Imbalanced Distributions. 1–48. http://arxiv.org/abs/1505.01658

Brownlee, J. (2020). *ROC Curves and Precision-Recall Curves for Imbalanced Classification*. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/

Carcillo, F., Dal Pozzolo, A., Le Borgne, Y. A., Caelen, O., Mazzer, Y., & Bontempi, G. (2018). SCARFF: A scalable framework for streaming credit card fraud detection with spark. *Information Fusion*, *41*, 182–194. https://doi.org/10.1016/j.inffus.2017.09.005

Carcillo, F., Le Borgne, Y. A., Caelen, O., Kessaci, Y., Oblé, F., & Bontempi, G. (2021). Combining unsupervised and supervised learning in credit card fraud detection. *Information Sciences*, *557*(May), 317–331. https://doi.org/10.1016/j.ins.2019.05.042

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). snopes.com: Two-Striped Telamonia Spider. *Journal of Artificial Intelligence Research*, *16*(Sept. 28), 321–357. https://arxiv.org/pdf/1106.1813.pdf%0Ahttp://www.snopes.com/horrors/insects/telamonia.asp

Chawla, N. V., Japkowicz, N., & Kotcz, A. (2004). Editorial. *ACM SIGKDD Explorations Newsletter*, *6*(1), 1–6. https://doi.org/10.1145/1007730.1007733

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2018). Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(8), 3784–3797. https://doi.org/10.1109/TNNLS.2017.2736643

Dal Pozzolo, A., Caelen, O., Le Borgne, Y. A., Waterschoot, S., & Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, *41*(10), 4915–4928. https://doi.org/10.1016/j.eswa.2014.02.026

Dal Pozzolo, A., Johnson, R., Caelen, O., Waterschoot, S., Chawla, N. V., & Bontempi, G. (2014). Using HDDT to avoid instances propagation in unbalanced and evolving data streams. *Proceedings of the International Joint Conference on Neural Networks*, 588–594. https://doi.org/10.1109/IJCNN.2014.6889638

Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and ROC curves. *ACM International Conference Proceeding Series*, *148*(June 2006), 233–240. https://doi.org/10.1145/1143844.1143874

Deufel, P., Kemper, J., & Brettel, M. (2019). Pay now or pay later: A cross-cultural perspective on online payments. *Journal of Electronic Commerce Research*, *20*(3), 141–

154.

Dhankhad, S., Mohammed, E. A., & Far, B. (2018). Supervised machine learning algorithms for credit card fraudulent transaction detection: A comparative study. *Proceedings - 2018 IEEE 19th International Conference on Information Reuse and Integration for Data Science, IRI 2018*, 122–125. https://doi.org/10.1109/IRI.2018.00025

Dietterich, T. G., & Oregon. (1996). Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Models. *Oncogene*, *12*(2), pp 1-15(265-275).

Drummond, C., & Holte, R. C. (2003). Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats Over-Sampling. *Physical Review Letters*, *91*(3).

Fang, Y., Zhang, Y., & Huang, C. (2019). Credit card fraud detection based on machine learning. *Computers, Materials and Continua*, *61*(1), 185–195. https://doi.org/10.32604/cmc.2019.06144

Farooq, Q., Fu, P., Hao, Y., Jonathan, T., & Zhang, Y. (2019). A Review of Management and Importance of E-Commerce Implementation in Service Delivery of Private Express Enterprises of China. *SAGE Open*, *9*(1). https://doi.org/10.1177/2158244018824194

He, H., Zhang, W., & Zhang, S. (2018). A novel ensemble method for credit scoring: Adaption of different imbalance ratios. *Expert Systems with Applications*, *98*, 105–117. https://doi.org/https://doi.org/10.1016/j.eswa.2018.01.012

Hordri, N. F., Yuhaniz, S. S., Azmi, N. F. M., & Shamsuddin, S. M. (2018). Handling class imbalance in credit card fraud using resampling methods. *International Journal of Advanced Computer Science and Applications*, *9*(11), 390–396. https://doi.org/10.14569/ijacsa.2018.091155

Hu, S., Liang, Y., Ma, L., & He, Y. (2009). MSMOTE: Improving classification performance when training data is imbalanced. *2nd International Workshop on Computer Science and Engineering, WCSE 2009*, *2*(January 2009), 13–17. https://doi.org/10.1109/WCSE.2009.756

Itoo, F., Meenakshi, & Singh, S. (2020). Comparison and analysis of logistic regression, Naïve Bayes and KNN machine learning algorithms for credit card fraud detection. *International Journal of Information Technology (Singapore)*. https://doi.org/10.1007/s41870-020-00430-y

Japkowicz, N., & Stephen, S. ju. (1998). The Class Imbalan e Problem: A Systemati Study. *Drugs and Therapy Perspectives*, *12*(7), 10.

Java Point. (2018). *Logistic Regression in Machine Learning - Javatpoint* (pp. 1–22). https://www.javatpoint.com/logistic-regression-in-machine-learning

Kaur, K. (2021). Credit Card Fraud Detection using Imbalance Resampling Method with Feature Selection. *International Journal of Advanced Trends in Computer Science and Engineering*, *10*(3), 2061–2071. https://doi.org/10.30534/ijatcse/2021/811032021

Khatri, S., Arora, A., & Agrawal, A. P. (2020a). Card Fraud Detection : A Comparison. *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 680–683.

Khatri, S., Arora, A., & Agrawal, A. P. (2020b). Supervised Machine Learning Algorithms Card Fraud Detection : A Comparison. *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 680–683.

Koehrsen, W. (2017). Random Forest Simple Explanation. In *Medium* (pp. 1–10). https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

KPMG International. (2019). Fraud and financial scams in banking sector increase worldwide - KPMG Malaysia. In *Global Banking Fraud Survey* (pp. 1–24).

Kumari, P., & Mishra, S. P. (2019). Analysis of Credit Card Fraud Detection Using Fusion Classifiers. In *Advances in Intelligent Systems and Computing* (Vol. 711). Springer Singapore. https://doi.org/10.1007/978-981-10-8055-5_11

Last, F., Douzas, G., & Bacao, F. (2017). *Oversampling for Imbalanced Learning Based on K-Means and SMOTE*. *November*. https://doi.org/10.1016/j.ins.2018.06.056

Makki, S., Assaghir, Z., Taher, Y., Haque, R., Hacid, M. S., & Zeineddine, H. (2019a). An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection. *IEEE Access*, *7*, 93010–93022. https://doi.org/10.1109/ACCESS.2019.2927266

Makki, S., Assaghir, Z., Taher, Y., Haque, R., Hacid, M. S., & Zeineddine, H. (2019b). An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection. *IEEE Access*, *7*, 93010–93022. https://doi.org/10.1109/ACCESS.2019.2927266

Measuring Performance: The Confusion Matrix – Glass Box. (n.d.). https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/

Meng, C., Zhou, L., & Liu, B. (2020). A case study in credit fraud detection with SMOTE and XGboost. *Journal of Physics: Conference Series*, *1601*(5). https://doi.org/10.1088/1742-6596/1601/5/052016

Mînăstireanu, E.-A., & Meșniță, G. (2020). Methods of Handling Unbalanced Datasets in

Credit Card Fraud Detection. *Brain. Broad Research in Artificial Intelligence and Neuroscience*, *11*(1), 131–143. https://doi.org/10.18662/brain/11.1/19

Mittal, S., & Tyagi, S. (2020). Computational Techniques for Real-Time Credit Card Fraud Detection. In B. B. Gupta, G. M. Perez, D. P. Agrawal, & D. Gupta (Eds.), *Handbook of Computer Networks and Cyber Security: Principles and Paradigms* (pp. 653–681). Springer International Publishing. https://doi.org/10.1007/978-3-030-22277-2_26

Mohammed, R., Rawashdeh, J., & Abdullah, M. (2020). Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *2020 11th International Conference on Information and Communication Systems, ICICS 2020*, *May*, 243–248. https://doi.org/10.1109/ICICS49469.2020.239556

Mrozek, P., Panneerselvam, J., & Bagdasar, O. (2020). Efficient resampling for fraud detection during anonymised credit card transactions with unbalanced datasets. *Proceedings - 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing, UCC 2020*, 426–433. https://doi.org/10.1109/UCC48980.2020.00067

Nath, D. M. S. J. D. J. S. (2014). Credit Card Fraud Detection Using Neural Network. *International Journal of Soft Computing and Engineering (IJSCE)*, *2*(April), 84–88.

Ngai, E. W. T., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, *50*(3), 559–569. https://doi.org/10.1016/j.dss.2010.08.006

Nikhil, R. (2020). COVID-19 accelerates e-commerce growth in Malaysia, says GlobalData - GlobalData. In *Global Data* (Issue June 2020, pp. 2020–2022). https://www.globaldata.com/covid-19-accelerates-e-commerce-growth-malaysia-says-globaldata/

Niveditha, G., Abarna, K., & Akshaya, G. V. (2019). Credit Card Fraud Detection Using Random Forest Algorithm. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 301–306. https://doi.org/10.32628/cseit195261

Pozzolo, A. D. (2015). Adaptive Machine Learning for Credit Card Fraud Detection Declaration of Authorship. *PhD Thesis*, *December*, 199. https://www.ulb.ac.be/di/map/adalpozz/pdf/Dalpozzolo2015PhD.pdf%0Ahttp://www.ulb.ac.be/di/map/adalpozz/

Pozzolo, A. D., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. *Proceedings - 2015 IEEE Symposium*

Series on Computational Intelligence, SSCI 2015, *December*, 159–166. https://doi.org/10.1109/SSCI.2015.33

Prabhakaran, S. (2018). How Naive Bayes Algorithm Works ? (with example and full code). In *Machinelearningplus.com* (pp. 1–12).

Provost, F., & Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning*, *42*(3), 203–231. https://doi.org/10.1023/A:1007601015854

Python Logistic Regression with Sklearn & Scikit - DataCamp. (n.d.). https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

Quah, J. T. S., & Sriganesh, M. (2008). Real-time credit card fraud detection using computational intelligence. *Expert Systems with Applications*, *35*(4), 1721–1732. https://doi.org/10.1016/j.eswa.2007.08.093

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/bf00116251

Sailusha, R., Gnaneswar, V., Ramesh, R., & Ramakoteswara Rao, G. (2020a). Credit Card Fraud Detection Using Machine Learning. *Proceedings of the International Conference on Intelligent Computing and Control Systems, ICICCS 2020*, *Iciccs*, 1264–1270. https://doi.org/10.1109/ICICCS48265.2020.9121114

Sailusha, R., Gnaneswar, V., Ramesh, R., & Ramakoteswara Rao, G. (2020b). Credit Card Fraud Detection Using Machine Learning. *Proceedings of the International Conference on Intelligent Computing and Control Systems, ICICCS 2020*, 1264–1270. https://doi.org/10.1109/ICICCS48265.2020.9121114

Saxena, R. (2017). How The Naive Bayes Classifier Works In Machine Learning. In *Dataaspirant* (pp. 1–12). http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/

Somasundaram, A., & Reddy, S. (2019). Parallel and incremental credit card fraud detection model to handle concept drift and data imbalance. *Neural Computing and Applications*, *31*, 3–14. https://doi.org/10.1007/s00521-018-3633-8

Swamidass, S. J., Azencott, C. A., Daily, K., & Baldi, P. (2010). A CROC stronger than ROC: Measuring, visualizing and optimizing early retrieval. *Bioinformatics*, *26*(10), 1348–1356. https://doi.org/10.1093/bioinformatics/btq140

Vakulenko, Y., Shams, P., Hellström, D., & Hjort, K. (2019). Online retail experience and customer satisfaction: the mediating role of last mile delivery. *International Review of Retail, Distribution and Consumer Research*, *29*(3), 306–320.

https://doi.org/10.1080/09593969.2019.1598466

Varmedja, D., Karanovic, M., Sladojevic, S., Arsenovic, M., & Anderla, A. (2019). 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH) : proceedings : March 20-21, 2019, Jahorina, East Sarajevo, Republic of Srpska, Bosnia and Herzegovina. *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, *March*, 1–5.

Zhu, R., Guo, Y., & Xue, J. H. (2020). Adjusting the imbalance ratio by the dimensionality of imbalanced data. *Pattern Recognition Letters*, *133*, 217–223. https://doi.org/10.1016/j.patrec.2020.03.004