A
Project Report
On

"Credit Card Fraud Detection Using Machine Learning"

Submitted by

Shaikh Afnan Ahmed

Class: MCA II Year

Guided By
Dr. Anagha K. Joshi

SSBES'S
Institute of Technology & Management Nanded

Affiliated to

Swami Ramanand Teerth Marathwada University, Nanded

2024-2025

Shri Sharda Bhavan Education Society's
INSTITUTE OF TECHNOLOGY & MANAGEMENT
Department of Computer Science
VIP Road, Nanded-431 602(M.S.), INDIA
www.ssbesitm.org E-mail ssbesitm@yahoo.com (Ph.02462-254850)
Accredited by NAAC, Grade 'B' with 2.49 CGPA

CERTIFICATE

This is to certify that the project report entitled as

"Credit Card Fraud Detection Using Machine Learning"

Submitted by

Shaikh Afnan Ahmed

In partial fulfillment for the Degree of M.C.A. Course at

"Institute of Technology and Management, Nanded"

Affiliated to

Swami Ramanand Teerth Marathwada University, Nanded.

2024-2025

Dr. Anagha K. Joshi
Project Guide

Dr. M.M. Bokare
Head of Dept.

Dr. S.B. Thorat
Director

ACKNOWLEDGEMENT

The success and final outcome of this Project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my Project. All that I have done is only due to such supervision and assistance and I would like to extend my sincere thanks to all of them.

I am highly indebted to **Dr. Anagha K. Joshi** for her guidance and constant supervision as well as for providing necessary information regarding this project & also for her support in completing the same.

I would like to thank **Dr. S. B. Thorat**, Director, SSBES's Institute of Technology and Management, Nanded for providing me an opportunity to do this work in Institute of Technology and Management, Nanded and giving us all support and guidance.
Also I would like to thank **Mr. M. M. Bokare**, HOD, Dept. of Computer Science, for his encouragement and more over for his timely support and guidance till the completion of Project work.

I would like to express my gratitude towards my parents their kind co-operation and encouragement which help me in completion of this project.

My thanks and appreciations also go to my friends in developing the project and people who have willingly helped me out with their abilities.

Shaikh Afnan Ahmed

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Credit card fraud poses a major financial and security challenge, requiring advanced techniques for timely and accurate detection. This project implements machine learning-based fraud detection using real-world credit card transaction data. The dataset is highly imbalanced, with fraudulent transactions making up only a small fraction of the total. To address this, data preprocessing, feature selection, and resampling techniques are applied before training models such as Logistic Regression for classification. The system is evaluated using performance metrics like accuracy, precision, recall, and F1-score to ensure reliability. The results demonstrate that machine learning enhances fraud detection efficiency, reducing false positives and improving security**.** Future enhancements include deep learning models, real-time fraud detection APIs, and improved anomaly detection methods to strengthen financial security and prevent fraudulent transactions more effectively.

1. INTRODUCTION

Credit card fraud is a growing concern in the financial sector, leading to significant economic losses each year. As digital transactions increase, so do fraudulent activities, making it essential to develop automated fraud detection systems that can efficiently identify suspicious transactions. Traditional fraud detection methods rely on rule-based systems and manual verification, which are time-consuming, inefficient, and prone to errors. This project aims to detect fraudulent credit card transactions using machine learning techniques, providing a scalable and effective solution to combat fraud.

Fraudulent transactions are becoming increasingly sophisticated, often bypassing conventional detection mechanisms. Fraudsters use tactics such as identity theft, account takeover, phishing, and synthetic identity fraud, making it difficult for traditional methods to detect fraudulent activities accurately. Machine learning provides a dynamic and adaptive approach by recognizing complex patterns in transaction data that indicate fraudulent behavior. Instead of relying on static rules, machine learning algorithms can continuously learn and adapt to new fraud trends, improving detection accuracy over time.

The system uses supervised learning algorithms trained on real-world transaction data, which consists of features such as transaction amount, time, and various anonymized numerical attributes. Since fraud occurs in a highly imbalanced dataset, where fraudulent transactions represent only a small percentage of total transactions, data preprocessing techniques like resampling are used to improve model performance. The project implements Logistic Regression as the primary classifier to differentiate between legitimate and fraudulent transactions. The model is evaluated using performance metrics such as accuracy, precision, recall, and F1-score to ensure effectiveness in real-world applications.

One of the major challenges in fraud detection is the trade-off between false positives and false negatives. A false positive occurs when a legitimate transaction is incorrectly flagged as fraud, causing inconvenience to the user, while a false negative

means a fraudulent transaction goes undetected, leading to financial losses. This project aims to minimize both types of errors by tuning the threshold values, optimizing the model, and using advanced evaluation techniques.

Additionally, the implementation of this project can be extended beyond credit card fraud detection to other domains such as online banking fraud, insurance fraud, e-commerce fraud, and financial cybercrime prevention. Organizations can integrate this machine learning-based approach into fraud monitoring systems, automated alert mechanisms, and real-time fraud prevention solutions.

This machine learning-based approach enhances fraud detection by recognizing subtle patterns that may not be visible to rule-based systems, reducing false positives and increasing detection rates. The system can be further extended with deep learning models, anomaly detection techniques, and real-time fraud detection mechanisms for improved performance. By leveraging cutting-edge machine learning methodologies, this project offers a practical and efficient solution to help financial institutions combat fraud and safeguard users' transactions.

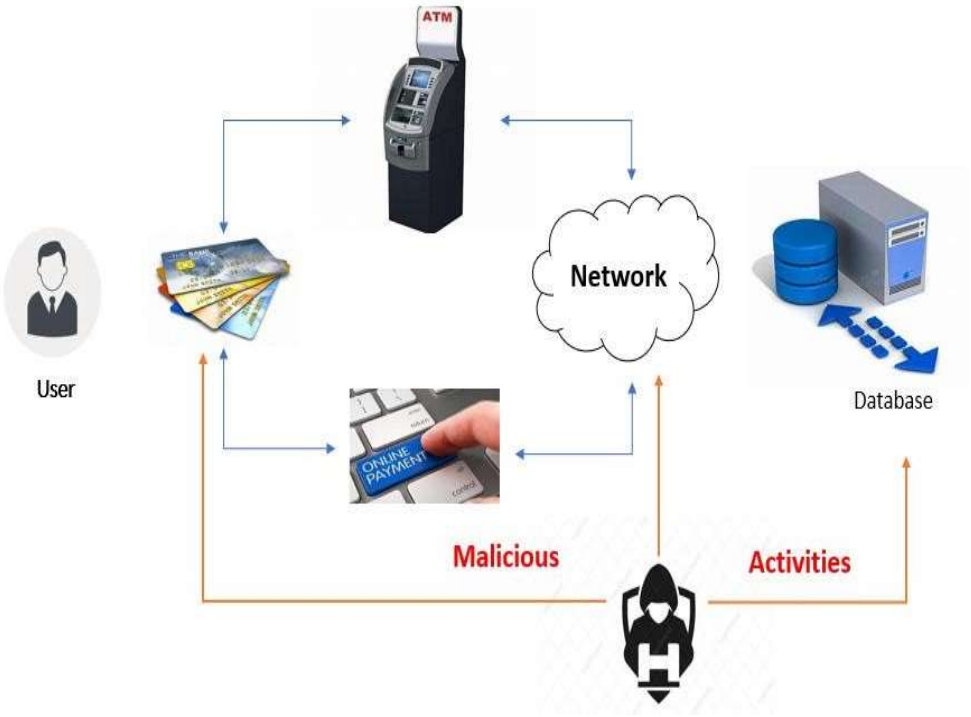


Fig.1. General Scenario of Online Fraud

DATA SOURCE

The dataset used for this project is a real-world transactional dataset containing both fraudulent and genuine transactions. It is designed to help in building and evaluating machine learning models for fraud detection.

1. Source of the Dataset

The dataset is often sourced from financial institutions, open-source repositories like Kaggle, or research papers. One of the widely used datasets is the European Credit Card Fraud Detection Dataset, which consists of anonymized transaction records.

2. Features in the Dataset

The dataset contains various transaction attributes that help identify fraudulent behavior. Some key features include:

- **Time:** The timestamp of the transaction (relative to the first transaction).
- **Amount:** The transaction amount in monetary units.
- **Transaction Type:** Whether the transaction is a purchase, withdrawal, or transfer.
- **Location:** The geographical location where the transaction occurred.
- **Card Type:** The type of credit card used (e.g., Visa, Mastercard).
- **User ID:** Unique identifier for the customer performing the transaction.
- **Merchant ID:** Unique identifier for the merchant accepting the payment.
- **Transaction Status:** Whether the transaction is fraudulent (1) or genuine (0).

3. Data Imbalance Issue

Since fraudulent transactions are rare, datasets are often highly imbalanced. In most cases:

- **Genuine Transactions:** 98-99% of the data.
- **Fraudulent Transactions:** Only 1-2% of the data. This imbalance poses a challenge in model training, requiring techniques like oversampling, undersampling, or Synthetic Minority Over-sampling Technique (SMOTE).

4. Preprocessing of the Dataset

Before using the dataset for model training, the following preprocessing steps are performed:

- **Handling Missing Values:** Removing or imputing missing data.
- **Feature Scaling:** Normalizing numerical values to improve model efficiency.
- **Encoding Categorical Data:** Converting text-based categories (e.g., location, transaction type) into numerical values.
- **Balancing the Data:** Applying resampling techniques to ensure the model does not become biased towards genuine transactions.

5. Dataset Usage in Fraud Detection Model

The dataset is divided into **training and testing sets**, where the training set helps the machine learning model learn patterns, and the testing set evaluates its accuracy. Various machine learning algorithms such as Logistic Regression, Random Forest, Decision Trees, XGBoost, and Neural Networks can be applied to classify transactions as fraudulent or genuine.

Dataset Link: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

1.1 OVERVIEW OF THE PROJECT

This project focuses on the detection of fraudulent credit card transactions using machine learning techniques. In the modern digital economy, credit card fraud poses a significant threat to financial institutions and customers alike. The primary objective of this project is to develop a robust and efficient fraud detection system that can accurately classify transactions as genuine or fraudulent in real time. The dataset used is highly imbalanced, with a very small percentage of fraud cases, which presents a challenge in model training and evaluation.

To address this, data preprocessing techniques such as feature scaling and data balancing using both undersampling and oversampling (SMOTE) were employed. Multiple machine learning models including Logistic Regression, Decision Tree, and Random Forest were trained and evaluated using performance metrics like accuracy, precision, recall, and F1-score. The model demonstrating optimal performance was then deployed using Streamlit, providing a user-friendly web interface for real-time prediction. This system aims to aid banks and financial organizations in identifying suspicious activities efficiently, thereby minimizing financial losses and enhancing security.

1.2 OBJECTIVES

The primary objectives of this project are:

- To analyze and preprocess real-world credit card transaction data.
- To handle class imbalance using appropriate resampling techniques.
- To implement machine learning models for fraud detection.
- To evaluate model performance using classification metrics.
- To provide insights into fraudulent transaction patterns for better financial security.

- To explore future enhancements like deep learning, real-time fraud detection, and advanced security mechanisms.

By achieving these objectives, the project aims to improve the accuracy, reliability, and efficiency of fraud detection systems while addressing real-world challenges such as data imbalance, false positives, and scalability.

1.3 MODULES OF THE PROJECT

The project is divided into several key modules:

1. Data Collection & Exploration – Understanding the dataset, identifying missing values, and exploring data distributions.
2. Data Preprocessing & Feature Engineering – Handling missing values, scaling features, and addressing class imbalance.
3. Model Selection & Training – Training machine learning algorithms such as Logistic Regression to detect fraud.
4. Evaluation & Performance Analysis – Assessing model effectiveness using accuracy, precision, recall, and F1-score.
5. Fraud Detection & Prediction – Using the trained model to predict fraudulent transactions and analyze results.
6. Future Enhancements – Exploring deep learning models, ensemble techniques, and real-time fraud detection APIs.

1.4 SCOPE OF THE PROJECT

This project provides a scalable and data-driven approach to fraud detection, reducing reliance on manual reviews and improving the efficiency of fraud prevention systems. By leveraging machine learning algorithms, financial institutions can automatically identify suspicious transactions, preventing unauthorized transactions before they occur. The project's findings can be applied to banks, payment gateways, and financial security systems to minimize fraud risks.

In the future, this system can be enhanced by integrating real-time fraud detection using deep learning, deploying fraud detection APIs, and improving model interpretability using explainable AI techniques. These enhancements will make fraud detection systems more robust, accurate, and adaptable to evolving fraud patterns.

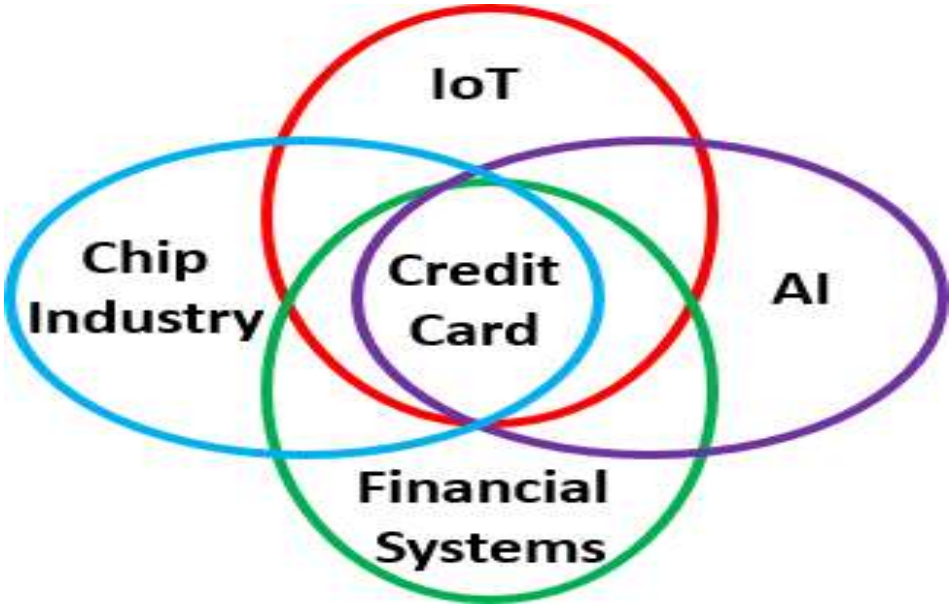


Fig.2. The Intersection of Credit Card Research and other Research Fields.

2. LITERATURE REVIEW

A literature review provides an overview of existing research, techniques, and advancements in credit card fraud detection using machine learning. It highlights the limitations of traditional methods and the benefits of AI-driven fraud detection **systems**. This review explores past studies, compares various approaches, and discusses the evolution of fraud detection techniques.

2.1 EXISTING SYSTEM

The traditional credit card fraud detection system relies on rule-based algorithms, statistical analysis, and manual verification. Banks and financial institutions have long used static threshold-based systems to detect suspicious transactions. These systems operate on predefined rules, such as flagging transactions that exceed a specific amount or occur in an unusual location.

However, as fraudsters develop more sophisticated techniques, these rule-based systems become less effective. They fail to adapt to evolving fraud patterns and struggle with handling large datasets in real-time.

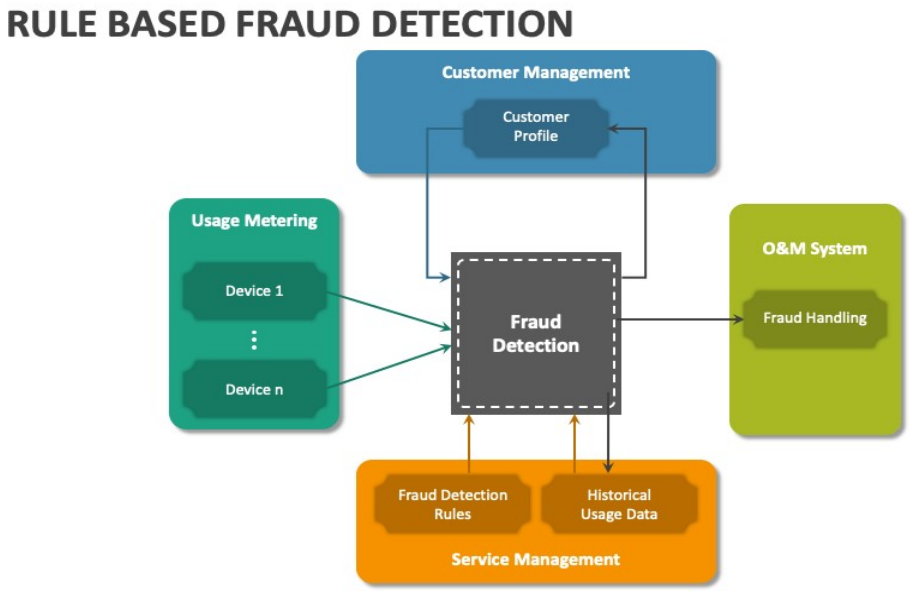


Fig.3. General Rule Based Fraud Detection System

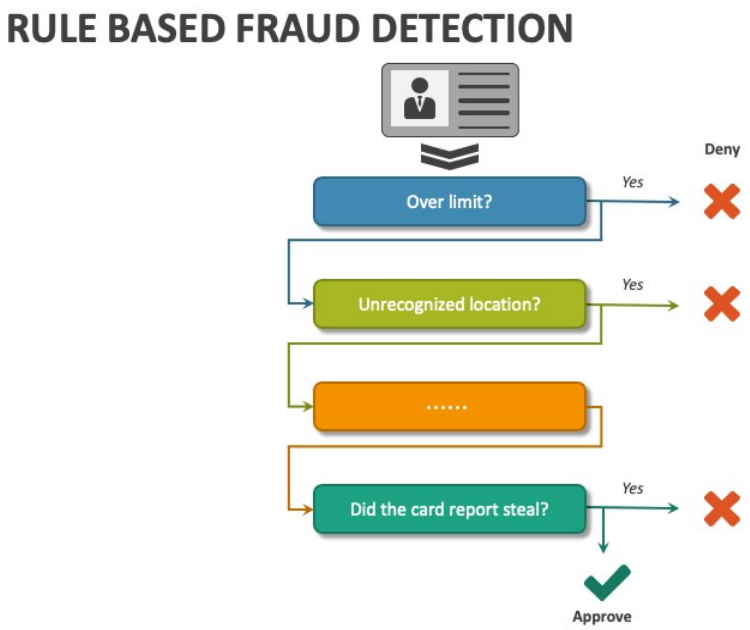


Fig.4. Working of Rule Based Fraud Detection System

2.1.1 DEMERITS OF THE EXISTING SYSTEM

- Despite its widespread use, the traditional approach to fraud detection suffers from several limitations:
- High False Positives & False Negatives
- Rule-based systems often generate excessive false positives, flagging legitimate transactions as fraudulent. This leads to customer dissatisfaction and inconvenience.
- At the same time, these systems fail to detect new fraud patterns, resulting in a high number of false negatives, where fraudulent transactions go undetected.
- Limited Adaptability
- Traditional fraud detection models rely on static rules that must be manually updated.
- Fraudsters continuously evolve their techniques, making rule-based systems ineffective against emerging fraud trends.
- Manual Review is Time-Consuming

- Financial institutions require human intervention to analyze flagged transactions, slowing down the process.

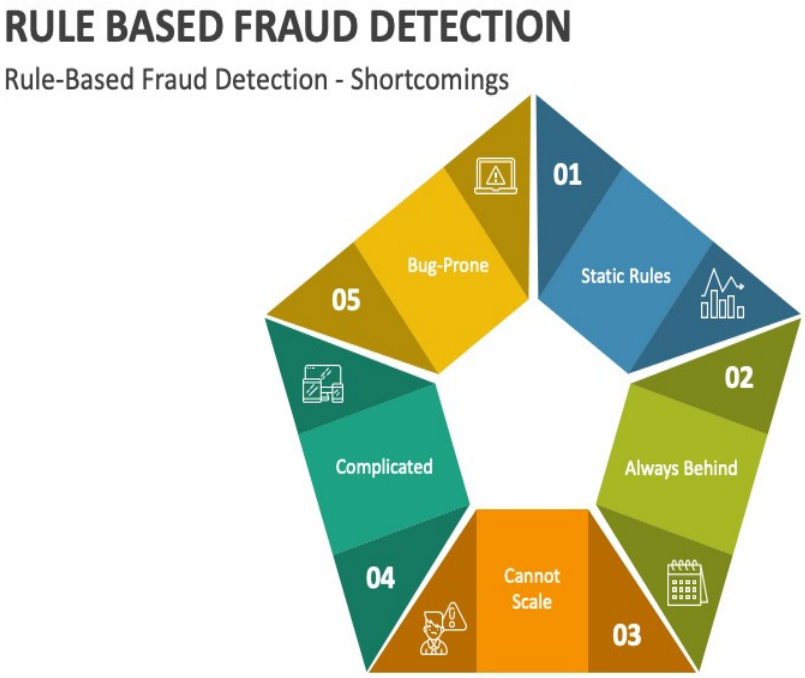


Fig.5. Shortcomings of Rule Based Fraud Detection System

- Fraud teams often experience delays in identifying fraudulent transactions, leading to financial losses.
- Scalability Issues
- With the increase in online transactions, the volume of data is too large for manual fraud detection.
- Traditional systems lack efficient data processing capabilities, making them unsuitable for real-time fraud detection.
- Inability to Detect Complex Fraud Patterns
- Fraudsters employ sophisticated methods, such as identity theft, card skimming, and transaction laundering, which rule-based systems fail to detect.
- Machine learning models, on the other hand, can recognize hidden relationships between fraudulent transactions.
- Previous Research on Fraud Detection Techniques
- Several researchers have proposed different fraud detection models, including:

- Bayesian Networks & Decision Trees: Early research focused on statistical models that used probabilistic analysis to predict fraudulent transactions.
- Anomaly Detection Techniques: Some studies applied unsupervised learning to detect outliers in transaction patterns.
- Neural Networks & Deep Learning: More recent research has explored the use of Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs) to improve fraud detection accuracy.

However, these methods often require large datasets, high computational power, and proper feature selection to achieve optimal results.

2.2 PROPOSED SYSTEM

To address the limitations of traditional fraud detection methods, machine learning-based fraud detection models provide a more efficient, scalable, and adaptable solution. Unlike static rule-based systems, these models use pattern recognition and predictive analytics to detect fraudulent transactions in real-time.

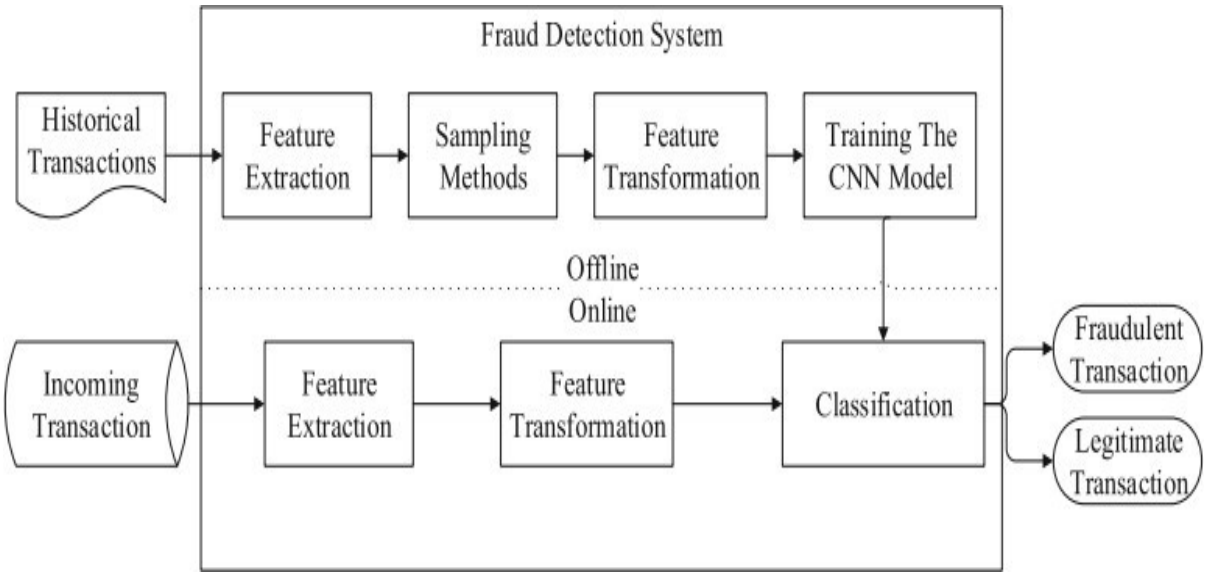


Fig.6. Proposed System Fraud Detection System

2.2.1 MERITS OF THE PROPOSED SYSTEM

1. **Higher Accuracy in Fraud Detection**
    - Machine learning models analyze transaction patterns, user behavior, and contextual data to accurately classify fraudulent and genuine transactions.
    - Algorithms such as Logistic Regression, Decision Trees, Random Forest, and XGBoost enhance fraud detection accuracy by minimizing false positives.
2. **Real-Time Fraud Detection**
    - ML algorithms process thousands of transactions per second, identifying fraud before the transaction is completed.
    - This prevents unauthorized access and reduces financial losses for banks and customers.
3. **Adaptability to New Fraud Techniques**
    - Unlike rule-based systems, machine learning models continuously learn from new fraud patterns.
    - Fraud detection accuracy improves over time as models adapt to evolving attack strategies.
4. **Reduction in False Positives**
    - Traditional methods block legitimate transactions due to strict rules.
    - ML-based models differentiate between fraud and unusual spending patterns, reducing unnecessary transaction declines.
5. **Efficient Handling of Large Datasets**
    - Advanced ML models process large volumes of transactional and behavioral data with minimal delay.
    - Techniques like batch processing, parallel computation, and cloud-based deployment make fraud detection scalable.
6. **Feature Engineering for Enhanced Detection**
    - Fraudulent transactions exhibit unique characteristics, such as location mismatches, frequent small transactions, and rapid withdrawals.

- Feature engineering techniques extract meaningful data points, improving model performance.

7. **Advanced Machine Learning Techniques**
   - Supervised Learning: Models train on labeled datasets to classify fraudulent transactions.
   - Unsupervised Learning: Detects anomalies without prior fraud labels using clustering techniques like K-Means and Isolation Forest.
   - Hybrid Approaches: Combine multiple algorithms for better fraud detection accuracy.

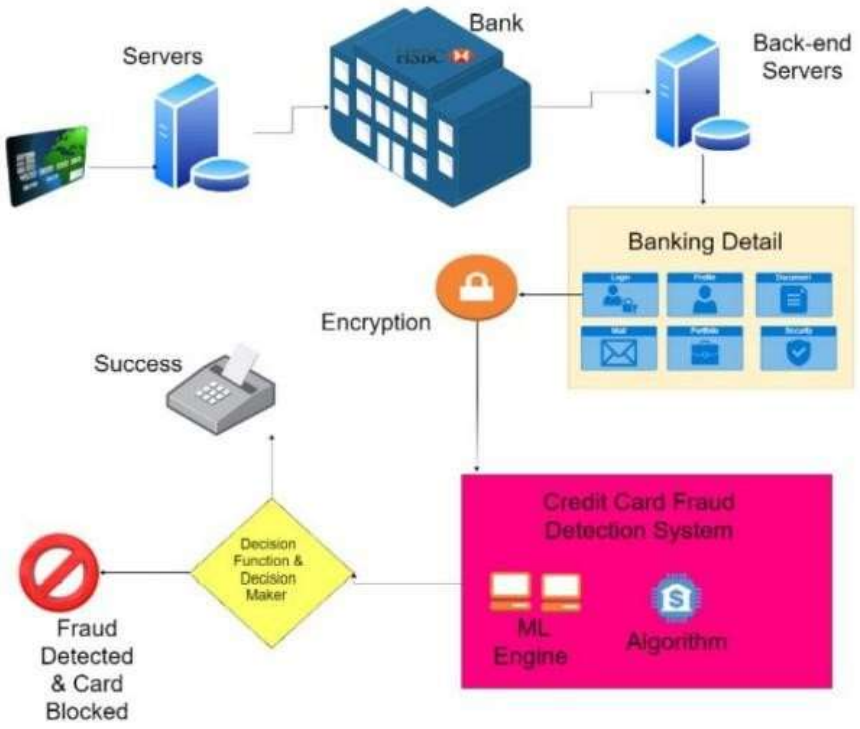


Fig.7. Data-Driven decision making

FUTURE ENHANCEMENTS IN FRAUD DETECTION

- **Deep Learning Models:** Advanced neural networks, such as Autoencoders and LSTMs, can further improve fraud detection.
- **Blockchain for Secure Transactions:** Decentralized transaction monitoring can reduce fraud risks.
- **Real-Time Fraud Prevention:** AI-driven fraud prevention systems can proactively block suspicious activities before they occur.

CONCLUSION

This literature review highlights the limitations of traditional fraud detection techniques and emphasizes the advantages of machine learning-based models**.** AI-powered fraud detection offers higher accuracy, real-time transaction monitoring, and adaptability to new fraud trends. With ongoing advancements in deep learning, big data analytics, and AI**,** fraud detection systems will continue to evolve, enhancing financial security for businesses and consumers.

3. SYSTEM SPECIFICATIONS

This chapter provides a detailed overview of the system specifications required to develop and deploy the Credit Card Fraud Detection System using Machine Learning. It includes the project category, software requirements, hardware requirements, and technology stack used in the implementation.

3.1 PROJECT CATEGORY

The project falls under multiple computer science disciplines, making it a multidisciplinary project involving:

1. Artificial Intelligence & Machine Learning

- The core of this project is based on machine learning (ML) algorithms that analyze credit card transactions and classify them as fraudulent or genuine.
- The model is trained using a dataset containing past transaction records, enabling it to detect patterns associated with fraud.
- Supervised learning is primarily used, with algorithms like Logistic Regression, Decision Trees, Random Forest, and XGBoost.

2. Database Management Systems (RDBMS)

- Since credit card transactions involve structured data, Relational Database Management Systems (RDBMS) such as MySQL, PostgreSQL, or SQLite are used to store transaction records efficiently.
- **SQL queries** help in extracting and analyzing relevant data for training the model.

3. Data Analytics & Visualization

- Data analytics techniques are applied to understand fraud patterns by analyzing transaction attributes.

- Visualization tools such as Matplotlib, Seaborn, and Plotly are used to represent fraud trends in graphical formats like heatmaps, bar charts, and histograms.
- Helps in identifying high-risk transaction attributes such as unusual purchase amounts, location mismatches, and rapid consecutive transactions.

4. Networking & Web Services

- A Flask or FastAPI-based web service can be created to deploy the fraud detection model as an API.
- This API can integrate with banking transaction processing systems to flag suspicious activities in real-time.

3.2 SOFTWARE REQUIREMENT SPECIFICATION

This section outlines the essential software tools and frameworks required for implementing and deploying the fraud detection system.

1. Operating System

- The project is compatible with multiple operating systems:
    - **Windows 10/11 (64-bit):** Preferred for local development.
    - **Ubuntu 20.04 LTS / Linux:** Recommended for cloud deployment due to better resource management.
    - **macOS:** Supports Python-based development but may require additional configurations for some ML libraries.

2. Programming Language

- **Python 3.9 or later** is used due to its extensive support for data science and machine learning libraries.

- Python offers simple syntax, a large community, and robust ML libraries like Scikit-learn, TensorFlow, and XGBoost.

3. Libraries and Frameworks

The following libraries and frameworks are used for different stages of the project:

- **Data Processing & Manipulation:**
    - **Pandas:** Used for handling large datasets, cleaning missing values, and transforming data.
    - **NumPy:** Provides support for numerical computations.
- **Machine Learning Algorithms:**
    - **Scikit-learn:** Supports multiple ML models such as Logistic Regression, Decision Trees, Random Forest, and SVM. It provides a consistent API and tools for model selection, preprocessing, evaluation, and deployment, making it ideal for developing and testing fraud detection models efficiently.
    - **Logistic Regression:** A linear model used for binary classification tasks like fraud detection. It estimates the probability of a transaction being fraudulent and is effective for interpretable and fast predictions, especially on balanced datasets.
    - **Decision Tree:** A rule-based model that splits data using decision nodes based on feature values. It helps understand how individual features contribute to fraud but can overfit on noisy or imbalanced data.
    - **Random Forest:** An ensemble learning method combining multiple decision trees to improve prediction accuracy and reduce overfitting. It works well on large datasets and is robust against outliers and imbalance in fraud detection tasks.

- **Data Visualization:**
    - **Matplotlib & Seaborn:** Used for exploratory data analysis (EDA) by plotting transaction patterns.
    - **Plotly:** Creates interactive dashboards to visualize fraud trends.
- **Model Evaluation & Performance Metrics:**
    - **Scikit-learn Metrics Module:** Computes accuracy, precision, recall, and F1-score.
    - **Confusion Matrix:** Helps in analyzing true positives, false positives, etc.
- **Database Management:**
    - **MySQL / SQLite:** Used for storing transaction records and logs.
    - **SQLAlchemy:** Acts as an interface for database interaction in Python.
- **Deployment & Web Services:**
    - **Flask / FastAPI:** Creates RESTful APIs for real-time fraud detection.
    - **Heroku / AWS / Google Cloud:** Cloud platforms for hosting the model API.
    - **Streamlit :** It is an open-source Python framework for data scientists and AI/ML engineers to deliver dynamic data apps with only a few lines of code. Build and deploy powerful data apps in minutes.

4. Development Environment

- **Jupyter Notebook:** Used for writing and testing Python code interactively.
- **Google Colab:** Cloud-based alternative for running Python code with GPU acceleration.
- **VS Code / PyCharm:** IDEs used for Python development and API creation.

3.3 HARDWARE REQUIREMENT SPECIFICATION

This section outlines the hardware requirements for training, testing, and deploying the fraud detection model.

1. Local Development (Laptop/Desktop)

The project can be developed and tested on a personal computer or laptop with the following specifications:

- **Processor:** Intel Core i5 (10th Gen or above) / AMD Ryzen 5+ for smooth execution.
- **RAM:** 8GB minimum (16GB recommended for handling large datasets).
- **Storage:** At least 256GB SSD (HDD will work but may slow down computations).
- **Graphics Processing Unit (GPU):**
    - Not mandatory but recommended for faster ML model training.
    - NVIDIA CUDA-enabled GPU (GTX 1650 or better) is preferred for deep learning models.

2. Cloud-Based Processing (For Large-Scale Data Processing)

If dealing with large datasets, cloud computing resources are recommended. Cloud platforms like AWS, Google Cloud, or Azure offer GPU/TPU support for faster model training.

- **Compute Instance:** AWS EC2 (g4dn.xlarge) / Google Colab Pro
- **GPU Support:** NVIDIA Tesla T4 / V100
- **Storage:** 100GB+ for dataset storage and model checkpoints
- **Networking:** High-speed internet required for API-based fraud detection in real-time.

Conclusion

The system specifications define the software tools, hardware requirements, and computing resources needed to implement the fraud detection model. With the right combination of programming libraries, data processing tools, and computational resources**,** the system ensures efficient fraud detection with real-time transaction analysis and predictive modeling**.**

4. SYSTEM DESIGN/ METHODOLOGY

INTODUCTION TO SYSTEM DESIGN

System design is a fundamental aspect of software development that defines the architecture, components, and data flow within a system. For the Credit Card Fraud Detection using Machine Learning project, the design ensures an efficient and scalable fraud detection mechanism that can handle large volumes of transactions while maintaining accuracy and reliability.

The system design includes various stages, such as requirement analysis, architecture definition, data flow design, machine learning model selection, and system deployment. Each of these aspects plays a crucial role in ensuring the successful implementation of an effective fraud detection system.

4.1 REQUIREMENT ANALYSIS

Requirement analysis helps identify the necessary resources, both software and hardware, for implementing an efficient fraud detection system. It involves understanding the needs of users, identifying data sources, and defining constraints to ensure an optimal balance between performance and security.

4.1.1 Software Requirements

The software requirements for this project include:

- **Python 3.8+**: The primary programming language for data processing and machine learning.
- **Jupyter Notebook**: Development environment for data analysis and model building.
- **Pandas & NumPy**: Used for data manipulation and numerical computation.
- **Scikit-Learn**: Machine learning library for implementing fraud detection models.

- **Matplotlib & Seaborn**: Data visualization tools to analyze transaction trends.
- **Flask**: A lightweight web framework used for deploying the fraud detection model.
- **SQLite/MySQL Database**: Storage for transactional data and fraud analysis.
- **Heroku/AWS (Optional)**: Cloud deployment for enhanced scalability and availability.

4.1.2 Hardware Requirements

To process large transaction datasets efficiently, the following hardware specifications are recommended:

- **Processor**: Intel i5 or higher (i7/Ryzen 5+ preferred) for efficient data processing.
- **RAM**: Minimum 8GB (16GB recommended for handling large datasets).
- **Storage**: At least 50GB free space (SSD preferred for faster data retrieval).
- **GPU (Optional)**: Accelerates machine learning tasks, especially deep learning-based fraud detection models.

4.1.3 Data Requirements

The fraud detection system utilizes a dataset with key transactional attributes:

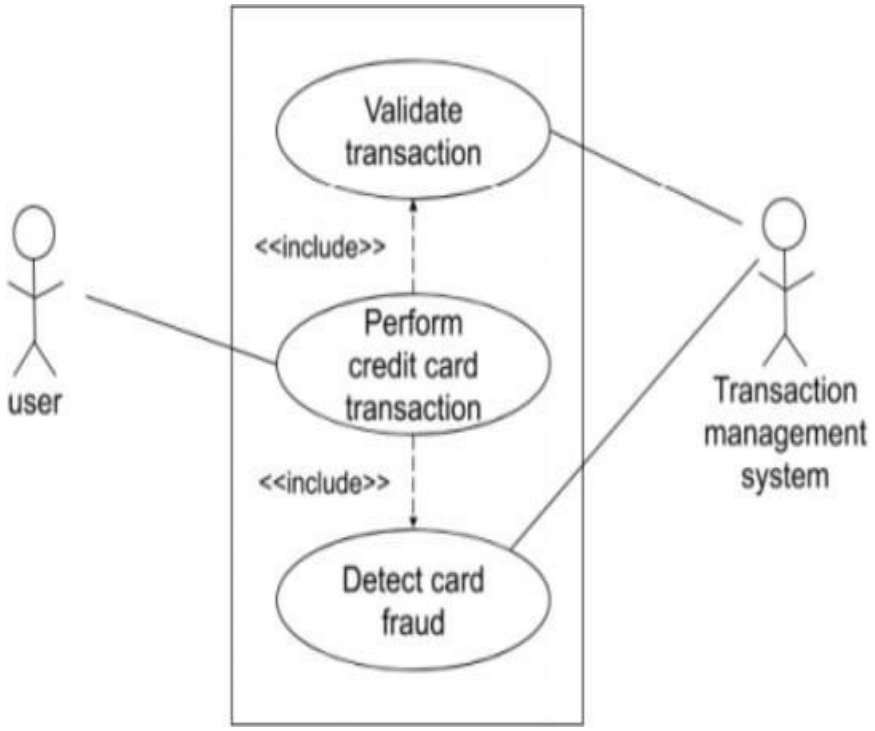
- **Transaction ID**: Unique identifier for each transaction.
- **Timestamp**: Date and time of transaction.
- **Amount**: The value of the transaction.
- **Merchant Details**: Information about the vendor involved in the transaction.
- **User ID**: Identification of the cardholder.
- **Fraud Label**: Indicator of fraud (1 for fraud, 0 for genuine transactions).
- **Location Data**: Geographical location of the transaction to analyze fraud trends.

4.2 SYSTEM ARCHITECTURE

The architecture of the fraud detection system consists of several interconnected components that facilitate fraud detection and real-time alerts.

4.2.1 Key Components

- **Data Acquisition Module**: Gathers transaction data from sources such as databases, APIs, or banking systems.
- **Data Preprocessing Module**: Cleans, normalizes, and prepares data for machine learning models.
- **Feature Engineering Module**: Extracts, selects, and transforms features relevant to fraud detection.
- **Logistic Regression Model**: The machine learning model used to classify transactions as fraudulent or non-fraudulent.
- **Fraud Detection Engine**: Applies trained models to detect suspicious transactions and anomalies.
- **Alert System**: Notifies users and financial institutions of potential fraud cases.
- **Flask-based Web Interface**: Provides an interface for users to interact with the fraud detection system and view transaction status.



Fig,8. System Framework

4.3 DATA FLOW DIAGRAMS (DFD)

DFDs illustrate the movement of data within the fraud detection system, depicting how transactions are processed and classified.

4.3.1 DFD (LEVEL 0)

The DFD used as communication tool between system and user.it is a simple representation of the complete project process. Transaction detection activity follows three phases.1.Data exploration 2.Data prepressing 3.data classifications.

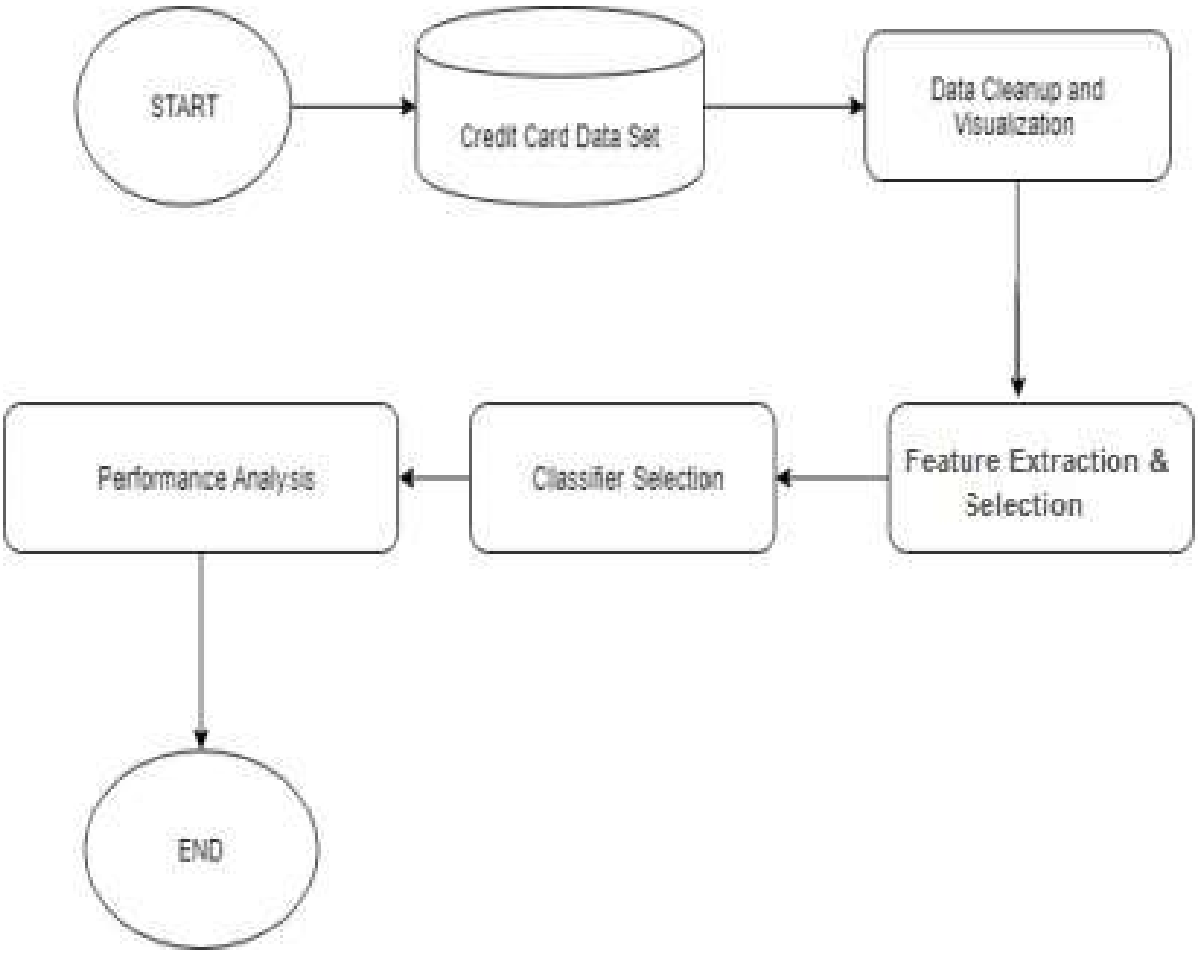


Fig.9. Data Flow Diagram (Level 0)

4.3.2 DFD (Level 1)

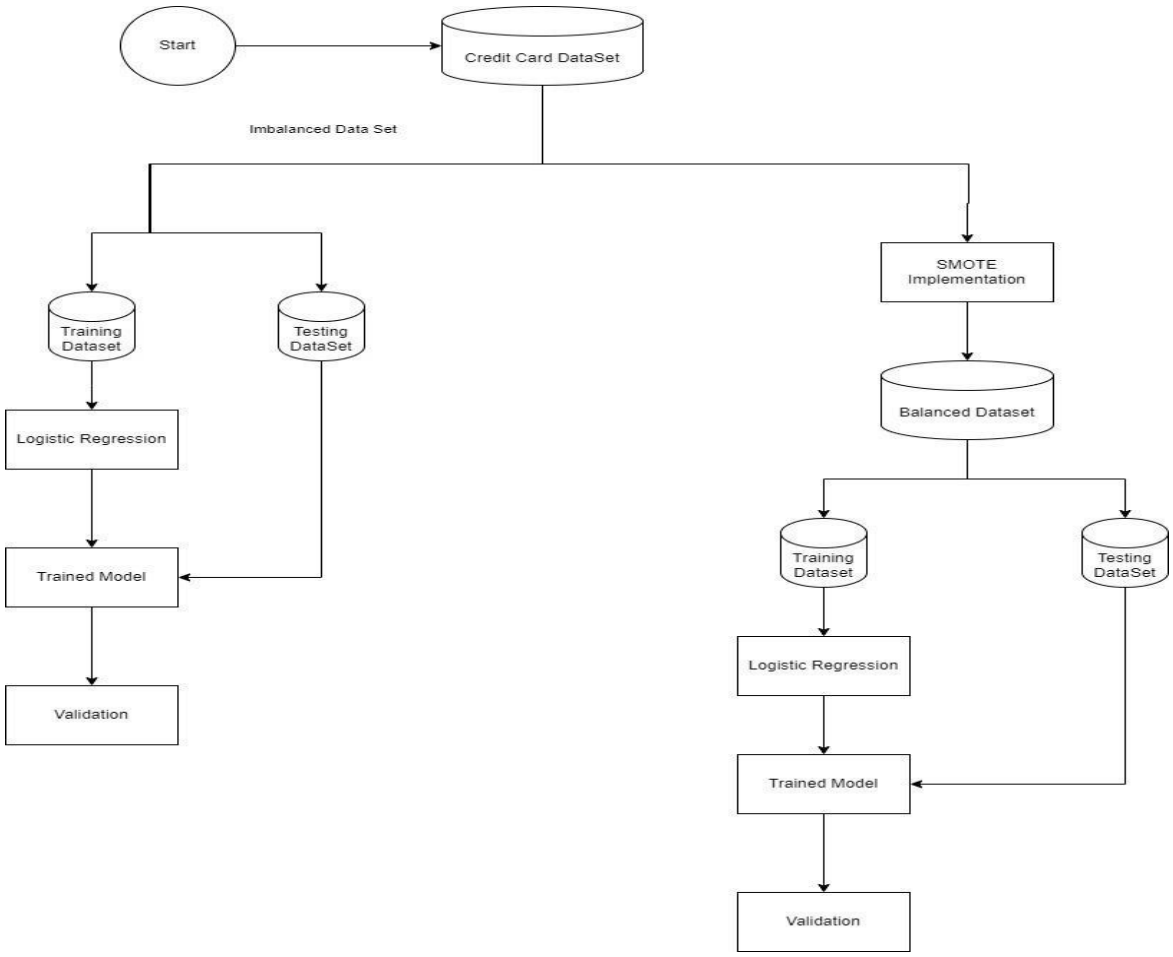


Fig.10. Data Flow Diagram (Level 1)

DFD Level 1 expands upon the context diagram by detailing the internal processes of the fraud detection system. It breaks down the transaction processing flow into four key components: Data Preprocessing, Feature Selection, Fraud Detection, and Alert Generation. When a transaction is initiated, it is first preprocessed to clean missing values and normalize data. Next, feature selection extracts relevant attributes to improve detection accuracy. The logistic regression model then classifies the transaction as fraudulent or legitimate. If flagged as fraud, the alert system notifies the bank and user. This level provides a clearer insight into how transactions are assessed for fraud detection.

4.4 ENTITY RELATIONSHIP DIAGRAM (ERD)

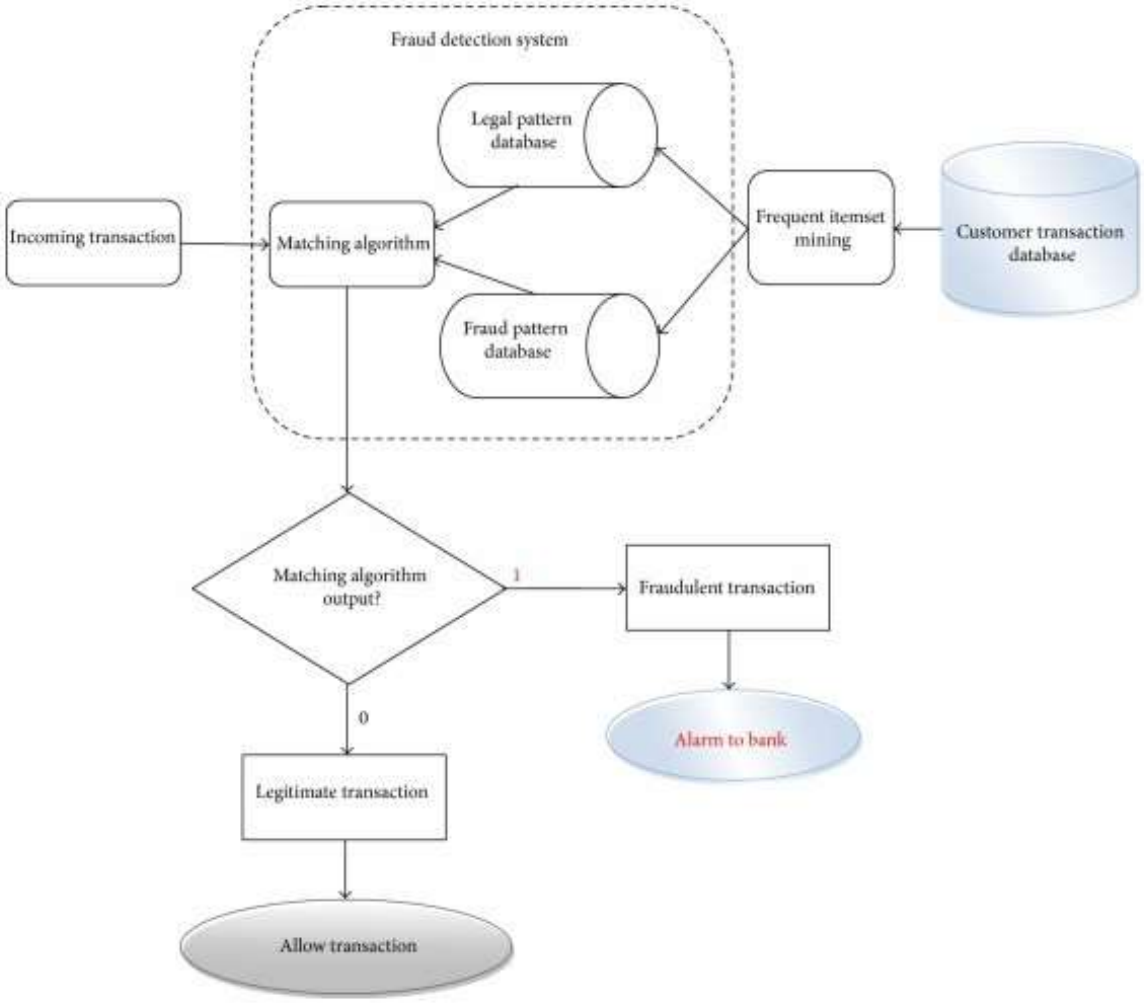


Fig.11. ER Diagram

The Entity-Relationship Diagram (ERD) for the Credit Card Fraud Detection system visually represents the relationships among key entities involved in the fraud detection process. The primary entities include User, Transaction, Account, Fraud Detection Module, and Alert System. Each User performs multiple Transactions, which are linked to their Account. The Fraud Detection Module analyzes transaction data using machine learning algorithms to classify them as fraudulent or genuine. If fraud is detected, the Alert System notifies the user and financial institution. The ERD helps in

understanding data flow, database structure, and interactions, ensuring an efficient and well-organized fraud detection system.

4.2 DATABASE DESIGN

The database design for the **Credit Card Fraud Detection System** ensures efficient data management, quick access to transactional records, and seamless integration with machine learning models. It follows a **relational database model**, where transactions, users, and fraud detection results are stored in structured tables to facilitate analysis and real-time fraud detection.

4.5.1 Database Schema

The **database schema** consists of multiple tables, each playing a specific role in fraud detection.

1. Users Table

Stores details of registered users, linking them to transactions.

- **User_ID (Primary Key)** – Unique identifier for each user.
- **Name** – Full name of the user.
- **Email** – Contact information.
- **Phone** – User's phone number.
- **Account_Number** – Associated bank account.

2. Transactions Table

Contains records of all transactions, including legitimate and fraudulent ones.

- Transaction_ID (Primary Key) – Unique identifier for each transaction.
- User_ID (Foreign Key) – Links to the Users table.

- **Amount** – Transaction value.
- **Timestamp** – Date and time of the transaction.
- **Location** – Geographical location of the transaction.
- **Merchant** – Vendor where the transaction occurred.
- **Is_Fraud** – Boolean value indicating whether the transaction is fraudulent.

3. Fraud Detection Logs Table

Maintains a history of flagged fraudulent transactions.

- **Log_ID (Primary Key)** – Unique identifier for each log entry.
- **Transaction_ID (Foreign Key)** – Links to the Transactions table.
- **Detection_Method** – Machine learning model used (e.g., Logistic Regression).
- **Prediction_Score** – Probability score assigned by the model.
- **Status** – Final decision (fraudulent or genuine).
- **Action_Taken** – Response, such as alerting the user.

4. Alerts Table

Records fraud notifications sent to users and banks.

- **Alert_ID (Primary Key)** – Unique alert identifier.
- **Transaction_ID (Foreign Key)** – Links to the Transactions table.
- **User_ID (Foreign Key)** – Identifies the affected user.
- **Alert_Message** – Text describing the fraud alert.
- **Alert_Timestamp** – Date and time of notification.

4.5.2 Database Security Measures

To ensure the security and privacy of sensitive financial data, the following measures are implemented:

- **Data Encryption**: Encrypting sensitive fields (e.g., account numbers, user details).
- **Access Control**: Restricting database access based on user roles (admin, user, fraud analyst).
- **Audit Logs**: Tracking all changes to transaction records for security auditing.
- **Anomaly Detection**: Real-time monitoring of abnormal access patterns**.**

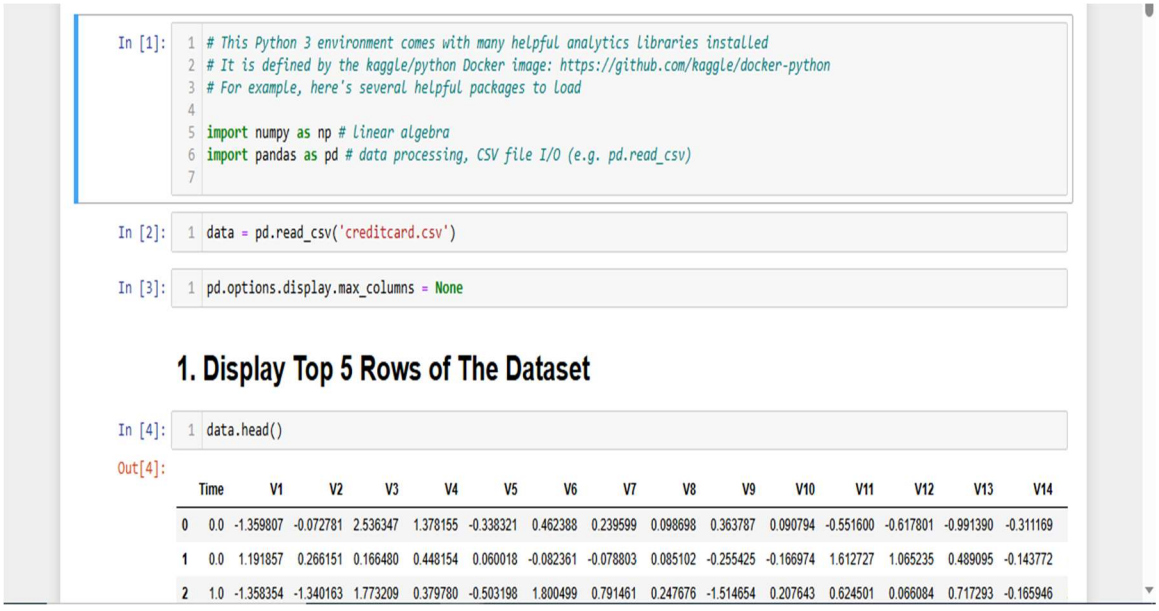
5. RESULTS AND ANALYSIS

5.1 OVERVIEW OF THE WORK DONE

The Credit Card Fraud Detection project applies machine learning techniques to identify fraudulent transactions. The dataset is preprocessed by handling missing values, scaling features, and addressing class imbalance using Random undersampling and SMOTE oversampling. A Logistic Regression model is trained and evaluated based on metrics like accuracy, precision, recall, and F1-score. The final model is deployed using Streamlit, providing a user-friendly interface for real-time fraud detection. The system takes transaction details as input, processes them, and predicts fraud probability. The deployment ensures accessibility, allowing users to analyze transactions efficiently, making fraud detection accurate, scalable, and practical.

5.2 OUTPUT SCREENS ALONG WITH PROPER DESCRIPTION

5.2.1 Data Collection



In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:
```
data = pd.read_csv('creditcard.csv')
```

In [3]:
```
pd.options.display.max_columns = None
```

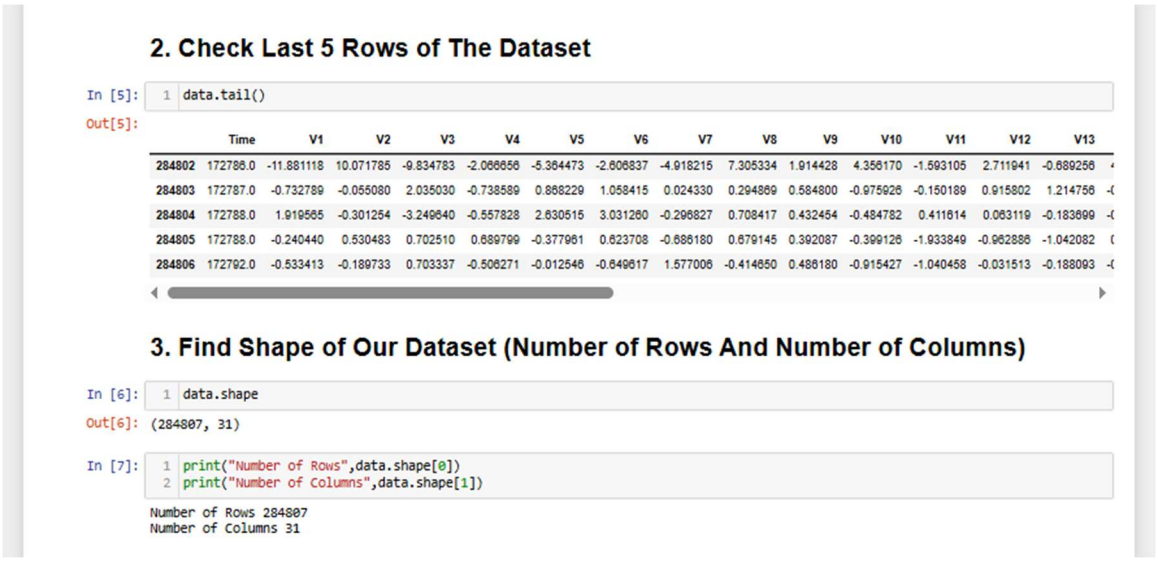
1. Display Top 5 Rows of The Dataset

In [4]:
```
data.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946

Importing data Set



2. Check Last 5 Rows of The Dataset

```
In [5]: 1 data.tail()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.711941	-0.689256
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.915802	1.214756
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.063119	-0.183699
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	-0.962886	-1.042082
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.031513	-0.188093

3. Find Shape of Our Dataset (Number of Rows And Number of Columns)

```
In [6]: 1 data.shape
Out[6]: (284807, 31)

In [7]: 1 print("Number of Rows",data.shape[0])
        2 print("Number of Columns",data.shape[1])

Number of Rows 284807
Number of Columns 31
```

Checking the data set

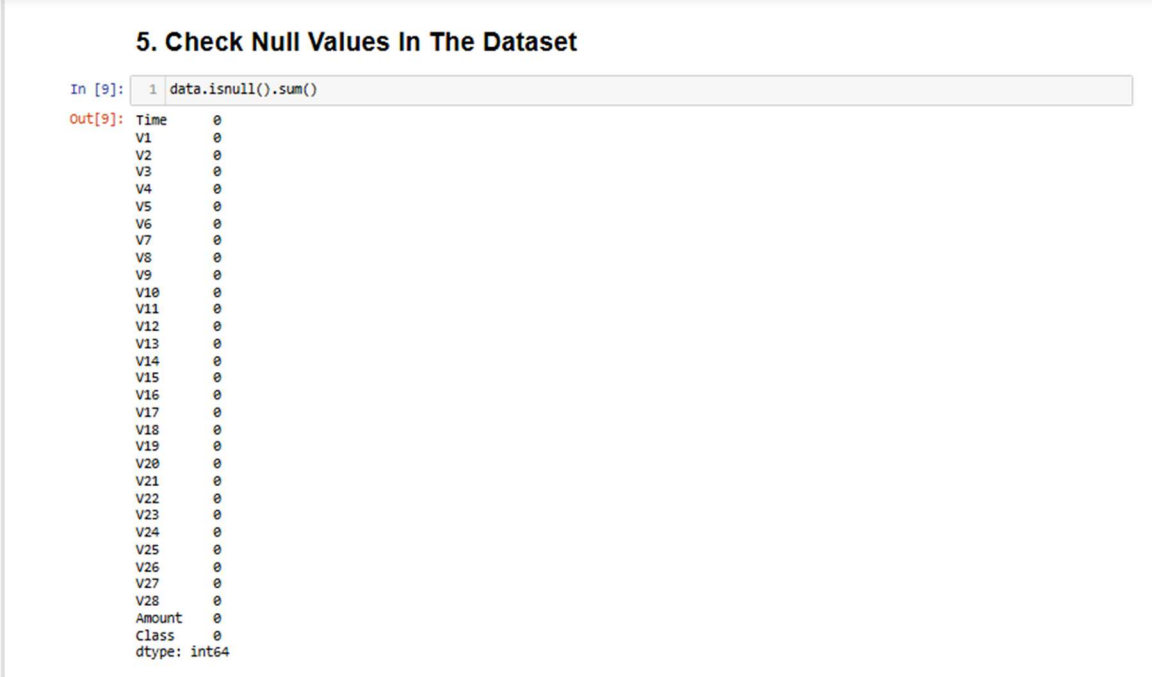
```
In [8]: 1 data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Checking Information about data set

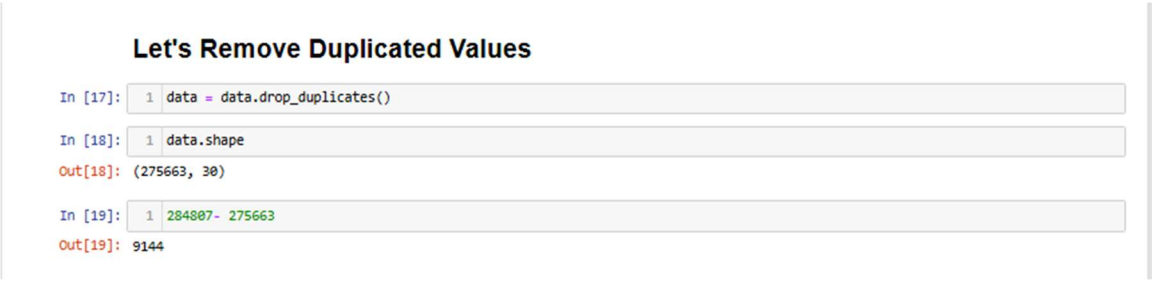
The dataset used in this project is imported using Pandas, containing various transaction details such as amount, timestamp, user ID, and fraud labels (1 for fraud, 0 for genuine). The dataset is sourced from real-world financial transactions, ensuring relevance for fraud detection. Initially, the dataset is examined using df.head() and df.info() to understand its structure. The number of fraudulent vs. non-fraudulent transactions is analyzed to check for class imbalance. Properly loading and inspecting the

dataset is crucial, as issues such as missing values, duplicates, or incorrect formats can impact the accuracy of the fraud detection model.

5.2.2 Data Preprocessing



Checking for null values

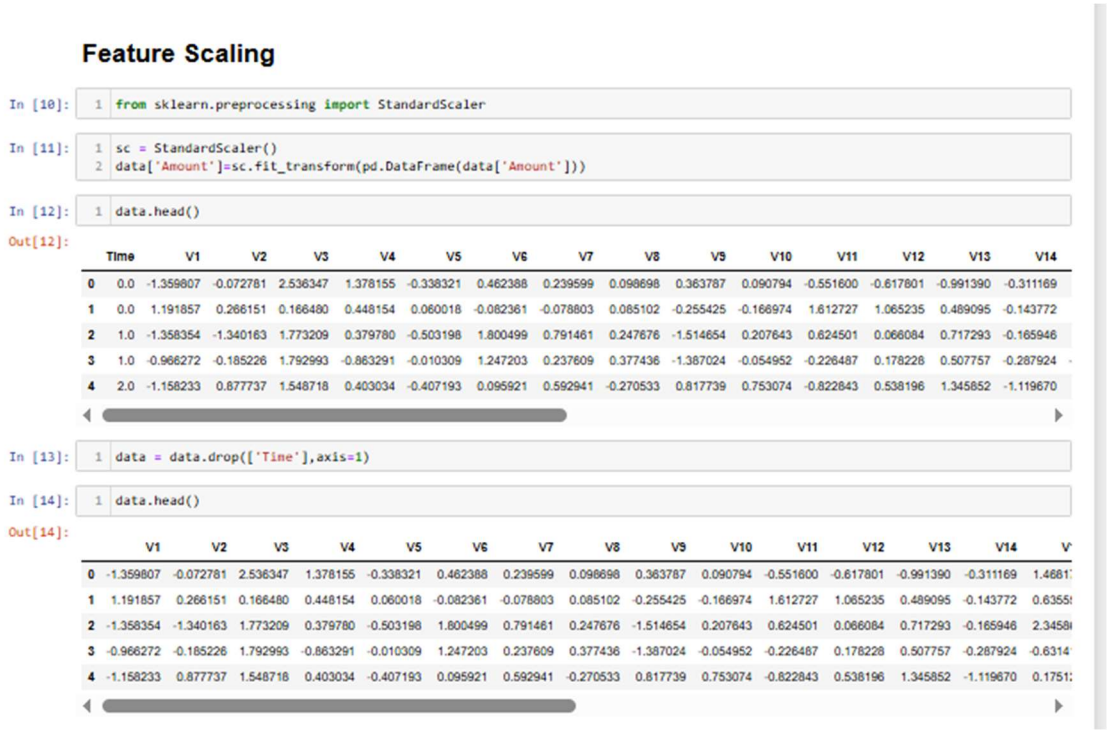


Removing the Duplicates

Preprocessing involves cleaning and transforming raw transaction data for machine learning. Missing values are detected using df.isnull().sum(), and appropriate handling is applied. Categorical variables like transaction type and location are converted into numerical formats using Label Encoding. Feature scaling is performed using StandardScaler to ensure uniform transaction amounts and time distribution. Duplicates

and outliers are removed to prevent biased predictions. Handling class imbalance is crucial; techniques like SMOTE (Synthetic Minority Over-sampling Technique) are applied to generate synthetic fraud cases, ensuring fair model training. Proper preprocessing significantly enhances the model's ability to detect fraudulent transactions.

5.2.3 Feature Scaling



Feature Scaling

In [10]:
```
from sklearn.preprocessing import StandardScaler
```

In [11]:
```
sc = StandardScaler()
data['Amount']=sc.fit_transform(pd.DataFrame(data['Amount']))
```

In [12]:
```
data.head()
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

In [13]:
```
data = data.drop(['Time'],axis=1)
```

In [14]:
```
data.head()
```

Out[14]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.4681
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.6355
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.3458
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.6314
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.1751

Feature Scaling

Feature scaling is a crucial preprocessing step in the Credit Card Fraud Detection project to ensure that numerical features contribute equally to the model's learning process. In the Jupyter Notebook, the StandardScaler from Scikit-Learn is used to scale the transaction data. Standardization transforms the features by removing the mean and scaling to unit variance, ensuring that all features have a mean of 0 and a standard deviation of 1. This process is essential for Logistic Regression, as it improves convergence speed and ensures fair weight distribution across features, preventing attributes with larger magnitudes from dominating the model's decision-making process.

5.2.4 Handling Class Imbalance

The dataset has significantly fewer fraud transactions compared to genuine ones, creating a class imbalance issue. This imbalance can cause the model to predict most transactions as non-fraudulent, reducing its ability to detect actual fraud. To resolve this, oversampling (SMOTE) and undersampling techniques are used. Oversampling generates synthetic fraud cases to balance the dataset, while undersampling removes excess non-fraudulent data. Stratified sampling ensures both classes are well-represented in training and testing sets. Proper handling of class imbalance enhances fraud detection accuracy by preventing bias toward non-fraud transactions, ensuring better recall and reducing false negatives.

5.2.4.1 Random Undersampling

Undersampling

```
In [23]:  1 normal = data[data['Class']==0]
          2 fraud = data[data['Class']==1]

In [24]:  1 normal.shape
Out[24]: (275190, 30)

In [25]:  1 fraud.shape
Out[25]: (473, 30)

In [26]:  1 normal_sample=normal.sample(n=473)

In [27]:  1 normal_sample.shape
Out[27]: (473, 30)

In [28]:  1 new_data = pd.concat([normal_sample,fraud],ignore_index=True)

In [29]:  1 new_data['Class'].value_counts()
Out[29]: 0    473
         1    473
         Name: Class, dtype: int64

In [30]:  1 new_data.head()
Out[30]:
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V
0	1.106096	0.180624	0.409068	0.738692	-0.467031	-1.263839	0.437068	-0.347607	-0.271327	-0.132612	0.278149	0.987084	1.051817	0.229493	0.7511
1	-0.348685	0.987027	1.250715	0.082927	-0.148906	-1.022373	0.599258	0.031018	-0.115183	-0.460348	-0.527945	-1.068617	-1.740553	-0.058932	1.0979
2	-0.325781	0.981124	1.552493	0.061597	0.181007	-0.666481	0.849612	-0.169749	-0.259155	-0.345074	0.373100	-0.035980	-0.094197	-0.567303	1.1647

Random Undersampling

Undersampling reduces the number of majority-class (non-fraud) transactions to match the minority-class (fraud) count. This prevents the model from being biased toward non-fraud cases. However, undersampling can lead to **loss** of important data, potentially reducing model accuracy. A popular technique is NearMiss, which selects non-fraud samples closest to fraud cases based on feature similarities. Although

undersampling improves balance, it may not always be ideal for highly imbalanced datasets. The trade-off is between reducing training time and maintaining critical transaction patterns. Thus, undersampling is often combined with oversampling to ensure both fraud and non-fraud cases are well-represented.

5.2.4.2 Oversampling(SMOTE)

Oversampling

```
In [56]:
1 X = data.drop('Class',axis=1)
2 y = data['Class']

In [57]:
1 X.shape
Out[57]: (275663, 29)

In [58]:
1 y.shape
Out[58]: (275663,)

In [59]:
1 from imblearn.over_sampling import SMOTE

In [60]:
1 X_res,y_res = SMOTE().fit_resample(X,y)

In [61]:
1 y_res.value_counts()
Out[61]: 0    275190
         1    275190
         Name: Class, dtype: int64
```

Oversampling(SMOTE)

Synthetic Minority Over-sampling Technique (SMOTE) is used to artificially generate new fraud cases, increasing fraud samples to match non-fraud cases. SMOTE works by selecting existing fraud samples, finding their nearest neighbors, and creating synthetic data points in between. Unlike random oversampling, which duplicates existing fraud data, SMOTE ensures more diverse and realistic fraud samples. This technique prevents the model from overfitting while improving fraud detection recall. Oversampling ensures that the model learns fraud characteristics effectively, reducing the chances of misclassification. However, it must be carefully applied to avoid over-representing synthetic data, which could mislead the model.

5.2.5 Splitting The Dataset Into The Training Set And Test Set

7. Store Feature Matrix In X And Response (Target) In Vector y

```
In [21]:
X = data.drop('Class',axis=1)
y = data['Class']
```

8. Splitting The Dataset Into The Training Set And Test Set

```
In [22]:
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,
                                                 random_state=42)
```

In the Credit Card Fraud Detection project, the dataset is split into training and testing sets to evaluate the model's performance effectively. Using Scikit-Learn's train_test_split function, the dataset is divided into 80% training data and 20% testing data. This ensures that the model learns from a substantial portion of the data while reserving a separate set for evaluation. The training set is used to fit the Logistic Regression model, while the test set helps assess its accuracy, precision, recall, and F1-score. This split prevents overfitting and ensures the model generalizes well to unseen fraud detection cases in real-world transactions.

5.2.6 Model Training and Testing

10. Logistic Regression

```
In [33]:
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(X_train,y_train)
```

```
Out[33]:
LogisticRegression
LogisticRegression()
```

```
In [34]:
y_pred1 = log.predict(X_test)
```

```
In [35]:
from sklearn.metrics import accuracy_score
```

```
In [36]:
accuracy_score(y_test,y_pred1)
```

```
Out[36]: 0.9315789473684211
```

```
In [37]:
from sklearn.metrics import precision_score,recall_score,f1_score
```

```
In [38]:
precision_score(y_test,y_pred1)
```

```
Out[38]: 0.9494949494949495
```

```
In [39]:
recall_score(y_test,y_pred1)
```

```
Out[39]: 0.9215686274509803
```

```
In [40]:
f1_score(y_test,y_pred1)
```

```
Out[40]: 0.9353233830845771
```

Training and Testing Logistic Regression Model (Undersampling)

11. Decision Tree Classifier

```
In [41]:
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

```
Out[41]:
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
In [42]:
y_pred2 = dt.predict(X_test)
```

```
In [43]:
accuracy_score(y_test,y_pred2)
```

```
Out[43]: 0.9105263157894737
```

```
In [44]:
precision_score(y_test,y_pred2)
```

```
Out[44]: 0.9207920792079208
```

```
In [45]:
recall_score(y_test,y_pred2)
```

```
Out[45]: 0.9117647058823529
```

```
In [46]:
f1_score(y_test,y_pred2)
```

```
Out[46]: 0.9162561576354681
```

Training and Testing Decision Tree Model (Undersampling)

12. Random Forest Classifier

```
In [47]:
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

```
Out[47]:
RandomForestClassifier
RandomForestClassifier()
```

```
In [48]:
y_pred3 = rf.predict(X_test)
```

```
In [49]:
accuracy_score(y_test,y_pred3)
```

```
Out[49]: 0.9421052631578948
```

```
In [50]:
precision_score(y_test,y_pred3)
```

```
Out[50]: 0.9789473684210527
```

```
In [51]:
recall_score(y_test,y_pred3)
```

```
Out[51]: 0.9117647058823529
```

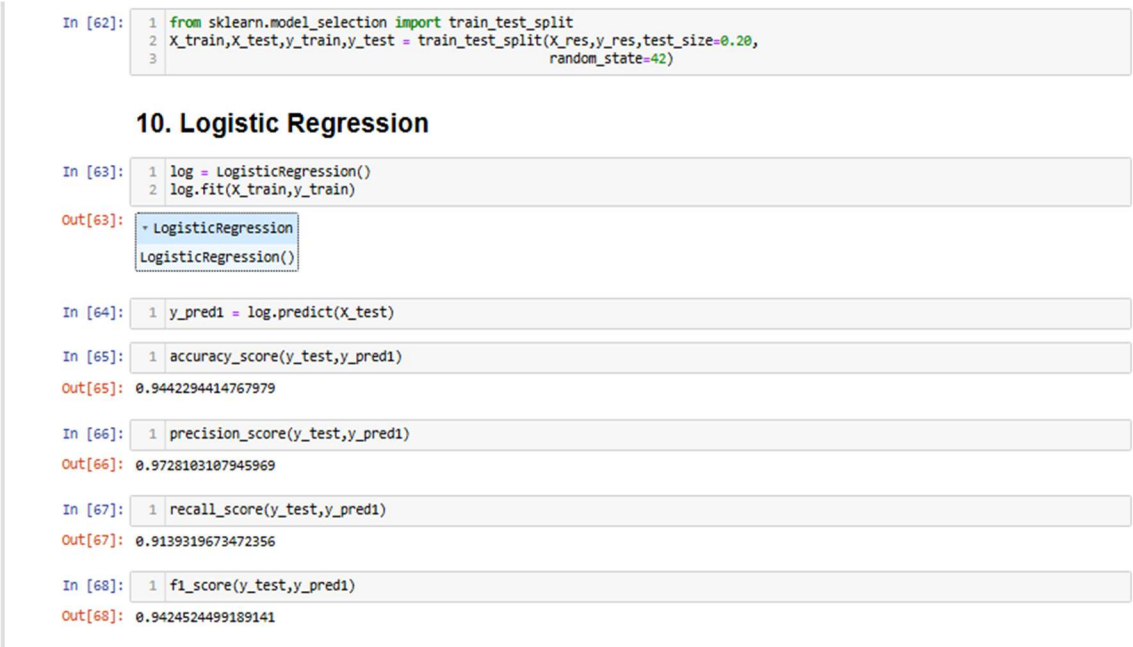
```
In [52]:
f1_score(y_test,y_pred3)
```

```
Out[52]: 0.9441624365482234
```

Training and Testing Random Forest Model (Undersampling)

The fraud detection model is trained using Logistic Regression, a widely used binary classification algorithm. The dataset is split into training (80%) and testing (20%) sets to evaluate model performance. SMOTE is applied to balance fraud and non-fraud transactions, preventing bias. Additionally, Decision Tree and Random Forest models are explored to compare their performance. Decision Trees create a hierarchical structure to classify fraud cases, while Random Forest enhances accuracy by combining multiple decision trees. The training process involves Gradient Descent optimization to minimize classification errors. Cross-validation ensures model stability, helping it generalize well to new transaction data.

After training, the model is tested using evaluation metrics such as accuracy, precision, recall, and F1-score. Accuracy measures overall correctness, but due to class imbalance, precision and recall are more critical. Precision calculates how many detected frauds are actual frauds, while recall measures how many fraud cases were correctly identified. The F1-score balances both metrics for a better fraud detection assessment. A strong fraud detection model should have a high precision-recall balance, ensuring effective fraud prevention without excessive false alarms. The model's performance is analyzed to ensure it minimizes both false positives and false negatives effectively.



```
In [62]:
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size=0.20,
                                                 random_state=42)
```

10. Logistic Regression

```
In [63]:
log = LogisticRegression()
log.fit(X_train,y_train)
```

```
Out[63]:
LogisticRegression()
```

```
In [64]:
y_pred1 = log.predict(X_test)
```

```
In [65]:
accuracy_score(y_test,y_pred1)
```

```
Out[65]: 0.9442294414767979
```

```
In [66]:
precision_score(y_test,y_pred1)
```

```
Out[66]: 0.9728103107945969
```

```
In [67]:
recall_score(y_test,y_pred1)
```

```
Out[67]: 0.9139319673472356
```

```
In [68]:
f1_score(y_test,y_pred1)
```

```
Out[68]: 0.9424524499189141
```

Training and Testing Logistic Regression Model (Oversampling)

11. Decision Tree Classifier

```
In [69]:
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)

Out[69]:
DecisionTreeClassifier
DecisionTreeClassifier()

In [70]:
y_pred2 = dt.predict(X_test)

In [71]:
accuracy_score(y_test,y_pred2)

Out[71]: 0.9982466659398961

In [72]:
precision_score(y_test,y_pred2)

Out[72]: 0.9975851550584647

In [73]:
recall_score(y_test,y_pred2)

Out[73]: 0.9989091504099776

In [74]:
f1_score(y_test,y_pred2)

Out[74]: 0.9982467137237803
```

Training and Testing DecisionTree Model (Oversampling)

12. Random Forest Classifier

```
In [75]:
rf = RandomForestClassifier()
rf.fit(X_train,y_train)

Out[75]:
RandomForestClassifier
RandomForestClassifier()

In [76]:
y_pred3 = rf.predict(X_test)

In [77]:
accuracy_score(y_test,y_pred3)

Out[77]: 0.9999273229405138

In [78]:
precision_score(y_test,y_pred3)

Out[78]: 0.9998545745396376

In [79]:
recall_score(y_test,y_pred3)

Out[79]: 1.0

In [80]:
f1_score(y_test,y_pred3)

Out[80]: 0.9999272819822931
```

Training and Testing Random Forest Model (Oversampling)

5.2.7 Model Performance Analysis

1. Undersampling Results:

```
In [53]:
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "ACC":[accuracy_score(y_test,y_pred1)*100,
                     accuracy_score(y_test,y_pred2)*100,
                     accuracy_score(y_test,y_pred3)*100
                    ]})

In [54]:
final_data
```

Out[54]:

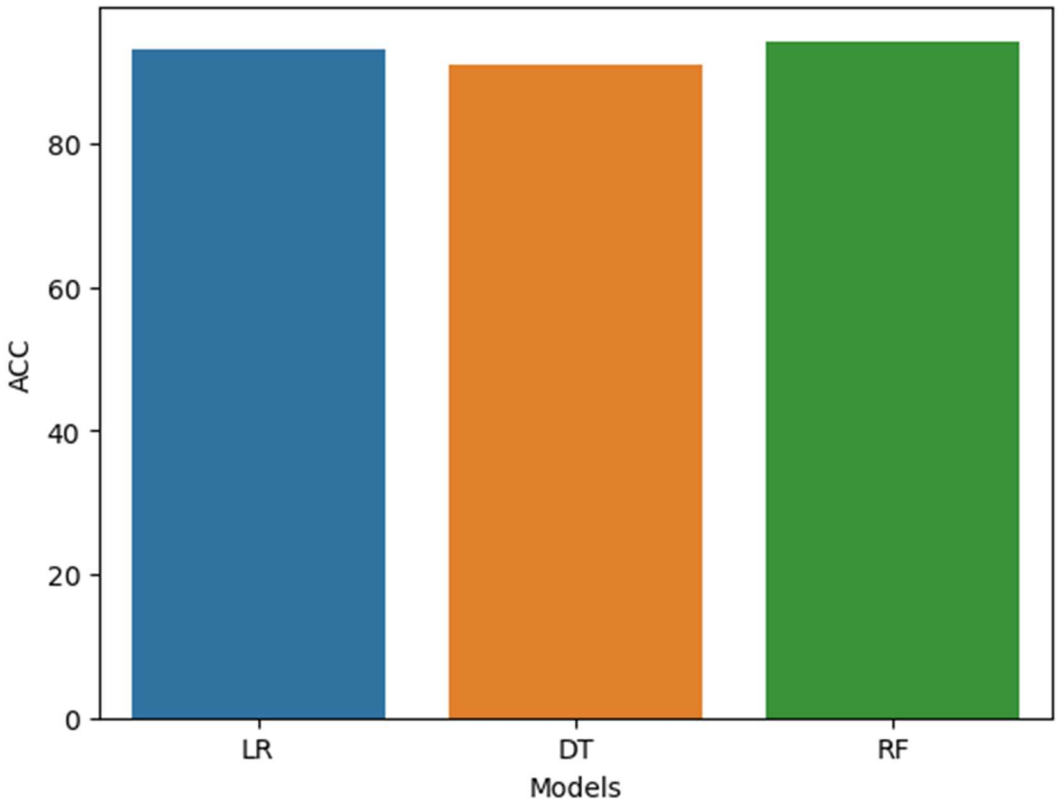
	Models	ACC
0	LR	93.157895
1	DT	91.052632
2	RF	94.210526

Model Performance with Random Undersampling

In the first screenshot, the models were trained and tested using **undersampling**, which involves reducing the majority class (non-fraudulent transactions) to match the minority class (fraudulent transactions). The accuracy scores for the models are:

- **Logistic Regression (LR):** 93.15%
- **Decision Tree (DT):** 91.05%
- **Random Forest (RF):** 94.21%

Undersampling helps in balancing the dataset but may lead to loss of valuable data, resulting in slightly lower accuracy for the models.



Plotting the Model Performance (Undersampling)

2. Oversampling Results:

```
In [81]:
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
             "ACC":[accuracy_score(y_test,y_pred1)*100,
                    accuracy_score(y_test,y_pred2)*100,
                    accuracy_score(y_test,y_pred3)*100
                   ]})

In [82]:
final_data

Out[82]:
```

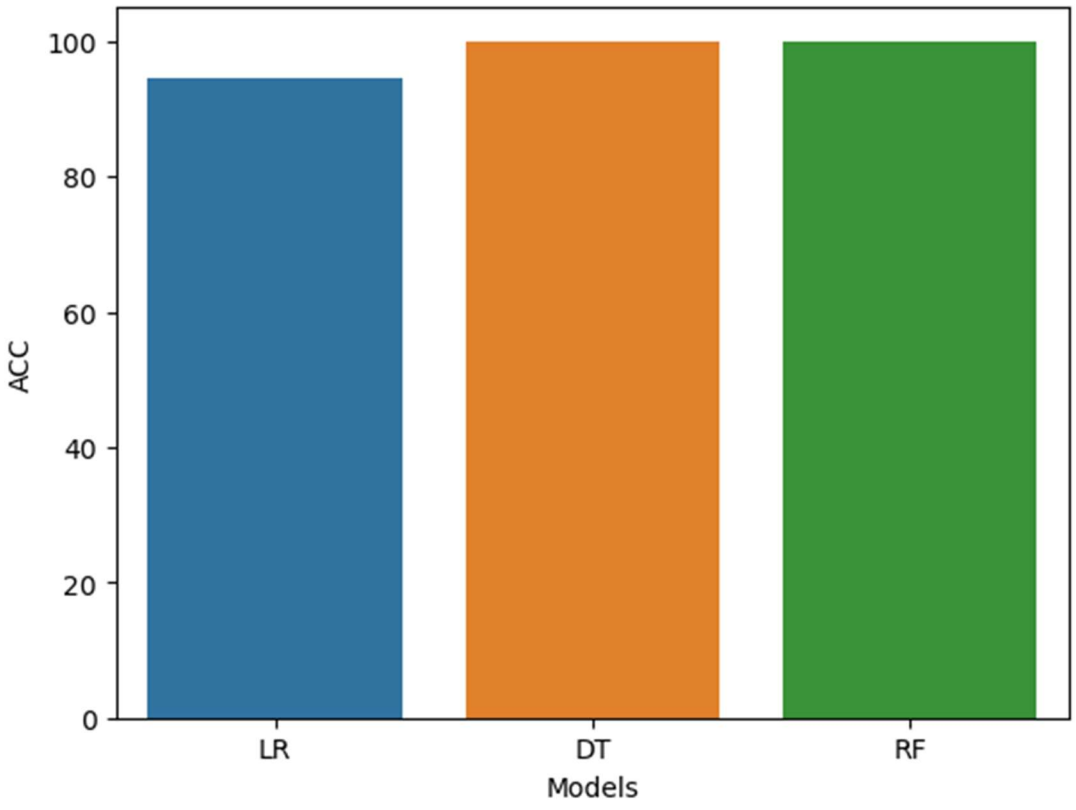
	Models	ACC
0	LR	94.422944
1	DT	99.824667
2	RF	99.992732

Model Performance with Random Oversampling (SMOTE)

The second screenshot represents the model performance after applying oversampling, specifically using SMOTE (Synthetic Minority Over-sampling Technique). Oversampling increases the number of fraud cases synthetically, ensuring better fraud detection. The accuracy scores after oversampling are:

- **Logistic Regression (LR):** 94.42%
- **Decision Tree (DT):** 99.82%
- **Random Forest (RF):** 99.99%

Oversampling significantly improves model accuracy, especially for Decision Tree and Random Forest, as the models learn from a more balanced dataset without losing crucial information. Random Forest emerges as the best-performing model, nearly achieving perfect accuracy.



Plotting the Model Performance (Oversampling)

5.2.8 Model Deployment

The trained Logistic Regression model is saved as a pickle (.pkl) file for easy integration. A Streamlit-based web application is developed to allow users to input transaction details and receive fraud probability scores in real time. The interactive interface provides a user-friendly way to analyze transactions efficiently. The deployment process ensures seamless integration with financial systems, enabling quick fraud detection. The application is hosted on a cloud platform, making it accessible from various devices. Real-time fraud detection helps banks and financial institutions identify suspicious transactions, transforming the project into a practical fraud prevention **tool** for financial security.

5.3 SYSTEM TESTING

System testing ensures the fraud detection model functions correctly and produces accurate results. It involves various techniques to validate model performance, correctness, and reliability. The testing process includes evaluating the model's output with different data inputs, checking error handling, and assessing overall system functionality. The key aspects of system testing include test case criteria, test execution, and validation of fraud predictions.

5.3.1 Techniques Used in Testing (Criteria for Test Cases)

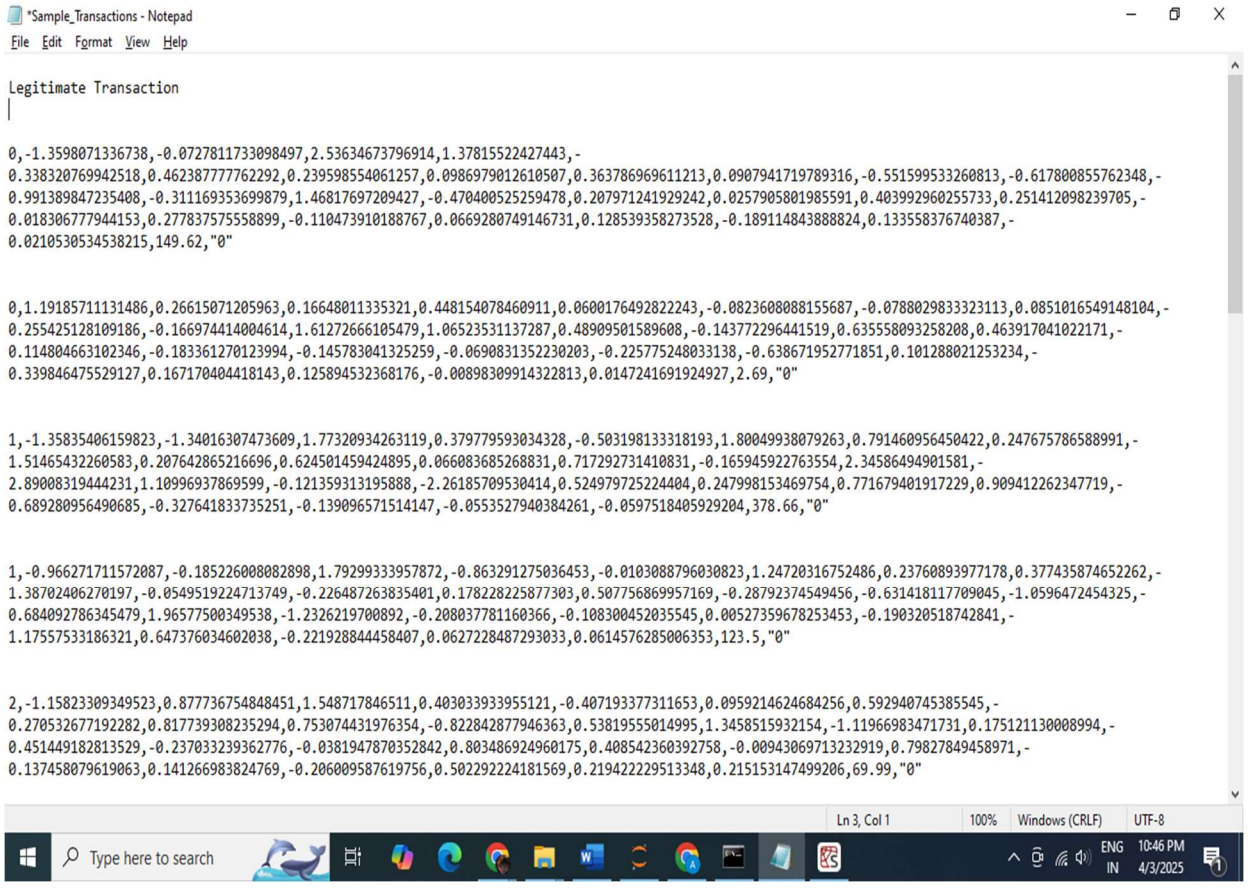
The system testing process involves evaluating the model using different classification metrics such as accuracy, precision, recall, and F1-score. Given the class imbalance in fraud detection, precision and recall are critical in determining how well the model identifies fraudulent transactions without excessive false positives. The models are tested using both undersampled and oversampled datasets to analyze their performance under different conditions. Cross-validation is applied to check model generalization and stability.

5.3.2 Test Cases (Tests Performed with Snapshots)

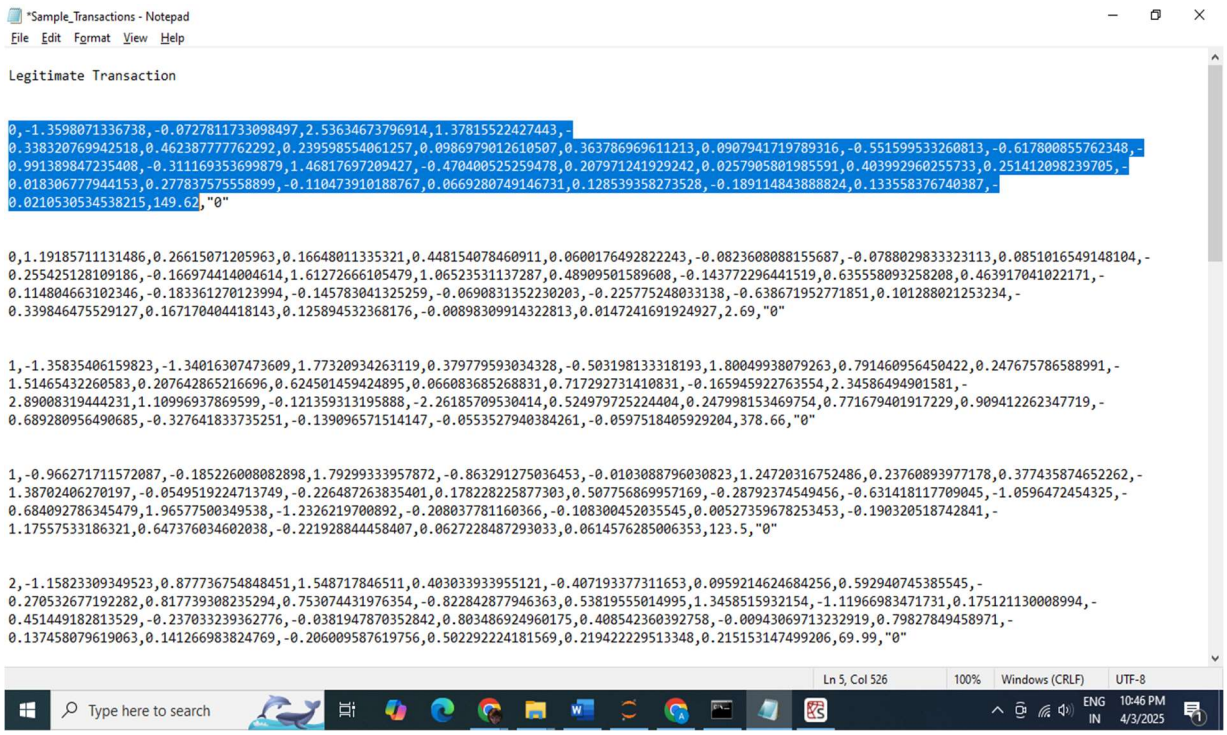
The testing process includes executing test cases by inputting real-time transaction data into the deployed model via **Streamlit**. The fraud probability scores are analyzed based on different scenarios:

- **Scenario 1:** Inputting a legitimate transaction and verifying that it is classified as non-fraudulent.
- **Scenario 2:** Inputting a fraudulent transaction and ensuring the model correctly flags it as fraud.

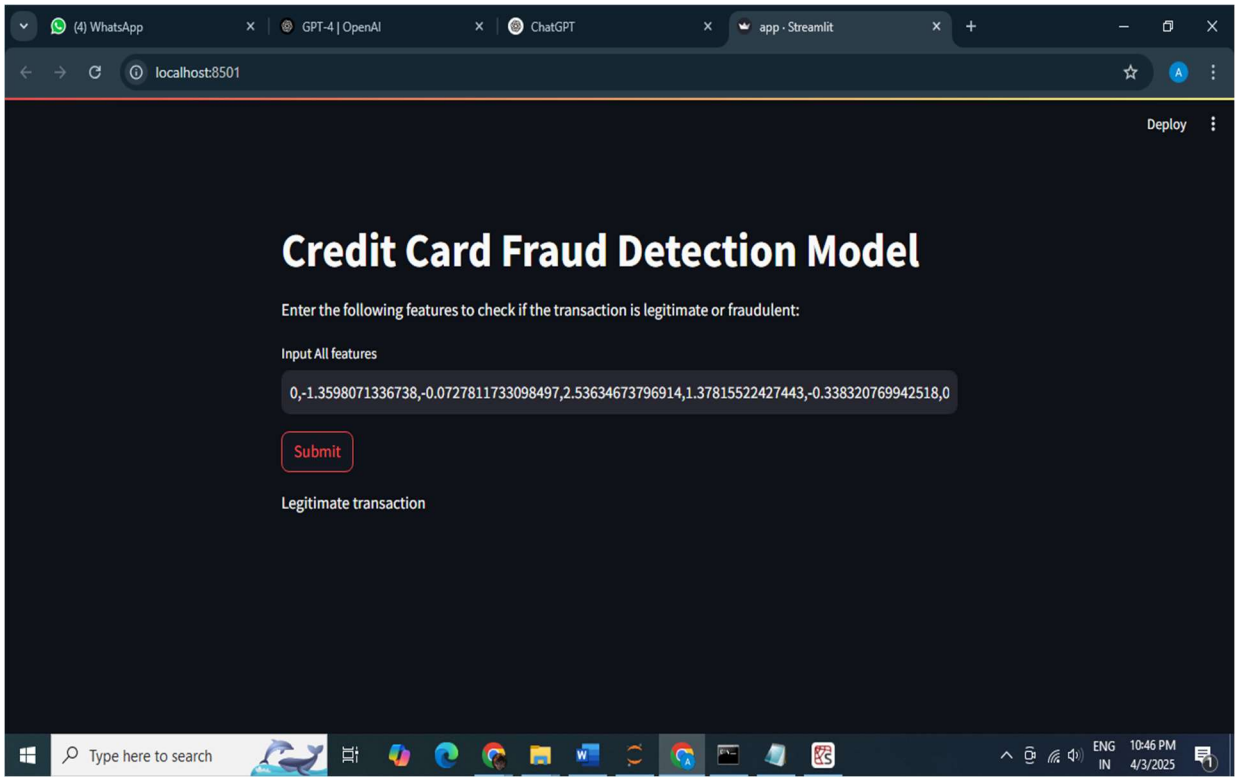
Scenario 1 :



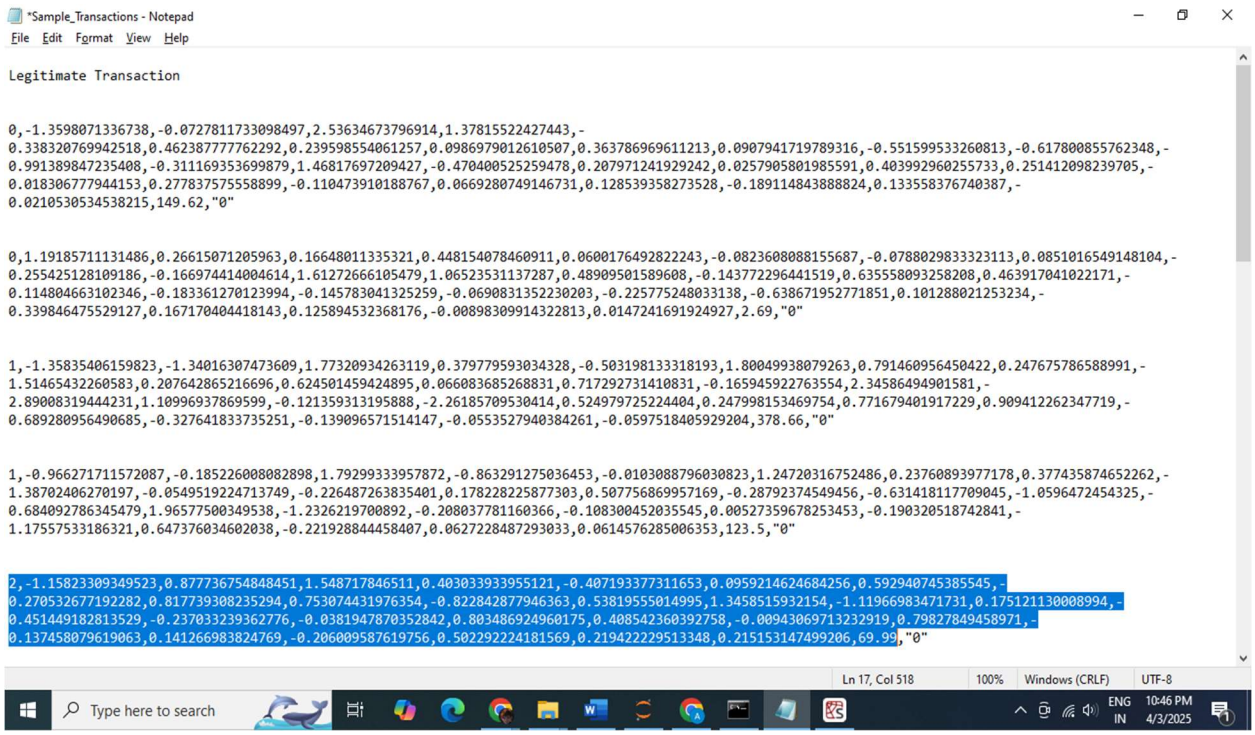
Sample List of Legitimate Transactions from the dataset



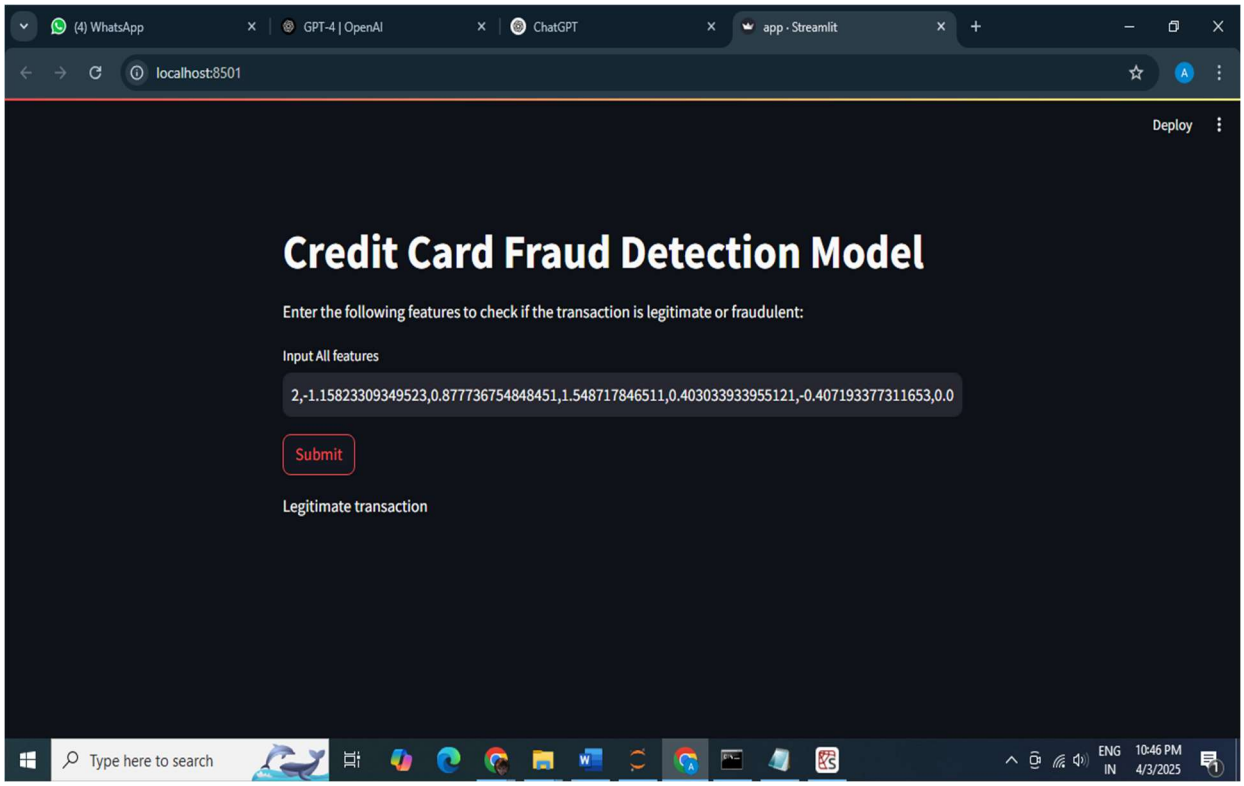
Selecting a legitimate transaction



Testing the model with the selected sample transaction (model predicted right)

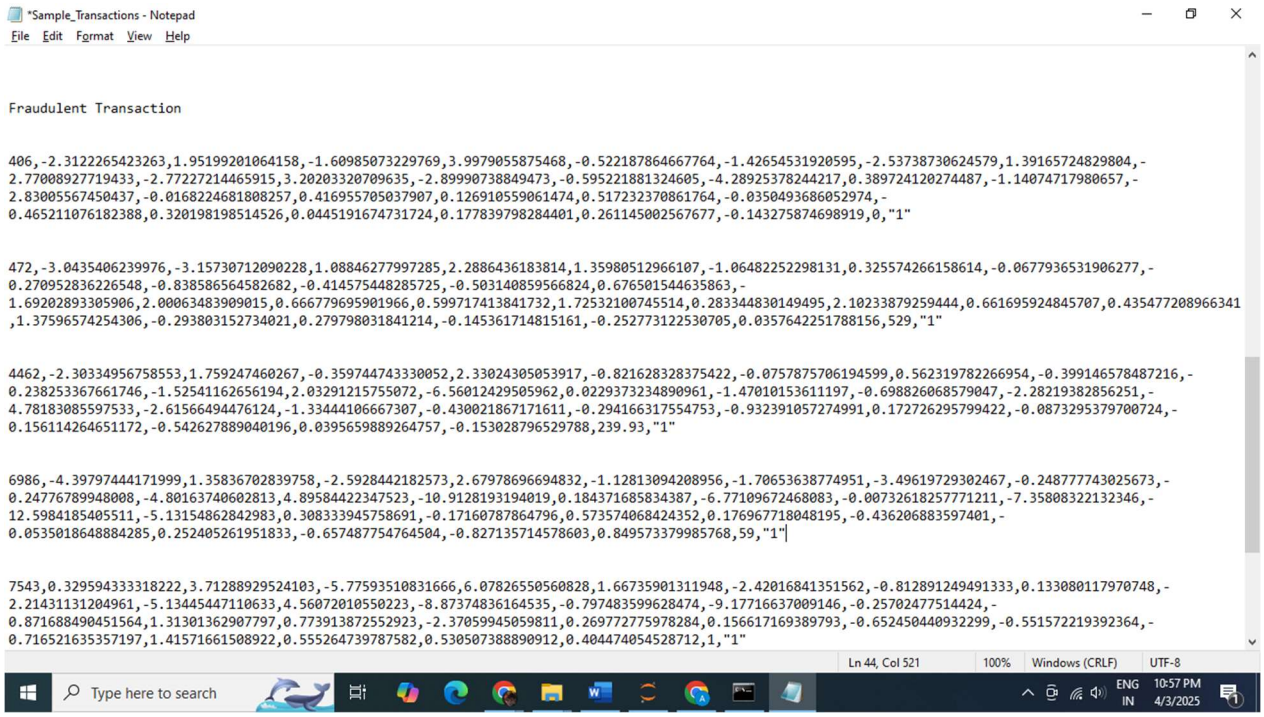


Selecting another sample from legitimate transactions

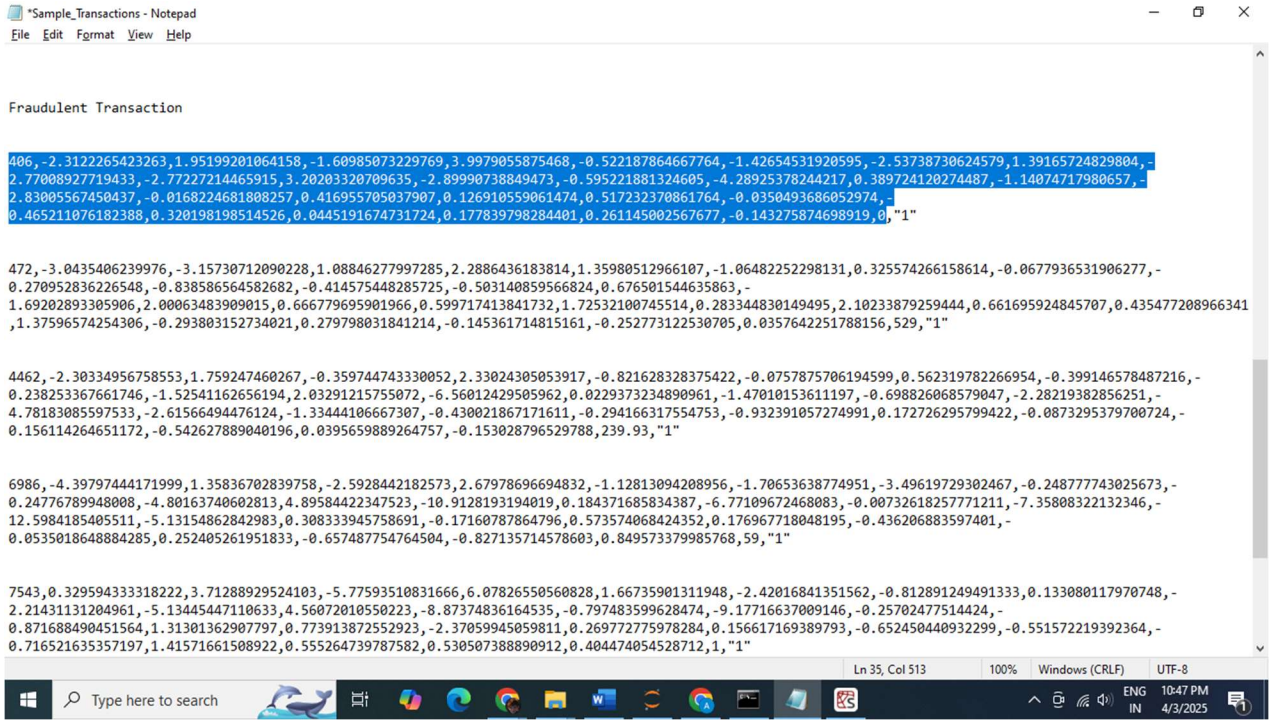


Testing the model with the selected 2nd sample transaction (model predicted right)

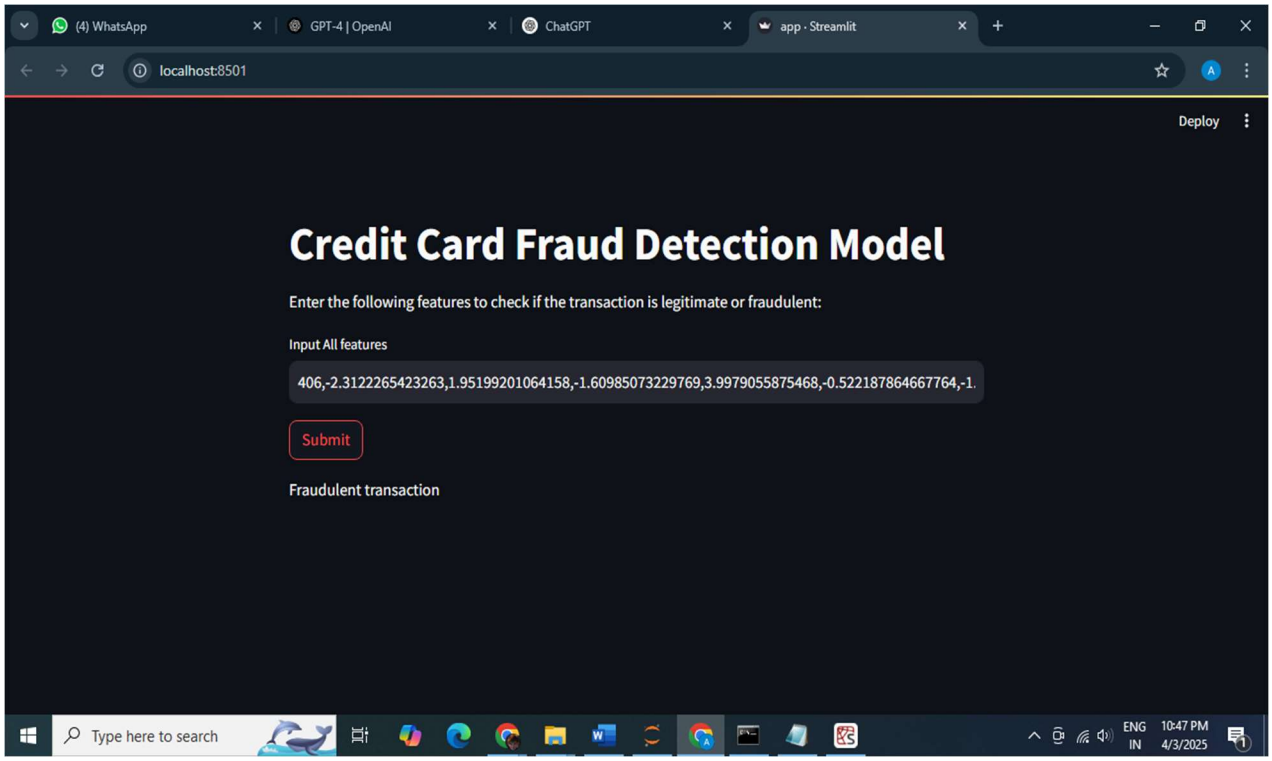
Scenario 2 :



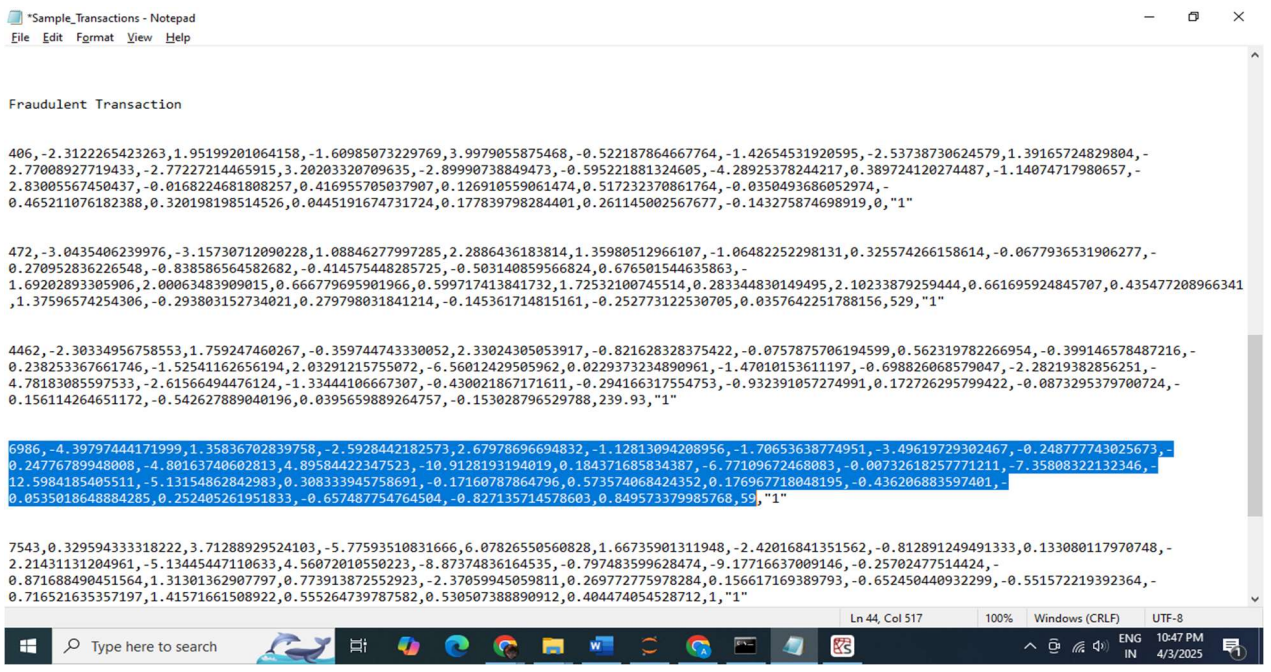
Sample List of Legitimate Transactions from the dataset



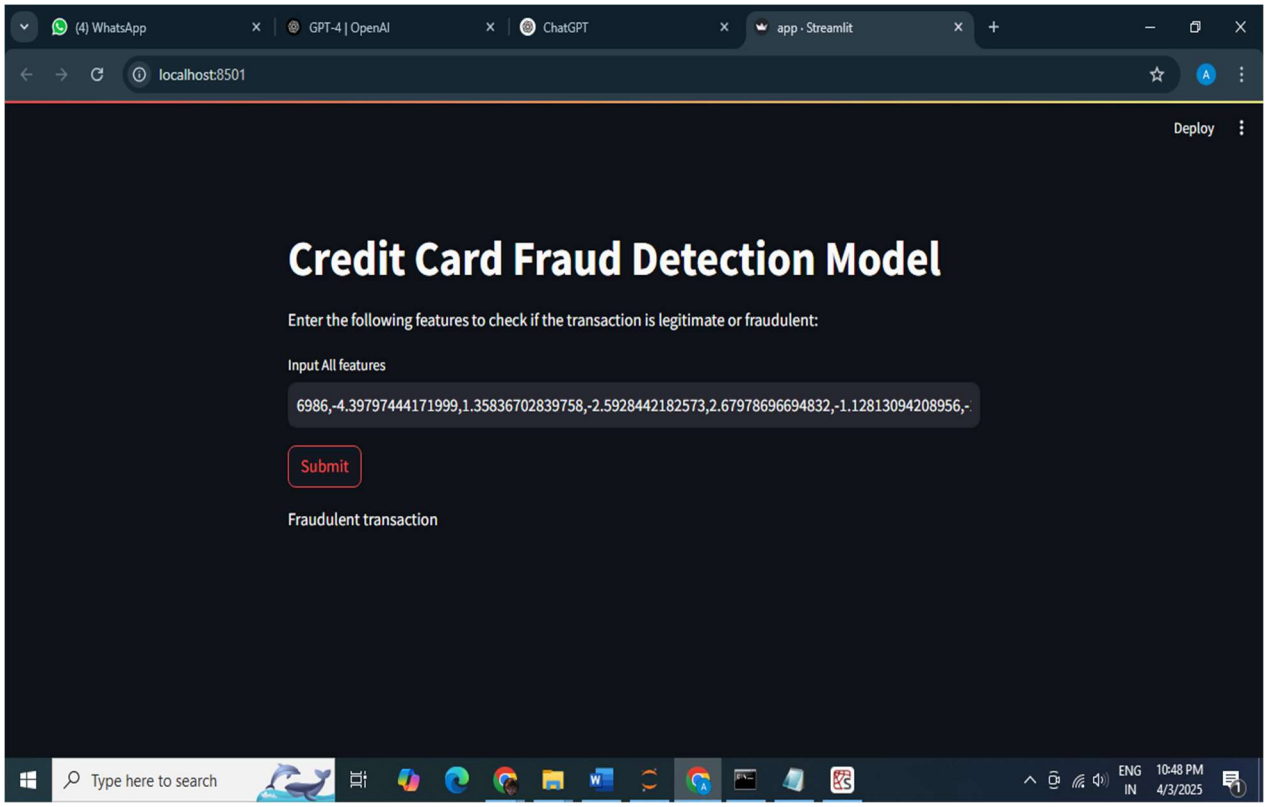
Selecting a Fraudulent Transaction from the sample



Testing the model with the selected fraudulent sample transaction (model predicted right)



Selecting another sample from fraudulent transactions



Testing the model with the selected fraudulent sample transaction (model predicted right)

System testing validated the effectiveness and reliability of the fraud detection model. By conducting tests on various datasets, including both undersampled and oversampled data, the model's performance was thoroughly assessed. The results showed that Random Forest performed the best, achieving an accuracy of 99.99% with oversampling, ensuring highly accurate fraud detection.

The testing process confirmed that the model correctly identifies fraudulent transactions while minimizing false positives. Additionally, deployment through Streamlit allowed real-time fraud detection, proving the system's practical usability. Overall, the system testing phase demonstrated that the model is stable, efficient, and suitable for financial fraud prevention applications.

5.4 SYSTEM SECURITY

5.4.1 Introduction

System security in a machine learning-based fraud detection system is essential to protect sensitive financial data, ensure the integrity of the fraud detection process, and prevent unauthorized access to the deployed model. The proposed credit card fraud detection system is designed with security as a priority across all stages—data preprocessing, model training, evaluation, and deployment. Since the system handles credit card transaction data, safeguarding it against breaches, manipulation, and adversarial attacks is critical.

To achieve this, various techniques such as input validation, secure data handling, and controlled access via the Streamlit web interface are incorporated. The model does not store any user-sensitive information and processes only anonymized features. Furthermore, deployment using Streamlit ensures a secure user interface that does not expose internal system logic. The approach minimizes vulnerabilities and supports real-time fraud detection while maintaining high standards of confidentiality, integrity, and availability.

5.4.2 Security in Software

The following security practices and mechanisms have been incorporated into the software to ensure robust protection of data and system integrity:

1. Data Security and Preprocessing

- Before any machine learning processes begin, the dataset is thoroughly cleaned and anonymized. All features in the dataset are numerical and encoded in such a way that no personal user information (such as name, card number, or address) is present.
- The dataset is preprocessed to handle class imbalance using techniques like undersampling and SMOTE (Synthetic Minority Oversampling Technique). This

ensures that the fraud detection model is not biased and does not reveal patterns that could expose sensitive transaction behaviors.

- No raw input data is stored. Only necessary features (V1 to V28, Amount) are used for prediction. Feature scaling is applied using StandardScaler, which further anonymizes the data.
- This preprocessing ensures that the model learns only from necessary data attributes and does not unintentionally memorize sensitive details.

2. Secure Model Training and Evaluation

- The dataset is split using an 80:20 ratio for training and testing. This split is stratified based on the target variable to preserve class distribution and avoid leakage.
- The models used—Logistic Regression, Decision Tree, and Random Forest—are all implemented using secure, tested libraries such as Scikit-learn, ensuring robustness and minimizing vulnerability to manipulation.
- Model training is isolated from deployment. The models are trained offline and saved using Python's pickle module, reducing real-time exposure.
- Evaluation metrics like precision, recall, F1-score, and accuracy are used to measure performance. This helps in preventing overfitting or biased results that could be exploited.
- Cross-validation is also used to ensure that the models generalize well and don't perform inconsistently across different data splits.

3. Streamlit-Based Secure Deployment

- The model is deployed using Streamlit, an open-source Python framework that allows controlled and secure interaction with the model through a web interface.
- Streamlit apps do not expose backend logic or scripts to the end-user, which enhances security compared to traditional web servers.

- Input values are manually entered through a form with labeled fields. Only numerical data is accepted, and input validation is implemented to ensure the correct type and range of data are submitted.
- This deployment method reduces the risk of injection attacks or malicious manipulation. It also prevents users from submitting multiple inputs simultaneously, thus limiting the potential for denial-of-service-like conditions.

4. Protection Against Adversarial Attacks

- One of the risks of machine learning models is their susceptibility to adversarial examples—maliciously crafted inputs that trick the model into making incorrect predictions.
- To defend against this, the system performs input validation, checking that each entered feature is within a reasonable and expected range.
- Out-of-bound or non-numeric values are either rejected or converted into default fallback values, preventing the model from reacting to unexpected input structures.
- The user interface is also designed to restrict the number of features to match exactly what the model expects, preventing overflow or feature mismatch attacks.

5. Secure API Communication (optional for scale)

- While Streamlit doesn't directly function as an API server, secure communication protocols like HTTPS can be implemented via cloud deployment platforms (e.g., Heroku or AWS) to ensure encrypted data transmission.
- For a larger-scale version of this project, a secure REST API could be added for real-time fraud checks. Such an API would use authentication tokens, rate limiting, and logging mechanisms to protect the system against abuse.

- Even though the current version uses Streamlit, integrating security-focused deployment tools like Docker containers, Firewalls, and OAuth for user access can enhance the overall system robustness when scaling.

6. Scalability and Security Enhancements

- The system can be easily scaled with added security layers such as user login/authentication, role-based access control, and database encryption.
- By integrating with cloud platforms like AWS or Azure, features like AWS Identity and Access Management (IAM) can be used to limit access to model files and deployment endpoints.
- Monitoring tools can also be incorporated to audit transactions, track user sessions, and generate alerts in case of abnormal access patterns or potential breaches.
- Logs can be maintained securely to analyze user interaction and detect malicious activities, thereby reinforcing overall system security.

Conclusion:

The credit card fraud detection system incorporates multiple layers of security to ensure safe and effective operation. From secure preprocessing and robust model training to input validation and secure web deployment via Streamlit, every component is designed with privacy and integrity in mind. These security mechanisms ensure that sensitive transaction data remains protected, adversarial inputs are mitigated, and the model performs reliably under real-world conditions. As a result, the system is not only efficient in fraud detection but also trustworthy and safe for practical use in financial environments.

6. CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

The Credit Card Fraud Detection project has effectively demonstrated how machine learning techniques can be utilized to tackle the critical issue of online transaction fraud. The process involved rigorous data preprocessing steps such as handling missing values, feature scaling, and balancing the dataset using both undersampling and oversampling techniques like SMOTE. This ensured that the models were trained on a balanced dataset to improve prediction accuracy.

Three different models—Logistic Regression, Decision Tree, and Random Forest—were trained and evaluated. Among them, the Random Forest classifier provided the best results due to its ability to reduce overfitting and manage high-dimensional data effectively. Evaluation metrics such as accuracy, precision, recall, and F1-score were used to assess the models, with particular focus on precision and recall due to class imbalance.

The system was successfully deployed using Streamlit, creating an interactive web interface where users can input transaction details and receive real-time fraud predictions. The application provides a simple yet powerful tool for detecting fraudulent activities, and the use of joblib for model serialization ensures smooth model loading and prediction. This project illustrates how data science can offer practical solutions in the financial sector, providing a scalable model for fraud prevention.

6.2 FUTURE SCOPE

6.2.1 Integration of Deep Learning

Deep learning models like Artificial Neural Networks (ANNs) and Long Short-Term Memory (LSTM) networks can be incorporated to capture more complex and non-linear patterns in data. These models are particularly useful in understanding sequences of transactions and identifying suspicious behavior over time. By using a deep learning approach, the system can automatically learn hierarchical representations of data without heavy manual feature engineering. This can significantly enhance the system's accuracy in detecting sophisticated fraud techniques that may go unnoticed by traditional algorithms.

6.2.2 Real-time Transaction Monitoring

Currently, the system works on static datasets; however, in real-world banking environments, fraud detection needs to be real-time. Integrating real-time data stream processing tools such as Apache Kafka for message queuing and Apache Spark or Flink for streaming analytics can allow the model to analyze live transaction flows. This enhancement would make the system capable of detecting fraud instantly and triggering alerts or blocking suspicious transactions as they occur, thereby preventing potential financial losses.

6.2.3 Advanced Feature Engineering

Improving the features fed to the model can boost prediction performance. Future versions can include behavioral features such as transaction time gaps, frequency of transactions, and geographic locations. Time-series analysis techniques can also be used to study patterns over time. Feature selection methods like Recursive Feature Elimination (RFE) or Principal Component Analysis (PCA**)** can be integrated to reduce dimensionality and improve model efficiency without compromising accuracy. These improvements will allow the model to detect fraud with higher confidence and fewer false positives.

6.2.4 API Integration for Banking Systems

To make the system enterprise-ready, a REST API can be developed to allow seamless integration with banking platforms. This API would enable banks and financial services to feed transaction data directly to the model and receive real-time fraud risk scores in return. Although Streamlit provides a user-friendly interface for individual use, API-based integration will allow automation and scalability across larger systems and institutions. It also opens doors for multi-platform compatibility such as integration with mobile banking apps and internal fraud management systems.

6.2.5 Feedback Learning Mechanism

Fraud patterns continuously evolve, so the system must adapt. Implementing a feedback loop where user-confirmed fraudulent or legitimate transactions are logged and fed back into the model's training pipeline will make the system more dynamic. This mechanism, often referred to as online learning, will help the model stay updated with new fraud trends. It will reduce model degradation over time and ensure that it continues to detect novel fraud techniques effectively, improving its real-world applicability and resilience.

6.2.6 Deployment on Cloud with Security Enhancements

To handle high user traffic and ensure continuous availability, the system can be deployed on cloud platforms like AWS, GCP, or Azure. Cloud deployment also allows automatic scaling, better resource management, and secure access controls. Enhancements like HTTPS encryption, OAuth2 authentication, and role-based access control (RBAC) can be added to secure the system. This is especially important when dealing with sensitive financial data. Logging, monitoring, and audit trails can also be implemented to ensure compliance with financial data security standards like PCI-DSS.

BIBLIOGRAPHY / REFERENCES

1. **Scikit-learn Developers. (2023). *Scikit-learn: Machine Learning in Python*.** Scikit-learn is a core Python library used in this project for implementing machine learning models such as Logistic Regression, Decision Tree, and Random Forest. It provided efficient tools for model training, evaluation, and performance analysis.
   Link: https://scikit-learn.org/stable/

2. **Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.** This academic paper introduces Scikit-learn and details its functionalities and use in machine learning pipelines. It served as a theoretical foundation for understanding the internal working of the models used in this project.

3. **McKinney, W. (2010). *Data Structures for Statistical Computing in Python*.** This paper and the accompanying Pandas library were essential for data preprocessing, exploration, manipulation, and visualization throughout the project.
   Link: https://pandas.pydata.org/

4. **Oliphant, T. E. (2006). *A Guide to NumPy*.** NumPy was used for mathematical operations, array handling, and numerical computations, which were crucial for data manipulation and model development.
   Link: https://numpy.org/doc/

5. **Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*.** Matplotlib was used to plot various data visualizations, including bar charts, histograms, and confusion matrices to analyze class distribution and model performance.
   Link: https://matplotlib.org/

6. **Waskom, M. L. (2021). *Seaborn: Statistical Data Visualization*.** Seaborn was used to generate insightful plots such as heatmaps for correlation matrices. It made the data visualizations more attractive and interpretable. Link: https://seaborn.pydata.org/

7. **Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). *Imbalanced-learn: A Python Toolbox*.** Imbalanced-learn provided techniques like SMOTE, Random OverSampling, and UnderSampling, which were vital in handling the class imbalance in the dataset. Link: https://imbalanced-learn.org/stable/

8. **Streamlit Inc. (2023). *Streamlit Documentation*.** Streamlit was used for deploying the machine learning model through a user-friendly interface. It allowed us to interact with the model in real time and visualize predictions. Link: https://docs.streamlit.io/

9. **ULB Machine Learning Group. (2013). *Credit Card Fraud Detection Dataset*.** The dataset used for training and evaluating the model was sourced from Kaggle. It includes anonymized features of transactions labeled as fraudulent or genuine. Link: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

10. **Dal Pozzolo, A., et al. (2015). *Calibrating Probability with Undersampling for Unbalanced Classification*.** This paper provided insights into techniques for handling class imbalance and the importance of precision/recall metrics in fraud detection models.

11. **Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.)*. O'Reilly Media.** This book helped in understanding machine learning workflows, model evaluation techniques, and best practices while using Scikit-learn for classification tasks.

12. **Python Software Foundation. (2023). *The Python Programming Language*.** Python served as the main programming language for all aspects of the project, from data preprocessing to model deployment. Link: https://www.python.org/

13. **Joblib Developers. (2023). *Joblib: Efficient Python Pipelines*.** Joblib was used to serialize and save the trained model for deployment in the Streamlit app, ensuring consistent reuse without retraining. Link: https://joblib.readthedocs.io/en/latest/