ASSIGNMENT COVER SHEET

PROGRAMME	:	Master of Business Analytics
SUBJECT CODE AND TITLE	:	BAA5073
ASSIGNMENT TITLE	:	Data Mining
LECTURER	:	Prof. Keshab Shrestha
ASSIGNMENT DUE DATE:		June 30, 2025

STUDENT'S DECLARATION

1. I hereby declare that this assignment is based on my own work except where acknowledgement of sources is made.
2. I also declare that this work has not been previously submitted or concurrently submitted for any other courses in Sunway University/College or other institutions.
   [Submit "Turn-it-in" report (please tick √): Yes √ No]

NO.	NAME	STUDENT ID NO.	SIGNATURE	DATE
1.	Suhail Dorasamy	20020574	Suhail Dorasamy	30/6/25
2.	Nidal Bencheikh Lehocine	19097617	Nidal Bencheikh Lehocine	30/6/25

E-mail Address / Addresses (according to the order of names above):

1. 20020475@imail.sunway.edu.my	4.
2. 19097617@imail.sunway.edu.my	5.
3.	6.

APPROVAL FOR LATE SUBMISSION OF ASSIGNMENT (If applicable)
IF extension is granted, what is the revised due date? ______

Signature of Lecturer: ______ Date: ______

Marker's Comments:

Marks and / or Grade Awarded: ______ Date:______

ADDENDUM

USE OF ARTIFICAL INTELLIGENCE (A.I.) DECLARATION

Students are allowed to use AI to support completion of assessments. However, students are reminded to do so ethically and transparently. This is so that (a) submissions can be fairly and accurately marked; and (b) feedback can be provided on the content that reflects student ability, in order to help with future submissions. Students are also reminded that in accordance with the University's Academic Malpractice Policy, Item 4.11.2, "*… the representation of work: written, visual, practical or otherwise, of any other person, including another student or* ***anonymous web-based material*** *[emphasis added], or any institution, as the candidate's own*" is considered malpractice.

Declaration

[ ] I / We used the following A.I. tools to produce content in this submission:

Tool	**Purpose**	**Prompts**	**Sections where AI output was used / Outcome(s) in the submission**
e.g. ChatGPT	*e.g. Generating points for the essay* *Structuring the essay*	*e.g. "Give me 5 key talking points for an essay on…"* *"Show me a structure for an essay on…"*	*e.g. The main point for Section 1.2 and 1.3 were generated by AI, but the discussion was not.* *The organization / structure of the essay was suggested by AI*
e.g. Grammarly	*e.g. Correcting grammar and spelling, improving sentence structure*	*N/A*	*e.g. Grammarly suggestions were used for all sections of the essay*

Note: Add additional rows if necessary.

OR

[X] I / We did not use any A.I. tools to produce any of the content in this submission.

NO.	NAME	STUDENT ID NO.	SIGNATURE	DATE
1.	Suhail Dorasamy	20020574	Suhail Dorasamy	22/6/25
2.	Nidal Bencheikh Lehocine	19097617	Nidal Bencheikh Lehocine	22/6/25

E-mail Address / Addresses (according to the order of names above):

1. 20020475@imail.sunway.edu.my	4.
2. 19097617@imail.sunway.edu.my	5.
3.	6.

Abstract

The purpose of our study was to implement and evaluate different machine learning models to detect credit card fraud as a rare event prediction problem. We also aimed to demonstrate how data mining can help to improve business-decision making in fraud prevention. The dataset utilized was highly imbalanced, with a low proportion of fraudulent transactions, and thus it was more representative of real-world data. Normalization and RobustScaler were both implemented during preprocessing to standardise the feature distribution, while SMOTE addressed the severe class imbalance and improve the minority class's proportion. The models selected for our analysis were Decision Tree, Random Forest and XGBoost, with hyperparameter tuning done through grid search CV. The evaluation of the three models found that Random Forest performed the best overall with XGBoost having better recall by worse interpretability, while also requiring extensive tuning, and Decision Tree was overall less accurate but was easier to interpret.

Table of Contents

Introduction

Fraud involves obtaining money, goods, or services through illegal or deceptive means. There are various types of fraud, including, business and investment fraud, elder fraud, cryptocurrency Fraud and most commonly credit card fraud. We will be focusing on Credit card fraud, with recent technological advancements the shift to online banking has become more prominent, while this shift offers convenience and efficiency to users, it simultaneously creates opportunities for fraudsters to exploit vulnerabilities and carry out fraudulent activities more easily and anonymously**.**

According to Cruz (2025), in 2024 alone, 62 million Americans experienced fraudulent charges on their credit cards, with total losses exceeding $6 billion dollars annually. This underscores the growing importance of implementing credit card fraud prediction mechanisms.

Although numerous fraud detection algorithms have been developed in response to the rise in credit card fraud, reactive systems alone may no longer be sufficient. "AI's predictive analytics provides proactive security as opposed to reactive security by anticipating such weaknesses before they are exploited" (Hafez et al., 2025). The authors strongly emphasizes the importance of prediction over detection, arguing that prediction can lead to earlier intervention and significantly enhances the overall security. To further support the superiority of predictive systems over reactive systems, Hafez et al. (2025) state that, "Static systems are reactive and outdated. Predictive models—especially those powered by AI—are portrayed as necessary advancements."

Data Mining Algorithms

1.0 EDA (visualization and feature comparisons)

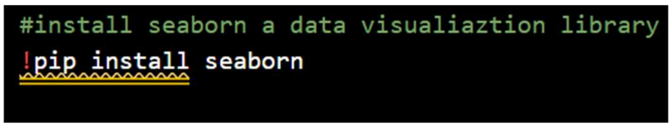
Exploratory Data Analysis (EDA) are a set of techniques used to summarise, visualise, and understand the structure and patterns of a dataset before applying any predictive modelling (Keim, 2002). It is a critical component of any data mining project as it guides subsequent preprocessing, feature engineering and model selection decisions. There are three main types of EDA, univariate, bivariate and multivariate, each type can also be broken down into graphical and non-graphical, with the main difference being that graphical EDA utilises graphics such as charts and diagrams to visualize data while non-graphical uses statistical techniques to explore data.

Univariate analysis examines each variable one at a time to understand its distribution and central tendency. Graphically, this form of EDA includes histograms, boxplots, and frequency tables. Bivariate analysis examines the relationship between two variables, especially between predictors and the target variable. Here, scatterplots and correlations coefficients are typically used to illustrate the variables relationships. Multivariate analysis facilitates the discovery of relationships among more than two variables simultaneously, typically using pair plots and heatmaps for visualization.

EDA has become increasingly important due to the increase in the amount of data generated daily, although it can be a time-consuming process, the results further outweigh the drawbacks. As stated by, Dhanshetti (2023) Exploratory Data Analysis (EDA) serves as a fundamental approach to understand, visualize, and draw preliminary conclusions from data before embarking on more advanced analytical techniques. Through EDA techniques we can not only gain a deep understanding of its features and trends but also understand whether we can use the dataset or not. In this study we will be using EDA to extract insights into our dataset, followed by training four predictive algorithms and comparing them based on efficiency and most importantly accuracy.

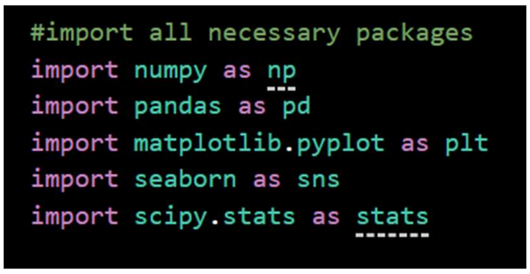
1.1 EDA implementation

In this report we have applied three main EDA techniques, Univariate, Bivariate and Multivariate. All three include two subsections, Graphical and Non-Graphical techniques. Before we can analyse the dataset, we must first import the dataset and the required libraries. In figure 1 and 2 we will see the code used to install seaborn a data visualization library and to import pandas, numpy, matplotlib, seaborn and stats. Which are libraries that help with data handling and EDA.



```
#install seaborn a data visualiaztion library
!pip install seaborn
```

Figure 1- seaborn installation



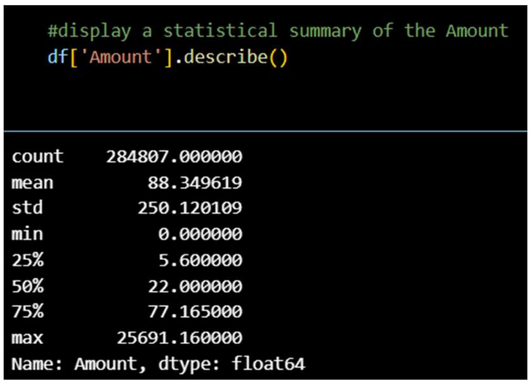
```
#import all necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

Figure 2 - package importing

1.1.1 Univariate

Our dataset came with 28 anonymized variables and 3 non-anonymized variables, First we started by analysing the non-anonymized variables, which are, Amount, Time and Class. Through Univariate analysis we can understand the variables better on a statistical level a, this helps with learning the shape of the data and how it can affect our model training.

In figure 3 we will see the function df.describe used to display a statistical analysis of the Amount column, through this analysis we can conclude that the Amount column has an absurd outlier of $25,691, while the mean transaction is only $88.35, such Outlier can pose significant issues during training.

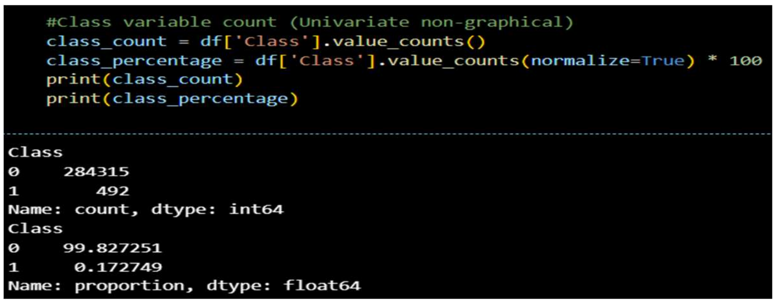


```
#display a statistical summary of the Amount
df['Amount'].describe()
```

```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

Figure 3 – statistical analysis of Amount

In figure 4 we used the function value.count() to count the number of entries in each class attribute (0 and 1), this can help us get an understanding of the overall balance of our dataset. And from the result we can see a huge difference in distribution, with "0" or "non-fraudulent transaction making up 99.82% of the dataset. Such an imbalance can cause biased model performance towards the majority class.

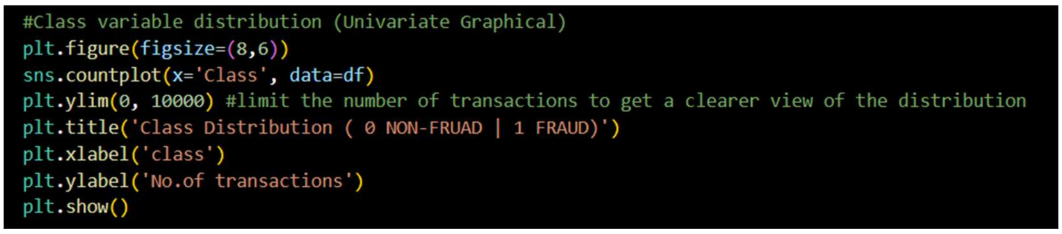


```
#Class variable count (Univariate non-graphical)
class_count = df['Class'].value_counts()
class_percentage = df['Class'].value_counts(normalize=True) * 100
print(class_count)
print(class_percentage)
```

```
Class
0    284315
1       492
Name: count, dtype: int64
Class
0    99.827251
1     0.172749
Name: proportion, dtype: float64
```

Figure 4 – 'Class' variable value count

In figure 5 we used the libraries matplotlib and seaborn to visualize the value counts generated previously and that can help give us a better picture of the imbalance. And as we can see in the plot, the difference in distribution s overwhelmingly large.



```
#Class variable distribution (Univariate Graphical)
plt.figure(figsize=(8,6))
sns.countplot(x='Class', data=df)
plt.ylim(0, 10000) #limit the number of transactions to get a clearer view of the distribution
plt.title('Class Distribution ( 0 NON-FRUAD | 1 FRAUD)')
plt.xlabel('class')
plt.ylabel('No.of transactions')
plt.show()
```

Figure 5 – histogram of Class(code)

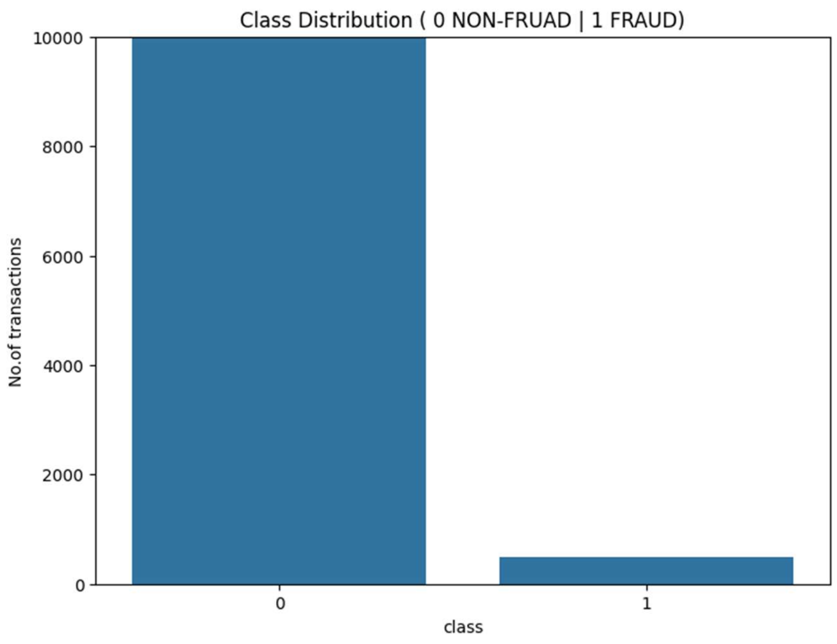


Figure 6 – histogram of imbalanced Class

1.1.2 Bivariate

To get a better understanding of the relationship between the Amount and Time columns with the class column, Bivariate analysis is done, in figure 7 the distribution of the transaction amount by class is visualized. The x-axis (Amount) is plotted with a logarithmic scale which helps us view a wider range of values. From the graph we can conclude that the non-fraudulent transactions (0) are very broad and spread out across multiple values, however the fraudulent transactions (1) are concentrated around smaller values, with significant peaks at “$1.0” and “$100.0”. From this observation we can conclude that most fraudulent transactions are of smaller values which indicates a card testing pattern, this is when fraudsters test the cards with very small amounts to check whether it is working. This indicates that Amount is a strong predictor of fraudulent transactions.

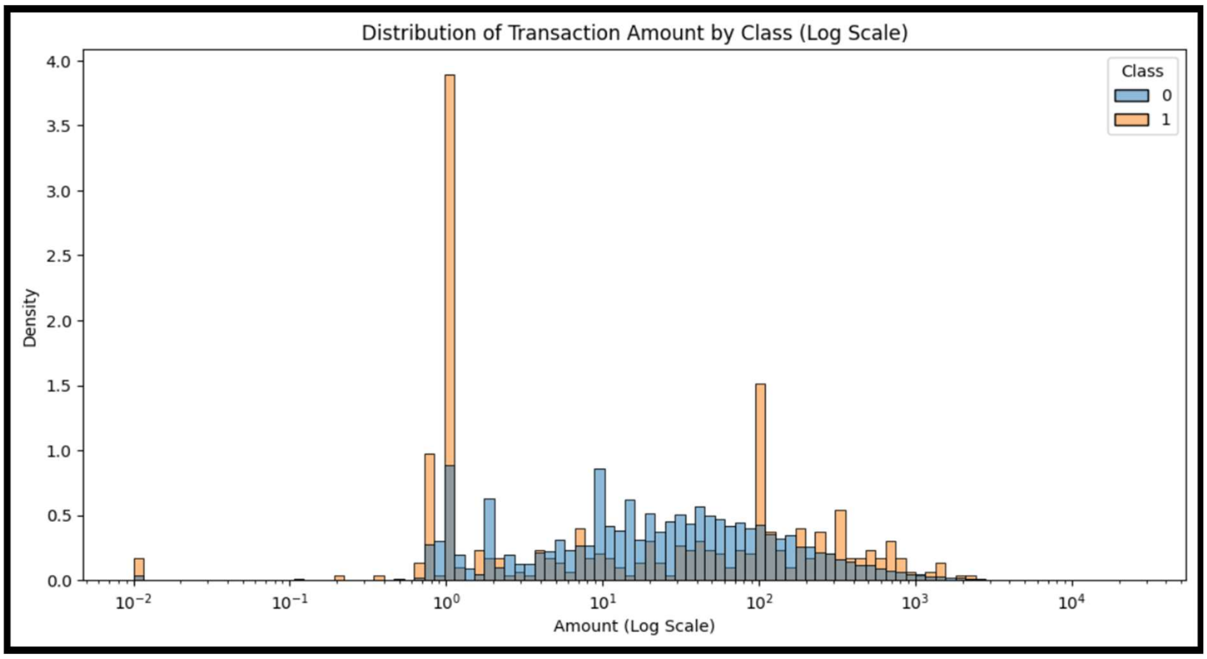


Figure 7 – Transaction amount distribution

Same method was used for time vs class graph; this would help us gauge when fraudulent transactions mainly occur. In figure 8, we can see the distribution of class attributes throughout a 48-hour period. We can conclude that the non-fraudulent transactions (Blue) inhibit a normal day/night cycle, with most transactions occurring at daytime followed by a significant dip in transactions throughout the nighttime. On the other hand, fraudulent transactions (orange) do no inhibit the same pattern, they occur around the clock but mostly throughout the night. As that is when users are most likely to be asleep and will not be able to report the transactions. This also shows that Time is strong predictor.

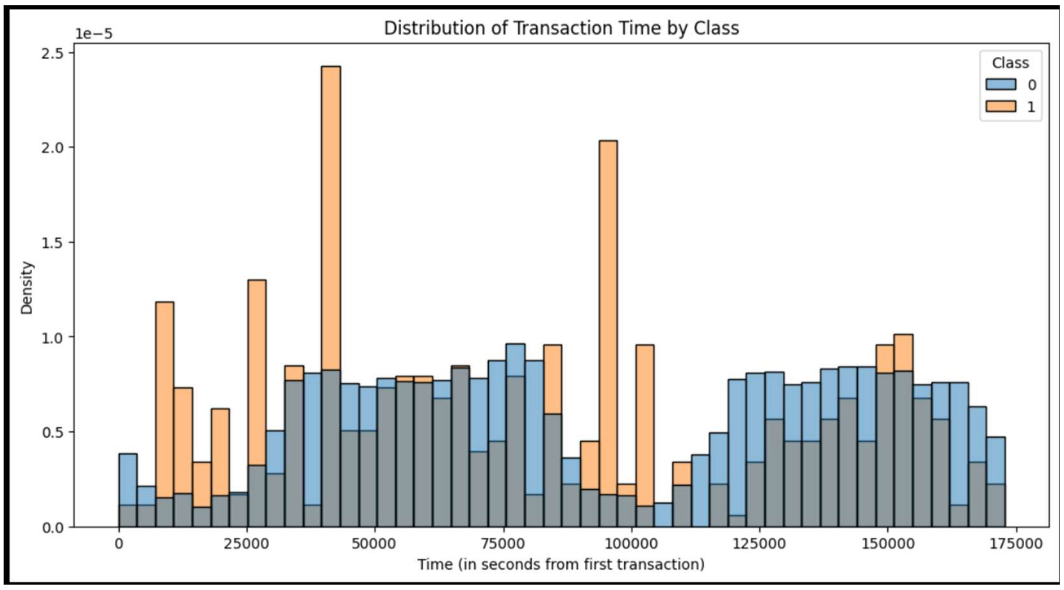
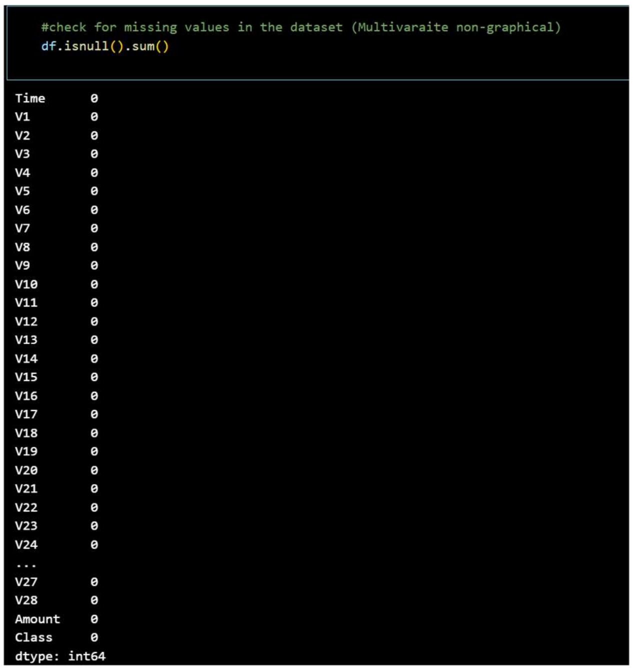


Figure 8 – Time vs Class

1.1.3 Multivariate

The final analysis is multivariate; by analysing multiple variables we can uncover even more insights. In figure 9, we used the df.isnull().sum() function to see how many empty variables there are in the dataset but fortunately we found none. Null instances can skew the model's predictions for instances with null data points or even without null data points, it introduces inconsistencies which can be detrimental.

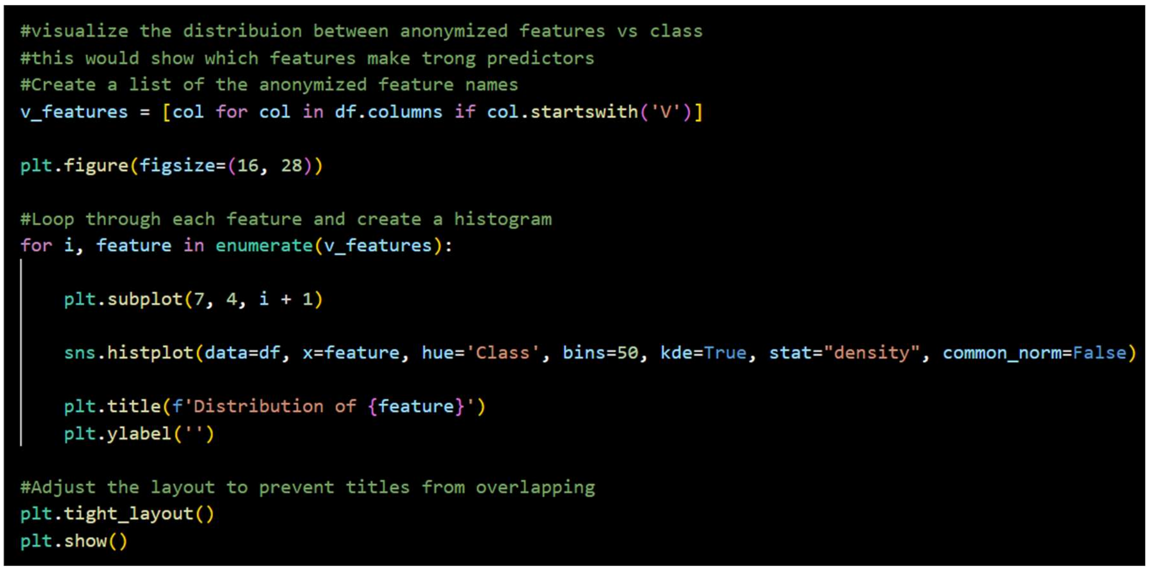


```
#check for missing values in the dataset (Multivaraite non-graphical)
df.isnull().sum()
```

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

Figure 9 – Null value count

Next in figure 10, we have dived into the anonymized data, without knowing the context it can be hard to make conclusions about the data. The best way to gain insights in this situation is to compare them with the class data and see which ones would make the best predictors. For efficiency an iteration loop was used to plot histograms for each variables V1-V28.



```python
#visualize the distribuion between anonymized features vs class
#this would show which features make trong predictors
#Create a list of the anonymized feature names
v_features = [col for col in df.columns if col.startswith('V')]

plt.figure(figsize=(16, 28))

#Loop through each feature and create a histogram
for i, feature in enumerate(v_features):

    plt.subplot(7, 4, i + 1)

    sns.histplot(data=df, x=feature, hue='Class', bins=50, kde=True, stat="density", common_norm=False)

    plt.title(f'Distribution of {feature}')
    plt.ylabel('')

#Adjust the layout to prevent titles from overlapping
plt.tight_layout()
plt.show()
```

Figure 10 – anonymous variables vs class

This code produced a total of 28 histograms, each visualizing the unique relationships between the anonymized features and the class. What we look for in these graphs is, difference in location, lack of difference and difference in shape. The difference in location indicates that there is a significant difference between the behaviour of the positive and negative class and the respective variable, whether in the positive or negative values. This is a strong indicator that the variable would be useful for detecting fraud. This is observed in variables V10, V12, V14, V17 which are visualized in figures 11 and 12. The lack of difference is an important indicator of whether a variables is a strong predictor or not because it shows us if there are visible difference between the behaviours of fraudulent and non-fraudulent transactions. For example in figure 11, variables V13 and V15 show no difference between fraudulent and non-fraudulent transactions. Finally is difference in shape, non-fraudulent transactions should have a smooth bell curve shape while fraudulent transactions should inhibit a series of sharp spikes mainly at $1 and 100$ as we observed during the univariate analysis.

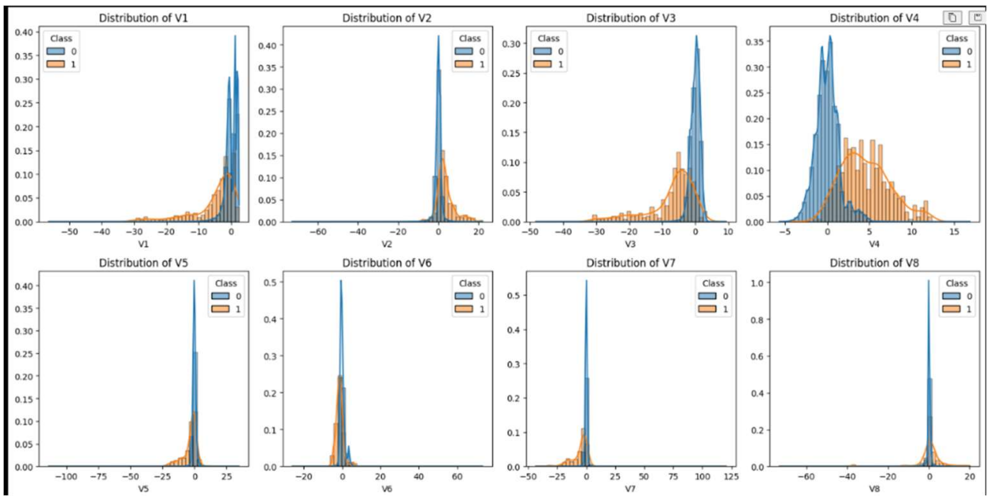


Figure 11 V1-V8

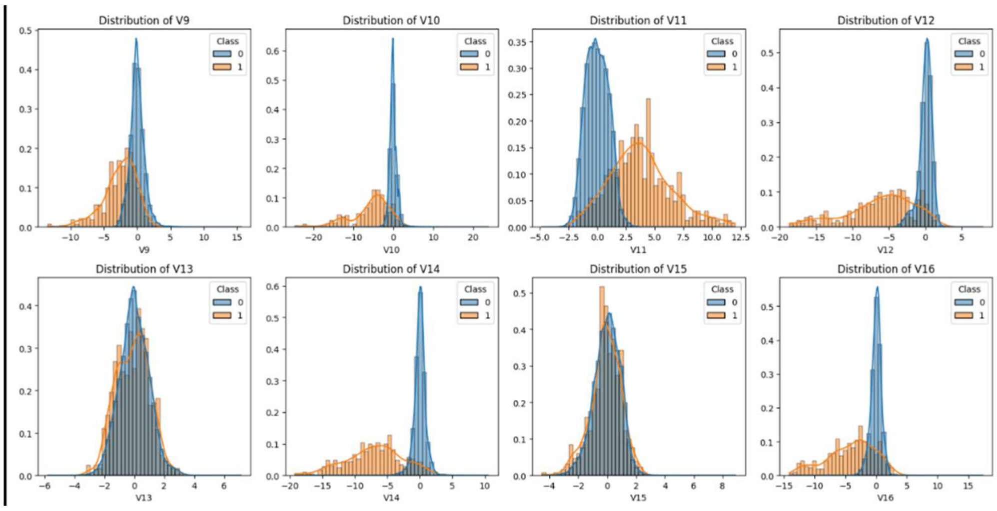


Figure 12 – V9-V16

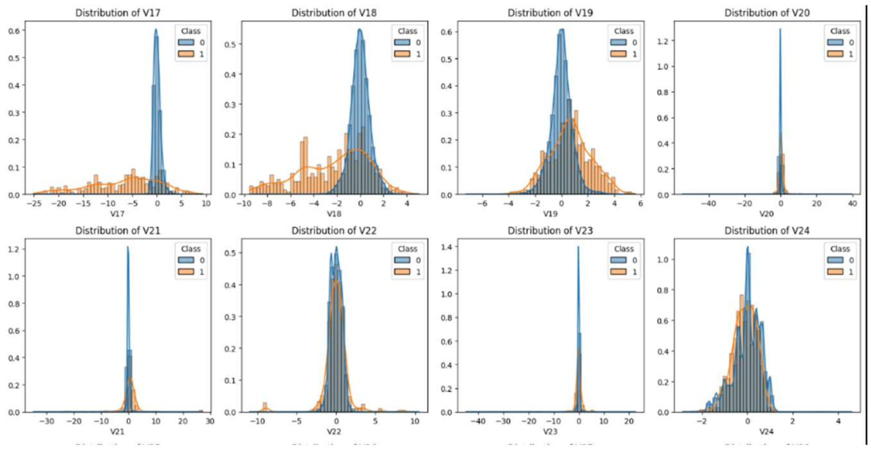


Figure 13 – V17-V24

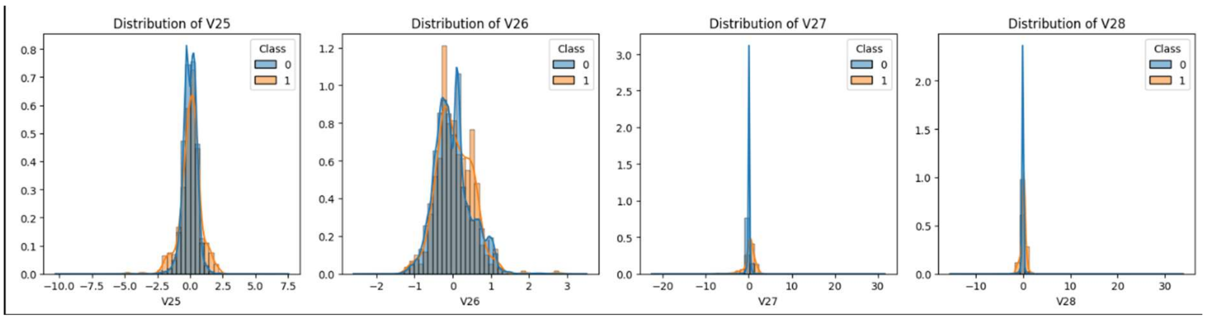


Figure 14 – V25-V28

In summary, through EDA we have discovered several insights that would aid us in training three separate models accurately. Through univariate analysis we noticed that our data is heavily imbalanced and that our amount and time variables have disproportionate shapes in comparison to the rest of the variables, bivariate analysis helped us learn when fraudulent transactions happen, their occurrence pattern and which transaction amounts are most likely to be fraudulent. And through multivariate analysis we learned which variables are most likely to be weak or strong predictors.

2.0 Data preprocessing (data cleaning and balancing (SMOTE))

2.1 Sampling and SMOTE

In data mining, sampling refers to the selection of subsets of data points from the overall dataset. Sampling is used to reduce computational cost, handle imbalanced datasets and improve generalization (M et al., 2025). Especially for classification in predictive modelling, sampling is utilized to address any class imbalance (Dewi et al., 2024). There are several different sampling techniques that can be implemented depending on the requirements of a given project.

Under sampling works by removing instances from the majority class to create a balanced class distribution. While it does reduce the model's potential bias towards the majority class, there is a risk of information loss when excluding those instances. By contrast over sampling is the random duplication of instances from the minority class to achieve a balanced dataset. This method preserves information while balancing classes, however there is a risk of the model overfitting due to repeat instances from the minority class.

2.1.1 Synthetic Minority Over-Sampling Technique (SMOTE)

Synthetic Minority Over-sampling Technique or SMOTE is an advanced over-sampling method. While standard over-sampling duplicates instances from the minority class, SMOTE works by creating synthetic instances based on instances from the minority class (B et al., 2025).

The general process that SMOTE follows to synthesize instances begins with the identification of the nearest k neighbours in the feature space for all instances within the minority class. Meaning that it will determine a specified number of instances (determined by k) that are most similar to the given instance, measured using Euclidean distance. Using the identified neighbours, it will generate new instances by interpolating between the original instance and its neighbours. This process will repeat until the desired class proportions are achieved (Asha et al., 2025).

With SMOTE generating new instances instead of duplicating existing one like in normal over-sampling the risk of overfitting is greatly reduced. This instance generation exposes the model to more varied instances of the minority class during training, building a classifier that is more familiar with patterns characteristic of rare events.

2.2 Normalization

Normalization refers to the transformation of numerical features such that they can be placed onto a shared scale or bounded range. This is generally done through the rescaling of feature values so that they fall within a desired numeric range, in most cases this range is between 0 and 1, this transformation removes differences in units or magnitude between features allowing all features to contribute to model training equally.

Normalization is generally used in implementations of machine learning algorithms that are sensitive to the scale of different features, improving the overall training efficiency and performance. Algorithms such as K-Nearest Neighbours, Support Vector Machines, and gradient-based optimizers in neural networks often compute distances or gradients that can be heavily influenced by the relative magnitudes of input variables. If features are not normalized, variables with larger scales can dominate the learning process and bias the model's results.

The convergence of a model can also be sped up through normalization, the model is provided with training data that is more uniform and numerically stable, in turn this leads to fewer training iterations, lower computational costs and improved prediction accuracy.

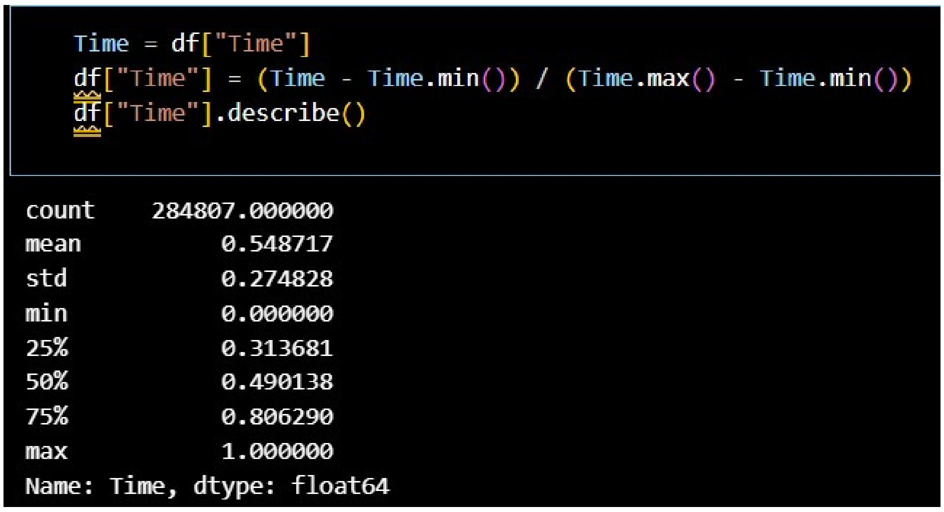
2.3 RobustScaler

The RobustScaler is a preprocessing technique that scales numerical features by removing the median and scaling according to the interquartile range (IQR), unlike traditional scalers that rely on minimum and maximum values, mean and standard deviation, RobustScaler uses statistics that are by their nature less sensitive to outliers. It ensures that the majority of the data is rescaled to a consistent range that is unaffected by extreme values by subtracting the median to centre the feature and divides it by the IQR to scale it. The result is that most values, after transformation, will fall between -1 and 1.

RobustScaler is used in scenarios where there are outliers or extreme values within the dataset that could negatively impact the performance of the model or other preprocessing. By relying on the median and IQR RobustScaler is essentially ignoring these outliers and their potential influence on the dataset, allowing for the stable and reliable scaling of the central portion of the distribution. This is particularly useful for machine learning algorithms that are sensitive to differences in feature magnitudes but can still tolerate outliers in the input data without requiring their removal. It offers a balance, providing the benefits of scaling while maintain robustness against extreme values, this is especially useful in situations where perfect data cleaning is not feasible such as with real world data.

2.4 Data Cleaning

Using the insights uncovered from the EDA phase, we can clean up the dataset so that we can have strong base for next phase, model training. Firstly, is the shape of the Time variable, the data starts at 0 and ends at 172,000, that's too large and will affect our training negatively. To address this, the Time variable will normalized. Figure 15 shows the formula used, (Time – minimum time) / (maximum time – minimum time). This formula transforms the data from 0 to 1; making it similar to the shape of the remaining variables. The function df.describe() is then used to show the new changes.

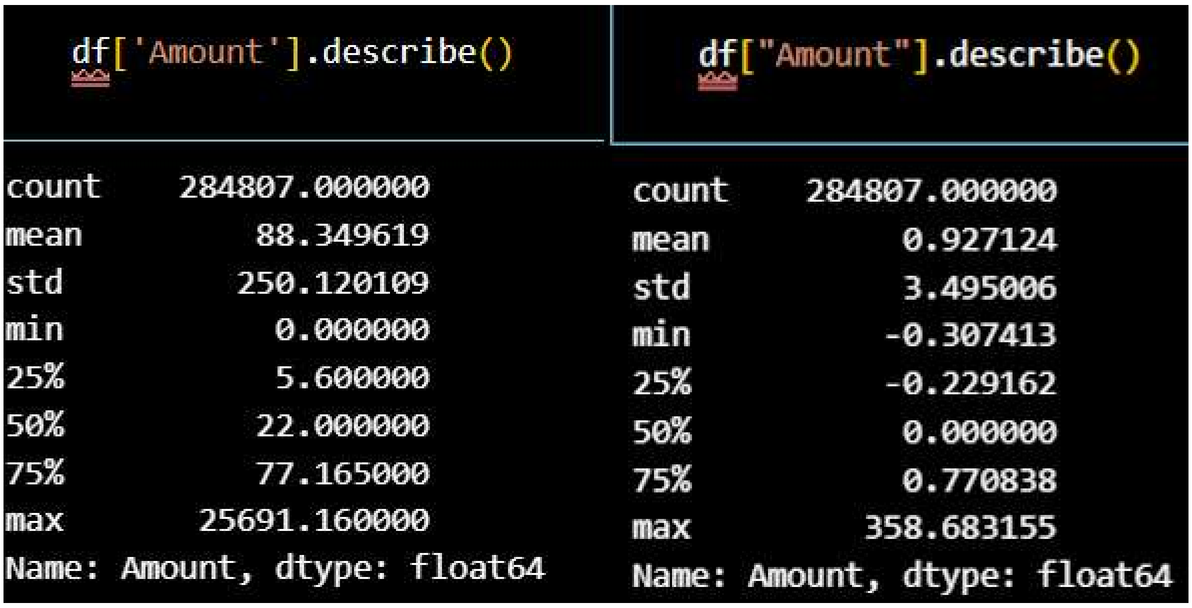


```
Time = df["Time"]
df["Time"] = (Time - Time.min()) / (Time.max() - Time.min())
df["Time"].describe()
```

```
count    284807.000000
mean          0.548717
std           0.274828
min           0.000000
25%           0.313681
50%           0.490138
75%           0.806290
max           1.000000
Name: Time, dtype: float64
```

Figure 15 – Time normalization

The second variable that needs to be transformed is the Amount, as seen during EDA it has large outliers and a mismatched shape. To address this, Robust scaler is used to reshape the data to a (-1,1) shape similar to the anonymized variables, this will reduce the effect of the outliers on our data without the need of deleting them. Figure 16 shows the pre and post transformation.



```
df['Amount'].describe()
```

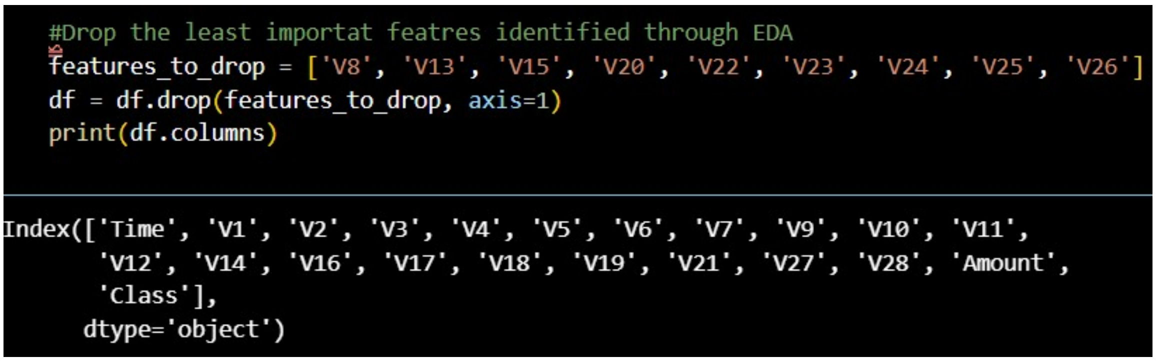
```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

```
df["Amount"].describe()
```

```
count    284807.000000
mean          0.927124
std           3.495006
min          -0.307413
25%          -0.229162
50%           0.000000
75%           0.770838
max         358.683155
Name: Amount, dtype: float64
```

Figure 16 – Amount before and after scaling

Secondly, the variables that were found to be weak predictors need to be removed, by removing unnecessary variables our training process becomes less intensive. The variables that are found to be weak are, V8, V13, V15, V20, V22, V23, V24, V25, V26.Figure 17 shows how that is done.

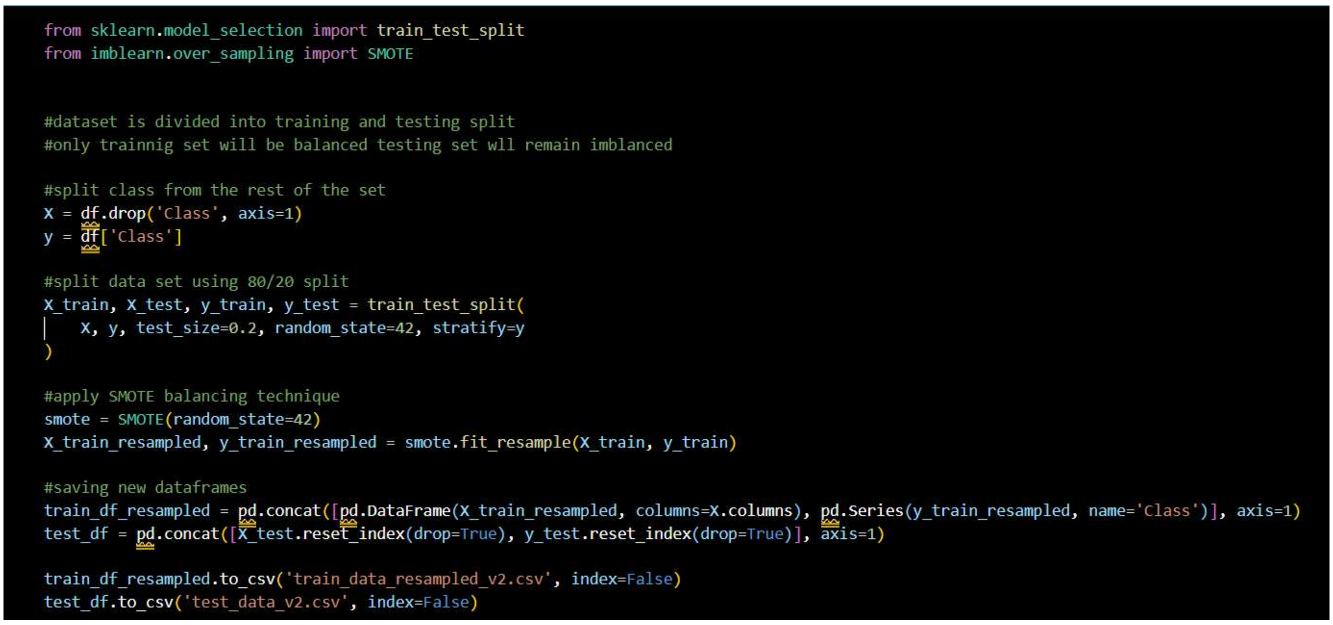


```
#Drop the least importat featres identified through EDA
features_to_drop = ['V8', 'V13', 'V15', 'V20', 'V22', 'V23', 'V24', 'V25', 'V26']
df = df.drop(features_to_drop, axis=1)
print(df.columns)
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V9', 'V10', 'V11',
       'V12', 'V14', 'V16', 'V17', 'V18', 'V19', 'V21', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Figure 17 – Dropping features

Finally, is balancing the dataset, in the EDA phase we discovered that the ratio of fraudulent and non-fraudulent transaction is too big 99.83% to 1.7%. such an imbalance can lead to a biased model. To address this we will be using the SMOTE balancing technique, but before we do that our dataset needs to be split into a training set and testing set, the testing set will be used to validate our model after training. Because of that it needs to remain imbalanced as that will produce a more accurate result, if it were to be tested on a balanced data set then that would not replicate a true real life scenario. Figure 18 will show the process from data splitting to balancing and saving and figure 19 will show the old vs new distributions. With pre-processing now done data training can begin.



```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE


#dataset is divided into training and testing split
#only trainnig set will be balanced testing set wll remain imblanced

#split class from the rest of the set
X = df.drop('Class', axis=1)
y = df['Class']

#split data set using 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

#apply SMOTE balancing technique
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

#saving new dataframes
train_df_resampled = pd.concat([pd.DataFrame(X_train_resampled, columns=X.columns), pd.Series(y_train_resampled, name='Class')], axis=1)
test_df = pd.concat([X_test.reset_index(drop=True), y_test.reset_index(drop=True)], axis=1)

train_df_resampled.to_csv('train_data_resampled_v2.csv', index=False)
test_df.to_csv('test_data_v2.csv', index=False)
```

Figure 18 – data balancing and splitting

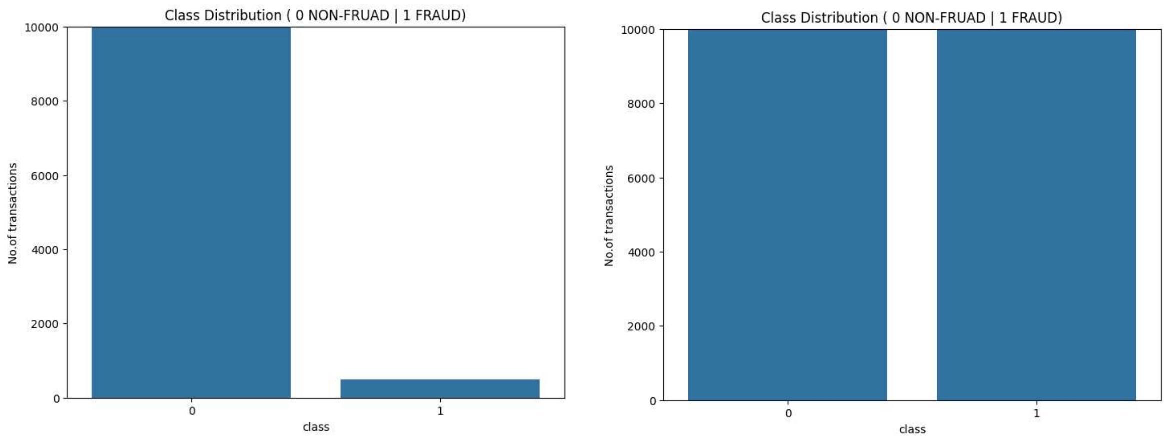


Figure 19 – old vs new distribution

3.0 Algorithm 1 Decision Tree

The first machine learning model that we implemented for this project was a decision tree, a supervised machine learning algorithm that can be utilized for both classification and regression. A decision tree works by splitting the dataset into branches based on feature values, with each internal node representing a condition on an attribute, each branch representing an outcome of that condition and the leaf node representing the final prediction (Hand & Fitkov-Norris, 2024).

The decision tree algorithm follows a clear process that begins with the selection of the best split. The algorithm will, at each node, evaluate all possible splits of all features to identify which one will result in the best separation of data. How exactly it evaluates the splits depends on whether it is being used for classification or regression, under classification Gini impurity or information gain is typically used to measure how pure the resulting subsets are. Mean Squared Error is typically used to evaluate the prediction error for splits in regression tasks. Splitting stops when one of the following conditions are met, the maximum depth is reached, all observations belong to the same class or the minimum number of samples per node is too small to split any further. Lastly, to make its prediction the algorithm will either take the majority class of the samples in the leaf node, in the case of classification tasks, or the mean of the target variable in the leaf node, for regression tasks.

We selected a Decision tree to be one of the algorithms used in this project due to its unique combination of transparency and versatility. The main detection approach described by Manimaran et al. (2024) relies on a Decision Tree, which is valued for providing clear, interpretable decision paths that stakeholders can easily understand and validate compared to more complex models. Datasets that include both numerical and categorical variables can also be seamlessly processed by decision trees without requiring extensive data transformation. These characteristics together makes the algorithm particularly adept in handling scenarios where model interpretability and flexibility to handle varying feature types are critical.

Another reason that compelled us to implement a Decision Tree is its ability to automatically perform feature selection and adapt dynamically to changing data. As Manimaran et al. (2024) highlight, Decision Trees can identify which variables are most informative for classification tasks, reducing the need for extensive manual feature engineering. The algorithms capacity to highlight key predictors not only improves model interpretability, but it also enhances efficiency through its focus on the most relevant information. Decision rules are also dynamic and can change over time to accommodate new data patterns as they emerge, ensuring accurate predictions even with changing environments. These qualities make them particularly well-suited for applications where data characteristics are expected to shift or expand over time (Manimaran et al., 2024).

3.1 Algorithm 1 Hyperparameters

Hyperparameter	Description
max_depth	Limits how deep the tree can grow. Shallower trees generalize better.
min_samples_split	Minimum number of samples required to split an internal node.
min_samples_lead	Minimum number of samples allowed in a leaf node.
Criterion	The function used to evaluate the quality of a split (e.g., Gini impurity or entropy).
Max Features	Limits the number of features considered when looking for the best split.

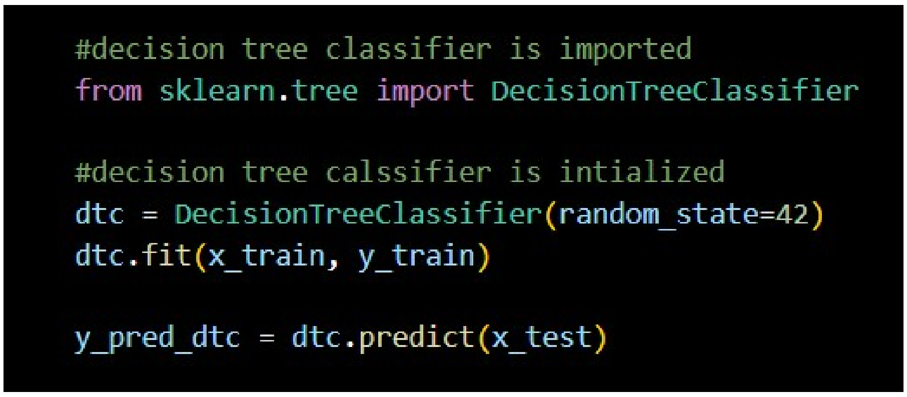
3.2 Grid Seach CV

Grid Search Cross-Validation or Grid Search CV is a class within the Scikit-learn library used to determine the most optimal combination of hyperparameters, the combination of hyperparameters that will result in the best performance, when tuning a machine learning model. This optimal combination is identified by creating a grid of all possible hyperparameter values, grid search will then thoroughly assess every combination using cross-validation to measure performance (Mangkunegara & Purwono, 2022). During training a machine learning model cannot determine the optimal hyperparameters based on the data it is being trained on. Thus, selecting which hyperparameters to use is essential to achieving the best model accuracy and generalisation.

Grid Search CV begins with the definition of the hyperparameter grid. Next, all possible combinations of hyperparameters from the grid are determined. Using k-fold cross-validation the model is then trained and evaluated on each of the combinations. The combination of hyperparameters that resulted in the best model performance are then selected. This method of determining the most optimal combination of hyperparameters ensures that all possible combinations are evaluated, leaving no room for a potentially better combination to be overlooked (Brindha et al., 2025).

3.3 Implementation

For the first run, we used the default hyperparameter settings, as shown in figure 20, the first run was used to gauge the model, the second run will utilize the grid search function to fine tune the model and get best result possible.



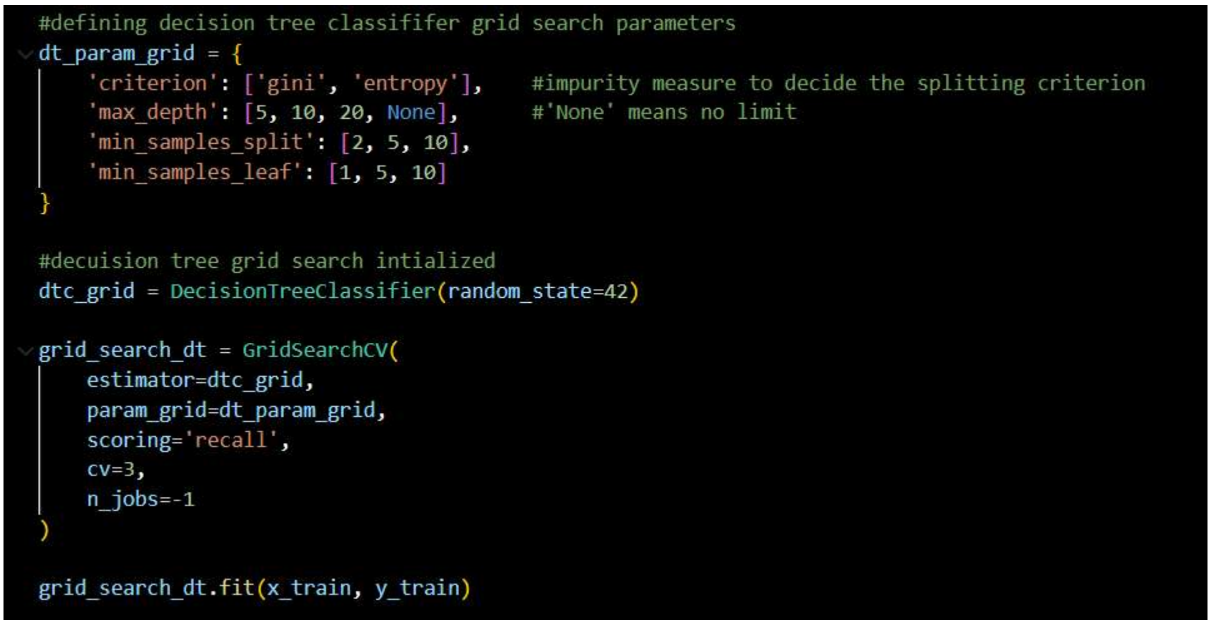
```
#decision tree classifier is imported
from sklearn.tree import DecisionTreeClassifier

#decision tree calssifier is intialized
dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(x_train, y_train)

y_pred_dtc = dtc.predict(x_test)
```

Figure 20 – Decision tree initializing (default)

Figure 21 will show the parameter grid that will be used for grid search training. This grid search will go through several hyper parameters and the best one can be extracted at the end. For "splitting criterion" two methods will be tried, gini index and entropy. "max_depth" will have 4 choices, "5, 10, 20, and None", none means there is no limit to the depth. "min_samples_leaf/split" will have three choices (1, 5, 10) and (2, 5, 10) respectively.



```
#defining decision tree classifer grid search parameters
dt_param_grid = {
    'criterion': ['gini', 'entropy'],    #impurity measure to decide the splitting criterion
    'max_depth': [5, 10, 20, None],      #'None' means no limit
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10]
}

#decuision tree grid search intialized
dtc_grid = DecisionTreeClassifier(random_state=42)

grid_search_dt = GridSearchCV(
    estimator=dtc_grid,
    param_grid=dt_param_grid,
    scoring='recall',
    cv=3,
    n_jobs=-1
)

grid_search_dt.fit(x_train, y_train)
```

Figure 21 - Decision tree initializing (gridsearch)

4.0 Algorithm 2 Random Forest

The second machine learning model that we decided to implement for this project was a Random Forest model. Random Forest models are ensemble learning algorithms that create and combine multiple decision trees to produce a single prediction that is more accurate and stable (Brindha et al., 2025). While individual decision trees may suffer from overfitting, the combination of multiple trees that are trained on different data subsets will produce a collective decision that is more robust and accurate, improving the end models generalization.

Random Forest models offer several advantages, such as its robustness to overfitting. As mentioned previously the variance that would normally make individual decision trees unstable is mitigated through the averaging of multiple trees (Taye et al., 2025). Aggregation of these trees also dilutes the influence that noisy data might have on the final model, compared to how a few outliers or misclassified instances can strongly influence the predictions of a single tree. The diverse outputs that the algorithm receives from its many trees allows it to capture complex, non-linear relationships in data without overfitting, achieving good generalization.

We selected Random Forest as one of our algorithms for this project for its proven ability to handle datasets with many input variables without suffering from multicollinearity issues, a statistical phenomenon where two or more independent variables become strongly correlated. Especially in scenarios where real-world data is involved, this is particularly important. Additionally, Random Forest naturally models nonlinear relationships between features and target variables, allowing it to capture complex patterns that simpler algorithms might miss. As noted in previous research, Random Forest algorithms are well-suited for problems involving many predictor variables because they are not impacted by multicollinearity and can effectively model complex, nonlinear associations between inputs and outputs (Chiaverini et al., 2023).

In addition to its handling of several predictors and complex model relationships, the model has been widely recognised for its consistent performance in a variety of different applications. With some researchers even characterizing Random Forest as perhaps the most effective general-purpose algorithm available (Hand & Fitkov-Norris, 2024).

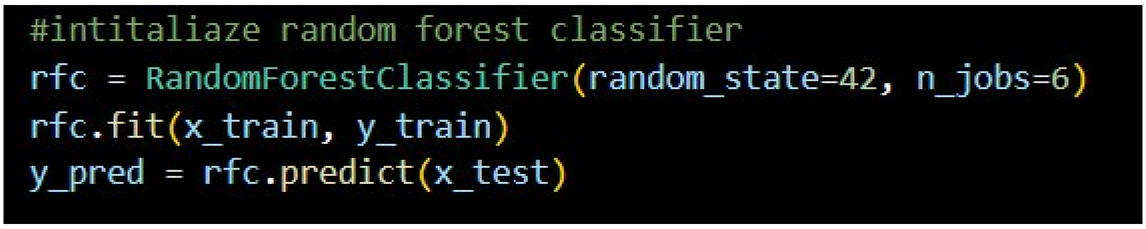
As explored by Asha et al. (2025), Random Forest partitions data based on threshold comparisons rather than relying on the magnitude of features, which means it does not require standardized or normalized input data to function effectively. This inherent insensitivity to feature scaling further supported our choice of Random Forests in our project. This insensitivity simplifies the preprocessing workflow and reduces any risk of reduced performance caused by inconsistent scaling. Additionally, Random Forest consistently achieved excellent predictive metrics regardless of the scaling method applied, underscoring its versatility and robustness in handling diverse datasets (Asha et al., 2025).

4.1 Algorithm 2 Hyperparameters

Hyperparameter	Description
n_estimators	How many trees the model builds. More trees improve stability but increase training time.
max_depth	The maximum number of levels each tree is allowed to grow. Shallower trees reduce overfitting risk but might underfit.
min_samples_split	Minimum number of samples required to split an internal node. Increasing this can make trees simpler.
min_samples_leaf	Minimum samples that must be at a leaf node.
max_features	Number of features considered when looking for the best split.

4.2 Implementation

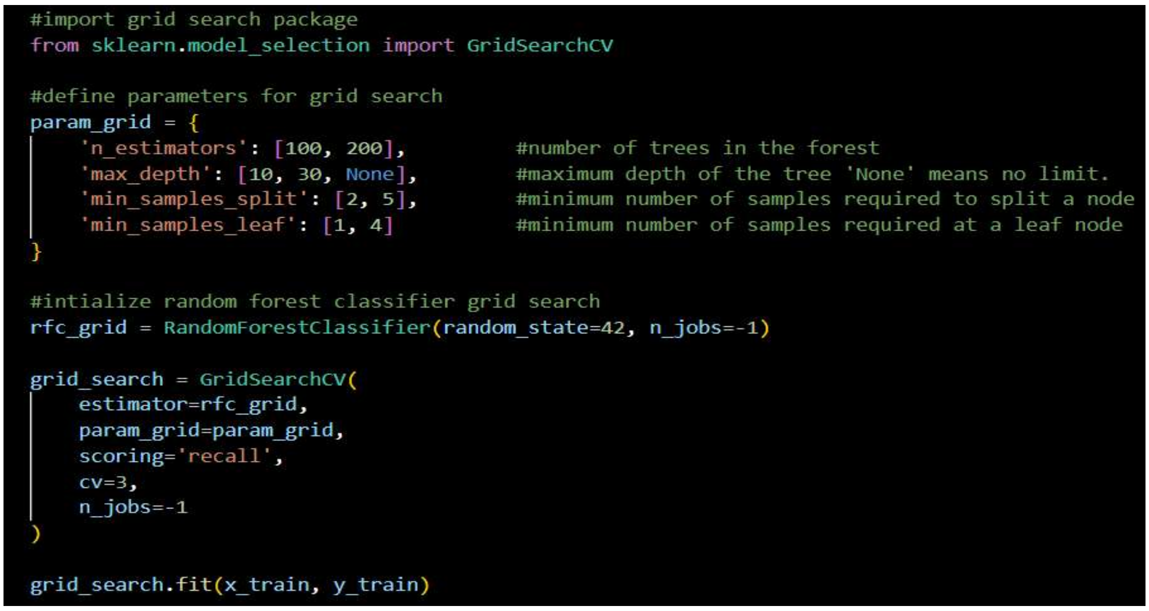
Following the same logic as the previous model, figure 22 will show the initial model.



```
#intitaliaze random forest classifier
rfc = RandomForestClassifier(random_state=42, n_jobs=6)
rfc.fit(x_train, y_train)
y_pred = rfc.predict(x_test)
```

Figure 22 – Random forest initializing (default)

Figure 23 will show the grid search parameters and model. For this grid search, there are two choice for “n_estimators” (100 and 200), for “max_depth” it will have three choices (10, 30 and none), “min_samples_split/leaf” will both have two choices (2,5) (1,4) respectively.



```
#import grid search package
from sklearn.model_selection import GridSearchCV

#define parameters for grid search
param_grid = {
    'n_estimators': [100, 200],          #number of trees in the forest
    'max_depth': [10, 30, None],         #maximum depth of the tree 'None' means no limit.
    'min_samples_split': [2, 5],         #minimum number of samples required to split a node
    'min_samples_leaf': [1, 4]           #minimum number of samples required at a leaf node
}

#intialize random forest classifier grid search
rfc_grid = RandomForestClassifier(random_state=42, n_jobs=-1)

grid_search = GridSearchCV(
    estimator=rfc_grid,
    param_grid=param_grid,
    scoring='recall',
    cv=3,
    n_jobs=-1
)

grid_search.fit(x_train, y_train)
```

Figure 23 - Random forest initializing (grid search)

5.0 Algorithm 3 XG boost

Extreme Gradient Boosting (XGBoost) is an advanced ensemble learning method based on gradient boosting decision trees. In XGBoost trees are built sequentially with each new tree correcting errors made by previous trees. The algorithm is designed to be highly efficient, scalable and accurate (Babu et al., 2024).

XGBoost models offer several advantages, such as their ability to achieve high predictive performance across a wide range of problems. Unlike simpler models that may struggle to capture complex interactions, XGBoost combines the outputs of many sequential trees, each one focused on correcting the errors of the previous iteration. This boosting process often results in better accuracy compared to individual decision trees or even Random Forests. Another strength of XGBoost is its built-in regularization, which applies penalties to overly complex trees. These penalties help control overfitting by discouraging splits or leaf weights that do not contribute meaningfully to reducing the loss. In addition, XGBoost is designed to handle missing data automatically by learning which direction to send missing values during training, reducing the need for extensive preprocessing. Lastly, XGBoost is highly efficient, as it includes optimizations for speed and memory usage that allow it to train quickly even on large datasets. This combination of strong performance, regularization, and computational efficiency makes it a popular choice for structured data tasks in both academic research and business applications.

XGBoost was also selected for this project because of its combination of strong predictive capabilities and advanced interpretability methods. As Yuan et al. (2025) demonstrate, pairing XGBoost with SHAP analysis allows practitioners to understand precisely how individual features impact predictions, making the model’s decisions transparent and actionable. In many data mining applications this level of transparency is critical, especially when stakeholders must trust and validate model results before implementing decisions. The consistently high performance of XGBoost across a variety of scenarios reinforces its value as a robust algorithm suitable for complex classification and regression tasks that require both accuracy and interpretability (Yuan et al., 2025).

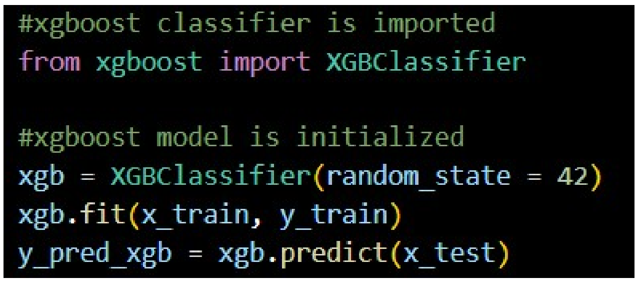
The algorithm’s ability to balance model complexity and predictive performance through integrated regularization was another important factor in our selection of it for this project. As Guo et al. (2024) note, unlike many conventional algorithms, XGBoost incorporates both L1 and L2 regularization terms that automatically constrain overly complex models. This mechanism allows the algorithm to be highly effective in reducing the chances of overfitting, especially when working with noisy datasets where the chances of overfitting are increased. By controlling complexity, XGBoost is better able to generalize its predictions to new, unseen data, making it a reliable choice for robust classification and regression tasks across a variety of domains (Guo et al., 2024).

5.1 Algorithm 3 Hyperparameters

Hyperparameters	Description
n_estimators	The number of boosting rounds (trees).
max_depth	The maximum depth of each tree. Deeper trees can capture more complex patterns but risk overfitting.
learning_rate	Controls how much each tree contributes to the model. Smaller values often improve performance but require more trees.
subsample	The fraction of the training data used to grow each tree, introducing randomness to improve generalization.
colsample_bytree	The fraction of features used per tree, like Random Forest feature sampling.
gamma	Minimum loss reduction required to make a split; higher values make the model more conservative.
Lambda and alpha	L2 and L1 regularization terms on weights.

5.2 Implementation

And for the final model, it will be displayed in figure 24 an the grid search will be shown in figure 25.

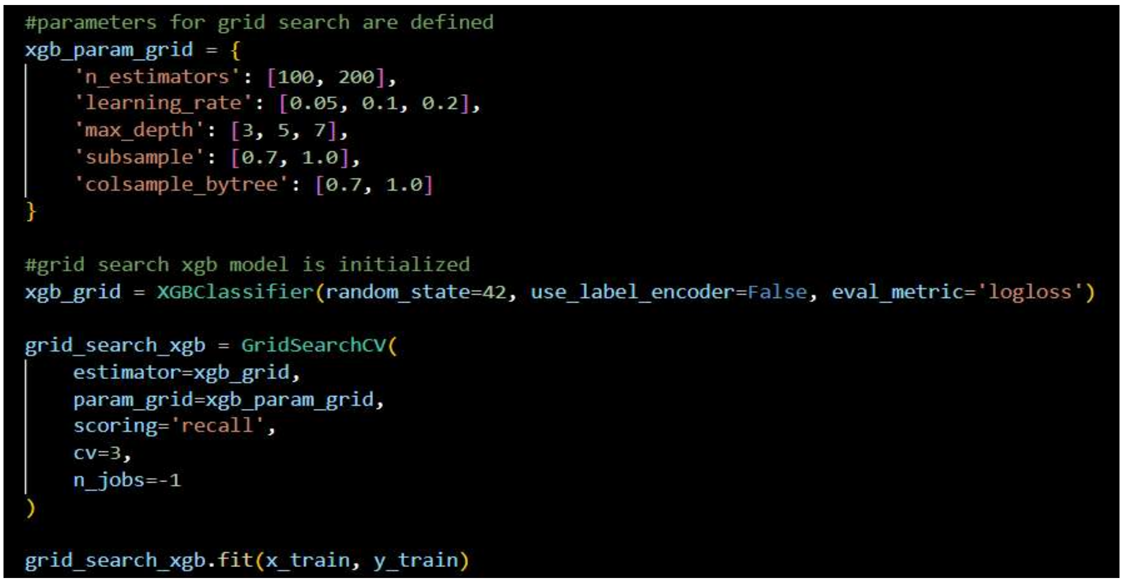


```
#xgboost classifier is imported
from xgboost import XGBClassifier

#xgboost model is initialized
xgb = XGBClassifier(random_state = 42)
xgb.fit(x_train, y_train)
y_pred_xgb = xgb.predict(x_test)
```

Figure 24 - XGboost initializing (default)

This grid search also has multiple hyperparameters, for "n_estimators' we have two choices (100 and 200), "learning_rate" we have three choices (0.05, 0.1 and 0.2), 'max_depth" also has three choices (3, 5, and 7), "sub_sample" has two choices (0.7 and 1.0) and the final one, 'colsample_bytree' has two choices (0.7 and 1.0).



```python
#parameters for grid search are defined
xgb_param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 1.0],
    'colsample_bytree': [0.7, 1.0]
}

#grid search xgb model is initialized
xgb_grid = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

grid_search_xgb = GridSearchCV(
    estimator=xgb_grid,
    param_grid=xgb_param_grid,
    scoring='recall',
    cv=3,
    n_jobs=-1
)

grid_search_xgb.fit(x_train, y_train)
```

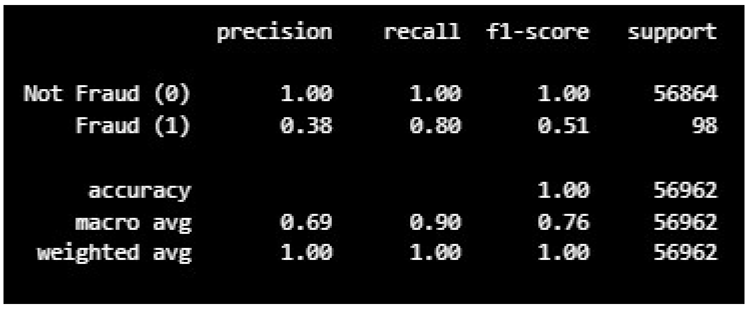
Figure 25 - XGboost initializing (gridsearch)

6.0 Result and discussion

In this section we will discuss the results of each model and grid search, they will be evaluated using a confusion matrix, it will show us the recall, precision, accuracy and F1-score. The models will then be ranked from best to worst based on performance.

6.1 Model 1 decision tree results

Figure 26 will show the results before tuning the hyper parameters, this is the result from the default settings of the decision tree classier. We can see that it achieved a very low precision score but a moderately a high recall rate. This means that model was able to catch 80% of actual fraud cases but when the model flags a fraud cases it is only correct 38% of the time. Overall the model has a 51% f1-score which is being drastically dragged down due to the poor precision. Figure 27 is a confusion matrix of the results.



```
               precision    recall  f1-score   support

Not Fraud (0)       1.00      1.00      1.00     56864
    Fraud (1)       0.38      0.80      0.51        98

     accuracy                           1.00     56962
    macro avg       0.69      0.90      0.76     56962
 weighted avg       1.00      1.00      1.00     56962
```

Figure 26 – Default Decision Tree result

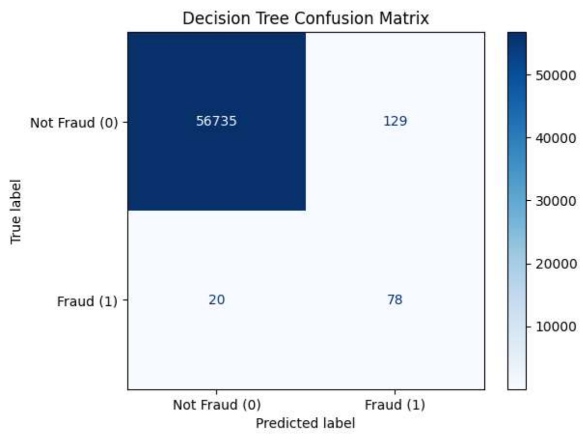
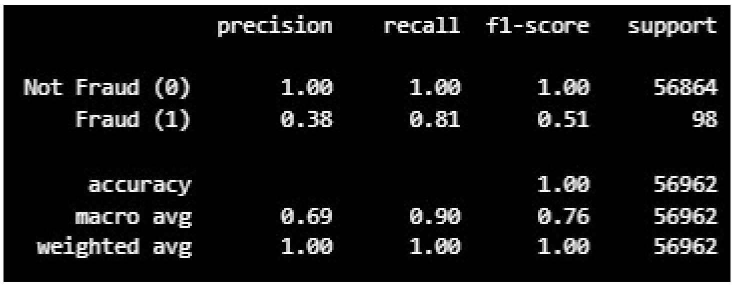


Figure 27 - Default Decision Tree CM

After grid search the best model was found to be with these hyperparameters < 'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, figure 28 displays the results from the tuned model. From the scores we can see that the recall has improved by 1% while the precision remained the same. Overall, the model did not improve by much, still at 51% f1-score. Figure 29 visualizes the confusion matrix.



```
               precision    recall  f1-score   support

Not Fraud (0)       1.00      1.00      1.00     56864
    Fraud (1)       0.38      0.81      0.51        98

     accuracy                           1.00     56962
    macro avg       0.69      0.90      0.76     56962
 weighted avg       1.00      1.00      1.00     56962
```

Figure 28 - Tuned Decision Tree result

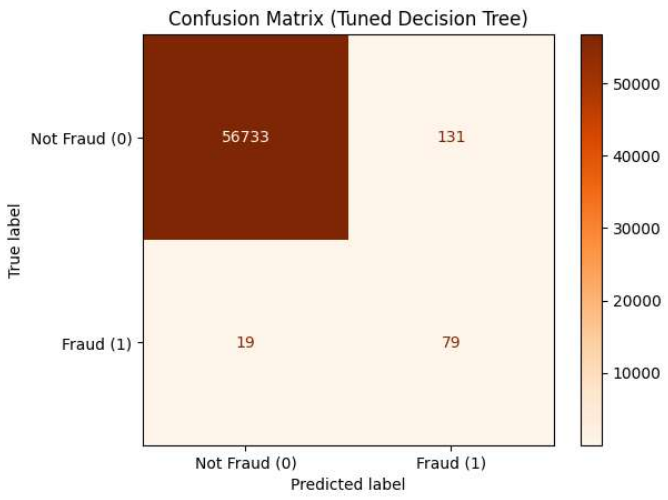
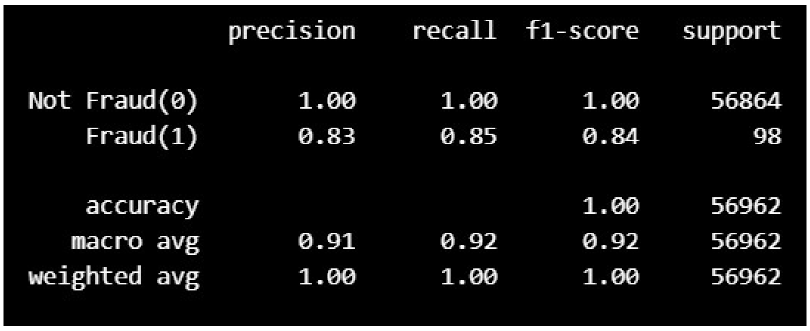


Figure 29 - Tuned Decision Tree CM

6.2 Model 2 Random Forest results

Figure 30 shows the results from the default random classifier model. This model has significantly better recall than the decision tree model, we achieved a precision of 83% and a recall of 85% bring the f1-score to 84% which is quite good, signifying that the model is able to pick up on the insights in the dataset. Figure 31 will visualize the confusion matrix.



```
              precision    recall  f1-score   support

Not Fraud(0)       1.00      1.00      1.00     56864
    Fraud(1)       0.83      0.85      0.84        98

    accuracy                           1.00     56962
   macro avg       0.91      0.92      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

Figure 30 - Deafult Random Forest result

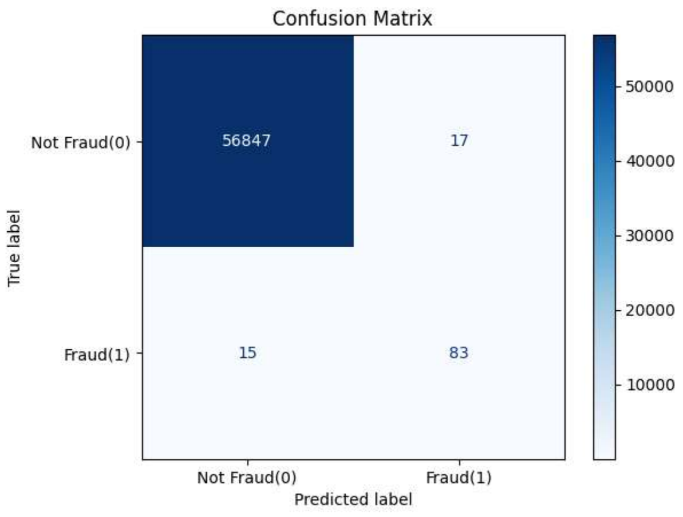
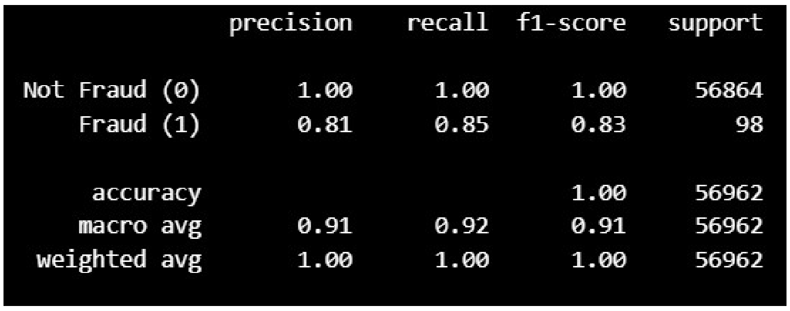


Figure 31 – Default Random Forest CM

After grid search the best model was the one with the following hyperparameters, 'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200, figure 32 shows the results from model validation. This model had a similar result with no significant improvement, meaning that the default model was enough to retrieve all the possible insights from the dataset, this puts the random forest classifier at the top. Figure 33 visualizes the confusion matrix,



```
              precision    recall  f1-score   support

 Not Fraud (0)       1.00      1.00      1.00     56864
     Fraud (1)       0.81      0.85      0.83        98

      accuracy                           1.00     56962
     macro avg       0.91      0.92      0.91     56962
  weighted avg       1.00      1.00      1.00     56962
```

Figure 32 - Tuned Random Forest result

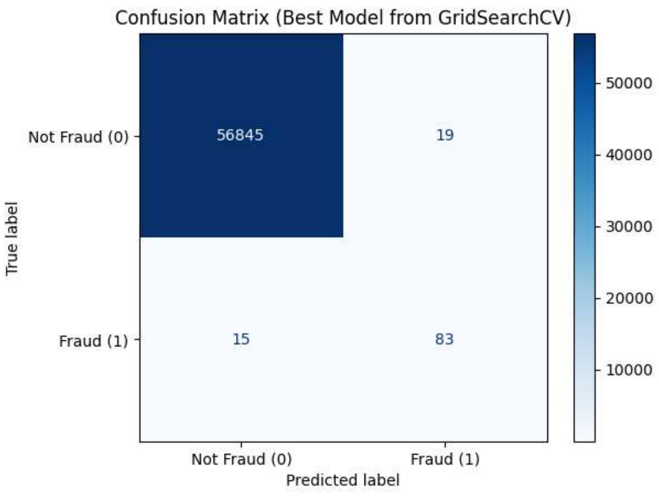
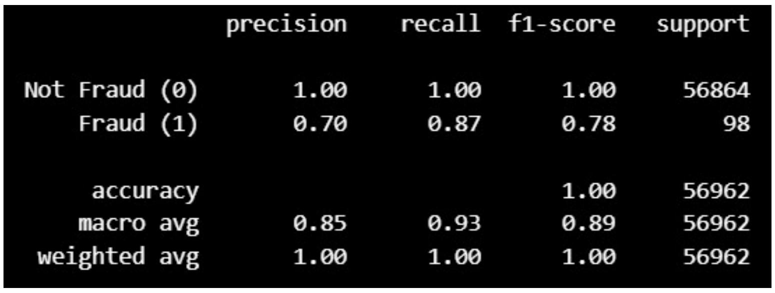


Figure 33 - Tuned Random Forest CM

6.3 Model 3 XGboost results

Figure 34 shows the results from the default XGboost model, it achieved the highest recall with 87% but the second highest precision with 70% brining the overall f1-score to 78%. The recall rate was amazing but similarly to the decision tree model the precision brough its overall performance lower. Figure 35 visualizes the confusion matrix.



```
              precision    recall  f1-score   support

 Not Fraud (0)       1.00      1.00      1.00     56864
     Fraud (1)       0.70      0.87      0.78        98

      accuracy                           1.00     56962
     macro avg       0.85      0.93      0.89     56962
  weighted avg       1.00      1.00      1.00     56962
```

Figure 34 - Default XGboost result

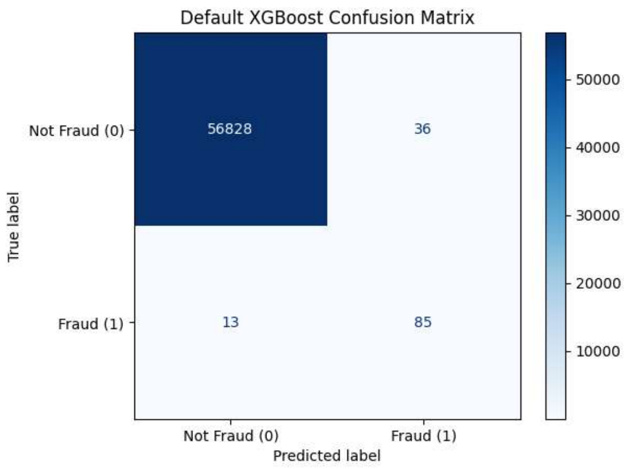
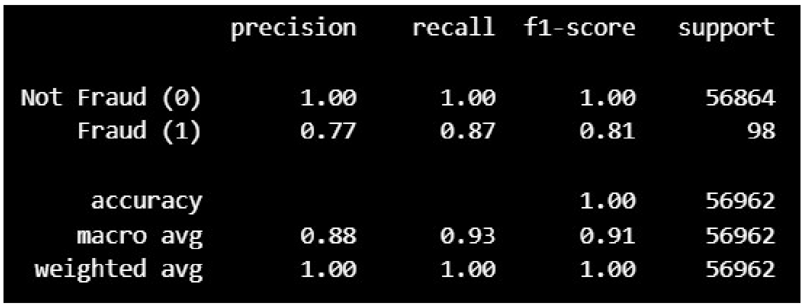


Figure 35 – Default XGBoost CM

After tuning the hyperparameters the best 'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 200, 'subsample': 1.0, figure 36 shows the results from the best model. This model manages to increase the precision to 77% which a significant improvement while maintating the same recall rate, bringing the overall f1-score to 81% only 2% away from beating the random forest classifier. Figure 37 visualizes the confusion matrix.



```
               precision    recall  f1-score   support

Not Fraud (0)       1.00      1.00      1.00     56864
    Fraud (1)       0.77      0.87      0.81        98

     accuracy                           1.00     56962
    macro avg       0.88      0.93      0.91     56962
 weighted avg       1.00      1.00      1.00     56962
```

Figure 36 – Tuned XGboost result

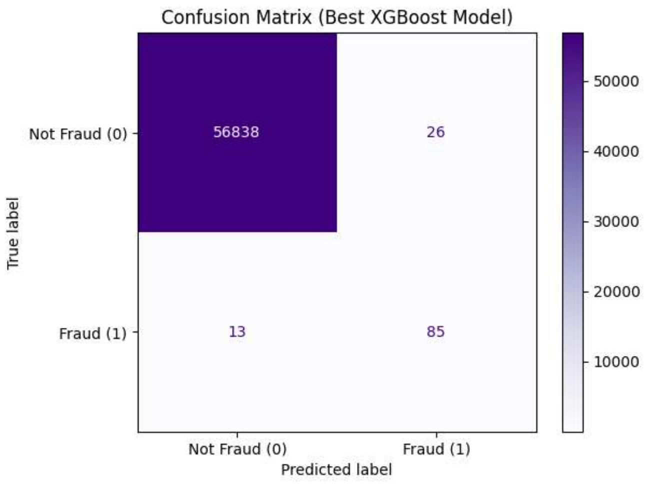


Figure 37 – Tuned XGBoost CM

6.4 Results conclusion

Based on the results of both default and tuned models, the random forest classifier successfully outperformed the remaining two models. Achieving the highest F1-score of 0.84 with the default model and 0.83 with the tuned model. This showcases a strong balance between recall and precision rates making it the most reliable model out of all three. Next up is XGboost model, it achieved an F1-score of 0.78 with the default model and 0.81 after tuning, though its F1-score is lower, it still managed to score better on recall rates with 0.87, but its precision was much lower at 0.70. This indicates that while XGboost was effective at classifying fraud, it came at the expense of precision. At the final place comes the decision tree classifier, it scored the worst in all categories with a precision score of 0.38 and a recall of 0.80.

Table 1 - Default model results

Placing	Model/performance metric	Precision	Recall	F1-score
1	Random Forest	0.83	0.85	0.84
2	XGboost	0.70	0.87	0.78
3	Decision Tree	0.38	0.80	0.51

Table 2 -Grid search model results

placing	Model/performance metric	Precision	Recall	F1-score
1	Random forest classifier	0.81	0.85	0.83
2	XGboost	0.77	0.87	0.81
3	Decision tree	0.38	0.81	0.51

Conclusion

During this project we have implemented and evaluated several machine learning models to address the challenge of credit card fraud detection in highly imbalanced dataset. The comparative analysis that we conducted demonstrated that ensemble methods, particularly Random Forest, achieved the most balanced performance with a combination of good precision and recall delivering superior performance compared to both Decision tree and XGBoost algorithms. While XGBoost delivered the highest recall, displaying its ability to identify fraudulent transactions, it also required extensive parameter tuning and was less interpretable than simpler models. By comparison, Decision Tree offered transparency but lower accuracy. Our findings present the trade-offs between model complexity, interpretability and performance. Although we found that the implemented models show promise in supporting proactive fraud prevention strategies, future research could explore more sophisticated techniques and real-time validation to improve robustness and generalizability. Overall, our project highlights the values of integrated data preprocessing and ensemble learning in solving complex business problems involving rare event prediction.

References

Asha, V., Vasumathi, M. T., Prasad, A., V, Y., P, Y. A., & Sivani, M. (2025). Evaluation of ML Models using SMOTE and Feature Scaling for Intrusion Detection System(IDS). *International Conference on Visual Analytics and Data Visualization (ICVADV-2025)*, 243–249. https://doi.org/10.1109/icvadv63329.2025.10961506

B, G., P, J., G, K., M, I. N., Mohanarathinam, N., & Velusamy, J. (2025). Optimized SMOTE for Imbalanced Data Handling in Machine Learning. *2025 3rd International Conference on Advancement in Computation & Computer Technologies (InCACCT)*, 349–354. https://doi.org/10.1109/incacct65424.2025.11011340

Babu, M., S, N. R., Moharir, M., & Mohana, N. (2024). Leveraging XGBoost Machine Learning Algorithm for Common Vulnerabilities and Exposures (CVE) Exploitability Classification. *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, 1–6. https://doi.org/10.1109/csitss64042.2024.10816942

Brindha, P., Boobesh, R., & Yokanandh, S. S. (2025). Optimization of ML Algorithms in CAD Diagnosis Using Grid Search CV. *Third International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT-2025)*, 2229–2234. https://doi.org/10.1109/idciot64235.2025.10914959

Chiaverini, L., Macdonald, D. W., Hearn, A. J., Kaszta, Ż., Ash, E., Bothwell, H. M., Can, Ö. E., Channa, P., Clements, G. R., Haidir, I. A., Kyaw, P. P., Moore, J. H., Rasphone, A., Tan, C. K. W., & Cushman, S. A. (2023). Not seeing the forest for the trees: Generalised linear model out-performs random forest in species distribution modelling for Southeast Asian felids. *Ecological Informatics*, *75*, 102026. https://doi.org/10.1016/j.ecoinf.2023.102026

Dewi, O. I. P., Santiko, V. N., Safa, S. W. P., Gunawan, A. a. S., & Setiawan, K. E. (2024). Machine Learning for Imbalanced Data in Telecom Churn Classification. *2024 International Conference on Information Technology Research and Innovation (ICITRI)*, 30–35. https://doi.org/10.1109/icitri62858.2024.10699226

El-Assady, M., Sevastjanova, R., Sperrle, F., Keim, D., & Collins, C. (2017). Progressive Learning of Topic Modeling Parameters: a Visual Analytics framework. *IEEE Transactions on Visualization and Computer Graphics*, *24*(1), 382–391. https://doi.org/10.1109/tvcg.2017.2745080

Guo, Y., Chen, M., Yuan, C., & Zheng, F. (2024). Research on Campus Network Security Situational Awareness Technology Based on XGBoost Machine Learning. *2024 5th International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, 948–952. https://doi.org/10.1109/icbase63199.2024.10762217

Hand, C., & Fitkov-Norris, E. (2024). Not seeing the wood for the trees: Influences on random forest accuracy. *International Journal of Market Research*, *66*(5), 559–566. https://doi.org/10.1177/14707853241255469

He, Q., & Yu, X. (2024). A novel feature extraction approach for predicting human essential genes by XGBoost machine learning. *2024 9th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, 855–860. https://doi.org/10.1109/iciibms62405.2024.10792770

Keim, D. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, *8*(1), 1–8. https://doi.org/10.1109/2945.981847

M, Y. V., P, K. K., Ajam, S. M., Kumar, S. H., & Hareshvar, S. (2025). Comprehensive survey on exploratory data analysis and machine learning approaches for lung cancer detection. *Third International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT-2025)*, 1791–1797. https://doi.org/10.1109/idciot64235.2025.10914897

Mangkunegara, I. S., & Purwono, P. (2022). Analysis of DNA Sequence Classification Using SVM Model with Hyperparameter Tuning Grid Search CV. *2022 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, 427–432. https://doi.org/10.1109/cyberneticscom55287.2022.9865624

Manimaran, A., GnanaJeyaraman, R., J, C. M. B. M., M, S., Kannan, E., & Sivaram, M. (2024). An Adaptive Framework for Low-Rate DDoS Detection in Cloud Environments Using Decision Tree Machine Learning Algorithm. *2024 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*, 1–5. https://doi.org/10.1109/icbds61829.2024.10837242

Pandey, P., & Garg, K. K. (2025). Credit Card Fraud Detection Using KNC, SVC, and Decision Tree Machine Learning Algorithms. *4th IEEE International Conference on AI in Cybersecurity (ICAIC)*, 1–3. https://doi.org/10.1109/icaic63015.2025.10848573

Pusuluri, V. B., Prasad, A. M., & Darimireddy, N. K. (2023). Decision-Tree Based Machine Learning Approach for the Design and Optimization of 5G n78 Sub-Band Antenna for WiMAX/WLAN Applications. *2022 IEEE Wireless Antenna and Microwave Symposium (WAMS)*. https://doi.org/10.1109/wams57261.2023.10242820

Taye, E. A., Woubet, E. Y., Hailie, G. Y., Zegeye, A. T., Arage, F. G., Zerihun, T. E., & Kassaw, A. T. (2025). Random forest algorithm for predicting tobacco use and identifying determinants among pregnant women in 26 sub-Saharan African countries: a 2024 analysis. *BMC Public Health*, *25*(1). https://doi.org/10.1186/s12889-025-22794-1

Yuan, Z., Xue, Y., Wang, H., Qu, S., Huang, C., Wang, H., Zhang, H., Zhang, M., & Xing, X. (2025). A predictive model for hospital death in cancer patients with acute pulmonary embolism using XGBoost machine learning and SHAP interpretation. *Scientific Reports*, *15*(1). https://doi.org/10.1038/s41598-025-02072-1

Zaman, A., Khan, S. A., Mohammad, N., Ateya, A. A., Ahmad, S., & ElAffendi, M. A. (2025). Distributed denial of service attack detection in Software-Defined networks using decision tree algorithms. *Future Internet*, *17*(4), 136. https://doi.org/10.3390/fi17040136

Zhao, Q., Li, J., Diao, Z., Zhang, X., Feng, S., Hou, G., Xu, W., Zhao, Z., Qiu, Z., Yang, W., Zhou, S., Tian, P., Zhang, Q., Chen, W., Li, H., Xiao, G., Qin, J., Hu, L., Li, Z., . . . Zhang, R. (2025). Early prediction of preeclampsia from clinical, multi-omics and laboratory data using random forest model. *BMC Pregnancy and Childbirth*, *25*(1). https://doi.org/10.1186/s12884-025-07582-4

Cruz, B. (2025, January 27). 62 million Americans experienced credit card fraud last year. Security.org. https://www.security.org/digital-safety/credit-card-fraud-report/

Dhanshetti, P. (2023). Techniques of exploratory data analysis. Madhya Pradesh Journal of Social Sciences, 28(2). https://doi.org/10.13140/RG.2.2.13578.03522

Hafez, I. Y., Hafez, A. Y., Saleh, A., El-Mageed, A. a. A., & Abohany, A. A. (2025). A systematic review of AI-enhanced techniques in credit card fraud detection. Journal of Big Data, 12(1). https://doi.org/10.1186/s40537-024-01048-8

Khalid, A. R., Owoh, N., Uthmani, O., Ashawa, M., Osamor, J., & Adejoh, J. (2024). Enhancing Credit Card Fraud Detection: An ensemble Machine learning approach. Big Data and Cognitive Computing, 8(1), 6. https://doi.org/10.3390/bdcc8010006

Milo, T., & Somech, A. (2020). Automating Exploratory data analysis via Machine Learning: An Overview. SIGMOD, 2617–2622. https://doi.org/10.1145/3318464.3383126