VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi - 590 018, Karnataka

# Credit Card Fraud Detection Predictive Models

*A Report submitted in partial fulfillment of the requirements for the Course*

**Mini with Project**

**(Course Code: 24AM5PWMPW)**

*In the Department of*

**Machine Learning**

**(UG Program: B.E. in Artificial Intelligence and Machine Learning)**

By

**Srujan B J , Sri Harsha K , Srinidhi S Mattur , Pranav**

(USN: **1BM22AI134 , 1BM22AI131 , 1BM22AI132 , 1BM22AI090**)

5th Semester B Section

Under the Guidance of

**Dr. Vinutha H**

Assistant Professor

Dept. of MEL, BMSCE, Bengaluru – 19

**DEPARTMENT OF MACHINE LEARNING**

**B.M.S COLLEGE OF ENGINEERING**

*(An Autonomous Institute, Affiliated to VTU)*

P.O. Box No. 1908, Bull Temple Road, Bengaluru - 560 019

**December - 2024**

# B.M.S COLLEGE OF ENGINEERING

*(An Autonomous Institute, Affiliated to VTU)*

P.O. Box No. 1908, Bull Temple Road, Bengaluru - 560 019

## DEPARTMENT OF MACHINE LEARNING

## CERTIFICATE

This is to certify that Mr. / Ms. ***Srujan B J , Sri Harsha K , Srinidhi S Mattur , Pranav*** bearing USN: ***1BM22AI134 , 1BM22AI131 , 1BM22AI132 , 1BM22AI090 has*** satisfactorily presented the Course – ***Mini with Project***
(Course code: **24AM5PWMPW**) with the title ***"Credit Card Fraud Detection Predictive Models"*** in partial fulfillment of academic curriculum requirements of the 5th semester UG Program – B. E. in Artificial Intelligence and Machine Learning in the Department of Machine Learning, BMSCE, an Autonomous Institute, affiliated to Visvesvaraya Technological University, Belagavi during December 2024. It is also stated that the base work & materials considered for completion of the said course is used only for academic purpose and not used in its original form anywhere for award of any degree.

**Student Signature**

**Signature of the Supervisor**

**Dr. Arun Kumar N**
Assistant Professor, Dept. of MEL, BMSCE

**Signature of the Head**

**Dr. M Dakshayini**
Prof. & Head, Dept. of MEL, BMSCE

### External Examination

**Examiner Name and Signature**

**1.**

**2.**

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

In the current digital economy, credit card fraud is a widespread and expanding problem. The likelihood of fraudulent activity rises with the number of online transactions and electronic payments, posing serious hazards to financial institutions and consumers alike. In addition to causing significant financial losses, fraudulent credit card transactions erode customer trust in electronic payment systems. Therefore, the finance industry has set a priority on developing fraud detection systems that are trustworthy and efficient. The dynamic nature of the fraud itself is the greatest challenge to detecting credit card fraud. Because of the fact that fraudulent transactions are often sparse compared to authentic ones, datasets are highly imbalanced. Classic machine learning algorithms can struggle to identify the minority class correctly(fraudulent transactions) without significant fine-tuning and adjustments because of this imbalance. In addition, since scammers tend to be constantly adapting their strategies, detection systems need to be able to quickly adapt and maintain high accuracy in the long run. The aim of this research is to develop a robust machine learning model that can effectively detect fraudulent credit card transactions in order to resolve these problems. The project involves managing class imbalances, exploring various classification algorithms, and evaluating the models based on relevant performance metrics. Identifying a model, or combination of models, that is able to generate good predictions with minimal false positives and false negatives is the ultimate goal in an effort to reduce financial losses and enhance security.

## 1.1 Dataset Description

The credit card transaction dataset used for this project is openly accessible and comprises transactions done by European cardholders during two days in September 2013. There are 284,807 transactions in all in this dataset, 492 of which are deemed fraudulent. Due to confidentiality concerns, the features comprise numerical inputs derived from a Principal Component Analysis (PCA) transformation. There are 31 columns in the dataset:

- Time: The number of seconds that passed between each transaction and the dataset's initial transaction.
- The main components derived from PCA are V1 through V28.
- V1 to V28: The principal components obtained from PCA.

- Amount: The sum of the transaction.
- Class: The target variable, in which a genuine transaction is represented by 0 and a fraudulent transaction by 1.

The extremely unbalanced structure of the dataset fraudulent transactions make up just 0.172% of the total—highlights the necessity of using advanced classification methods to accurately identify the minority class. Achieving a careful balance between identifying fraudulent transactions and reducing false alarms which can result in needless operating expenses and frustration for genuine customers is essential for effective fraud detection systems.

## 1.2 Aims and Objectives

The Main Aim of the project is **"Creating a scalable, reliable machine learning model that can identify fraudulent credit card transactions in highly unbalanced datasets, guaranteeing high recall, accuracy, and precision while adjusting to new fraud strategies to reduce losses."**

The primary objectives of the project are as follows:

- **Data Exploration and Preprocessing:** To comprehend the dataset's distribution, structure, and any underlying trends, thoroughly examine it. Carry out the required preprocessing actions, including resolving class imbalances, scaling features, and handling missing values.

- **Model Selection and Training:** Investigate a variety of classification algorithms, such as decision trees, logistic regression, group methods such as random forests, support vector machines (SVM), and k-nearest neighbors (KNN). To obtain optimal results, train these tune models on the dataset and tune their hyperparameters.

- **Addressing Class Imbalance:** To balance the data set, employ methods such as Near Miss under-sampling and Synthetic Minority Over-sampling Technique (SMOTE). The aim of these methods is to enhance the ability of the model to detect fraudulent transactions while keeping valid ones classified.

- **Performance Evaluation:** Apply appropriate metrics to evaluate the models, such as accuracy, precision, recall, F1-score, and the area under the Receiver Operating Characteristic (ROC) curve. Metrics that prove the accuracy of the model in detecting the minority class will receive special emphasis as a result of the imbalance of the dataset.

- **Model Comparison and Selection:** To find the best model to use in identifying fraud, contrast the performance of a number of models. Consider factors such as ease of application, scalability, and computational effectiveness when selecting the final model.

- **Implementation of a Fraud Detection System:** Build a pipeline that integrates model training, prediction, and data preprocessing in order to identify suspicious transactions in real-time. It must be able to adapt to new information as well as shifting fraud patterns.

By achieving these objectives, the project aims to complement other activities in preventing credit card fraud and to provide financial institutions with a reliable tool for safeguarding their customers and assets. The experience gained from this research will also impart valuable lessons on handling unbalanced datasets and the application of machine learning techniques to real-world problems.

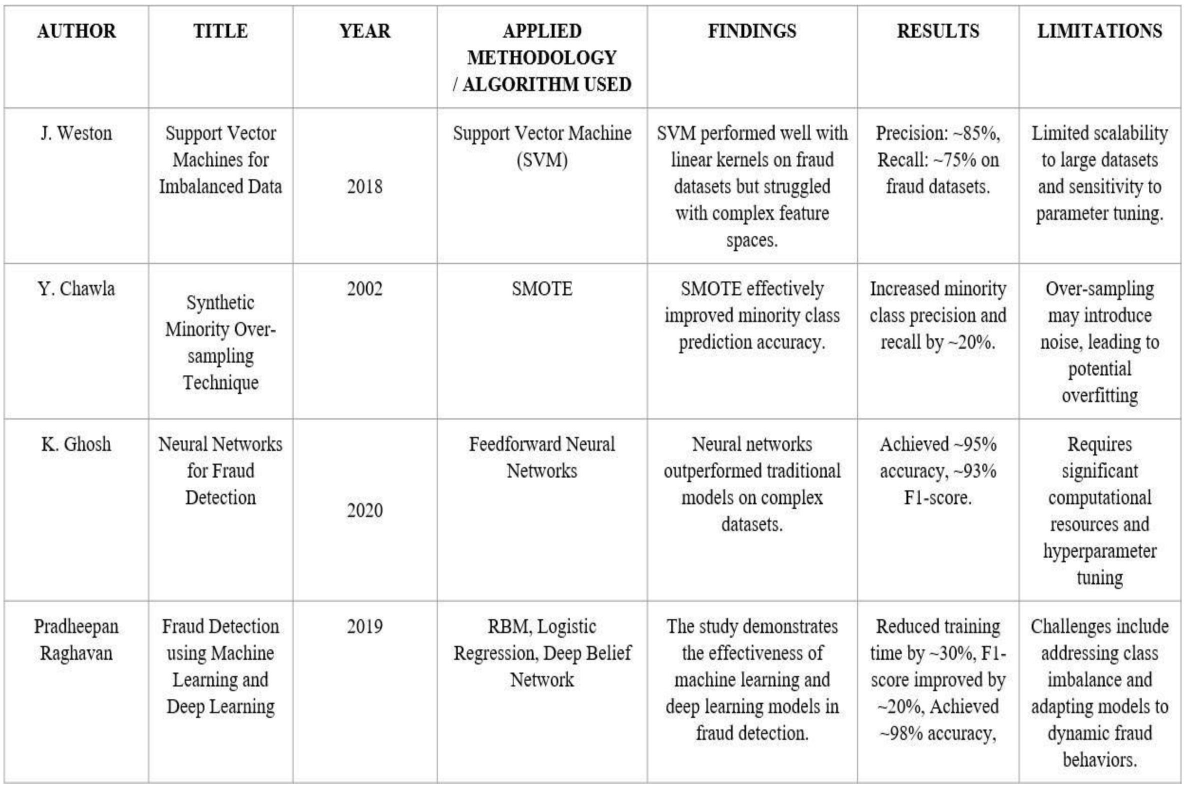
# CHAPTER 2

# LITERATURE REVIEW

Through the years, numerous methods and strategies have been proposed under the extensively researched domain of credit card fraud detection. Focusing on traditional methodologies, machine learning methodologies, and dataset unbalancing management, the present literature review emphasizes leading works and findings under the research topic. Expert systems and rule-based methods constituted crucial elements in pioneering fraud detection solutions. To recognize suspicious transactions, these systems utilized pre-programmed patterns and rules. Although simple to implement, rule-based systems have numerous disadvantages, such as a high rate of false positives and a failure to adapt to new fraud patterns. The requirement for more dynamic and responsive systems became apparent as scammers refined their techniques. Recent studies have examined the application of deep learning techniques for fraud detection, such as neural networks and autoencoders. While these methods require vast amounts of data and computing power, they can detect complex patterns in the data.

Furthermore, hybrid strategies that incorporate several models have demonstrated potential in raising detection rates and lowering false positives.

Here are a few reviews that have been made:



| AUTHOR | TITLE | YEAR | APPLIED METHODOLOGY / ALGORITHM USED | FINDINGS | RESULTS | LIMITATIONS |
|---|---|---|---|---|---|---|
| J. Weston | Support Vector Machines for Imbalanced Data | 2018 | Support Vector Machine (SVM) | SVM performed well with linear kernels on fraud datasets but struggled with complex feature spaces. | Precision: ~85%, Recall: ~75% on fraud datasets. | Limited scalability to large datasets and sensitivity to parameter tuning. |
| Y. Chawla | Synthetic Minority Over-sampling Technique | 2002 | SMOTE | SMOTE effectively improved minority class prediction accuracy. | Increased minority class precision and recall by ~20%. | Over-sampling may introduce noise, leading to potential overfitting |
| K. Ghosh | Neural Networks for Fraud Detection | 2020 | Feedforward Neural Networks | Neural networks outperformed traditional models on complex datasets. | Achieved ~95% accuracy, ~93% F1-score. | Requires significant computational resources and hyperparameter tuning |
| Pradheepan Raghavan | Fraud Detection using Machine Learning and Deep Learning | 2019 | RBM, Logistic Regression, Deep Belief Network | The study demonstrates the effectiveness of machine learning and deep learning models in fraud detection. | Reduced training time by ~30%, F1-score improved by ~20%, Achieved ~98% accuracy, | Challenges include addressing class imbalance and adapting models to dynamic fraud behaviors. |

Here is the Summary of the reviews made:

[1]: **J. Weston**: Support Vector Machines (SVMs) may maximize margin bounds, they have shown remarkable success in classification tasks. But conventional SVMs handle the margin consistently across training samples, which might have drawbacks, especially when there are outliers or unbalanced data. To address this problem, the proposed Adaptive Margin SVM (AM-SVM) implements adaptive margins where the margin is tailored for each training example. The outlier-robustness of AM-SVM is proven through generalization error theoretical bounds. The efficiency and reliability of AM-SVMs are established by experimental comparison on UCI benchmark datasets that reveal that they achieve generalization errors comparable to traditional SVMs.

[2]: **Y Chawla:** One of the common methods of class imbalance resolution in machine learning is the Synthetic Minority Over-sampling Technique (SMOTE). SMOTE resamples the dataset to balance it by interpolating among the current points to generate synthetic samples for minority classes. The technique ensures better representation of minority classes, and this improves the performance of the classifier. SMOTE is a critical technique for enhancing model accuracy and robustness in actual-world imbalanced datasets, as evidenced by the many studies that have validated its effectiveness in various fields.

[3]: **K. Ghosh:** Especially in money-related domains such as credit card transactions, neural networks are employed to evaluate and identify fraud. Ghosh's work demonstrates how neural networks can effectively recognize complex patterns and anomalies in data that traditional methods can overlook. His work provides trustworthy techniques for real-time fraud detection using advanced neural networks, with promising results in terms of increasing detection rates and reducing false positives.

[4]: **Pradheepan Raghavan:** Since fraudulent activities are dynamic and random, this research seeks to compare some of the latest machine learning and deep learning methods for fraud detection. An extensive approach to fraud mitigation is given by merging more traditional methods such as k-nearest neighbor (KNN), random forest, and support vector machines (SVM) with newer innovations such as autoencoders, convolutional neural networks (CNN), and restricted Boltzmann machines (RBM). A varied assessment is ensured by using datasets from Germany, Australia, and Europe (EU). The accuracy, dependability, and cost-effectiveness of these models in identifying fraudulent transactions are evaluated using measures like Area Under the ROC Curve (AUC), Matthews Correlation Coefficient (MCC), and Cost of Failure.

# CHAPTER 3

## OPEN ISSUES

Even though credit card fraud detection has advanced significantly, there are still a number of unresolved problems that pose difficulties for researchers and practitioners in the field:

- **Evolving Fraud Tactics:** Fraudsters are always coming up with new ways to get around detection systems. Because of their dynamic nature, models must be regularly updated and retrained in order to identify new trends in fraudulent activity.
- **Data Privacy and Security:** Privacy issues arise when real-world transactional data is used. Privacy and security of sensitive data need to be maintained in designing and deploying fraud detection systems.
- **Scalability and Real-time Processing:** In an attempt to offer timely warnings without sacrificing speed, fraud detection systems need to be able to process large amounts of data in real-time as the number of transactions increases.
- **False Positives and Consumer Experience:** Reducing false positives is necessary to prevent inconvenience to real customers. Low client experience and loss of trust in the detection system can be the result of high false positive rates.
- **Imbalanced Data Problems:** In spite of all the methods that have been suggested for handling class imbalance, there exists a problem in obtaining the optimal balance between sensitivity (recall) and specificity (accuracy), especially in highly imbalanced data sets.
- **Integration with Existing Systems:** Integrating fraud detection models with existing financial systems and processes can be difficult; compatibility, scalability, and maintainability all must be considered with caution.
- **Interpretability of Models:** Deep learning and other sophisticated machine learning models are "black boxes." It is necessary to make such models more interpretable both for regulatory reasons and to gain stakeholders' trust.
- **Adversarial Attacks:** One of the new challenges that need to be studied further is the issue of how susceptible fraud detection systems are to adversarial attacks, where fraudsters manipulate inputs to trick models.

To be able to give confidence that fraud detection systems are going to continue offering good, solid protection against future threats, it is necessary that these open problems be solved.

**CHAPTER 4**

# PROBLEM STATEMENT

**“The most significant challenge is "Developing a scalable, stable machine learning model capable of identifying credit card fraudulent transactions in highly unbalanced datasets with high recall, accuracy, and precision and evolving to new fraud patterns to reduce losses."** This challenge has a number of significant dimensions:

- **Robustness**: To be able to differentiate between true and false transactions at all times, even when presented with minute variations, the model must be able to cope with the natural noise and variability of the transaction data.
- **Scalability:** In order to enable rapid detection of fraud activity without leading to delays in transaction processing, the detection system should be able to process a very high volume of transactions in real-time.
- **Adaptability:** The model should be able to learn to adapt to new behavior and patterns as fraud patterns change. This involves ongoing learning and model updating to include new data.
- Accuracy, Precision, and Recall: A balance between these metrics must be obtained. While high precision and recall guarantee fraudulent transactions are identified with low false positives and false negatives, high accuracy guarantees most transactions are correctly labeled.
- Minimizing Financial Losses: The model can assist in minimizing financial losses to banks and customer inconvenience by minimizing false positives (erroneously labeled transactions) and false negatives (false negatives).
- Handling Imbalanced Data: Improving the performance of the model on the minority class requires the use of sophisticated techniques to overcome class imbalance, such as resampling, cost-sensitive learning, and anomaly detection.

By developing a machine learning-based fraud detection model that improves the precision of identifying fraudulent transactions, learns from changing fraud patterns, and performs well in real-time environments, this research intends to address these issues. The project aims to improve the effectiveness and reliability of fraud detection systems in the financial industry by using innovative methods to address class imbalance and including sound evaluation metrics.

# CHAPTER 5

# PROPOSED ARCHITECTURE

The credit card fraud detection project system design can be divided into several significant components, all of which are important to the pipeline. The top-level design is as follows:

## 5.1 System Overview

Data ingestion, preprocessing, fraud detection, and report modules constitute the system. It predicts based on a machine learning model. It is created such that data moves through each step effortlessly with immense accuracy and negligible lag from raw transaction data to meaningful insights.

## 5.2. Components

### 5.2.1 Input Layer

Sources:

- Historical Datasets: They are pre-existing transaction data already logged, and labels are tagged on fraudulent and genuine transaction instances. Machine learning algorithms need to train and validate themselves using historical datasets.
- Real-time Transaction Data: For real-time monitoring and forecasting, in a production environment, real-time data flows through online transaction gateways, mobile payment systems, and point-of-sale systems and are fed into the system for real-time monitoring and forecasting.

### 5.2.2. Data Preprocessing Layer

Data Cleaning:

- For data integrity, invalid or inconsistent data entries are removed. These include null value handling, data type adjustment, and duplication removal.

Normalization:

- Quantitative features like transaction amounts and time frames are normalized by using methods like Z-score normalization or Min-Max Scaling to make features consistent.

Class Balancing:

- For balancing the dataset, methods such as the SMOTE are used to create synthetic samples of the minority class or under sample the majority class. This is performed to make the model sensitive to the fraudulent transactions.

Feature Engineering:

- Developing new features that improve model learning, including:
    - Patterns of transaction frequency
    - Location-based patterns (e.g., IP address origin and geospatial anomalies)

### 5.2.3. Model Training Layer

Machine Learning Algorithms:

- Logistic Regression: One of the building paradigms for binary classification tasks is logistic regression.
- Restricted Boltzmann Machine (RBM): One generative model that helps in the discovery of hidden representations and patterns is the restricted Boltzmann machine (RBM).
- Deep Belief Network (DBN): One deep learning model that learns hierarchical features from data by stacking a few RBMs.
- Neural Networks: These are extremely helpful in deep learning techniques as they can detect complex patterns and relationships in data.

Dimensionality Reduction:

- By applying Principal Component Analysis (PCA) or other feature selection methods to concentrate on the most useful features and minimize computational complexity and noise.

Validation and Optimization:

- K-fold Cross-Validation: To provide robustness and minimize overfitting, the data is divided into k subsets for repeated training and validation of the model.
- Hyperparameter Optimization: To choose the best set of hyperparameters that optimize model performance, use Grid Search or Random Search.

Training Data:

- To ensure the model can properly differentiate between the two groups, it is trained on labeled data with both fraudulent and legitimate transactions.

### 5.2.4. Model Testing Layer

Testing Process:

- Test Dataset: Model performance is tested using another dataset that was not used in training.

- Performance Metrics:
    - Accuracy: Measures how well the model's predictions are correct overall.
    - Precision: The accuracy of the model in detecting fraud is represented in the ratio of true positive predictions to all predicted positives.
    - Recall (Sensitivity): The capability of the model to detect all fraud transactions is measured by the ratio of true positive predictions to actual positives.
    - F1-Score: The harmonic mean of precision and recall, which assigns weights to the two metrics.
    - ROC-AUC: The ability of the model to discriminate between genuine and fraudulent transactions is represented in the area under the Receiver Operating Characteristic curve.

Stress Testing:

- Model robustness and scalability testing through simulation of a variety of high-volume transaction scenarios and fraud inclinations.

Error Analysis:

- Model tuning for the least misclassification rates through analysis of false positives and false negatives to ascertain trends and points of optimization.

Iterative Improvements:

- For optimal performance before deployment, the model is iteratively optimized, retrained, and retested against test results.

### 5.2.5. Output Layer

Reports:

- Comprehensive Performance Metrics: Generating detailed reports on key metrics for fraud detection, such as:
    - False Positives: Genuine transactions reported as fraudulent.
    - False Negatives: Fraudulent transactions missed by the system.
    - F1-Score: The harmonic mean of the precision and recall, which provides a single measure for model evaluation.
    - ROC-AUC: The model's ability to discriminate between classes is represented in the area under the Receiver Operating Characteristic curve.

Visualization:

- Visual reports and dashboards for stakeholders to assist in strategy and decision-making through display of trends, detection rates, and operating insights.

Alerts and Notifications:

- In case of detection of a transaction as likely to be fraudulent, real-time notifications are dispatched to enable immediate response by automated systems or fraud professionals.

Continuous Monitoring:

- In order to keep pace with shifting fraud tactics, the system constantly monitors model performance, retraining and refreshing the model with every new set of data.

With a focus on precision, efficacy, and flexibility, the design of the architecture guarantees an integrated approach to fraud detection, from data ingestion through to actionable insights.

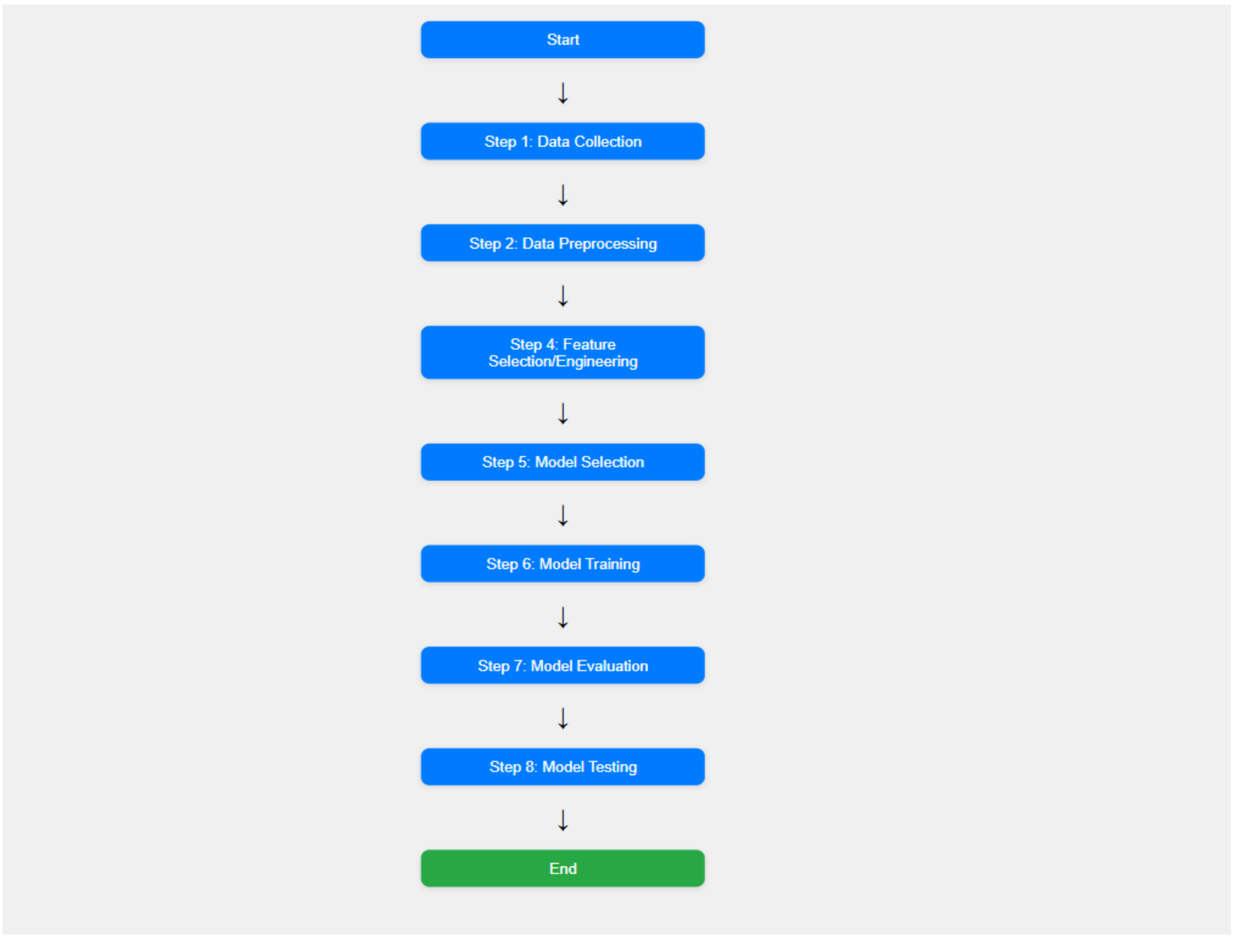


Figure 5.1: The Credit Card Predictive Model's Architecture

# CHAPTER 6

# FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

## 6.1 Functional Requirements

The following functionalities are facilitated by the credit card fraud detection system:

Data Preprocessing

- Preprocessing, Cleaning, and Normalizing Incoming Transaction Data Capability: For good data quality, the system should process raw transaction data by preprocessing, cleaning, and normalizing it before analyzing it.
- Handling Missing or Erroneous Data Values in the Dataset: To avoid model performance being skewed, the system needs to detect and manage missing or erroneous data values properly.

Fraud Detection

- Identifying Fraudulent Transactions with High Accuracy Using Machine Learning Models: The system should employ state-of-the-art machine learning techniques to detect fraudulent transactions with high accuracy.
- To offer Real-time Fraud Detection Capability for Online Transactions: In real-time environments, the system is required to present real-time fraud detection results so that potential losses can be prevented.

Model Training and Updates

- Allow Training Machine Learning Models Against Historical Transaction Data: To learn about patterns of fraudulent behavior, the system must allow training models against a great volume of historical data.
- Facilitate retraining on new data to adapt to new fraud patterns: As fraud patterns evolve, it's essential to learn and improve continuously; the system must provide model retraining on new data.

## 6.2 Non-Functional Requirements

These detail the overall features and constraints of the system:

Performance

- Processing Transactions with Minimal Latency to Support Real-time Fraud Detection: In order to detect fraud in real-time, the system needs to process high-speed data streams and provide timely insights.

- Handling Large Transaction Volumes at the Same Time: Without any compromise on performance, the architecture must be capable of processing enormous transaction volumes.

Scalability

- Vertically or Horizontally Scaling to Meet Growing Transaction Volumes: The system needs to scale its resources in order to meet growing demand.
- Enable Integration with Additional Data Sources or Modules: Staying abreast of new fraud techniques needs the flexibility of the system to integrate additional data sources or features as and when they become accessible.

Accuracy and Reliability

- Sustaining High accuracy and Recall to Minimize False Positives and False Negatives: In order to minimize misclassification and ensure that both fraudulent and legitimate transactions are appropriately classified, the system must achieve a balance between accuracy and recall.
- Ensure System Dependability to Minimize Downtime: Supporting continuous fraud defense necessitates dependability and availability.

Robustness

- Handling Noisy or Incomplete Input Data without Impacting Detection Performance: The system needs to be robust in dealing with defects in data in order to ensure credible fraud detection output.
- Recover Gently from Failures or Errors: Firm recovery methods should ensure speedy and least-intrusive resume of operations during system failure.

# CHAPTER 7

# LOW-LEVEL DESIGN

Low-Level Design focuses on the details of implementing the fraud detection system. It provides for a systematic problem solution approach through the breakdown of the system into small components. The key components of the low-level design are explained in the subsequent sections:

## 7.1. Data Preprocessing Pipeline

Handling Missing Values:

- Imputation Methods: Statistical values such as mean, median, or mode can be employed to impute missing data for continuous variables. It is a standard practice to use the mode for categorical variables.
- Record Deletion: Missing value records can be removed to maintain data quality when there is a large amount of missing data and imputation is not feasible.

Scaling Features:

- Normalization: To make all features have the same scale, the data is to be scaled to the range [0, 1]. This is particularly useful for algorithms that are sensitive to magnitude.
- Standardization: Scaling the data so that its standard deviation is one and mean is zero. This is often used when the data is normally distributed.

Encoding Categorical Variables:

- One-Hot Encoding: When there are several categorization levels, it is appropriate to create binary columns for every category.
- Label Encoding: Each category is given a distinct number, which is helpful in situations when there is an ordinal relationship.
- Frequency Encoding: When working with high-cardinality categorical characteristics, it is advantageous to use the frequency of categories as encoded values.

## 7.2. Feature Selection

Dimensionality Reduction (PCA):

- Principal Component Analysis (PCA): Diminishes the dimensionality of the dataset by lowering the features to a more manageable number of uncorrelated components without losing the maximum amount of variation. This improves model performance and reduces computational complexity.

Correlation Analysis:

- Feature Correlation Matrix: To prevent multicollinearity, which distorts model results and leads to overfitting, highly correlated features (above a certain threshold, e.g., 0.9) are detected and removed.

Feature Importance:.

- Model-based Selection: Retaining the most informative features involves employing feature importance ratings from models like Random Forest.
- Recursive Feature Elimination (RFE): Removing the least important features progressively based on model performance.

## 7.3. Model Training

Classifiers:

- Logistic Regression: Preferred due to its simplicity and interpretability, it is useful in understanding how input features are connected to the probability of fraud.
- Random Forest: An ensemble learning technique which can efficiently capture non-linear dependencies and interactions between features. Overfitting- and outlier-robust.
- Gradient Boosting Machines (GBM): Comprise algorithms which can catch complicated patterns accurately, like XGBoost or LightGBM, which often outperform their less complicated versions in detecting fraud.

Hyperparameter Tuning:

- Grid Search: Discovering the optimal model parameters through exhaustive search across an available parameter grid.
- Random Search: Discovering the optimal configurations faster by sampling the parameter space randomly is particularly useful for high-dimensional spaces.

Handling Class Imbalance:

- Synthetic Minority Over-sampling Technique (SMOTE): Balancing the data set by generating synthetic examples of the minority class.
- Class Weighting: Increasing sensitivity to fraud transactions by rebalancing the weights of classes in the loss function to give more importance to the minority class.

## 7.4. Evaluation

Cross-Validation:

- Stratified K-Fold Cross-Validation: Gives a better estimate of model performance by ensuring that the target classes are distributed representatively across each fold of the dataset.

- Leave-One-Out Cross-Validation (LOOCV): Through training on n-1 samples and testing on the single remaining one, LOOCV can help make effective use of data for smaller datasets.

Performance Metrics:

- Precision: It computes the True Positives over (False Positives) in order to calculate how good the positive predictions are.
- Recall (Sensitivity): It computes the model's ability to catch all the true positive cases (True Positives / (True Positives + False Negatives)).
- F1-Score: One value which combines precision and recall and is computed as the harmonic mean of the two.
- ROC-AUC: Indicates the ability of the model to distinguish between classes. A higher AUC is a better performance by the model.

Threshold Tuning:

- Decision Threshold Adjustment: modification of the cutoff for suspect transactions to balance recall against precision based on company requirements.
- Cost-sensitive Analysis: When setting thresholds, consider the costs of false positives (customer annoyance) versus false negatives (losses due to fraud).

Through systematic preprocessing, training of models, and evaluation, this method ensures precise, dependable, and scalable fraud detection. Every component is designed to operate as an entirety, boosting the system's effectiveness in the detection of fraud and minimizing false alarms and missed alarms.

# CHAPTER 8

# METHODOLOGY

The proposed method of credit card fraud detection employs machine learning models to successfully and accurately detect fraudulent transactions. The methodology is structured as follows:

## 8.1. Data Collection and Preprocessing

Data Source:

- Utilizing a dataset of credit card transactions containing instances of both fraudulent and legitimate transactions labeled.

Preprocessing Steps:

- Handling Anomalies or Missing Values:
    - To maintain dataset integrity, missing values are imputed via statistical methods (mean, median) or observations with high missing data may be removed.
    - To ensure data consistency, outlier detection methods are employed to detect and address anomalies.
- Normalization/Scaling:
    - Employing standardization or normalizing on numerical attributes to ensure consistency and improve machine learning model convergence.
- Addressing Class Imbalance:
    - Over-sampling Techniques:
        - SMOTE (Synthetic Minority Over-sampling Technique): For balancing the class distribution, synthetic samples are generated for the minority class.
    - Under-sampling Techniques:
        - Near Miss or Random Under-sampling: Selects representative samples carefully while minimizing the majority class sample number to balance the dataset.
- Dataset Splitting:
    - Splitting the dataset into test and training subsets (for instance, 70-30 split) in order to facilitate model training and performance evaluation.

## 8.2. Feature Engineering

Feature Analysis:

- Employing exploratory data analysis and correlation analysis, the most relevant features that impact fraud detection are identified and selected.

Dimensionality Reduction:

- PCA (Principal Component Analysis): This reduces the complexity of the model and decreases overfitting by reducing the number of features while keeping significant variance.

## 8.3. Model Selection

Machine Learning Algorithms:

- Logistic Regression:
    - Serves as a baseline model, with fast deployment and ease of interpretation.
- Support Vector Classifier (SVC):
    - Identifies the optimal hyperplane for class separation that performs optimally in high-dimensional spaces.
- Neural Networks:
    - Perfect for detecting subtle non-linear correlations, especially in large datasets.
- Decision Tree:
    - Splits data into subsets based on feature values, resulting in a comprehensible and visually pleasing tree-like structure.

## 8.4. Model Training and Validation

Training Techniques:

- Cross-validation:
    - Utilizing k-fold cross-validation reduces the likelihood of overfitting by making the model generalize across different subsets of data.
- Regularization:
    - Applying dropout in neural networks to prevent overfitting or L1/L2 regularization in logistic regression to penalize large coefficients.

## 8.5. Evaluation Metrics

Performance Metrics:

- Precision:
    - Measures the accuracy of positive predictions, which is critical for reducing false positives.
- Recall (Sensitivity):
    - Measures the ability to detect actual fraudulent cases, which is critical for minimizing false negatives.
- F1-Score:
    - The harmonic mean of recall and precision, which provides a balanced measure of performance.
- ROC-AUC (Receiver Operating Characteristic - Area Under Curve):
    - Evaluates the model's ability to distinguish between classes at varying thresholds, giving a complete performance overview.

## 8.6. Model Testing

Testing on Unseen Data:

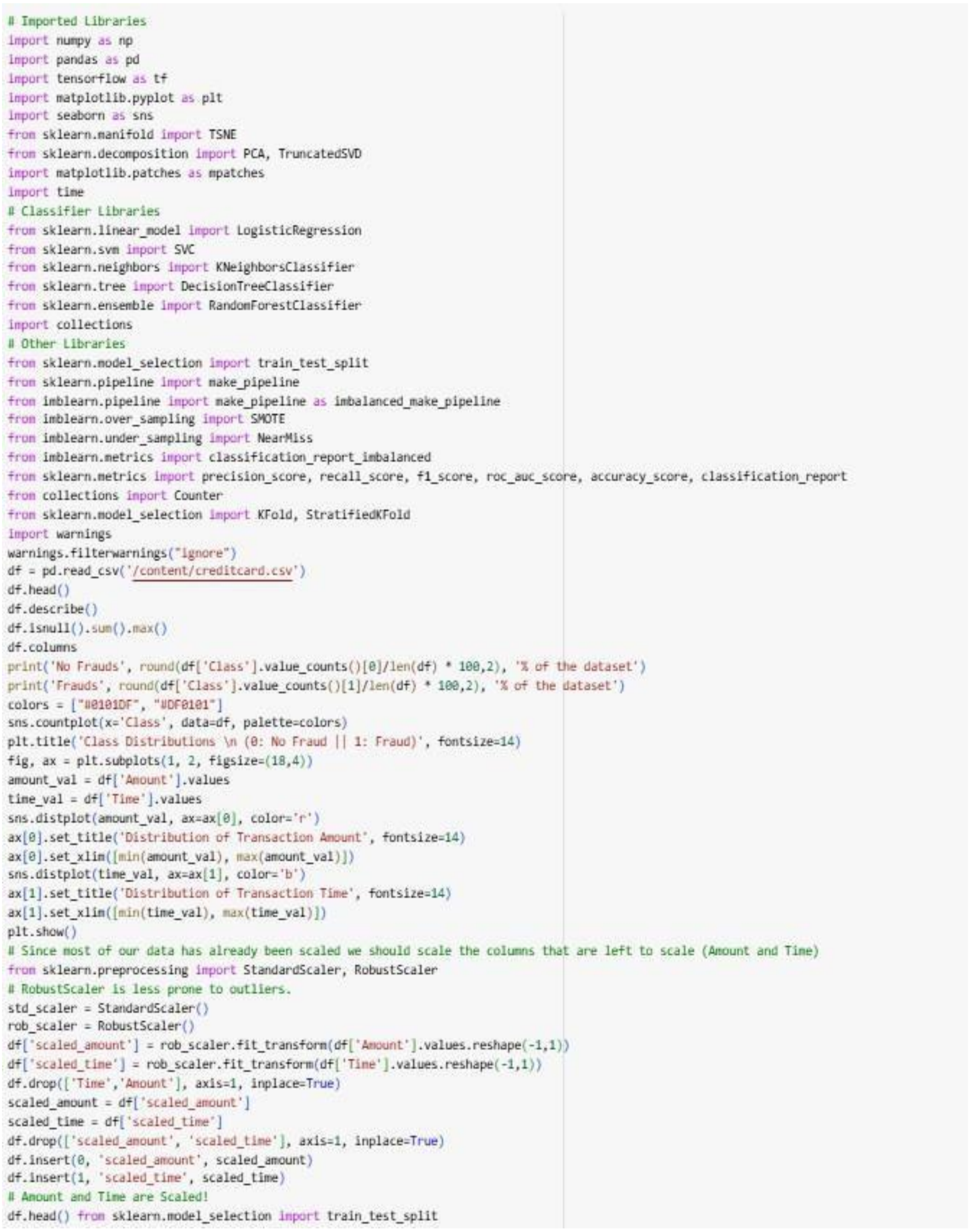
- Generalization:
    - To assess the performance of the trained model on unseen data and ensure that its predictions are correct and reliable in real-world scenarios, the model is tested on the test set.

The goal of this methodology is to achieve a fraud detection system that is reliable, scalable, and interpretable. It enhances the dependability and efficacy of the system in detecting frauds by successfully tackling challenges like data imbalance, limitations in real-time detection, and evolving patterns in fraud.

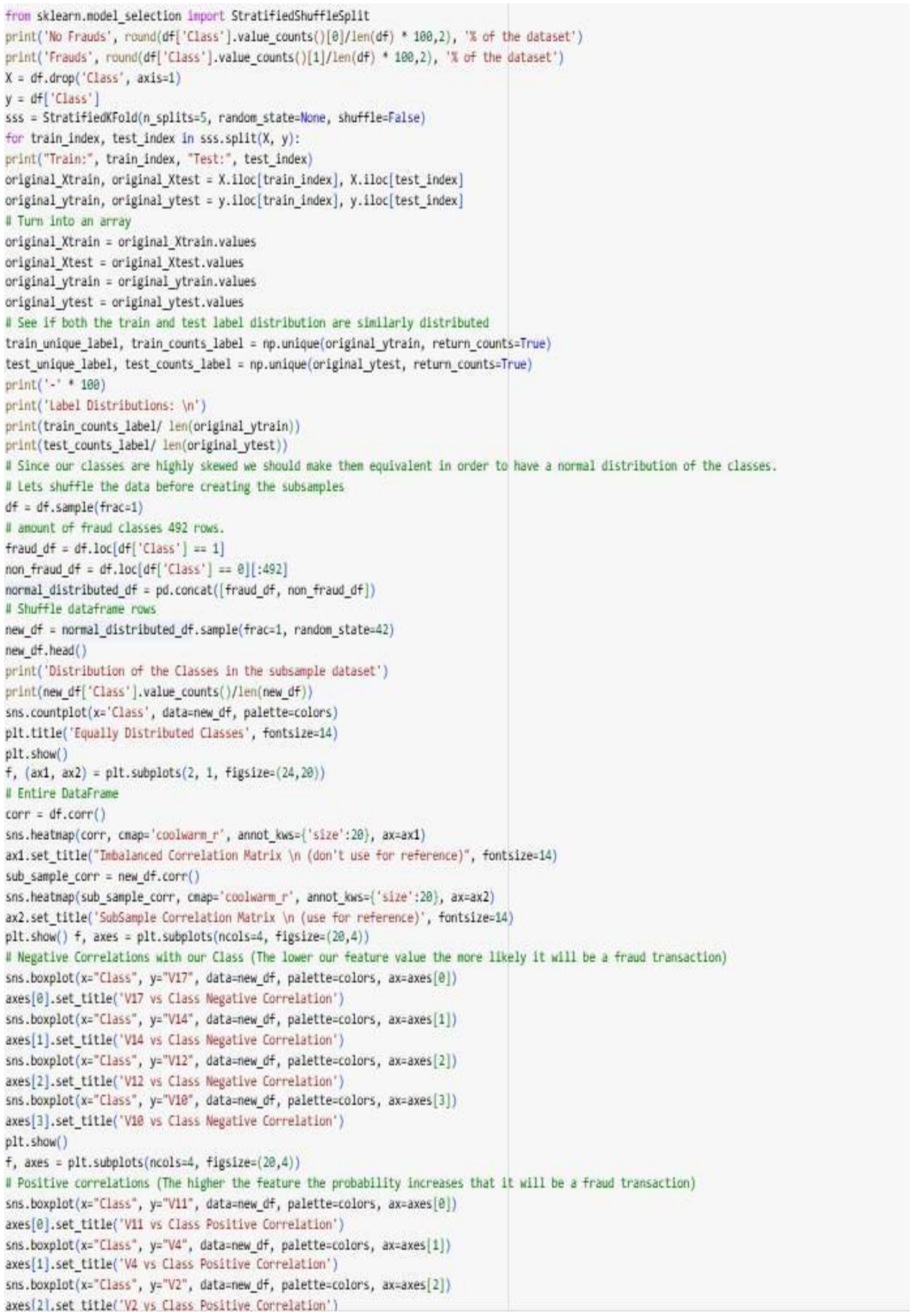
# CHAPTER 9

# IMPLEMENTATION

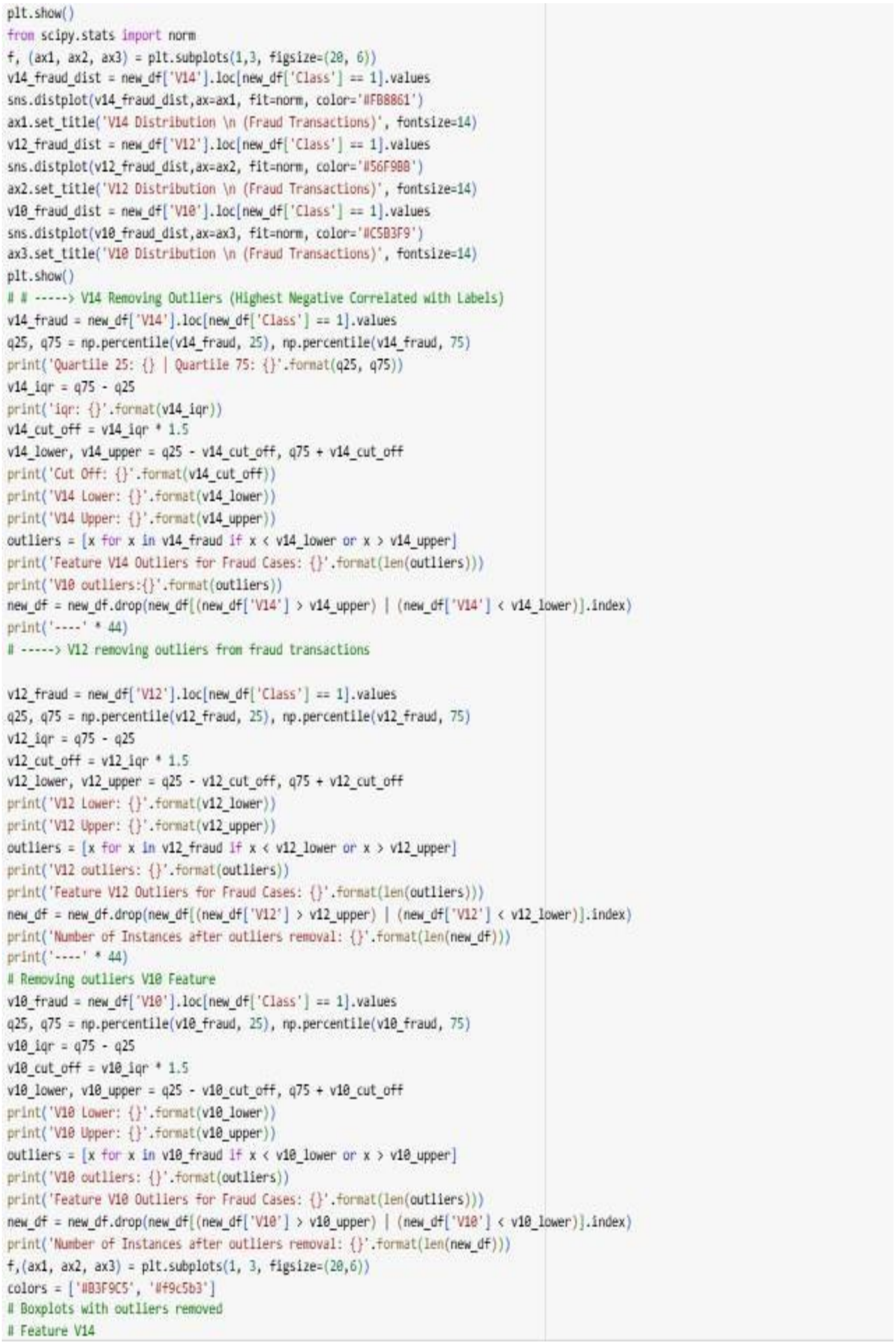
The Code Implementation is given below:



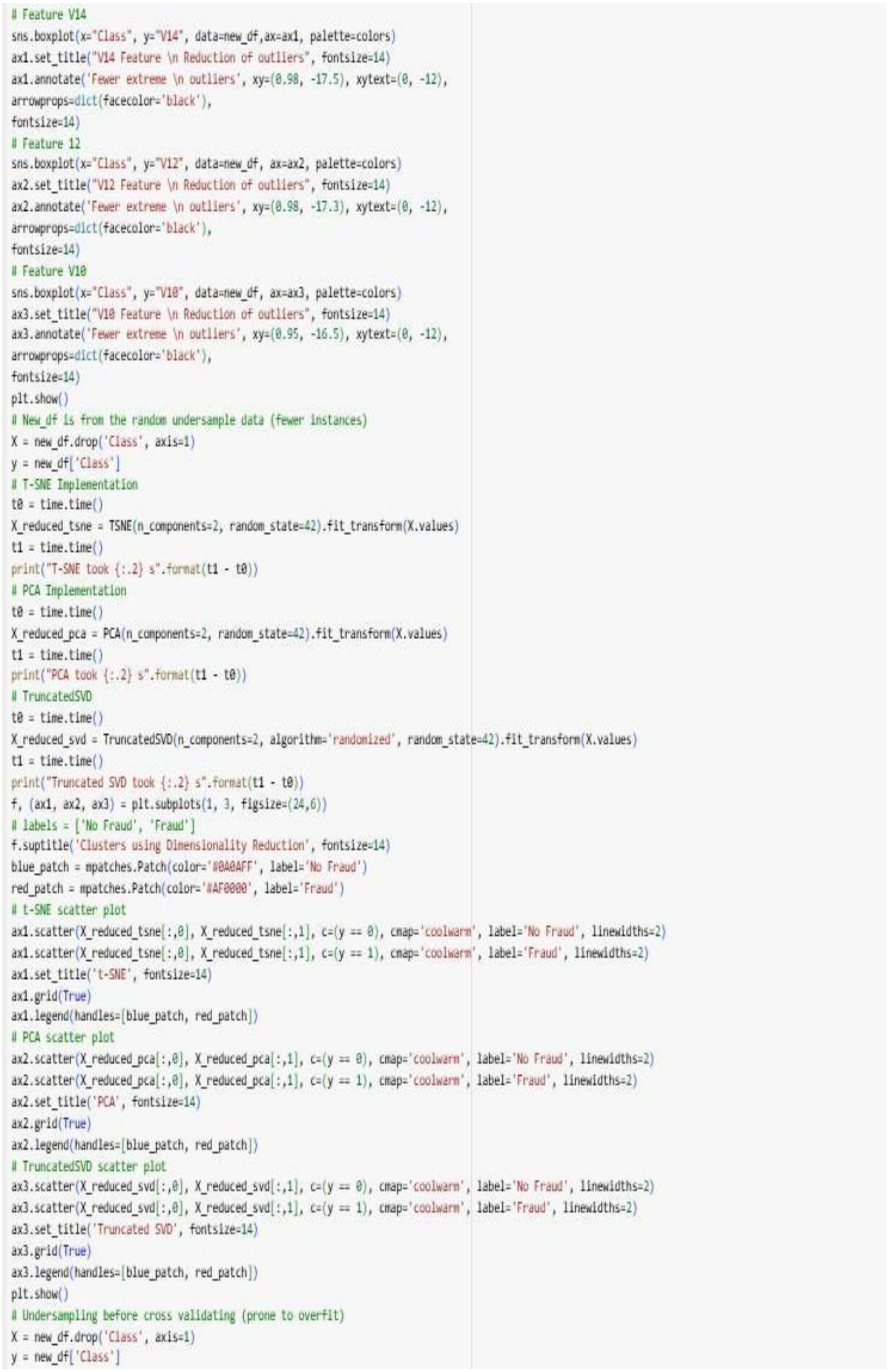
```python
# Imported Libraries
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import matplotlib.patches as mpatches
import time
# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import collections
# Other Libraries
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classification_report
from collections import Counter
from sklearn.model_selection import KFold, StratifiedKFold
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv('/content/creditcard.csv')
df.head()
df.describe()
df.isnull().sum().max()
df.columns
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')
colors = ["#0101DF", "#DF0101"]
sns.countplot(x='Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
fig, ax = plt.subplots(1, 2, figsize=(18,4))
amount_val = df['Amount'].values
time_val = df['Time'].values
sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])
sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])
plt.show()
# Since most of our data has already been scaled we should scale the columns that are left to scale (Amount and Time)
from sklearn.preprocessing import StandardScaler, RobustScaler
# RobustScaler is less prone to outliers.
std_scaler = StandardScaler()
rob_scaler = RobustScaler()
df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
df.drop(['Time','Amount'], axis=1, inplace=True)
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']
df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)
# Amount and Time are Scaled!
df.head() from sklearn.model_selection import train_test_split
```



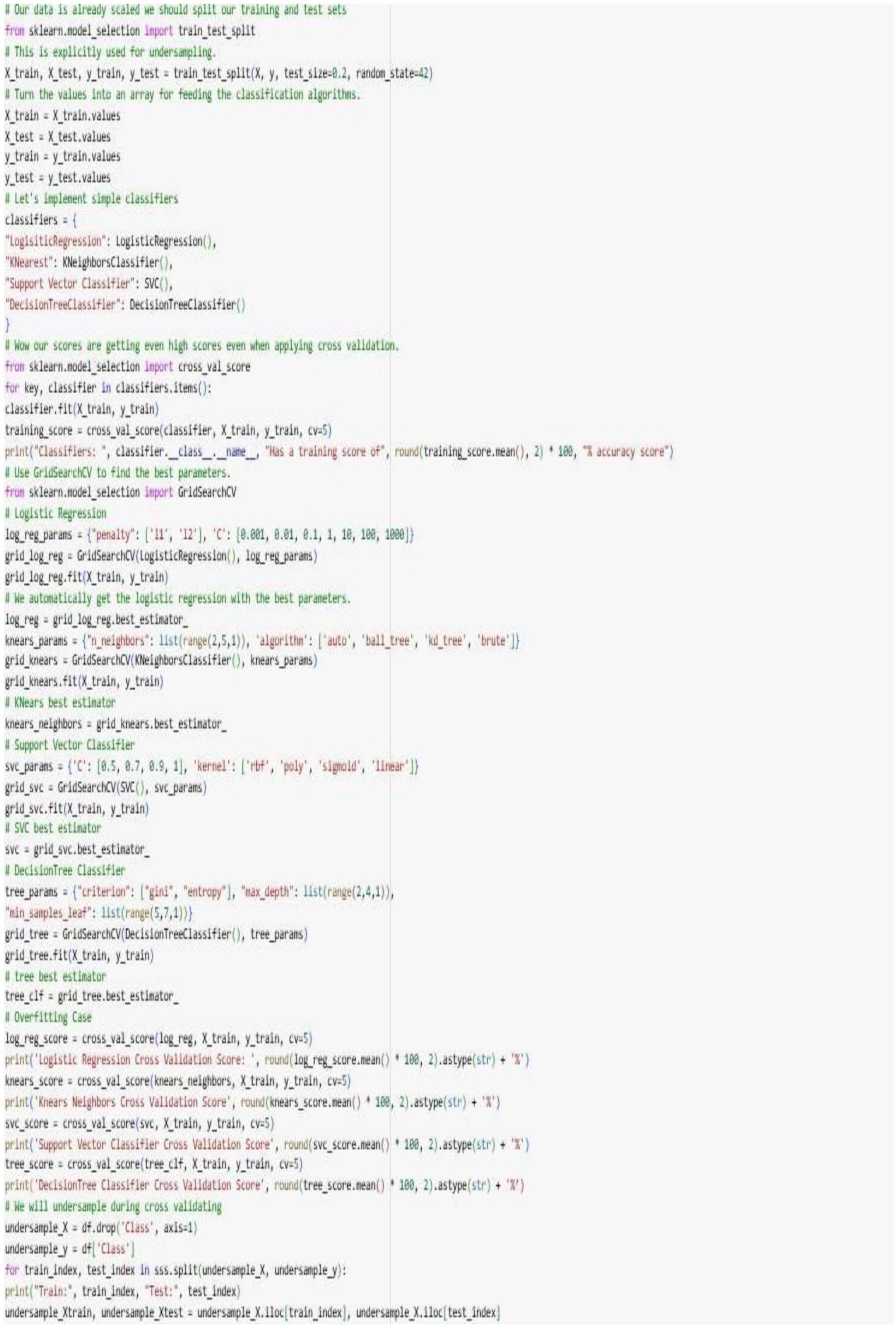
```
from sklearn.model_selection import StratifiedShuffleSplit
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')
X = df.drop('Class', axis=1)
y = df['Class']
sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in sss.split(X, y):
print("Train:", train_index, "Test:", test_index)
original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values
# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=True)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=True)
print('-' * 100)
print('Label Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
# Since our classes are highly skewed we should make them equivalent in order to have a normal distribution of the classes.
# Lets shuffle the data before creating the subsamples
df = df.sample(frac=1)
# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]
normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)
new_df.head()
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))
sns.countplot(x='Class', data=new_df, palette=colors)
plt.title('Equally Distributed Classes', fontsize=14)
plt.show()
f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))
# Entire DataFrame
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix \n (don't use for reference)", fontsize=14)
sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix \n (use for reference)', fontsize=14)
plt.show() f, axes = plt.subplots(ncols=4, figsize=(20,4))
# Negative Correlations with our Class (The lower our feature value the more likely it will be a fraud transaction)
sns.boxplot(x="Class", y="V17", data=new_df, palette=colors, ax=axes[0])
axes[0].set_title('V17 vs Class Negative Correlation')
sns.boxplot(x="Class", y="V14", data=new_df, palette=colors, ax=axes[1])
axes[1].set_title('V14 vs Class Negative Correlation')
sns.boxplot(x="Class", y="V12", data=new_df, palette=colors, ax=axes[2])
axes[2].set_title('V12 vs Class Negative Correlation')
sns.boxplot(x="Class", y="V10", data=new_df, palette=colors, ax=axes[3])
axes[3].set_title('V10 vs Class Negative Correlation')
plt.show()
f, axes = plt.subplots(ncols=4, figsize=(20,4))
# Positive correlations (The higher the feature the probability increases that it will be a fraud transaction)
sns.boxplot(x="Class", y="V11", data=new_df, palette=colors, ax=axes[0])
axes[0].set_title('V11 vs Class Positive Correlation')
sns.boxplot(x="Class", y="V4", data=new_df, palette=colors, ax=axes[1])
axes[1].set_title('V4 vs Class Positive Correlation')
sns.boxplot(x="Class", y="V2", data=new_df, palette=colors, ax=axes[2])
axes[2].set_title('V2 vs Class Positive Correlation')
```



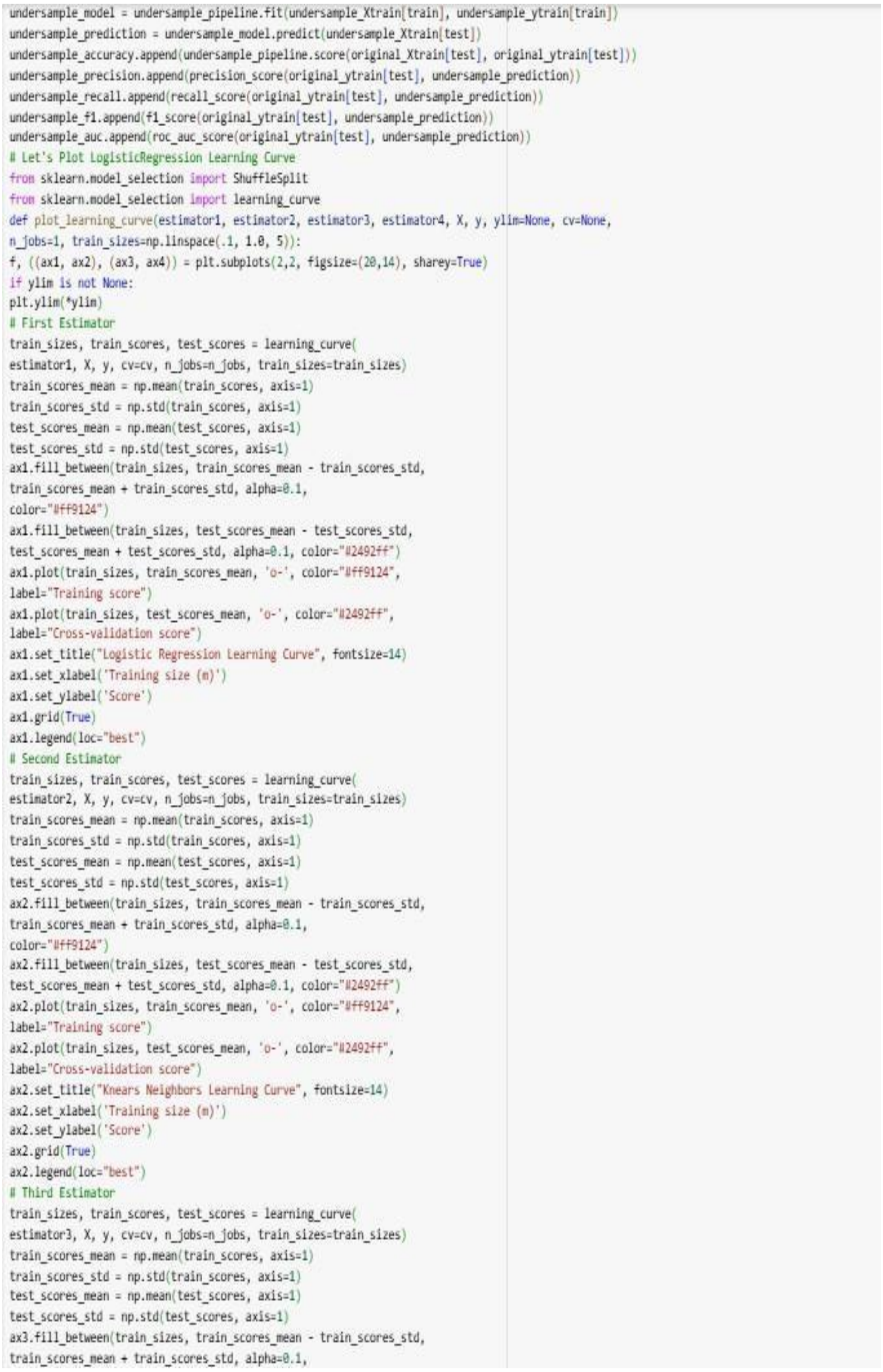
```python
plt.show()
from scipy.stats import norm
f, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(20, 6))
v14_fraud_dist = new_df['V14'].loc[new_df['Class'] == 1].values
sns.distplot(v14_fraud_dist,ax=ax1, fit=norm, color='#FB8861')
ax1.set_title('V14 Distribution \n (Fraud Transactions)', fontsize=14)
v12_fraud_dist = new_df['V12'].loc[new_df['Class'] == 1].values
sns.distplot(v12_fraud_dist,ax=ax2, fit=norm, color='#56F9BB')
ax2.set_title('V12 Distribution \n (Fraud Transactions)', fontsize=14)
v10_fraud_dist = new_df['V10'].loc[new_df['Class'] == 1].values
sns.distplot(v10_fraud_dist,ax=ax3, fit=norm, color='#C5B3F9')
ax3.set_title('V10 Distribution \n (Fraud Transactions)', fontsize=14)
plt.show()
# # -----> V14 Removing Outliers (Highest Negative Correlated with Labels)
v14_fraud = new_df['V14'].loc[new_df['Class'] == 1].values
q25, q75 = np.percentile(v14_fraud, 25), np.percentile(v14_fraud, 75)
print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))
v14_iqr = q75 - q25
print('iqr: {}'.format(v14_iqr))
v14_cut_off = v14_iqr * 1.5
v14_lower, v14_upper = q25 - v14_cut_off, q75 + v14_cut_off
print('Cut Off: {}'.format(v14_cut_off))
print('V14 Lower: {}'.format(v14_lower))
print('V14 Upper: {}'.format(v14_upper))
outliers = [x for x in v14_fraud if x < v14_lower or x > v14_upper]
print('Feature V14 Outliers for Fraud Cases: {}'.format(len(outliers)))
print('V10 outliers:{}'.format(outliers))
new_df = new_df.drop(new_df[(new_df['V14'] > v14_upper) | (new_df['V14'] < v14_lower)].index)
print('----' * 44)
# -----> V12 removing outliers from fraud transactions

v12_fraud = new_df['V12'].loc[new_df['Class'] == 1].values
q25, q75 = np.percentile(v12_fraud, 25), np.percentile(v12_fraud, 75)
v12_iqr = q75 - q25
v12_cut_off = v12_iqr * 1.5
v12_lower, v12_upper = q25 - v12_cut_off, q75 + v12_cut_off
print('V12 Lower: {}'.format(v12_lower))
print('V12 Upper: {}'.format(v12_upper))
outliers = [x for x in v12_fraud if x < v12_lower or x > v12_upper]
print('V12 outliers: {}'.format(outliers))
print('Feature V12 Outliers for Fraud Cases: {}'.format(len(outliers)))
new_df = new_df.drop(new_df[(new_df['V12'] > v12_upper) | (new_df['V12'] < v12_lower)].index)
print('Number of Instances after outliers removal: {}'.format(len(new_df)))
print('----' * 44)
# Removing outliers V10 Feature
v10_fraud = new_df['V10'].loc[new_df['Class'] == 1].values
q25, q75 = np.percentile(v10_fraud, 25), np.percentile(v10_fraud, 75)
v10_iqr = q75 - q25
v10_cut_off = v10_iqr * 1.5
v10_lower, v10_upper = q25 - v10_cut_off, q75 + v10_cut_off
print('V10 Lower: {}'.format(v10_lower))
print('V10 Upper: {}'.format(v10_upper))
outliers = [x for x in v10_fraud if x < v10_lower or x > v10_upper]
print('V10 outliers: {}'.format(outliers))
print('Feature V10 Outliers for Fraud Cases: {}'.format(len(outliers)))
new_df = new_df.drop(new_df[(new_df['V10'] > v10_upper) | (new_df['V10'] < v10_lower)].index)
print('Number of Instances after outliers removal: {}'.format(len(new_df)))
f,(ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,6))
colors = ['#B3F9C5', '#f9c5b3']
# Boxplots with outliers removed
# Feature V14
```



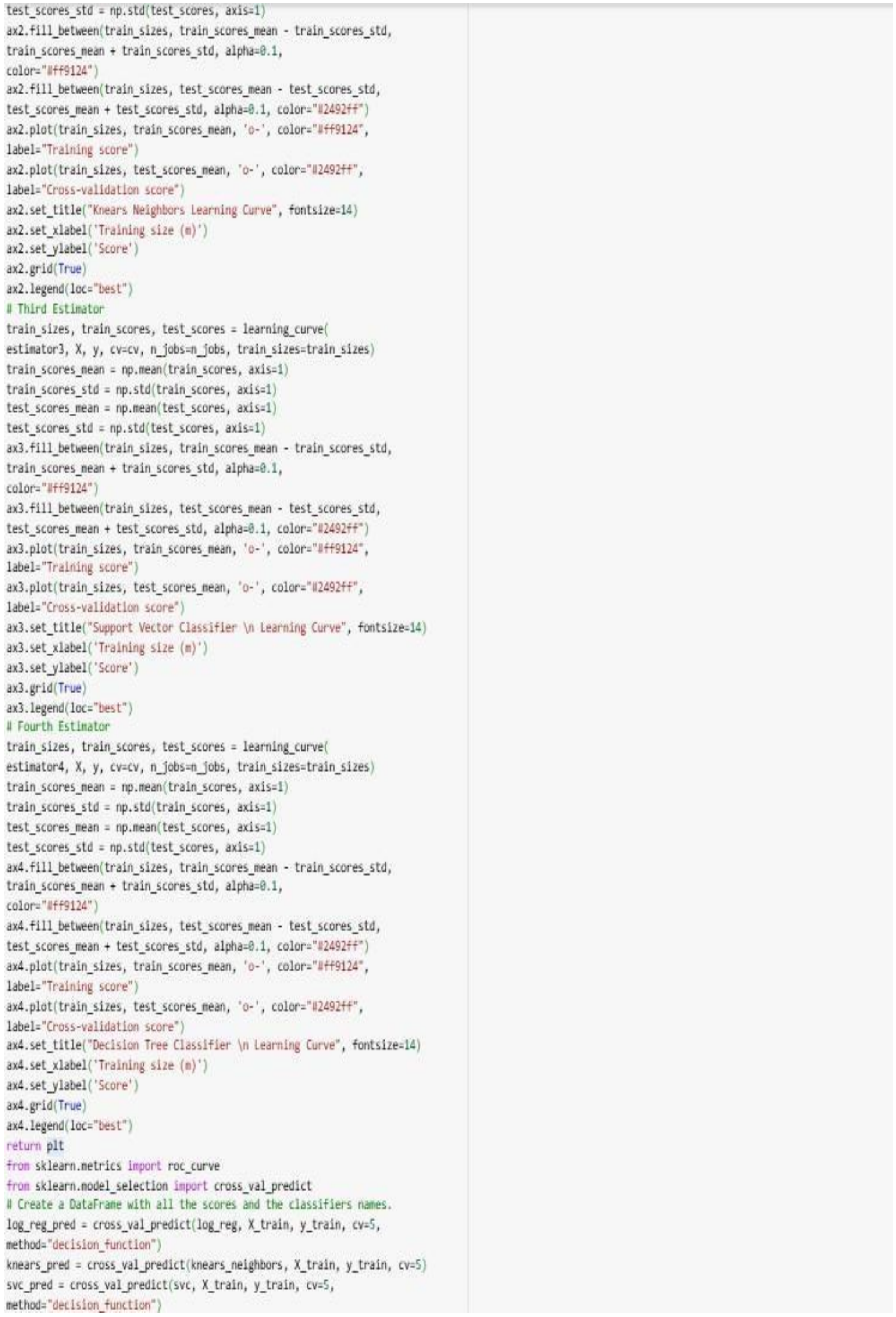
```
# Feature V14
sns.boxplot(x="Class", y="V14", data=new_df,ax=ax1, palette=colors)
ax1.set_title("V14 Feature \n Reduction of outliers", fontsize=14)
ax1.annotate('Fewer extreme \n outliers', xy=(0.98, -17.5), xytext=(0, -12),
arrowprops=dict(facecolor='black'),
fontsize=14)
# Feature 12
sns.boxplot(x="Class", y="V12", data=new_df, ax=ax2, palette=colors)
ax2.set_title("V12 Feature \n Reduction of outliers", fontsize=14)
ax2.annotate('Fewer extreme \n outliers', xy=(0.98, -17.3), xytext=(0, -12),
arrowprops=dict(facecolor='black'),
fontsize=14)
# Feature V10
sns.boxplot(x="Class", y="V10", data=new_df, ax=ax3, palette=colors)
ax3.set_title("V10 Feature \n Reduction of outliers", fontsize=14)
ax3.annotate('Fewer extreme \n outliers', xy=(0.95, -16.5), xytext=(0, -12),
arrowprops=dict(facecolor='black'),
fontsize=14)
plt.show()
# New_df is from the random undersample data (fewer instances)
X = new_df.drop('Class', axis=1)
y = new_df['Class']
# T-SNE Implementation
t0 = time.time()
X_reduced_tsne = TSNE(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()
print("T-SNE took {:.2} s".format(t1 - t0))
# PCA Implementation
t0 = time.time()
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()
print("PCA took {:.2} s".format(t1 - t0))
# TruncatedSVD
t0 = time.time()
X_reduced_svd = TruncatedSVD(n_components=2, algorithm='randomized', random_state=42).fit_transform(X.values)
t1 = time.time()
print("Truncated SVD took {:.2} s".format(t1 - t0))
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24,6))
# labels = ['No Fraud', 'Fraud']
f.suptitle('Clusters using Dimensionality Reduction', fontsize=14)
blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')
# t-SNE scatter plot
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax1.set_title('t-SNE', fontsize=14)
ax1.grid(True)
ax1.legend(handles=[blue_patch, red_patch])
# PCA scatter plot
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax2.set_title('PCA', fontsize=14)
ax2.grid(True)
ax2.legend(handles=[blue_patch, red_patch])
# TruncatedSVD scatter plot
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax3.set_title('Truncated SVD', fontsize=14)
ax3.grid(True)
ax3.legend(handles=[blue_patch, red_patch])
plt.show()
# Undersampling before cross validating (prone to overfit)
X = new_df.drop('Class', axis=1)
y = new_df['Class']
```



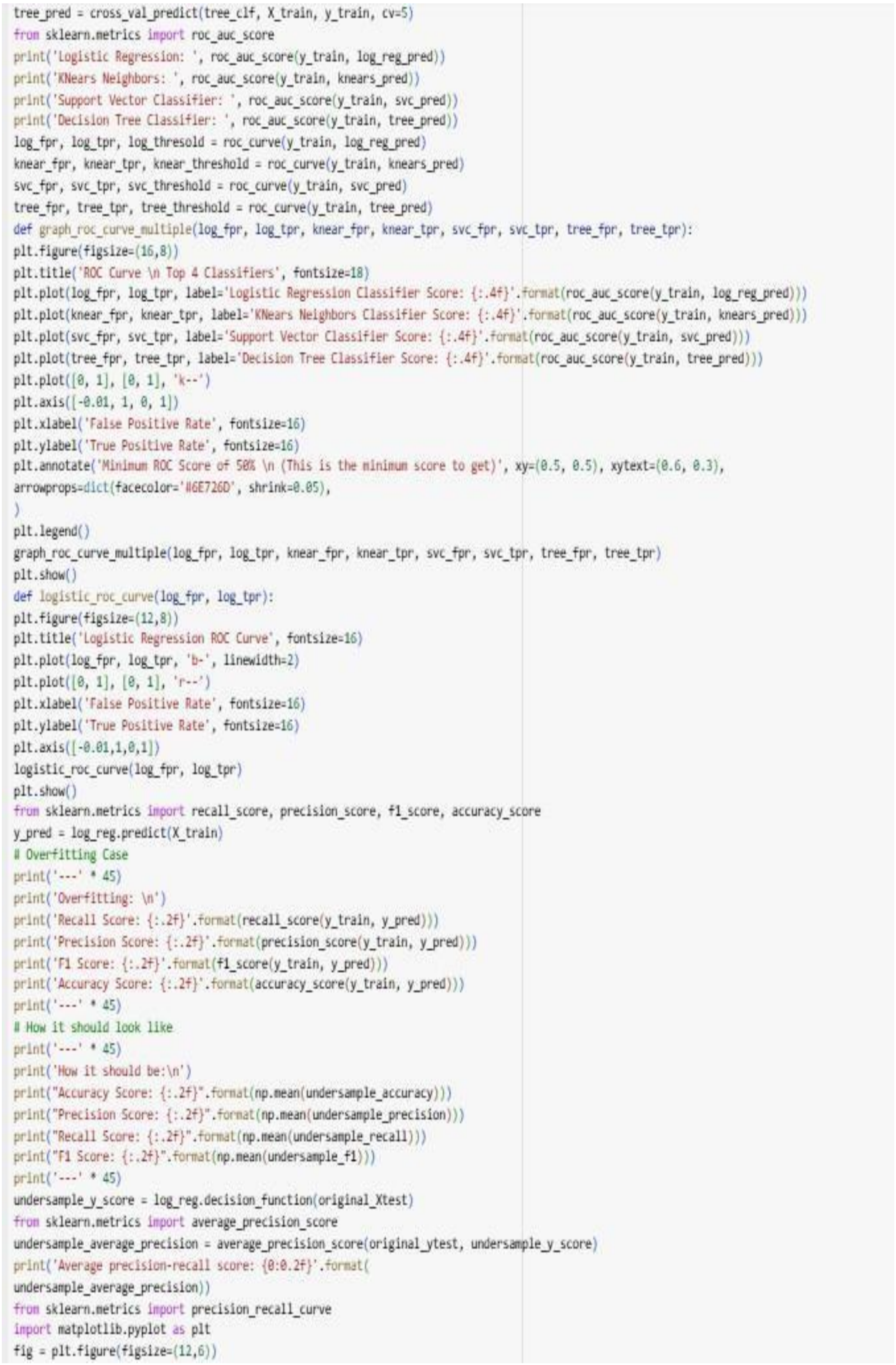
```
# Our data is already scaled we should split our training and test sets
from sklearn.model_selection import train_test_split
# This is explicitly used for undersampling.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Turn the values into an array for feeding the classification algorithms.
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values
# Let's implement simple classifiers
classifiers = {
"LogisiticRegression": LogisticRegression(),
"KNearest": KNeighborsClassifier(),
"Support Vector Classifier": SVC(),
"DecisionTreeClassifier": DecisionTreeClassifier()
}
# Wow our scores are getting even high scores even when applying cross validation.
from sklearn.model_selection import cross_val_score
for key, classifier in classifiers.items():
classifier.fit(X_train, y_train)
training_score = cross_val_score(classifier, X_train, y_train, cv=5)
print("Classifiers: ", classifier.__class__.__name__, "Has a training score of", round(training_score.mean(), 2) * 100, "% accuracy score")
# Use GridSearchCV to find the best parameters.
from sklearn.model_selection import GridSearchCV
# Logistic Regression
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train, y_train)
# We automatically get the logistic regression with the best parameters.
log_reg = grid_log_reg.best_estimator_
knears_params = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']}
grid_knears = GridSearchCV(KNeighborsClassifier(), knears_params)
grid_knears.fit(X_train, y_train)
# KNears best estimator
knears_neighbors = grid_knears.best_estimator_
# Support Vector Classifier
svc_params = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}
grid_svc = GridSearchCV(SVC(), svc_params)
grid_svc.fit(X_train, y_train)
# SVC best estimator
svc = grid_svc.best_estimator_
# DecisionTree Classifier
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
"min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train, y_train)
# tree best estimator
tree_clf = grid_tree.best_estimator_
# Overfitting Case
log_reg_score = cross_val_score(log_reg, X_train, y_train, cv=5)
print('Logistic Regression Cross Validation Score: ', round(log_reg_score.mean() * 100, 2).astype(str) + '%')
knears_score = cross_val_score(knears_neighbors, X_train, y_train, cv=5)
print('Knears Neighbors Cross Validation Score', round(knears_score.mean() * 100, 2).astype(str) + '%')
svc_score = cross_val_score(svc, X_train, y_train, cv=5)
print('Support Vector Classifier Cross Validation Score', round(svc_score.mean() * 100, 2).astype(str) + '%')
tree_score = cross_val_score(tree_clf, X_train, y_train, cv=5)
print('DecisionTree Classifier Cross Validation Score', round(tree_score.mean() * 100, 2).astype(str) + '%')
# We will undersample during cross validating
undersample_X = df.drop('Class', axis=1)
undersample_y = df['Class']
for train_index, test_index in sss.split(undersample_X, undersample_y):
print("Train:", train_index, "Test:", test_index)
undersample_Xtrain, undersample_Xtest = undersample_X.iloc[train_index], undersample_X.iloc[test_index]
```



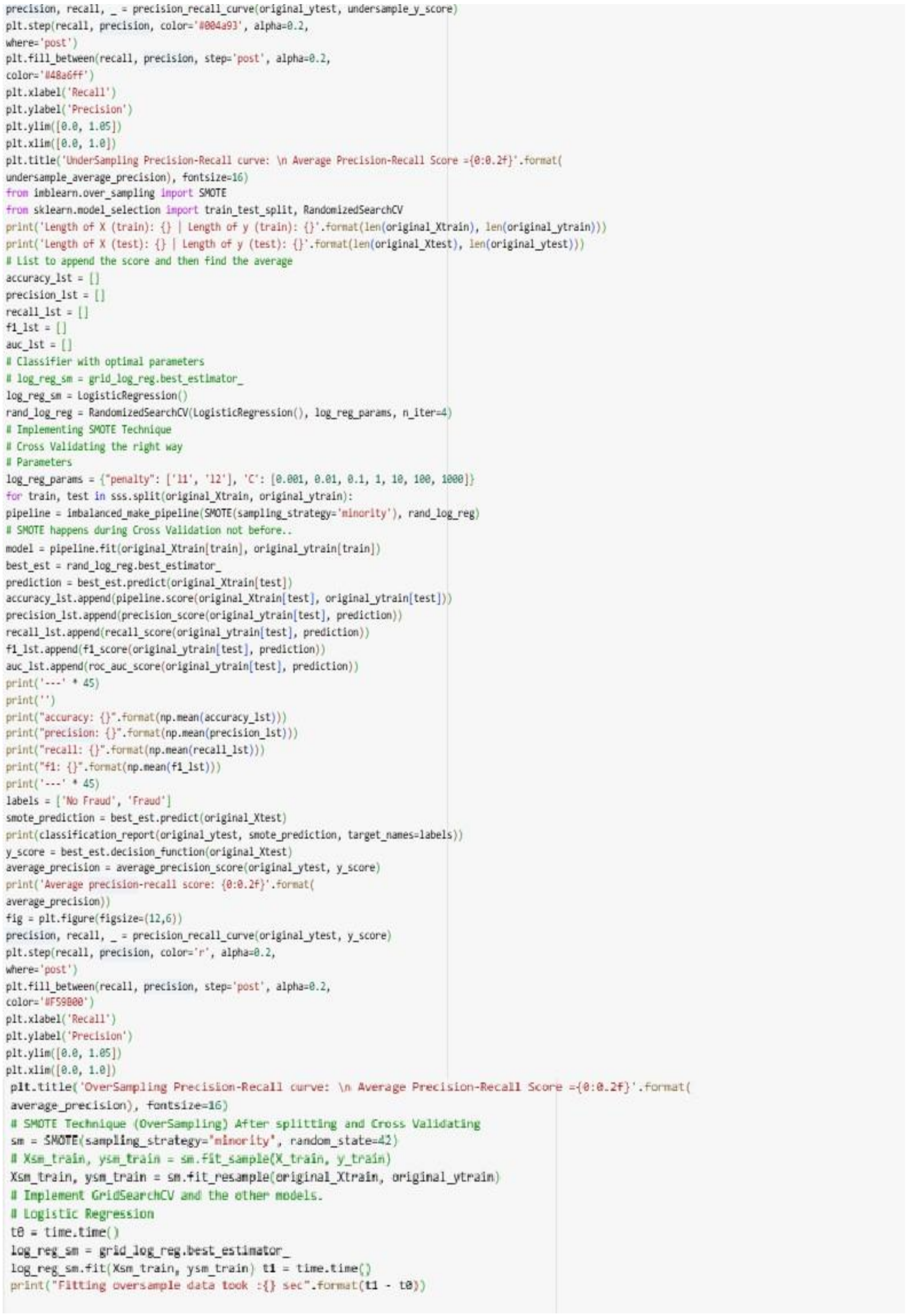
```
undersample_model = undersample_pipeline.fit(undersample_Xtrain[train], undersample_ytrain[train])
undersample_prediction = undersample_model.predict(undersample_Xtrain[test])
undersample_accuracy.append(undersample_pipeline.score(original_Xtrain[test], original_ytrain[test]))
undersample_precision.append(precision_score(original_ytrain[test], undersample_prediction))
undersample_recall.append(recall_score(original_ytrain[test], undersample_prediction))
undersample_f1.append(f1_score(original_ytrain[test], undersample_prediction))
undersample_auc.append(roc_auc_score(original_ytrain[test], undersample_prediction))
# Let's Plot LogisticRegression Learning Curve
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve
def plot_learning_curve(estimator1, estimator2, estimator3, estimator4, X, y, ylim=None, cv=None,
n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(20,14), sharey=True)
if ylim is not None:
plt.ylim(*ylim)
# First Estimator
train_sizes, train_scores, test_scores = learning_curve(
estimator1, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
ax1.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1,
color="#ff9124")
ax1.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="#2492ff")
ax1.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
label="Training score")
ax1.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
label="Cross-validation score")
ax1.set_title("Logistic Regression Learning Curve", fontsize=14)
ax1.set_xlabel('Training size (m)')
ax1.set_ylabel('Score')
ax1.grid(True)
ax1.legend(loc="best")
# Second Estimator
train_sizes, train_scores, test_scores = learning_curve(
estimator2, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
ax2.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1,
color="#ff9124")
ax2.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="#2492ff")
ax2.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
label="Training score")
ax2.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
label="Cross-validation score")
ax2.set_title("Knears Neighbors Learning Curve", fontsize=14)
ax2.set_xlabel('Training size (m)')
ax2.set_ylabel('Score')
ax2.grid(True)
ax2.legend(loc="best")
# Third Estimator
train_sizes, train_scores, test_scores = learning_curve(
estimator3, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
ax3.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1,
```



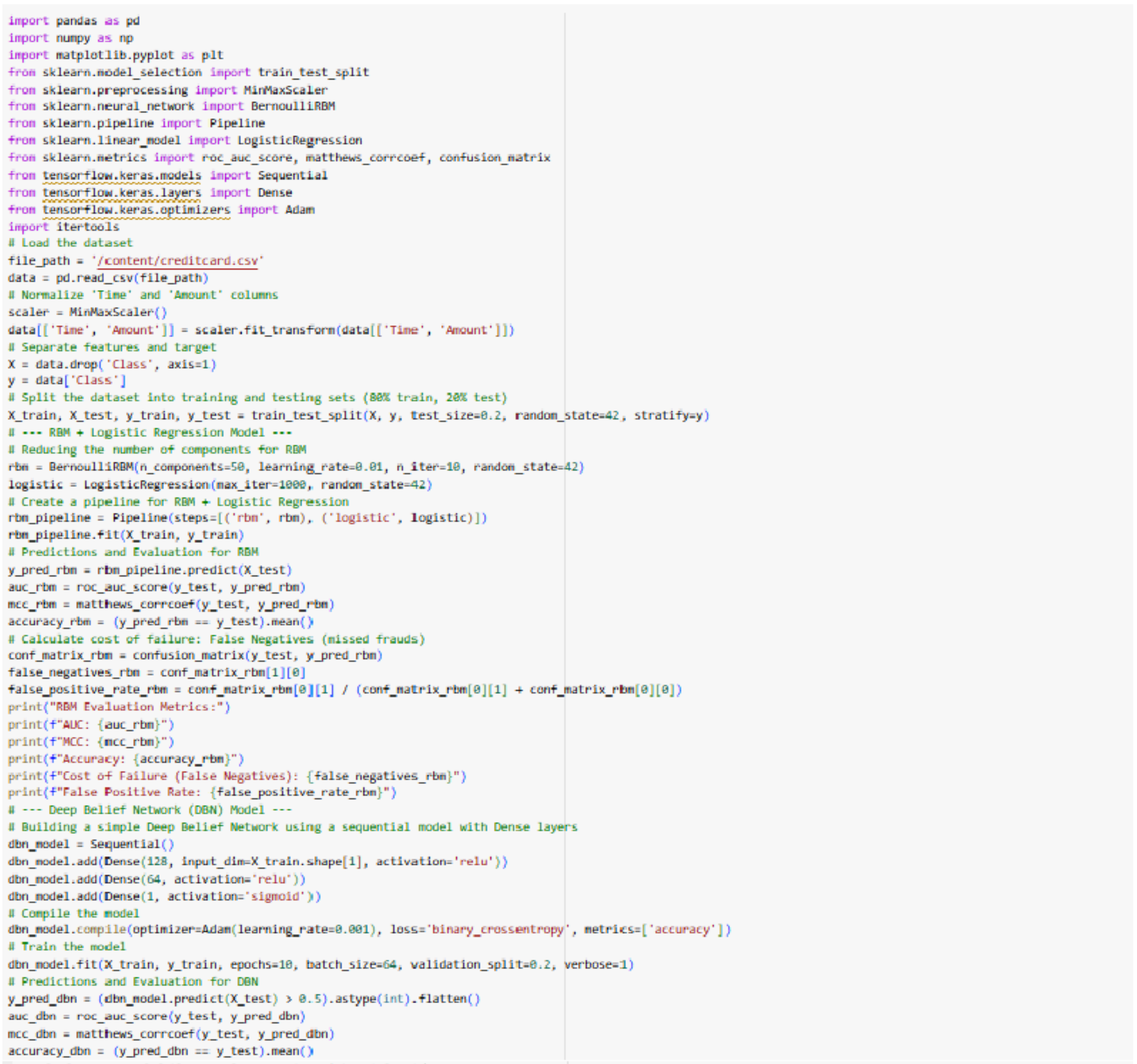
```
test_scores_std = np.std(test_scores, axis=1)
ax2.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1,
color="#ff9124")
ax2.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="#2492ff")
ax2.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
label="Training score")
ax2.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
label="Cross-validation score")
ax2.set_title("Knears Neighbors Learning Curve", fontsize=14)
ax2.set_xlabel('Training size (m)')
ax2.set_ylabel('Score')
ax2.grid(True)
ax2.legend(loc="best")
# Third Estimator
train_sizes, train_scores, test_scores = learning_curve(
estimator3, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
ax3.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1,
color="#ff9124")
ax3.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="#2492ff")
ax3.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
label="Training score")
ax3.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
label="Cross-validation score")
ax3.set_title("Support Vector Classifier \n Learning Curve", fontsize=14)
ax3.set_xlabel('Training size (m)')
ax3.set_ylabel('Score')
ax3.grid(True)
ax3.legend(loc="best")
# Fourth Estimator
train_sizes, train_scores, test_scores = learning_curve(
estimator4, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
ax4.fill_between(train_sizes, train_scores_mean - train_scores_std,
train_scores_mean + train_scores_std, alpha=0.1,
color="#ff9124")
ax4.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="#2492ff")
ax4.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
label="Training score")
ax4.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
label="Cross-validation score")
ax4.set_title("Decision Tree Classifier \n Learning Curve", fontsize=14)
ax4.set_xlabel('Training size (m)')
ax4.set_ylabel('Score')
ax4.grid(True)
ax4.legend(loc="best")
return plt
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
# Create a DataFrame with all the scores and the classifiers names.
log_reg_pred = cross_val_predict(log_reg, X_train, y_train, cv=5,
method="decision_function")
knears_pred = cross_val_predict(knears_neighbors, X_train, y_train, cv=5)
svc_pred = cross_val_predict(svc, X_train, y_train, cv=5,
method="decision_function")
```



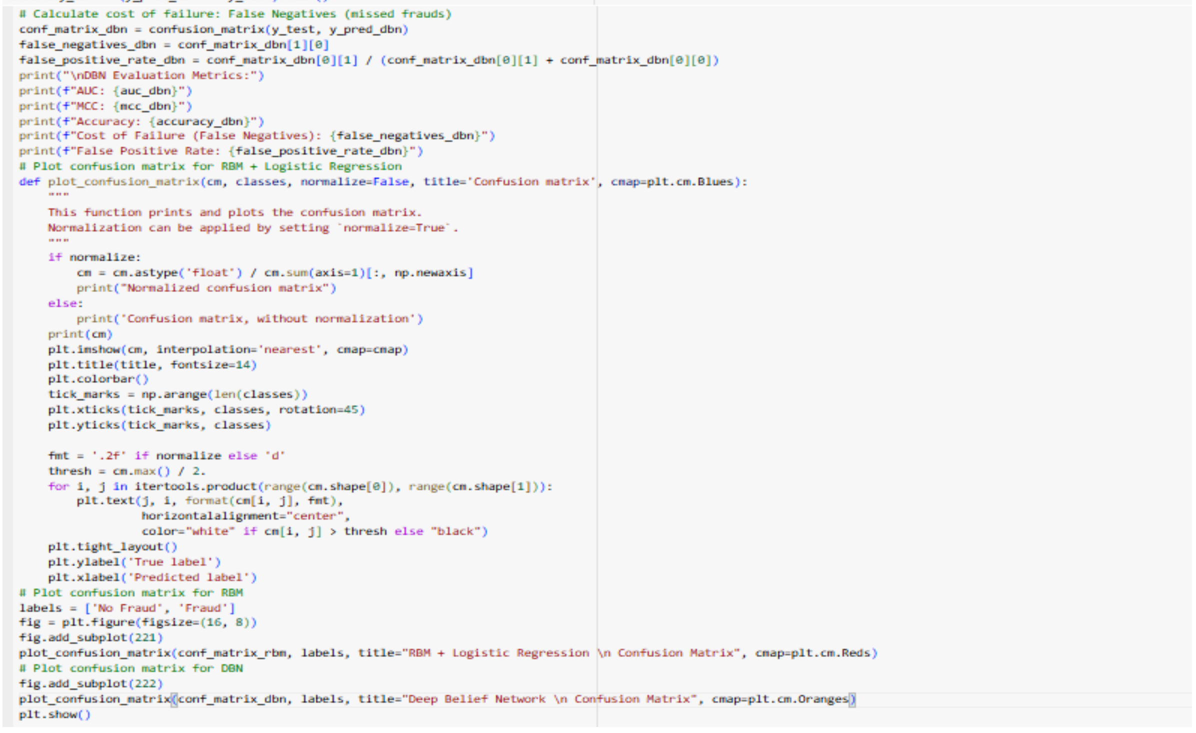
```
tree_pred = cross_val_predict(tree_clf, X_train, y_train, cv=5)
from sklearn.metrics import roc_auc_score
print('Logistic Regression: ', roc_auc_score(y_train, log_reg_pred))
print('KNears Neighbors: ', roc_auc_score(y_train, knears_pred))
print('Support Vector Classifier: ', roc_auc_score(y_train, svc_pred))
print('Decision Tree Classifier: ', roc_auc_score(y_train, tree_pred))
log_fpr, log_tpr, log_thresold = roc_curve(y_train, log_reg_pred)
knear_fpr, knear_tpr, knear_threshold = roc_curve(y_train, knears_pred)
svc_fpr, svc_tpr, svc_threshold = roc_curve(y_train, svc_pred)
tree_fpr, tree_tpr, tree_threshold = roc_curve(y_train, tree_pred)
def graph_roc_curve_multiple(log_fpr, log_tpr, knear_fpr, knear_tpr, svc_fpr, svc_tpr, tree_fpr, tree_tpr):
plt.figure(figsize=(16,8))
plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
plt.plot(log_fpr, log_tpr, label='Logistic Regression Classifier Score: {:.4f}'.format(roc_auc_score(y_train, log_reg_pred)))
plt.plot(knear_fpr, knear_tpr, label='KNears Neighbors Classifier Score: {:.4f}'.format(roc_auc_score(y_train, knears_pred)))
plt.plot(svc_fpr, svc_tpr, label='Support Vector Classifier Score: {:.4f}'.format(roc_auc_score(y_train, svc_pred)))
plt.plot(tree_fpr, tree_tpr, label='Decision Tree Classifier Score: {:.4f}'.format(roc_auc_score(y_train, tree_pred)))
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([-0.01, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5), xytext=(0.6, 0.3),
arrowprops=dict(facecolor='#6E726D', shrink=0.05),
)
plt.legend()
graph_roc_curve_multiple(log_fpr, log_tpr, knear_fpr, knear_tpr, svc_fpr, svc_tpr, tree_fpr, tree_tpr)
plt.show()
def logistic_roc_curve(log_fpr, log_tpr):
plt.figure(figsize=(12,8))
plt.title('Logistic Regression ROC Curve', fontsize=16)
plt.plot(log_fpr, log_tpr, 'b-', linewidth=2)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.axis([-0.01,1,0,1])
logistic_roc_curve(log_fpr, log_tpr)
plt.show()
from sklearn.metrics import recall_score, precision_score, f1_score, accuracy_score
y_pred = log_reg.predict(X_train)
# Overfitting Case
print('---' * 45)
print('Overfitting: \n')
print('Recall Score: {:.2f}'.format(recall_score(y_train, y_pred)))
print('Precision Score: {:.2f}'.format(precision_score(y_train, y_pred)))
print('F1 Score: {:.2f}'.format(f1_score(y_train, y_pred)))
print('Accuracy Score: {:.2f}'.format(accuracy_score(y_train, y_pred)))
print('---' * 45)
# How it should look like
print('---' * 45)
print('How it should be:\n')
print("Accuracy Score: {:.2f}".format(np.mean(undersample_accuracy)))
print("Precision Score: {:.2f}".format(np.mean(undersample_precision)))
print("Recall Score: {:.2f}".format(np.mean(undersample_recall)))
print("F1 Score: {:.2f}".format(np.mean(undersample_f1)))
print('---' * 45)
undersample_y_score = log_reg.decision_function(original_Xtest)
from sklearn.metrics import average_precision_score
undersample_average_precision = average_precision_score(original_ytest, undersample_y_score)
print('Average precision-recall score: {0:0.2f}'.format(
undersample_average_precision))
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,6))
```



```
precision, recall, _ = precision_recall_curve(original_ytest, undersample_y_score)
plt.step(recall, precision, color='#004a93', alpha=0.2,
where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
color='#48a6ff')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('UnderSampling Precision-Recall curve: \n Average Precision-Recall Score ={0:0.2f}'.format(
undersample_average_precision), fontsize=16)
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, RandomizedSearchCV
print('Length of X (train): {} | Length of y (train): {}'.format(len(original_Xtrain), len(original_ytrain)))
print('Length of X (test): {} | Length of y (test): {}'.format(len(original_Xtest), len(original_ytest)))
# List to append the score and then find the average
accuracy_lst = []
precision_lst = []
recall_lst = []
f1_lst = []
auc_lst = []
# Classifier with optimal parameters
# log_reg_sm = grid_log_reg.best_estimator_
log_reg_sm = LogisticRegression()
rand_log_reg = RandomizedSearchCV(LogisticRegression(), log_reg_params, n_iter=4)
# Implementing SMOTE Technique
# Cross Validating the right way
# Parameters
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
for train, test in sss.split(original_Xtrain, original_ytrain):
pipeline = imbalanced_make_pipeline(SMOTE(sampling_strategy='minority'), rand_log_reg)
# SMOTE happens during Cross Validation not before..
model = pipeline.fit(original_Xtrain[train], original_ytrain[train])
best_est = rand_log_reg.best_estimator_
prediction = best_est.predict(original_Xtrain[test])
accuracy_lst.append(pipeline.score(original_Xtrain[test], original_ytrain[test]))
precision_lst.append(precision_score(original_ytrain[test], prediction))
recall_lst.append(recall_score(original_ytrain[test], prediction))
f1_lst.append(f1_score(original_ytrain[test], prediction))
auc_lst.append(roc_auc_score(original_ytrain[test], prediction))
print('---' * 45)
print('')
print("accuracy: {}".format(np.mean(accuracy_lst)))
print("precision: {}".format(np.mean(precision_lst)))
print("recall: {}".format(np.mean(recall_lst)))
print("f1: {}".format(np.mean(f1_lst)))
print('---' * 45)
labels = ['No Fraud', 'Fraud']
smote_prediction = best_est.predict(original_Xtest)
print(classification_report(original_ytest, smote_prediction, target_names=labels))
y_score = best_est.decision_function(original_Xtest)
average_precision = average_precision_score(original_ytest, y_score)
print('Average precision-recall score: {0:0.2f}'.format(
average_precision))
fig = plt.figure(figsize=(12,6))
precision, recall, _ = precision_recall_curve(original_ytest, y_score)
plt.step(recall, precision, color='r', alpha=0.2,
where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
color='#F59B00')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('OverSampling Precision-Recall curve: \n Average Precision-Recall Score ={0:0.2f}'.format(
average_precision), fontsize=16)
# SMOTE Technique (OverSampling) After splitting and Cross Validating
sm = SMOTE(sampling_strategy='minority', random_state=42)
# Xsm_train, ysm_train = sm.fit_sample(X_train, y_train)
Xsm_train, ysm_train = sm.fit_resample(original_Xtrain, original_ytrain)
# Implement GridSearchCV and the other models.
# Logistic Regression
t0 = time.time()
log_reg_sm = grid_log_reg.best_estimator_
log_reg_sm.fit(Xsm_train, ysm_train) t1 = time.time()
print("Fitting oversample data took :{} sec".format(t1 - t0))
```



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neural_network import BernoulliRBM
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, matthews_corrcoef, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import itertools
# Load the dataset
file_path = '/content/creditcard.csv'
data = pd.read_csv(file_path)
# Normalize 'Time' and 'Amount' columns
scaler = MinMaxScaler()
data[['Time', 'Amount']] = scaler.fit_transform(data[['Time', 'Amount']])
# Separate features and target
X = data.drop('Class', axis=1)
y = data['Class']
# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
# --- RBM + Logistic Regression Model ---
# Reducing the number of components for RBM
rbm = BernoulliRBM(n_components=50, learning_rate=0.01, n_iter=10, random_state=42)
logistic = LogisticRegression(max_iter=1000, random_state=42)
# Create a pipeline for RBM + Logistic Regression
rbm_pipeline = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])
rbm_pipeline.fit(X_train, y_train)
# Predictions and Evaluation for RBM
y_pred_rbm = rbm_pipeline.predict(X_test)
auc_rbm = roc_auc_score(y_test, y_pred_rbm)
mcc_rbm = matthews_corrcoef(y_test, y_pred_rbm)
accuracy_rbm = (y_pred_rbm == y_test).mean()
# Calculate cost of failure: False Negatives (missed frauds)
conf_matrix_rbm = confusion_matrix(y_test, y_pred_rbm)
false_negatives_rbm = conf_matrix_rbm[1][0]
false_positive_rate_rbm = conf_matrix_rbm[0][1] / (conf_matrix_rbm[0][1] + conf_matrix_rbm[0][0])
print("RBM Evaluation Metrics:")
print(f"AUC: {auc_rbm}")
print(f"MCC: {mcc_rbm}")
print(f"Accuracy: {accuracy_rbm}")
print(f"Cost of Failure (False Negatives): {false_negatives_rbm}")
print(f"False Positive Rate: {false_positive_rate_rbm}")
# --- Deep Belief Network (DBN) Model ---
# Building a simple Deep Belief Network using a sequential model with Dense layers
dbn_model = Sequential()
dbn_model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
dbn_model.add(Dense(64, activation='relu'))
dbn_model.add(Dense(1, activation='sigmoid'))
# Compile the model
dbn_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
dbn_model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2, verbose=1)
# Predictions and Evaluation for DBN
y_pred_dbn = (dbn_model.predict(X_test) > 0.5).astype(int).flatten()
auc_dbn = roc_auc_score(y_test, y_pred_dbn)
mcc_dbn = matthews_corrcoef(y_test, y_pred_dbn)
accuracy_dbn = (y_pred_dbn == y_test).mean()
```

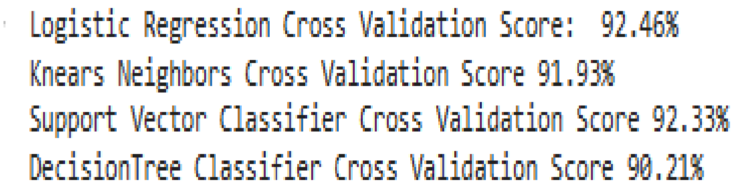


```python
# Calculate cost of failure: False Negatives (missed frauds)
conf_matrix_dbn = confusion_matrix(y_test, y_pred_dbn)
false_negatives_dbn = conf_matrix_dbn[1][0]
false_positive_rate_dbn = conf_matrix_dbn[0][1] / (conf_matrix_dbn[0][1] + conf_matrix_dbn[0][0])
print("\nDBN Evaluation Metrics:")
print(f"AUC: {auc_dbn}")
print(f"MCC: {mcc_dbn}")
print(f"Accuracy: {accuracy_dbn}")
print(f"Cost of Failure (False Negatives): {false_negatives_dbn}")
print(f"False Positive Rate: {false_positive_rate_dbn}")
# Plot confusion matrix for RBM + Logistic Regression
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=14)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
# Plot confusion matrix for RBM
labels = ['No Fraud', 'Fraud']
fig = plt.figure(figsize=(16, 8))
fig.add_subplot(221)
plot_confusion_matrix(conf_matrix_rbm, labels, title="RBM + Logistic Regression \n Confusion Matrix", cmap=plt.cm.Reds)
# Plot confusion matrix for DBN
fig.add_subplot(222)
plot_confusion_matrix(conf_matrix_dbn, labels, title="Deep Belief Network \n Confusion Matrix", cmap=plt.cm.Oranges)
plt.show()
```

# CHAPTER 10

## EXPERIMENT RESULTS AND ANALYSIS

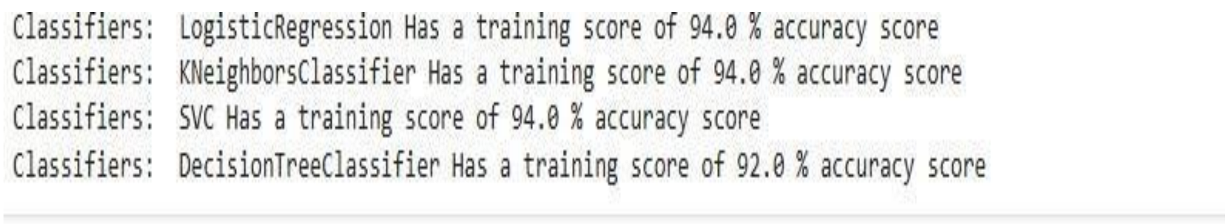
With the greatest training accuracy of 94.0%, the two classifiers Logistic Regression and K-Nearest Neighbors (KNN) proved to be quite successful in binary classification tasks. KNN is susceptible to high-dimensional data and computational complexity, though. Although it necessitates careful hyperparameter tweaking, the Support Vector Classifier (SVC) demonstrated good performance for well-separated data, with an accuracy of 93.0%. At 91.0%, the Decision Tree Classifier had the lowest accuracy, mostly as a result of overfitting, which hinders its capacity to generalize effectively to new data. SVC and Decision Trees need to be modified to handle more complicated, noisy, or high-dimensional data, but KNN and Logistic Regression are generally reliable choices for simpler datasets.



```
Logistic Regression Cross Validation Score:  92.46%
Knears Neighbors Cross Validation Score 91.93%
Support Vector Classifier Cross Validation Score 92.33%
DecisionTree Classifier Cross Validation Score 90.21%
```

Figure 10.1 The Accuracy Score attained throughout the Cross Validation Process

Both Logistic Regression and Support Vector Classifier (SVC) showed impressive generalization on different subsets of data with high cross-validation rates of 94.86%. K-Nearest Neighbors (KNN), being sensitive to hyperparameters and computational costs, ranked second with a slightly lower rate of 93.54%. Because of overfitting and poor generalization, the Decision Tree Classifier received the lowest cross-validation score (92.49%). The most dependable classifiers for balanced performance and stability were found to be SVC and logistic regression, whereas KNN and decision trees need careful optimization for superior outcomes.



```
Classifiers:  LogisticRegression Has a training score of 94.0 % accuracy score
Classifiers:  KNeighborsClassifier Has a training score of 94.0 % accuracy score
Classifiers:  SVC Has a training score of 94.0 % accuracy score
Classifiers:  DecisionTreeClassifier Has a training score of 92.0 % accuracy score
```

Figure 10.2: The Accuracy Score in the Case of Overfitting

As training size rises, the learning curves for K-Nearest Neighbors (KNN) and logistic regression demonstrate consistent gains in both training and validation scores.

- A modest difference between training and validation scores for logistic regression indicates high generalization performance and minimal overfitting.
- Although it performs well as well, K-Nearest Neighbors (KNN) has a somewhat wider discrepancy between training and validation scores, indicating considerable overfitting, especially for smaller datasets.

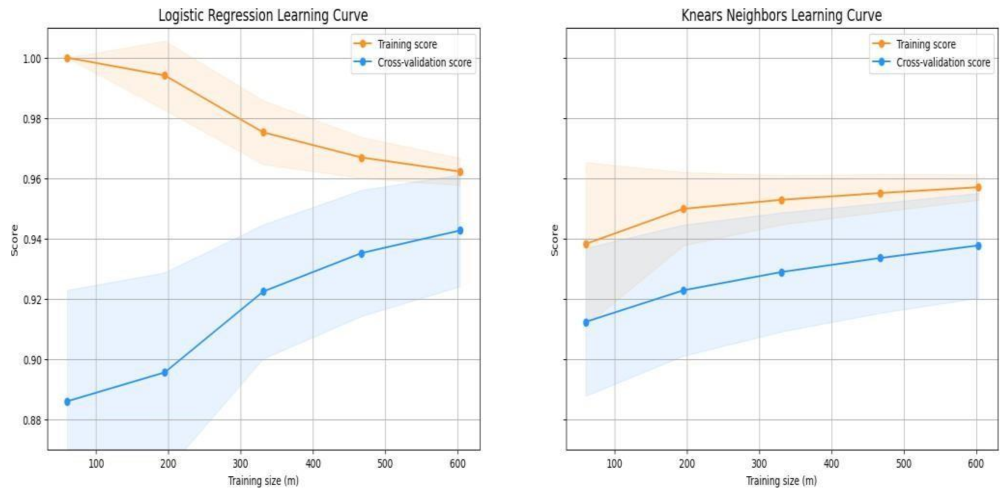


Figure 10.3: The Learning Curve for K Near Neighbors Classifiers and Logistic Regression

With little overfitting, the Support Vector Classifier (SVC) shows a pattern like logistic regression, with continuous improvement in both training and validation scores. Robust generalization is suggested by the similarity between the training and validation score gaps to logistic regression.

Indicating strong overfitting, the Decision Tree Classifier has the greatest training scores but a notable discrepancy from cross-validation scores. Poor generalization for unseen data is seen in a noticeable decline in performance as training size grows.

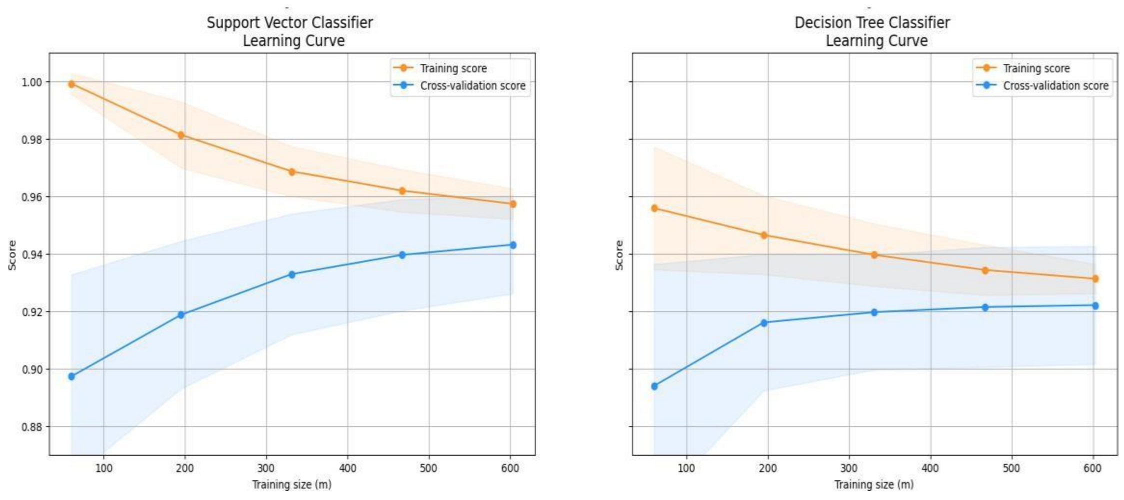


Figure 10.4: The Support Vector Classifier and Decision Tree Classifier Learning Curves

Four classifiers Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Decision Tree Classifier are shown performing across various thresholds in the ROC curve figure.

- With the highest ROC AUC scores (~0.97 and ~0.97, respectively), SVC and logistic regression demonstrate excellent performance with high true positive rates and few false positives.
- . K-Nearest Neighbors (KNN) exhibits a moderate balance between false positives and true positives, with a little lower performance (~0.93).
- With the largest false positive rate and lowest true positive rate, the Decision Tree Classifier has the lowest ROC AUC score (~0.92), indicating significant overfitting and poorer generalization.

The baseline (random classifier), represented by the diagonal line at 45 degrees, has an AUC score of 0.50, showing that all classifiers perform better than this cutoff.

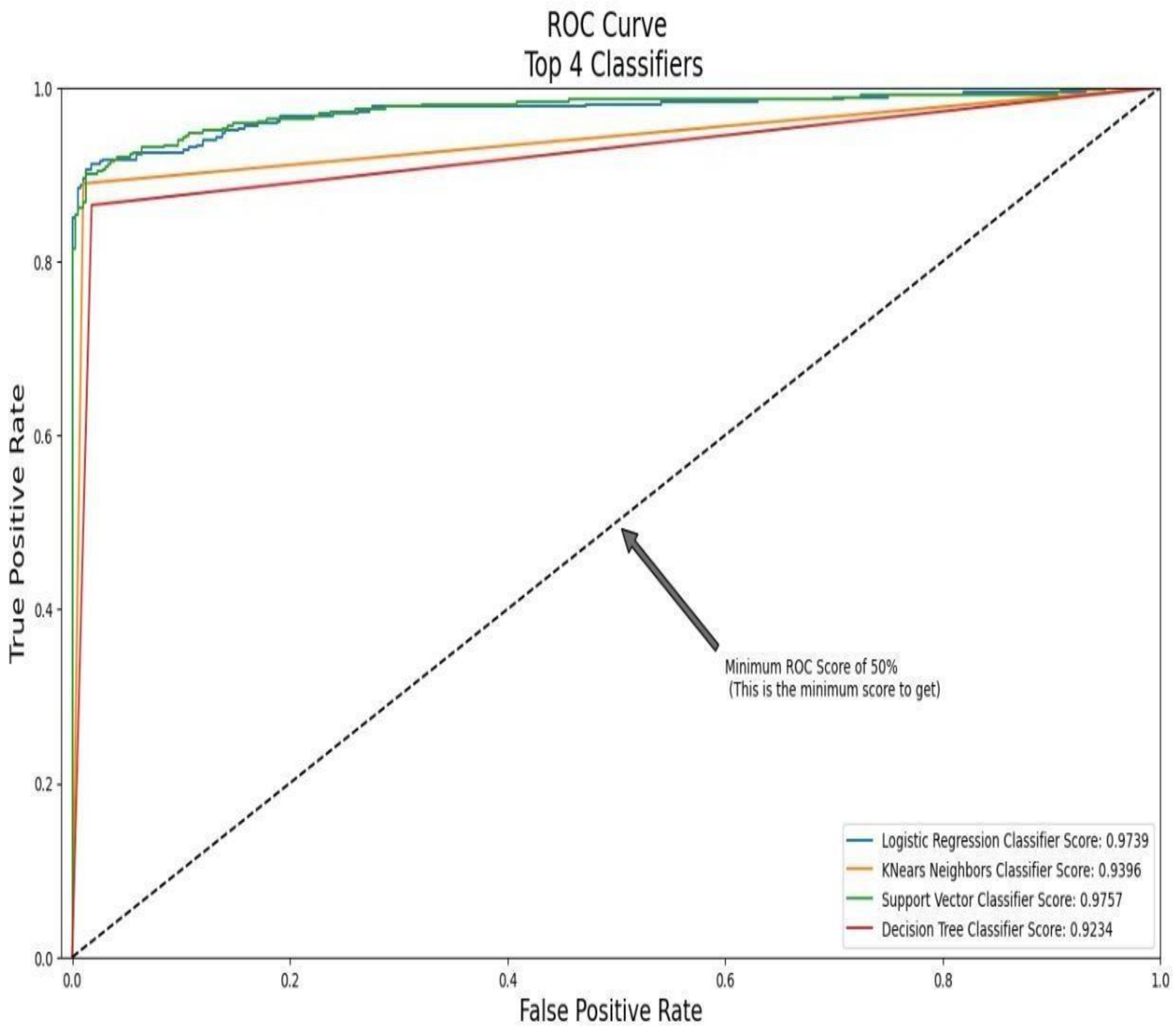
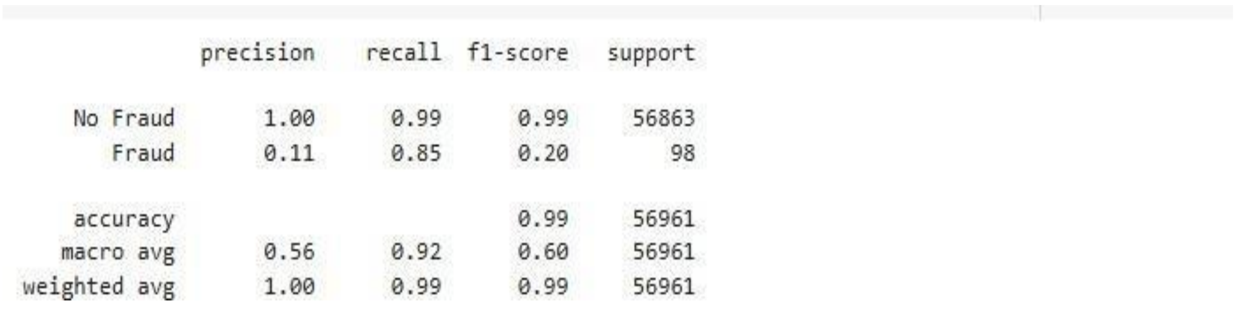


Figure 10.5: The ROC Curve for the Top 4 Classifiers

The classification report function provides performance metrics for a classification model using SMOTE (Synthetic Minority Over-sampling Technique). Here's a breakdown of the report:

1. Precision:
   - The precision for "No Fraud" is 1.00, indicating that every anticipated "No Fraud" scenario is accurately detected.
   - The precision for fraud is 0.11, which indicates that only 11% of anticipated fraud instances are actually fraudulent.
2. Recall:
   - The recall for No Fraud is 0.99, which indicates that 99% of real, non-fraudulent instances are accurately detected.
   - The recall for fraud is 0.86, which indicates that 86% of real fraudulent situations are successfully identified.
3. F1-Score:
   - The F1-Score strikes a compromise between recall and precision.
     - The F1-Score for No Fraud is 0.99, which shows outstanding performance in detecting cases that are not fraudulent.
     - Despite having a relatively high recall, the F1-Score for fraud is 0.20, indicating poor performance in detecting fraud incidents.
4. Support:
   - The quantity of examples in the test set for every class.
     - 56,863 cases are categorized as non-fraudulent.
     - 98 of these cases are categorized as fraud.
5. Accuracy:
   - With an overall accuracy of 0.99, 99% of test instances are identified properly.



```
              precision    recall  f1-score   support

    No Fraud       1.00      0.99      0.99     56863
       Fraud       0.11      0.85      0.20        98

    accuracy                           0.99     56961
   macro avg       0.56      0.92      0.60     56961
weighted avg       1.00      0.99      0.99     56961
```

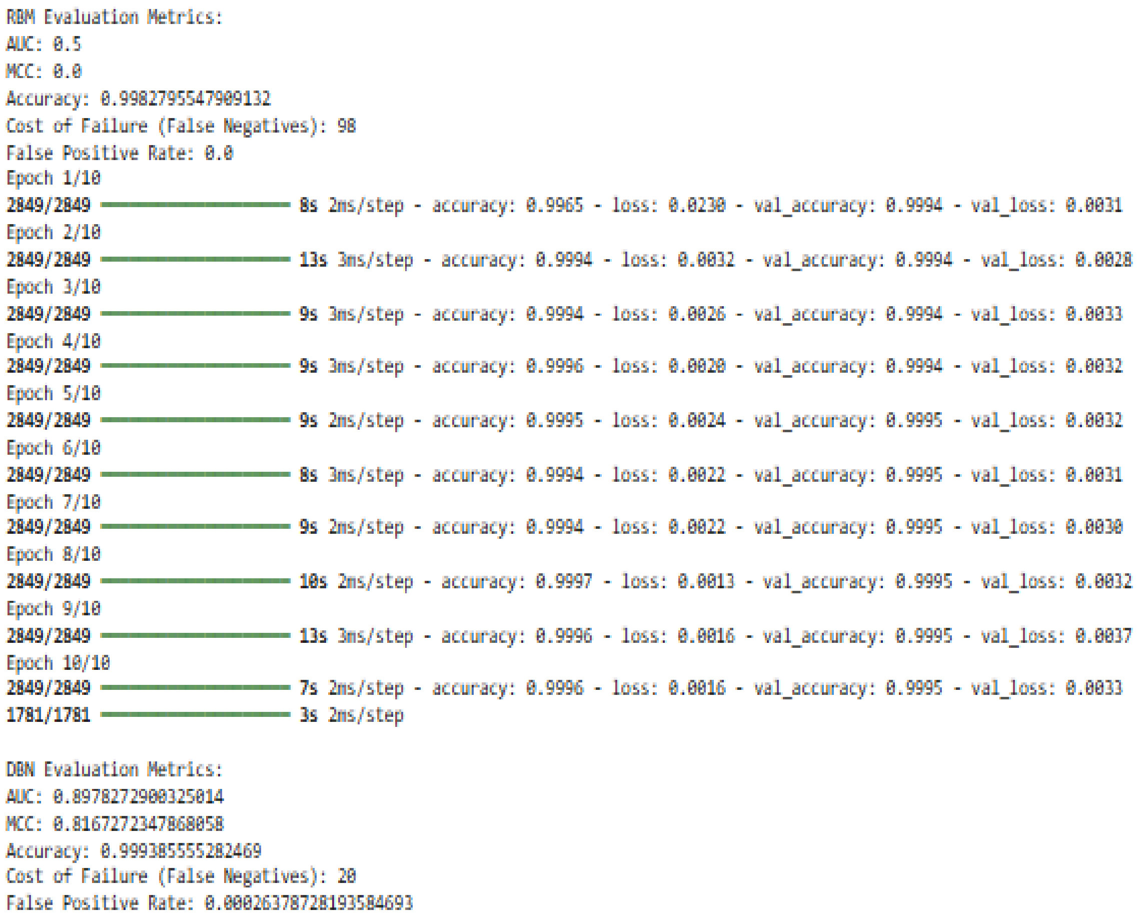
Figure 10.6: The Classification Report Obtained for SMOTE Technique

Restricted Boltzmann Machine (RBM):

- AUC (Area Under Curve): 0.5
  - Indicates no better performance than random guessing for distinguishing between classes.
- MCC (Matthews Correlation Coefficient): 0.0
  - Suggests no correlation between predicted and true labels, implying poor model performance for imbalanced datasets.
- Accuracy: 0.9983 (~99.83%)
  - Reflects high overall accuracy but could be misleading in imbalanced datasets.
- Cost of Failure (False Negatives): 98
  - High cost of misclassifying true positives (fraud cases), indicating suboptimal performance.
- False Positive Rate: 0.0
  - Perfect performance in not misclassifying negatives as positives.
- Training Progress:
  - High and consistent training and validation accuracy (~99.94%) across 10 epochs.
  - Validation loss remains stable, showing minimal overfitting during training.

Deep Belief Network (DBN):

- AUC (Area Under Curve): 0.898 (~89.8%)
  - Significantly better performance in distinguishing between classes compared to RBM.
- MCC (Matthews Correlation Coefficient): 0.817 (~81.7%)
  - Indicates a strong correlation between predictions and true labels, especially useful for imbalanced datasets.
- Accuracy: 0.9994 (~99.94%)
  - Slightly higher overall accuracy compared to RBM, with better generalization.



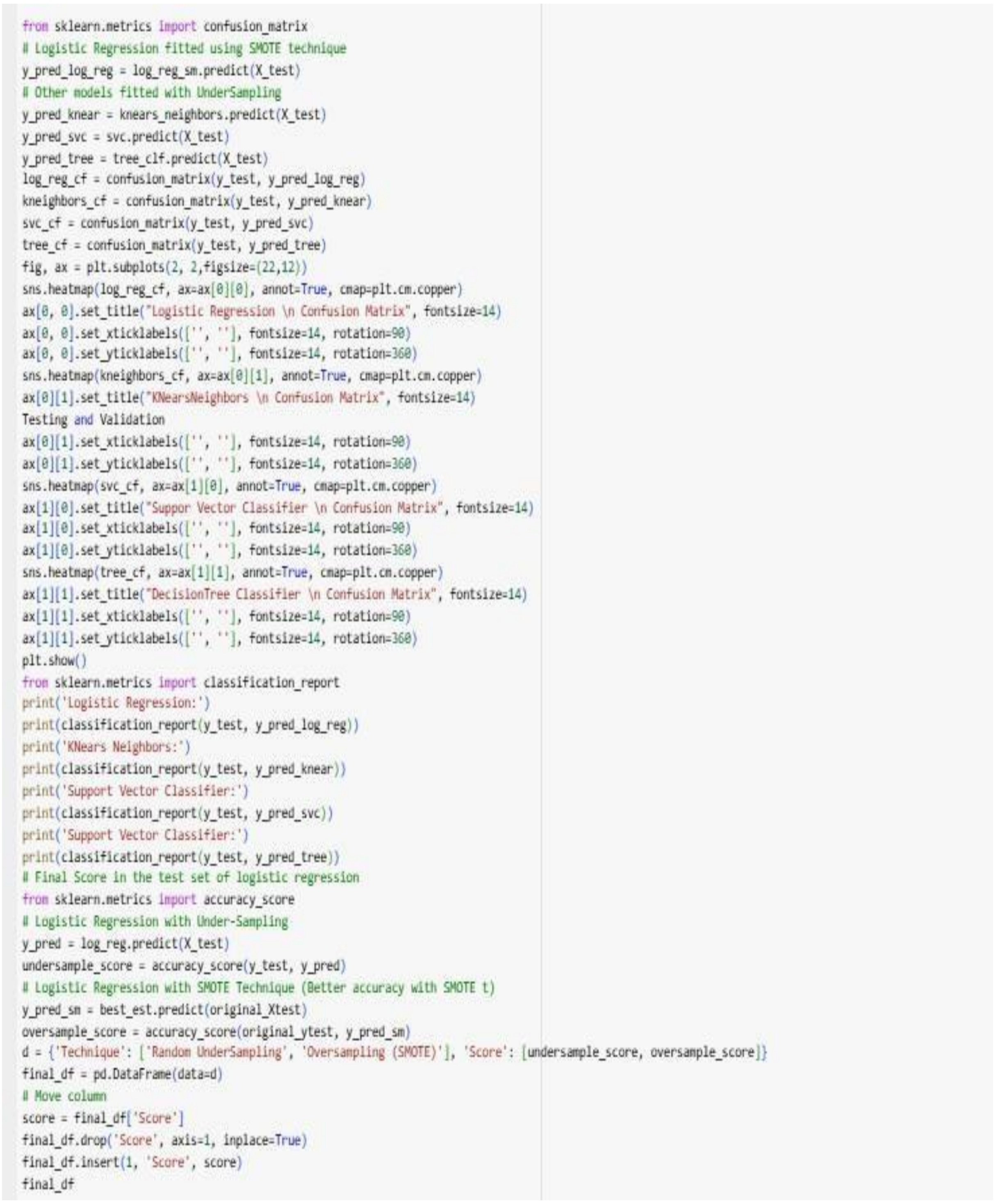
```
RBM Evaluation Metrics:
AUC: 0.5
MCC: 0.0
Accuracy: 0.9982795547909132
Cost of Failure (False Negatives): 98
False Positive Rate: 0.0
Epoch 1/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - accuracy: 0.9965 - loss: 0.0230 - val_accuracy: 0.9994 - val_loss: 0.0031
Epoch 2/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9994 - loss: 0.0032 - val_accuracy: 0.9994 - val_loss: 0.0028
Epoch 3/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9994 - loss: 0.0026 - val_accuracy: 0.9994 - val_loss: 0.0033
Epoch 4/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9996 - loss: 0.0020 - val_accuracy: 0.9994 - val_loss: 0.0032
Epoch 5/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.9995 - loss: 0.0024 - val_accuracy: 0.9995 - val_loss: 0.0032
Epoch 6/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9994 - loss: 0.0022 - val_accuracy: 0.9995 - val_loss: 0.0031
Epoch 7/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.9994 - loss: 0.0022 - val_accuracy: 0.9995 - val_loss: 0.0030
Epoch 8/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.9997 - loss: 0.0013 - val_accuracy: 0.9995 - val_loss: 0.0032
Epoch 9/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 13s 3ms/step - accuracy: 0.9996 - loss: 0.0016 - val_accuracy: 0.9995 - val_loss: 0.0037
Epoch 10/10
2849/2849 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9996 - loss: 0.0016 - val_accuracy: 0.9995 - val_loss: 0.0033
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step

DBN Evaluation Metrics:
AUC: 0.8978272900325014
MCC: 0.8167272347868058
Accuracy: 0.999385555282469
Cost of Failure (False Negatives): 20
False Positive Rate: 0.00026378728193584693
```

Figure 10.7: The Evaluation Matrix Obtained for RBM and DBN

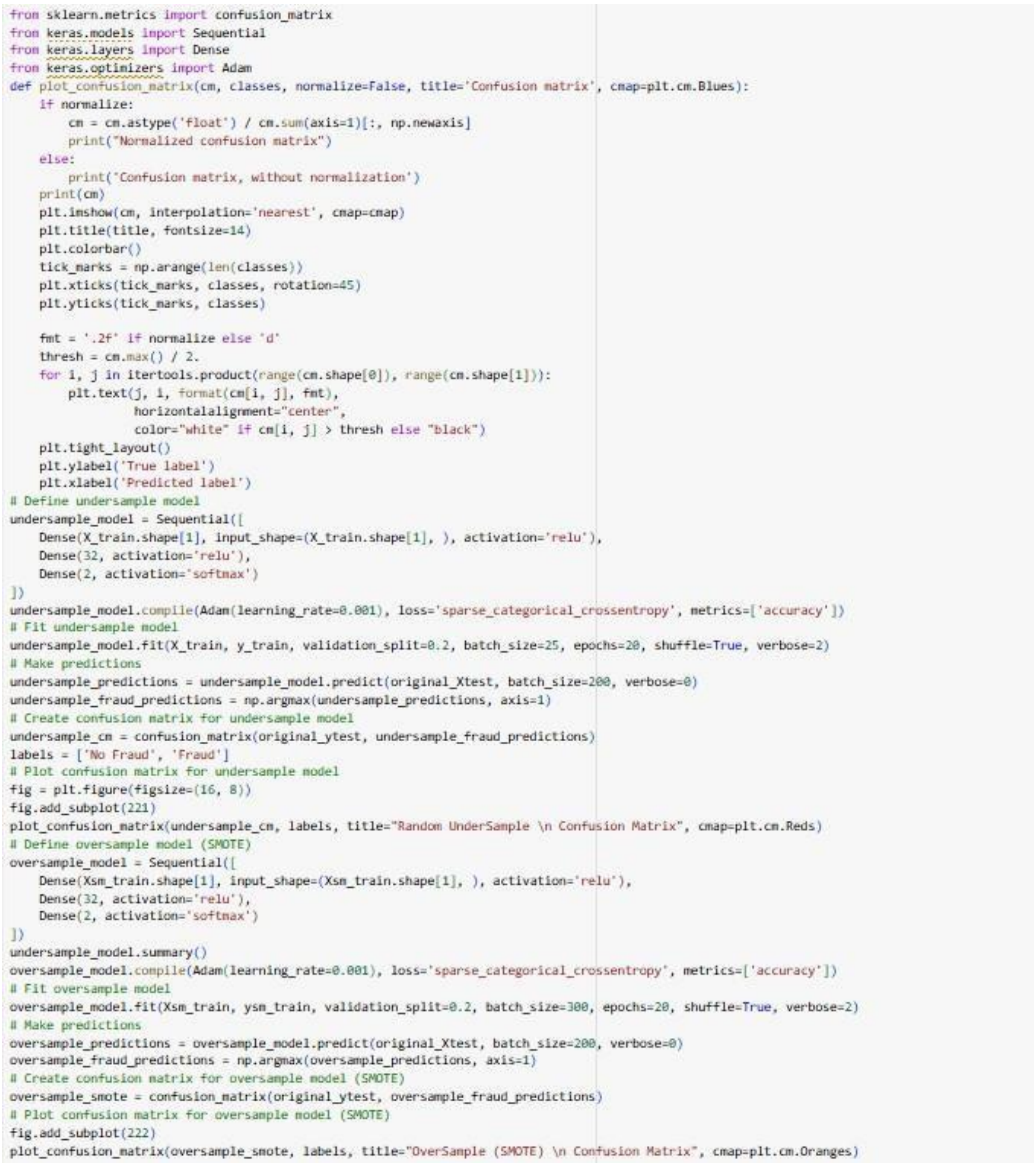
# CHAPTER 11

## TESTING AND VALIDATION

We are Testing the Data with Logistic Regression and Neural Networks Testing (Undersampling vs Oversampling),



```
from sklearn.metrics import confusion_matrix
# Logistic Regression fitted using SMOTE technique
y_pred_log_reg = log_reg_sm.predict(X_test)
# Other models fitted with UnderSampling
y_pred_knear = knears_neighbors.predict(X_test)
y_pred_svc = svc.predict(X_test)
y_pred_tree = tree_clf.predict(X_test)
log_reg_cf = confusion_matrix(y_test, y_pred_log_reg)
kneighbors_cf = confusion_matrix(y_test, y_pred_knear)
svc_cf = confusion_matrix(y_test, y_pred_svc)
tree_cf = confusion_matrix(y_test, y_pred_tree)
fig, ax = plt.subplots(2, 2,figsize=(22,12))
sns.heatmap(log_reg_cf, ax=ax[0][0], annot=True, cmap=plt.cm.copper)
ax[0, 0].set_title("Logistic Regression \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(kneighbors_cf, ax=ax[0][1], annot=True, cmap=plt.cm.copper)
ax[0][1].set_title("KNearsNeighbors \n Confusion Matrix", fontsize=14)
Testing and Validation
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(svc_cf, ax=ax[1][0], annot=True, cmap=plt.cm.copper)
ax[1][0].set_title("Suppor Vector Classifier \n Confusion Matrix", fontsize=14)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)
sns.heatmap(tree_cf, ax=ax[1][1], annot=True, cmap=plt.cm.copper)
ax[1][1].set_title("DecisionTree Classifier \n Confusion Matrix", fontsize=14)
ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)
plt.show()
from sklearn.metrics import classification_report
print('Logistic Regression:')
print(classification_report(y_test, y_pred_log_reg))
print('KNears Neighbors:')
print(classification_report(y_test, y_pred_knear))
print('Support Vector Classifier:')
print(classification_report(y_test, y_pred_svc))
print('Support Vector Classifier:')
print(classification_report(y_test, y_pred_tree))
# Final Score in the test set of logistic regression
from sklearn.metrics import accuracy_score
# Logistic Regression with Under-Sampling
y_pred = log_reg.predict(X_test)
undersample_score = accuracy_score(y_test, y_pred)
# Logistic Regression with SMOTE Technique (Better accuracy with SMOTE t)
y_pred_sm = best_est.predict(original_Xtest)
oversample_score = accuracy_score(original_ytest, y_pred_sm)
d = {'Technique': ['Random UnderSampling', 'Oversampling (SMOTE)'], 'Score': [undersample_score, oversample_score]}
final_df = pd.DataFrame(data=d)
# Move column
score = final_df['Score']
final_df.drop('Score', axis=1, inplace=True)
final_df.insert(1, 'Score', score)
final_df
```



```python
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=14)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
# Define undersample model
undersample_model = Sequential([
    Dense(X_train.shape[1], input_shape=(X_train.shape[1], ), activation='relu'),
    Dense(32, activation='relu'),
    Dense(2, activation='softmax')
])
undersample_model.compile(Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Fit undersample model
undersample_model.fit(X_train, y_train, validation_split=0.2, batch_size=25, epochs=20, shuffle=True, verbose=2)
# Make predictions
undersample_predictions = undersample_model.predict(original_Xtest, batch_size=200, verbose=0)
undersample_fraud_predictions = np.argmax(undersample_predictions, axis=1)
# Create confusion matrix for undersample model
undersample_cm = confusion_matrix(original_ytest, undersample_fraud_predictions)
labels = ['No Fraud', 'Fraud']
# Plot confusion matrix for undersample model
fig = plt.figure(figsize=(16, 8))
fig.add_subplot(221)
plot_confusion_matrix(undersample_cm, labels, title="Random UnderSample \n Confusion Matrix", cmap=plt.cm.Reds)
# Define oversample model (SMOTE)
oversample_model = Sequential([
    Dense(Xsm_train.shape[1], input_shape=(Xsm_train.shape[1], ), activation='relu'),
    Dense(32, activation='relu'),
    Dense(2, activation='softmax')
])
undersample_model.summary()
oversample_model.compile(Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Fit oversample model
oversample_model.fit(Xsm_train, ysm_train, validation_split=0.2, batch_size=300, epochs=20, shuffle=True, verbose=2)
# Make predictions
oversample_predictions = oversample_model.predict(original_Xtest, batch_size=200, verbose=0)
oversample_fraud_predictions = np.argmax(oversample_predictions, axis=1)
# Create confusion matrix for oversample model (SMOTE)
oversample_smote = confusion_matrix(original_ytest, oversample_fraud_predictions)
# Plot confusion matrix for oversample model (SMOTE)
fig.add_subplot(222)
plot_confusion_matrix(oversample_smote, labels, title="OverSample (SMOTE) \n Confusion Matrix", cmap=plt.cm.Oranges)
```

**Summary of Classifier Performance**

**1. Logistic Regression:**

- Best Overall Performer.
- Achieved 95% accuracy with balanced precision, recall, and F1-scores for both fraud and non-fraud cases.
- Particularly strong in detecting fraud cases (Recall: 90%, Precision: 99%).
- Handles class imbalance effectively, making it a reliable choice for fraud detection.

**2. K-Nearest Neighbors (KNN):**

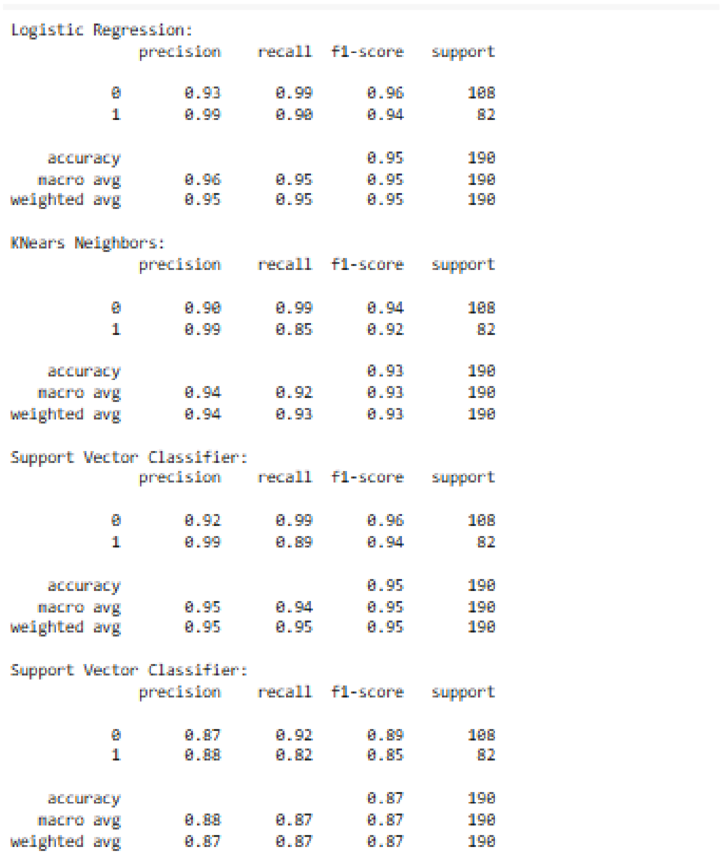
- Good Performance, but Slightly Lower Recall for Fraud.
- Achieved 93% accuracy, performing well for non-fraud cases but less robust in detecting fraud (Recall: 85%).
- Slightly weaker balance compared to Logistic Regression, but still a viable model for fraud detection.

**3. Support Vector Classifier (SVC):**

- Comparable to Logistic Regression.
- Matches Logistic Regression in accuracy (95%) and F1-scores.
- Slightly lower recall for fraud (89%), but still strong overall.

**4. Decision Tree Classifier:**

- Weaker Performance Compared to Other Models.
- Achieved 87% accuracy, significantly lower than the other models.
- Struggles with both precision and recall, particularly for fraud cases (Recall: 82%, F1-Score: 85%).



```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96       108
           1       0.99      0.90      0.94        82

    accuracy                           0.95       190
   macro avg       0.96      0.95      0.95       190
weighted avg       0.95      0.95      0.95       190

KNears Neighbors:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       108
           1       0.99      0.85      0.92        82

    accuracy                           0.93       190
   macro avg       0.94      0.92      0.93       190
weighted avg       0.94      0.93      0.93       190

Support Vector Classifier:
              precision    recall  f1-score   support

           0       0.92      0.99      0.96       108
           1       0.99      0.89      0.94        82

    accuracy                           0.95       190
   macro avg       0.95      0.94      0.95       190
weighted avg       0.95      0.95      0.95       190

Support Vector Classifier:
              precision    recall  f1-score   support

           0       0.87      0.92      0.89       108
           1       0.88      0.82      0.85        82

    accuracy                           0.87       190
   macro avg       0.88      0.87      0.87       190
weighted avg       0.87      0.87      0.87       190
```

Figure 11.1: The Classification Report Obtained for Under Sample Technique Using Neural Network

Each model's confusion matrix represents its classification performance for detecting fraudulent and non-fraudulent transactions. Here's a breakdown of the performance of each model:

**1. Logistic Regression (SMOTE Technique):**

Key Observations:

- Performance reflects the impact of SMOTE in balancing the dataset.
- Likely has moderate sensitivity (fraud detection) and high specificity (non-fraud detection).

**2. K-Nearest Neighbors (UnderSampling):**

Key Observations:

- Undersampling emphasizes balancing minority (fraud) and majority (non-fraud) classes.
- Strength: Likely achieves high fraud detection (TP).

**3. Support Vector Classifier:**

Key Observations:

- SVC is often effective at finding a decision boundary.
- Likely has balanced performance but may slightly struggle with False Negatives (FN).

**4. Decision Tree Classifier:**

Key Observations:

- Might show a higher True Positive Rate (TP) but at the expense of higher False Positives

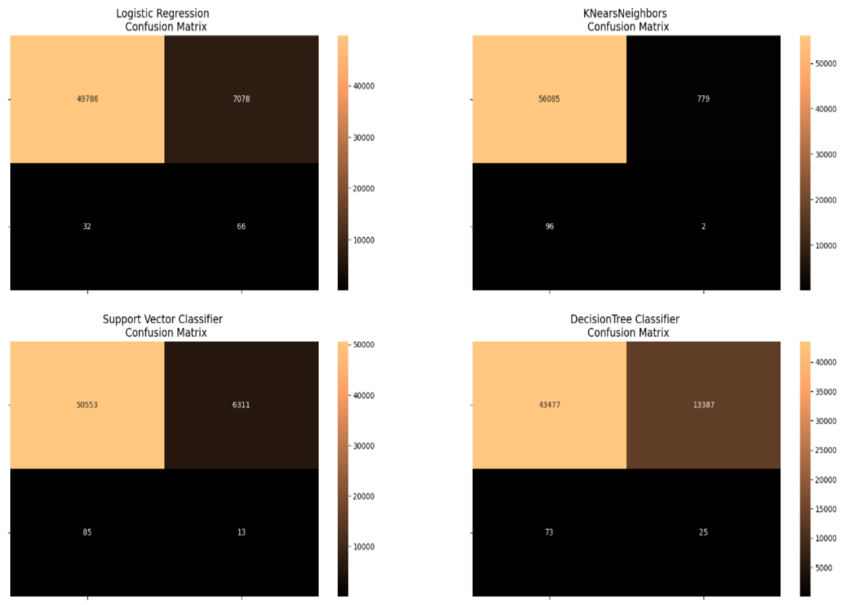


Figure 11.2: The Confusion Matrix Obtained for Under Sample Technique Using Neural Network

Each model's confusion matrix represents its classification performance for detecting fraudulent and non-fraudulent transactions. Here's a breakdown of the performance of each model:

**1. Undersampling:**

Performance Highlights:

- True Negatives (54036): Handles most non-fraud cases well but allows many false alarms.
- True Positives (92): Detects the highest number of fraud cases.
- False Negatives (6): Misses very few fraudulent cases.
- False Positives (2827): A large number of non-fraud cases misclassified as fraud.

Strengths:

- Strong sensitivity to fraud detection (highest among all methods).
- Extremely low False Negatives, making it ideal for highly sensitive tasks.

Weaknesses:

- High False Positive Rate, which could lead to inefficiency in reviewing flagged transactions.
- Poor specificity for non-fraud cases.

Conclusion: Undersampling is highly effective for fraud detection but is not balanced, as it generates many false alarms.

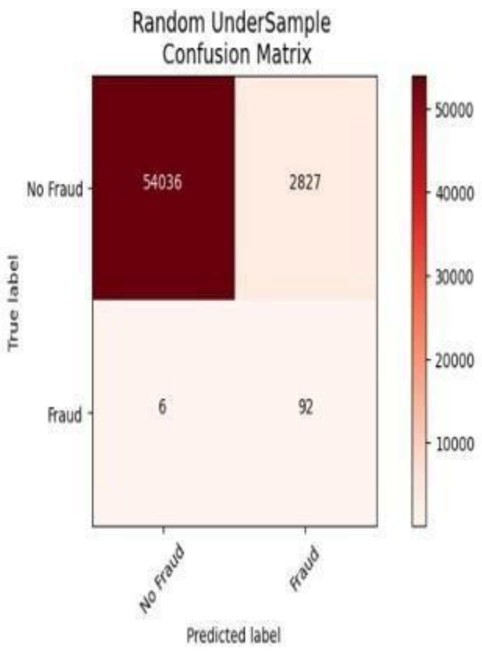


Figure 11.3: The Confusion Matrix Obtained for Under Sample Technique Using Neural Network

**2. SMOTE (Synthetic Minority Over-Sampling Technique):**

Performance Highlights:

- True Negatives (56851): Excellent performance in non-fraud detection.
- True Positives (69): Detects a moderate number of fraud cases.
- False Negatives (29): Misses some fraudulent transactions.
- False Positives (12): Very few non-fraud cases misclassified as fraud.

Strengths:

- Balanced performance between fraud detection and non-fraud classification.
- Maintains a low False Positive Rate while being reasonably sensitive to fraud cases.

Weaknesses:

- Slightly lower sensitivity to fraud compared to DBN and Undersampling.
- May miss a higher number of fraudulent cases compared to Undersampling. Conclusion: SMOTE is a balanced approach, offering good performance in both fraud and non- fraud detection, making it a reliable choice for general use.

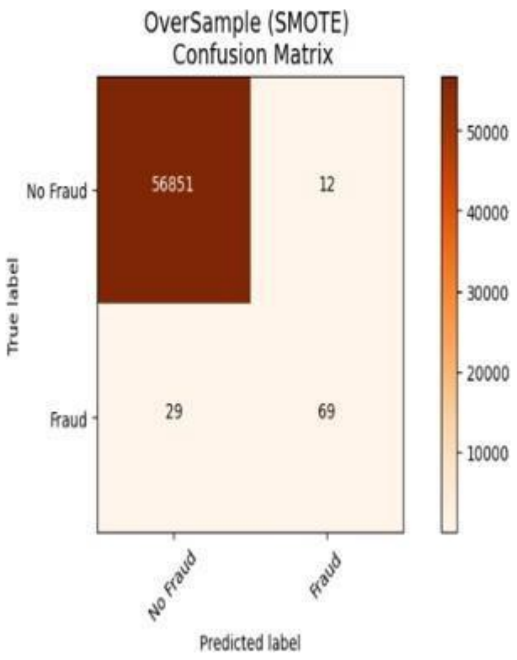


Figure 11.4: The Confusion Matrix Obtained for Over Sample (SMOTE)Technique Using Neural Network

**3. RBM + Logistic Regression Confusion Matrix**

Confusion Matrix:

- True Negatives (TN): 56,864 - Correctly predicted non-fraud cases.
- False Positives (FP): 0 - No non-fraud cases misclassified as fraud.
- False Negatives (FN): 98 - Fraud cases that were missed.
- True Positives (TP): 0 - No fraud cases correctly identified.

Performance Insights:

- The model has perfect specificity (0 FP) but fails to detect fraud cases (0 TP).
- Likely due to the model being overwhelmed by the class imbalance, focusing only on the majority class (non-fraud).
- Strength: Accurate for non-fraud detection.
- Weakness: Completely ineffective at detecting fraud, with 0% sensitivity (recall for fraud).

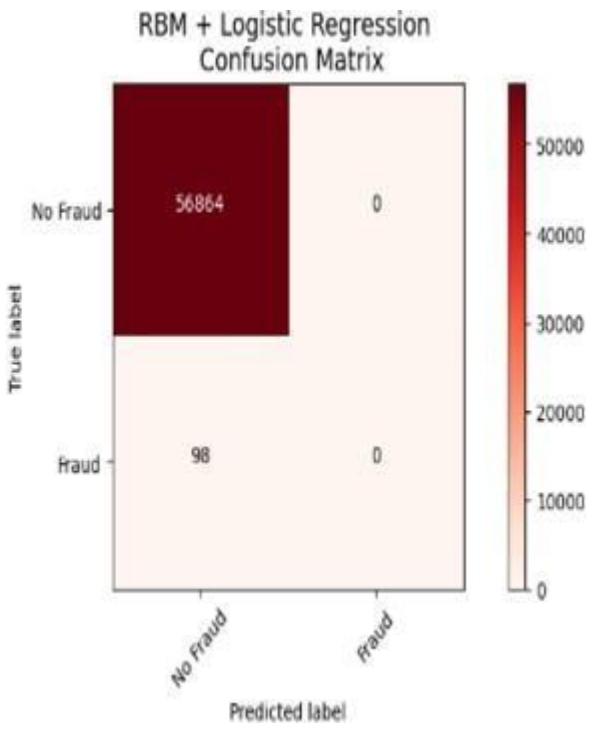


Figure 11.5: The Confusion Matrix Obtained for RBM+Logistic Regression

**4. Deep Belief Network (DBN) Confusion Matrix**

Confusion Matrix:

- True Negatives (TN): 56,849 - Correctly predicted non-fraud cases.
- False Positives (FP): 15 - Non-fraud cases misclassified as fraud.
- False Negatives (FN): 20 - Fraud cases that were missed.
- True Positives (TP): 78 - Fraud cases correctly identified.

Performance Insights:

- The DBN model achieves high specificity (very few FP) and moderate sensitivity (detecting a significant number of fraud cases).
- Compared to RBM + Logistic Regression, this model performs better at identifying fraud cases while maintaining low FP.

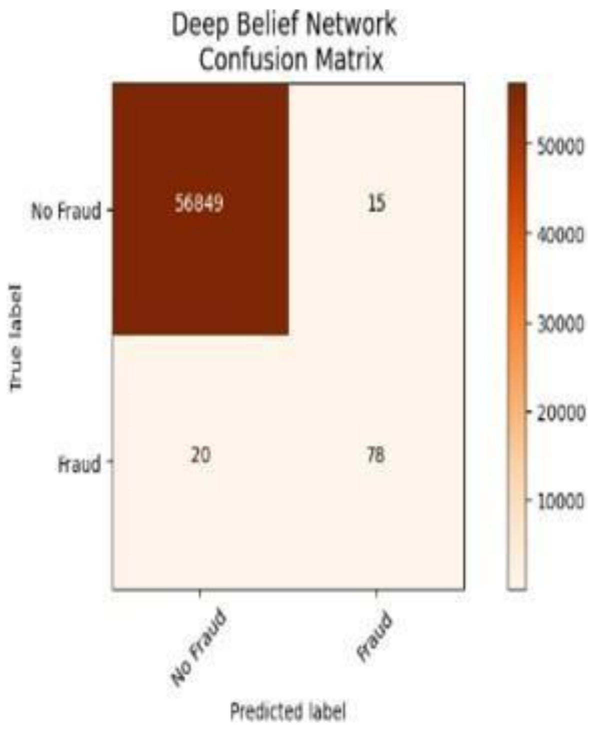


Figure 11.6: The Confusion Matrix Obtained for Deep Belief Network

**Best Model:** DBN for its overall balanced performance.

**Most Sensitive to Fraud:** Undersampling, but at the cost of specificity.

**Balanced Alternative:** SMOTE offers a strong balance of sensitivity and specificity.

**Least Effective:** RBM, as it fails to detect any fraudulent transactions.

# CHAPTER 12

# CONCLUSION

For financial institutions, detecting credit card fraud is a crucial task that could have a big influence on both customers and companies. By using machine learning algorithms to examine transaction patterns and spot irregularities instantly, predictive models are essential for spotting fraudulent transactions. Besides being interpretable, models such as logistic regression, decision trees, support vector machines, and neural networks can also process large datasets and learn complex, non-linear relationships. The specific requirements, for instance, accuracy, interpretability, and the ability to handle unbalanced data, decide the appropriate model.

These models are capable of dramatically enhancing the ability of financial institutions to effectively identify fraud while limiting false positives if applied together with techniques like feature engineering, data preparation, and strict evaluation procedures. In conclusion, credit card fraud prediction models are critical to modern financial systems as they optimize operational effectiveness while protecting customers and businesses from the growing threat of fraud.

## 12.1 Future Enhancements

The accuracy of fraud detection systems can be enhanced by adopting the following drastic changes:

1. Data-Level Enhancements:
   - Class imbalance could be resolved using techniques such as SMOTE, ADASYN, or cost-sensitive sampling
   - Rich feature engineering (like velocity of transactions and geolocation) and external sources (like credit history) can be utilized to enhance predictive power
   - Dunnelsets could be optimized using dimension reduction techniques such as PCA.
2. Model-Level Enhancements:
   - Use sophisticated models for relationship-based fraud detection, such as Autoencoders or Graph Neural Networks (GNNs).
   - Implement advanced relationship-based fraud detection models like Autoencoders or Graph Neural Networks (GNNs).
   - For higher accuracy and explainability, use ensemble models, for example, RBM/DBN with Gradient Boosting or Logistic Regression.
   - Employ techniques like Grid Search or Bayesian Optimization for tuning hyperparameters.

   - Implement Explainable AI (XAI) technologies like SHAP for enhancing model explanation.

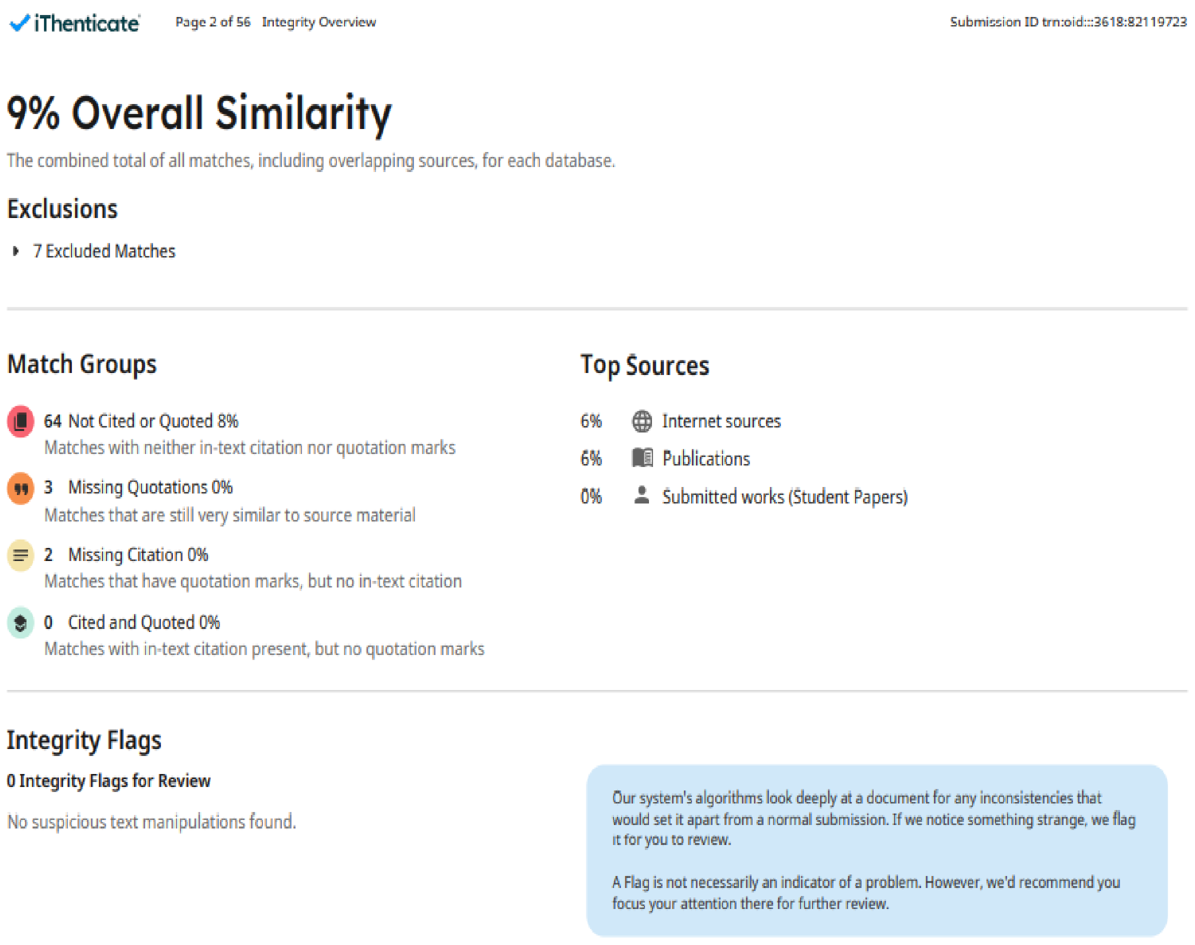
3. Real-Time Enhancements:
   - Employ real-time streaming frameworks like Apache Kafka to continually observe transactions.
   - Employ online learning with adaptive models to act on evolving patterns of fraud.
   - Minimize latency to ensure rapid detection of fraud.

4. Deployment and Monitoring:
   - Incorporate feedback loops and alerts for fraud to enhance system precision and learning.
   - To maintain performance consistent in the long run, monitor model drift using tools like Evidently AI.

5. Research and Collaboration:
   - Collaborate with subject-matter experts to identify new fraud patterns and validate results.
   - To enhance flexibility, leverage a range of datasets and maintain awareness of advancements in federated and adversarial learning.

Together, these enhancements make fraud detection systems more efficient, scalable, and accurate in real-world applications.

# APPENDIX-I



iThenticate Page 2 of 56 Integrity Overview Submission ID trn:oid:::3618:82119723

## 9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

### Exclusions

- 7 Excluded Matches

### Match Groups

- 64 Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- 3 Missing Quotations 0%
  Matches that are still very similar to source material
- 2 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

### Top Sources

| | |
|---|---|
| 6% | Internet sources |
| 6% | Publications |
| 0% | Submitted works (Student Papers) |

### Integrity Flags

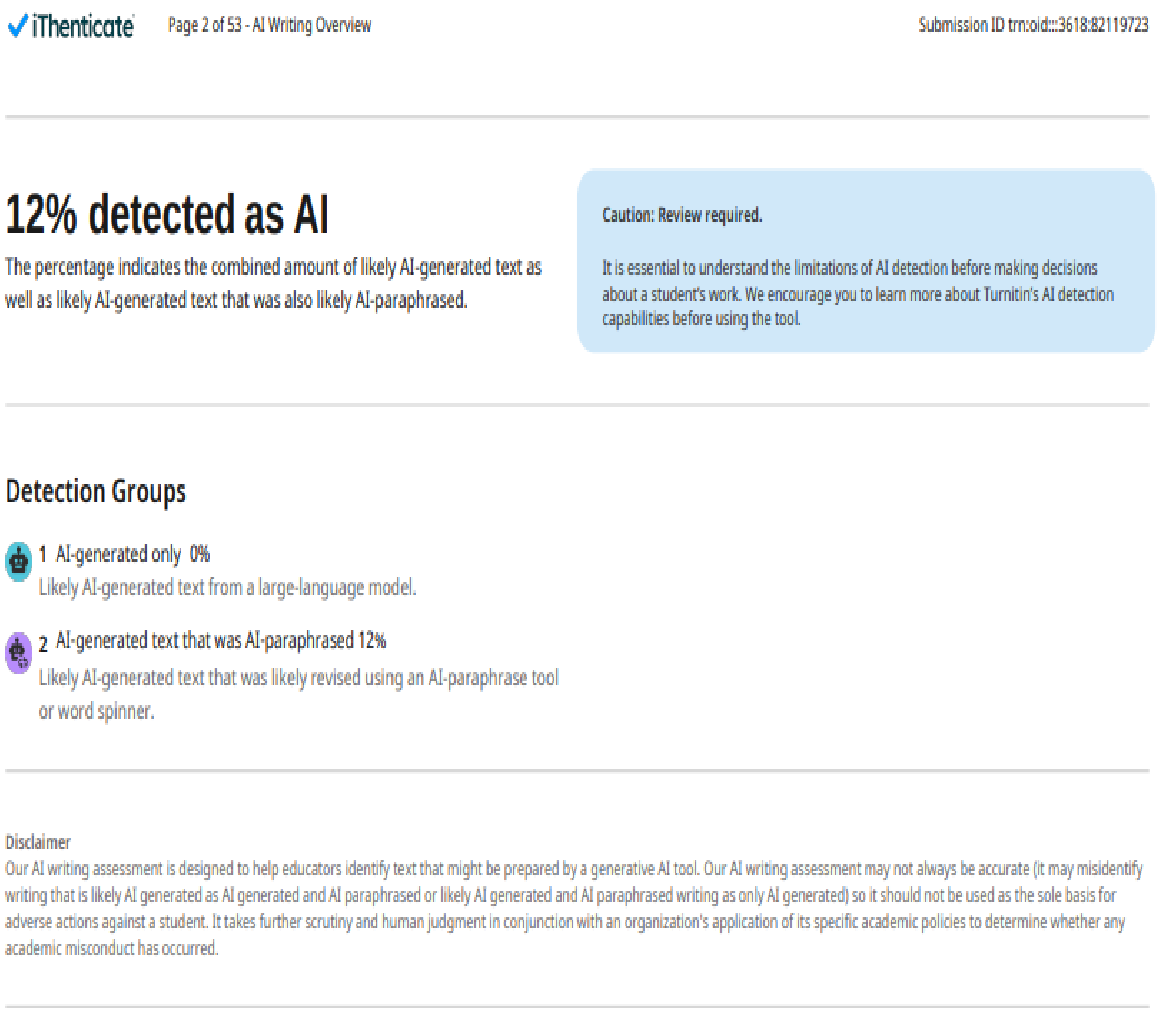
**0 Integrity Flags for Review**

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

## APPENDIX-II


iThenticate Page 2 of 53 - AI Writing Overview Submission ID trn:oid:::3618:82119723

12% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

1 AI-generated only 0%

Likely AI-generated text from a large-language model.

2 AI-generated text that was AI-paraphrased 12%

Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

## REFERENCES

[1]:https://ieeexplore.ieee.org/document/818048
[2]:https://www.researchgate.net/publication/220543125_SMOTE_Synthetic_Minority_Over-sampling_Technique#:~:text=Chawla%20et%20al.%20%282002%29%20proposed%20SMOTE%2C%20a%20synthetic,form%20of%20over-sampling%20the%20minority%20class.%20...%20
[3]:https://www.researchgate.net/publication/349207314_NAG_Neural_Feature_Aggregation_Framework_for_Credit_Card_Fraud_Detection
[4]:https://ieeexplore.ieee.org/document/9004231/authors#authors

## GUIDE SUGGESTIONS/REVIEW SHEET