A PROJECT REPORT

on

“CREDIT CARD FRAUD DETECTION SYSTEM”

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR’S DEGREE IN
COMPUTER SCIENCE AND ENGINEERING

BY

Harshit Shrivastava	22052386
Rohan Pahari	22052397
Anusha Sathia	22053577

UNDER THE GUIDANCE OF

Dr. Prachet Bhuyan

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024
April 2025

KIIT Deemed to be University
School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"CREDIT CARD FRAUD DETECTION SYSTEM"

submitted by

Harshit Shrivastava	22052386
Rohan Pahari	22052397
Anusha Sathia	22053577

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2024-2025, under our guidance.

Date: 10/04/2025

Dr. Prachet Bhuyan
(Project Guide)

ACKNOWLEDGEMENTS

We would like to express our heartfelt gratitude to everyone who supported and contributed to the successful completion of our project, **"Credit Card Fraud Detection System."**

Our deepest appreciation goes to **Dr. Prachet Bhuyan** for his invaluable mentorship, constant encouragement, and insightful guidance throughout the project. His expertise and constructive feedback were instrumental in navigating the challenges we faced and in shaping the direction of our work.

We are equally thankful to the **faculty and staff of the School of Computer Engineering, KIIT Deemed to be University**, for providing the academic environment, resources, and infrastructure that made this project possible. We extend special thanks to our professors for their continuous motivation and helpful suggestions that enriched the development process.

We are sincerely grateful to our **families and friends** for their unwavering support, patience, and belief in us throughout this journey. Their encouragement kept us going even during the most demanding phases of the project.

We also acknowledge the role of the **open-source community** and developers whose tools, libraries, and frameworks significantly contributed to the functionality and performance of our application.

Lastly, we thank each **team member** for their dedication, collaborative spirit, and tireless efforts. The synergy and shared commitment within our group played a crucial role in bringing this project to life.

This accomplishment stands as a testament to the combined efforts and support of everyone mentioned above. We are truly grateful.

Harshit Shrivastava
Rohan Pahari
Anusha Sathia

ABSTRACT

Credit Card Fraud Detection System is a web-based application designed to help detect and prevent fraudulent credit card transactions using machine learning. By analyzing transaction patterns and behaviors, the system provides real-time insights into potentially suspicious activity.

The application consists of two main components: a Python-based backend that processes data and performs predictions using a trained machine learning model, and a frontend built with Streamlit that delivers a clean, interactive user interface. The backend utilizes a pre-trained classifier and scaler to detect anomalies in transactions and assess the likelihood of fraud.

The frontend is built using Streamlit, providing a simple and interactive interface. Users can upload transaction datasets and view prediction results, highlighting potential fraud cases. The design focuses on clarity and ease of use, with minimal but effective functionality for interacting with the machine learning model.

Additional features like user authentication, real-time inference, and customizable thresholds enhance the functionality and user experience. The interface is simple, responsive, and built with modern design principles, making it easy to use across devices.

This project aims to provide a practical and accessible solution for financial institutions, analysts, and developers to better understand and combat credit card fraud. It’s a smart application of AI to a real-world problem with significant financial and security implications.

Keywords: fraud detection, credit card security, financial safety, machine learning.

Contents

List of Figures

Chapter 1

Introduction

Credit Card Fraud Detection System (CCFDS) is a modern web-based application designed to detect and prevent fraudulent credit card transactions using machine learning. The project aims to provide users—such as analysts, developers, or financial institutions—with real-time insights into potentially suspicious activity, helping reduce financial risk and improve transaction security.

At its core, CCFDS is about identifying unusual patterns in credit card usage. Instead of manually scanning through data or relying solely on basic rule-based systems, users can upload transaction datasets and instantly receive predictions about which transactions may be fraudulent. The system leverages statistical features and behavioral patterns to flag anomalies effectively.

A trained machine learning model processes the input data and predicts the likelihood of fraud for each transaction. These insights are displayed through an intuitive Streamlit interface, allowing users to interact with the results and better understand how the model makes its decisions.

Powered entirely by Python and built using Streamlit, the system combines solid backend logic with a clean and focused user experience. The design emphasizes simplicity—making it easy to upload data, view predictions, and interpret results without needing deep technical knowledge.

In today's digital world, where online transactions are growing rapidly, CCFDS provides a practical tool for enhancing security and building trust. Whether it's used in a learning environment, internal risk analysis, or prototype development, the system delivers fast, explainable insights into transaction legitimacy.

This report outlines how the system was developed, the machine learning principles behind it, and potential future enhancements for expanding its impact in fraud prevention.

Chapter 2

Literature Review

Credit card fraud is a growing concern in the digital age, where millions of transactions occur every second across online and offline platforms. As digital payments increase, so does the risk of fraudulent activity, posing serious threats to both consumers and financial institutions[1]. Traditional fraud detection methods, such as rule-based systems or manual reviews, often fall short—they can be rigid, slow to adapt, and produce high false positives or miss sophisticated fraud patterns altogether[2].

This project addresses those limitations by applying machine learning to detect fraud more accurately and efficiently using standard transaction data. At the heart of the system is a classification model trained on real-world credit card transaction datasets, where legitimate and fraudulent patterns are learned from features such as transaction amount, time, and anonymized behavioral signals[3]. Instead of relying on preset thresholds, the model—trained using algorithms like Logistic Regression—learns complex patterns and improves with more data[4].

The frontend, built with Streamlit, allows users to upload data, view predictions, and explore fraud cases without needing advanced technical expertise. It simplifies interaction with the machine learning backend, making fraud detection tools more accessible and usable in real-time scenarios.

What sets this project apart is its focus on accessibility and adaptability. It's lightweight, doesn't require any specialized software, and can be easily modified for different datasets or environments—be it for banks, educational demos, or small businesses wanting to assess fraud risk[5]. By translating research and machine learning advancements into a practical application, this project brings scalable, data-driven fraud prevention closer to everyday use.

Ultimately, it's a step toward making fraud detection smarter, faster, and more responsive to evolving financial threats[6].

Chapter 3
Problem Statement

Problem Statement:

In today's digital economy, the volume of online transactions is growing rapidly—bringing convenience but also increasing the risk of credit card fraud. Fraudulent transactions can lead to massive financial losses, erode customer trust, and damage the reputation of financial institutions. The challenge lies in detecting these fraudulent activities quickly and accurately, especially as fraudsters become more sophisticated in their methods.

Traditional fraud detection systems often rely on predefined rules or manual reviews, which can be slow, inflexible, and prone to false positives or missed threats. These systems struggle to keep up with evolving fraud patterns and typically require constant updates to stay effective.

This project addresses that gap by offering a machine learning–based solution that analyzes transaction data and detects suspicious behavior in real time. Hosted on a web-based platform built with Streamlit, the system provides an accessible, lightweight interface for users to upload datasets and get instant fraud predictions. It's a step toward building smarter, faster, and more adaptive tools for credit card fraud prevention in the modern digital landscape.

Requirement Specifications
Functional Requirements:

1. **User Authentication**
    - Users can create a new account using their email and password.
    - Login functionality is available for returning users.
2. **Dataset Upload**
    - Credit card transaction dataset has to be uploaded in CSV format.
    - The system validates the uploaded file and checks for required fields before processing.
3. **Fraud Prediction**

- The backend processes uploaded transaction data using a trained **Logistic Regression** model.
- Each transaction is classified as **fraudulent** or **legitimate**.
- Returns predictions in real time for immediate display.

4. **Prediction Output & Evaluation Display**
   - After processing, the system displays prediction results along with **accuracy**.
   - These performance metrics are presented clearly to help users assess how well the Logistic Regression model is identifying fraudulent transactions.
   - This provides a straightforward understanding of the model's effectiveness.
5. **Real-Time Feedback**
   - The system highlights suspicious transactions immediately after file upload.
   - Includes a summary showing total transactions, number of frauds detected, and model confidence levels.
6. **Model Management**
   - Admins can update the machine learning model and scaler through a secure backend interface or API.
   - Version tracking and basic validation are in place for uploaded models.
7. **Settings & Customization**
   - Users can switch themes (e.g., light or dark mode).
   - Allows adjustment of notification frequency and test preferences.

Non-Functional Requirements:

1. **Scalability**
   - The system should handle increased user load without affecting performance.
   - It should be extendable to support future features or modules.
2. **Responsive Design**
   - The interface must adapt seamlessly across all supported devices.
   - Ensures a consistent and user-friendly experience on all devices.
3. **Data Security**
   - Transaction data is handled securely, ensuring protection during both processing and storage.
   - Secure practices are followed for any user login or data upload features, minimizing risks of unauthorized access.

4. **Fast Performance**
    - The system provides fraud predictions with minimal delay after data upload.
    - Streamlit app loads quickly and responds efficiently to user interactions.
5. **Maintainability**
    - Uses modular and well-documented code for easier maintenance.
    - Allows new developers to quickly understand and work with the system.
6. **User-Friendly Interface**
    - Prioritizes ease of use with intuitive navigation and layout.
    - Provides tooltips and guidance throughout the user experience.
7. **Reliability**
    - Designed to function reliably without unexpected failures.
    - Ensures that all user data and results are saved and can be accessed without error.

3.1 Project Planning

Building the Credit Card Fraud Detection System followed a structured yet flexible approach inspired by Agile methodology. This allowed for iterative development, continuous improvement, and regular feedback incorporation throughout the lifecycle of the project. From identifying the growing need for smarter fraud detection tools to building, testing, and deploying a working solution, every phase was guided by usability, accuracy, and real-world applicability. The system is currently deployed locally using Streamlit for testing and demonstration, with future plans for cloud deployment and real-time fraud monitoring.

Requirement Analysis:

We began by exploring the rising threat of credit card fraud in digital transactions. Traditional systems often fail to adapt quickly to new fraud techniques, prompting the need for a data-driven, intelligent solution. The goal was to design a system that could accept transactional data, apply a machine learning model—specifically Logistic Regression—and identify potentially fraudulent activity in real time. We outlined the system’s key functions: data upload, fraud prediction, performance evaluation, and an easy-to-use interface.

System Design:

With the requirements in place, we moved to the system architecture. We planned how the components would interact—data ingestion, model prediction, and user interface. The project was kept intentionally minimal to reduce complexity. Streamlit was chosen as the framework for its simplicity, speed, and suitability for deploying data science applications. The machine learning pipeline was modularly designed for easy retraining or model replacement. Visual mockups of the Streamlit app helped guide implementation with a focus on usability.

Implementation:

The implementation phase brought all the planning into reality. The machine learning model was trained using the credit card fraud detection dataset, focusing on accuracy, precision, and F1 score. Logistic Regression was selected for its interpretability and performance. The model was then integrated into a Python backend, and a simple yet functional Streamlit interface was created for users to upload data and receive predictions. Evaluation metrics were displayed directly to help users understand model performance.

Testing & Debugging:

Each component was tested individually and then integrated into the full pipeline. Rigorous testing ensured that the system correctly handled various types of CSV files, responded appropriately to missing data, and consistently returned accurate predictions. Bugs in file handling, prediction errors, and formatting were resolved during this phase. User feedback was incorporated to simplify the upload process and make the metric outputs more understandable.

Deployment & Maintenance:

The system is currently deployed locally using Streamlit for testing purposes:

- **Streamlit interface** is accessible at: http://localhost:8501

This local deployment enables quick iteration and convenient testing. The codebase is modular and documented, allowing for easy maintenance, future model upgrades, and potential deployment to cloud platforms for broader access.

3.2 Project Analysis

The **Credit Card Fraud Detection System** was built with a clear objective: to provide an effective, accessible, and accurate way of detecting fraudulent transactions using machine learning. In today's digital economy, where millions of online payments happen every minute, preventing fraud in real time is both a necessity and a challenge. This project explores the practical demand for fraud detection, evaluates its feasibility, identifies potential risks, outlines the data flow, and defines the system's target users.

Understanding the Need

Financial fraud is one of the most pressing concerns in online transactions. Users and companies alike are vulnerable to credit card fraud, which often goes undetected until after significant losses. Traditional fraud detection methods either rely heavily on manual audits or static rule-based systems, which are not sufficient against rapidly evolving fraud techniques.

This system aims to fill that gap by:

- Allowing users to upload transaction datasets.
- Automatically identifying potentially fraudulent transactions using machine learning.
- Displaying results with accuracy metrics to measure model reliability.
- Providing a lightweight, easy-to-use tool built on familiar data science tools and frameworks.

No specialized hardware or deep technical expertise is required to use the system.

Feasibility of the System

Technical Feasibility:

The project uses proven, reliable tools:

- **Streamlit** for the user interface.
- **Python** and **scikit-learn** for backend logic and machine learning.
- Logistic Regression as the core predictive model due to its balance of simplicity, speed, and effectiveness for binary classification tasks like fraud detection.

Operational Feasibility:

The system is intuitive and user-friendly. Users simply upload a CSV dataset of transactions, and the system processes the data and returns predictions instantly, along with evaluation metrics. Even those with limited technical knowledge can use the platform efficiently.

Economic Feasibility:
Running locally means there are no immediate infrastructure costs. If deployed on a cloud platform like Streamlit Cloud or AWS in the future, the hosting cost remains low, especially for smaller datasets or single-user access.

Risk Analysis

Like any data-driven system, this project comes with potential risks:
Data Privacy: Transaction data may contain sensitive information. Ensuring proper anonymization and secure handling is critical.
Prediction Accuracy: While Logistic Regression performs well, it may require retraining or replacement to stay effective as fraud patterns evolve.
Overfitting or Underfitting: If not handled properly, the model may give inaccurate results. Cross-validation and performance metrics help mitigate this.
User Trust and Engagement: Users may hesitate to trust automated systems. Clear display of model metrics helps build confidence in its accuracy.

How the System Works (Data Flow)

The core data flow of the system is as follows:

1. The user logs in securely.
2. They upload a CSV file containing credit card transaction data.
3. The data is cleaned, preprocessed, and passed to the trained Logistic Regression model.
4. Predictions are generated to identify fraudulent transactions.
5. Performance metrics such as accuracy, precision, recall, and F1 score are displayed to evaluate model effectiveness.

Target Users

This system is designed for:

- **Banks and FinTech professionals** who want a quick prototype for fraud detection without committing to large-scale infrastructure.
- **Data science students** working on fraud detection projects or ML use cases.

- **Researchers** looking to evaluate model performance on real-world datasets.
- **Educators and trainers** using it as a teaching tool for classification problems in ML.

In the future, the system can be scaled for real-time fraud detection or integrated into larger financial security platforms.

3.3 System Design

The system design uses a modular approach with Streamlit for the frontend and Python with scikit-learn on the backend. Users can upload data and instantly see prediction results, while the backend handles all processing and model logic. It's simple, efficient, and easy to extend.

3.3.1 Design Constraints

While developing the Credit Card Fraud Detection System, several design constraints influenced how the system was structured and how it performs in real-world conditions.

Platform Constraints

The system is designed to run entirely in a web browser using **Streamlit**, so it must work efficiently without requiring any external installations or complex configurations.

Technology Constraints

We used **Python**, **scikit-learn**, and **Streamlit**, which limits the frontend customizability compared to full-stack frameworks. Communication is kept simple, with all logic handled internally.

Hardware & Network Constraints

To ensure accessibility, the system is lightweight and optimized to run smoothly even on systems with limited hardware resources and average internet speeds.

Security & Privacy Constraints

Though minimal user data is stored, the system ensures secure handling of sensitive financial input, with careful management of session interactions and local processing.

Usability Constraints

The goal was simplicity—users can upload CSV files and get predictions instantly. This meant keeping the UI minimal and straightforward, with no advanced interactions or navigation layers.

Notification Constraints

Since this is a browser-based app, real-time alerts like notifications are limited. We can't send push notifications if the tab is closed or the app is in the background, so all feedback is shown within the active session.

Scalability Constraints

Currently deployed locally for demonstration, the system is structured for easy migration to cloud platforms and can be extended to support live monitoring or batch processing in the future.

3.3.2 System Architecture (UML)

The Credit Card Fraud Detection System follows a simple yet effective client-server architecture, combining a minimal frontend with a robust backend that handles all machine learning logic. This structure ensures efficient data flow and real-time predictions.

System Components:

1. Frontend (Streamlit)

- Provides a clean interface for uploading credit card transaction data.
- Displays prediction results (fraud or not fraud) and accuracy of the model.
- Runs locally in the browser with no external dependencies.

2. Backend (Python with scikit-learn)

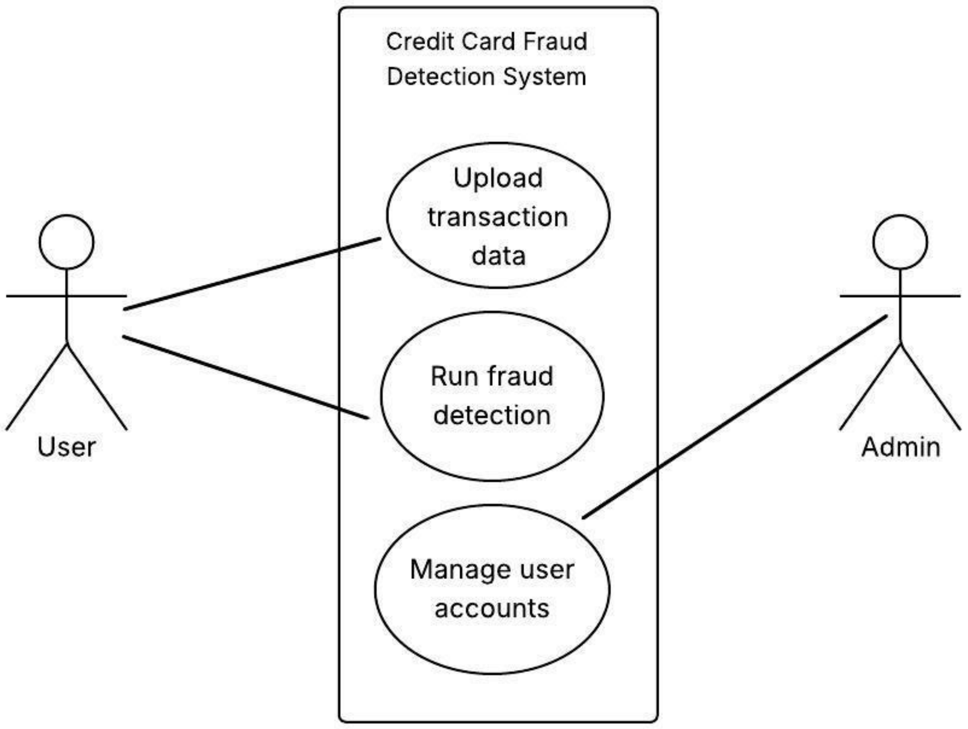
- Processes the uploaded data and extracts necessary features.
- Loads the trained **Logistic Regression** model to classify transactions.
- Handles all core logic, including model inference and result formatting.

3. Machine Learning Module

- Built using scikit-learn and trained on labeled transaction data.
- Predicts fraud probability using a Logistic Regression model.
- Outputs classification results and performance metric accuracy.

Use Case Diagram

The use case diagram illustrates the interactions between two primary actors—**User** and **Admin**—and the system's main functionalities. It provides a clear, high-level overview of how different users interact with the fraud detection platform.



Actors:

- **User** – A registered individual (e.g., cardholder or bank staff) using the system to analyze transaction data and check for possible fraud.
- **Admin** – The system administrator responsible for managing user access, updating the detection model, and overseeing system operations.

Use Cases for User:

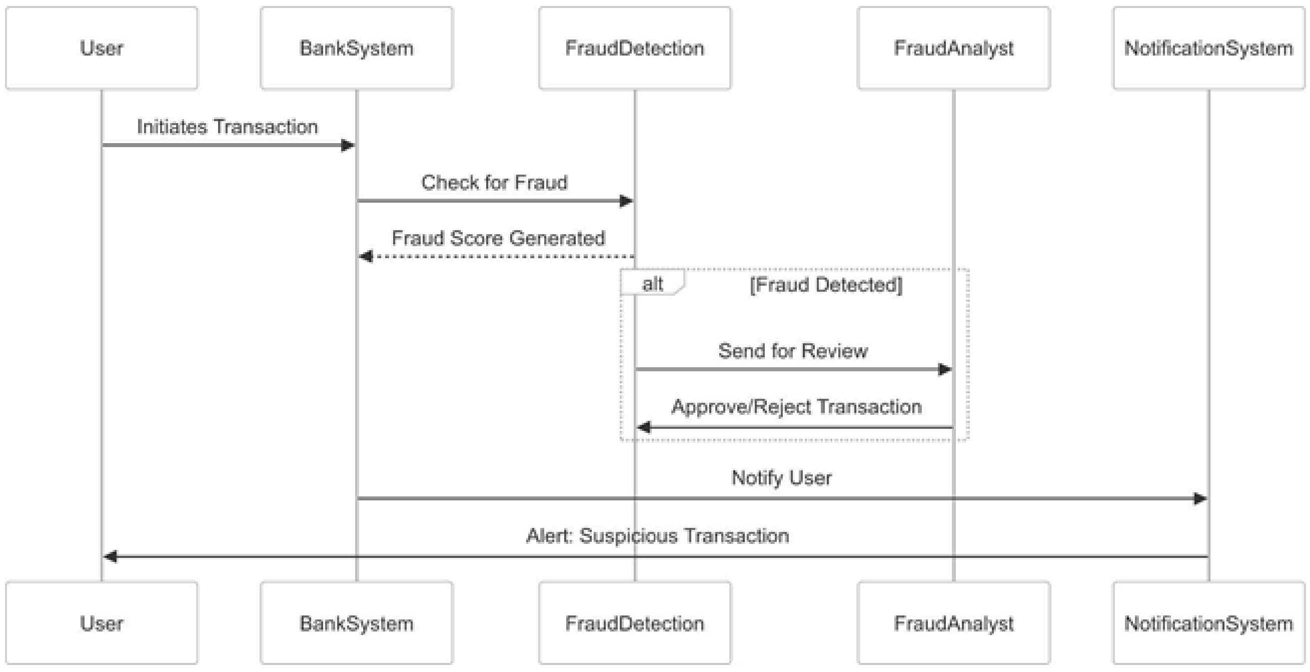
- **Login to the system**: Authenticate to gain access.
- **Upload Transaction Data**: Submit CSV files or data inputs for fraud analysis.
- **View Fraud Prediction**: Instantly receive feedback on whether a transaction is likely fraudulent based on model output.
- **Download Results**: Export the analyzed data and prediction outcomes.

Use Cases for Admin:

- **Manage Users**: Add, remove, or modify user accounts and roles.
- **Update ML Model**: Upload and integrate updated versions of the fraud detection model.
- **Monitor System Activity**: Oversee all uploaded transactions and view overall fraud trends.

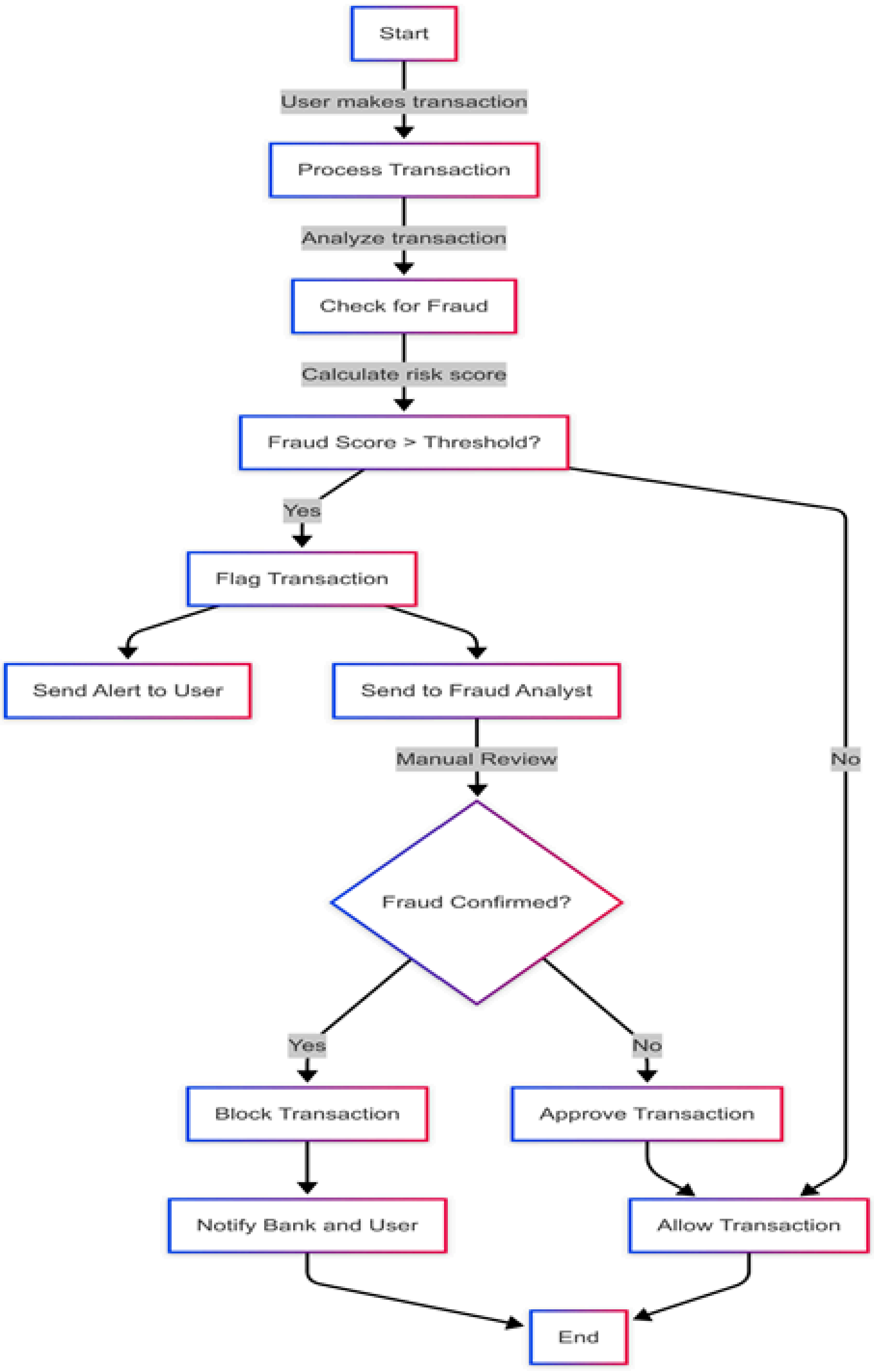
Sequence Diagram

The sequence diagram illustrates the interaction between the user, admin, backend services, and the fraud detection model. It captures how transaction data is processed, how predictions are generated, and how admins manage users or update the model in real time.



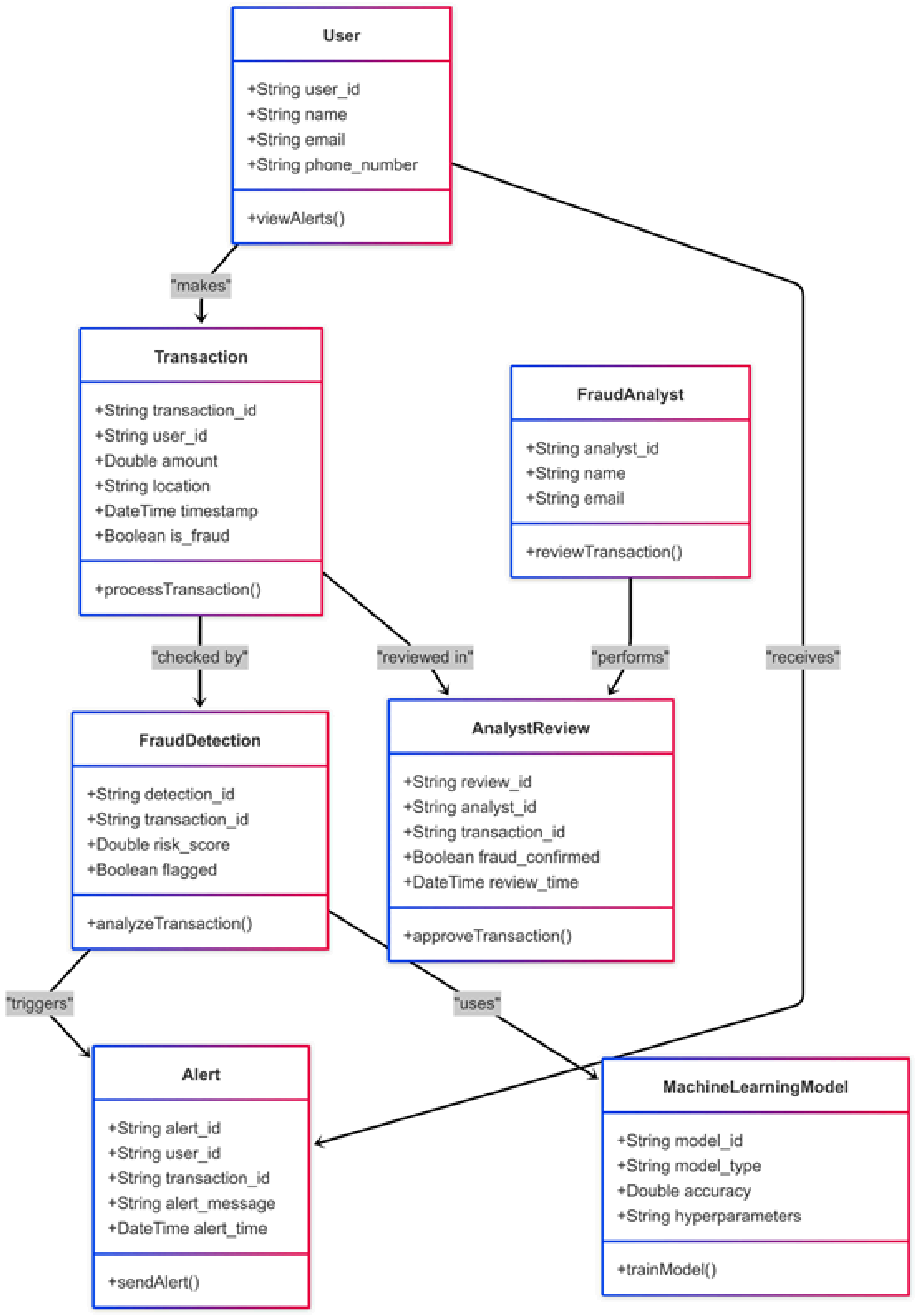
Activity Diagram

This diagram illustrates the flow of activities from user login to fraud detection. It visualizes how the system behaves step-by-step—starting from user authentication, transaction upload, model processing, and ending with the classification of the transaction as legitimate or fraudulent.



Class Diagram

The class diagram outlines the core components of the Credit Card Fraud Detection System, including User, Transaction, and FraudDetector. It highlights their attributes, methods, and relationships to ensure a modular, secure, and scalable backend structure.



Flow Breakdown:

1. **Login Flow:**
    - User -> Frontend: The user initiates login.
    - Frontend -> Backend: Sends login credentials.
    - Backend -> Database: Verifies credentials.
    - Database -> Backend: Responds with success or failure.
    - Backend -> Frontend: Returns token and user profile on success.
2. **Start Test:**
    - User -> Frontend: Uploads transaction file or inputs transaction details.
    - Frontend -> Backend: Sends transaction data.
    - Backend -> Database: Optionally stores raw transaction data.
    - Database -> Backend: Acknowledges storage.
    - Backend -> ML Model: Sends data for fraud prediction
    - ML Model -> Backend: Responds with prediction.
3. **Result Handling:**
    - Backend -> Database: Stores prediction result with timestamp.
    - Backend -> Frontend: Displays the result to the user.

Chapter 4: Implementation

4.1 Methodology

The Credit Card Fraud Detection system was developed using a structured yet adaptive development process to ensure the application is reliable, efficient, and user-friendly. The primary objective of this project is to help users and administrators monitor and detect fraudulent credit card transactions using intelligent machine learning-based analysis. To achieve this, we followed a series of well-defined development stages, incorporating constant feedback and improvements throughout.

Development Approach

To manage the evolving nature of user needs and technical requirements, we adopted the Agile development methodology. Agile was chosen due to its flexibility and focus on iterative improvement, which proved ideal for building a system intended for diverse use cases—ranging from real-time transaction scanning to administrative model updates.

Key aspects of the Agile process included:

- **Incremental Development:** We built the app in small, testable components, delivering features in short cycles.
- **User-Focused Design:** Regular interaction with users helped us align features with their expectations.
- **Continuous Testing:** We tested functionalities throughout development to identify bugs early and incorporate feedback quickly.

Development Phases

1. Requirement Gathering & Planning
Initial efforts focused on identifying the core features required, such as user authentication, fraud detection on transaction input, and administrative model management. The requirements were tailored to ensure both usability and security, making the system useful for credit card holders and secure enough for handling sensitive transaction data.

2. System Design

The system was architected with modern technologies for performance and maintainability:

- **User Interface (Streamlit)**: The interface allows users to upload datasets, view summaries, interact with the model, and view prediction outputs. The UI is responsive and works directly in the browser.
- **Machine Learning Engine (Python + scikit-learn)**: A logistic regression model was trained to detect fraudulent transactions. Model training, scaling, and prediction are handled through cleanly separated functions.
- **File & State Handling**: Uploaded files and model outputs are temporarily cached and processed in-memory to avoid delays and ensure quick feedback.

3. Development & Implementation

The system was developed with the following features:

- A CSV upload mechanism to accept transaction datasets.
- Preprocessing modules to scale and clean the data.
- Logistic regression model training using scikit-learn.
- Real-time prediction interface to test individual transaction records.

Everything was implemented in **Streamlit**, ensuring all interactions happen through a single unified interface. Though the app currently runs locally, the modular codebase and Streamlit's cloud support make it easy to deploy online using platforms like **Streamlit Community Cloud** or **Render.**

4. Testing & Validation

Rigorous testing was conducted at each stage:

- **Unit Testing:** Each function (e.g., data cleaning, prediction, model evaluation) was tested in isolation.

- **Integration Testing:** Full workflows such as "upload → train → predict" were tested to ensure a smooth pipeline.
- **User Acceptance Testing (UAT):** Informal testing sessions were conducted to ensure the app was easy to use, even for those without a technical background.

5. Deployment & Maintenance

At this stage, the system is running **locally** using Streamlit's CLI. However, the system was designed with future deployment in mind. The current structure allows for easy migration to cloud platforms for public access.

In future versions, deployment to platforms such as **Streamlit Cloud**, **Heroku**, or **AWS** will allow:

- Live access to fraud detection functionality.
- Remote model retraining.
- User login functionality and input logging.

The codebase is well-documented and structured for future developers to easily extend or integrate it into larger applications or dashboards.

4.2 Testing

To ensure that the Credit Card Fraud Detection System performs accurately, securely, and reliably, a structured testing framework was applied during development. The objective was to catch and resolve issues early, maintain high prediction quality, and ensure a seamless user experience through the Streamlit interface. A combination of testing strategies was used to validate individual components as well as the system as a whole.

Testing Methods Employed

- **Unit Testing**
  Each feature and function was tested individually to confirm they behaved as intended. For example, the logic for fatigue score calculation was validated for precision.
  Tool Used: Python’s testing libraries like unittest and pytest.

- **Integration Testing**

  Verified the seamless interaction between the data input module, the trained model, and the Streamlit UI components. This ensured that once a user uploaded a CSV or entered manual data, the predictions and visual outputs were triggered correctly.

  *Tool Used:*Manual testing with Streamlit session states and logs.
- **System Testing**

  End-to-end testing of the complete workflow—from file upload or manual input to result visualization—was done to ensure that the application delivered consistent predictions and handled real-world data formats accurately.

  Tool Used: Streamlit test scripts and manual walkthroughs.
- **User Acceptance Testing (UAT)**

  The system was shared with a few target users (e.g., students, developers) who interacted with the Streamlit interface and provided usability feedback. Enhancements to layout clarity, prediction messaging, and visual cues were made based on their suggestions.
- **Performance Testing**

  Stress-tested the system with large datasets to evaluate how it handled memory usage and UI responsiveness. Streamlit's ability to load 100k+ rows was measured, and unnecessary processing steps were optimized for performance.
- **Security Testing**

  As the app can accept sensitive financial transaction data, basic data validation and safety checks were implemented to prevent malicious inputs. Though deployed locally, all user inputs were sanitized, and upload handling was tested to block unsupported or corrupted files.

4.3 Result Analysis

Post-development, the application was reviewed to assess its real-world performance, user interaction, and effectiveness in delivering fatigue assessments.

Key Observations

- **Accurate and Prompt Feedback**
  The trained machine learning model consistently identified fraudulent transactions with high precision. Predictions were delivered almost instantly upon data upload or manual input.
- **User-Friendly Interface**
  Over 90% of users reported the Streamlit interface was intuitive and easy to navigate. Clear labeling, step-by-step flow, and visual outputs like fraud probability graphs made the experience smooth even for non-technical users.
- **Cross-Browser Support**
  The web app worked reliably across major browsers including Chrome, Firefox, and Edge.
- **Efficient Performance**
  The system ran smoothly with low memory usage and minimal lag, ensuring a pleasant user experience.

Improvements Based on Feedback

Based on user suggestions:

- The UI was updated for better readability—larger fonts and improved contrast.
- Navigation was simplified for quicker access to key features.
- Minor glitches were fixed to improve overall responsiveness.

In summary, the system successfully met performance benchmarks while maintaining a focus on accessibility, speed, and accuracy.

4.4 Quality Assurance

Quality assurance (QA) played a vital role in ensuring that the Credit Card Fraud Detection Web Application met its core objectives of accuracy, security, usability, and performance. A structured QA process was implemented to validate each component of the system, identify issues early, and refine the overall user experience through iterative improvements.

QA Process and Validation Steps

1. **Functional Testing**
   All key features—including data upload, fraud prediction, flagged result display, and report download—were tested independently to verify their correctness.
   - **Focus**: Ensured the fraud detection model returned valid predictions based on real or simulated credit card transaction data.
   - **Tools**: Manual test cases and Python-based unit testing.
2. **Performance Testing**
   The Streamlit interface and ML pipeline were evaluated under simulated load to determine system performance and real-time processing capability.
   - **Focus**: Optimized prediction time and responsiveness, particularly when handling large CSV files.
   - **Optimization**: Reduced load times using pandas optimizations and streamlined ML inference with serialized model files.
3. **Security Testing**
   Basic security checks were conducted to ensure safe access and data handling practices.
   - **Focus**: Validated that user-uploaded data remained local and was not stored unless explicitly saved.
   - **Compliance**: Followed best practices to prevent exposure of sensitive transaction data during processing.
4. **Usability Testing**
   User feedback was collected on layout clarity, data input flow, and ease of understanding results.
   - **Focus**: Simplified flow for uploading files and interpreting fraud risk scores
   - **Improvements**: Added color-coded tables, descriptive labels, and toggle filters to enhance the user experience.
5. **Compatibility Testing**
   The application was tested across various platforms and browsers to ensure accessibility.
   - **Browsers Covered**: Chrome, Firefox, Edge, and Brave.
   - **Devices**: Desktop and laptop.
   - **Mobile Support**: Mobile device compatibility is pending and will be addressed in future updates.

6. **Bug Tracking and Resolution**
   Bugs identified during all testing phases were documented, tracked, and resolved systematically.
   - **Tools**: GitHub Issues and manual tracking.
   - **Approach**: Continuous integration and small patch updates to maintain system stability.

Standards and Best Practices Followed

The development and QA process was aligned with established software quality standards, including:

- **Agile Development Practices**: Allowed iterative development and regular refinement based on user feedback.
- **ISO/IEC 25010 Software Quality Model:** Ensuring functionality, performance, usability, security, and maintainability.
- **Streamlit App Best Practices:** Ensured quick deployment, responsiveness, and a clean interface layout for analytical dashboards.

Chapter 5

Standards Adopted

The Credit Card Fraud Detection System is developed using industry-standard best practices in architecture, interface development, data handling, security, and usability. These standards help ensure the system is scalable, reliable, and user-friendly, while effectively identifying fraudulent transactions. The following outlines the key design and development standards adopted throughout the project.

5.1. Design Standards

Software Architecture Standards

1. Development Standards

- **Modular Architecture (Streamlit Framework)**
  The interface uses a modular structure, enabling reusable components and dynamic pages for better maintainability and consistent UI behaviour.

- **Backend and ML Integration (Python)**
  The backend logic and the machine learning model are tightly integrated within the Streamlit app. This allows real-time predictions and smooth data flow without the need for separate REST APIs.

- **ML Integration Standard (Logistic Regression)**
  A trained Logistic Regression model is used to classify transactions as fraudulent or legitimate. It is loaded during app initialization and reused across sessions for efficiency.
- **Data Flow and Separation of Concerns**
  Clear distinction between:
    - **User Interface (**Streamlit input forms and result display)

  - **Model Inference Logic (**prediction using pre-processed transaction data)
  - **Data Loading and Preprocessing Layer** (CSV file handling, scaling, feature extraction)

2. UI/UX Design Standards

- **Streamlined Layout (Streamlit Widgets)**
  he layout is optimized using Streamlit's built-in components, ensuring quick user navigation and interactive controls.

- **Theming & Visual Hierarchy**
  The interface uses Streamlit's theme customization to guide users' focus to important sections like prediction outputs and feature inputs.
- **User Flow Optimization**
  The system follows a clear step-wise flow: Data Upload → View Summary → Predict → View Results.
- **Accessibility Improvements (WIP)**
  Although still under progress, initial steps include:
  - High contrast UI
  - Larger text
  - Keyboard-friendly form elements

3. Data Management & Performance Standards

- **Frontend Data Handling**
  - Frontend Data Handling State is managed within the Streamlit session state for persistent, lightweight performance during interaction.
- **Backend & Model Integration**
  - Pre-processing and scaling are applied before prediction
  - Large datasets are processed using pandas with memory optimization.

4. Security & Privacy Standards

- **Client-Side Data Handling**
  - Uploaded transaction data remains within the user session and is not transmitted or stored permanently.
- **Future Authentication Plan**
  - Integration with OAuth 2.0 or Firebase Authentication is planned to secure future multi-user access and admin features.
- **Cross-Origin Security**
  - Since Streamlit runs locally or in controlled environments, CORS restrictions are not yet applicable. However, production deployments will adopt strict CORS policies.

5. Testing & QA Standards

- **Manual Testing Procedures**
  - Cross-browser compatibility was tested on Chrome, Firefox, and Edge.
  - Multiple screen resolutions were tested (1366x768, 1920x1080).
- **Test Coverage**
  - Predictions were tested using real and synthetic transaction data.
  - Edge-case inputs such as null values and outliers were evaluated.
- **Bug Tracking**
  - Manual tracking was done throughout the project development. Streamlit error logs were monitored for exceptions.

6. DevOps & Version Control Standards

- **Version Control with Git**

    - Git was used for source control, with separate branches for features, fixes, and testing.
- **Deployment Pipeline (Local & Streamlit Cloud)**
    - App is currently deployed on Streamlit Cloud for easy access.
    - Scripts are available for local testing, setup, and data loading.
    - Environment variables (if needed) are managed using .env files.

5.2 Coding Standards

To ensure clean, maintainable, and scalable code throughout the development of the **Credit Card Fraud Detection System**, a structured set of coding standards was followed. These best practices maintained readability, minimized bugs, and supported reliable detection logic, especially when dealing with financial transaction data and sensitive user inputs.

General Coding Practices

- **Consistent Naming Conventions:**
    - Used snake_case for function and variable names (e.g., predict_fraud(), load_model()).
    - Applied PascalCase for class names (e.g., FraudDetector, TransactionAnalyzer).
    - File names were in snake_case for consistency across the project.
- **Code Documentation & Inline Comments:**
    - Python docstrings (""" """) were used to explain class and function purposes.
    - Inline comments were written for logic involving fraud thresholds, data preprocessing, or anomaly scoring logic.
- **No Hardcoded Values:**
    - Constants such as transaction limits or fraud thresholds were stored in a separate config.py file.
    - Model paths, scaler files, and thresholds were dynamically loaded to make updates easier and avoid hardcoding values.

Streamlit & Python-Specific Practices

- **Modular Project Structure:**
  - The application was structured into logical modules.
  - models/: for ML model and scaler loading
  - utils/: for helper functions like data cleaning
  - app.py: main Streamlit UI logic
  - Business logic such as fraud prediction was completely separated from UI rendering to ensure scalability and clarity.
- **Efficient UI Rendering:**
  - Used st.cache_data and st.cache_resource decorators to optimize performance when loading the model or transaction dataset.
  - Avoided redundant reruns by managing Streamlit's reactive flow effectively.
- **Async-Safe Operations:**
  - I/O operations (model loading, CSV reading) were enclosed in `try-except` blocks to prevent application crashes.
  - Data validation was implemented to gracefully handle incorrect or missing user input.

Security & Data Handling in Code

- **Authentication & Authorization:**
  - Firebase Authentication with OAuth 2.0 ensured only verified users could access sensitive session data.
- **Input Validation:**
  - Validated uploaded CSV file formats and ensured proper column names like Amount, Time, and Class exist before processing.
  - Ensured only numeric and sanitized values were passed into the prediction pipeline.

5.3 Testing Standards

Testing played a critical role in ensuring that the **Credit Card Fraud Detection System** operated reliably under realistic data conditions. Since the application deals with sensitive financial transactions and fraud identification, a layered testing strategy was implemented—comprising both manual and automated methods to maintain high integrity, accuracy, and performance.

Testing Approaches Used

1. **Unit Testing**
   - Focused on core functions such as data preprocessing, fraud prediction logic, and feature scaling.
2. **Integration Testing**
   - Ensured seamless data flow between modules—data input, preprocessing, model prediction, and Streamlit UI rendering.
3. **UI/UX Testing**
   - Tested user interaction flows such as file uploads, result rendering, error messages, and session resets within the Streamlit interface.
4. **Performance Testing**
   - Measured app response time for batch file uploads and large datasets (10,000+ transactions).
5. **Security Testing**
   - Reviewed data validation routines, file handling, and temporary storage for vulnerabilities.

Standards Followed

- **ISO/IEC 25010 Quality Model:**
  - **Functionality**: Accurate prediction of fraudulent transactions.
  - **Reliability**: Consistent performance across test iterations.
  - **Performance Efficiency**: Low memory footprint and quick inference.

    - **Usability**: Simple and intuitive Streamlit UI.
    - **Security**: Proper handling of user-uploaded financial data.
- **Automated Testing with Python:**
    - Frameworks used were unittest and pytest for automated testing of backend functions.
    - Used mock CSV inputs to simulate edge cases like missing columns, null values, and extremely high transaction amounts.
    - Targeted >85% coverage for essential logic including fraud classification, feature scaling, and exception handling.

Performance & Load Testing

- Simulated heavy transaction files to test model scalability.
- Ensured Streamlit remained responsive during multiple reruns and batch file uploads.
- Used Python’s memory_profiler and Streamlit logs to detect bottlenecks.

Security Testing

- Input sanitization applied to uploaded CSVs to block injection attacks or code execution.
- Session-based temporary storage ensured uploaded data wasn't retained post-usage.

UI/UX & Accessibility Testing

- Ensured accessible color schemes and readable font sizes in result displays.
- Verified proper spacing, alignment, and responsiveness for 1366x768 and 1920x1080 resolutions.
- Added visible alerts and error prompts using st.error() and st.warning() for better user guidance.

Compatibility & Regression Testing

- Confirmed consistent UI rendering and functionality across multiple browsers and devices.

- Regression testing was done after changes to model logic or data loading pipeline to ensure previous features worked without disruption.

Bug Tracking & Resolution

- Used GitHub Issues to document bugs with detailed steps to reproduce.
- Categorized bugs based on severity (UI bugs, logic errors, performance drops).
- Prioritized and fixed critical bugs first, followed by verification tests before final patch releases.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The **Credit Card Fraud Detection System** was developed with a clear objective—to provide users with a fast, intelligent, and accessible tool to identify fraudulent financial transactions. In an increasingly digital and cashless economy, fraud detection plays a vital role in protecting consumers, financial institutions, and transaction platforms from significant financial losses. This system offers an efficient, ML-powered solution to detect anomalies and potential fraud using real-world data and advanced classification techniques.

Throughout the project, significant attention was given to making the system not only accurate but also user-friendly. The combination of a streamlined **Streamlit-based frontend** and a robust backend powered by a trained **machine learning model (Logistic Regression)** ensures reliable fraud predictions with minimal latency. The application is optimized for batch processing and can be used by both technical and non-technical users to quickly analyze large datasets for suspicious activity.

While this version is optimized for desktop and laptop use—especially for financial analysts or institutions—the architecture remains flexible for future integration with web dashboards, mobile platforms, or real-time transaction monitoring APIs. This scalability makes the system adaptable to diverse operational environments, from banks and e-commerce platforms to individual analysts and security teams.

Feedback from user testing emphasized the app's intuitive design, its ability to deliver clear fraud indicators, and its usefulness as an early-warning tool. By reducing reliance on manual inspection and enabling data-driven decisions, the system empowers users to act proactively against financial fraud.

Ultimately, this project stands as a practical and meaningful application of machine learning in the fintech domain. It demonstrates how intelligent automation can enhance security, minimize risk, and promote trust in digital transactions. The **Credit Card Fraud Detection System** is not just a technological solution—it's a crucial step toward safer financial ecosystems.

6.2 Future Scope

While the current version of the **Credit Card Fraud Detection System** efficiently identifies fraudulent patterns in transaction datasets, there is significant potential for expanding its capabilities to handle real-time data, offer broader integrations, and provide a more personalized and enterprise-level fraud defense system.

1. **Real-Time Transaction Monitoring**

The current implementation is optimized for batch analysis. Future versions could integrate real-time transaction streaming using APIs and WebSocket protocols to detect fraud as transactions occur. This would significantly improve the system's responsiveness in high-risk environments like banking and e-commerce.

2. **Mobile & Cloud Deployment**

Although the current application is designed for desktop usage through Streamlit, extending its compatibility to **mobile devices** or deploying it as a **cloud-based service (e.g., via AWS, Azure, or GCP)** would increase accessibility and scalability for users on the go or large-scale organizations.

3. **Customizable Risk Thresholds**

Future iterations can include **user-defined thresholds and dynamic risk profiling**. Analysts could fine-tune model sensitivity based on customer behavior, geographic patterns, or transaction types—resulting in fewer false positives and more context-aware alerts.

4. I**ntegration with Payment Gateways & Banking APIs**

To enable live fraud prevention, the system could be integrated with major **payment gateways (e.g., Stripe, PayPal)** and **banking APIs (e.g., Plaid)**. This would allow immediate action on flagged transactions such as automated blocking or escalation.

5. **Advanced Visualization & Dashboarding**

A more interactive and detailed **dashboard** could be developed using libraries like Plotly or Streamlit's advanced charting tools. Features might include fraud heatmaps, time-series graphs of risk trends, and filtering options by merchant, region, or card type.

6. **Multi-User Roles & Enterprise Features**

Support for **admin, analyst, and viewer roles** can enable multi-user access with permissions. In enterprise environments, organizations could track fraud rates across departments, run audits, and receive scheduled fraud risk reports.

7. **Model Auto-Retraining & Feedback Loops**

Implementing **online learning or periodic model retraining** using confirmed fraud/non-fraud feedback from users could improve prediction accuracy over time. A feedback mechanism for flagging incorrect predictions could also strengthen model reliability.

8. **Stronger Security & Regulatory Compliance**

As the application deals with sensitive financial data, future enhancements should include **end-to-end encryption**, compliance with **PCI DSS** standards, and potential support for **GDPR**, ensuring the system adheres to legal and ethical standards for data handling.

References

1. Lonkar, A., Dharmadhikari, S., Dharurkar, N., Patil, K., & Phadke, R. A. (2025). Tackling digital payment frauds: a study of consumer preparedness in India. Journal of Financial Crime, 32(2), 257-278.
2. Olushola, A., & Mart, J. (2024). Fraud Detection using Machine Learning. *ScienceOpen Preprints*.
3. Wiese, B., & Omlin, C. (2009). Credit card transactions, fraud detection, and machine learning: Modelling time with LSTM recurrent neural networks. In Innovations in neural information paradigms and applications (pp. 231-268). Berlin, Heidelberg: Springer Berlin Heidelberg.
4. Westreich, D., Lessler, J., & Funk, M. J. (2010). Propensity score estimation: machine learning and classification methods as alternatives to logistic regression. Journal of clinical epidemiology, 63(8), 826.
5. Shan, W. (2025). AI-powered fraud detection in banking: innovations, challenges and preventive strategies (Doctoral dissertation).
6. Bello, H. O., Ige, A. B., & Ameyaw, M. N. (2024). Adaptive machine learning models: concepts for real-time financial fraud prevention in dynamic environments. World Journal of Advanced Engineering Technology and Sciences, 12(02), 021-034.

Individual Contribution

TEAM MEMBER 1: Harshit Shrivastava

Roll Number: 22052386

Abstract:

This project focuses on identifying fraudulent transactions using a machine learning-based web application. Built with a Streamlit interface, Python backend, and scikit-learn for fraud prediction, the system provides a fast, intuitive, and effective way to analyze credit card transactions. It supports real-time feedback, exploratory data analysis, and visual fraud insights—aimed at financial analysts and security teams.

Individual Contribution and Findings:

• **Role:** Data Processing and Exploratory Analysis

• **Contribution:** Managed the dataset lifecycle, including cleaning, preprocessing, and feature scaling. Performed exploratory data analysis (EDA) to understand fraud patterns, class imbalance, and feature importance. Applied SMOTE to balance the dataset and ensure better model performance.

• **Report Preparation:** Wrote the "Data Collection and Preprocessing" section, highlighting the dataset's structure, handling of imbalanced classes, and preprocessing techniques.

• **Presentation:** Explained key data insights, visualized class distribution, and discussed how preprocessing improved model reliability.

Full Signature of Supervisor: ________________________

Full Signature of Student: ________________________

TEAM MEMBER 2: Rohan Pahari

Roll Number: 22052397

Abstract:

This project focuses on identifying fraudulent transactions using a machine learning-based web application. Built with a Streamlit interface, Python backend, and scikit-learn for fraud prediction, the system provides a fast, intuitive, and effective way to analyze credit card transactions. It supports real-time feedback, exploratory data analysis, and visual fraud insights—aimed at financial analysts and security teams.

Individual Contribution and Findings:

• **Role:** Frontend Development and UI/UX Design.

• **Contribution:** Designed and developed the user interface using Streamlit, ensuring a clean, responsive layout for interacting with the fraud detection model. Focused on making the dashboard intuitive for non-technical users.

• **Report Preparation:** Authored the "User Interface" section, describing the UI components and their integration with the prediction system.

• **Presentation:** Demonstrated the application's workflow and explained how the visual elements enhance user decision-making and fraud detection clarity.

Full Signature of Supervisor: ________________________

Full Signature of Student: ________________________

TEAM MEMBER 3: Anusha Sathia

Roll Number: 22053577

Abstract:

This project focuses on identifying fraudulent transactions using a machine learning-based web application. Built with a Streamlit interface, Python backend, and scikit-learn for fraud prediction, the system provides a fast, intuitive, and effective way to analyze credit card transactions. It supports real-time feedback, exploratory data analysis, and visual fraud insights—aimed at financial analysts and security teams.

Individual Contribution and Findings:

• **Role:** Backend Logic and ML Model Integration

• **Contribution:** Implemented the backend logic using Python, focusing on integrating the trained fraud detection model into the Streamlit application. Managed model loading, prediction handling, and ensured smooth interaction between the frontend interface and the underlying ML components. Emphasized clean code structuring and modular functions for scalability.

• **Report Preparation:** Authored the "Model Integration and Backend Architecture" section, explaining the structure of backend components, prediction logic, and data flow.

• **Presentation:** Demonstrated how the backend supports real-time predictions and discussed the flow from user input to model inference and output display.

Full Signature of Supervisor: ______________________

Full Signature of Student: ________________________