**Project Documentation: Online Payment Fraud Detection System**

**1. Introduction**

The **Online Payment Fraud Detection System** aims to identify fraudulent transactions in real-time, enhancing the security of online payment platforms. The system uses the **LightGBM model**, known for its speed and high performance, to predict fraudulent transactions with accuracy. The application is deployed using **Streamlit**, offering an interactive interface for users to test transactions and analyze batch files.

**2. Project Scope**

* **In Scope:**
  + Detection of fraudulent transactions using supervised learning.
  + Real-time individual transaction prediction via Streamlit.
  + Batch processing for fraud detection across datasets.
  + Visualization of transaction statistics and fraud distribution.
* **Out of Scope:**
  + Integration with live payment gateways.
  + Detection of fraud based on external behavioral patterns or user metadata.
* **Constraints:**
  + Model performance is limited to the quality of training data.
  + Deployment is designed for local or small-scale environments.

**3. Requirements**

* **Functional Requirements:**
  + Predict whether a transaction is fraudulent based on inputs: transaction type, amount, old balance, and new balance.
  + Support batch file uploads for bulk predictions.
  + Provide clear visualizations for fraud analysis.
  + Allow users to report suspicious transactions.
* **Non-Functional Requirements:**
  + Ensure fast predictions (< 1 second per transaction).
  + Provide a user-friendly interface.
  + Maintain data privacy and security.
* **User Stories:**
  + As a user, I want to input transaction details and determine if they are fraudulent.
  + As an admin, I want to analyze large datasets for fraud trends.

**4. Technical Stack**

* **Programming Languages:** Python
* **Frameworks/Libraries:**
  + Machine Learning: LightGBM, scikit-learn
  + Data Analysis: Pandas, NumPy
  + Visualization: Plotly, Matplotlib
  + UI/Deployment: Streamlit
* **Databases:** CSV files for input/output data storage
* **Tools/Platforms:** Streamlit, GitHub, Google Colab

**5. Architecture/Design**

* **System Architecture:**
  + Data is preprocessed and passed into the **LightGBM model** for predictions.
  + The Streamlit interface connects the user to the prediction engine.
  + Batch predictions are processed and visualized with fraud statistics.
* **Diagrams:**
  + System architecture and workflow diagrams (e.g., Input → Preprocessing → Model → Output).
  + Fraud distribution pie charts for batch analysis.
* **Design Decisions:**
  + Chose LightGBM for its ability to handle imbalanced datasets efficiently.
  + Streamlit was selected for rapid UI development and deployment.

**6. Development**

* **Technologies and Frameworks:**
  + LightGBM for classification.
  + Streamlit for interactive visualization and deployment.
* **Coding Standards:**
  + Followed PEP8 for Python coding style.
  + Used modular programming to enhance code readability and reusability.
* **Challenges:**
  + Balancing precision and recall due to class imbalance in fraud detection.
  + Ensuring Streamlit's interface remains responsive during batch processing.

**7. Testing**

* **Approach:**
  + **Unit Tests:** Validated individual functions, including data preprocessing and predictions.
  + **Integration Tests:** Ensured seamless interaction between the Streamlit UI and LightGBM model.
  + **System Tests:** Evaluated the entire application with real-world datasets.
* **Results:**
  + Validation Accuracy: 98.70%
  + Validation Precision: 97.88%
  + Validation Recall: 99.50%
  + Validation F1 Score: 98.69%
  + Test Set Evaluation:
  + Test Accuracy: 98.74%
  + Test Precision: 98.21%
  + Test Recall: 99.37%
  + Test F1 Score: 98.79%
  + No critical bugs identified; resolved UI rendering delays.

**8. Deployment**

* **Process:**
  + Deployed using Streamlit for a web-based interface.
  + Dependencies managed via requirements.txt for easy setup.
  + Hosted locally for demonstration purposes.
* **Instructions:**
  + Clone the repository.
  + Install dependencies using pip install -r requirements.txt.
  + Run streamlit run app.py to start the application.

**9. User Guide**

* **Setup:**
  + Ensure Python is installed.
  + Install the required packages from requirements.txt.
* **Usage:**
  + **Individual Transaction:** Input details in the Streamlit form to predict fraud.
  + **Batch Processing:** Upload a CSV file to analyze multiple transactions simultaneously.
* **Troubleshooting:**
  + If predictions fail, check if the required columns (type, amount, oldbalanceOrg, newbalanceDest) are present in the uploaded CSV.

**10. Conclusion**

The **Online Payment Fraud Detection System** successfully identifies fraudulent transactions using machine learning. The LightGBM model demonstrated high accuracy, making it suitable for real-world scenarios. The Streamlit interface provides an intuitive platform for predictions and batch analysis.

* **Lessons Learned:**
  + Class imbalance is a critical challenge in fraud detection.
  + Simplified UIs are crucial for user adoption.
* **Future Enhancements:**
  + Integrate with live payment systems for real-time detection.
  + Add support for additional fraud detection models.

**11. Appendices**

* **Sample Input Data:**
  + A sample CSV file with required fields for batch predictions.
* **Code Snippets:**
  + Example of LightGBM training script.
* **References:**
  + LightGBM documentation, Streamlit user guide.

**Additional Files**

* A zip file containing:
  + Source code (app.py, LightGBM model file).
  + requirements.txt for dependencies.
  + Sample dataset for testing.