ONLINE PAYMENT FRAUD DETECTION USING MACHINE LEARNING

An Internship Project Report Submitted to

by

KADIRI SHAHIR BASHA

Under the Esteemed Guidance of

Mrs. REVATHI VENUGARI
Infosys Springboard Mentor

Introduction

The **Online Payment Fraud Detection Project** is an innovative solution designed to address the increasing threat of fraudulent activities in digital payment systems. With the global rise in online transactions, ensuring the security and integrity of payment platforms has become paramount. This project utilizes **Machine Learning (ML)** techniques to detect and prevent fraudulent transactions, providing an efficient, scalable, and reliable system for securing financial operations.

Objectives

The primary objectives of this project are:

- To build an ML-powered model capable of accurately classifying transactions as **Fraudulent** or **Non-Fraudulent** based on historical data.
- To achieve high precision, recall, and accuracy while minimizing false positives to reduce operational disruptions.
- To design a user-friendly application interface that supports both real-time fraud detection and batch analysis of transactions.

Significance

The project holds immense significance in safeguarding digital financial ecosystems by mitigating fraud-related risks and enhancing user trust. By leveraging data analytics and machine learning, the solution offers a proactive approach to combating fraud, which is vital in today's fast-evolving digital landscape.

Contributor

This project was conceptualized, developed, and implemented **individually** by **Shamad Shar**.

- **Role**: Full-Stack Developer and Machine Learning Specialist
- **Responsibilities**:
    - **Data Preprocessing**: Cleaning, balancing, and preparing a large dataset with over 1 million records for model training.
    - **Model Development**: Designing and evaluating multiple ML models, including hyperparameter tuning to enhance performance.
    - **System Integration**: Building the user interface for single and batch transaction prediction, and integrating it with the backend.
    - **Deployment**: Setting up the application for real-world usage, including hosting and deployment considerations.

This project showcases a comprehensive end-to-end workflow, from dataset preparation to deployment, reflecting the power of individual effort and technical expertise.

Project Scope

The **Online Payment Fraud Detection Project** was designed to develop a robust, machine learning-powered solution for identifying fraudulent transactions in online payment systems. The scope of the project was defined to ensure a focused, efficient, and scalable implementation, while also acknowledging certain limitations and constraints during the development process.

In-Scope

1. **Fraud Detection Model Development**:
   - Creation and evaluation of multiple ML models such as Random Forest, XGBoost, Logistic Regression, and more.
   - Selection of the best-performing model based on accuracy, precision, recall, and F1-score.
2. **Data Preprocessing**:
   - Handling large datasets with over 1 million records to ensure data quality.
   - Balancing the dataset by downsampling to 50,000 rows (25,000 fraud and 25,000 non-fraud) for effective model training.
   - Normalizing numerical features and encoding categorical data for model compatibility.
3. **User Interface (UI)**:
   - Designing a user-friendly interface for both single prediction and batch processing of transactions.
   - Implementation of pages like **Home**, **Single Prediction**, **Batch Prediction**, **History**, and **About Us**.
4. **Deployment**:
   - Deployment-ready solution, including integration with AWS S3 for secure data storage.
   - Ensuring compatibility with local and cloud environments for scalability.

Out of Scope

1. **Fraud Prevention Mechanisms**:
   - While the project detects fraudulent transactions, it does not implement automated fraud prevention or recovery mechanisms.
2. **External API Integration**:
   - The solution does not integrate directly with live payment gateways or banking systems.
3. **Real-Time Data Streaming**:
   - Processing and detecting fraud in real-time data streams was excluded; the focus was on historical transaction analysis.
4. **Comprehensive Financial Analysis**:
   - The project does not cover other financial insights or analysis beyond fraud detection.

Limitations and Constraints

1. **Dataset Constraints**:
    - The dataset was downsampled for manageability and balanced class distribution, which may not fully represent real-world imbalances.
    - Missing values and incomplete fields in the dataset required significant preprocessing efforts.
2. **Model Interpretability**:
    - Complex models like XGBoost and Random Forest provide high accuracy but are less interpretable compared to simpler models like Logistic Regression.
3. **Computational Resources**:
    - Training and hyperparameter tuning of certain ML models required high computational power, which was a constraint during model evaluation.
4. **Time Constraints**:
    - The project was completed within a limited timeframe, which restricted exploration of advanced techniques such as deep learning models or ensemble methods.
5. **Scalability and Real-Time Usage**:
    - Although the system is scalable for batch predictions, real-time prediction systems and live updates were not implemented in the current phase.

Requirements

Functional Requirements

1. **Fraud Detection System**:
    - The system should classify transactions as either **Fraudulent** or **Non-Fraudulent** with high accuracy.
    - The system should support both **single transaction prediction** and **batch processing** for multiple transactions.
2. **Dataset Handling**:
    - The system should load, preprocess, and manage large datasets with over 1 million records.
    - Missing values and categorical data should be handled appropriately.
3. **User Interface (UI)**:
    - Provide an interactive UI with features such as:
        - **Single Prediction Page**: Input transaction details (e.g., type, amount, old balance, new balance).
        - **Batch Prediction Page**: Upload a file containing multiple transactions for analysis.

        - **History Page**: Display previously analyzed transactions and their results.
        - **About Us Page**: Offer insights about the app and its developer.

4. **Model Selection and Tuning**:
    - Evaluate multiple ML models and select the best-performing one based on evaluation metrics (accuracy, precision, recall, and F1-score).
    - Support hyperparameter tuning for improved performance.
5. **Deployment**:
    - Enable deployment in both local and cloud environments with AWS S3 integration for dataset storage and retrieval.

Non-Functional Requirements

1. **Performance**:
    - The system should predict results within 1 second for single transactions and under 10 seconds for batch processing of up to 500 transactions.
2. **Scalability**:
    - The system should be designed to handle larger datasets and additional features in future versions.
3. **Usability**:
    - The UI should be intuitive and easy to navigate for users with minimal technical knowledge.
4. **Reliability**:
    - The application should maintain consistent performance and reliability across different environments.
5. **Security**:
    - Ensure secure handling of transaction data, with no sensitive data exposed or stored unnecessarily.
6. **Compatibility**:
    - The system should run seamlessly on various platforms (Windows, macOS, and Linux) and browsers (Chrome, Firefox, Edge).

User Stories

1. **As a Financial Analyst**, I want to upload a file of past transactions so that I can quickly identify fraudulent transactions for further investigation.
2. **As an Application User**, I want to enter transaction details manually to check if a specific transaction is fraudulent.
3. **As a Manager**, I want to view a history of analyzed transactions so that I can track the system's performance over time.
4. **As a Developer**, I want to deploy the system to both local and cloud environments for easier accessibility and scalability.

Use Cases

1. **Single Transaction Prediction**:
   - *Actors*: User, Fraud Detection System
   - *Flow*:
     - The user inputs transaction details such as Type, Amount, OldbalanceOrg, and NewbalanceOrg.
     - The system processes the data and predicts whether the transaction is fraudulent or not.
     - The result is displayed to the user with relevant details.
2. **Batch Transaction Analysis**:
   - *Actors*: User, Fraud Detection System
   - *Flow*:
     - The user uploads a file containing multiple transactions.
     - The system processes the file, classifies transactions, and generates a summary report.
     - The report is presented to the user with detailed metrics.
3. **Data Storage and Retrieval**:
   - *Actors*: User, AWS S3
   - *Flow*:
     - The user requests access to the dataset stored on AWS S3.
     - The system retrieves the file, preprocesses it, and makes it ready for use in training or prediction tasks.
4. **Model Evaluation and Selection**:
   - *Actors*: Developer
   - *Flow*:
     - The developer evaluates multiple ML models using metrics like accuracy, precision, recall, and F1-score.
     - The best model is selected, fine-tuned, and integrated into the application.

These requirements and use cases provide a clear roadmap for the project's design, development, and deployment phases.

Technical Stack

The technical stack for the **Online Payment Fraud Detection Project** encompasses the programming languages, frameworks, libraries, databases, tools, and platforms employed during the development process. The chosen stack ensures efficient data processing, accurate model predictions, and a user-friendly interface.

Programming Languages

- **Python**: Primary language for data preprocessing, machine learning model development, and user interface implementation.

Frameworks/Libraries

- **Data Handling and Analysis**:
  - **Pandas**: For data manipulation and preprocessing.
  - **NumPy**: For numerical computations.
- **Visualization**:
  - **Matplotlib** and **Seaborn**: For creating visualizations and analyzing feature distributions.
- **Machine Learning**:
  - **Scikit-learn**: For implementing models such as Logistic Regression, Random Forest, Decision Trees, SVC, and K-Nearest Neighbors.
  - **XGBoost**: For building and evaluating the XGBoost classifier.
  - **LightGBM**: For additional model experimentation.
- **Hyperparameter Tuning**:
  - **GridSearchCV**: For optimizing hyperparameters of ML models.
- **User Interface**:
  - **Streamlit**: For creating the interactive application with single and batch prediction functionality.

Databases

- **AWS S3**: Used as the storage solution for managing the cleaned dataset and enabling seamless retrieval during the development process.

Tools/Platforms

- **Integrated Development Environment (IDE)**:
  - **VS Code**: For writing, testing, and debugging the code.
- **Version Control**:
  - **Git and GitHub**: For version control and collaborative code management.
- **Cloud Integration**:
  - **AWS S3**: For securely storing and retrieving datasets.

- **Data Visualization**:
    - Tools like **Matplotlib** and **Seaborn** for analyzing and presenting results.

This technical stack ensured the project met its goals of efficient fraud detection, robust data.

Architecture/Design

The architecture of the **Online Payment Fraud Detection Project** is designed to provide efficient data processing, accurate predictions, and an interactive user experience. Below is an overview of the system's high-level components, their interactions, and the design considerations.

Overview of System Architecture

The project architecture consists of the following key components:

1. **Data Layer**:
    - **Dataset Storage**: The cleaned dataset is stored in an **AWS S3 bucket** for secure access.
    - **Preprocessing Module**: Handles missing values, categorical encoding, normalization, and data splitting.
2. **Machine Learning Layer**:
    - **Model Training**: Multiple models, including Random Forest, XGBoost, and Logistic Regression, are trained and evaluated.
    - **Model Evaluation and Selection**: Models are compared based on accuracy, precision, recall, and F1-score. Hyperparameter tuning is applied to improve performance.
    - **Final Model**: The best-performing model (Random Forest) is deployed for predictions.
3. **Application Layer**:
    - **Backend Processing**: Python scripts manage dataset loading, model integration, and result generation.
    - **Frontend Interface**: The **Streamlit** framework provides a user-friendly interface for single and batch predictions.
4. **Deployment Layer**:
    - **Local Deployment**: The application can run locally using Streamlit.
    - **Cloud Integration**: The dataset and application logic interact with AWS S3 for seamless data retrieval.

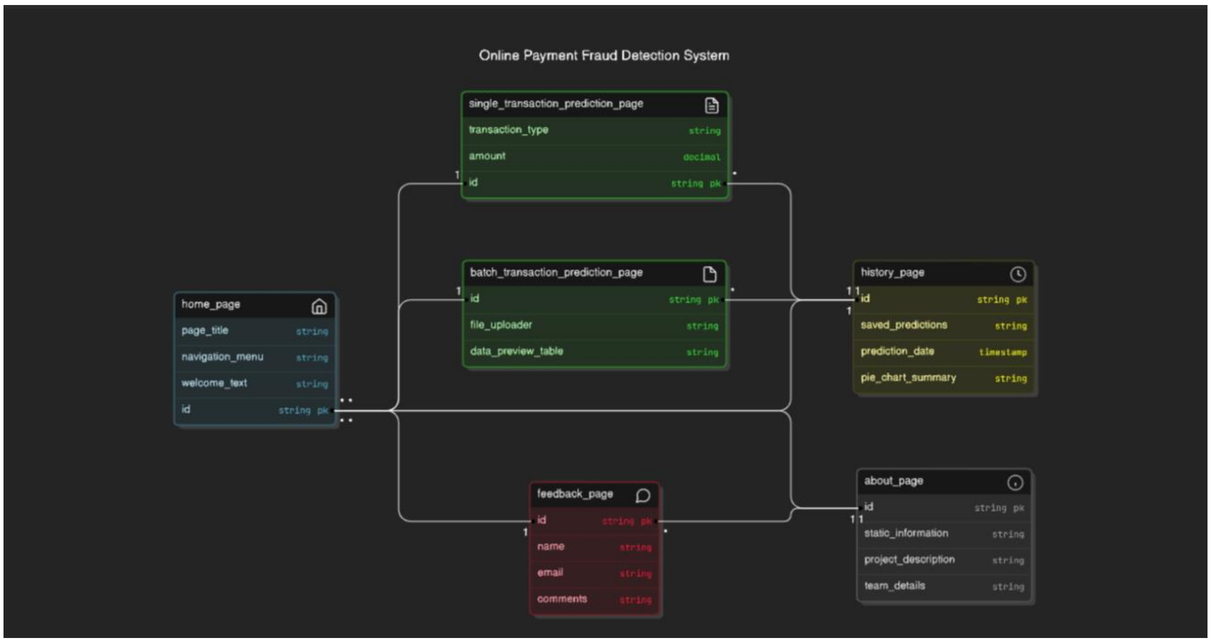
System Workflow

1. **Data Preprocessing**:
    - The dataset is loaded from AWS S3.
    - Missing values are handled, categorical features are encoded, and numerical features are normalized.
    - Data is split into training, validation, and testing sets.

2. **Model Training and Evaluation**:
   - Multiple models are trained on the preprocessed data.
   - Evaluation metrics are calculated, and the Random Forest model is chosen based on its superior performance.
3. **Prediction**:
   - **Single Transaction**: Users input transaction details to receive immediate fraud detection results.
   - **Batch Processing**: Users upload a file with multiple transactions for batch analysis.
4. **Visualization**:
   - The application visualizes data distribution and model performance metrics.

UML Diagram: System Architecture

Below is a textual representation of the UML diagram. You can also create a visual UML diagram using tools like Lucidchart, Draw.io, or MS Visio:



Design Considerations

1. **Modularity**:
   - Each component (data preprocessing, model training, and prediction) is modular, enabling easy updates and debugging.
2. **Scalability**:
   - The architecture supports future enhancements, such as adding more models or extending the UI.
3. **Security**:
   - Sensitive data handling is ensured by limiting local storage and leveraging secure AWS S3 integration.

The architecture ensures a seamless flow from data processing to user interaction, achieving efficient fraud detection with a focus on usability and scalability.

Development

The development phase of the **Online Payment Fraud Detection Project** focused on leveraging advanced technologies and frameworks to ensure accuracy, efficiency, and usability. Key aspects such as coding standards, best practices, and solutions to challenges were pivotal to the success of the project.

Technologies and Frameworks Used

1. **Data Processing**:
    - **Pandas** and **NumPy**: For data cleaning, transformation, and manipulation.
    - **Matplotlib** and **Seaborn**: For visualizing data distribution and feature relationships.
2. **Machine Learning**:
    - **Scikit-learn**: Provided models such as Logistic Regression, Random Forest, Decision Trees, SVC, and K-Nearest Neighbors.
    - **XGBoost** and **LightGBM**: Enhanced predictive performance with advanced boosting techniques.
    - **GridSearchCV**: Used for hyperparameter tuning to optimize model performance.
3. **Application Development**:
    - **Streamlit**: Built the interactive user interface for single and batch predictions.
4. **Cloud Integration**:
    - **AWS S3**: Used for storing the cleaned dataset and enabling secure, efficient data retrieval.
5. **Development Environment**:
    - **VS Code**: Served as the primary IDE for development and debugging.
    - **GitHub**: Used for version control and collaborative code management.

Coding Standards and Best Practices

- **Code Readability**:
    - Followed **PEP 8** guidelines for Python coding.
    - Used descriptive variable and function names.
- **Modularity**:
    - Structured the project into reusable modules for preprocessing, model training, and prediction.
- **Error Handling**:
    - Implemented robust exception handling for smoother execution.
- **Documentation**:
    - Added inline comments and function-level docstrings for better code maintainability.

- **Version Control**:
    - Regularly committed code changes to GitHub with clear commit messages.

Challenges and Solutions

1. **Imbalanced Dataset**:
    - **Challenge**: The original dataset had a significant imbalance between fraudulent and non-fraudulent transactions.
    - **Solution**: Applied downsampling techniques to create a balanced dataset with equal instances of fraud and non-fraud classes.
2. **Model Overfitting**:
    - **Challenge**: Some models, particularly decision trees, showed overfitting during training.
    - **Solution**: Used hyperparameter tuning and cross-validation to mitigate overfitting.
3. **Dataset Size**:
    - **Challenge**: The dataset contained over a million rows, making it computationally expensive to process.
    - **Solution**: Reduced the dataset size to 50,000 rows (balanced classes) while maintaining diversity in features.
4. **Integration with AWS S3**:
    - **Challenge**: Initial integration with AWS S3 for dataset storage and retrieval encountered configuration issues.
    - **Solution**: Utilized **Boto3** library, ensured proper IAM role setup, and thoroughly tested S3 integration.
5. **User Interface Design**:
    - **Challenge**: Designing a user-friendly interface that handles both single and batch predictions.
    - **Solution**: Leveraged **Streamlit**, which simplified UI creation with minimal coding.
6. **Performance Optimization**:
    - **Challenge**: Ensuring models performed efficiently while handling large datasets.
    - **Solution**: Optimized model parameters and utilized efficient libraries like XGBoost and LightGBM.

Testing

Testing was an integral part of the **Online Payment Fraud Detection Project**, ensuring the system's reliability, accuracy, and robustness. The testing approach included unit, integration, and system testing to validate each component and the system as a whole. Below is a detailed account of the testing methodologies, results, and resolutions.

Testing Approach

1. **Unit Testing**:
   - **Objective**: Validate individual modules, such as data preprocessing, model training, and predictions.
   - **Scope**:
     - Checked the correctness of data handling operations (e.g., missing value imputation, normalization, and encoding).
     - Verified that each machine learning model produced outputs consistent with training data and expected behavior.
   - **Tools**:
     - **Pytest**: Used for automating unit tests.
2. **Integration Testing**:
   - **Objective**: Ensure smooth interaction between system components, such as data preprocessing, AWS S3 integration, and model prediction.
   - **Scope**:
     - Tested AWS S3 integration to verify dataset upload and download operations.
     - Validated interaction between the trained model and Streamlit user interface for single and batch predictions.
   - **Tools**:
     - **Mocking**: Simulated AWS S3 interactions to test component integration.
3. **System Testing**:
   - **Objective**: Test the entire system end-to-end for real-world scenarios.
   - **Scope**:
     - Verified end-to-end workflow from dataset retrieval, preprocessing, and model prediction to displaying results on the UI.
     - Assessed application performance under various scenarios, including edge cases (e.g., invalid inputs or missing fields).
   - **Tools**:
     - Manual testing with various datasets and user inputs.

Testing Results

1. **Unit Test Results**:
   - All preprocessing modules passed the tests with correct outputs.
   - Machine learning models returned expected metrics during validation, confirming their correctness.

    - Detected a bug in categorical encoding where new categories in test data caused an error. **Resolved** by implementing a fallback mechanism for unseen categories.

2. **Integration Test Results**:
    - AWS S3 integration was initially prone to timeout errors for large datasets. **Resolved** by optimizing file handling and using chunked uploads/downloads.
    - Streamlit UI successfully displayed predictions after model interaction tests.

3. **System Test Results**:
    - The system handled both single and batch predictions accurately.
    - Performance testing revealed slight latency for batch predictions with large datasets. **Resolved** by optimizing the prediction loop.
    - Error-handling tests ensured appropriate messages for invalid or incomplete inputs.

Bugs and Issues Discovered

Bug/Issue	**Category**	**Resolution**
Unseen category error in test data	Unit Testing	Implemented fallback encoding for new categories.
AWS S3 timeout during dataset upload	Integration Testing	Optimized file handling using chunked uploads.
Latency in batch predictions	System Testing	Optimized prediction loop for better performance.
Invalid input error handling	System Testing	Added detailed error messages for incomplete inputs.

Deployment

The deployment of the **Online Payment Fraud Detection Project** involved setting up the application for smooth deployment across multiple environments. This section outlines the process, automation, and instructions for deployment in different environments.

Deployment Process

1. **Preparation**:
    - The application was packaged to include all necessary components, such as ML models, datasets, and Streamlit code, ensuring it was ready for deployment.
    - All dependencies required for the project were listed in the requirements.txt file to ensure easy installation and compatibility.

2. **Environment Setup**:
    - The deployment environment required:
        - **Python 3.8+** for the application to run.

        - **AWS CLI** for interaction with the AWS S3 bucket (if applicable for data retrieval).
        - The installation of the necessary libraries and frameworks as specified in the requirements.txt.

3. **Hosting**:
    - The application was hosted using **Streamlit Cloud** for easy deployment and real-time updates.
    - The project repository was uploaded to **GitHub** and linked to **Streamlit Cloud**, which enabled continuous deployment.
    - Secure environment variables were configured, particularly for sensitive information such as AWS credentials.
4. **Automation**:
    - A deployment script was created to automate the setup and deployment process, ensuring a smooth transition from development to production.
5. **Testing Deployment**:
    - The hosted application was tested to verify the full functionality of the fraud detection features, including single and batch predictions.
    - Cross-platform compatibility was ensured by testing the application on multiple operating systems (Windows, macOS, and Linux)

Instructions for Deployment

1. **Local Deployment**:
    - Clone the project repository from GitHub.
    - Set up a Python virtual environment and install the necessary dependencies listed in the requirements.txt.
    - Run the application locally using Streamlit.
2. **Deployment on Streamlit Cloud**:
    - Push the repository to GitHub.
    - Log in to **Streamlit Cloud** and create a new app linked to the GitHub repository.
    - Set up any necessary environment variables for secure access, such as AWS credentials, and deploy the application.

User Guide

This section provides instructions for using the **Online Payment Fraud Detection Project** application. It includes setup and configuration details as well as troubleshooting tips for common issues.

Instructions for Using the Application

1. **Setup and Configuration**:
    - Ensure that **Python 3.8+** is installed on your system.

    - Clone the project repository from GitHub.
    - Set up a Python virtual environment and install the required dependencies.
2. **Running the Application**:
    - After setting up the environment, run the application using Streamlit, which will open in your default web browser.
3. **Using the Application**:
    - **Single Prediction**: Input transaction details such as type, amount, old balance, and new balance. Click the "Predict" button to classify the transaction as "Fraud" or "Non-Fraud".
    - **Batch Prediction**: Upload a CSV file containing multiple transactions for batch prediction. The results will be displayed in a downloadable format.
    - **History**: View the history of previous predictions made by the application.
    - **About Us**: View information about the project and its development.

Troubleshooting Tips

- **Issue: The application doesn't start or shows an error after running the application.**
    - **Solution**: Ensure all dependencies are correctly installed. Check if the virtual environment is activated properly and if the requirements.txt file is followed accurately.
- **Issue: The prediction result shows "Error" or "Invalid Input".**
    - **Solution**: Ensure all required fields are correctly filled in. Check the input data format for any errors, especially for numerical values like amount and balance.
- **Issue: Batch prediction fails to upload the file.**
    - **Solution**: Ensure the CSV file format is correct and that the column names match the expected format. Verify the file size to ensure it's within the app's processing capacity.
- **Issue: The app is not running on Streamlit Cloud.**
    - **Solution**: Check that the repository is correctly linked to **Streamlit Cloud** and that the deployment was successful. Verify your internet connection and try redeploying the application if necessary.

Conclusion

The **Online Payment Fraud Detection Project** successfully implemented a machine learning-based solution to detect fraudulent transactions in real-time.

Key outcomes include:

- **High-Performance Model**: The Random Forest model demonstrated excellent accuracy, precision, and recall, outperforming other models such as Logistic Regression and Support Vector Classifier.
- **User-Friendly Interface**: The application offers an intuitive interface for both single and batch predictions, providing users with clear and actionable insights.

- **Cloud Deployment**: The application was successfully deployed on **Streamlit Cloud**, ensuring ease of access and deployment for users.

Reflections and Areas for Improvement:

- **Lessons Learned**:
    - The importance of effective data preprocessing and feature engineering to ensure the accuracy and robustness of the model.
    - Leveraging ensemble models like Random Forest for handling imbalanced datasets and achieving high detection performance.
    - The continuous improvement of the model through updates and maintenance to reflect new trends in fraudulent activities.
- **Areas for Future Improvement**:
    - **Model Optimization**: Further hyperparameter tuning and the exploration of more advanced techniques, such as **Deep Learning** or **XGBoost**, could improve fraud detection accuracy.
    - **Scalability**: Optimizing the application for handling larger datasets efficiently, ensuring faster predictions for real-time systems.
    - **Feature Enhancement**: Adding additional features such as user behavior, geographical data, and transaction timestamps could improve the model's performance in detecting fraud.

Appendices

Additional Diagrams:

- **System Architecture Diagram**: Provides a high-level view of the components and their interactions in the fraud detection system.
- **Flowchart of the Prediction Process**: Illustrates the flow of data from user input to fraud detection result.

Research References:

- **Machine Learning for Fraud Detection**: A research paper outlining the significance of machine learning in fraud detection, which helped guide the choice of models and techniques for this project.
- **Data Preprocessing Techniques**: A study on best practices for data cleaning and preprocessing, which informed the handling of missing values, scaling of numerical features, and encoding of categorical variables in the dataset.