COMPARATIVE ANALYSIS USING DIFFERENT SAMPLING METHODS FOR CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING

NAME: WALID RAAD

STUDENT ID: 210191462

SUPERVISOR: DR. VIKTOR PEKAR

Dissertation submitted to Aston Business School, Aston University

In partial fulfilment of requirements for MSc in Business Analytics

Acknowledgement

I would like to express my heartfelt appreciation to Dr. Viktor Pekar for his unwavering support and guidance throughout my research project and academic journey here at Aston, for which I will be eternally grateful.

I would also like to thank my parents and siblings for their unconditional moral and financial support. Without you, none of this would have been possible. Thank you so much; forever grateful for you all.

Table of Content

Table of Figures and Tables

Figures:

Tables:

Abstract

Ever since the arise of covid 19, a large portion of the population shifted to online shopping and contactless payments resulting in a huge spike in the usage of credit card. Due to this increase, cases in credit card fraud have also been on the rise. Many difficulties arise when tackling credit fraud using data mining techniques, one of them is class imbalance.

The aim of this paper is to conduct an extensive comparative study on different sampling methods namely random oversampling, under sampling, SMOTE and ADASYN along different algorithms and observe if any sampling method do play a significant role in improving model's ability in detecting fraud.

The result of our study showed that sampling methods are not effective on extremely imbalance datasets in improving our model's ability in better predicting fraudulent transactions. Most balancing methods when applied were making the model more susceptible in producing more false prediction in fraud, especially for distance-based algorithms such as KNN and SVM.

Chapter 1 Introduction

This chapter includes a brief discussion on the rising concern of credit card fraud, the importance of tackling it, the issues that arise with creating an effective credit fraud detection system and the aim of my research project.

1.1 Problem Background

The use of digital and contactless payments expanded enormously during covid 19, and this ongoing trend has continued to develop even after covid (Koss and Szemere,2021). In 2020, the UK reduced its use of physical cash by 60%, while online shopping increased significantly, accounting for 28% of all sales, a 19% increase from the previous year (Whatman, 2022). As a result, this change has resulted in a sharp rise in the number of credit and debit card frauds, subsequently causing the financial sectors to acquire lots of monetary losses and a lack of customer trust in the financial sector. There are a few popular ways where credit card fraud happens:

1. **Stolen/Lost Cards**: This type of fraud is when the fraudster steals or finds someone's physical card and makes several purchases without their consent. It’s easy for the fraudster to use the stolen credit card since most shops do not require pin codes for small transactions. Therefore the fraudster can keep purchasing until the card owner realizes and blocks it immediately.
2. **Stolen card information:** This fraud often occurs when a fraudster steals the cardholder's information, such as name, credit card number, CVV number, and expiration date, and uses them to make online purchases.
3. **Skimming:** This is a popular approach where fraudsters use a device known as a skimmer that is injected into the card reader of an ATM, and it collects information from the card's magnetic strip, and then the fraudster takes the device and decodes it to collect the user's personal information to use it for online purchases.

1.2 Increase in fraud cases over time

In the UK alone, in 2021, credit card fraud accounted for nearly 40% of all the fraud cases that occurred in the country (Maps,2021), causing the banking sector and credit card companies to spend millions of dollars in reimbursements every year. According to the Federal bureau of investigation and internet crime (IC) report, in 2021, they received 16,750 complaints with a total loss of around 173 million dollars in the US alone. Another report from the annual fraud statistics published by The Nilson Report, 2019 forecasting that by 2027 the global financial institutions are expected to hit a loss of 40 billion dollars loss in credit card fraud which is a 43.6% increase from 2018, which was around $27.85 billion in credit loss.

1.3 Obstacles faced in creating a credit fraud detection system

A lot of pressure has been put on the financial sector to develop better fraud detection systems that can detect fraudulent transactions more precisely. Therefore, it has been a hot topic for many researchers around the world to find ways to detect fraudulent transactions with high precision. however, it is challenging to publicly find real-world datasets that contain real insights/information about customer transactions due to confidentiality reasons which make it difficult for researchers around the world to conduct experiments and come with real solutions (Tuyls, Maes and Bram, 2022). A credit card fraud detection system comes with a lot of complexity, and the fraud detection system should be able to tackle the following:

1. Have low latency. Meaning transactions need to be processed and come up with a decision in milliseconds to the end user.
2. Have the ability to tackle heavily imbalanced datasets where most of the data points are genuine while only a tiny proportion is fraudulent, making it difficult for simple models to learn from (Tuyls, Maes and Bram, 2022).
3. Fraudsters tend to constantly change their tactics and behavioural patterns, making it difficult for humans or conventional systems such as rule-based methods to adapt to those changes; therefore, the system should be able to adapt to those changes (Tuyls, Maes and Bram, 2022).
4. Should be able to select valuable features that would enhance the model and eliminate noise.
5. Should be able to predict fraudulent transactions with high precision without compromising on falsely predicting genuine transactions

1.4 Aim of the Project

This study aims to shed light on an area of high interest in relation to credit card fraud detection – Sampling methods. This study identifies the most prominent sampling techniques currently available and aims to observe and record their behavior of these in the context of fraud detection.

This study shall help gain advancements with the recorded usage and comparison of the various sampling techniques with reference to the dataset described in section 3.1 such that a step forward can be taken in the ongoing research on fraud detection and if they do create an impact in better detecting credit fraud.

1.5 Study Questions

Below are the questions we intend to address during our research study.

- Which balancing technique is most appropriate for detecting fraud?
- Do the models perform better once balancing methods are applied?
- Which algorithm works best in detecting fraudulent transactions without having high false predictions for genuine transactions?
- What features contribute the most to detecting fraud?

1.6 Project Objectives

Below are the general objectives we intend to follow to answer the research questions.

- Do extensive exploratory data analysis to help us understand the data and find any hidden patterns that will aid us in creating better models.
- Extract feature importance and analyze the features that contribute the most to detecting fraudulent transactions.
- Apply different balancing methods to our models
- Build machine learning models to predict if a particular credit card transaction is a fraud or not.
- Use a grid search function for hyperparameter tuning to find the optimal parameters for the best possible results.

Chapter 2 Literature Review

In this chapter, we are going first to review the general fields of credit card fraud and discuss what previous researchers have contributed to detecting credit card fraud and identify any relevant gaps that would need to be further experimented with.

2.1 Credit Card Fraud

By definition, fraud is the deception of someone's persona for personal or financial gain (Bolton and Hand, 2002). According to (Laleh and Azgomi, 2009), credit card fraud can be classified into offline and online credit card fraud. Offline credit card fraud is when a fraudster steals a physical card and uses it to purchase things from stores as the actual owner. However, this type of fraud is easily detectable since the card owner can immediately report the loss of their card, and it gets blocked instantly. On the other hand, online credit cards are a more familiar and harder-to-detect type of fraud involving the fraudster stealing a person's credit card information and using it to pursue online transactions in tiny amounts without the user's awareness.

Historically, the conventional way of detecting credit fraud was by using rule-based methods applied by humans to detect fraudulent behaviours (Fu et al., 2016). However, this method has become inadequate since it does not consider crucial cases such as extreme class imbalance of the target variable and the fact that fraudsters are always looking for new ways to trick the system which must be updated and adapted to these changes (Pozzolo et al., 2014). With the rise of data and computational power, the world has shifted to a new era of ways to detect fraud using data mining techniques. Data mining is the process that uses machine learning, artificial intelligence, and statistical methods to extract data, identify hidden patterns and develop new class models that can detect fraudulent cases without the interference of human experts (Ngai et al., 2011).

Databases which credit card transactions are extracted from usually contain a mix of continuous and categorical features. The classic continuous feature we get is the transactional amount the person credited, while the categorical features are the merchant name, merchant code, merchant city, merchant country, date and time of the transaction, etc. Some of those categorical features in data sets can contain hundreds and thousands of categories. Therefore, many challenges can arise when dealing with attributes with many categorical and few numerical features; this is where leveraging statistical and data mining techniques are essential in making intelligent tools and decisions (Bhattacharyya et al., 2011).

2.2 Class Imbalance

Extreme class imbalance is one of the fundamental issues faced when building a fraud detection system using machine learning models. Class imbalance occurs when the target variable has very few observations for one class and a lot for another. It is the case in fraud detection, where very few transactions are fraudulent compared to genuine. Many methods have been approached by researchers on ways to handle this problem of class imbalance.

The paper, Exploratory under-sampling for class-Imbalance Learning by Liu et al. (2009) introduced how to use the under-sampling method efficiently to tackle class imbalance without ignoring or removing many instances in the majority class. He proposed two types of ensemble methods to tackle the deficiency that comes with using under-sampling. The first method is EasyEnsemble which takes several subsets from the majority class and trains the model using each subset and combines the predictions of all the models. The second method is BalanceCascade, it works by training the model sequentially, and in every step, if an instance in the majority class is classified correctly, then it is removed in the following training model. The results showed that both ensemble methods performed really well. They achieved a higher Area under the curve ROC score than many other class imbalanced learning methods with faster computational time.

In another paper by Elrahman and Abraham (2013), they introduced several methods to deal with class imbalance. During the sampling methods, they first used random oversampling and then tried under-sampling. However, they did not recommend Under sampling since it removed a lot of instances from the majority class, and random oversampling since it introduced overfitting to the model and increased its computational time. Finally, they proposed a synthetic minority oversampling technique (SMOTE), which creates synthetic data for the minority class rather than replicating them. This technique performed well on classifiers and eliminated bias towards the minority class.

A paper by Mishra and Ghorpade (2018) also conducted different approaches to balancing the data by using random oversampling, random under-sampling, and SMOTE techniques to predict fraudulent transactions with high accuracy. Under-sampling proved to be the best balancing method, along with random forest, which is an ensemble model, in detecting fraudulent transactions achieving a high recall score. However, it achieved a very low precision of only 5%, meaning that the model was not able to identify genuine transactions accurately.

In a study conducted by Zhang and Qin (2022), he mentioned how Extreme learning machine (ELM), which is a single hidden layer feedforward neural network algorithm, has a low computational time and great performance. However, it performed poorly on imbalanced class labels. He instead introduced an improved method of ELM to deal with the class imbalance called OWA-ELM, where OWA is short for output weight adjusted. It works by incrementally increasing the connection weights between the minority output neurons and the hidden neurons and moves the decision boundary of the algorithm to the majority class, which improved the performance significantly. The adjusted algorithm OWA-ELM was compared with other algorithms such as ELM, weight extreme learning machine (WELM), class-specific cost regulation (CCR-ELM), and CS-ELM. OWA-ELM was able to predict the minority class with high accuracy without affecting the score of the majority class and it outperformed all other mentioned algorithms.

2.3 Predictive Methods

When building fraud detection models, we are dealing with a classification problem that is part of supervised machine learning. A problem is considered a classification problem when a predictive model is being built to predict a certain class label from a given input. A lot of predictive methods have been experimented with in the literature for fraud detection, such as logistic regression, naïve Bayes, random forest, etc., that will be discussed.

A study was conducted by Srivastava et al. (2008) to prove the effectiveness of detecting credit card fraudulent transactions using the Hidden Markov Model. The method implemented was by categorizing the user's transactions into low, medium, and high spenders groups using the K-means clustering algorithm and then applying initial probabilities for every spending group creating a pattern from the training data and trains the model. The model proved to be easily scalable, has fast computation, and is effective in tracing the transaction pattern of users. The accuracy of the model has achieved an accuracy score of close to 80%.

In another study done by Maes et al. (2002), they experimented with two machine learning models that are artificial neural network (ANN) and Bayesian belief network, in order to detect fraudulent transactions. Bayesian belief network performed better than ANN achieving a 68% recall score with 10% false positive. However, in an article conducted by (Fu et al., 2016), they criticized that ANN and bayesian belief network are complex to detect fraud and are more likely to occur overfitting. They instead proposed a convolutional neural network model to decrease predictors redundancy effectively.

Bhattacharyya et al. (2011) performed a study on detecting credit card transactional fraud by using two machine learning models which are random forest, and logistic regression. They used different levels of under-sampling of the data to tackle class imbalance in the training set and reduced the percentage of fraud in the testing set so that the performance of the models when applied can be expected to be the same when deployed. All three models showed adequate results with random forest exhibiting a better overall performance resulting in identifying 90% of all fraud cases while logistic regression identifies 80% of all fraud cases.

Varmedja et al. (2019) performed a study on fraud detection where they focus on using different machine learning models with extensive hyper tuning in order to predict fraudulent transactions with high accuracy and recall scores. They used the permutation feature selection method retrieved from GitHub and eliminated the features that have no significance in predicting the outcome has been eliminated. He then used SMOTE technique for balancing the target variable. During the modeling stage, they applied logistic regression, random forest, naive Bayes, and multilayer perceptron models and chose the best-performing one. The result showed that random forest gave the best outcome in predicting whether the transaction is fraudulent or not with high precision of 83%.

2.4 Feature Engineering

As with any other machine learning problems, detecting credit card fraud relies a lot on including relevant features into the feature set. The features used to allow our model to learn from are what distinguishes a successful model from a failed one.. The model can easily learn from the feature and identify the patterns if they correlate well with our target class however if the target class has a complicated model with its features then it can be difficult to learn from it and thus produces bad results (Domingos, 2012). The aim of feature engineering is to create new features from existing ones with the goal of improving the model's accuracy and learning better without affecting the transformation of data and its computational speed (Patel, 2021).

In a paper conducted by Fu et al. (2016), they introduced a new type of feature engineering that can describe the complicated patterns of consumer spending for detecting credit card fraud using convolutional neural network (CNN). They began by tackling the imbalanced problem using SMOTE technique and then introduced the new feature called trading entropy, where trading entropy gives out the expected fraud every time a transaction occurs for a type of merchant by a customer. The larger the number of entropy is the higher the probability of the transaction being fraudulent. The model's accuracy has significantly improved by adding a trading entropy feature to the model. After tackling the imbalanced problem and adding the engineered feature, they applied the CNN algorithm and compared it with other traditional models, and it achieved the best performance out of all other models.

In another paper conducted by Correa Bahnsen et al. (2016), they introduced different types of feature engineering strategies for credit card fraud detection by applying different types of aggregation techniques on transactions and other engineered features that are able to analyze the periodic consumer behaviour to improve the performance of the models through feature engineering. They added a new set of aggregated features that allows complicated relations of the data to be established. Also, other features were added that enables periodic consumer behaviour to be analyzed using Von Mises distribution by the time of transaction happens. Adding those aggregated features was able to improve the performance of the models used by more than 200%.

2.5 Feature Selection

Having irrelevant features in our machine learning model can deteriorate the performance of predicting credit card fraud. Feature selection is the process of selecting a subset of features that contributes the most to predicting a certain outcome, and it is a crucial step when developing your model. Particularly in unbalanced data sets, a high dimensional data set and irrelevant predictors introduces overfitting and may hinder the effectiveness of the classifier and enhance the misclassification rate (Elrahman and Abraham, 2013). Elrahman and Abraham (2013) also indicates that feature selection can be categorized into two metrics, one as binary and the other as continuous, depending on the type of features, where chi-square and odd ratio methods handle nominal and binary data while Pearson correlation coefficients handle continuous data.

According to Noghani and Moattar (2017), the three common methods into which feature selection can be divided are filter, wrapper, and embedded methods. A filter approach, which is unaffected by the algorithm selected to solve the problem, picks out features one at a time and evaluates them in accordance with a predetermined standard, while Wrapper methods select features based on important variables that contributed to the highest prediction based on the selected algorithm used (Noghani and Moattar, 2017). However, wrapper methods are less commonly used than filter methods due to the fact that wrapper methods require a lot of computational power and can be very hard to deal with large data sets (Noghani and Moattar, 2017). Feature selection in embedded methods is similar to wrapper methods, where the method selects a subset of features from the data set which have strong predictive properties (Noghani and Moattar, 2017). Embedded methods are feature selection properties that are integrated into the algorithm itself; however, not all algorithms have this property. The most commonly used embedded methods are random forests and XG boost (Noghani and Moattar, 2017).

In a paper done by Marko (2019), they experimented with how naïve model classifier performed on 11 different feature ranking methods using a textual dataset extracted from the yahoo website. After checking the performance of the classifier on the subset of features selected, the results showed that the highest performance achieved using F1 and recall metrics was by Odd Ratio.

Saheed et al. (2020) conducted a feature selection experiment on applicational credit card fraud data set using a genetic algorithm (GA) which is an algorithm that belongs to evolutionary algorithms. The genetic algorithm feature selection method was experimented with twice; where during the first try, the top 8 features that have been selected by the genetic algorithm were chosen. On the second try, again, the top 8 features were selected. Those two subsets of features were experimented on naïve Bayes, random forest, and support vector machine algorithms in detecting fraud. The first subset of features achieved greater results than the second one, having sensitivity, precision, and accuracy scores all greater than 85%.

In another paper conducted by Forman, Guyon and Elisseeff (2003), they experimented with how support vector machines would perform using 12 feature ranking methods using a text mining unbalanced binary data set. He measured the performance of the trained support vector machine using different metrics such as precision, F1 and recall. Out of the 12 feature ranking methods experiment, the Bi-Normal separation ranking method has achieved the highest score in the selected features.

In a paper conducted by Chen and Wasikowski (2009), he proposed a method called feature assessment sliding thresholds (FAST) which is a ranking feature selection method that is derived from AUC (Area under the curve), ROC (Reciever operating characteristics) curve and it works by sliding the decision boundaries of the selected feature to its selected threshold that is derived from an even bin distribution of the predictor instances. The way even bin distribution is approached is by dividing the instances evenly by the number of bins selected. The threshold is the average number of instances in each bin. The experiment found that the FAST method was nearly 10x faster, when the number of bins is equal to 10 the estimated area under the curve (AUC) is very similar to the actual area under the curve score while considering all possible thresholds. The FAST method was compared with the RELIEF method and correlation coefficient method. The result showed that FAST method outperformed them both on microarray, text mining, and spectrometry data.

2.6 Summary of literature review

Based on the thorough literature review we have examined; a lot of academic research has been conducted concerning credit fraud. Most researchers agree on the following:

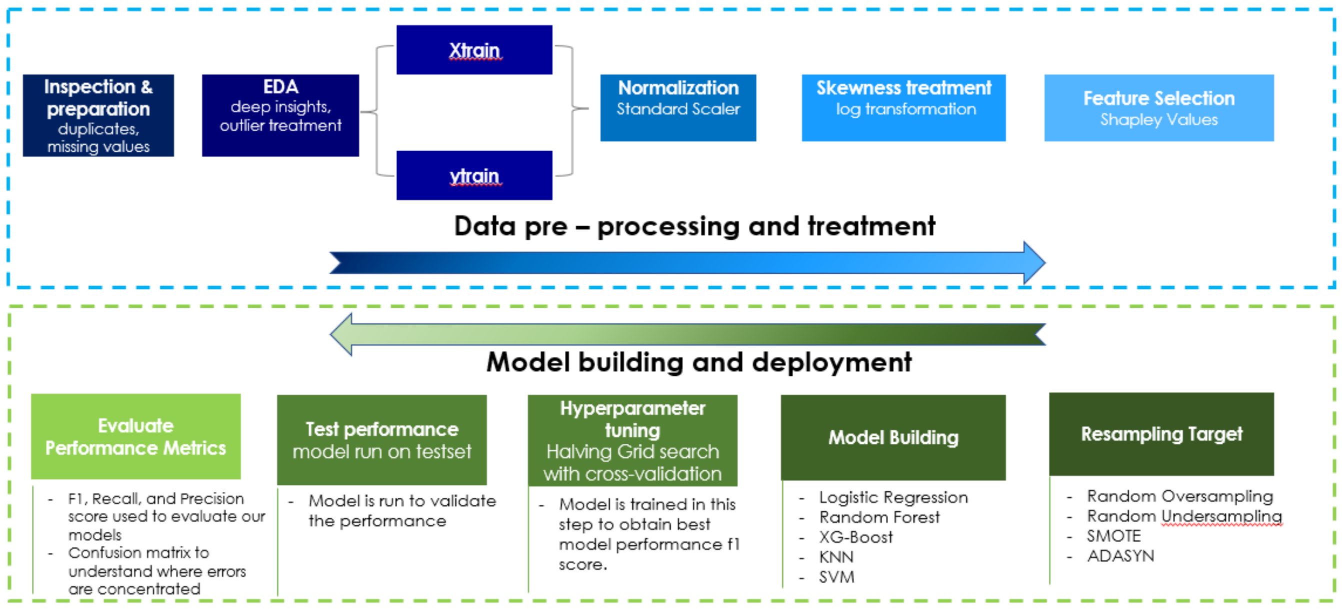
- Class imbalance is an important obstacle to tackle in fraud detection, and that balancing methods such as under-sampling and SMOTE were able to improve the models in detecting credit fraud.
- Applying feature selection methods by removing irrelevant features has a significant impact on improving modelling performance in detecting fraud.
- Feature engineering can have a great impact on improving fraud detection by creating new features through existing ones.

However, it is not clear yet which is the best balancing method to use for class imbalance in detecting credit fraud. In the literature, some people recommended SMOTE while others recommended Under-sampling. Hence, I am going to be experimenting with both methods alongside different balancing methods and examine which balancing method is best suited for tackling class imbalance and if it really does have a huge impact on the model’s performance

Chapter 3 Methodology

This chapter describes in detail the methodology used and why it has been used in building the fraud detecting system.

The figure below displays in a graphical manner the methodology adopted for this research project.



Inspection & preparation
duplicates, missing values
EDA
deep insights, outlier treatment
Xtrain
ytrain
Normalization
Standard Scaler
Skewness treatment
log transformation
Feature Selection
Shapley Values
Data pre – processing and treatment
Model building and deployment
Evaluate Performance Metrics
- F1, Recall, and Precision score used to evaluate our models
- Confusion matrix to understand where errors are concentrated
Test performance
model run on testset
- Model is run to validate the performance
Hyperparameter tuning
Halving Grid search with cross-validation
- Model is trained in this step to obtain best model performance f1 score.
Model Building
- Logistic Regression
- Random Forest
- XG-Boost
- KNN
- SVM
Resampling Target
- Random Oversampling
- Random Undersampling
- SMOTE
- ADASYN

FIGURE 1: METHODOLOGY

The above diagram is designed to achieve the objectives listed in section 1.6.

3.1 Data Description

The data set used in this research for our proposed model was obtained from Kaggle, which has been previously collected by the University Libre de Bruxelles in Belgium and covers credit card transactions from European cardholders in September 2013. The data was accumulated over a 48-hour period that comprised of 284,807 transactions, 492 of which were fraudulent, and the remaining 284,315 were legitimate. This data set that is obtained is severely unbalanced, where only 0.172% of the two classes are labelled as fraud while the other 99.828% are labelled as genuine transactions.

The dataset has already undergone some pre-processing where feature transformation has been done to preserve customer confidentiality by applying principal component analysis (PCA) for features V1, V2, V3, V4, V5, …., V28. PCA is a dimensionality reduction technique that is used to reduce datasets with a large number of features. The way it works is it creates new artificial components or features that are not correlated with each other while pertaining the same information of the original datasets but with fewer features (Richardson, 2009). Therefore, all the predictors in our data set are continuous variables, and the only features that have not been transformed through PCA are variables “Time” and “Amount”. The feature "Time" shows the time elapsed between the initial transaction and the current one, whereas the feature "Amount" represents the transactional amount credited by the individual. Finally, the target variable in the data is the variable “Class” it contains two classes which are represented as 0 (Non-Fraud) and 1 (Fraud).

TABLE 1: DATA DESCRIPTION

Feature	Description	Data Type
Time	Time elapsed in seconds between the current transaction and the first one.	Numerical
V1, V2, V3, V4, V5, …, V28	Those features has been transformed using PCA in order to protect customer confidentiality.	Numerical
Amount	Transaction amount credited by the individual.	Numerical
Class	The labelled outcome of the transaction as 1 for fraud and 0 for genuine	Categorical

3.2 Data Inspection and Preparation

Before we begin with any extensive data exploration, it is essential to make sure that our data is clean and empty of any errors. However, since the data has already undergone some pre-processing previously, as mentioned in the data description, there wasn’t much cleaning to be done. Here are the following steps which I took to ensure our data is empty of errors:

- Obtained the shape of the array to check if it matches the data description that was read on Kaggle
- Checked the first five rows of the data to get a feel of it and ensure all columns were correctly labelled
- Checked for missing values, and none were detected

- Checked for duplication, and it was detected
- Dropped duplicated rows

3.3 Exploratory Data Analysis

Before building any models, it is critical to understand the data we are dealing with and explore any hidden patterns or insights and spot any anomalies in our data through visualizations and descriptive statistics. Exploratory data analysis will aid us in achieving all this and form any hypothesis based on our findings and understanding of the data in order to build a good model.

3.3.1 Univariant Analysis

A great way to begin our data exploration is by visualizing each variable in order to understand the distribution of the features and quantify our features in an understandable way. However, in our case we will not be able to explore all our variables since features V1-V28 have been transformed using the principal components analysis.

3.3.1.1 Class (Target Variable)

It is vital to begin with exploring our target variable first in order to understand the type of classification problem we are dealing with. Since it is a classification problem with two possible outcomes (Fraud or Genuine), we will plot the target variable on a bar plot using seaborn package from python.

From the below bar plot, we can observe that our target variable is highly imbalanced where 99.827% of the instances in our data are classified as genuine transactions while only 0.1727% are classified as fraudulent transactions. However, this is predictable since fraudulent transactions are a rare event.

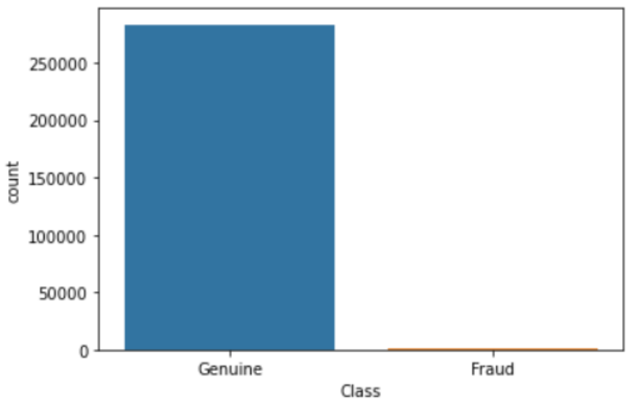


FIGURE 2: BAR PLOT OF VARIABLE CLASS

3.3.1.2 Time Variable

The time feature distribution below indicates that the data has been collected during a 2-day time period. There appear to be two peaks (Bi-modal distribution) which correspond to the high transaction volume that is happening during the daytime while the base of the distribution account for the low transactional volume that is occurring during the nighttime.

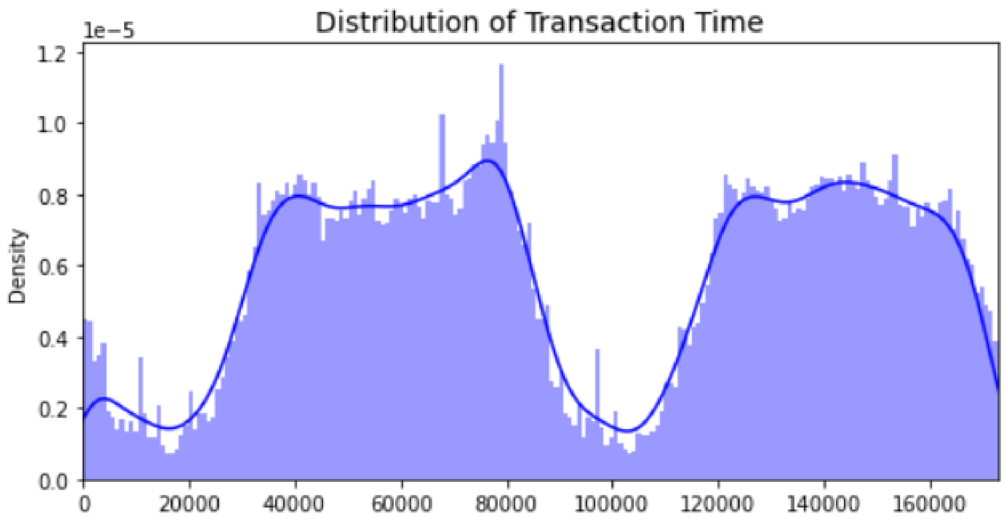
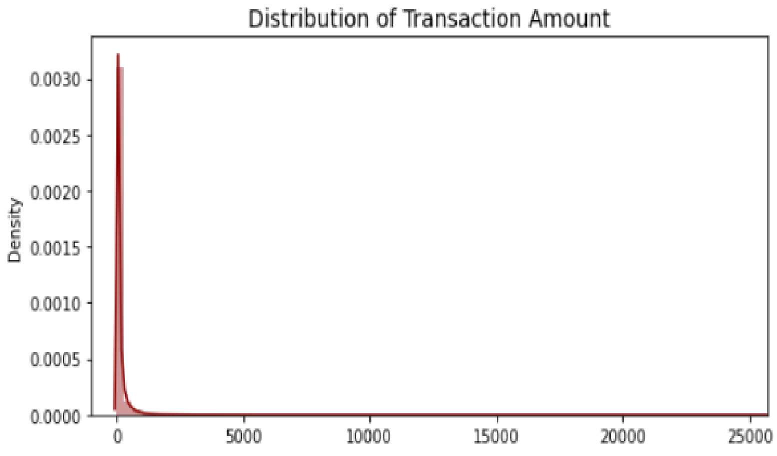
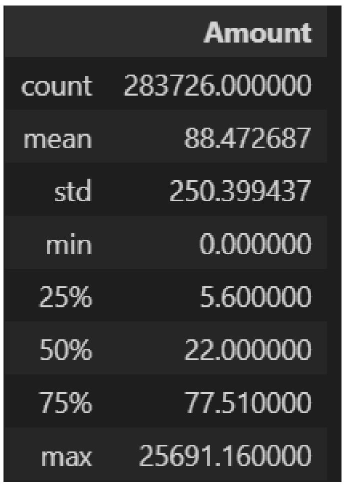


FIGURE 3: DISTRIBUTION OF TIME

3.3.1.3 Amount Variable

The transaction amount distribution below shows that it is asymmetrical and extremely right skewed which is shown by the stretched right tail. The transaction amount ranges between 0 to 25691.16 with a median amount of $22 and around 75% of the transaction amounts range between 0 to $77.5 which explains the extreme skewness in the graph that indicates the presence of outliers. However, it is not recommended to remove outliers since the data points for fraudulent transactions are very low and removing them would mean removing very valuable information and it would make it harder for our model to learn from the fraudulent transactions. Therefore, we will be treating this extreme skewness later through transforming the feature.





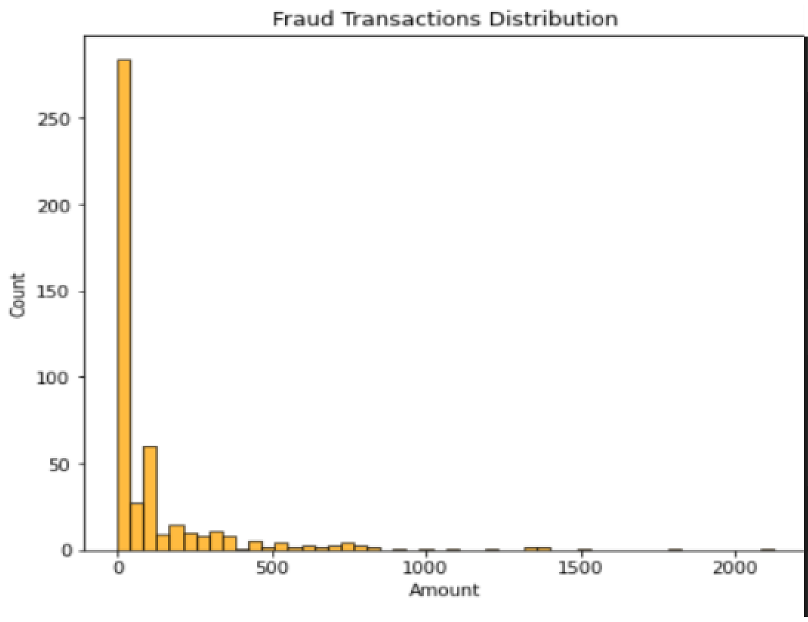
	Amount
count	283726.000000
mean	88.472687
std	250.399437
min	0.000000
25%	5.600000
50%	22.000000
75%	77.510000
max	25691.160000

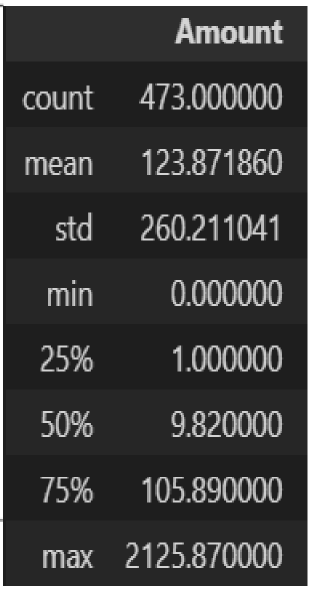
FIGURE 4: HISTOGRAM PLOT AND DESCRIPTIVE STATS ON VARIABLE AMOUNT

We will now try to dig further and see if we can find any further insight by plotting the distribution of variable Time and Amount according to their class type.

3.3.1.4 Variable Amount Distribution for Fraudulent Transactions

The below graph indicates that the amount distribution for fraudulent transactions is right skewed that ranges between 0 and $2125.87 with a mean value of $123.88 and a median amount of $9.82. This indicates that most fraudsters tend to transact money in small amounts. Moreover, the data points at the end of the stretched right tail are an indication of outliers; however, these outliers won't be discarded since there are very few fraudulent data points.



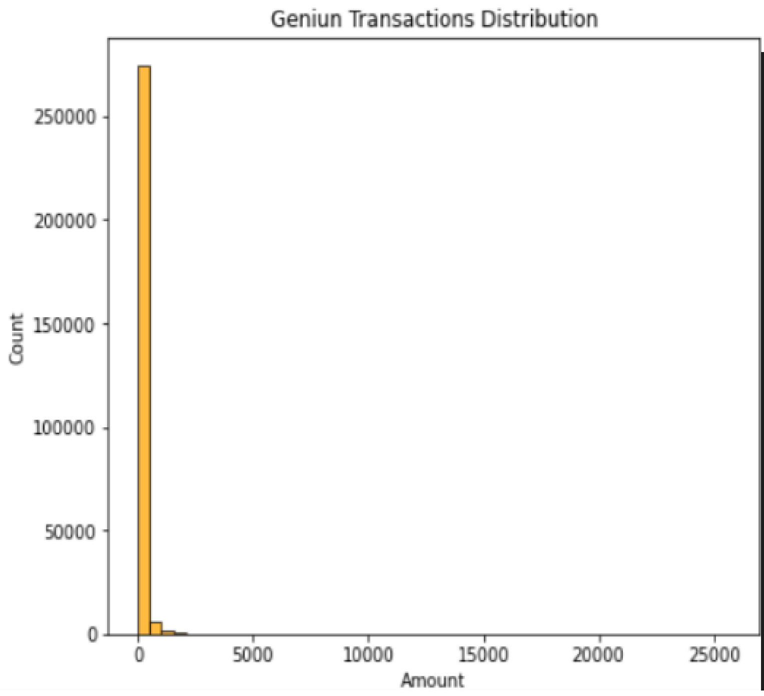


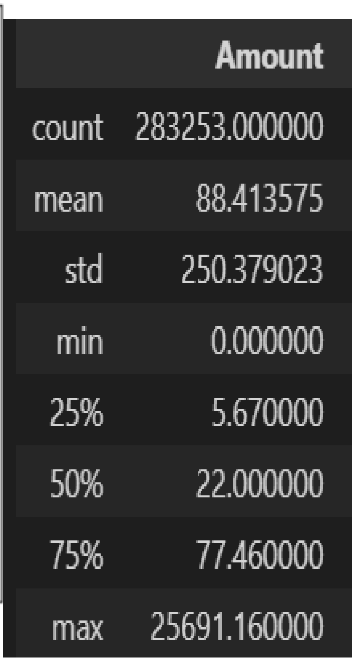
	Amount
count	473.000000
mean	123.871860
std	260.211041
min	0.000000
25%	1.000000
50%	9.820000
75%	105.890000
max	2125.870000

FIGURE 5: HISTOGRAM PLOT AND DESCRIPTIVE STATS ON FRAUDULENT TRANSACTIONS

3.3.1.5 Variable Amount Distribution for Genuine Transactions

The below graph indicates that the amount distribution for genuine transactions is right skewed that ranges between 0 and $25691.16 and has larger mean and median transactions with a mean value of $88.4 and a median amount of $22. Genuine transactions tend to transact in larger amounts than fraudulent ones. This skewness will be transformed later on.





	Amount
count	283253.000000
mean	88.413575
std	250.379023
min	0.000000
25%	5.670000
50%	22.000000
75%	77.460000
max	25691.160000

FIGURE 6: HISTOGRAM PLOT AND DESCRIPTIVE STATS ON GENUINE TRANSACTIONS

3.3.1.6 Time Variable Distribution between Fraudulent & Genuine Transactions

The orange line indicates the genuine transaction activity over time and the blue line indicates the fraudulent transaction activity over time. We can notice that fraudulent activity over time is relatively evenly distributed without any obvious peak in fraudulent activity. Unlike genuine transactions, we can noticeably observe that there is a large activity in genuine transactions during the day and low activity during the night.

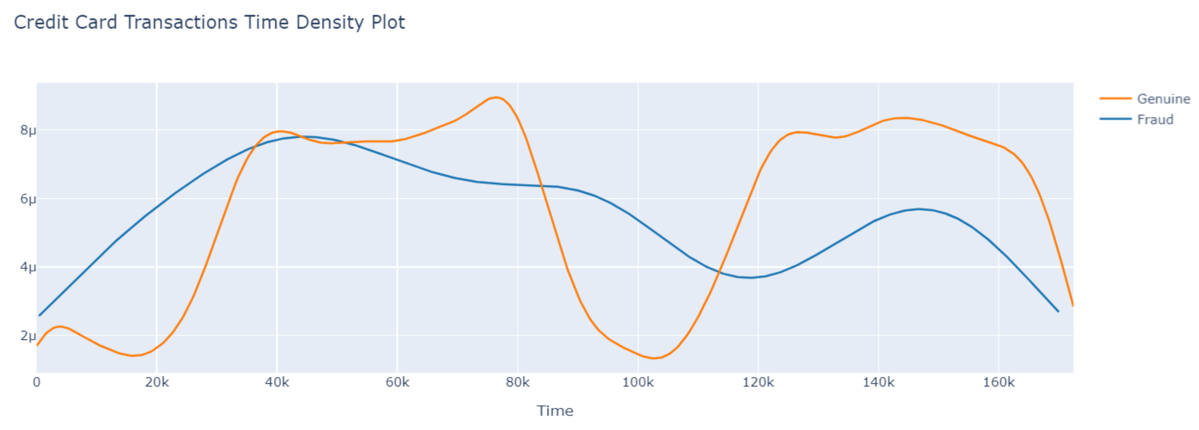


FIGURE 7: TRANSACTIONS DENSITY PLOT

3.3.1.7 Correlation Matrices

A correlation matrix is a table that helps you identify the relationship between the variables in the data set. Before proceeding with bivariant analysis, we would like to see if there are any features that have a strong positive or negative influence on whether a certain transaction is a fraud or not in order to conduct further analysis.

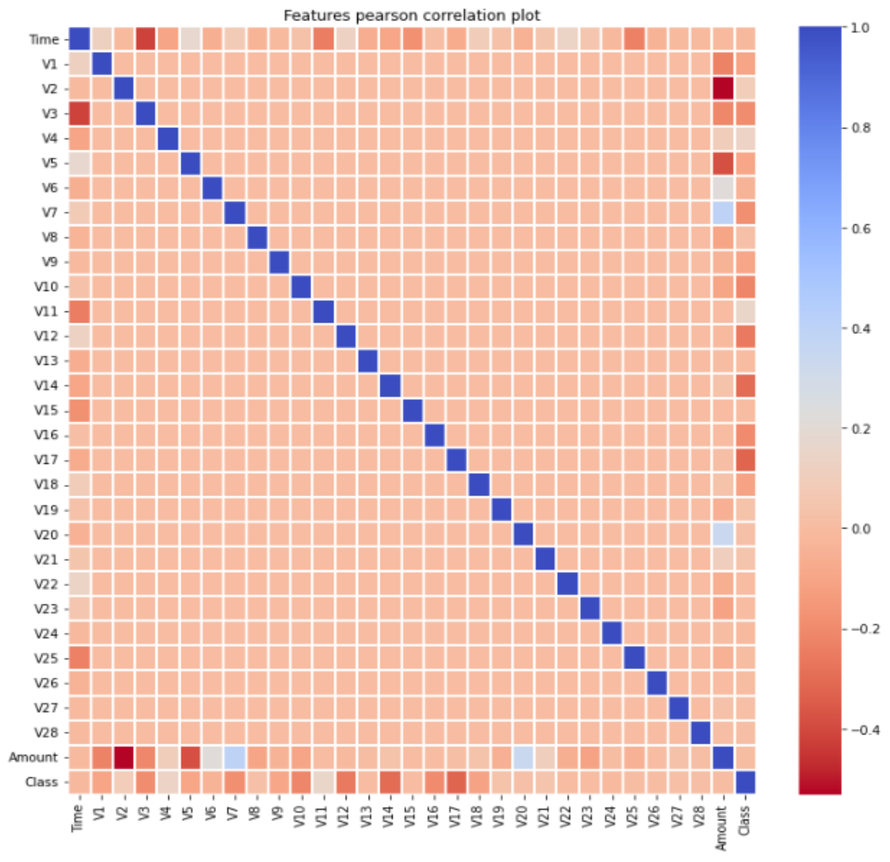


FIGURE 8: CORRELATION MATRIX

As expected, there is no correlation between V1-V28 since they have been transformed by principal components and this method removes multicollinearity in the dataset. We can notice that V17 and V14 have a very strong correlation with the target variable, indicating that these two features could be important in detecting credit fraud, while features like V19 to V28 have no correlation with its target variable, indicating that those feature could be discarded since they don't contribute in detecting fraud. However, we will confirm in discarding those features through applying a more sophisticated method of feature selection to confirm the following.

3.3.2 Bivariate Analysis

After interpreting our correlation matrix, we can now perform further analysis to understand the distribution between the strongly correlated features through visualization.

3.3.2.1 Class-V17 & Class-V14

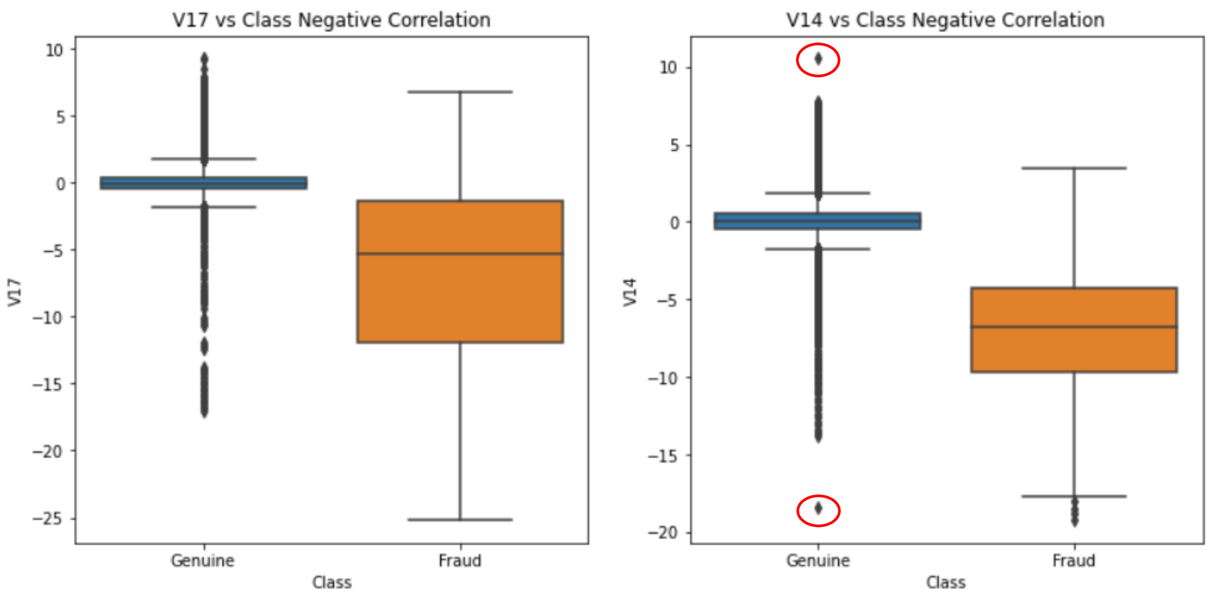


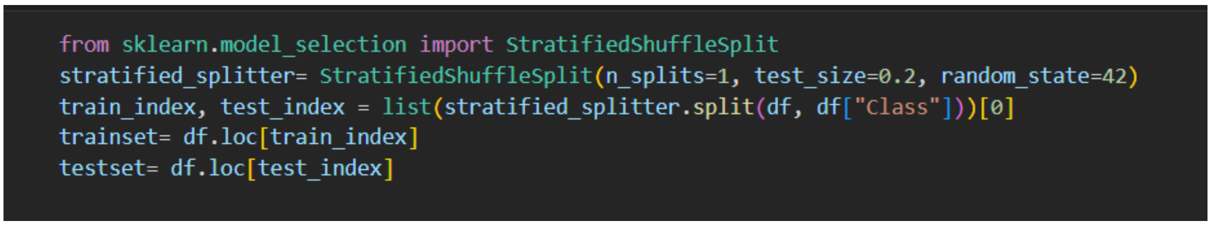
FIGURE 9: DISTRIBUTION OF OUR STRONGLY CORRELATED VARIABLES

From the above box plot, we can notice that both plots have very similar distributions with extreme outliers on genuine class. The two extreme outliers that are presented for the genuine class for V14 will be discarded since they can distort our model's performance.

3.3.3 Splitting the Data

Before we can develop any machine learning models, we must first divide our data set into training and testing sets, with the training set used to train our models and the testing set used to see how our models perform on unseen data. There are different methods of splitting our data, and it is essential to choose the right one in order for our models to perform best. Since we are dealing with a severely unbalanced classification problem, we will be splitting our data using stratified sampling. The stratified sampling method ensures that the distribution or ratio of our target classes is the same in our training and testing sets, this means that the “Class” variable in our training set will contain 99.8273% genuine transactions and 0.1727% fraudulent ones and the same thing goes for the “Class” variable in our testing set.

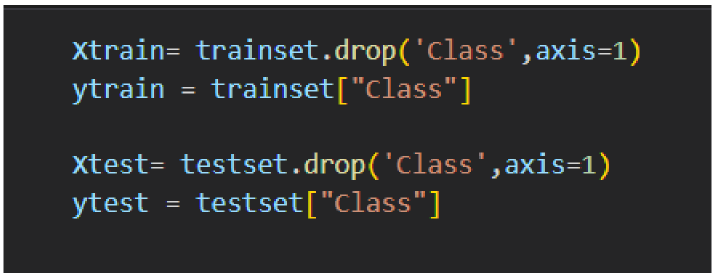
In order to perform this sampling strategy, we will be using **StratifiedShuffleSplit()** function from Scikit-Learn package. It can be seen from the figure below that we have split 20% of our data into a testing set and the remaining 80% into our training set.



```
from sklearn.model_selection import StratifiedShuffleSplit
stratified_splitter= StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_index, test_index = list(stratified_splitter.split(df, df["Class"]))[0]
trainset= df.loc[train_index]
testset= df.loc[test_index]
```

FIGURE 10: STRATIFIED SAMPLING

After we split our data into trainset and testset, we divided our trainset and testset into Xtrain, ytrain, Xtest, and ytest as seen from the figure below. Xtrain and Xtest are our predictors and ytrain and ytest is our target variable. Xtrain and ytrain will be used to train our models, while ytest and Xtest will be used to see how our models perform.



```
Xtrain= trainset.drop('Class',axis=1)
ytrain = trainset["Class"]

Xtest= testset.drop('Class',axis=1)
ytest = testset["Class"]
```

FIGURE 11:SPLITTING THE DATA SET

3.3.4 Feature Transformation

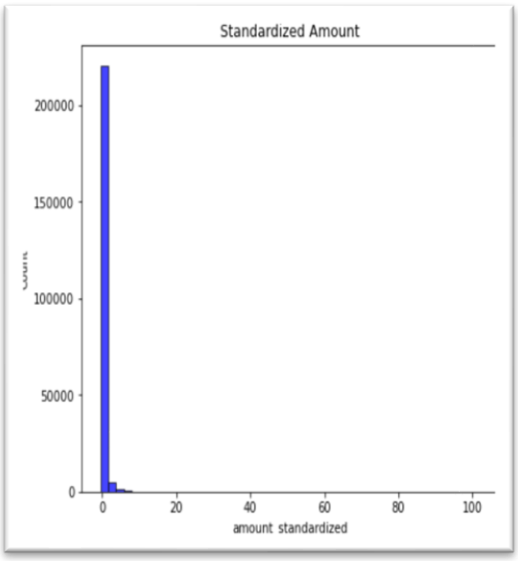
It was essential to split our data before we perform any feature transformation to our data in order to avoid any data leakage (Brownlee, 2020). Feature tranformation is a pre-processing step, it transforms numerical variables and is an important part of building a good machine learning model. Features scaling is a crucial step for classifiers such as support vector machines and K-nearest neighbor since they work by calculating the distance between data points and without scaling features, large values will dominate others features in predictions even though the feature can be less significant than the others (Alshaher, 2021).

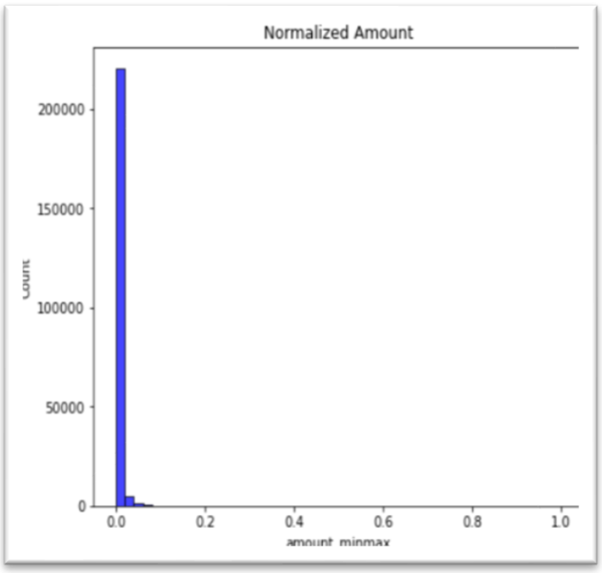
During our EDA process, we saw that the "Amount" variable in our data set is heavily skewed therefore we tried applying different scaling approaches and chose the one that worked best for our data set. The feature scaling methods that we applied are:

Log Transformation: It transforms each data point "x" in a feature to the log(x) and its aim is to change the distribution of a feature from a skewed to a normally distributed one.

Normalization (min-max scaling): It transforms the feature by constraining the data points between [0-1] by subtracting each point with the minimum value of the variable and dividing it by the variable's range.

Standardization: This scaling approach works by transforming the variables where they all follow a standard distribution with a mean value of 0 and a standard deviation of 1.





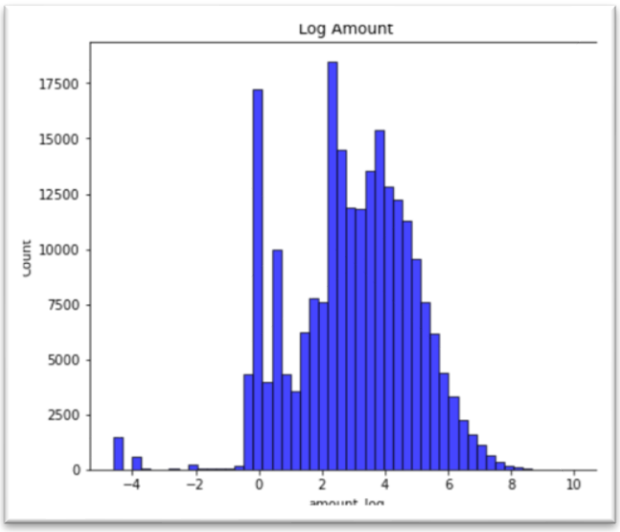


FIGURE 12: OUTPUTS OF AMOUNT AFTER BEING TRANSFORMED

From the above 3 graphs, we can see that the logarithmic transformation was able to treat extreme outliers and its skewness best by normalizing it, therefore we will proceed with the logarithmic amount feature and disregard the other methods of scaling since it has not caused any noticeable change to the distribution of our feature.

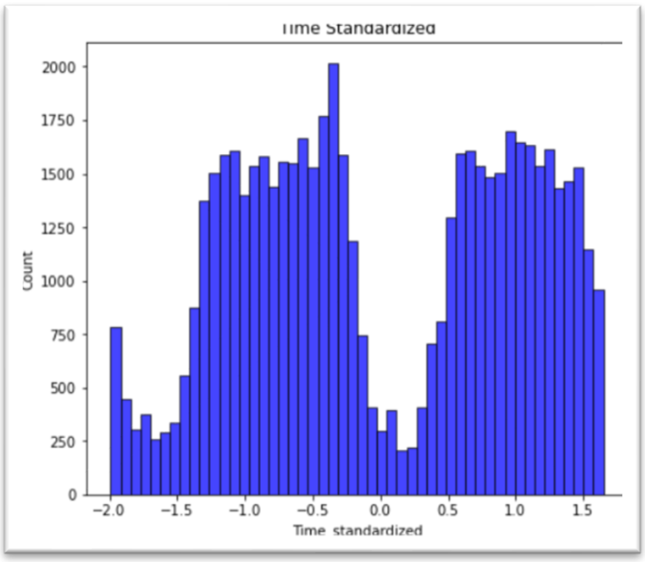


FIGURE 13: VARIABLE TIME AFTER BEING STANDARDIZED

Variable "Time" is not skewed and showed a bimodal distribution however we have normalized the feature using **standard scalar** so that it has a similar scale as the other features, this is because distance-based algorithms performs better on features with a similar scale.

3.3.4 Visualizing All Features in a 2D Space using t-SNE

t-SNE is short for t-distributed stochastic neighbor embedding and it is an unsupervised nonlinear technique. It is a great method for visualizing large numbers of features on a 2-dimensional space and see if it is able to structure the different class points clearly and have a feel of how our data points are distributed (Com and Hinton, 2008). We have shuffled our data set and down sampled the majority class (Genuine) to the minority class and applied the algorithm so that we can get a clear visualization of our data points.

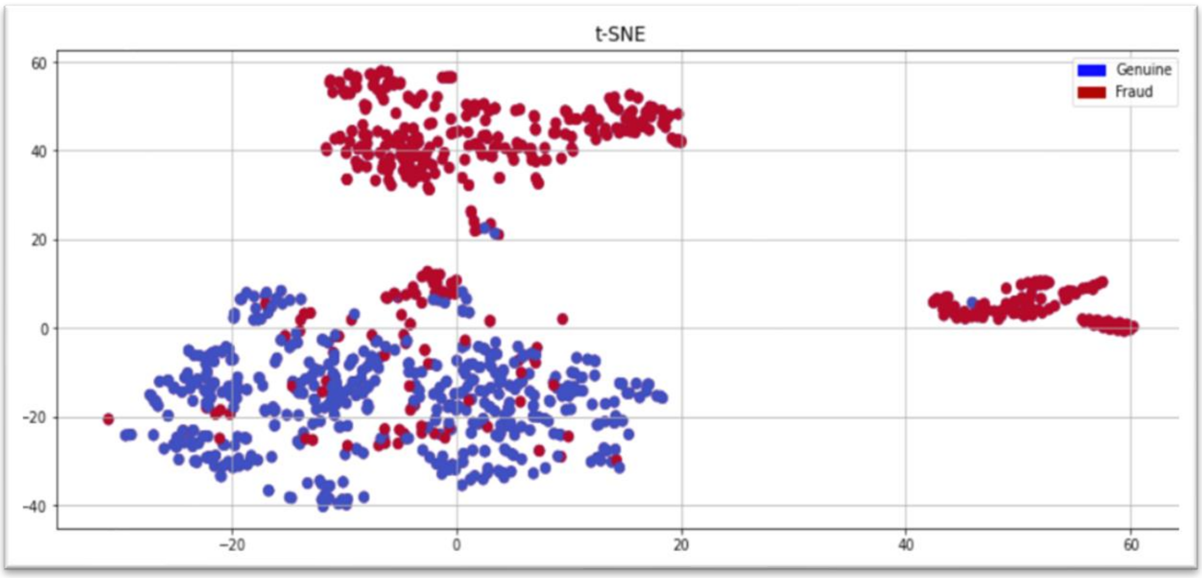


FIGURE 14: T-SNE OUTPUT

We can notice from the above scatter that t-SNE was able to clearly visualize the data points and we can spot 3 clusters where two small clusters are fraudulent points, and one large cluster mostly contains genuine transactions. This is a good indication that the models we are going to apply will be able to distinguish between classes with high precision

3.4 Feature Selection/Importance Interpretation

Before we proceed with building our models it is essential to make sure we are using the best variables that contribute the most to predicting our target outcome and disregard any noise that is presented which will help our models to perform better and execute faster.

Even though most of the features in our data set are transformed using PCA and it can be hard to interpret their importance. However, it is a good insight to understand what predictors are important and see if we can interpret why these features are important and what insights can we also extract from the least important ones.

Most people extract feature importance through ensemble models such as random forests by calculating the Gini impurity. This method calculates how much loss in the impurity standard was obtained as a result of all splits made by the trees based on that attribute. However, tree-based ensemble models tend to inflate the feature importance of continuous variables (Soleymani, 2022).

I will be instead using a wrapper feature selection approach called SHAPLEY VALUE which was invented by Lloyd and is based on a game theory (Ross et al., 2021). It works by taking each predictor individually and calculates how much each feature is contributing to predicting the final target variable (Ross et al., 2021).

The SHAP value was used on a Random Forest model to extract the feature importance where it initially achieved a high F1 score of 85%. The algorithm was runned using the SHAP value library from python resulting in the following importance:

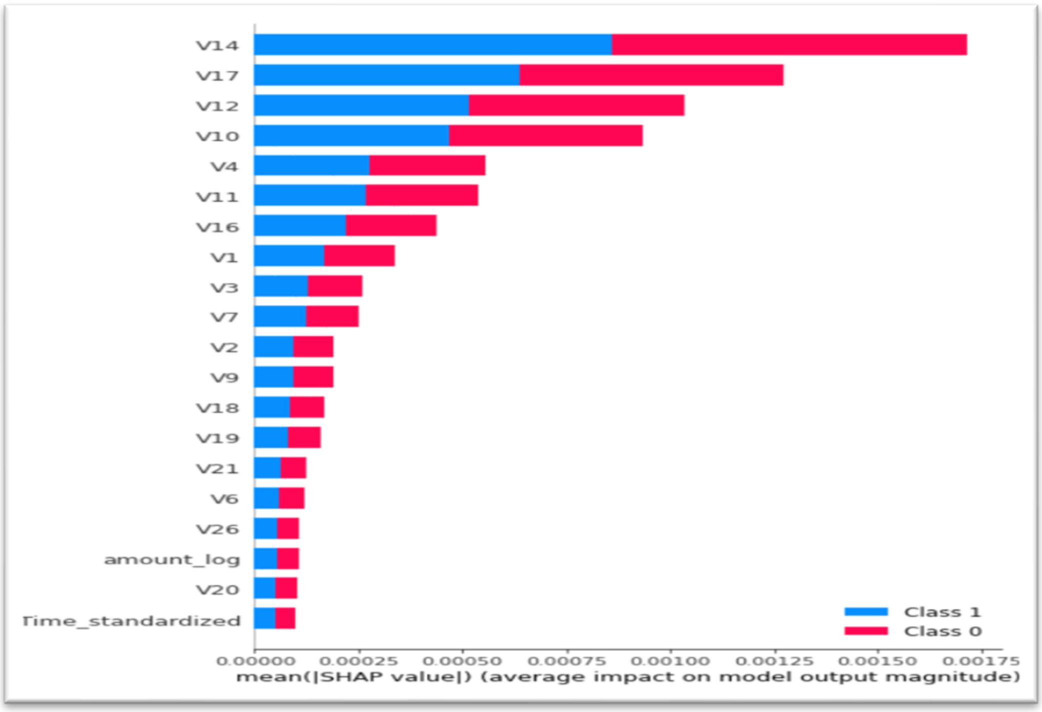
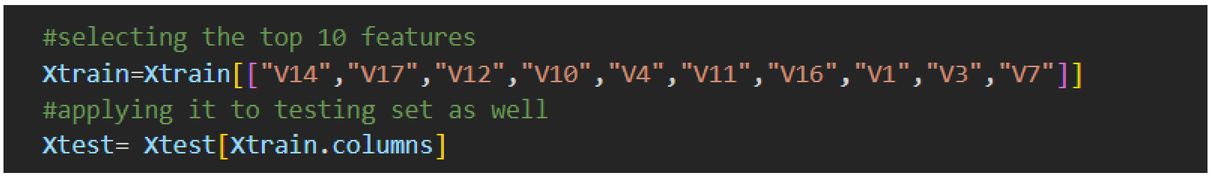


FIGURE 15: FEATURE IMPORTANCE USING SHAP VALUE

The above horizontal stacked bar chart depicts the level of importance each feature contributes to predicting its target variable. We can see from the sorted horizontal bar chart that features "V14", "V17", "V12", and "V10" contributed the most to predicting whether a certain transaction is fraudulent or not while the bottom 6 features contributed the least to predicting its target variable, therefore, they most likely are acting as noise. We can also interpret that variables "Time" and "Amount" has very little impact in helping the model predict its target variable, proving that there is no noticeable behavior between genuine customers and fraudsters when it comes to the amount being transacted and the time the person is transacting. Each bar is subdivided into two classes “0” for genuine and “1” for fraud, we can observe that all features have the same level of importance in predicting both classes.



```
#selecting the top 10 features
Xtrain=Xtrain[["V14","V17","V12","V10","V4","V11","V16","V1","V3","V7"]]
#applying it to testing set as well
Xtest= Xtest[Xtrain.columns]
```

FIGURE 16: FEATURES SELECTED

We chose the top 10 features as seen above and disregarded the rest, since the bottom variables have very little contribution in predicting its target variable and are most likely acting as noise, meaning that they do not have any impact on improving our model’s performance and removing those unnecessary features will also reduce our computational time when building our models

3.5 Modeling Techniques and Prediction Type

In order to choose the right and suitable algorithms for our machine learning models, we need to identify what type of supervised learning problem we are dealing with. In this paper, we are trying to predict a Class variable in which if a certain transaction being made is a fraud or not. The Class variable that we want to predict is presented as a binary classification with 1 being fraud and 0 being a genuine transaction.

Since the output we are trying to predict is a binary classification, this means that the type of problem we are dealing with is a classification problem. There are many algorithms that we can use for classification problems such as Logistic Regression, XGBoost, Random Forest, K-Nearest Neighbors, Naïve Bayes, etc. We will try different algorithms to tackle our problem and with each model used we will experiment with different balancing techniques and see which model performs the best out of all and to which balancing method.

In the following section we will be discussing in detail the types of algorithms we will be using for our models and how these algorithms work in detail:

3.5.1 Logistic Regression

Logistic regression is a classification problem-solving supervised machine learning algorithm. It works by fitting the optimal nonlinear S curve using the sigmoid function to calculate the likelihood of an event occurring or not. The two most commonly used types of logistic regression are binary and multinomial logistic regression. Binary logistic regression is used when the output of a certain problem we are trying to predict is a 1 or 0 which refers to a yes/no or true/false type of answer we are looking for. However, if the number of classes we are trying to predict becomes more than two then this becomes a multinomial logistic regression.

EQUATION 1

$$Sigmoid\ function : S(x) = \frac{1}{1+e^{-x}}$$

$$Probability\ function : \frac{1}{1+e^{-(\beta_0+x\beta_i)}}$$

In the binary logistic regression approach, when the optimal S curve is fitted using the sigmoid function it starts calculating for new inputs based on the created sigmoid function, the probability of whether a certain output belongs to the upper or lower selected threshold which by default it is set to 0.5. If the probability of an output belonging to region 1 is equal to or greater than 0.5 then it will belong to region 1 else, it belongs to region 0.

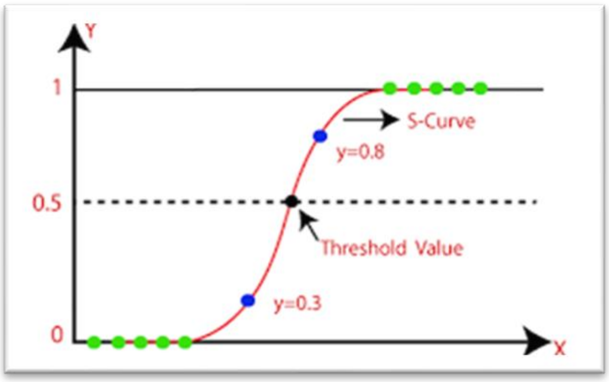


FIGURE 17: PLOT OF SIGMOID FUNCTION

3.5.2 Random Forest

Random forest is a classification and a regression problem-solving supervised machine learning algorithm. It is an ensemble method that uses multiple decision trees that grow together, and it is less sensitive to its training data since it uses a subset of the data set for every decision tree and trains it, thus this results in low variance and low correlation between the features for each model.

Bootstrapping is the process of randomly selecting a portion of the data set, and for each subset, we select a set of features to train each decision tree with. After all decision trees have been trained, our random forest becomes ready to make predictions. Once a new data point enters the model this data point will go through every decision tree that's been created and they all make predictions, the most voted class will turn out the predicted output.

Random forest is an excellent model to use since it can process massive volumes of data with high dimensionality efficiently while preventing overfitting. The model can handle missing data on its own and learn well even when the features in the data set are not scaled out.

3.5.3 K-Nearest Neighbors

K-nearest neighbor is a classification and a regression problem-solving non-parametric supervised machine learning algorithm. It is one of the simplest and fastest algorithms out there and the way it works is, based on the training model it tries to predict the new data point by identifying the points k nearest classified neighbors where k is the number of closest neighbors that the new data point belongs to which is defined by the user himself (Chomboon et al., 2015). The metric that is mostly used to calculate the distance is usually the Euclidean distance where the formula is:

EQUATION 2

$$Euclidean\ distance : d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

For classification problems, the algorithm identifies the k closet neighbors for the new point and assigns it to the most voted class while for regression problems the algorithm predicts the output of the new data point by taking the average values of the k closest neighbors.

Since the algorithm works by calculating the Euclidean distance it is essential to make sure that all features in the dataset are scaled and then normalized since features with a large range can dominate other important features in predictions even though the feature can be less significant than the others and thus scaling the features can significantly improve the accuracy of the model.

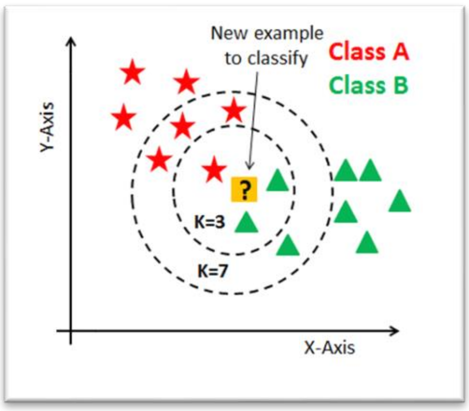


FIGURE 18: APPLICATION OF KNN

3.5.4 XG-Boost

XGboost stands for extreme gradient boosted tree which is an ensemble method, and it is a classification and a regression problem-solving supervised machine learning algorithm. XGBoost is able to prevent the model from overfitting on the training set through regularization techniques known as Lasso and ridge regression and the algorithm is able to find the most suitable way to handle missing values (Priscilla and Prabha, 2020). The algorithm works by introducing different trees on top of each other and from every new tree, it learns from the previous one about which features led to misclassifying wrongly and tries to improve on the new one. With every new tree or model added to the algorithm, it tries to minimize its loss from the previous model until it does not reduce any further.

3.5.5 Support Vector Machine (SVM)

Support vector machine is a linear supervised machine learning model which aims to tackle classification and regression problems. The way the model works is it begins with drawing a hyperplane that easily separates the different classes from each other. After drawing the line that separates the classes from each other, the SVM algorithm then works by finding the data point that is closest to the hyperplane, these points are called the support vector lines, and the distance measured between the support vector line and the hyperplane is called the margin. The objective of the algorithm is to maximize the marginal distance between the support vector line and the hyperplane, and once this is maximized, the hyperplane is considered the optimal line.

In some cases, the classified data points are not easily separable by simply drawing a straight. However, this nonlinearly separable data set can be transformed into a linearly separable problem by adding an additional dimension to its space which allows the dataset to become easily separable by drawing a hyperplane, and then this hyperplane is transformed back to its original dimension through a complex mathematical formula (Fletcher,2008).

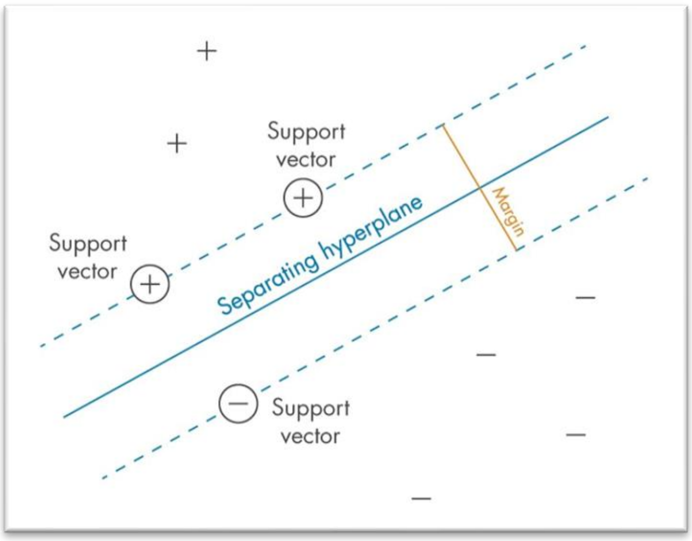


FIGURE 19: SVM PLANE

3.6 Class Balancing Methods

One of the main obstacles we are dealing with is that our target variable in our dataset is extremely imbalanced where 99.8273% of the instances in the Class variable are considered genuine transactions while only 0.1727% are considered fraudulent transactions. When building our models, traditional algorithms tend to learn very well from the majority class while ignoring the minority class since the algorithms are seeking to achieve a high accuracy score (Guo, et. al, 2008). Therefore, several balancing techniques will be applied in order to tackle this imbalanced problem in order to create better-performing models. The balancing methods that will be applied to our models are:

Random Over-Sampling: This balancing method works by randomly making a copy or a replica of the data points in the minority class until it balances well with the majority class. However, some of the drawbacks to using this method is that since we are increasing significantly the number of instances in the minority class this can lead to higher computational time while training our model and may also cause other issues such as overfitting (Mishra et al., 2018).

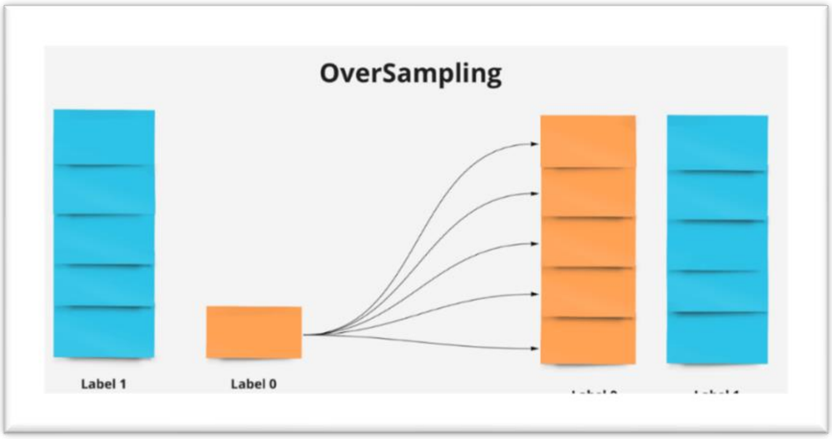


FIGURE 20: OVER SAMPLING

Random Under-Sampling: This balancing method works the opposite way of oversampling, instead of replicating data points, it removes data points from the majority class until it balances well with the minority class and this method significantly reduces the computational time on the training model. However, the drawback of this method is that you are removing a lot of instances from the majority class that could be valuable for our algorithms.

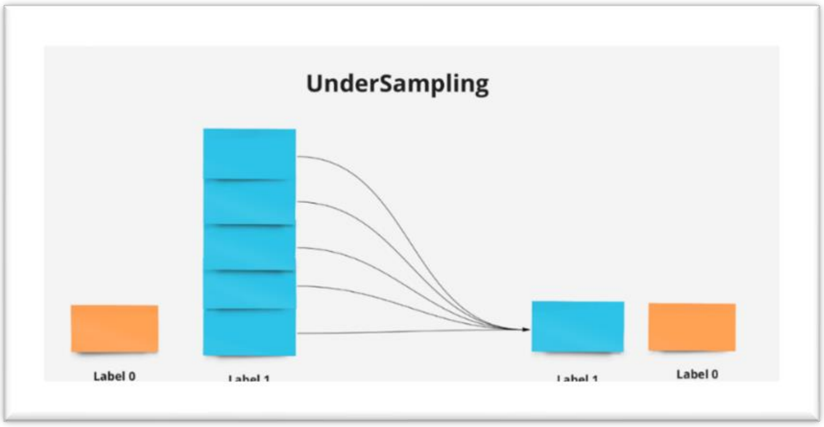


FIGURE 21: UNDER SAMPLING

Synthetic Minority Over-Sampling Technique (SMOTE): The SMOTE method is another type of oversampling and the idea behind this method was to reduce overfitting that occurs with random oversampling. The way it works is by instead of randomly copying data points from the minority class it creates new synthetic data points by performing an interpolation from its neighboring instances in the minority class until it balances that target variable out (Fernandez et al., 2018).

Adaptive Synthetic Samples (ADASYN): The ADADYN method is another type of oversampling technique, and the way it works is instead of randomly replicating data points from the minority class, it generates more synthetic data based on the data points from the minority class that are harder to learn from and less synthetic data from the minority class which are easier to learn from and this is determined through a weighted distribution. This approach aids in learning by reducing bias and shifting the decision boundaries toward the data points that are more difficult to learn from in the minority class (He et al., 2008).

Chapter 4 Model Building

In this chapter, we will begin building our prediction models by applying the algorithms that we mentioned in chapter 3 to every chosen balancing method (Oversampling, Under sampling, SMOTE, and ADASYN) and then evaluate them based on the metrics we chose and see which model alongside its balancing method performs the best.

4.1 Model Approach

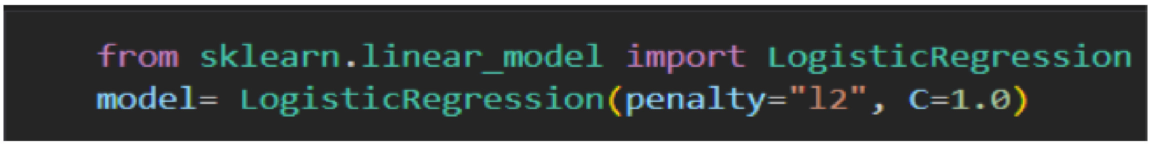
For every algorithm (Logistic Regression, Random Forest, XG-Boost, etc.) and balancing technique (SMOTE, ADSYN, etc.) mentioned earlier we will be following the proposed approach in building our models:

1. Choose an algorithm to build the model.
2. Implement a balancing method to balance our target variable
3. Choose the hyperparameters for our model
4. Use HalvingGridSearchCV to find the optimal parameters for our model
5. Fit our model using our Training set
6. Test our model using our Testing set
7. See how our model performed using specific performance metrics
8. Repeat from step 2 using a different balancing method until all sampling methods used
9. Compare results and see which model using the balancing methods performed the best for our selected algorithm.

4.2 Hyperparameter Optimization

Let us say we want to build a logistic regression model that is able to predict if a credit card transaction is a fraud or not.

We would begin by importing the logistic regression model from sklearn using the default parameters as shown below and fit our model to the training data and then test it on our testing set.



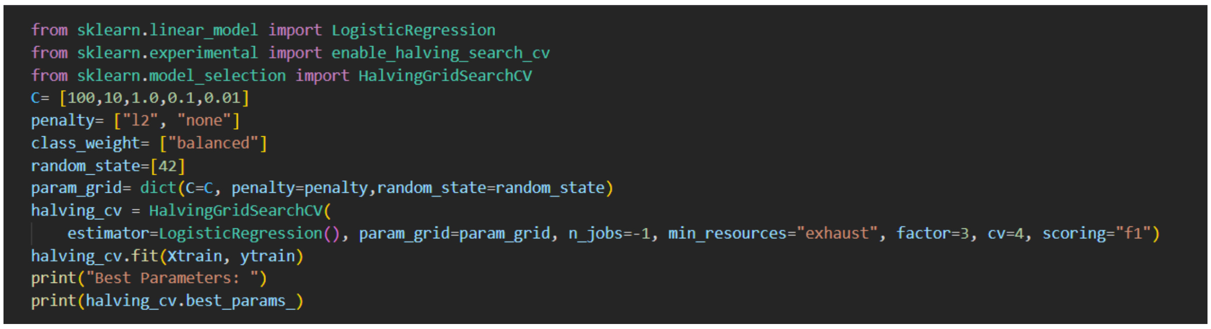
```
from sklearn.linear_model import LogisticRegression
model= LogisticRegression(penalty="l2", C=1.0)
```

FIGURE 22: LOGREG DEFAULT PARAMETERS

However, what if we can achieve better performance from our model by tuning our parameters? Hyper tuning the parameters of our model can in fact improve its performance, however, using functions such as **GridSearchCV** which is an exhaustive method that uses all possible combinations from a given grid of parameters on the whole data set in order to find that optimal parameter for the given model can be very time-consuming and takes a lot of computational power. Nevertheless, in 2020 sklearn came out with a new method/class for tuning our hyperparameters that is a lot faster than GridSearchCV in finding the optimal combination of parameters for our model, and this new method/class is called **HalvingGridSearchCV** (Bex T, 2021).

The way it works is similar to GridSearchCv but instead of training the whole dataset on every possible combination of the parameters that are listed on the parameter grid, it instead starts training all the possible combinations on a fraction of the whole dataset and the best group of parameters that performed well on the small data set are chosen and the rest are disregarded. For the next iteration, the chosen sets of hyperparameters are trained again on a larger dataset than the previous one. This process is repeated until only one set of hyperparameters is left.

To find the optimal parameters for our logistic regression example we will begin by defining the combination of parameters we are going to try out for our model. The parameters chosen for our models are inspired by a book written by Abishek Thakur (2022) on "Approaching (Almost) any Machine Learning Problem"



```
from sklearn.linear_model import LogisticRegression
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
C= [100,10,1.0,0.1,0.01]
penalty= ["l2", "none"]
class_weight= ["balanced"]
random_state=[42]
param_grid= dict(C=C, penalty=penalty,random_state=random_state)
halving_cv = HalvingGridSearchCV(
    estimator=LogisticRegression(), param_grid=param_grid, n_jobs=-1, min_resources="exhaust", factor=3, cv=4, scoring="f1")
halving_cv.fit(Xtrain, ytrain)
print("Best Parameters: ")
print(halving_cv.best_params_)
```

FIGURE 23: INSTALLATION OF HALVING GRID SEARCH

The above code will start training our logistic regression model using 5 different combinations (1x5) and for each parameter set, the model will be evaluated using 4-fold cross-validation, and the parameter set that achieved the best model based on f1 scoring will be used to test our model.

This approach will be used for every algorithm and for every balancing technique used for this algorithm.

4.3 Evaluation Metrics of Choice

It is essential for our model to be able to predict precisely fraudulent transactions since misclassifying a fraudulent transaction as genuine can be very costly for credit companies since they will have to reimburse their customers for the money that has been stolen.

Most people when it comes to dealing with classification problems tend to use accuracy as their metric of choice. However, since we are dealing with an imbalanced problem this metric is not ideal since the model could be predicting really well towards the majority class while performing poorly on the minority class and achieve a really high accuracy score which is misleading since the metric does not differentiate between classes.

Therefore, it is critical to choose the right metrics when it comes to measuring the performance of our model. For this particular problem, a good model for us would be that our model is able to accurately identify fraudulent transactions (1) without compromising much on genuine transactions (0).

Before mentioning the type of metrics that we will be using to measure the performance of our models, we will begin by assuming that fraudulent transactions are our positive class (1) while genuine transactions will be our negative class (0).

The following metrics which we are going to test our models with are:

4.3.1 Confusion Matrix

The confusion matrix is a table that provides us with detailed insight into how our model performed on unseen data. It shows us how many instances were classified correctly for each class label and how many were misclassified for each class label (Susmaga, 2004).

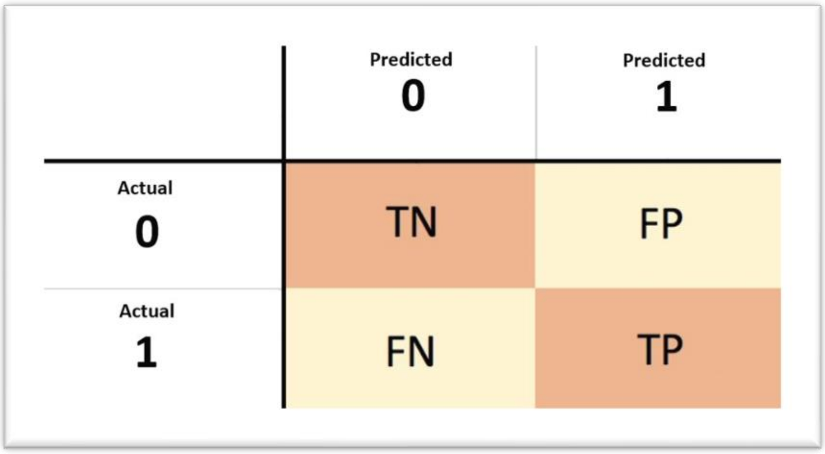


FIGURE 24: CONFUSION MATRIX

True Positive (TP): Given a transaction, if our model predicted that the transaction is fraudulent and the actual value for our target class is fraudulent for the given transaction, then it is considered a True Positive (TP)

False Positive (FP): Given a transaction, if our model predicted that the transaction is fraudulent and the actual value for our target class is **NOT** fraudulent for the given transaction then it is considered a False Positive (FP)

False Negative (FN): Given a transaction, if our model predicted that the transaction is genuine and the actual value for our target class is fraudulent, then it is considered a False Negative (FN).

False Positive (FP): Given a transaction, if our model predicted that the transaction is a fraud and the actual value for our target class is genuine then it is considered a False Positive (FN).

4.3.2 Precision

This metric gives us the likelihood of our model correctly identifying a given transaction as being fraudulent. The higher the ratio the better our model is in correctly identifying fraudulent transactions. It is measured using the following formula:

EQUATION 3

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

4.3.3 Recall

This metric gives us how much our model was able to correctly identify fraudulent transactions. It is measured using the following formula:

EQUATION 4

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

4.3.4 F1 score

If our recall score increased then the precision score tends to decrease, and the opposite is true. F1 score is defined as the (harmonic mean) between the two metrics and we will be focusing mostly on this metric to compare our models. It is measured using the following formula:

EQUATION 5

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

4.4 Naïve/Baseline Model

Before we jump into building any models, it is essential to have a baseline or a reference that can guide us into understanding how our models are really performing and this reference we get is from building a naïve/baseline model. A naïve model is the least sophisticated predictive model we can build that lacks complexity and performance which is why it is used as a guide or a benchmark to other more complex models.

We are dealing with a classification problem therefore the way the naïve model works for such a problem is it takes the most frequent class (mode) from our target variable and uses it as its predictive outcome.

The majority class in our target variable is **genuine(0)** containing 227451 transactions while **fraudulent (1)** class contains only 394 transactions resulting in a total of **227854** transactions; hence class genuine(0) will be our predictive output for all the training set.

We will begin building our naïve model and assess its performance based on recall, precision, and F1 scores and use them as our benchmark/baseline:

For “Genuine (0)” class the 3 performance measures will be:

- Precision: 227451/227854 = 0.998
- Recall: 227451/227451= 1
- F-score: 2/(1/0.998 + 1/1.0) = 0.998

For “Fraud (1)” class the 3 performance measures will be:

- Precision: 0/0= 0
- Recall: 0/394= 0
- F-score: 0

The baseline score will be the average of the two classes resulting in:

- **Precision: 0.499**
- **Recall: 0.5**
- **F-score: 0.499**

During our model building phase, any model that achieves a score higher than our baseline score will be considered an improvement.

Chapter 5 Results and Discussions

In this chapter, we will be presenting and interpreting the results achieved from our models and discuss how we retrieved them.

We care about our model to be able to predict fraudulent transactions accurately but not at the expense of falsely predicting genuine transactions. Therefore, we will be looking at the F1-score (harmonic mean of precision and recall) to compare different balancing methods and models.

5.1 Logistic Regression

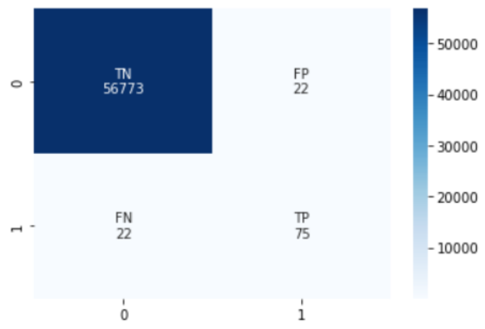
5.1.1 **Balancing Method**: Without Sampling

We began by building our logistic regression model without applying any sampling method to it. During the hyperparameter grid optimization we tried values of C ranging from 0.01 to 100 and penalty terms of "l2" and "none". After running the Halving Grid Search method our optimal parameters retrieved were 'C': 0.01 and 'penalty': 'none'. Those optimal parameters were used to train our logistic regression model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 2: RESULTS OF LOGREG WITHOUT SAMPLING

Metric	**Output**
Recall Score	**0.773**
F1-Score	**0.773**
Precision Score	**0.773**

FIGURE 25: CONFUSION MATRIX OF LOGREG WITHOUT SAMPLING



By looking at the precision score we can conclude that the likelihood of our model correctly identifying a fraudulent transaction is 77.3% which is reflected by looking at the recall score. This is quite impressive since we have not applied any balancing technique to our model, and it has still performed well to a certain extent. Looking at the confusion matrix we can notice that FP=FN, this explains why our recall score has the same score as precision resulting in an F1 score of 77.3%.

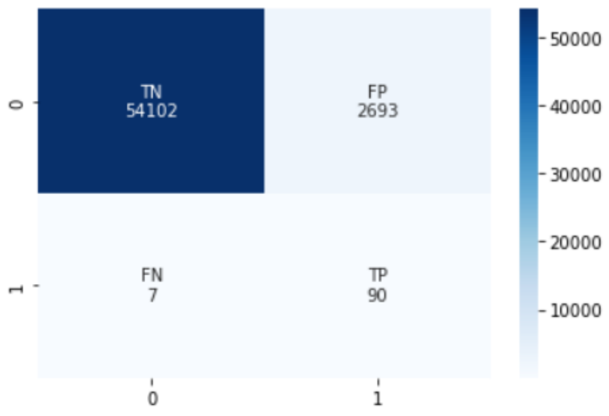
5.1.2 **Balancing Method**: Random Under-Sampling

Here we randomly under sampled our target variable of our training set where we reduced the instances of genuine transactions to match the fraudulent one and started building our logistic model. During the hyperparameter grid optimization we tried values of C ranging from 0.01 to 100 and penalty terms of “l2” and “none”. After running the Halving Grid Search method our optimal parameters retrieved were **'C': 1.0 and 'penalty': “l2”.** Those optimal parameters were used to train our logistic regression model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were calculated from our confusion matrix.

TABLE 3: RESULTS OF LOGREG USING UNDER-SAMPLING

Metric	**Output**
Recall Score	**0.927**
Precision Score	**0.0323**
F1-Score	**0.062**

FIGURE 26: CONFUSION MATRIX OF LOGREG WITH UNDER-SAMPLING



By using under-sampling our model became better at identifying fraudulent transactions, achieving a higher recall score of 92.7%. However, this came at the expense of our model having more false positives resulting in a much lower precision rate which is something we want to avoid because it can result in a negative experience for the customers since they can't proceed with their payments.

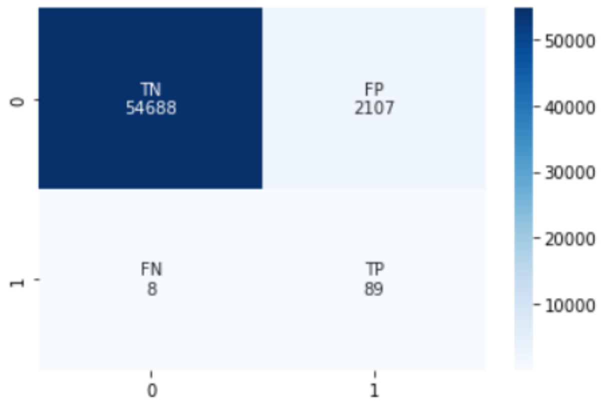
5.1.3 **Balancing Method**: Random Over-Sampling

Here we randomly over sampled our target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our logistic model. During the hyperparameter grid optimization we tried values of C ranging from 0.01 to 100 and penalty terms of "l2" and "none". After running the Halving Grid Search method our optimal parameters retrieved were **'C': 0.1 and 'penalty': "none".** Those optimal parameters were used to train our logistic regression model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were calculated from our confusion matrix.

TABLE 4: RESULTS OF LOGREG USING RANDOM OVERSAMPLING

Metric	**Output**
Recall Score	**0.917**
Precision Score	**0.04**
F1-Score	**0.077**

FIGURE 27: CONFUSION MATRIX OF LOGREG USING RANDOM OVERSAMPLING



By using the random oversampling technique our model was able to improve slightly compared to under-sampling by resulting in fewer false positives while maintaining the same number of true positives which resulted in a slightly higher precision score of 4%.

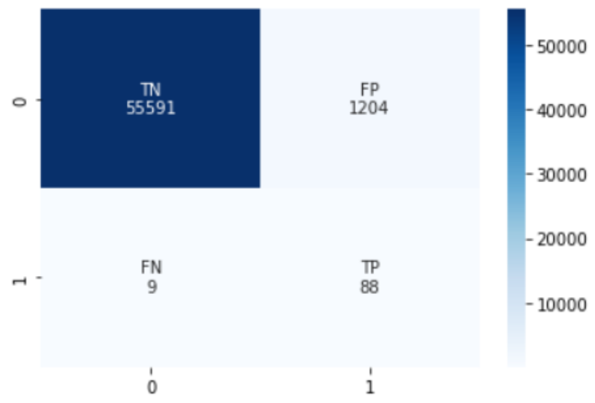
5.1.4 **Balancing Method**: SMOTE

Here we used the SMOTE technique to over-sample the target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our logistic mode. During the hyperparameter grid optimization we also tried values of C ranging from 0.01 to 100 and penalty terms of "l2" and "none". After running the Halving Grid Search method our optimal parameters retrieved were **'C': 10 and 'penalty': "l2".** Those optimal parameters were used to train our logistic regression model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were calculated from our confusion matrix.

TABLE 5: RESULTS OF LOGREG USING SMOTE

Metric	Output
Recall Score	**0.907**
Precision Score	**0.068**
F1-Score	**0.127**

FIGURE 28: CONFUSION MATRIX OF LOGREG USING SMOTE



By using the SMOTE technique, our model was able to reduce the number of false positives significantly compared to other balancing methods while reducing slightly the number of true positives resulting in a higher precision score of 6.8% and a slightly lower recall score of 90.7%.

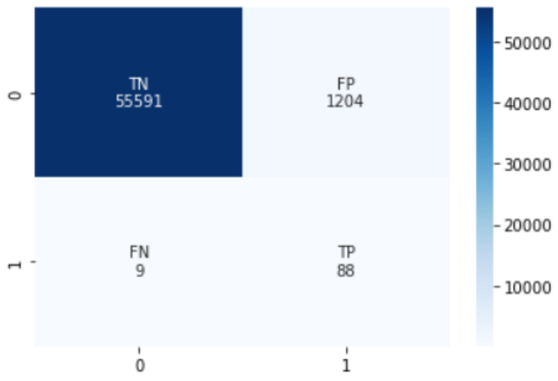
5.1.5 **Balancing Method**: ADASYN

Here we used the ASASYN technique to over-sample the target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our logistic mode. During the hyperparameter grid optimization we also tried values of C ranging from 0.01 to 100 and penalty terms of “l2” and “none”. After running the Halving Grid Search method our optimal parameters retrieved were **'C': 0.1 and 'penalty': “none”**. Those optimal parameters were used to train our logistic regression model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were calculated from our confusion matrix.

TABLE 6: RESULTS OF LOGREG USING ADASYN

Metric	Output
Recall Score	**0.068**
Precision Score	**0.907**
F1-Score	**0.127**

FIGURE 29: CONFUSION MATRIX OF LOGREG USING ADASYN



Using the ADASYN technique, our model produced the same results as the SMOTE technique. No notable change occurred.

5.1.6 **Summary of Logistic regression using different balancing methods:**

We can notice from the table below that logistic regression performed best **without** applying any sampling method to it, achieving an F1 score of 77.3%, which is a **54.9% improvement** from its baseline model. Under sampling method performed the worst, resulting in an F1 score of only 6.2%

TABLE 7: SUMMARY OF RESULTS FOR LOGISTIC REGRESSION

Balancing Method	F1-Score	Recall-Score	Precision-Score
Without Sampling	0.773	0.773	0.773
Under-Sampling	0.062	0. 927	0.0323
Random Over-Sampling	0.076	0. 917	0.04
SMOTE	0.127	0. 907	0.068
ADASYN	0.127	0.068	0.907

5.2 Random Forest

5.2.1 **Balancing Method:** Without Sampling

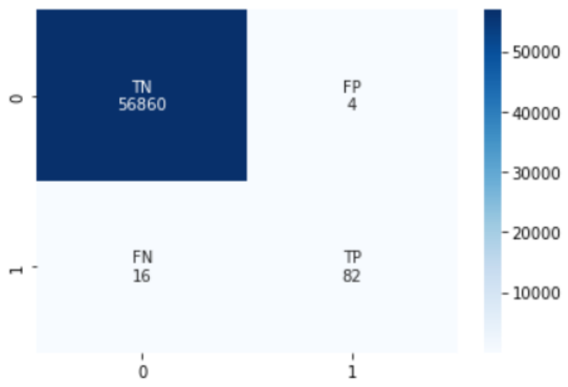
We began by building our random forest model without applying any sampling method to it. During the hyperparameter grid optimization we tried values for number of trees in the forest (n_estimator) ranging from 300 to 800 with maximum depth (max_depth) of the tree ranging from 8 to 25 followed by the minimum samples required to split (min_samples_split) and minimum samples leaf ranging from 2 to 5 and the max number of features are “log2” and “sqrt”.

After running the Halving Grid Search method our optimal parameters retrieved were **'max_depth'= 15, 'max_features'= 'sqrt', 'min_samples_leaf'= 2, 'min_samples_split'= 2, and 'n_estimators'= 500**. Those optimal parameters were used to train our random forest model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 8: RESULTS OF RANDOM FOREST WITHOUT SAMPLING

Metric	Output
Recall Score	**0.837**
Precision Score	**0.953**
F1-Score	**0.891**

FIGURE 30: CONFUSION MATRIX OF RANDOM FOREST WITHOUT SAMPLING



By looking at the confusion our model predicted fraudulent transactions very well, achieving a recall score of 83.7% while maintaining very low false positives that resulted in a very high precision score of 95.3%.

5.2.2 **Balancing Method**: Under-Sampling

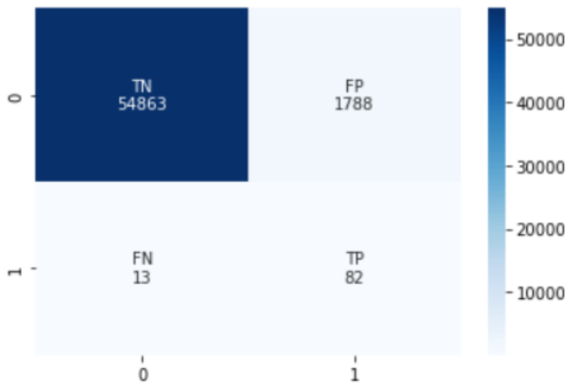
Here we randomly under sampled our target variable of our training set where we reduced the instances of genuine transactions to match the fraudulent one and started building our random forest model. During hyperparameter grid optimization, same parameters were given as section 5.2.1.

After running the Halving Grid Search method our optimal parameters retrieved were **'max_depth'=None,'max_features'='sqrt', 'min_samples_leaf'=2, 'min_samples_split'= 2, and 'n_estimators'= 500.** Those optimal parameters were used to train our random forest model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 9: RESULTS OF RANDOM FOREST USING UNDERSAMPLING

Metric	**Output**
Recall Score	**0.863**
Precision Score	**0.044**
F1-Score	**0.083**

FIGURE 31:CONFUSION MATRIX OF RANDOM FOREST USING UNDERSAMPLING



Using under-sampling has worsened our random forest model by increasing the number of false positives significantly which resulted in achieving a precision score of only 4.4% and a low overall F1 score of only 8.3%. Even though the model increased the number of true positives, this trade-off is not feasible as it can leave a lot of customers unsatisfied with getting their credit cards blocked which can result in them turning to other competitors.

5.2.3 **Balancing Method**: Random Over-Sampling

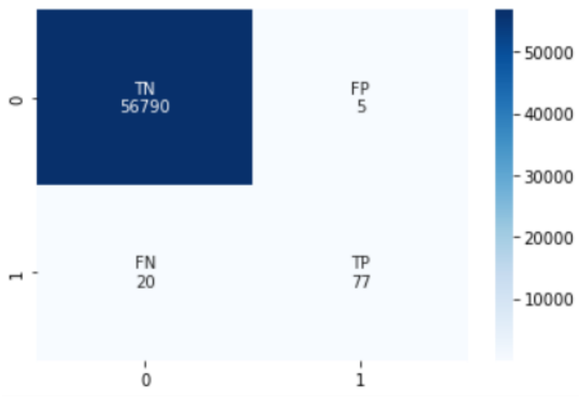
Here we randomly over sampled our target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our random forest model. During hyperparameter grid optimization, same parameters were given as section 5.2.1.

After running the Halving Grid Search method our optimal parameters retrieved were **'max_depth'=None,'max_features'='log2','min_samples_leaf'=2, 'min_samples_split'= 2, and 'n_estimators'= 500.** Those optimal parameters were used to train our random forest model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 10: RESULTS OF RANDOM FOREST USING RANDOM OVERSAMPLING

Metric	Output
Recall Score	**0.794**
Precision Score	**0.939**
F1-Score	**0.86**

FIGURE 32: CONFUSION MATRIX OF RANDOM FOREST USING RANDOM OVERSAMPLING



The model improved significantly by using oversampling compared to under-sampling resulting in a high precision score of 93.9% and a high recall score of 79.4% showing that the model is able to predict fraudulent transactions with high accuracy while maintaining low false positives.

5.2.4 **Balancing Method**: SMOTE

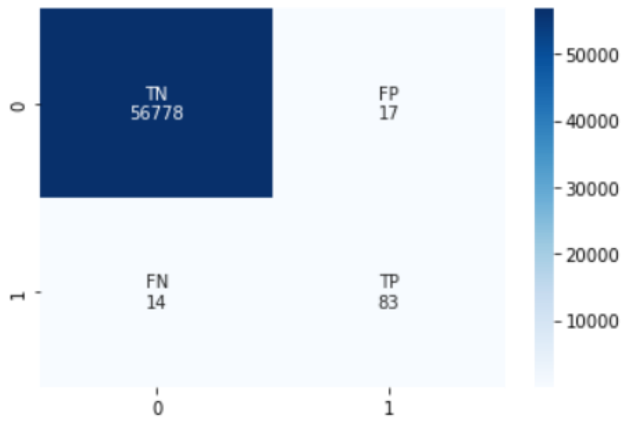
Here we used the SMOTE technique to over-sample the target variable of our training set, where we increased the instances of our fraudulent transactions to match the genuine ones and started building our random forest model. During hyperparameter grid optimization, same parameters were given as section 5.2.1.

After running the Halving Grid Search method our optimal parameters retrieved were **'max_depth'=None,'max_features'='log2','min_samples_leaf'=2, 'min_samples_split'= 2, and 'n_estimators'= 300**. Those optimal parameters were used to train our random forest model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 11: RESULTS OF RANDOM FOREST USING SMOTE

Metric	**Output**
Recall Score	**0.856**
Precision Score	**0.83**
F1-Score	**0.84**

FIGURE 33: CONFUSION MATRIX OF RANDOM FOREST USING SMOTE



By using the SMOTE technique our random forest model was able to predict correctly more fraudulent transactions compared to random oversampling. However, it increased its false positives resulting in an overall slight decrease in its F1 score to 84%.

5.2.5 **Balancing Method**: ADASYN

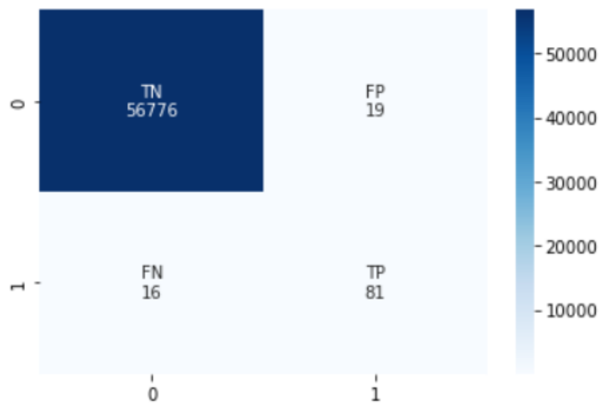
Here we used the ASASYN technique to over-sample the target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our random forest mode. During the hyperparameter grid optimization, we tried different values for number of trees in the forest (n_estimator) ranging from 100 to 800 with maximum depth (max_depth) of the tree ranging from 15 to 30 followed by the minimum samples required to split (min_samples_split) and minimum samples leaf ranging from 2 to 5 and the max number of features are “log2” and “sqrt”.

After running the Halving Grid Search method our optimal parameters retrieved were **'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100.** Those optimal parameters were used to train our random forest model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 12: RESULTS OF RANDOM FOREST USING ADASYN

Metric	Output
Recall Score	**0.835**
Precision Score	**0.81**
F1-Score	**0.822**

FIGURE 34: CONFUSION MATRIX OF RANDOM FOREST USING ADASYN



Very similar results compared to SMOTE achieving an F-score of 82.32% and a precision score of 81% showing that our model is able to predict fraudulent transactions fairly well.

5.2.6 **Summary of Random Forest using different balancing methods:**

We can notice from the table below that Random Forest performed best **without** applying any sampling method to it, achieving an F1 score of 77.3% which is a **74.3% improvement** from the baseline model. Under-sampling performed the worst achieving an F1 score of only 8.3%.

TABLE 13: SUMMARY OF RESULTS FOR RANDOM FOREST

Balancing Method	F1-Score	Recall-Score	Precision-Score
Without Sampling	0.891	0.837	0.953
Under-Sampling	0.083	0. 863	0.044
Random Over-Sampling	0.86	0. 794	0.939
SMOTE	0.84	0. 856	0.83
ADASYN	0.822	0.835	0.81

5.3 XG-Boost Classifier

5.3.1 **Balancing Method**: Without Sampling

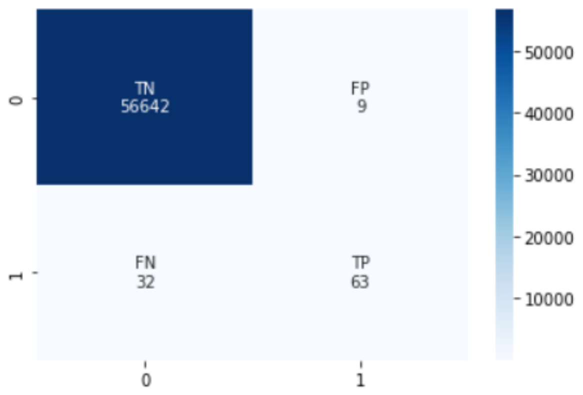
We began by building our xg-boost classifier model without applying any sampling method to it. During the hyperparameter grid optimization we have given the loss function (loss) two inputs being “deviance” and “exponential”. The values for learning rate ranged from 0.1 to 1 and for number of boosting trees in the forest (n_estimator) values ranged from 50 to 400 and the fraction of samples to be used for base learners (subsample) values ranged from 0.6 to 1 and the minimum number of samples needed to begin splitting the tree (min_samples_split) ranged from 2 to 5 and finally the values for the minimum number of samples needed to form a leaf node (min_samples_leaf) ranged from 1 to 10.

After running the Halving Grid Search method our optimal parameters retrieved were **'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300, 'subsample': 1.0**. Those optimal parameters were used to train our random forest model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 14: RESULTS FOR XGB WITHOUT SAMPLING

Metric	**Output**
Recall Score	**0.66**
Precision Score	**0.875**
F1-Score	**0.75**

FIGURE 35: CONFUSION MATRIX FOR XGB WITHOUT SAMPLING



By looking at the confusion matrix the model was able to achieve a low false positive resulting in a high precision score of 87.5% however it had a lot of false negatives resulting in a low recall score of 66%.

5.3.2 **Balancing Method**: Under-Sampling

Here we randomly under sampled our target variable of our training set where we reduced the instances of genuine transactions to match the fraudulent one and started building our xg-boost model. During hyperparameter grid optimization we used the same parameter and values in section 5.3.1.

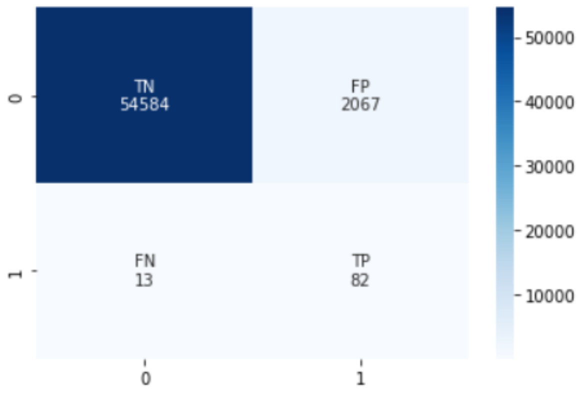
After running the Halving Grid Search method our optimal parameters retrieved were **'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150, 'subsample': 0.7**.

Those optimal parameters were used to train our xg-boost model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 15: RESULTS FOR XGB USING UNDERSAMPLING

Metric	Output
Recall Score	**0.863**
Precision Score	**0.038**
F1-Score	**0.073**

FIGURE 36: CONFUSION MATRIX FOR XGB USING UNDERSAMPLING



Like all other models using an undersampling technique, it has achieved a very low precision score of only 3.8% resulting in very high false positives.

5.3.3 **Balancing Method**: Random Over-Sampling

Here we randomly over sampled our target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our xg-boost model. We used different values for hyperparameter grid optimization, we have given the loss function (loss) two inputs being “deviance” and “exponential”. The values for learning rate ranged from 0.1 to 1 and for number of boosting trees in the forest (n_estimator) values ranged from 300 to 800 and the fraction of samples to be used for base learners (subsample) values ranged from 0.6 to 1 and the minimum number of samples needed to begin splitting the tree (min_samples_split) ranged from 2 to 100 and finally the values for the minimum number of samples needed to form a leaf node (min_samples_leaf) ranged from 1 to 10.

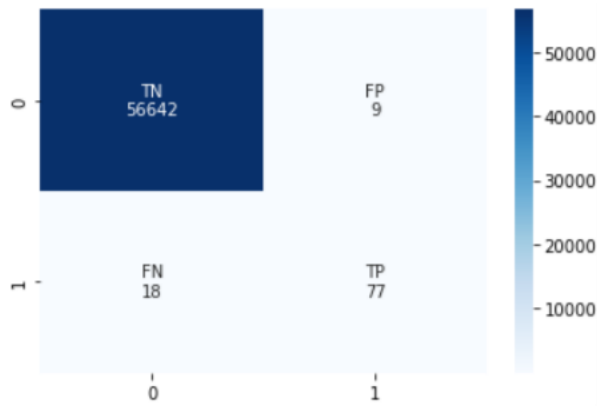
After running the Halving Grid Search method our optimal parameters retrieved were **'min_samples_leaf': 20, 'min_samples_split': 2, 'n_estimators': 800, 'subsample': 0.7.**

Those optimal parameters were used to train our xg-boost model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 16: RESULTS FOR XGB USING RANDOM OVERSAMPLING

Metric	Output
Recall Score	**0.81**
Precision Score	**0.895**
F1-Score	**0.85**

FIGURE 37: CONFUSION MATRIX FOR XGB USING RANDOM OVERSAMPLING



By using oversampling, our xg-boost model has improved significantly achieving low false positives and low false negatives resulting with an F1 score of 85%.

5.3.4 **Balancing Method**: SMOTE

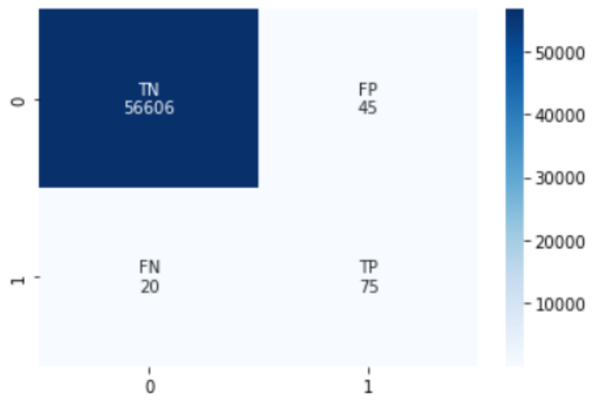
Here we used the SMOTE technique to over-sample the target variable of our training set, where we increased the instances of our fraudulent transactions to match the genuine ones and started building our xg-boost model. During hyperparameter grid optimization we used the same parameter and values in section 5.3.3.

After running the Halving Grid Search method our optimal parameters retrieved were **'min_samples_leaf': 20, 'min_samples_split': 2, 'n_estimators': 800, 'subsample': 0.7**. Those optimal parameters were used to train our xg-boost model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 17: RESULTS FOR XGB USING SMOTE

Metric	Output
Recall Score	**0.789**
Precision Score	**0.625**
F1-Score	**0.696**

FIGURE 38: CONFUSION MATRIX FOR XGB USING SMOTE



The model has worsened by using SMOTE through getting higher false positives and negatives resulting in an F1 score of only 69.6%

5.3.5 **Balancing Method**: ADASYN

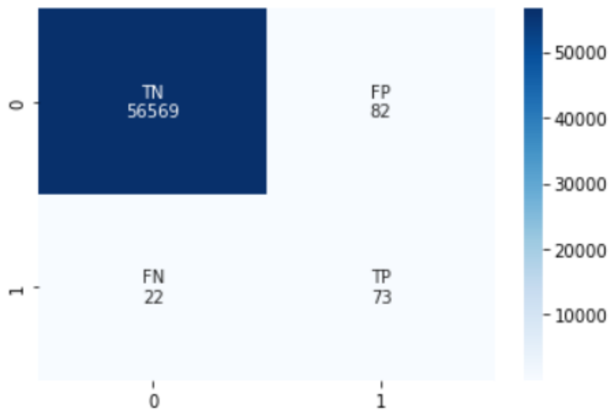
Here we used the ASASYN technique to over-sample the target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our xg-boost model. During hyperparameter grid optimization we used the same parameter and values in section 5.3.3 except for the number of boosting trees in the forest (n_estimator) where values ranged from 400 to 900.

After running the Halving Grid Search method our optimal parameters retrieved were **'min_samples_leaf': 25, 'min_samples_split': 2, 'n_estimators': 900, 'subsample': 0.7**. Those optimal parameters were used to train our xg-boost model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 18: RESULTS FOR XGB USING ADASYN

Metric	Output
Recall Score	**0.768**
Precision Score	**0.471**
F1-Score	**0.584**

FIGURE 39: CONFUSION MATRIX FOR XGB USING ADASYN



ADASYN performed the worst compared to the other oversampling methods for xg-boost achieving an F1 score of only 58.4%.

5.3.6 **Summary of XG-Boost using different balancing methods**:

We can notice from the table below that XG-Boost performed best using **random oversampling** to it, achieving an F1 score of 85% which is around a **70% improvement** from its baseline model. Under sampling performed the worst, resulting in an F1 score of only 3.8%.

TABLE 19: SUMMARY OF RESULTS FOR XG-BOOST

Balancing Method	F1-Score	Recall-Score	Precision-Score
Without Sampling	0.75	0.66	0.875
Under-Sampling	0.073	0. 863	0.038
Random Over-Sampling	0.85	0. 81	0.895
SMOTE	0.696	0. 789	0.625
ADASYN	0.584	0.768	0.471

5.4 Support Vector Machine (SVM)

5.4.1 **Balancing Method**: Without Sampling

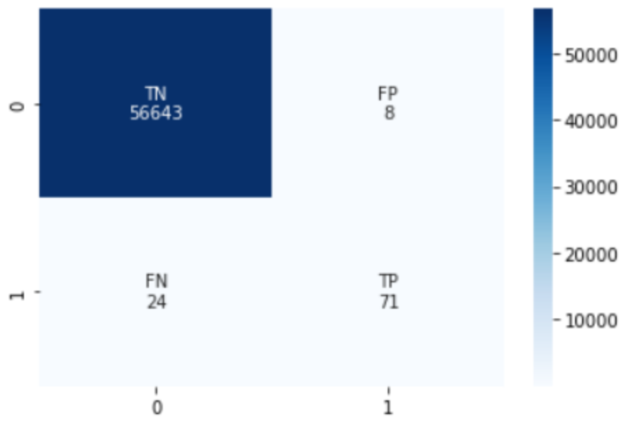
We began by building our support vector machine model without applying any sampling method to it. During hyperparameter grid optimization linear and radial basis functions were given to the kernel.

After running the Halving Grid Search method, the optimal parameter retrieved was the **linear function**. The optimal parameter was used to train our SVM model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 20: RESULTS FOR SVM WITHOUT SAMPLING

Metric	**Output**
Recall Score	**0.747**
Precision Score	**0.900**
F1-Score	**0.816**

FIGURE 40: CONFUSION MATRIX FOR SVM WITHOUT SAMPLING



The model performed really well without applying any balancing method to it, achieving a precision score of 90% having low false positives and relatively low false negatives resulting to an F1 score on 81.6%.

5.4.2 **Balancing Method:** Under Sampling

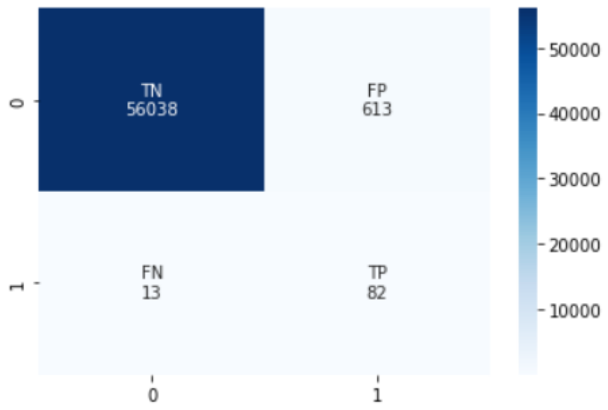
Here we randomly under sampled our target variable of our training set where we reduced the instances of genuine transactions to match the fraudulent one and started building our SVM model. During hyperparameter grid optimization, same parameters were given as section 5.4.1.

After running the Halving Grid Search method, the optimal parameter retrieved was the **radial basis function.** The optimal parameter was used to train our SVM model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 21: RESULTS FOR SVM USING UNDERSAMPLING

Metric	Output
Recall Score	**0.863**
Precision Score	**0.117**
F1-Score	**0.207**

FIGURE 41: CONFUSION MATRIX FOR SVM USING UNDERSAMPLING



Just like all the other models so far, under sampling has performed poorly on SVM as well achieving an F1 score of only 20.7%.

5.4.3 **Balancing Method:** Random Over Sampling

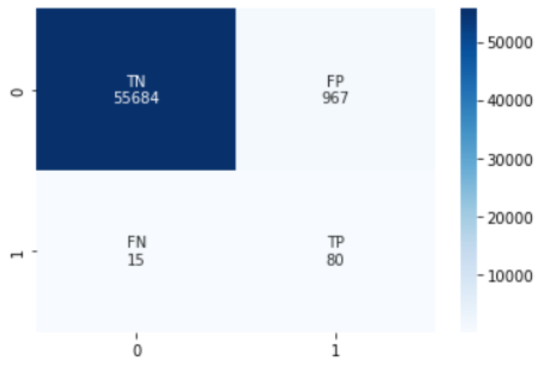
Here we randomly over sampled our target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our SVM model. During hyperparameter grid optimization, same parameters were given as section 5.4.1.

After running the Halving Grid Search method, the optimal parameter retrieved was the **radial basis function**. The optimal parameter was used to train our SVM model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 22: RESULTS FOR SVM USING RANDOM OVERSAMPLING

Metric	Output
Recall Score	**0.842**
Precision Score	**0.08**
F1-Score	**0.140**

FIGURE 42: CONFUSION MATRIX FOR SVM USING RANDOM OVERSAMPLING



By using oversampling the model was able to achieve a relatively high recall score of 84.2% but at the expense of high false positives that resulted in a low precision of only 8%.

5.4.4 **Balancing Method**: SMOTE

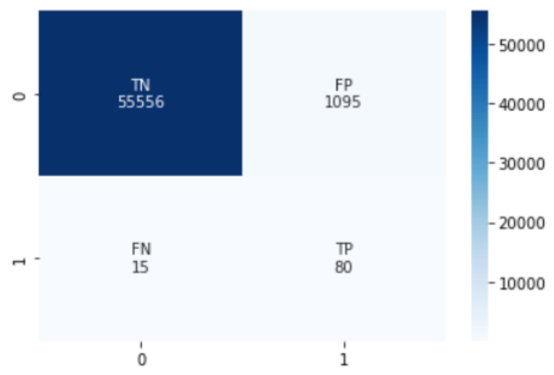
Here we used the SMOTE techniques to over sample the fraudulent class. During hyperparameter grid optimization, same parameters were given and the same optimal parameter retrieved as section 5.4.3.

The optimal parameter was used to train our SVM model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 23: RESULTS FOR SVM USING SMOTE

Metric	Output
Recall Score	**0.842**
Precision Score	**0.068**
F1-Score	**0.126**

FIGURE 43: CONFUSION MATRIX FOR SVM USING SMOTE



By using SMOTE the model achieved very similar results to random oversampling achieving a low F1 score of only 12.6%.

5.4.5 **Balancing Method:** ADASYN

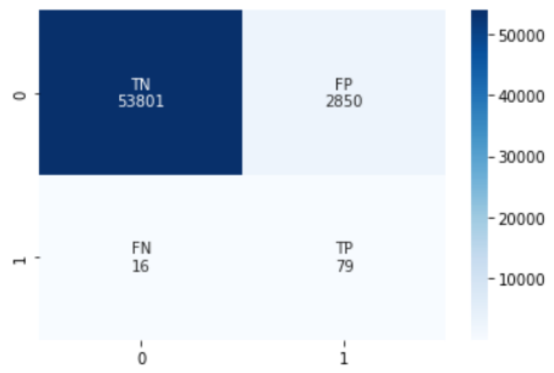
Here we used the ADASYN techniques to over sample the fraudulent class. During hyperparameter grid optimization, same parameters were given, and the same optimal parameter retrieved as section 5.4.3.

The optimal parameter was used to train our SVM model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 24: RESULTS FOR SVM USING ADASYN

Metric	**Output**
Recall Score	**0.83**
Precision Score	**0.027**
F1-Score	**0.05**

FIGURE 44: CONFUSION MATRIX FOR SVM USING ADASYN



ADASYN has performed poorly just like all the other over-sampling methods for SVM model achieving an F1 score of only 5% with extremely high false positives which are reflected in the recall score achieving 83%.

5.4.6 Summary of SVM using different balancing methods

We can notice from the table below that SVM performed best **without** using any sampling technique achieving an F1 score of 81.6% which is around a **63.5% improvement** from its baseline model. ADASYN performed the worst, resulting in an F1- score of only 5%.

TABLE 25: SUMMARY OF RESULTS FOR SVM

Balancing Method	F1-Score	Recall-Score	Precision-Score
Without Sampling	0.816	0.747	0.9
Under-Sampling	0.207	0.863	0.117
Random Over-Sampling	0.140	0.842	0.08
SMOTE	0.126	0.842	0.068
ADASYN	0.05	0.83	0.027

5.5 K-Nearest Neighbour (KNN)

5.5.1 **Balancing Method**: Without Sampling

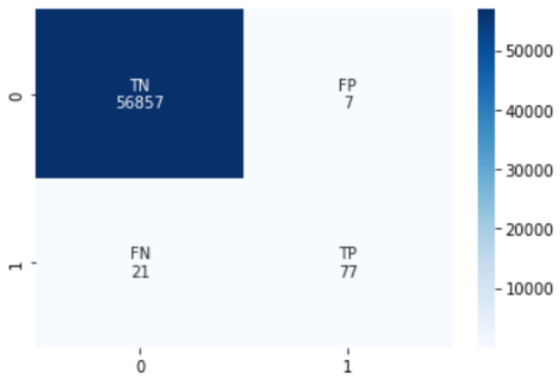
Started building our KNN model without applying any sampling methods. For hyperparameter grid optimization the values given for the number of neighbors (n_neighbors) ranged from 2 to 8 and values for power parameter (p) ranged from 2 to 5.

After running the Halving Grid Search method, the optimal parameters retrieved were 'n_neighbors'= 8 and 'p'= 4.The optimal parameters was used to train our KNN model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 26: RESULTS FOR KNN WITHOUT SAMPLING

Metric	Output
Recall Score	**0.785**
Precision Score	**0.916**
F1-Score	**0.846**

FIGURE 45: CONFUSION MATRIX FOR KNN WITHOUT SAMPLING



KNN performed relatively well without applying any balancing technique to it, achieving very low false positives and relatively low false negatives resulting in an F1- Score of 84.6%.

5.5.2 **Balancing Method:** Random Under-Sampling

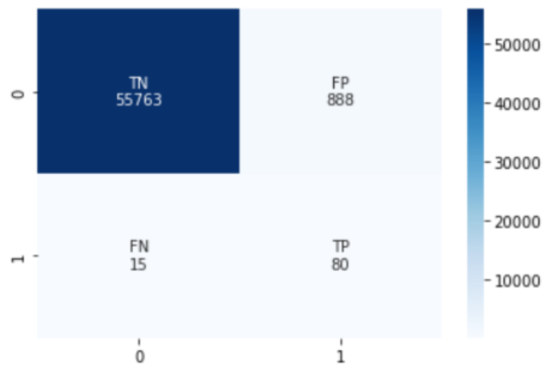
We randomly under sampled our target variable of our training set where we reduced the instances of genuine transactions to match the fraudulent one and started building our KNN model. During hyperparameter grid optimization we tried different values from number of nearest neighbors ranging from 2 to 16 and power parameter ranging from 2 to 5.

After running the Halving Grid Search method, the optimal parameters retrieved were 'n_neighbors'= 2 and 'p'= 2. The optimal parameters was used to train our KNN model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 27: RESULTS FOR KNN USING UNDERSAMPLING

Metric	**Output**
Recall Score	**0.842**
Precision Score	**0.082**
F1-Score	**0.15**

FIGURE 46: CONFUSION MATRIX FOR KNN USING UNDERSAMPLING



By applying random under sampling our model has worsened significantly resulting in an F1 score of only 15% and very high false positives.

5.5.3 **Balancing Method**: Random Over-Sampling

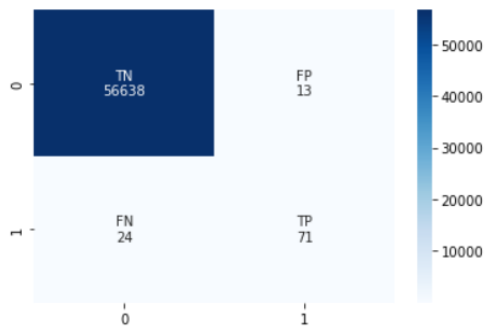
We randomly over sampled our target variable of our training set where we increased the instances of our fraudulent transactions to match the genuine ones and started building our KNN model. During hyperparameter grid optimization, same parameters were given as section 5.5.2.

After running the Halving Grid Search method, the optimal parameters retrieved were **'n_neighbors'= 2, 'p'= 4**. The optimal parameter was used to train our KNN model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 28: RESULTS FOR KNN USING RANDOM OVERSAMPLING

Metric	**Output**
Recall Score	**0.747**
Precision Score	**0.845**
F1-Score	**0.793**

FIGURE 47:CONFUSION MATRIX FOR KNN USING RANDOM OVERSAMPLING



By applying random oversampling our model did not improve compared to without any sampling method achieving an F1 score of 79.3%

5.5.4 **Balancing Method**: SMOTE

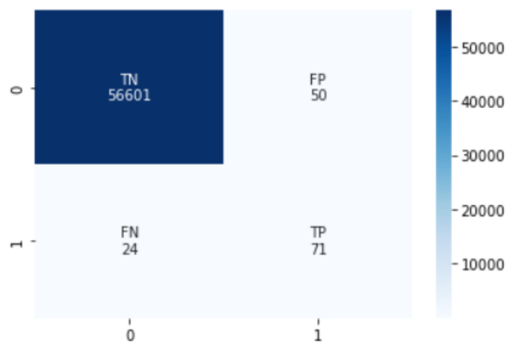
Here we used the SMOTE techniques to over sample the fraudulent class. During hyperparameter grid optimization, same parameters were given as section 5.5.2.

After running the Halving Grid Search method, the optimal parameters retrieved were **'n_neighbors'= 2, 'p'= 2**. The optimal parameter was used to train our KNN model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 29: RESULTS FOR KNN USING SMOTE

Metric	**Output**
Recall Score	**0.747**
Precision Score	**0.586**
F1-Score	**0.657**

FIGURE 48: CONFUSION MATRIX FOR KNN USING SMOTE



SMOTE methods have even performed lower than random over sampling resulting in an F1 score of 65.7%.

5.5.5 Balancing Method: ADASYN

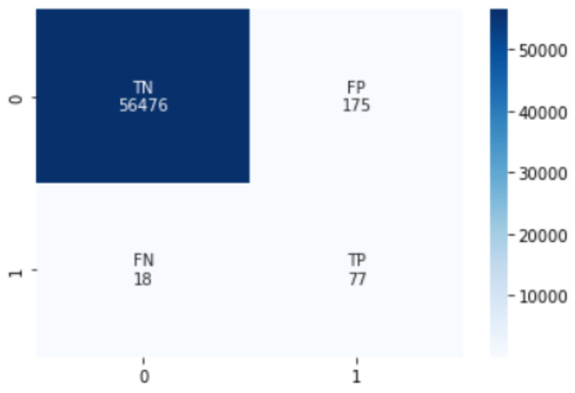
Here we used the ADASYN techniques to over sample the fraudulent class. During hyperparameter grid optimization, same parameters were given as section 5.5.2.

After running the Halving Grid Search method, the optimal parameters retrieved were 'n_neighbors'= 8, 'p'= 2. The optimal parameter was used to train our KNN model using our training sets and then tested our trained model on unseen test data and compared its predicted values with its true values of the test set using F1, recall, and precision that were be calculated from our confusion matrix.

TABLE 30: RESULTS FOR KNN USING ADASYN

Metric	Output
Recall Score	**0.810**
Precision Score	**0.305**
F1-Score	**0.444**

FIGURE 49: CONFUSION MATRIX FOR KNN USING ADASYN



ADASYN performed the poorest comparing to other over-sampling methods for KNN resulting in an F1 score of only 44.4%.

5.5.6 **Summary of KNN using different balancing methods:**

We can notice from the table below that KNN performed best **without** using any sampling technique, achieving an F1 score of 84.6% which is around a **69.5% improvement** from its baseline model. Under sampling performed the worst, resulting in an F1- score of 15%.

TABLE 31: SUMMARY OF RESULTS FOR KNN

Balancing Method	F1-Score	Recall-Score	Precision-Score
Without Sampling	0.846	0.785	0.916
Under-Sampling	0.15	0.842	0.082
Random Over-Sampling	0.793	0.747	0.845
SMOTE	0.657	0.747	0.586
ADASYN	0.444	0.810	0.306

5.6 Comparison of the top performing models

After we have interpreted the results of all the balancing methods for every model, we are going to evaluate and compare which top model performed the best and what balancing technique was used. We will mainly be focusing on using the F1 score to compare the models since it measures the harmonic mean of recall and precision, and we care about our models to be able to predict fraudulent transactions with high precision but not at the expense of high false positives since this would cause customer dissatisfaction which would result in them switching to different competitors.

TABLE 32: RESULTS OF BEST BALANCING METHOD FOR EACH MODEL

	Best Performing				
Model	**Balancing Method**	**F1- Score**	**Recall Score**	**Precision Score**	**Improvement from baseline based on F1 score**
Logistic Regression	Without Sampling	77.3%	77.3%	77.3%	**54.9%**
Random Forest	Without Sampling	89.1%	83.7%	95.3%	**78.6%**
XG-Boost	Random Over-Sampling	85.0%	81.0%	89.5%	**70.3%**
Support Vector Machi	Without Sampling	81.6%	74.7%	90.0%	**63.5%**
K-Nearest Neighbor	Without Sampling	84.6%	78.5%	91.6%	**69.5%**

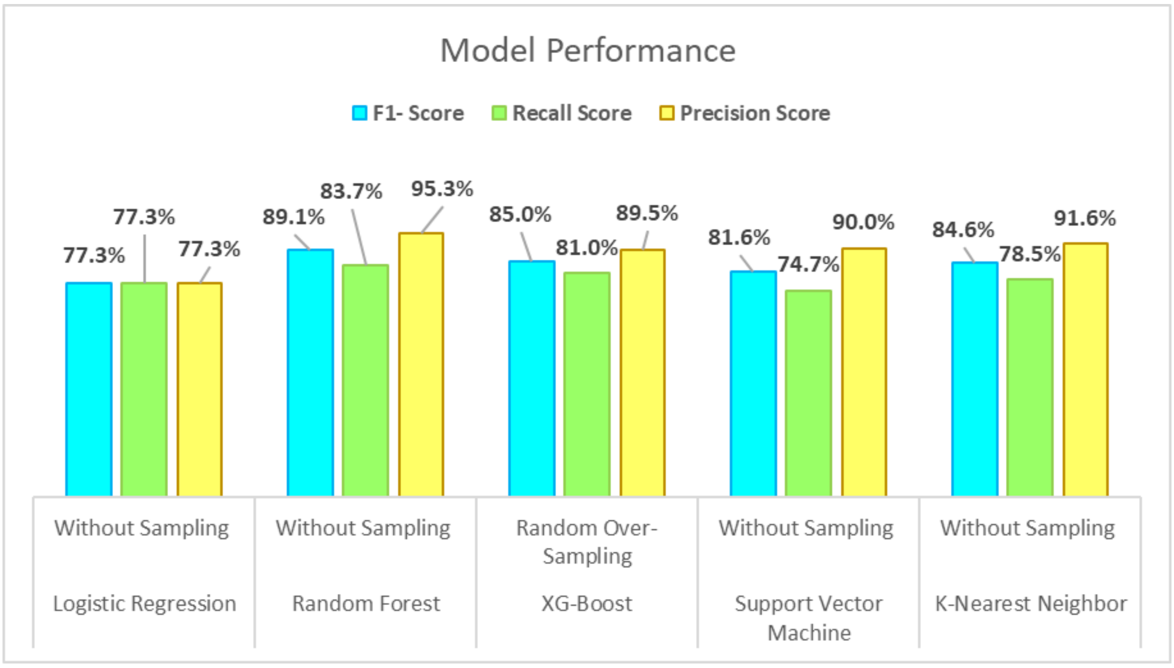


FIGURE 50: BAR GRAPH OF BEST BALANCING METHOD FOR EACH MODEL

From the above graph, we can see that all top models performed really well without having to apply any sampling technique to them except for XG-boost where its optimal performance was achieved by using random oversampling.

Models F1 scores ranged from 77.3% - 89.1%, with random forest **without any sampling** performing the best, achieving a **78.6%** improvement from its baseline model with an F1, recall and precision scores of 89.1%, 83.7%, and 95.3%, respectively showing that the model was able to predict fraudulent transactions with high precision while maintaining low false positives and low negatives. Ensemble methods performed better than distance-based algorithms where random forest and XG-boost were the top two best models.

Chapter 6 Conclusion and Recommendations

In this chapter we aim to conclude the findings of our research questions and come up with recommendations that future researchers can do in order to improve on our current findings.

Having an effective fraud detection system that identifies fraudulent transaction accurately is a challenge faced by many financial sectors around the world, especially with the rise of credit usage. The financial sectors are losing millions of dollars each year in reimbursements caused by credit fraud. Many challenges arise in detecting credit card fraud amongst which are severe class imbalance. Therefore, we built machine learning models using different sampling methods on a European dataset through:

- Applying an extensive exploratory data analysis where we discarded extreme outliers and treated skewness through feature transformation and ensured all our predictors had similar scaling in order for our distance-based algorithms would perform efficiently.
- Using the SHAPLEY value algorithm for feature selection has improved the performance of our model by increasing the performance for random forest from an F1 score of 85% to 89.1%, and it improved the model's computational speed.
- Using supervised learning algorithms by applying Logistic Regression, Random Forest, XG-Boost, KNN, and SVM and compared them using several sampling strategies by using random under-sampling, random oversampling, SMOTE, and ADASYN to balance our target. Sampling methods turned out to be ineffective in treating extreme class imbalance for our data set. Applying sampling methods on distance-based algorithms namely logistic regression, SVM and KNN resulted in our models to produce poor results in detecting fraudulent transactions. While ensemble algorithms namely XGB and random forest produced similar or slightly worse results comparing to not applying any sampling methods to our models, except for XGB using random oversampling were it produced a slight better result than without any sampling. Showing that balancing methods do not add value in producing better results on extreme imbalanced class.
- Under sampling method performed poorly on all algorithms. It produced the worst results on all algorithms except for SVM. This shows that under sampling method is extremely ineffective in performing well on extreme imbalance data sets.
- Random forest without any sampling methods proved to be the best in differentiating between genuine and fraudulent transactions achieving high precision and recall scores

of 83.7% and 95.3%, respectively, resulting in an overall F1 score of 89.1%. A 78.6% improvement from the baseline model.

The **recommendations** based on the conducted analysis and the produced models are as follows:

- Having a dataset that has **not been** previously transformed through PCA would allow us to gain better insight.
- The data retrieved from Kaggle is from 2013, therefore, having a more recent dataset would have increased our credibility in our findings since people's purchasing behaviours always change, and fraudsters always look for new ways to commit fraud.
- Our data set was extremely imbalanced; adding instances to our minority class would help the algorithms learn much better
- Having machines that can carry heavy computations would have allowed us to run deep learning models such as Artificial neural networks since it has the ability to learn well from complex relations.
- Balancing methods did not improve model's ability to learn better in most case, therefore switching your efforts more on applying extensive feature engineering and feature selection could enhance you model's ability to learn well.
- Switching the problem to an anomaly detection instead of a classification problem by using unsupervised learning algorithms such as autoencoders could produce better results in identifying frauds
- Having explainable features instead of transformed features (PCA), since this will allow us to come up with various feature engineering which would help create greater associations with our target variable.
- Having explainable features instead of transformed features (PCA) would have allowed us to come up with better decision-making and interpretations for our important features.
- Try different methods to balance our data such as adding more penalty weights to our minority class could improve the model's differentiating between classes.

References

Abd, S. and Abraham, A. (2013). A Review of Class Imbalance Problem. *Journal of Network and Innovative Computing*, [online] 1, pp.332–340. Available at: http://ias04.softcomputing.net/jnic2.pdf.

Abdallah, A., Maarof, M.A. and Zainal, A. (2016). Fraud detection system: A survey. *Journal of Network and Computer Applications*, [online] 68, pp.90–113. doi:10.1016/j.jnca.2016.04.007.

ACM Conferences. (2022). *FAST | Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. [online] Available at: https://dl.acm.org/doi/10.1145/1401890.1401910 [Accessed 25 Aug. 2022].

Bex T (2021). *11 Times Faster Hyperparameter Tuning with HalvingGridSearch*. [online] Medium. Available at: https://medium.com/towards-data-science/11-times-faster-hyperparameter-tuning-with-halvinggridsearch-232ed0160155 [Accessed 25 Aug. 2022].

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

Bolón-Canedo, V. and Alonso-Betanzos, A. (2019). Ensembles for feature selection: A review and future trends. *Information Fusion*, [online] 52, pp.1–12. doi:10.1016/j.inffus.2018.11.008.

Bolton, R.J. and Hand, D.J. (2002). Statistical Fraud Detection: A Review. *Statistical Science*, [online] 17(3). doi:10.1214/ss/1042727940.

Brownlee, J. (2020). *How to Avoid Data Leakage When Performing Data Preparation*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/data-preparation-without-data-leakage/ [Accessed 20 Aug. 2022].

Chan, C. (2018). *What is a ROC Curve and How to Interpret It*. [online] Displayr. Available at: https://www.displayr.com/what-is-a-roc-curve-how-to-interpret-it/ [Accessed 9 Aug. 2022].

Chomboon, K., Chujai, P., Teerarassammee, P., Kerdprasop, K. and Kerdprasop, N. (2015). An Empirical Study of Distance Metrics for k-Nearest Neighbor Algorithm. *The Proceedings* *of the 2nd International Conference on Industrial Application Engineering 2015*. doi:10.12792/iciae2015.051.

Com, L. and Hinton, G. (2008). Visualizing Data using t-SNE Laurens van der Maaten. *Journal of Machine Learning Research*, [online] 9, pp.2579–2605. Available at: https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf?fbcl.

Correa Bahnsen, A., Aouada, D., Stojanovic, A. and Ottersten, B. (2016). Feature engineering strategies for credit card fraud detection. *Expert Systems with Applications*, [online] 51, pp.134–142. doi:10.1016/j.eswa.2015.12.030.

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

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10), p.78. doi:10.1145/2347736.2347755.

Fadaei Noghani, F and Moattar, M. (2017). Ensemble Classification and Extended Feature Selection for Credit Card Fraud Detection. *Journal of AI and Data Mining*, [online] 5(2), pp.235–243. doi:10.22044/jadm.2016.788.

Fletcher, T. (2008). *Support Vector Machines Explained*. [online] Available at: https://www.csd.uwo.ca/~xling/cs860/papers/SVM_Explained.pdf.

Forman, G., Guyon, I. and Elisseeff, A. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research*, [online] 3, pp.1289–1305. Available at: https://www.jmlr.org/papers/volume3/forman03a/forman03a_full.pdf [Accessed 25 Aug. 2022].

Fu, K., Cheng, D., Tu, Y. and Zhang, L. (2016). Credit Card Fraud Detection Using Convolutional Neural Networks. *Neural Information Processing*, [online] pp.483–490. doi:10.1007/978-3-319-46675-0_53.

Haibo He, Yang Bai, Garcia, E.A. and Shutao Li (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on*

Neural Networks (IEEE World Congress on Computational Intelligence). [online] doi:10.1109/ijcnn.2008.4633969.

Ieee.org. (2013a). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9317228 [Accessed 25 Aug. 2022].

Ieee.org. (2013b). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4667275 [Accessed 25 Aug. 2022].

Ieee.org. (2013c). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4633969 [Accessed 25 Aug. 2022].

Ieee.org. (2013d). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9214206&tag=1 [Accessed 25 Aug. 2022].

Ieee.org. (2013e). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9214206&tag=1 [Accessed 25 Aug. 2022].

Jair.org. (2022). *View of SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary*. [online] Available at: https://www.jair.org/index.php/jair/article/view/11192/26406 [Accessed 25 Aug. 2022].

Laleh, N. and Abdollahi Azgomi, M. (2009). A Taxonomy of Frauds and Fraud Detection Techniques. *Information Systems, Technology and Management*, [online] pp.256–267. doi:10.1007/978-3-642-00405-6_28.

MaPS. (2021). *What is credit card fraud and how can I prevent it?* [online] Available at: https://www.moneyhelper.org.uk/en/blog/scams-and-fraud/what-is-credit-card-fraud-how-prevent-it [Accessed 16 Aug. 2022].

Marko, M. (2019). *Feature selection for unbalanced class distribution and Naive Bayes*. [online] In Proceedings of the 16th International Conference on Machine Learning (ICML.

Available at: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.2544&rank=1&q=Feature%20selection%20for%20unbalanced%20class%20distribution%20and%20Naive%20Bayes&osm=&ossid= [Accessed 25 Aug. 2022].

Mishra, A. and Ghorpade, C. (2018). Credit Card Fraud Detection on the Skewed Data Using Various Classification and Ensemble Techniques. *2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*. [online] doi:10.1109/sceecs.2018.8546939.

Ngai, E.W.T., Hu, Y., Wong, Y.H., Chen, Y. and 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*, [online] 50(3), pp.559–569. doi:10.1016/j.dss.2010.08.006.

Patel, H. (2021). *What is Feature Engineering — Importance, Tools and Techniques for Machine Learning*. [online] Medium. Available at: https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10 [Accessed 25 Aug. 2022].

Proquest.com. (2022). *Access Error - Unidentified Location 3001 - ProQuest*. [online] Available at: https://www.proquest.com/openview/739321feff86eff0c2eeeaee3ea8d3bb/1?pq-origsite=gscholar&cbl=18750&diss=y [Accessed 25 Aug. 2022].

Richardson, M. (2009a). *Principal Component Analysis*. [online] Available at: http://aurora.troja.mff.cuni.cz/nemec/idl/09bonus/pca.pdf.

Richardson, M. (2009b). *Principal Component Analysis*. [online] Available at: http://aurora.troja.mff.cuni.cz/nemec/idl/09bonus/pca.pdf.

Ross, G., Das, S., Sciro, D. and Raza, H. (2021). CapitalVX: A machine learning model for startup selection and exit prediction. *The Journal of Finance and Data Science*, [online] 7, pp.94–114. doi:10.1016/j.jfds.2021.04.001.

Soleymani, A. (2022). *Stop using random forest feature importances. Take this intuitive approach instead.* [online] Medium. Available at:

https://medium.com/@ali.soleymani.co/stop-using-random-forest-feature-importances-take-this-intuitive-approach-instead-4335205b933f [Accessed 22 Aug. 2022].

Srivastava, A., Kundu, A., Sural, S. and Majumdar, A.K. (2008). Credit Card Fraud Detection Using Hidden Markov Model. *IEEE Transactions on Dependable and Secure Computing*, [online] 5(1), pp.37–48. doi:10.1109/tdsc.2007.70228.

Susmaga, R. (2004). Confusion Matrix Visualization. *Intelligent Information Processing and Web Mining*, [online] pp.107–116. doi:10.1007/978-3-540-39985-8_12.

thakur (2022). *abhishekkrthakur/approachingalmost*. [online] GitHub. Available at: https://github.com/abhishekkrthakur/approachingalmost/blob/master/AAAMLP.pdf.

The Nilson Report (2019). *Payment Card Fraud Losses Reach $27.85 Billion*. [online] Prnewswire.com. Available at: https://www.prnewswire.com/news-releases/payment-card-fraud-losses-reach-27-85-billion-300963232.html [Accessed 16 Aug. 2022].

Tuyls, K., Maes, S. and Bram Vanschoenwinkel (2022). *Machine Learning Techniques for Fraud Detection*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/254198382_Machine_Learning_Techniques_for_Fraud_Detection [Accessed 25 Aug. 2022].

Varmedja, D., Karanovic, M., Sladojevic, S., Arsenovic, M. and Anderla, A. (2019). Credit Card Fraud Detection - Machine Learning methods. *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*. [online] doi:10.1109/infoteh.2019.8717766.

Whatman, P. (2022). *Credit card statistics 2022: 65+ facts for Europe, UK, and US*. [online] Spendesk.com. Available at: https://blog.spendesk.com/en/credit-card-statistics [Accessed 23 Aug. 2022].

www.kaggle.com. (n.d.). *Credit Card Fraud Detection*. [online] Available at: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

Xu-Ying Liu, Jianxin Wu and Zhi-Hua Zhou (2009). Exploratory Undersampling for Class-Imbalance Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, [online] 39(2), pp.539–550. doi:10.1109/tsmcb.2008.2007853.

Zhang, X. and Qin, L. (2022). An Improved Extreme Learning Machine for Imbalanced Data Classification. *IEEE Access*, [online] 10, pp.8634–8642. doi:10.1109/access.2022.3142724.