Credit Card Fraud Detection

Soumitra Chakraborty

Student ID - 201754798

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

September 2024

Student Declaration

SIGNATURE	Soumitra Chakraborty
DATE	September 19, 2024

Acknowledgments

I would like to convey and express my sincerest gratitude to everyone who has supported and guided me throughout the course of this dissertation.
Firstly, I would like to extend my sincerest appreciation towards my primary supervisor **Mrs. Ramya Chavali**, whose guidance, insight and encouragement were invaluable and helped me complete this dissertation. I would also like to express my gratitude towards my secondary supervisor **Dr. Prudence Wong** for her helpful insights and suggestions which helped me enhance the dissertation. I would also like to thank **Dr. Tony Tan** and **Dr. Stuart Thomason** for constantly uodating us about the project requirements specifications and ethical considerations.
To my parents, I express my unending love and gratitude for providing me with the love, encouragement and support which helped me complete this dissertation.
I would also like to thank my peers and friends whose companionship and moral support have been indispensable. Finally, I would like to thank each and everyone of the authors and contributors whose work has been referenced here without whose contribution it would have been much more harder to take on this topic.

Credit Card Fraud Detection

Abstract

In the era of digital transactions, credit card fraud is a pervasive issue that results in significant financial losses and compromises consumer trust. Detecting fraudulent transactions, which often represent a small percentage of overall activity, poses unique challenges due to the imbalanced nature of the data. This study explores a range of supervised learning and deep learning techniques to identify the most effective anomaly detection model for credit card fraud. Five supervised learning algorithms are evaluated: Random Forest, K-Nearest Neighbours (KNN), Support Vector Machine (SVM), Gradient Boosting, and Logistic Regression, along with two deep learning approaches: Multilayer Perceptron (MLP) and 1D Convolutional Neural Networks (CNN). Given the imbalance in fraud detection datasets, the models are evaluated primarily using the ROC-AUC score, which provides a robust measure of the trade-off between true-positive and false-positive rates. Other key performance metrics, such as precision, recall, and F1 score, are also used to provide a comprehensive evaluation.

Furthermore, a Streamlit dashboard is developed to visualise model performance, including the ROC curves, allowing stakeholders to interactively explore each algorithm's effectiveness. This dashboard offers a practical tool for real-time monitoring and comparison of fraud detection models. The ultimate goal is to identify the most accurate and reliable model for detecting fraudulent transactions while providing a user-friendly platform to assist financial institutions in decision making and fraud prevention efforts.

Statement of Ethical Compliance

Declaration

Statement of Ethical Compliance

Data Category: **B**
Participant Category: **0**

I confirm that I have read the ethical guidelines and will follow them during this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the increasing reliance on digital transactions, the occurrence of credit card fraud has risen substantially, posing significant challenges for financial institutions. Fraudulent transactions not only lead to financial losses, but also erode consumer trust. "Traditional methods of fraud detection often rely on rule-based systems" as written in [1], which do not adapt to the evolving tactics used by fraudsters. This calls for the development of advanced anomaly detection systems, particularly using machine learning and deep learning techniques, to accurately detect fraudulent transactions in real time.
This dissertation focuses on building a robust system for detecting credit card fraud using a combination of supervised learning and deep learning algorithms. By evaluating and comparing the performance of these algorithms, the goal is to find the most effective model for anomaly detection in highly imbalanced datasets, where fraudulent transactions constitute a very small percentage of overall transactions. The system also includes a user-friendly Streamlit dashboard to visually and compare the performance of various models, making it easier for end users to interpret the results and make informed decisions.

1.1 Background and Literature Review

The most important work at the beginning of this project is the review of the literature on the topic and the specifics and how it can be implemented. The core of the work is based on the research done by Dal Pozzolo et al. in their paper [2], which gives a detailed overview on how to detect credit card fraud using machine learning as a tool. The idea was then expanded upon by performing a comparative analysis by Awoyemi et al. in their article [3], where a comparative analysis between three machine learning models was discussed. The data set for the training and evaluation was found[4], which was then found to be heavily unbalanced in nature, The article by Parekh et al.[5] delves deeper into the sampling techniques. Due to the decision to take MLP and CNNs into account during the comparative analysis, the article by Yu et al. [6] was referred to the use of deep learning models for the use credit fraud and anomaly detection. Finally, the author of [7] had shown the use of Streamlit as a viable candidate for the purpose of a similar comparative analysis. These were the primary research papers and works that were used a starting point for the topic and further work was based on these works and implemented.

1.2 Aims and Requirements

Aims

- Develop a credit card fraud detection system that can handle imbalanced data effectively.
- Compare multiple machine learning and deep learning algorithms to find the best-performing model for fraud detection.
- Integrate these models into a user-friendly Streamlit dashboard for real-time monitoring and visualisation of model performance.
- Ensure scalability and accuracy in real-world applications, where fraudulent transactions represent a small fraction of all transactions.

Requirements

For this credit card fraud detection project, the hardware requirements include a computer with a reasonably powerful processor and sufficient RAM to handle large datasets and complex algorithms. Graphics Processing Units (GPUs) could be beneficial for accelerating machine learning tasks. On the software side, a robust operating system like Windows, macOS, or Linux would be necessary. The Windows operating system version 11 was used. Programming tools and environments such as Python[8], R, or MATLAB are essential, along with integrated development environments (IDEs) like PyCharm, RStudio, or Jupyter Notebooks. Specialized libraries and frameworks for data analysis and machine learning, such as Pandas[9], NumPy[10], scikit-learn[11], Matplotlib[12], seaborn[13], TensorFlow[14], or PyTorch for deep learning models and algorithms, are crucial. The setup involved using Python version 3.12 alongside Jupyter Notebooks and Google Colab[15] for certain models requiring computational power. For the application of deep learning methods like MLP and CNN, TensorFlow, developed by Google, and the Keras API[16] built on TensorFlow were used. This setup was chosen because of the familiarity with TensorFlow and Python for deep learning models and supervised machine learning models. Python is relatively simple and powerful, with a host of libraries for data and statistical analysis. Additionally, TensorFlow allows for implementing the models and their subsequent hidden layers with greater ease and simplicity. ChatGPT[17] and Google Gemini[18] has been used for synthetic testing data generation, text paraphrasing, code debugging and code evaluation..

1.3 Scope

This dissertation will focus on implementing, testing, and evaluating the following algorithms for credit card fraud detection.
Random Forest, KNN, SVM, Gradient Boosting, and Logistic Regression (Supervised Learning) Multilayer Perceptron (MLP) and 1D Convolutional Neural Networks (Deep Learning) The system will utilise techniques such as SMOTE (Synthetic Minority Oversampling Technique) and RandomUnderSampler to handle data imbalance. Additionally, a Streamlit dashboard will be developed to visualise the results of each algorithm and allow users to interact with and compare model performance.

1.4 Problem Statement

Credit card fraud detection is a complex problem primarily due to the highly imbalanced nature of the dataset, where fraudulent transactions make up a very small percentage of all transactions. Existing rule-based systems fail to adapt to new fraud patterns, leading to poor detection rates and high false positives. There is a need for a more dynamic, intelligent system that can adapt to changing fraud patterns while maintaining high accuracy in detecting fraudulent activities.

1.5 Approach

The approach taken in this dissertation includes:

1. Data Preprocessing: The raw transaction data will be cleaned, and feature selection techniques will be applied to identify relevant features for fraud detection.
2. Handling Imbalanced Data: The data imbalance will be addressed using SMOTE for oversampling the minority class and RandomUnderSampler for undersampling the majority class.
3. Model Implementation: Several machine learning and deep learning algorithms will be implemented and trained on the preprocessed dataset. Model Evaluation: The models will be evaluated based on metrics such as precision, recall, F1 score, and ROC-AUC, with a focus on handling imbalanced data.
4. Streamlit Dashboard: A dashboard will be built to visualise the performance of each algorithm, providing interactive tools to compare and interpret the model results.

1.6 Outcome

The final outcome of this dissertation will be a credit card fraud detection system that:

1. Identifies fraudulent transactions with high precision and recall.
2. Uses a combination of machine learning and deep learning models to find the best anomaly detection technique.
3. Provides an interactive Streamlit dashboard for stakeholders to monitor and compare model performance.
4. Addresses data imbalance effectively using SMOTE and undersampling techniques, ensuring scalability and adaptability to real-world fraud detection scenarios.

By the end of this dissertation, a fully functional and scalable system will be developed, capable of accurately detecting fraud in real-world transaction data, and presented through an intuitive and interactive dashboard.

Chapter 2

Design

The intended outcome of the design process is first to identify the data set that can be used to train and test the models. This process involves performing some exploratory data analysis and outlier detection of the dataset to identify biases and the balance within. The next step is to identify suitable models that can be used for comparative analysis. These include some supervised machine learning algorithms, as the training data will be labelled, followed by models based on the Multilayer Perceptron (MLP) algorithm and the 1D Convolutional Neural Network. The models will be hyperparameter tuned to get the best possible results. Finally, the resultant models will then be loaded into a Streamlit dashboard which will be run on the local machine to visualise some test transactions and see the relevant evaluation metrics involved in judging and choosing the appropriate anomaly detection method/(s). Following is a step-by-step explanation of each of the procedures and take a closer look.

2.1 Initial Design Description

The initial design of the dashboard was as follows. The user would be taken to a dashboard where they would be asked to upload a csv file of the transactions they would like to analyse. After uploading, the user would have a choice between 7 models to perform the detection via a drop-down menu. After selection, the system would take the time required to pre-process the data and would then show the user the detected fraud transactions, the classification report of the data which has been uploaded alongside a confusion matrix for the model selected and any other evaluation that would be necessary for the user. An initial sketch of the dashboard has been shown in the image 2.1.

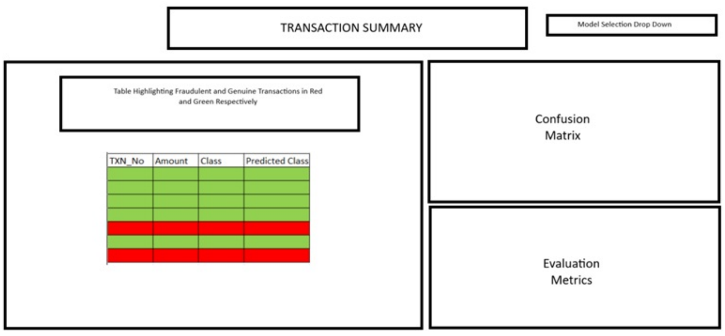

TRANSACTION SUMMARY
Model Selection Drop Down
Table Highlighting Fraudulent and Genuine Transactions in Red and Green Respectively
TXN_No
Amount
Class
Predicted Class
Confusion Matrix
Evaluation Metrics

Figure 2.1: Initial Design of the Dashboard

2.2 The Dataset and Exploratory Data Analysis (EDA)

2.2.1 Dataset Description

The authors of the articles [19], [3] have used a particular dataset for this. Awoyemi et al [3] have specifically used this dataset, performing a comparative analysis of different supervised machine learning models and this was a major starting point for the project as well. The dataset is open-source and can be found at kaggle.com [4]. The dataset contains 284,807 credit card transactions. These transactions were made in September 2013. To be precise, it contains 492 fraudulent transactions flagged as class '1' and the rest are flagged as genuine or class '0'. Initial inspection suggests that the data are imbalanced with around 0.172% of the data belonging to class '1'. This shows a highly unbalanced dataset. The values are numerical, which was achieved after a principal component analysis (PCA) [4]. There are 30 feature sets, namely 'Amount' of the transaction, 'Time' or time taken for the transaction, and column names 'V1', 'V2', all the way to 'V28'. The classification column is named 'Class' and only contains classes '0' and '1', to represent the class or type of transaction. Due to the data already being normalised by a PCA transformation, further PCA transformation would be inefficient with varying results for each and every model.

2.2.2 EDA and Understanding of the Data

EDA was performed on the data and the following conclusions were found.

- The data set is highly biased for the amount variable, with a mean of 88.35 and a maximum of 25,691.16.
- There is a low correlation between most of the features shown in the image 2.2, indicating that they are largely independent. This is expected due to PCA transformation. The correlation matrix was used by Parekh et al [5] for a similar analysis in their paper.
- The amount and time variables do not strongly correlate with other features.
- The amount is skewed to the right, with many transactions of small value and few transactions of high value as in the image 2.4.
- Time does not show a strong pattern, but is almost uniformly distributed (2.5).

.

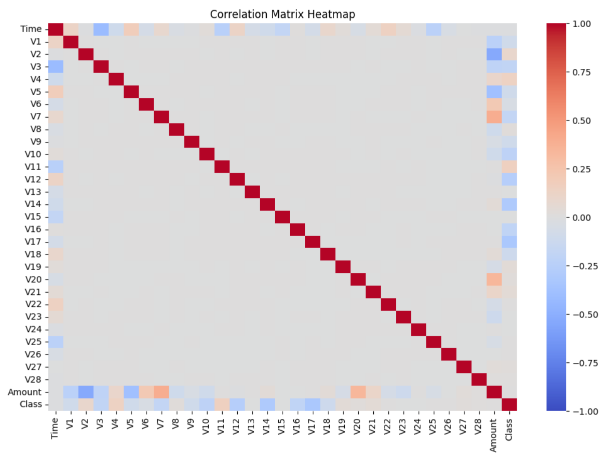


Figure 2.2: Correlation Matrix of all the feature sets

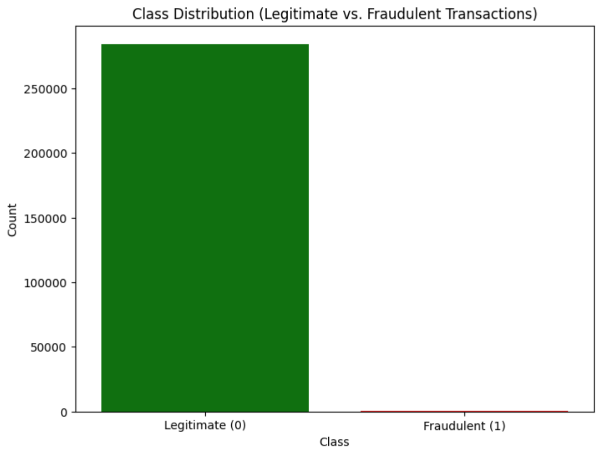


Figure 2.3: Class distribution of the Dataset

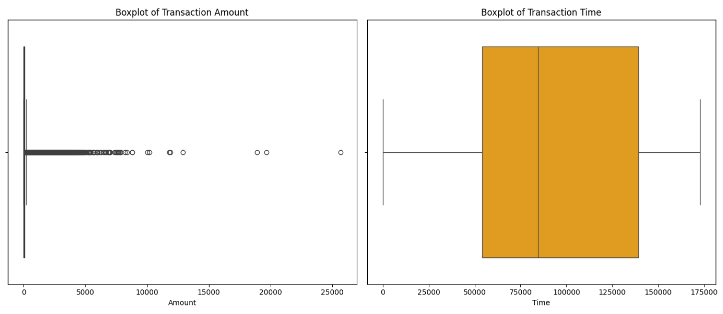


Figure 2.4: Outliers for the Amount column and Time column

2.3 Data Preprocessing and Preparation

The above analysis and description of the data provide an idea about what should be done in the data preprocessing and preparation process. It was decided to use a combination of oversampling and undersampling to get a more balanced dataset which was also manageable in nature. The general idea was shared by [20] in their research paper dealing with incomplete and unbalanced data. Parekh et al [5] also used a similar idea for their study in the same dataset.

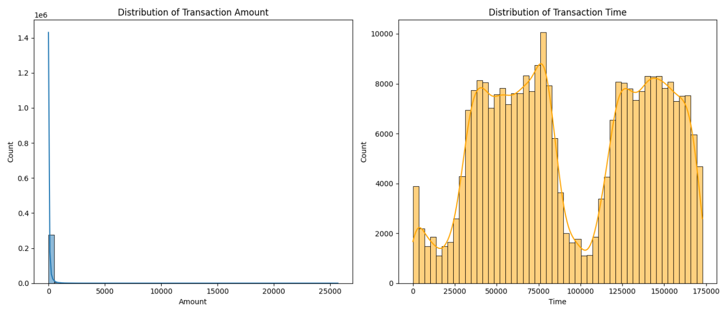


Figure 2.5: Data Distribution of the Amount columns and Time Columns

- The data will first be fed through an oversampling process to match the class count to a suitable number so as to have enough data of both kinds, for proper classification and identification.
- This up sampled data will then go through an undersampling layer so that the number of records is sufficient for the corresponding models without the risk of overfitting.
- Finally, the entire resampled data will be normalised to reduce the effect of outliers and deviation in results using a normalising layer.

The plan is to achieve the following steps with the help of a data pipeline to maintain a proper sequence. Each of the steps mentioned above will be executed sequentially. Further imputation and cleaning of data is not required as the dataset itself has no missing values, and there are no categorical features but only numerical features. The preprocessing and normalisation stage has been highlighted in the diagram 2.6 for future reference.

2.3.1 Sampling and Normalisation Techniques Used for Data Preprocessing

- RandomUnderSampler - This method was used for the purpose of this comparative analysis and study. In the research paper written by Zhu et al. [21], Random undersampling has been mentioned as one of the methods to remove noisy data for credit card fraud detection and thus was used. The implementation of this method is by importing imblearn.RandomUnderSampler in Python[22].
- Synthetic Minority Over-sampling Technique (SMOTE) - For upsampling the data, SMOTE was chosen for this particular dataset as it has been shown in [23], SMOTE was used in their research as well into deep network models. SMOTE synthesises new instances by interpolating between existing minority class instances. This ensures that the model receives more balanced data, helping it better learn the patterns of both classes. the implementation of this method is done by importing the imblearn.SMOTE library in Python[24].
- Pipeline - The use of the pipeline is important in this method to sequentially preprocess the data. The use of imblearn.pipeline imported in Python [25].
- It is important to normalise the data for the sake of faster and more efficient computation. The normalisation method used is the Standard Normalisation which has

been used in the research article[26], where the authors have developed the standard normalisation to detect credit card fraud / anomalies. The formula for this transformation is as follows where z is the changed or normalised value:

$$z = \frac{x - \mu}{\sigma} \tag{2.1}$$

where x is the value of a data, μ is the Mean and, σ is the Standard Deviation. Python library sklearn.preprocessing.StandardScaler was used to achieve this[27].

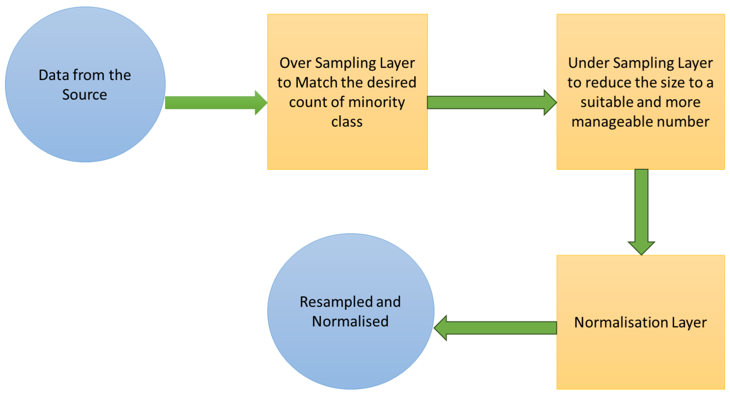

Data from the Source
Over Sampling Layer to Match the desired count of minority class
Under Sampling Layer to reduce the size to a suitable and more manageable number
Normalisation Layer
Resampled and Normalised

Figure 2.6: UML Diagram of the Pipeline used for Data Pre-processing and Preparation

2.4 Model Selection and Hyperparameter Tuning Methods

2.4.1 Model Selection

Selecting appropriate models is essential for detecting credit card fraud. In this subject, the effectiveness of deep learning and machine learning algorithms has been thoroughly investigated and validated.
Based on research from several projects where their performance has previously been shown, the models used for this project were selected. Random Forest, K-Nearest Neighbour (KNN), Logistic Regression, Support Vector Machine (SVM), and XGBoost are some of the machine learning models in use and the models chosen for this study.
In addition to these traditional machine learning models, I'm using deep learning techniques like the Multilayer Perceptron (MLP) and 1-Dimensional Convolutional Neural Network (1D CNN). Though 1D CNN is commonly used for applications like image or signal processing, it is surprisingly excellent at finding patterns in transaction data. MLP, on the other hand, excels in pattern recognition because it uses several layers for data analysis.

Random Forest Classifier

A very powerful machine learning technique called Random Forest (RF) aggregates the output of several decision trees. Random Forest constructs several decision trees during training as opposed to depending simply on one. The ultimate outcome for classification issues, like as credit card fraud detection, is the majority vote of all these trees. Put more simply, every tree makes a forecast, and the model selects the prediction that is most likely to occur[28],[29],[30]. By using this method, the likelihood of overfitting, a

situation in which a model performs well on training data but poorly on fresh, untested data—is decreased and the model becomes more accurate.
Random Forest's ability to handle complicated data is one of its many features. It can handle large feature (or variable) datasets without experiencing performance degradation. At every stage, only a random subset of characteristics is taken into account by each decision tree in the forest, which examines various portions of the data. The model remains adaptable and is less likely to overfit to noisy input due to this unpredictability. Random Forest is also a wonderful way to run several trees in parallel and speed up training time, especially with huge datasets, because each tree is independent of the others.
For this project, the use of Python scikit-learn library to build the Random Forest model. Specifically implementing it using the sklearn.ensemble.RandomForestClassifier() function to train and test the model. Parameter like 'n_estimators', 'max_depth', 'max_features' and 'gini' were chosen for tuning the model further for the use case[31].
The table 2.1 shows the range of values tested during hyperparameter tuning and the final parameters selected. Taking the time to fine-tune these parameters was key to making sure the model was both accurate and efficient, which is critical when trying to catch fraudulent transactions in real-world scenarios.

Parameters	Range
n_estimators	50, 100, 200
max_features	auto, sqrt, log2
max_depth	None, 10, 20, 30
criterion	gini, entropy

Table 2.1: Parameters and Range for Random Forest Model

Support Vector Machine (SVM)

Support Vector Machine (SVM)is a method of classification, useful in the identification of credit card fraud. The way SVM operates is by determining the ideal border, or "hyperplane," that best distinguishes between authentic and fraudulent transactions. Maximising the gap between the two groups is the key to making sure that even the transactions that are near the boundary—are correctly categorised. Because of this, SVM is especially effective in situations where the fraud data is greatly skewed or when there are minute distinctions between authentic and fraudulent transactions.
SVM is a good choice for handling this complexity because it can handle both linear and non-linear data with effectiveness and is resilient in high-dimensional areas[32],[33]. SVM may detect patterns and correlations in the data that might not be apparent in the original feature space by employing a kernel method to project the data into higher dimensions. Even in cases when fraudulent activities are uncommon and nuanced, SVM's capacity to identify hidden patterns makes it an excellent tool for fraud detection.
It was decided to implement the SVM method by importing the sklearn.svm.SVC() module in Python. The parameters of the SVM which were decided upon to be used for hyperparameter tuning are the 'C' value and the 'gamma' value where C is the regularisation component where the l2 regularisation method is and gamma is the kernel coefficient[34]. The parameters and the range chosen can be viewed in the table 2.2

Parameter	Range
C	0.1, 1, 10, 100
gamma	1, 0.1,0.01,0.001

Table 2.2: Parameters and Range for Support Vector Machine

K Nearest Neighbour Classifier (KNN)

The KNN classifier is used in anomaly detection algorithms and in this case is a good candidate for Credit Card Fraud Detection [35]. The goal is to compare the transactions with previously labelled data and to check if the said transaction is actually genuine. It is non-parametric in nature and instance-based learning algorithm.
The algorithm tries to find out the 'k' nearest neighbour to the data point and then makes the decision upon a majority type of the transaction it is. The distance metric that is used to find the neighbours are either the Euclidean distance, the Manhattan distance or the Minkowski distance[3]. It is also important to choose the correct value of 'k' or the number of nearest neighbours to be considered for the final evaluation to be as accurate as possible.
Due to its robustness and adaptability, KNN is often chosen as a method for credit card fraud detection. However, with higher-dimensionality and inbalaced datasets, a few issues can also be experienced.
The way the KNN algorithm was implemented was using the sklearn.neighbors.kNeighborsCLassifier() library from Python. The only parameter used for hyperparemeter tuning was the value 'k' within a range of 1 to 10 neighbors[36].

Logistic Regression

Logistic Regression is a statistical method for the binary classification of data. It uses multiple features from the dataset to make predictions[32]. It uses a logistic function like a sigmoid function to model the probability of a data point to be either class '0' or class '1'[37]. The output that is received is a probability score where a threshold value of 0.5 is kept for classification. A probability score less than 0.5 is class '0' while the probability score greater than 0.5 is class '1'. Since the problem statement at hand is basically an application of a binary classification, methods like Logistic Regression is a useful tool for analysis and has been used in a number of articles and research such [3], [38], [33], etc.
In the context of credit card fraud detection, due to its interpretability and efficiency, this model is a suitable candidate for this comparative analysis. Due its simplicity, Logistic Regresssion can also be considered as somewhat a baseline for this.
Logistic Regression was implemented using the sklearn.linear_model.LogisticRegression() module of Python. The parameters for tuning are the penalty which is used for regularisation, the 'C' value for the inverse of regularisation strength and the 'solver' to toggle through the different types of solver [39]. The parameter and the range have been shown in the following table 2.3.

Parameter	Range
penalty	l1,l2,elasticent,none
C	0.1,1,10,100
solver	lbfgs,liblinear,saga

Table 2.3: Paramaters and Range for Logistic Regression

Extreme Gradient Boosting Method (XGBoost

Extreme Gradient Boosting or XGBoost is one of the machine learning algorithms that has been considered for this comparative analysis. This makes use of ensemble learning where each tree is used to learn and improve from the output of the previous tree and corrects the error of the previous one as well. When a negative loss occurs, the algorithm then splits the nodes to the maximum possible depth and uses backpropagation to perform tree pruning to remove the splits[40].
This algorithm has been used in effect in many research papers and experiments involving credit card fraud, including[6],[32]. It is an effective ensemble strategy for anomaly detection. It is adept in handling missing value data and is quite quick and efficient. While the challenge of hyperparameter tuning remains, for this specific study, the XGBoost model is one of the ideal candidates.
The XGBoost model will be implemented in Python using the xgboost library[41]. The parameters chosen for the evaluatiopn and their respective values are given in table 2.4

Paramaters	Values
max_depth	6
learning_rate	0.1
eta	0.3
objective	binary:logistic
eval_metric	logloss

Table 2.4: Parameters and their values for XGBoost algorithm

Deep Learning Methods: Multilayer Perceptron (MLP)

A Multilayer Perceptron is a type of Artificial Neural Network, which is feedfowrawrd in nature. This neural netwrok has mutliple hidden layers and a output layer [42]. Due to its effectiveness in representing non-linear connections between features and variables, it has been used as one of the methods of study for credit card fraud detection such as [43], [32]. It is suitable for classification problems[44] such as a fraud detection problem. The input is fed through layers where each neuron adds a weight-adjusted sum and a non-linear activation function such as ReLU or a Sigmoid function is used to perform the classification. If the classification is incorrect, back-propogation is used to then change the weights and move forward. The process is robust and self learning. Some of the downsides include overfitting and computational cost.
The MLP was implemented using Tensorflow frame work for deep learning developed by Google [14] alongside the Keras API[16]. A view of the hidden layers and their specifications can be found in the table 2.5.

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 256)	7,936
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32,896
dropout_3 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 1)	129

Table 2.5: The Layers of the Multilayer Perceptron (MLP)

Deep Learning Methods: Convolutional Neural Networks (CNN)

Although CNNs have traditionally been used in image processing and image data, such as medical data such as radiology reports [45], they can be used to perform classification analysis on sequential data such as credit card transaction summaries[46],[26],[43]. It can extract relevant features from the data, reducing the need for manual feature engineering and extraction. CNNs are adept at identifying temporal patterns and can use the trends of genuine transactions overtime to form a basis for identification and deviations. It is also useful in the case of anomaly detection[47], [48], [49] where the CNN model can learn the salient features and properties of genuine transactions and then identify the fraudulent ones.
The advantages include its adaptability, performance in the event of pattern recognition, and automatic feature extraction. CNNs are often robust and scalable models and thus can be implemented with minimal hassle. Some of the problems that might be faced during the model implementation might include its computational power required to train, the complex architecture, the lack of interpretability due to being referred to as "black boxes"[50] and difficulty in discerning what is going on inside. Other risks with deep learning, such as overfitting, while being an issue, can also be mitigated by using regularisation and dropouts.
Just like the MLP model, the use of Keras and TensorFlow has been employed for the implementation of the CNN model, which is 1D in nature for the specific use case.
The model architecture and the associated layers can be seen in the table 2.6

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 24, 32)	256
batch_normalization_2 (BatchNormalization)	(None, 24, 32)	128
dropout_3 (Dropout)	(None, 24, 32)	0
conv1d_3 (Conv1D)	(None, 22, 64)	6208
dense_2 (Dense)	(None, 256)	360704
dropout_5 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 1)	257

Table 2.6: The layers of the 1D Convolutional Neural Network

2.4.2 Hyperparemeter Tuning

The main objective of credit card fraud detection models or anomaly detection models in general is to properly categorise or identify transactions or patterns as either genuine transactions or fraudulent in nature. It is essential to reach high values of recall score and accuracy score as misclassification can cause financial loss or annoyance for customers. Hyperparameter tuning is essential for improving the performance of machine learning model performance and neural network accuracy. GridSearchCV[51] and keras_tuner.RandomSearch[52] are two ways to achieve this goal. They maximise the performance of the model by identifying the ideal hyperparameters and searching in a specific predefined range.

GridSearchCV

GridSearchCV is an ideal candidate for performing hyperparameter tuning in supervised machine learning models, and the application is widespread and used in articles on medical classification, such as heart disease[53], pneumonia detection[54]. It also has application in credit card fraud detection as mentioned in [55]. Through a thorough search across a predetermined grid of parameter values, it automates the hyperparameter tuning process. Some of the features are:

- It helps find the ideal model which will give the maximum recall and accuracy score.
- It uses kfold()[56] for cross-validation to reduce overfitting.
- The process of hyperparameter tuning is automated and the final result is the best-fit model.

This is the method chosen for the supervised machine learning models such as Random Forest, SVM, Logistic Regression, and KNN Classifier.

RandomSearch

In cases of MLP and CNN, manually performing hyperparameter tuning can be a tideos and often time consuming process due to the large number of hyperparameters to be used for this process. As mentioned by [57] and [58], this process can be automated and simplified by using kers_tuner.RandomSearch(). Since the use of CNN and MLP is included in the comparative analysis, using RandomSearch makes sense for hyperparameter tuning and finding the correct model for evaluation and application. Some of the features are:

- RandomSearch selects random samples from the search space rather than testing every conceivable combination of hyperparameters. It becomes more computationally efficient as a result, particularly for deep learning models that demand long training times.
- For deep learning models used in fraud detection, hyperparameters like the number of layers, learning rate, batch size, and optimizer selection are crucial. RandomSearch efficiently explores these high-dimensional areas and offers a good balance between computing cost and precision.
- RandomSearch is a strong fit for the high-dimensional data that is commonly used to identify credit card fraud, and it performs well with larger datasets and complex models.
- It is ideal for real-time or almost real-time fraud detection systems because it does away with the necessity for the thorough search that grid search requires, which accelerates the model development cycle.

2.4.3 Evaluation Metrics

As this is a classification model analysis, the evaluation tools that should be used for gauging the performance should be the classification report containing the precision, recall, f1 score and accuracy. In addition to these metrics, the confusion matrix Receiver Operator Characteristic and Area Under the Curve (ROC-AUC), and the Precision-Recall Curve (PRC) will also be used to get a finer distinction between the models.
The use of the classification report and confusion matrix have been illustrated by the paper [32] and [59]. The use of the ROC-AUC matrix has been highlighted in the paper [59] and the use of the PRC has been shown in the paper[60].

Accuracy

The ratio of accurately anticipated transactions—fraud and non-fraud—to total transactions is known as accuracy. It is computed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

Precision

Precision quantifies the proportion of expected fraudulent transactions that turn out to be fraudulent, sometimes referred to as the positive predictive value. It is computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

Recall/Sensitivity/True Positivity

Recall quantifies the number of real fraudulent transactions that the model accurately identifies. It is computed as follows:

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

F1 Score

The F1 score is the harmonic mean of precision and recall, providing a balanced metric when there is a trade-off between these two. It is calculated as follows:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.5}$$

Model	Precision	Recall	F1 Score	Accuracy
SVM	0.9977	1	0.9989	0.9989
Random Forest	0.9995	0.9997	0.9996	0.9996
XGBoost	0.9997	0.9997	0.9997	0.9996
KNN	0.997	0.9998	0.9984	0.9983
Logistic Regression	0.9902	0.9666	0.9783	0.9842
MLP	0.996	0.9998	0.9979	0.9979
CNN	0.9925	0.9871	0.9898	0.9934

Table 2.7: Preliminary Evaluation Results for the models with respect to the fraudulent class in percentages

ROC-AUC Curve

The trade-off between the genuine positive rate (recall) and the false positive rate (1 - specificity) at different threshold values is displayed graphically via the ROC curve. The total capacity of the model to distinguish between classes is measured by the AUC (Area Under the Curve). A model that has a higher AUC is more effective in differentiating between authentic and fraudulent transactions.

Precision-Recall Curve

Plotting accuracy versus recall for various probability thresholds is what the precision-recall curve illustrates. It is especially helpful for datasets that are unbalanced since it concentrates on the positive class, or fraudulent transactions. When the positive class—fraud—is uncommon, the precision-recall curve's area under the curve (PR AUC) serves as a helpful performance indicator.
Here the terms TP, TN, FP, FN have the following terminologies:

- TP means True Positive which means when the genuine transaction has been identified as a genuine transaction.
- TN means True Negative which means when the fraudulent transaction has been identified as a fraudulent transaction.
- FP means False Positive which means the fraudulent transaction has been flagged as a genuine transaction.
- FN men's False Negative which means a fraudulent transaction has been identified as a genuine transaction.

The goal is to maximise the TP and TN values while minimising the FP and FN values as much as possible.

2.5 Flow Diagram for the Training Process

The following is the process represented in a flow diagram of the flow of data from the raw data source to the model training and model implementation of the models in the dashboard. Here are the following steps according to figure :

- The workflow starts with an imbalanced dataset where fraudulent transactions represent a small proportion of the total data. Imbalanced datasets are typical in fraud detection.
- Sampling techniques such as SMOTE and RandomUnderSampler will be used, ensuring that the model has enough fraudulent data points to learn from and avoid bias toward the majority class.
- The balanced dataset is split into a training dataset for model training. This training data is used to fit machine learning or deep learning models.
- GridSearchCV or keras_tuner.RandomSearch is applied to optimize the model's hyperparameters for better performance on the fraud detection task
- The trained and tuned model is evaluated using a test dataset that was not used during training. This ensures the model's ability to generalize to new, unseen data. The evaluation metrics will be used for this step.
- The final tuned model is integrated into a Streamlit app. The app allows users to interact with the model and analyze its results, providing a user-friendly interface for predictions. This would then be showed in a dashboard view in a browser, hosted locally.

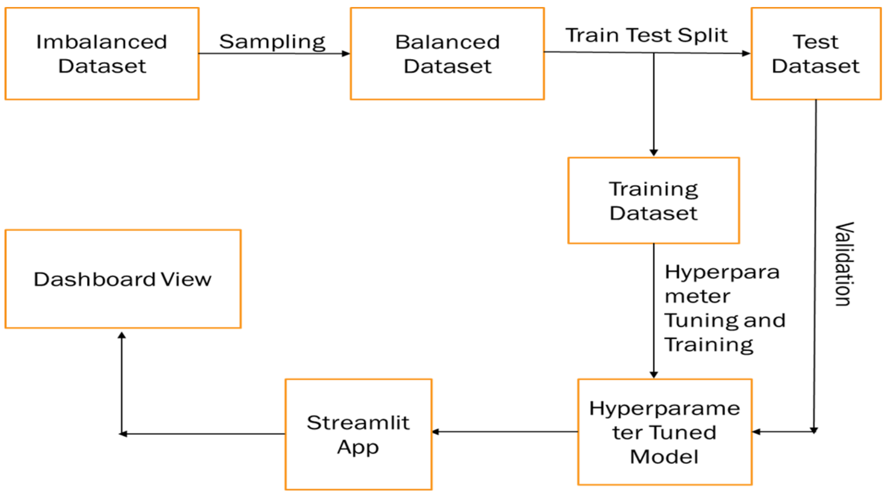


Figure 2.7: UML Design for the flow of Data and Final Model implementation in the Streamlit app and dashboard

2.6 Streamlit and the Final Design

2.6.1 Streamlit

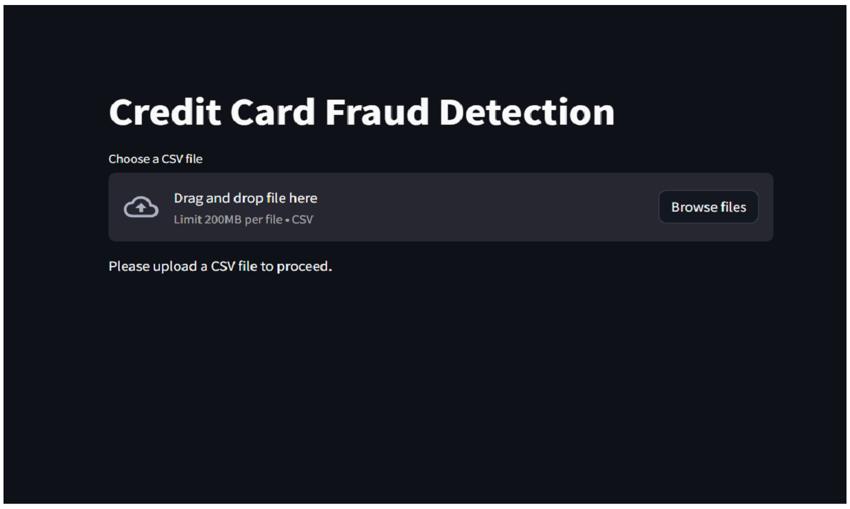
To use the models and implement them in a dashboard for visualisation and comparison, the help of Streamlit was used. Streamlit is an open source Python based framework, using which developers can construct functionally, dynamic data driven applications. It is a good candidate for the development of dashboards which focus on machine learning, data analysis and data science[61]. In the paper by Jain et al.[7], Streamlit has been used for the development of a dashboard for the comparative analysis of different machine learning models of credit card fraud detection, which is similar to the application it has been used for this study. Some of the features of Streamlit are:

- It is easy to use and implement without prior knowledge of HTML, CSS and JavaScript for web development.
- It has real interactivity and dynamic widgets and buttons so that the actions intended can be viewed instantly.
- It seamlessly incorporates and integrates Matplotlib, pandas and Numpy for data visualisation.

2.6.2 Final Design

Let us look at the final design of the dashboard and see some of the features it has along with some of the visualisation. For the purposes of this visualisation, we will take the example of the Random Forest Classifier to see the working of the dashboard app.

1. Loading the data - We come to the first screen where the user is prompted to load the data they want to analyse in a csv file format.



Credit Card Fraud Detection

Choose a CSV file

Please upload a CSV file to proceed.

Figure 2.8: Opening Screen of the Dashboard

2. Selecting the Model for Analysis - The user is then prompted to select the model they wish to view in a drop-down menu. The Random Forest Classification model is chosen for demonstration.

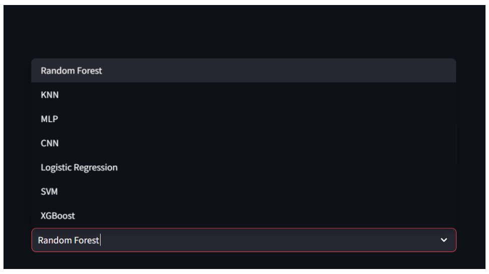
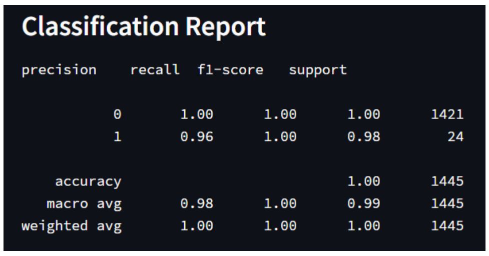


Figure 2.9: Choice of Models from the Dashboard

3. Visualisation and Metrics:

 - Classification Report - The user can view the classification report i.e, Precision, Recall, F1 Score and Accuracy. They can view it on a class-by-class metric, weighted average or the macro average.



Classification Report

```
precision    recall  f1-score   support

           0       1.00      1.00      1.00      1421
           1       0.96      1.00      0.98        24

    accuracy                           1.00      1445
   macro avg       0.98      1.00      0.99      1445
weighted avg       1.00      1.00      1.00      1445
```

Figure 2.10: Classification Report of the model for the data uploaded

 - Confusion Matrix - The User can also view the confusion matrix to see the TP, TN, FP and FN values for better understanding of the classification report.

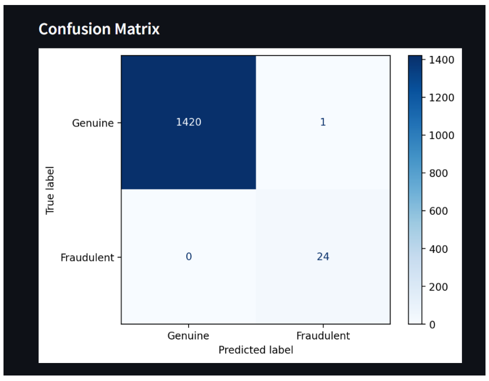


Figure 2.11: Confusion Matrix for the Uploaded Data

- Precision-Recall Curve (PRC) and Receiver Operator Characteristic - Area Under The Curve (ROC-AUC) - The PRC and ROC-AUC can also be viewed for further reference for the performance of the model.

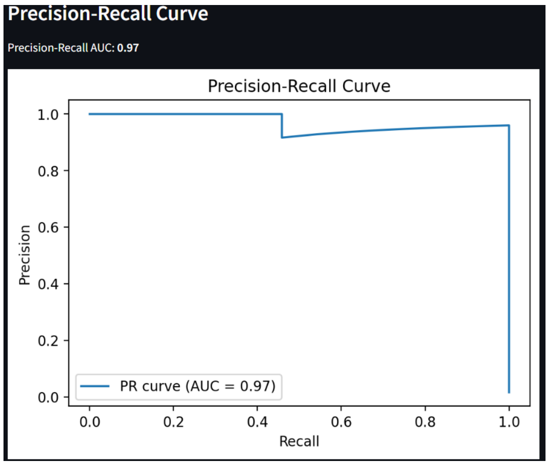


Figure 2.12: Precision-Recall Curve (PRC) of the Model

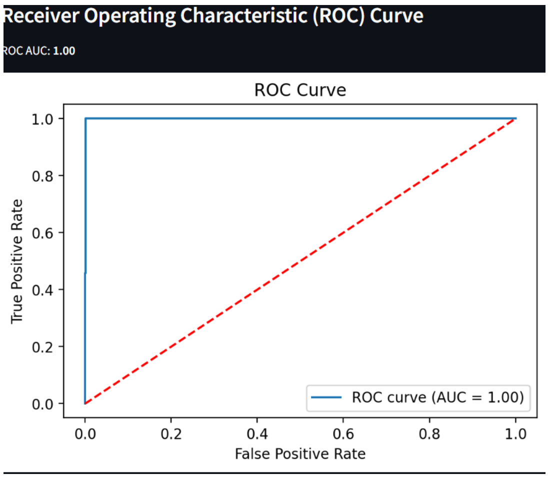
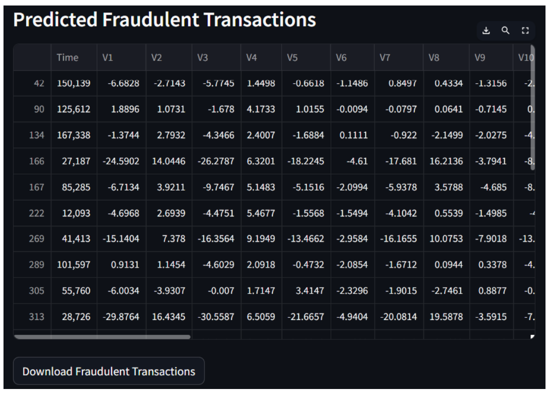


Figure 2.13: Receiver Operator Characteristic - Area Under The Curve (ROC-AUC) of the Model

- Predicted Fraudulent Transactions and Predicted Distribution of Transactions - The User can then view and download the predicted fraudulent transactions as well as view the distribution of the transactions in a bar graph.
- Additional Information - Additional information such as the mean value of Fraudulent transaction and the mean time taken can also be viewed.



Predicted Fraudulent Transactions

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
42	150,139	-6.6828	-2.7143	-5.7745	1.4498	-0.6618	-1.1486	0.8497	0.4334	-1.3156	-2.
90	125,612	1.8896	1.0731	-1.678	4.1733	1.0155	-0.0094	-0.0797	0.0641	-0.7145	0.
134	167,338	-1.3744	2.7932	-4.3466	2.4007	-1.6884	0.1111	-0.922	-2.1499	-2.0275	-4.
166	27,187	-24.5902	14.0446	-26.2787	6.3201	-18.2245	-4.61	-17.681	16.2136	-3.7941	-8.
167	85,285	-6.7134	3.9211	-9.7467	5.1483	-5.1516	-2.0994	-5.9378	3.5788	-4.685	-8.
222	12,093	-4.6968	2.6939	-4.4751	5.4677	-1.5568	-1.5494	-4.1042	0.5539	-1.4985	-
269	41,413	-15.1404	7.378	-16.3564	9.1949	-13.4662	-2.9584	-16.1655	10.0753	-7.9018	-13.
289	101,597	0.9131	1.1454	-4.6029	2.0918	-0.4732	-2.0854	-1.6712	0.0944	0.3378	-4.
305	55,760	-6.0034	-3.9307	-0.007	1.7147	3.4147	-2.3296	-1.9015	-2.7461	0.8877	-0.
313	28,726	-29.8764	16.4345	-30.5587	6.5059	-21.6657	-4.9404	-20.0814	19.5878	-3.5915	-7.

Figure 2.14: Predicted Fraudulent Transactions and a Download Button to save it offline

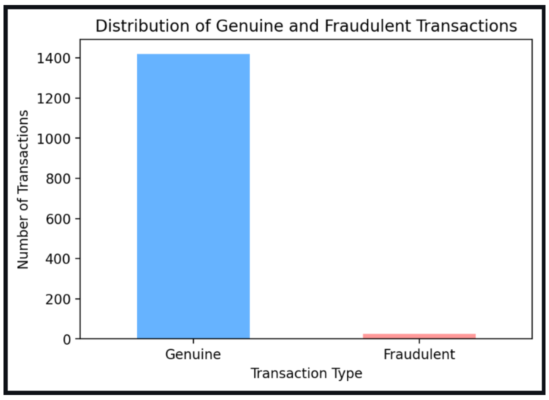
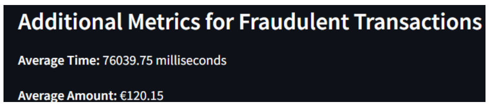


Figure 2.15: Distribution of the predicted genuine and fraudulent transactions



Additional Metrics for Fraudulent Transactions

Average Time: 76039.75 milliseconds

Average Amount: €120.15

Figure 2.16: Additional information on the data

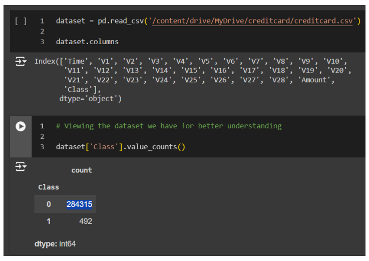
Chapter 3

Implementation

3.1 Reading the Dataset and Performing Exploratory Data Analysis

There has been extensive use of Google Colab, particularly during the early stages of development and training. The primary library used to load the data is the Pandas library. The function `pd.read_csv()` has been used to load the dataset since the data is in a csv file format. To start the process of EDA these steps were taken:

- Loading the data - The data is first loaded into a dataset known as 'dataset'.
- Class distribution - The class count or distribution is then viewed using the `dataset['Class'].value_counts()` function. We see that class '0' has a count of 284315 records and class '1' has a count of 492 records.



```
[ ]  dataset = pd.read_csv('/content/drive/MyDrive/creditcard/creditcard.csv')

     dataset.columns
```

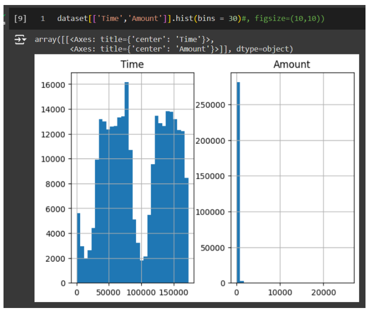
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

```
# Viewing the dataset we have for better understanding

dataset['Class'].value_counts()
```

```
          count
Class
    0    284315
    1       492

dtype: int64
```

Figure 3.1: Code Snippet for the loading of the data and the count of the classification types

- Distribution of data in the 'Amount' and 'Time' - The dataset[['Amount','Time']].hist(bins=30) to visualise the data distribution of these two columns. As discussed before, the 'Amount' column seems to be right skewed.

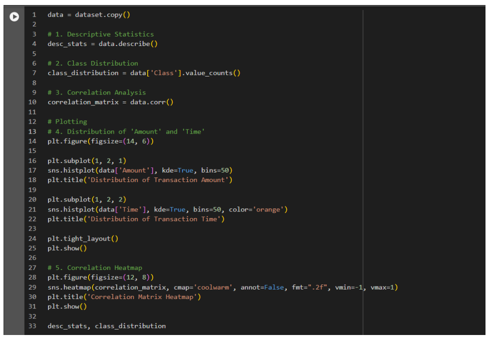


```
[9]  dataset[['Time','Amount']].hist(bins = 30)#, figsize=(10,10))
```

```
array([[<Axes: title={'center': 'Time'}>,
        <Axes: title={'center': 'Amount'}>]], dtype=object)
```

Figure 3.2: Amount and time Data Distribution in the Dataset

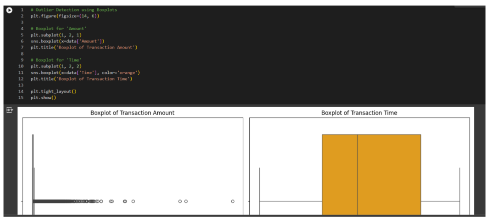
- Correlation Matrix and Data description - the pandas function .describe() is applied to the dataset to find relevant statistical information on the data. Then sns.hisplot() from seaborn library was used to once again to visualise the data distribution. data.cor() function was used to find out the correlation matrix and then sns.heatmap() was used to print it using matplotlib library.



```python
data = dataset.copy()

# 1. Descriptive Statistics
desc_stats = data.describe()

# 2. Class Distribution
class_distribution = data['Class'].value_counts()

# 3. Correlation Analysis
correlation_matrix = data.corr()

# Plotting
# 4. Distribution of 'Amount' and 'Time'
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.histplot(data['Amount'], kde=True, bins=50)
plt.title('Distribution of Transaction Amount')

plt.subplot(1, 2, 2)
sns.histplot(data['Time'], kde=True, bins=50, color='orange')
plt.title('Distribution of Transaction Time')

plt.tight_layout()
plt.show()

# 5. Correlation Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False, fmt=".2f", vmin=-1, vmax=1)
plt.title('Correlation Matrix Heatmap')
plt.show()

desc_stats, class_distribution
```

Figure 3.3: Code snippet for the correlation matrix and distribution of data graphs

- Outlier Detection - The sns.boxplot() function was used to detect any outliers from the 'AMount' and 'Time' column for further analysis.



```python
# Outlier Detection using Boxplots
plt.figure(figsize=(14, 6))

# Boxplot for 'Amount'
plt.subplot(1, 2, 1)
sns.boxplot(x=data['Amount'])
plt.title('Boxplot of Transaction Amount')

# Boxplot for 'Time'
plt.subplot(1, 2, 2)
sns.boxplot(x=data['Time'], color='orange')
plt.title('Boxplot of Transaction Time')

plt.tight_layout()
plt.show()
```

Figure 3.4: Code Snippet for the outlier detection from the Amount and Tome column

It was decided after viewing and studying the various plots that the data seemed to be heavily unbalanced, which might cause problems and issues while model training and needs to be resampled for better results. It was also found that the values in the 'Amount' and 'Time' columns were heavily skewed and uneven and would require normalisation techniques such as the Standard Normalisation or Minimum Maximum Normalisation to ensure better model performance.

3.2 Model Training for the Supervised Machine Learning and Deep Learning Models

In this part of the discussion, the ways and methods used for the training, testing and model evaluation of the supervised machine learning models, namely Random Forest Classifier, Support Vector Machine, Extreme Gradient Boosting, Logistic Regression, and K Nearest-Neighbour Classifier, have been discussed. The steps that were followed are as follows:

- The dataset was first loaded using Pandas library function pd.read_csv() into a Pandas Dataframe.
- The dataset was then fed through a pipeline to perform hybrid sampling which is a combination of oversampling and undersampling to get a new dataset suitable for training and testing.

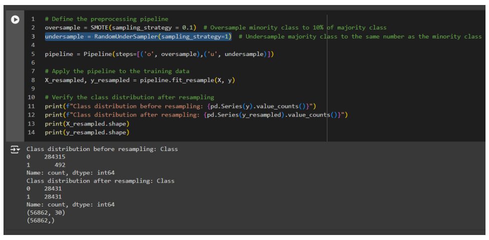
- This new dataset is then normalised using a StandardScaler() function to achieve Standard Normalisation for ease of handling the data.
- The Dataset is then split into the training data and testing data for model training and evaluation.
- The models are then hyperparameter tuned using GridSearchCV() from sklearn in addition to cross-validation to get the best fit models.
- The models are then evaluated using a range of metrics and values to see their performance.

3.2.1 Resampling the Data and Normalising the data

It was decided to perform a hybrid resampling technique, which involves SMOTE() oversampling and RandomUnderSampler() from the imblearn of Python. However, the initial attempt was to perform only SMOTE oversampling and then train the models to that resampled data. There were issues with this approach.

- The dataset was too large and models like SVM and Logistic were unable to perform properly even though T4 GPU capabilities were activated for the Google Notebook. This is due to sklearn's inability to use GPU acceleration for model training unlike TensorFlow and PyTorch.
- Even for the models which were trained using this approach, the model evaluation result was not satisfactory and hence not ideal.

The use of imblearn.pipeline was important to streamline the entire effort. The SMOTE() layer was the first in the pipeline were the minority class was upsampled to 10% of the majority class using `oversample = SMOTE(sampling_strategy = 0.1)` and then Random UnderSampler was used to downsample the data to match the majoirty and minority class count using undersample = RandomUnderSampler(sampling_strategy=1). This is then finalised by using the `X_resampled, y_resampled = pipeline.fit_resample(X, y)` function to finalise the changes. The count of the classes is as follows: Class '0' = Class '1' = 28431.



```
# Define the preprocessing pipeline
oversample = SMOTE(sampling_strategy = 0.1)  # Oversample minority class to 10% of majority class
undersample = RandomUnderSampler(sampling_strategy=1)  # Undersample majority class to the same number as the minority class

pipeline = Pipeline(steps=[('o', oversample),('u', undersample)])

# Apply the pipeline to the training data
X_resampled, y_resampled = pipeline.fit_resample(X, y)

# Verify the class distribution after resampling
print(f"Class distribution before resampling: {pd.Series(y).value_counts()}")
print(f"Class distribution after resampling: {pd.Series(y_resampled).value_counts()}")
print(X_resampled.shape)
print(y_resampled.shape)
```

```
Class distribution before resampling: Class
0    284315
1       492
Name: count, dtype: int64
Class distribution after resampling: Class
0    28431
1    28431
Name: count, dtype: int64
(56862, 30)
(56862,)
```

Figure 3.5: Code snippet to show the resampling technique and the final data count

The dataset is then normalised using the StandardScaler() module using the function `X_resampled = scaler.fit_transform(X_resampled)`. This ensures that the data have been normalised in the standard normal form.

3.2.2 Model Training and Hyperparameter Tuning for Supervised Learning Models

For the purpose of the supervised learning implementation, the scikit-learn() of Python was used to implement the following models: Random Forest Classifier, Support Vector

Machine, Logistic Regression, and K-Nearest Neighbours. For the XGBoost algorithm, the xgboost() library was imported in Python.

Splitting the data into the Training data and Testing Data

The subsequent dataset is split into training and testing data. The ratio of training to testing is set at 80:20 where 80% of the data goes to training the dataset and 20% goes to performing preliminary tests. This is done by implementing `sklearn.model-selection.train_test_split()` function as follows:

```
X_train_resampled, X_test_resampled, y_train_resampled, y_test_resampled =
train_test_split(X_resampled, y_resampled, test_size=0.2, shuffle = True,
random_state=42)
```

Here, the shuffle is kept true to shuffle the data so that model trains rather than memorises the patterns, and random_state is kept at 42 for reproducibility.

Importing the Models and Model Implementation

1. Random Forest - The Random Forest model is found in the sklearn.ensemble.RandomForestClassifier() function in Python. By calling it and saving it in a model variable, the model will then be hyperparameter-tuned to find the best-fit model for the dataset. which will be shown as the following:

   `rf = RandomForestClassifier(random_state=42).`

   Here, the random_state is kept at 42 for the reproducibility of the results elsewhere.

2. Support Vector Machine (SVM) - The SVM model is implemented in this study using the sklearn.svm.SVC module. We again repeat the procedure and store the model in a variable and use it in GridSearchCV to hyperparameterize the model using the following:

   `svm = SVC(random_state = 42)`

3. K-Nearest Neighbours Classifier (KNN) - To implement the KNN classifier into the code, the skleanr.neighbors.kNeighborsClassifier model is used from scikit_learn. The implementation of the model is as follows:

   `knn= KNeighborsClassifier()`

4. Logistic Regression CLassifier - The implementation of this methods is done on Python by importing the sklearn.linear_model.LogisticRegression model. The model was saved in a variable for further use in hyperparameter tuning in the following code:

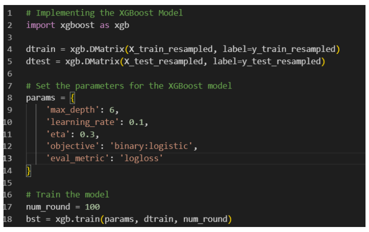
   `logistic = LogisticRegression()`

5. Extreme Gradient Boosting (XGBoost) - To use xgboost in Python, the xgboost library was first imported. The data needs a slight bit of modification for the libraries use case. The data is first converted into a dmatric using the following:

   `dtrain = xgb.DMatrix(X_train_resampled, label=y_train_resampled)`

   `dtest = xgb.DMatrix(X_test_resampled, label=y_test_resampled)`

   After this, the hyperparameters are set and the model is trained. Due to its use of GPU accelerations, the training time of this model is quite low and responsive.

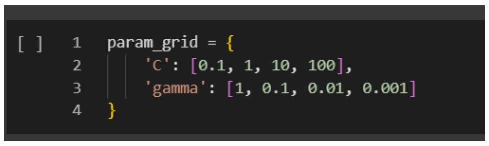


```python
# Implementing the XGBoost Model
import xgboost as xgb

dtrain = xgb.DMatrix(X_train_resampled, label=y_train_resampled)
dtest = xgb.DMatrix(X_test_resampled, label=y_test_resampled)

# Set the parameters for the XGBoost model
params = {
    'max_depth': 6,
    'learning_rate': 0.1,
    'eta': 0.3,
    'objective': 'binary:logistic',
    'eval_metric': 'logloss'
}

# Train the model
num_round = 100
bst = xgb.train(params, dtrain, num_round)
```

Figure 3.6: Model Training for the XGBoost Model

Hyperparameter Tuning using GridSearchCV and KFold() cross-validation

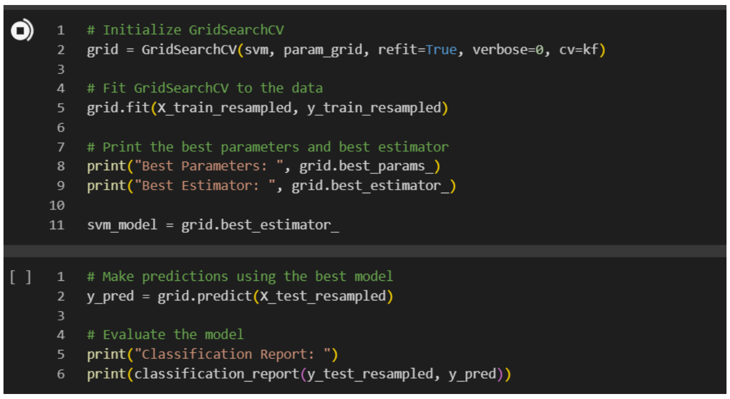
The method chosen to perform hyperparameter tuning was GridSearchCv in accordance with the kFold() cross-validation technique. It is important to use a cross-validation technique like KFold() or any other to prevent the model for overfitting and hence, underperforming during preliminary testing. GridSearchCV is used by importing the `sklearn.model_selection.GridSearchCV`. Before this process, the kfold cross-validation technique is implemented by importing the sklearn.model_selection.KFold and then setting the number of splits to 3. The code for this portion is as follows:
`kf = KFold(n_splits=3, shuffle=True, random_state=42)`
The parameters of the model are saved in a dictionary variable called a param_grid which is then gonna be fed into the GridSearchCV function to perform the hyperparameter tuning on the range specified in parma_grid.



```python
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001]
}
```

Figure 3.7: Parameter Grid for Hyperparameter Tuning in SVM taken as an example

Then the model is fed through a variable where the GridSearchCV() function is used to start the hyperparameter tuning sequentially and get the optimal model. The optimal model is then viewed by calling the function `grid.best_params_` and then the model is saved using the `grid.best_estimator_`.
Then the models are trained again using the best-trained model and the preliminary evaluations are done on that model with the testing dataset that was created.
The model prediction is then saved in a Numpy array using the `model.predict(X_test_resampled, y_test_resampled)` and stored for further evaluation.
GridSearchCv has been used for all the scikit-learn supervised learning models. The n_splits value of is kept at 3 due to the processing constraints on the Google Colab notebook that was used to perform the analysis. The first attempt was to put the value at `n_splits = 10` and `n_splits = 5`, but the running time for each of the models surpassed the allowed time with the high compute environment and hence it was decided to use the n_splits to 3.


```
# Initialize GridSearchCV
grid = GridSearchCV(svm, param_grid, refit=True, verbose=0, cv=kf)

# Fit GridSearchCV to the data
grid.fit(X_train_resampled, y_train_resampled)

# Print the best parameters and best estimator
print("Best Parameters: ", grid.best_params_)
print("Best Estimator: ", grid.best_estimator_)

svm_model = grid.best_estimator_
```

```
# Make predictions using the best model
y_pred = grid.predict(X_test_resampled)

# Evaluate the model
print("Classification Report: ")
print(classification_report(y_test_resampled, y_pred))
```

Figure 3.8: Example of Hyperparameter tuning and Model Prediction on the Test Dataset split using SVM

3.2.3 Model Training and Hyperparameter Tuning in the Deep Learning Models

For implementing deep learning models like the Multilayer Perceptron (MLP) and the Convolutional Neural Network 1 Dimensional (1D CNN), the use of TensorFlow and Keras API due to the familiarity of these APIs as compared to PyTorch. The data preprocessing and splitting into the training and testing phase is similar to that of the supervised machine learning algorithms. For hyperparameter tuning the models, the use of keras_tuner.RandomSearch was used to find the best performing model and save that model for preliminary evaluation. The cross-validation was done during RandomSearch step and the model's tendency to overfit was also visualised with the help of the validation loss and validation accuracy curves and to try and find a convergence of the values to ensure that the model is performing as expected.

Model Implementation and Building

1. **Multilayer Perceptron (MLP) -** The description of building the MLP model is given as below:

   The First Dense Layer:

   - `units=hp.Int('dense_units_1', min_value=64, max_value=256, step=64)`: This code defines a `hyperparameter dense_units_1` which will tune the number of neurons in the first hidden layer. It will try and input values between 64 and 256 neurons with a step of 64.
   - `activation='relu'`: ReLU stands for Rectified Linear Unit (ReLU). It is a common activation function used in deep learning.
   - `input_shape=(X_train.shape[1],)`: The shape is set to the number of features in the training data which is X_train.

   The First Dropout Layer

   - `rate=hp.Float('dropout_1', min_value=0.1, max_value=0.5, step=0.1)`: A dropout layer is added to avoid overfitting. The rate is set from 0.1 to 0.5 to be chosen during hyerparameter tuning.

   The Second Dense Layer

- This layer is also similar in function to the first dense layer, but with `units=hp.Int('dense_units_2', min_value=64, max_value=256, step=64)`

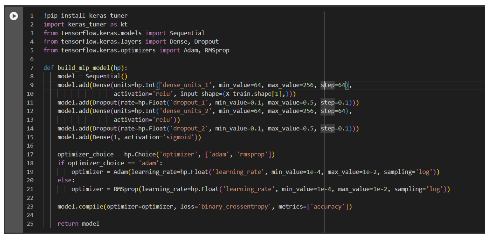
The Second Dropout Layer

- This is the second dropout layer where the dropout rate is varied in a range from 0.1 to 0.5

The Output Layer

- This is the Output Layer. It has a single output unit because the task is to perform binary classification binary classification. The activation function used is the sigmoid function to determine whether the output is 0 or 1.

The choice of optimiser is between the 'adam', and 'rmsprop' and the adaptive learning rate is kept from 1e-4 to 1e-2. The model is then compiled using the `model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])`, where the loss is kept at binary_crossentropy and the metric for evaluation is the accuracy of the model.



```python
!pip install keras-tuner
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop

def build_mlp_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('dense_units_1', min_value=64, max_value=256, step=64),
                    activation='relu', input_shape=(X_train.shape[1],)))
    model.add(Dropout(rate=hp.Float('dropout_1', min_value=0.1, max_value=0.5, step=0.1)))
    model.add(Dense(units=hp.Int('dense_units_2', min_value=64, max_value=256, step=64),
                    activation='relu'))
    model.add(Dropout(rate=hp.Float('dropout_2', min_value=0.1, max_value=0.5, step=0.1)))
    model.add(Dense(1, activation='sigmoid'))

    optimizer_choice = hp.Choice('optimizer', ['adam', 'rmsprop'])
    if optimizer_choice == 'adam':
        optimizer = Adam(learning_rate=hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='log'))
    else:
        optimizer = RMSprop(learning_rate=hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='log'))

    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

Figure 3.9: The Building of the Multilayer Perceptron

2. **1-Dimensional Convolutional Neural Network (CNN 1D) -** The description of building the 1D CNN model is given as follows:

   Regularisation:

   - The regularisation is set to either l1 or l2 and then decided upon after hyperparameter tuning.
     `regularizer_type = hp.Choice('regularizer', ['l1', 'l2'])`

   First Convolutional Layer:

   - Filters: The number of filters, which can be tuned between 32 and 128.
   - Kernel Size: Can be hyperparameter tuned using `keras_tuner.RandomSearch()`. The kernel size can be chosen between 3 and 7.

   BatchNormalization: Batch normalisation is added to counteract overfitting.

   Dropout Layer 1: The dropout rate is chosen between 0.25 to 0.5 and can be hyperparameter tuned using `keras_tuner.RandomSearch()`

   Second Convolutional Layer:

   - Filters: The number of filters, which can be tuned between 32 and 128.
   - Kernel Size: Can be hyperparameter tuned using `keras_tuner.RandomSearch()`. The kernel size can be chosen between 3 and 7.

   BatchNormalization: Batch normalisation is added to counteract overfitting.

   Dropout Layer 2: The dropout rate is chosen between 0.25 to 0.5 and can be hyperparamter tuned using `keras_tuner.RandomSearch()`

   Flatten Layer: This layer is used to flatten the output from convolutional layers to and then fed the dense layer.

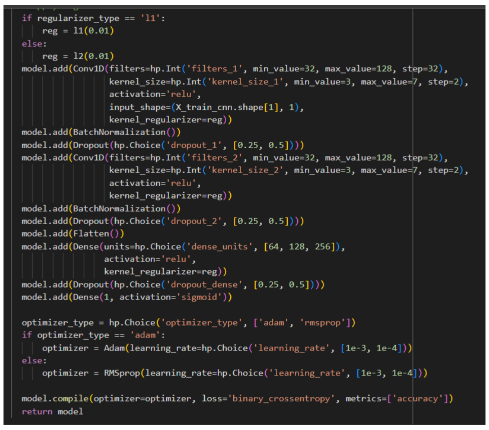
   Dense Layer:

   - Units: This will be a tunable number of units, 64, 128, and 256 for the fully connected layer.
   - Activation: ReLU

   Dropout Layer 3: The dropout rate is chosen between 0.25 to 0.5 and can be hyperparameter tuned using `keras_tuner.RandomSearch()`.

   The Output Layer: The output layer is a single output unit used for binary classification.

   The choice of optimiser is between the 'adam', and 'rmsprop' and the adaptive learning rate is kept from 1e-4 to 1e-2. The model is then compiled using the `model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])`, where the loss is kept at binary_crossentropy and the metric for evaluation is the accuracy of the model.

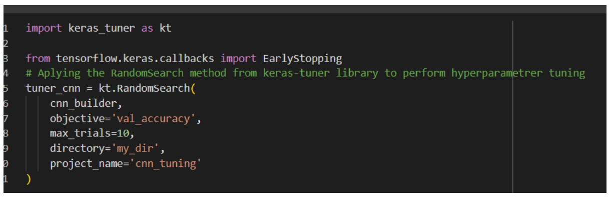


```python
if regularizer_type == 'l1':
    reg = l1(0.01)
else:
    reg = l2(0.01)
model.add(Conv1D(filters=hp.Int('filters_1', min_value=32, max_value=128, step=32),
                 kernel_size=hp.Int('kernel_size_1', min_value=3, max_value=7, step=2),
                 activation='relu',
                 input_shape=(X_train_cnn.shape[1], 1),
                 kernel_regularizer=reg))
model.add(BatchNormalization())
model.add(Dropout(hp.Choice('dropout_1', [0.25, 0.5])))
model.add(Conv1D(filters=hp.Int('filters_2', min_value=32, max_value=128, step=32),
                 kernel_size=hp.Int('kernel_size_2', min_value=3, max_value=7, step=2),
                 activation='relu',
                 kernel_regularizer=reg))
model.add(BatchNormalization())
model.add(Dropout(hp.Choice('dropout_2', [0.25, 0.5])))
model.add(Flatten())
model.add(Dense(units=hp.Choice('dense_units', [64, 128, 256]),
                activation='relu',
                kernel_regularizer=reg))
model.add(Dropout(hp.Choice('dropout_dense', [0.25, 0.5])))
model.add(Dense(1, activation='sigmoid'))

optimizer_type = hp.Choice('optimizer_type', ['adam', 'rmsprop'])
if optimizer_type == 'adam':
    optimizer = Adam(learning_rate=hp.Choice('learning_rate', [1e-3, 1e-4]))
else:
    optimizer = RMSprop(learning_rate=hp.Choice('learning_rate', [1e-3, 1e-4]))

model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
return model
```

Figure 3.10: Building the CNN Model

Hyperparameter Tuning using keras_tuner.RandomSearch()

The models will be hyperparameter tuned using RandomSearch(), the epochs will be set to 10. There will be Callback function for early stopping to prevent the values for overfitting. The tuning will be done the following way:
`tuner_cnn.search(X_train_cnn, y_train, epochs=10, validation_split=0.2, callbacks=[EarlyStopping(monitor='val_loss', patience=3)])`.
The best model is then chosen by the following code and stored. `best_hps_cnn = tuner_cnn.get_best_hyperparameters (num_trials=1)[0]`.
The predictions are stored in a numpy array after doing some formatting to the data as the CNN and MLP outputs as a probabilistic output and then the outputs are rounded off.
`y_pred = best_model_cnn.predict(X_test)` and `y_pred = (y_pred > 0.5).astype(int)`



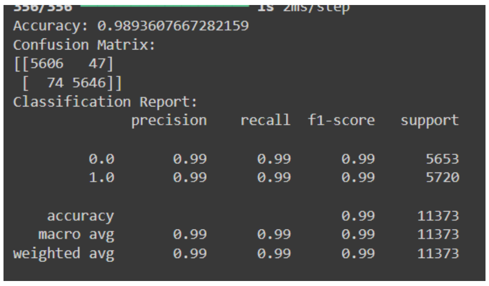
```python
import keras_tuner as kt

from tensorflow.keras.callbacks import EarlyStopping
# Aplying the RandomSearch method from keras-tuner library to perform hyperparametrer tuning
tuner_cnn = kt.RandomSearch(
    cnn_builder,
    objective='val_accuracy',
    max_trials=10,
    directory='my_dir',
    project_name='cnn_tuning'
)
```

Figure 3.11: Hyperparameter Tuning Using keras_tuner.RandomSearch()

3.3 Model Evaluation Metrics

The model evaluations of the models are listed below:

1. Classification Report - The classification report is implemented by calling the sklearn.metrics.classif function. The implementation is as follows:



```
356/356 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Accuracy: 0.9893607667282159
Confusion Matrix:
[[5606   47]
 [  74 5646]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99      5653
         1.0       0.99      0.99      0.99      5720

    accuracy                           0.99     11373
   macro avg       0.99      0.99      0.99     11373
weighted avg       0.99      0.99      0.99     11373
```

Figure 3.12: Example of a Classification Report

2. Confusion Matrix - The confusion matrix is implemented using the sklearn.metrics.confusion_matri function as follows:

   `conf_matrix = confusion_matrix(y_test, y_pred)`. The confusion matrix is then visualised using the sklearn.metrics.ConfusionMatrixDisplay() in the following way:

   `disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.unique(y_resampled))`

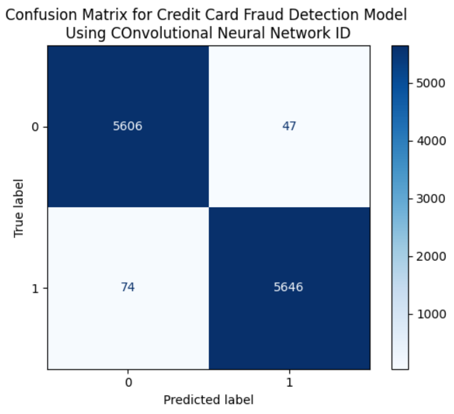
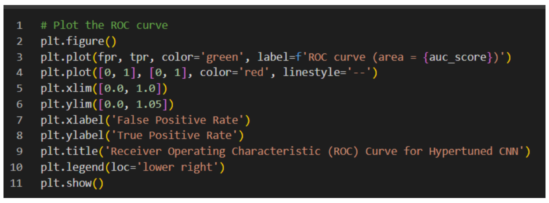


Figure 3.13: Confusion Matrix example

3. ROC-AUC Curve - The ROC-AUC curve is implemented into the code using the sklearn.metrics.roc_curve() function where the true positive ration (tpr) and false positive ratio (for) is saved alongside the threshold value. These values are then plotted using matplotlib to find out the ROC-AUC curve.
   `fpr, tpr, thresholds = roc_curve(y_test, y_pred)`

The the auc score is calculated by using the skelearn.metrics.auc_score() function in the following.
`auc_score = roc_auc_score(y_test, y_pred)`



```
# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='green', label=f'ROC curve (area = {auc_score})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Hypertuned CNN')
plt.legend(loc='lower right')
plt.show()
```

Figure 3.14: Code Snippet for plotting the ROC-AUC Curve

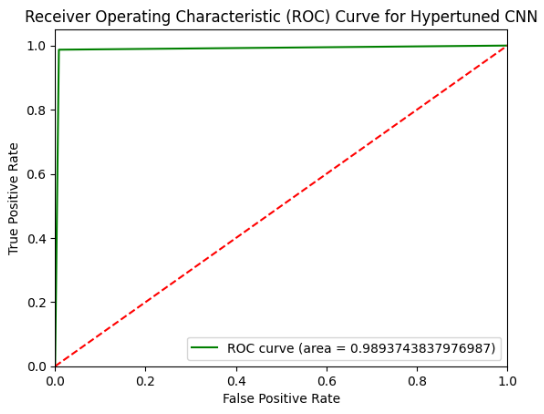
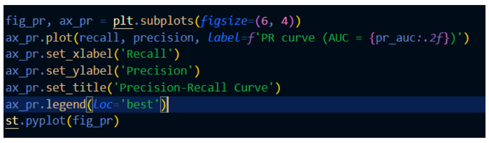


Figure 3.15: CNN ROC=AUC Curve taken as an example from the preliminary testing

4. Precision-Recall Curve - The precision-recall curve was implemented only in the streamlit app for the dahsboard view. It was implemnted by using the sklearn.metrics.precison_reca function in the fowlloing way:
`precision, recall, thresholds_pr = precision_recall_curve(y, prediction_probs)`
`pr_auc = auc(recall, precision)`
The precision recall score are first found out and then the auc is caluclated. The plotting is implemented using matplotlib.
`auc_score = roc_auc_score(y_test, y_pred)`



```
fig_pr, ax_pr = plt.subplots(figsize=(6, 4))
ax_pr.plot(recall, precision, label=f'PR curve (AUC = {pr_auc:.2f})')
ax_pr.set_xlabel('Recall')
ax_pr.set_ylabel('Precision')
ax_pr.set_title('Precision-Recall Curve')
ax_pr.legend(loc='best')
st.pyplot(fig_pr)
```

Figure 3.16: Code Snippet for the Precision-Recall Curve

5. Validation Accuracy and Validation Loss - An extra step is taken in the deep learning models to check for the model's performance. The training accuracy and loss is often compared with the validation accuracy and loss. If there is a convergence of values, it is said that the model has been trained well. If the model diverges after a certain point then the model is said to be overfit.

   This was done by saving the training history of the model and then plotting the validation loss and validation accuracy to its training analogues.

The training accuracy and loss are found out by `model.history['accuracy']` and `model.history['loss']` respectively.
The validation accuracy and loss are found out by the `model.history['val_accuracy']` and `model.history['val_loss]` respectively.

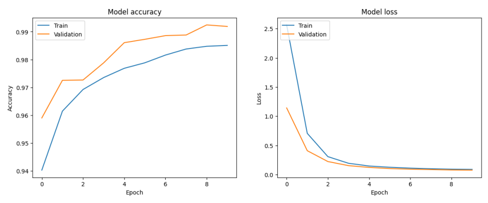


Figure 3.17: Training loss to Validation loss and Training Accuracy to Validation Accuracy for the CNN Model

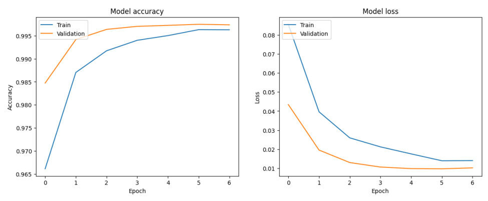


Figure 3.18: Training loss to Validation loss and Training Accuracy to Validation Accuracy for the MLP Model

3.4 Streamlit Dashboard

The Streamlit dashboard code is sub-divided into several parts. The first part of the code for review would be the preprocessing and loading the data in real time.

Uploading the Data

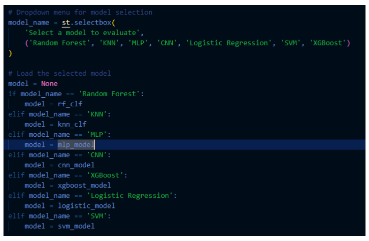
The data is uploaded into the dashboard by using the inbuilt function of Streamlit called st.file_upload(). The implementation is done as follows:
`uploaded_file = st.file_uploader("Choose a CSV file", type="csv")`
Here, the file type has been specified to be a csv file for the data preparation and preprocessing steps.

Selecting the Model For Evaluation

The models are then listed and presented to the user in the form of a drop-down button. The code for the drop-down button is implemented by calling the st.selectbox() function. The selected option is then fed through an if-else statement to select the model and then load the model.

Preprocessing the data

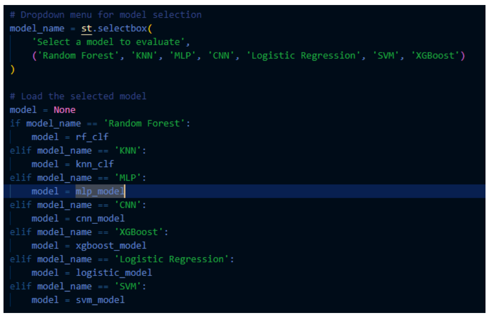
Two user defined functions are used to prepocess the data for the model tom perform predictions on namely load_and_preprocess_data() and create_dmatrix(). The first function is used to preprocess and normalise the data according to the supervised m,achin elearn ing models and deep learning models. For the deep learning models, it refactors the data



```python
# Dropdown menu for model selection
model_name = st.selectbox(
    'Select a model to evaluate',
    ('Random Forest', 'KNN', 'MLP', 'CNN', 'Logistic Regression', 'SVM', 'XGBoost')
)

# Load the selected model
model = None
if model_name == 'Random Forest':
    model = rf_clf
elif model_name == 'KNN':
    model = knn_clf
elif model_name == 'MLP':
    model = mlp_model
elif model_name == 'CNN':
    model = cnn_model
elif model_name == 'XGBoost':
    model = xgboost_model
elif model_name == 'Logistic Regression':
    model = logistic_model
elif model_name == 'SVM':
    model = svm_model
```

Figure 3.19: Selecting and Setting up the selected model

to numpy arrays in the CNN as `X_cnn = X_mlp.reshape(-1, X.shape[1], 1)`. Here the normal X data is just converted into a float32 version for mlp data.

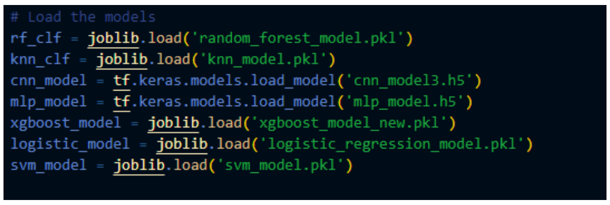


```python
# Dropdown menu for model selection
model_name = st.selectbox(
    'Select a model to evaluate',
    ('Random Forest', 'KNN', 'MLP', 'CNN', 'Logistic Regression', 'SVM', 'XGBoost')
)

# Load the selected model
model = None
if model_name == 'Random Forest':
    model = rf_clf
elif model_name == 'KNN':
    model = knn_clf
elif model_name == 'MLP':
    model = mlp_model
elif model_name == 'CNN':
    model = cnn_model
elif model_name == 'XGBoost':
    model = xgboost_model
elif model_name == 'Logistic Regression':
    model = logistic_model
elif model_name == 'SVM':
    model = svm_model
```

Figure 3.20: Data preprocessing for the different models

Loading the Models

The models are loaded using the joblib[62] library of Python for supervised machine learning models and tf.keras.load_model() function from the Keras API for the deep learning model. The following code snippet shows the process of loading the individual models.



```python
# Load the models
rf_clf = joblib.load('random_forest_model.pkl')
knn_clf = joblib.load('knn_model.pkl')
cnn_model = tf.keras.models.load_model('cnn_model3.h5')
mlp_model = tf.keras.models.load_model('mlp_model.h5')
xgboost_model = joblib.load('xgboost_model_new.pkl')
logistic_model = joblib.load('logistic_regression_model.pkl')
svm_model = joblib.load('svm_model.pkl')
```

Figure 3.21: Loading the models for use

Model Predictions

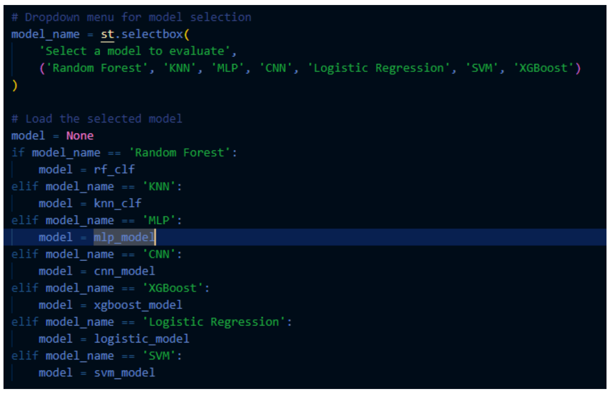
The models are then used for prediction relying on if-else statements depending on which model is loaded. For the supervised machine learning models the output is calculated as follows:
`prediction_probs = model.decision_function(X_scaled)`
For MLP and CNN, the model predictions are calculated as follows:

`prediction_probs = model.predict(X_mlp).ravel()`
`predictions = (prediction_probs > 0.5).astype("int32")` for MLP.

`prediction_probs = model.predict(X_cnn).ravel()`
`predictions = (prediction_probs > 0.5).astype("int32")` for CNN.

For the XGBoost model, the following is the code for predictions.

```
prediction_probs = model.predict(test)
predictions = [round(value) for value in prediction_probs]
```

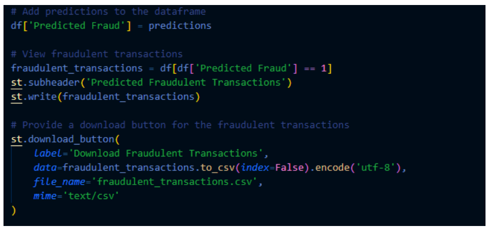


```
# Dropdown menu for model selection
model_name = st.selectbox(
    'Select a model to evaluate',
    ('Random Forest', 'KNN', 'MLP', 'CNN', 'Logistic Regression', 'SVM', 'XGBoost')
)

# Load the selected model
model = None
if model_name == 'Random Forest':
    model = rf_clf
elif model_name == 'KNN':
    model = knn_clf
elif model_name == 'MLP':
    model = mlp_model
elif model_name == 'CNN':
    model = cnn_model
elif model_name == 'XGBoost':
    model = xgboost_model
elif model_name == 'Logistic Regression':
    model = logistic_model
elif model_name == 'SVM':
    model = svm_model
```

Figure 3.22: Code Snippet for finding out the predicted value from each model

Download Button for the Predicted Data

The predicted fraudulent data is then extracted from the original dataset and a new data set with a prediction column is then available for download to the user. the column is added by using the following command and with the help of Pandas `df['Predicted Fraud'] = predictions`.



```
# Add predictions to the dataframe
df['Predicted Fraud'] = predictions

# View fraudulent transactions
fraudulent_transactions = df[df['Predicted Fraud'] == 1]
st.subheader('Predicted Fraudulent Transactions')
st.write(fraudulent_transactions)

# Provide a download button for the fraudulent transactions
st.download_button(
    label='Download Fraudulent Transactions',
    data=fraudulent_transactions.to_csv(index=False).encode('utf-8'),
    file_name='fraudulent_transactions.csv',
    mime='text/csv'
)
```

Figure 3.23: Code snippet for the download button for the predicted fraudulent transactions

Chapter 4

Testing and Evaluation

4.1 Creation of the Test Datasets for Final Evaluation

For the evaluation of the model performance, three test credit card files were created using ChatGPT and Google Gemini. These were created by taking the original credit card file used for training the datasets and then fed into the language models for them to randomly generate the required file type with user given specification on the total number of genuine and fraudulent transaction to be present.

4.2 Steps For the Testing and Evaluation

1. The dataset is first uploaded into a user-defined code B in the appendix, for plotting the models precision-recall curve, the ROC-AUC curve and the classification report.
2. The graphs will then be displayed in a single graph with different colors for further analysis.
3. The classification report will be plotted in a bar graph for all the different models.
4. The dataset is then put in another user-defined code A in the appendix, Where the classification reports is presented in a tabular format for viewing.

4.3 The First Test

4.3.1 Description of the test dataset

The first dataset consists of 2100 transactions in total. Out of these, 2000 transactions are genuine while the rest of the 100 are fraudulent.This was done by putting the original dataset into ChatGPT and prompting it to generate the specified transactions randomly.

4.3.2 Classification Report

In table 4.1, the values for the classification report for each model has been listed and tabulated. With a quick glance, it is evident that the MLP and CNN have the best precision, recall and f1-score out of all the models. The overall accuracy of these two models is also the highest of the bunch. It is then followed by the Random Forest Classifier whose recall score is at 80% and the precision score is at 98.765% suggesting that the fraudulent transactions were favourably identified but not to a extent while the genuine transactions where identified accurately. The issue comes with the XGBoost model as even though during the preliminary evaluation the model had favourable outcomes, the testing phase shows a poor precision score showing that it is not able to differentiate properly between a genuine and fraudulent transaction but the recall score suggests that the all the fraudulent transactions were flagged correctly.

Model	Precision	Recall	F1 Score	Accuracy
Random Forest	98.765%	80.000%	88.398%	99.000%
KNN	73.529%	100.000%	84.746%	98.286%
MLP	98.958%	95.000%	96.939%	99.714%
CNN	96.875%	93.000%	94.898%	99.524%
Logistic Regression	79.817%	87.000%	83.254%	98.333%
SVM	73.276%	85.000%	78.704%	97.810%
XGBoost	7.899%	100.000%	14.641%	44.476%

Table 4.1: Classification Report Analysis for test 1

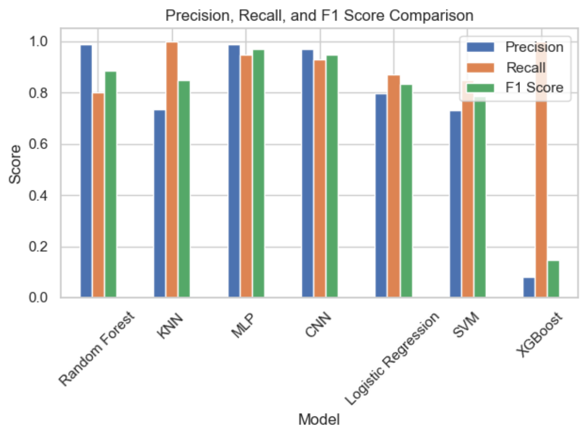


Figure 4.1: Visual Representation of the Classification Report of the models in test 1

4.3.3 ROC-AUC Curve and the Precision-Recall Curve Analysis

The ROC-AUC curve and Precision-Recall Curve have been put in for further reference. With these visualisations, it is evident that the MLP, CNN and Random Forest algorithms have the best performance respectively, closely followed by the Decision Tree, Logistic Regression and XGBoost.

4.4 The Second Test

Thye second test file has 3000 genuine transactions and 60 fraudulent transactions. The test file was generated using ChatGPT. It generated 3060 random data points using the original dataset.

4.4.1 Classification Report Analysis

The classification report analysis has been tabulated in the table. From this it is observed that the MLP model has the best overall result in terms of F1 score and recall score, followed by the Random Forest model with a perfect precision score and then the CNN

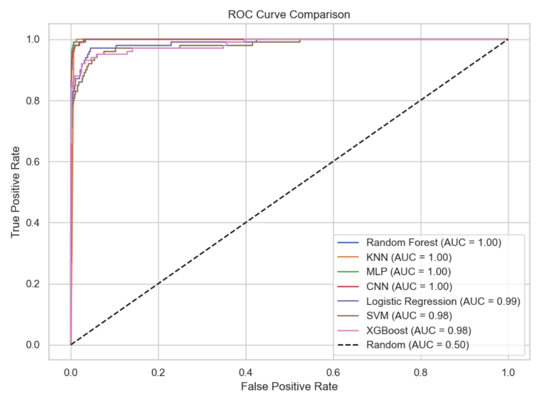


Figure 4.2: ROC-AUC Curve for the Models in test 1

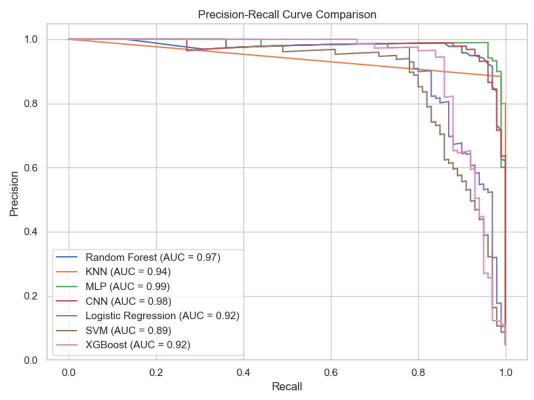


Figure 4.3: Precision-Recall Curve for the models in test 1

model. We see continuation of the trend where the XGBoost model has low precision score and high recall score hinting that the model can efficiently locate the fraudulent transaction but has problem differentiating between the types of transaction.

Model	Precision	Recall	F1 Score	Accuracy
Random Forest	93.651%	98.333%	95.935%	99.837%
KNN	65.217%	100.000%	78.947%	98.954%
MLP	90.909%	100.000%	95.238%	99.804%
CNN	80.556%	96.667%	87.879%	99.477%
Logistic Regression	51.887%	91.667%	66.265%	98.170%
SVM	43.750%	93.333%	59.574%	97.516%
XGBoost	3.297%	100.000%	6.383%	42.484%

Table 4.2: Classification Report Analysis for test 2

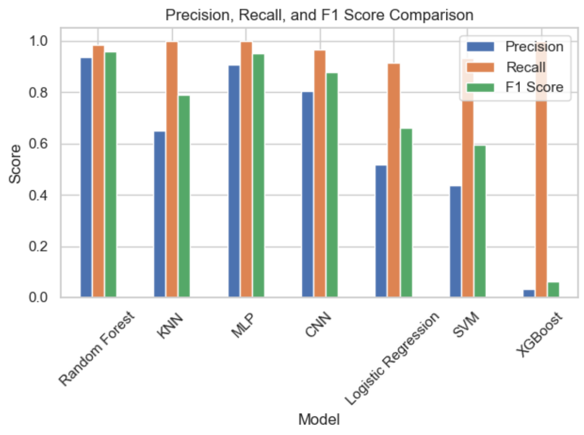


Figure 4.4: Visualisation of the Classification Reports of the models in test 2

4.4.2 ROC-AUC Curve and Precision-Recall Curve Analysis

The ROC-AUC Curve and the Precision-Recall have been put in for further reference. With these visualisations, it illustrates the performance metrics of the MLP, CNN and Random Forest is the best out of the bunch, followed by KNN, Logistic Regression and XGBoost.

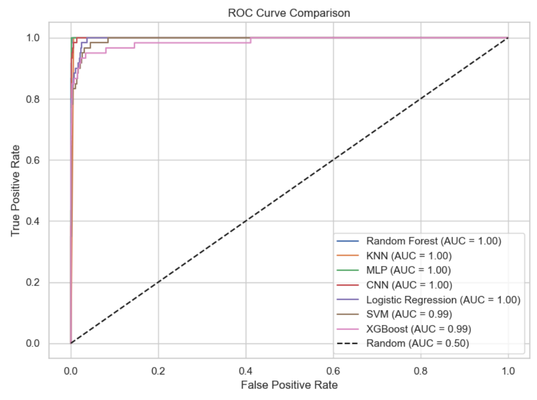


Figure 4.5: ROC-AUC Curve for the Models in test 2

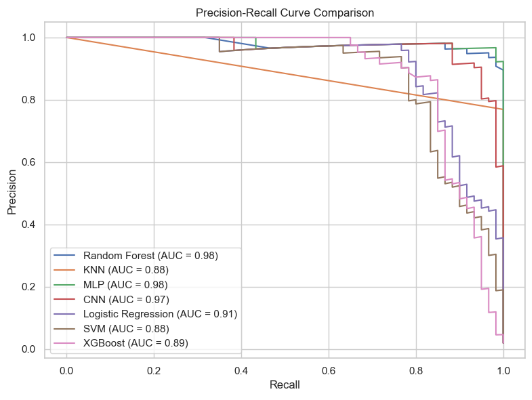


Figure 4.6: Precision-Recall Curve for the models in test 2

4.5 The Third Test

The third data set consists of 1421 genuine transactions and 24 fraudulent transactions. This dataset has been generated using Google Gemini by putting the original training dataset into Gemini and then it generates random data points on the basis of the reference dataset.

4.5.1 Classification Report Analysis

Model	Precision	Recall	F1 Score	Accuracy
Random Forest	96.000%	100.000%	97.959%	99.931%
KNN	60.000%	100.000%	75.000%	98.893%
MLP	85.714%	100.000%	92.308%	99.723%
CNN	85.714%	100.000%	92.308%	99.723%
Logistic Regression	46.939%	95.833%	63.014%	98.131%
SVM	35.484%	91.667%	51.163%	97.093%
XGBoost	2.824%	100.000%	5.492%	42.837%

Table 4.3: Classification Report Analysis for test 3

As seen in the tabulated data from 4.3, The performance for the MLP, CNN and Random Forest have the best results interms of F1 score, Accuracy, Precision and Recall. With the MLP we have a perfect recall score showing the model's ability to perfectly identify fraudulent transactions. The trend for the XGBoost continues from the previous runs showing high recall scores but low precision scores.

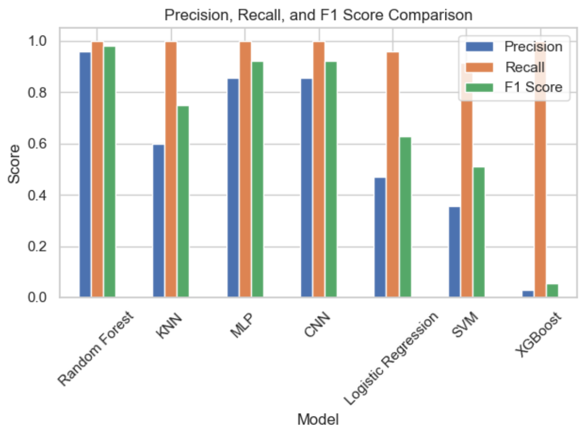
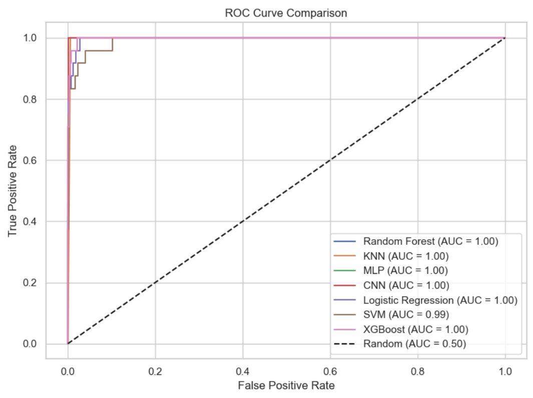


Figure 4.7: Visualisation of the Classification Reports for all the models in test 3

4.5.2 ROC-AUC Curve and Precision-Recall Curve Analysis

As seen from the figures for all the ROC-AUC Curves and the Precision-Recall Curves, the best performing models are MLP, CNN and Random Forest, followed by KNN, Logistic Regression and XGBoost.



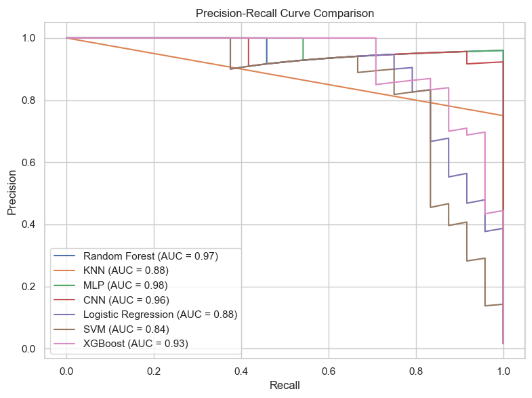


Figure 4.8: ROC-AUC Curve for the Models in test 3

Figure 4.9: Precision-Recall Curve for the models in test 3

4.6 Inferences Drawn

While reviewing the 3 test result evaluations, we can see that the best performing the model on average was the MLP or Mulilayer Perceptron, followed by the Convolutional Network and Random Forest Classifier. The XGBoost model has the worst performance out of all.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The study started with the intention of performing a comparative analysis between the different machine learning and deep learning models as discussed above. The models represent a subset of all the available models that can be used to achieve the intended results and then implemented in a dashboard for comparison and visualisation. The approach was also to make the model's final outcome scalable and portable and to ensure accuracy in real world scenarios. Taking into account some of these features, the following inferences were drawn by the aims of this study as mentioned before.

- Develop a credit card fraud detection system that can handle imbalanced data effectively - The models that have been trained and tested are of sufficient capability of handling the credit card fraud problem and can be integrated in real-world applications for the benefit of the wider audience in general.
- Compare multiple machine learning and deep learning algorithms to find the best-performing model for fraud detection - A comparative analysis was done on a number of different machine learning and deep learning models. The outcome of this study shows that in real-world scenarios, the Multilayer Perceptron (MLP), Convolutional Neural Network (CNN) and the Random Forest Classifiers are the best candidates for this purpose. The implementation of the Random Forest Classifier is the simplest out of all the preferred models however, with the MLP and CNN, the accuracy all around the testing process shows a certain level of uniformity with particularly the models' ability to differentiate between borderline data points.
- Integrate these models into a user-friendly Streamlit dashboard for real-time monitoring and visualisation of model performance - The Streamlit library can be used effectively to create a simple to use user-friendly dashboard for analysis and visualisation of these models. The ease of implementation is development-friendly and requires little prior knowledge of front-end development to implement it successfully.
- Ensure scalability and accuracy in real-world applications, where fraudulent transactions represent a small fraction of all transactions - The entire development process took place within the Google Colab Jupyter Notebooks and Python. This shows that the approach was done on hardware found commonly and does not necessarily require specialised components like GPUs for data analysis. The approach is highly scalable and can be fine-tuned to get better results on more capable hardware.

In conclusion, credit card fraud detection is a real threat to the wider population as this form of transaction is predominant. But with the right models like the Multilayer Perceptron (MLP), Convolutional Neural Network (CNN) and Random Forest Classifier, this problem can be tackled effectively and a suitable solution can be found.

5.2 Future Work

There is scope for further analysis and building on the work that has been done. This includes implementing regulatory compliance and ethical considerations of the study, live detection of transactions, scaling and development of the process, and continuous retraining. Some of these points have been discussed further.

- Deep Learning Enhancements - The use of Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) models can be further explored to look into performance enhancement.
- Hybrid Model Architectures - A combination of deep learning techniques and traditional supervised models can be used.
- Multi-modal Data Sources - Incorporating the use of multi-modal data sources like email IDs, fingerprints, behavioural pattern, and location data for enhanced accuracy.
- Up scaling and Development - Up scaling the models and implementation to integrate it into a cloud based architecture for real-time detection by making it into a microservices.
- Blockchain Implementation - The models can be integrated into the block chain for example the Etherium blockchain so that the transactions are recorded into immutable ledgers and hence the detection and prevention can be done with better effectiveness.

Chapter 6

BCS Project Criteria and Self-Reflection

1. Practical and Analytical Skills: The project will apply machine learning techniques learned during the degree program to solve a real-world problem. Learning techniques like supervised machine learning, deep learning, unbalanced dataset handling and sampling, and creation of a dashboard have been discussed in this project.

2. Innovation and Creativity: The project explores the use of various machine learning models and deep learning techniques to enhance fraud detection. A comparative analysis is done between a different machine learning and deep learning models. Some enhancement and model performance improvement have also been discussed. The models are then implemented in a user-friendly dashboard as discussed in the aims of the project.

3. Synthesis of Information: The project integrates information from various sources, including literature and technical documentation, to design and implement a solution. A thorough literature review was done on topics related to machine learning, deep learning, anomaly detection, application of Streamlit for dashboard creation and previous research done into credit card fraud detections.

4. Real Need: Credit card fraud detection is a critical need in the financial industry, addressing a significant problem. Traditional methods are often reactive in nature and are sometimes inadequate. With the rise of digital payments and e-commerce, the use of credit cards is in high and demand and as a result of this, credit card fraud will also be at a rise.

5. Self-Management: The project requires managing multiple tasks, The self-management begins from the start of the literature reviews of the relevant topics concerned with the topic and then selecting which of the methods are to be implemented. The research proposal was then submitted outlining the design of the project. This is then followed by a model training and tuning phase to bring out the best performance and results. After this, the dashboard development phase took place followed by the testing and evaluation. Finally, the documentation of the project was completed after a demonstration of the dashboard and its working components.

6. Critical Self-Evaluation:

   I have learnt a lot during the entire process of the MSc Project. It is one thing to learn the different machine learning techniques, and different to how you implement the teachings and knowledge into your work or study. This project required a lot of dedication and discipline to be completed. My primary goal was to analyse and find out which model would be the most suitable for credit card fraud detection and anomaly detection and then create a dashboard to see the results of that. In finding the answer to that question, I had to tackle problems like unbalanced dataset and data handling at large, training and evaluating different anomaly detection models and building my own Multilayer Perceptron and Convolutional Nueural Network. Some of the things I feel were lacking and I could have done better for improvement in results are as follows:

   - I could have tested the model efficiency and performance by applying different sampling techniques such as Adaptive Synthetic (ADASYN) technique for oversampling and Cluster-Based undersampling. I wasn't able to implement all of these due to the time constraint and would like to evaluate these techniques further in the future as a process of refinement.
   - The models that were selected were suitable for the task at hand. However, A wider comparative analysis could have been done using algorithms like CatBoost and LSTM networks to get a better understanding.
   - In some of the research papers, XGBoost has been mentioned as one of the most effective ways for detecting fraud and if we look into the preliminary evaluation of the models, we see that the performance of the XGBoost model is indeed quite good. However, the final results of the test tells a different story in terms of general accuracy. Even though the XGBoost model was good at finding out which transaction was fraud, it failed differentiate the borderline genuine transactions. This might be an issue of how the model was ported and implemented and requires further analysis on my part to see what was the issue at large.
   - I could have looked into the aspect of the dashboard being implemented on a blockchain network and have an analysis or reading on the real-world performance of the models. This aspect I have decided to invest in, and further develop the idea to incorporate this portion.

   Some of the positive outcomes that came from this project can be listed as follows:

   - During this project, my EDA skills have been improved upon and refined. This will help me in performing EDA on different subjects with greater confidence and efficacy.
   - The extensive use of Python and its library have honed my programming skills to a greater level where I am confident in using and developing machine learning and deep learning based solutions to problems.
   - Due to rigor required to complete an MSc project, my time management skills, discipline and attention span have definitely improved which will help me further in my future endeavours.

   To conclude, I would like to say that the main purpose of the project was achieved when I aimed to create a dashboard and find out the best-fit model for credit card fraud detection. However, there is significant room for improvement in my work and I will keep on working on this project until I receive the best possible result.

Chapter 7

Project Ethics

The project has been reported to have followed ethical compliances as listed below:

Data Category: B

Participant Category: 0

The data was sourced from kaggle.com. It contains only numerical input variables which are the result of a PCA transformation. Due to confidentiality, further information on the features is not available and anonymised. Features V1, V2, . . . V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.
Due to the sourcing of Human data, which are credit card transaction data from a public dataset and anonymised, the data category has been classified as B.
Due to the lack of human participation in any of the phases of the data collection or performance evaluation, the Participation Category is set to 0.

References

[1] A. Furniture, "The role of ai in revolutionizing financial services," 2024. Accessed: 2024-09-12.

[2] A. D. Pozzolo, *Adaptive Machine Learning for Credit Card Fraud Detection.* PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, December 2015. Ph.D. Thesis.

[3] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare, "Credit card fraud detection using machine learning techniques: A comparative analysis," in *2017 international conference on computing networking and informatics (ICCNI)*, pp. 1–9, IEEE, 2017.

[4] A. Dal Pozzolo, O. Caelen, and G. Bontempi, "Credit card fraud detection." `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`, 2016. Accessed: 2024-09-09.

[5] P. Parekh, C. Rana, K. Nalawade, and S. Dholay, "Credit card fraud detection with resampling techniques," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–7, IEEE, 2021.

[6] X. Yu, X. Li, Y. Dong, and R. Zheng, "A deep neural network algorithm for detecting credit card fraud," in *2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pp. 181–183, 2020.

[7] V. Jain, H. Kavitha, and S. M. Kumar, "Credit card fraud detection web application using streamlit and machine learning," in *2022 IEEE International Conference on Data Science and Information System (ICDSIS)*, pp. 1–5, IEEE, 2022.

[8] P. S. Foundation, *Python: A Dynamic, Open-Source Programming Language*, 2023. Version 3.12.4.

[9] W. McKinney, *Data Structures for Statistical Computing in Python*, 2010. Accessed: 2023-09-18.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'io, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, *Array programming with NumPy*, 2020. Accessed: 2023-09-18.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. Accessed: 2023-09-18.

[12] J. D. Hunter, *Matplotlib: A 2D Graphics Environment*, 2007. Accessed: 2023-09-18.

[13] M. Waskom, *Seaborn: Statistical Data Visualization*, 2021. Version 0.11.2.

[14] TensorFlow, *TensorFlow Python API — TensorFlow v2.13.0 documentation*, 2023. Accessed: 2024-09-12.

[15] G. Research, "Google colaboratory," 2023. Accessed: 2023-09-18.

[16] Keras, *Keras: The Python Deep Learning API*, 2023. Accessed: 2024-09-12.

[17] OpenAI, "Chatgpt: Language model for conversational ai." `https://chat.openai.com/`, 2023. Accessed: 2024-09-19.

[18] G. R. Team, "Google gemini: Next-generation ai for google search and beyond." `https://blog.google/products/search/google-gemini-ai/`, 2023. Accessed: 2024-09-19.

[19] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *Symposium on Computational Intelligence and Data Mining (CIDM), IEEE*, 2015.

[20] F. Ferreira, N. Lourenço, B. Cabral, and J. P. Fernandes, "When two are better than one: Synthesizing heavily unbalanced data," *IEEE Access*, vol. 9, pp. 150459–150469, 2021.

[21] H. Zhu, M. Zhou, G. Liu, Y. Xie, S. Liu, and C. Guo, "Nus: Noisy-sample-removed undersampling scheme for imbalanced classification and application to credit card fraud detection," *IEEE Transactions on Computational Social Systems*, 2023.

[22] G. Lemaître, F. Nogueira, and C. K. Aridas, "Randomundersampler - under-sampling by randomly picking samples." `https://imbalanced-learn.org/dev/references/generated/imblearn.under_sampling.RandomUnderSampler.html`, 2024. Accessed: 2024-09-10.

[23] M. N. Y. Ali, T. Kabir, N. L. Raka, S. S. Toma, M. L. Rahman, and J. Ferdaus, "Smote based credit card fraud detection using convolutional neural network," in *2022 25th International Conference on Computer and Information Technology (ICCIT)*, pp. 55–60, IEEE, 2022.

[24] G. Lemaître, F. Nogueira, and C. K. Aridas, "Smote - synthetic minority over-sampling technique." `https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html`, 2024. Accessed: 2024-09-10.

[25] G. Lemaître, F. Nogueira, and C. K. Aridas, "Pipeline." `https://imbalanced-learn.org/stable/references/generated/imblearn.pipeline.Pipeline.html`, 2024. Accessed: 2024-09-10.

[26] X. Yu, X. Li, Y. Dong, and R. Zheng, "A deep neural network algorithm for detecting credit card fraud," in *2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pp. 181–183, IEEE, 2020.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *scikit-learn: Machine Learning in Python*, 2011. Accessed: 2024-09-11.

[28] A. Singh, A. Singh, A. Aggarwal, and A. Chauhan, "Design and implementation of different machine learning algorithms for credit card fraud detection," in *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pp. 1–6, 2022.

[29] R. Raman, V. Kumar, B. G. Pillai, D. Rabadiya, R. Divekar, and H. Vachharajani, "Detecting credit card fraud: A comparative analysis of knn, random forest, and logistic regression methods," in *2024 Second International Conference on Data Science and Information System (ICDSIS)*, pp. 1–5, IEEE, 2024.

[30] S. Xuan, G. Liu, Z. Li, L. Zheng, S. Wang, and C. Jiang, "Random forest for credit card fraud detection," in *2018 IEEE 15th international conference on networking, sensing and control (ICNSC)*, pp. 1–6, IEEE, 2018.

[31] scikit-learn, *sklearn.ensemble.RandomForestClassifier — scikit-learn 1.2.2 documentation*, 2023. Accessed: 2024-09-12.

[32] S. Mittal and S. Tyagi, "Performance evaluation of machine learning algorithms for credit card fraud detection," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 320–324, IEEE, 2019.

[33] R. R. Popat and J. Chaudhary, "A survey on credit card fraud detection using machine learning," in *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1120–1125, 2018.

[34] scikit-learn, *sklearn.svm.SVC — scikit-learn 1.2.2 documentation*, 2023. Accessed: 2024-09-12.

[35] A. Peter, K. Manoj, and P. Kumar, "Blockchain and machine learning approaches for credit card fraud detection," in *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 1034–1041, IEEE, 2023.

[36] scikit-learn, *sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.2.2 documentation*, 2023. Accessed: 2024-09-12.

[37] A. H. Nadim, I. M. Sayem, A. Mutsuddy, and M. S. Chowdhury, "Analysis of machine learning techniques for credit card fraud detection," in *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*, pp. 42–47, IEEE, 2019.

[38] T. Wang and Y. Zhao, "Credit card fraud detection using logistic regression," in *2022 International Conference on Big Data, Information and Computer Network (BDICN)*, pp. 301–305, 2022.

[39] scikit-learn, $sklearn.linear_model.LogisticRegression|scikit - learn1.2.2documentation, 2023. Accessed : 2024 - 09 - 12.$

[40] C. V. Priscilla and D. P. Prabha, "Influence of optimizing xgboost to handle class imbalance in credit card fraud detection," in *2020 third international conference on smart systems and inventive technology (ICSSIT)*, pp. 1309–1315, IEEE, 2020.

[41] XGBoost, *XGBoost Python Package — xgboost 2.0.0 documentation*, 2023. Accessed: 2024-09-12.

[42] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009.

[43] A. Abd El Naby, E. E.-D. Hemdan, and A. El-Sayed, "Deep learning approach for credit card fraud detection," in *2021 International Conference on Electronic Engineering (ICEEM)*, pp. 1–5, IEEE, 2021.

[44] C. Zhang, X. Pan, H. Li, A. Gardiner, I. Sargent, J. Hare, and P. M. Atkinson, "A hybrid mlp-cnn classifier for very fine resolution remotely sensed image classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 140, pp. 133–144, 2018.

[45] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, pp. 611–629, 2018.

[46] M. L. Gambo, A. Zainal, and M. N. Kassim, "A convolutional neural network model for credit card fraud detection," in *2022 International Conference on Data Science and Its Applications (ICoDSA)*, pp. 198–202, IEEE, 2022.

[47] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1595–1598, IEEE, 2018.

[48] M. Sabokrou, M. Fayyaz, M. Fathy, Z. Moayed, and R. Klette, "Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes," *Computer Vision and Image Understanding*, vol. 172, pp. 88–97, 2018.

[49] B. Staar, M. Lütjen, and M. Freitag, "Anomaly detection with convolutional neural networks for industrial surface inspection," *Procedia CIRP*, vol. 79, pp. 484–489, 2019.

[50] S. Azam, S. Montaha, K. U. Fahim, A. R. H. Rafid, M. S. H. Mukta, and M. Jonkman, "Using feature maps to unpack the cnn 'black box'theory with two medical datasets of different modality," *Intelligent Systems with Applications*, vol. 18, p. 200233, 2023.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python." `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`, 2011. Accessed: 2024-09-14.

[52] T. O'Malley, E. Bursztein, H. Long, F. Chollet, *et al.*, "Kerastuner." `https://keras.io/api/keras_tuner/tuners/random/`, 2019. Accessed: 2024-09-14.

[53] G. N. Ahmad, H. Fatima, S. Ullah, A. S. Saidi, *et al.*, "Efficient medical diagnosis of human heart diseases using machine learning techniques with and without gridsearchcv," *IEEE Access*, vol. 10, pp. 80151–80173, 2022.

[54] D. Kartini, D. T. Nugrahadi, A. Farmadi, *et al.*, "Hyperparameter tuning using gridsearchcv on the comparison of the activation function of the elm method to the classification of pneumonia in toddlers," in *2021 4th International Conference of Computer and Informatics Engineering (IC2IE)*, pp. 390–395, IEEE, 2021.

[55] M. Tiwari, V. Sharma, D. Bala, *et al.*, "Credit card fraud detection," *Journal of Algebraic Statistics*, vol. 13, no. 2, pp. 1778–1789, 2022.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python." `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html`, 2011. Accessed: 2024-09-14.

[57] R. Andonie and A.-C. Florea, "Weighted random search for cnn hyperparameter optimization," *arXiv preprint arXiv:2003.13300*, 2020.

[58] A. R. M. Rom, N. Jamil, and S. Ibrahim, "Multi objective hyperparameter tuning via random search on deep learning models," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 22, no. 4, pp. 956–968, 2024.

[59] E. Ileberi, Y. Sun, and Z. Wang, "Performance evaluation of machine learning methods for credit card fraud detection using smote and adaboost," *IEEE Access*, vol. 9, pp. 165286–165294, 2021.

[60] A. Singh, A. Singh, and A. Chauhan, "Design and implementation of different machine learning algorithms for credit card fraud detection," *Proceedings of the Dept. of Information Technology, Delhi Technological University*, 2020.

[61] Streamlit, *Streamlit: A Faster Way to Build and Share Data Apps*, 2024. Accessed: 2024-09-15.

[62] G. Varoquaux and team, "Joblib: Efficient computation with python." `https://joblib.readthedocs.io/en/latest/`, 2023. Version 1.3.2, Accessed: 2024-09-19.

Appendix A

Python Code for Compiling test results

The following Python code is used for model evaluation for each test case compiled into a spreadsheet:

```python
import pandas as pd
import joblib
import xgboost as xgb
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, recall_score, f1_score,
    accuracy_score

# Load the data
def load_data(file_path):
    df = pd.read_csv(file_path)
    X = df.drop(columns=['Class'])
    y = df['Class']
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, y

# Model predictions and probabilities
def model_predictions(model, X, model_type='sklearn'):
    if model_type == 'keras':
        preds = model.predict(X).ravel()
        return (preds > 0.5).astype(int), preds
    elif model_type == 'xgboost':
        preds = model.predict(xgb.DMatrix(X))
        return (preds > 0.5).astype(int), preds
    else:
        if hasattr(model, "predict_proba"):
            preds = model.predict_proba(X)[:, 1]
        else:
            preds = model.decision_function(X)
        return model.predict(X), preds

# Load the models
def load_models():
    models = {
        'Random Forest': joblib.load('random_forest_model.pkl'),
        'KNN': joblib.load('knn_model.pkl'),
        'MLP': tf.keras.models.load_model('mlp_model.h5'),
        'CNN': tf.keras.models.load_model('cnn_model3.h5'),
        'Logistic Regression': joblib.load('logistic_regression_model.
            pkl'),
        'SVM': joblib.load('svm_model.pkl'),
        'XGBoost': joblib.load('xgboost_model.pkl')
    }
    return models

# Evaluate all models and return tabular results
def evaluate_models(models, X, y):
    results = []

    for name, model in models.items():
        if name in ['MLP', 'CNN']:
            preds, _ = model_predictions(model, X, model_type='keras')
        elif name == 'XGBoost':
            preds, _ = model_predictions(model, X, model_type='xgboost'
                )
        else:
            preds, _ = model_predictions(model, X)

        precision = precision_score(y, preds)
        recall = recall_score(y, preds)
        f1 = f1_score(y, preds)
        accuracy = accuracy_score(y, preds)

        results.append({
            'Model': name,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1,
            'Accuracy': accuracy
        })

    df_results = pd.DataFrame(results).set_index('Model')
    return df_results

if __name__ == "__main__":
    # Load data and models
    X, y = load_data('sample creditcard files.csv')
    models = load_models()

    # Evaluate models and output results
    df_results = evaluate_models(models, X, y)
    print(df_results)

    # Save the results to an Excel file
    df_results.to_excel('test 3.xlsx')
```

Appendix B

Python Code for Visualising the Test Results

The following Python code is used for model evaluation and visualisation for each test case:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    precision_recall_curve, roc_curve, auc, roc_auc_score, f1_score,
        recall_score, precision_score
)
import numpy as np
import pandas as pd
import joblib
import xgboost as xgb
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load the data


def load_data(file_path):
    df = pd.read_csv(file_path)
    X = df.drop(columns=['Class'])
    y = df['Class']
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, y

# Model predictions and probabilities


def model_predictions(model, X, model_type='sklearn'):
    if model_type == 'keras':
        preds = model.predict(X).ravel()
        return (preds > 0.5).astype(int), preds
    elif model_type == 'xgboost':
        preds = model.predict(xgb.DMatrix(X))
        # Ensure preds are binary (0 or 1)
        return (preds > 0.5).astype(int), preds
    else:
        if hasattr(model, "predict_proba"):
            preds = model.predict_proba(X)[:, 1]
        else:
            preds = model.decision_function(X)
        return model.predict(X), preds

# Load the models


def load_models():
    models = {
        'Random Forest': joblib.load('random_forest_model.pkl'),
        'KNN': joblib.load('knn_model.pkl'),
        'MLP': tf.keras.models.load_model('mlp_model.h5'),
        'CNN': tf.keras.models.load_model('cnn_model3.h5'),
        'Logistic Regression': joblib.load('logistic_regression_model.
           pkl'),
        'SVM': joblib.load('svm_model.pkl'),
        'XGBoost': joblib.load('xgboost_model.pkl')
    }
    return models

# Evaluate all models and plot comparison


def evaluate_models(models, X, y):
    results = []

    sns.set(style="whitegrid")

    # Precision-Recall Curve
    plt.figure(figsize=(8, 6))
    for name, model in models.items():
        if name in ['MLP', 'CNN']:
            preds, pred_probs = model_predictions(model, X, model_type=
               'keras')
        elif name == 'XGBoost':
            preds, pred_probs = model_predictions(
                model, X, model_type='xgboost')
        else:
            preds, pred_probs = model_predictions(model, X)

        precision_curve, recall_curve, _ = precision_recall_curve(
            y, pred_probs)
        pr_auc = auc(recall_curve, precision_curve)
        plt.plot(recall_curve, precision_curve,
                 label=f'{name} (AUC = {pr_auc:.2f})')

    plt.title('Precision-Recall Curve Comparison')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()

    # ROC Curve
    plt.figure(figsize=(8, 6))
    for name, model in models.items():
        if name in ['MLP', 'CNN']:
```

```python
            preds, pred_probs = model_predictions(model, X, model_type='keras')
        elif name == 'XGBoost':
            preds, pred_probs = model_predictions(
                model, X, model_type='xgboost')
        else:
            preds, pred_probs = model_predictions(model, X)

        fpr, tpr, _ = roc_curve(y, pred_probs)
        roc_auc = roc_auc_score(y, pred_probs)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.50)')
    plt.title('ROC Curve Comparison')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()

    # Precision, Recall, and F1 Score Comparison (Bar Plot)
    for name, model in models.items():
        if name in ['MLP', 'CNN']:
            preds, _ = model_predictions(model, X, model_type='keras')
        elif name == 'XGBoost':
            preds, _ = model_predictions(model, X, model_type='xgboost')
        else:
            preds, _ = model_predictions(model, X)

        precision = precision_score(y, preds)
        recall = recall_score(y, preds)
        f1 = f1_score(y, preds)

        results.append({
            'Model': name,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1
        })

    df_results = pd.DataFrame(results).set_index('Model')
    plt.figure(figsize=(10, 6))
    df_results.plot(kind='bar')
    plt.title('Precision, Recall, and F1 Score Comparison')
    plt.ylabel('Score')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    # Load data and models
    X, y = load_data('sample creditcard files.csv')
    models = load_models()

    # Evaluate and plot models
    evaluate_models(models, X, y)
```

Appendix C

Running the Streamlit Dashboard

Steps to Run the DashboardApp

1. Make sure the models are properly loaded into the path described by the code.
2. Make sure that Streamlit is installed in the Python environment to be used in. If not please use the following command in the Python terminal to install Streamlit: `pip install streamlit`
3. Make sure other libraries are imported and installed as well for example Scikit-Learn, Matplotlib, TensorFlow, xgboost, Pandas and Numpy. To install the listed libraries, please use the following command:
   - `pip install sklearn` for Scikit-Learn.
   - `pip install matplolib` for Matplotlib.
   - `pip install tensorflow` for TensorFlow.
   - `pip install xgboost` for XGBoost.
   - `pip install numpy` for Numpy.
   - `pip install pandas` for Pandas.
4. Run the app in the Python terminal by using the following command: `streamlit run app.py`