Infosys Internship 4.0 Project Documentation

Title: Project Documentation: Online Fraud Payment Detection System

Name: Nihaarika Jami
Role: AI-ML Intern

•Introduction:

Our project aims to create a robust system to detect fraudulent transactions in real-time and to create a secure and trustworthy environment for digital transactions, benefiting consumers, businesses, and the economy as a whole. With the rapid increase in online transactions, traditional methods of fraud detection, such as manual review, are no longer sufficient. Cybercriminals are constantly evolving their tactics, making it essential to employ sophisticated techniques to safeguard financial transactions.
The outcome of this project is a predictive model capable of accurately classifying transactions as fraudulent or non-fraudulent, thereby empowering financial institutions to take proactive measures against fraudulent activities.

Objectives:

1. **Real-Time Fraud Detection:** Utilize advanced machine learning techniques to analyze transaction data and identify potential fraud instantly.
2. **User-Friendly Interface:** Develop an intuitive platform that allows users to input transaction details or upload batch files for bulk prediction effortlessly.
3. **Enhanced Security:** Minimize false positives by accurately distinguishing between legitimate transactions and fraudulent ones.
4. **Scalability:** Ensure the system can handle both individual transaction checks and bulk analysis efficiently.

Significance:

The Online Fraud Payment Detection System is not just a technological solution; it is a critical component of modern financial infrastructure. Its importance lies in its ability to:

- Protect financial integrity and enhance consumer trust.
- Mitigate financial losses and ensure regulatory compliance.
- Reduce operational costs and adapt to evolving threats.
- Support economic growth by providing a secure and reliable digital financial ecosystem.

Limitations and Constraints:

- **File Size Limitation:** The system restricts the size of the uploaded .csv files to a maximum of 200MB to ensure smooth processing and avoid performance issues.
- **Input Format Restriction:** The .csv file for batch prediction must have specific columns ("type," "amount," "oldbalanceOrg," and "newbalanceOrig") in a predefined order. Any deviation from this format results in a column mismatch error.
- **Computational Resources:** The efficiency and speed of real-time fraud detection depend on the computational resources available. Limited resources may affect the performance of the system

•Requirements:

1. **Functional Requirements:**

   a. **Transaction Analysis**
   - The system should be able to analyse transaction data to identify potential fraudulent transactions.
   - The system should be able to use machine learning algorithms to predict the likelihood of a transaction being fraudulent.
   - Single Transaction Prediction and Batch Transaction Prediction: Users must be able to input transaction details such as type, amount, old balance at origin, and new balance at origin for the Single transaction prediction and .csv file for batch file prediction.

   b. **Data Handling**
   - The system should be able to handle large volumes of transaction data.
   - The system should be able to process transactions in real-time.

   c . **Prediction and Alerting**
   - The system should be able to provide real-time predictions of fraudulent transactions.
   - The system should be able to generate alerts for transactions that are predicted to be fraudulent.

Non-Functional Requirements:

a. **Performance:**
   - o The system should provide real-time predictions for single transactions and process batch files efficiently.
   - o It should handle up to 200MB of data for batch processing without significant delays.

b. **Scalability:**

  - The system should be scalable to handle an increasing number of transactions as the user base grows.

c. **Usability:**
  - The interface should be user-friendly and intuitive, allowing users to easily input transaction details or upload files.
  - It should provide clear instructions and feedback to guide users through the process.

d. **Security:**
  - The system should ensure the privacy and security of the transaction data provided by the users.
  - It should have basic security measures in place to prevent unauthorized access.

Technical Stack:

Programming Languages:

Python: Used for developing the core functionality, data processing, machine learning model training, and prediction.
CSS: Utilized sparingly within the Streamlit framework to create engaging frontend interactions.

Frameworks/Libraries:

- **Streamlit:** For building the user interface, including features like single transaction input, batch file upload, and displaying results.
- **Pandas:** For data manipulation and handling CSV files.
- **Scikit-learn:** For machine learning, including training the Random Forest Classifier model and scaling the data.
- **Joblib:** For saving and loading the trained machine learning model.
- **Pickle:** For saving and loading the pre-fitted StandardScaler.
- **PIL (Python Imaging Library):** For handling image operations, such as loading and displaying the logo.
- **Streamlit-option-menu:** For creating the horizontal menu in the user interface.

Databases:

AWS S3: Used for storing and retrieving pre-processed datasets, ensuring security and reliability. Intermediate datasets are saved in the form of csv file.

Tools/Platforms:

VSCode: For coding and debugging the Streamlit application.
GitHub: For code repository and project management.

•Architecture/Design:

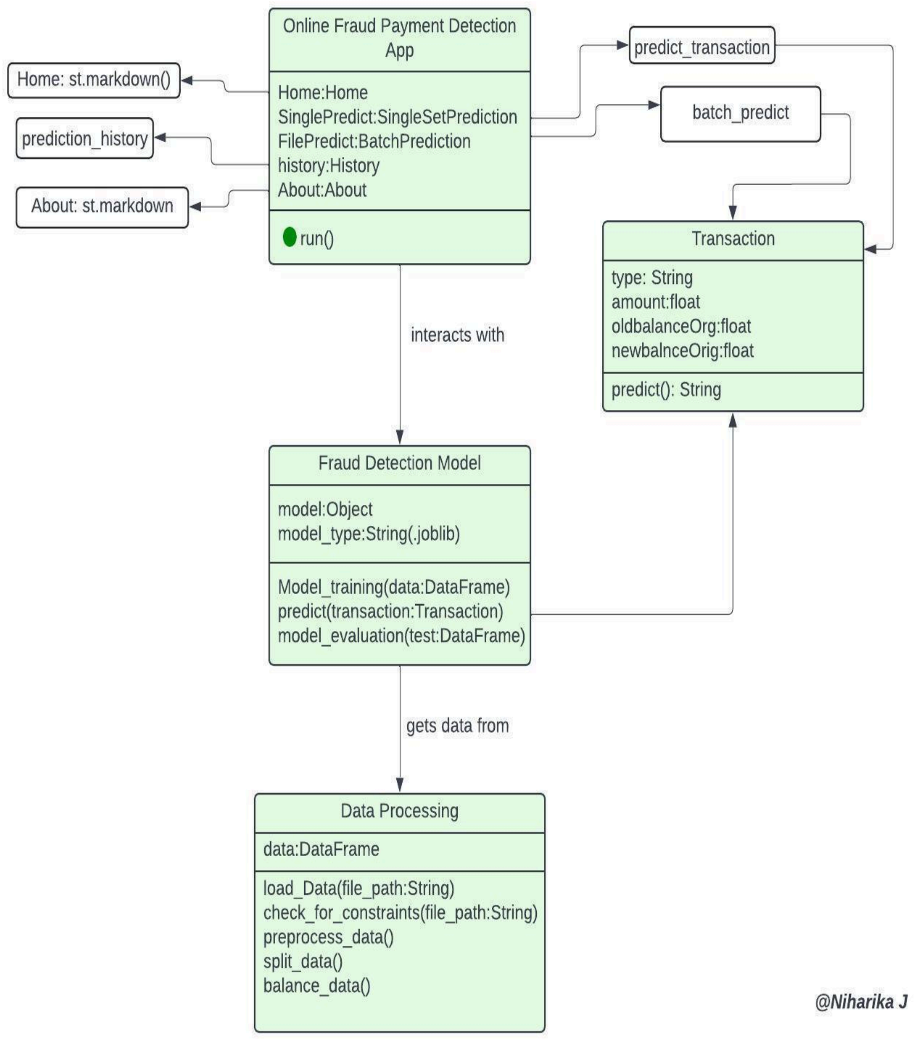
Overview:
The Online Fraud Payment Detection System consists of several high-level components that interact seamlessly to provide real-time and batch transaction fraud detection. The architecture is designed to be modular, scalable, and user-friendly.

High-level components:
User Interface (UI)
Backend Processing
Model and Scaler
File Handling
Error Handling and Reporting

Design Decisions:

- **Design Pattern:** MVC (Model-View-Controller) pattern for separation of concerns.
- **Trade-offs:** Chose Streamlit for rapid development despite its limitations in creating highly customizable UI components.
- **Training Model:** Choose RandomForestClassifier for its robustness and performance.

UMLDiagram



Trade-offs and Alternatives Considered:

1. **Database Integration:**
   - Initially considered using a database for storing transaction data and prediction results.
   - Decided against it to keep the project scope focused and simplify implementation. The current design processes data directly from CSV files.

2. **Model Choice:**

- o Evaluated several machine learning models (e.g., Logistic Regression, Decision Trees) before selecting Random Forest Classifier for its balance of accuracy and interpretability.

3. **Scaling and Performance:**

    - o Considered implementing a distributed processing system for handling larger datasets.
    - o Opted for local processing with a file size limit (200MB) to keep the system lightweight and manageable within the project constraints.

Development:

- The project was developed using Python and a combination of machine learning libraries like Scikit-learn and TensorFlow.
- Python's wide range of machine learning libraries required were crucial for developing the model.
- The model was built using a structured approach, starting with data exploration, cleaning and preprocessing, followed by model selection, and hyperparameter tuning, splitting and training the datasets.
- The development process adhered to coding standards and best practices, including modular code design, documentation, and code review.

Challenges:

Data Scaling

One of the significant challenges encountered during the development of the Fraud Detection System was related to data scaling, which initially led to incorrect predictions. Here's an in-depth look at this challenge and how it was resolved:

1. **Initial Incorrect Predictions:**
    - o **Issue:** During the early stages of testing, the machine learning model often produced inaccurate predictions.
    - o **Cause:** The primary cause was identified as the inconsistency in the scale of input features. The model struggled to accurately interpret features with vastly different ranges, such as transaction amounts and account balances.
2. **Solution: Data Scaling:**
    - o **Approach:** To address this issue, the dataset was standardized using techniques like normalization or standardization. This process involved scaling them to a fixed range.
    - o **Implementation:** Libraries such as scikit-learn were used to implement data scaling techniques.

•Testing:

7.1. Testing Approach:

1. **Unit Tests**: Individual components of the system, such as the machine learning model, data preprocessing, and prediction logic, were tested in isolation to ensure they functioned correctly.
2. **Integration Tests**: Components were tested together to ensure they interacted correctly and produced the expected results.
3. **System Tests**: The entire system was tested end-to-end to ensure it met the requirements and worked as expected.

Results:

The performance of the Random Forest model in detecting online fraud payments was evaluated using various metrics. The results are presented below:
Achieved high accuracy and precision on the validation dataset

1. **Accuracy**: The model achieved an accuracy of **0.9921**, indicating that it correctly classified approximately **99.21%** of the transactions.
2. **Precision**: The model's precision was **0.9906**, indicating that approximately **99.06%** of the transactions predicted as fraudulent were actual fraud cases.
3. **Recall**: The model's recall was **0.9941**, indicating that it detected approximately **99.41%** of the actual fraud cases.
4. **F1-Score**: The model's F1-score was **0.9923**, providing a balanced measure of precision and recall.

Deployment process:

The deployment process for the Online Payment Fraud Detection System using Machine Learning involves the following steps:

1. **Testing and Validation**: The code is tested and validated to ensure it meets the requirements and works as expected.
2. **Packaging**: The code is packaged into a deployable format, including any dependencies and configurations.

Deployment Scripting: Deployment scripts are used to automate the deployment process, including environment setup, configuration, and deployment of the application.

User Guide for Fraud Detection System

Setup and Configuration

1. **Environment Setup:**
   - Ensure you have a stable internet connection.
   - Use a modern web browser (Chrome, Firefox, Safari, etc.).
2. **Application Access:**
   - Navigate to the provided URL or local host where the Streamlit application is hosted.

Using the Application

1. **Home Page:**
   - **Overview:** Provides an introduction to the application and its key features.
   - **Functionality:** Explains the real-time fraud detection capabilities and the use of advanced machine learning models.
2. **Single-Predict Page:**
   - **Inputs:**
     - Select transaction type from the dropdown (e.g., CASH_IN, CASH_OUT).
     - Enter transaction amount, old balance original, and new balance original.
   - **Prediction:**
     - Click "Predict" to see if the transaction is fraudulent or not.
     - Results are displayed immediately below the input fields.
   - **Reporting Fraudulent Transactions:**
     - If a transaction is predicted as fraudulent, use the "Report Fraudulent Transaction" expander.
     - Enter transaction ID and account ID for reporting purposes.
3. **File-Predict Page:**
   - **Upload CSV File:**
     - Choose a CSV file with columns: type, amount, oldbalanceOrg, newbalanceOrig.
     - Click "Predict" to analyze multiple transactions at once.
     - Results are shown and can be downloaded as an updated CSV file.
4. **History Page:**
   - **View Prediction History:**
     - Displays a table of all predictions made, including transaction details and prediction outcomes.
     - Download prediction history as a CSV file for further analysis.
5. **About Page:**
   - **Usage Instructions:**
     - Step-by-step guide on how to use each feature of the application.
     - Explanation of the Random Forest model used for fraud detection and its performance metrics.

Troubleshooting Tips

1. **Session State Issues:** If session variables (e.g., prediction history) are not persisting, try refreshing the page or restarting the application.
2. **File Upload Errors:** Ensure CSV files uploaded for batch prediction follow the specified format and contain required columns.

Project Conclusion

In conclusion, the Fraud Detection System project not only addresses current security challenges in online transactions but also sets the stage for ongoing innovation and improvement in AI-driven fraud prevention technologies. The project has achieved significant milestones in enhancing online transaction security through advanced machine learning techniques.
Key outcomes include:

1. **Real-time Fraud Detection:** Implemented a robust system capable of analyzing transactions in real-time, providing immediate feedback on potential fraudulent activities.

2. **User-Friendly Interface:** Designed an intuitive Streamlit-based interface that simplifies the process of inputting transaction data and interpreting results.
3. **Historical Analysis:** Incorporated a feature to maintain a detailed history of predictions, enabling users to review past detections and improve future fraud prevention strategies.
4. **Model Performance:** Deployed a Random Forest model with high accuracy, precision, recall, and F1 score metrics, ensuring reliable fraud predictions.

Lessons Learned

1. **Data Quality Importance:** Ensuring high-quality input data is crucial for accurate predictions. Cleaning and preprocessing data effectively significantly impact model performance.
2. **User Experience:** Balancing functionality with user interface design is critical. Prioritizing user feedback and iterative improvements helped refine the application's usability.
3. **Deployment Challenges:** Overcoming deployment complexities, such as managing session states and handling concurrent user interactions, proved essential for a smooth user experience.

Areas for Improvement

- **Model Updates:** Implementing mechanisms for periodic model retraining and updates based on new data could improve prediction accuracy over time.
- **Performance Optimization:** Continuously optimizing the application's performance to handle larger datasets and increased user traffic more efficiently.

- **Datasets:** For a large application, a large sized data must be trained for very accurate result.

Appendices:

Appendix A:

Single Transaction Prediction code:

```python
def predict_transaction(type, amount, oldBalOrg, newBalOrg):
    data = [
        [type_dict[type], amount, oldBalOrg, newBalOrg]
    ]
    columns = [
        "type",
        "amount",
        "oldbalanceOrg",
        "newbalanceOrig",
    ]
    df = pd.DataFrame(data, columns=columns)
    df[columns] = scaler.transform(df)
    prediction = model.predict(df)
    return prediction

def predict_one():
    if "oldBalanceOrg" not in st.session_state:
        st.session_state.oldBalanceOrg = 0.0
    if "newBalanceOrig" not in st.session_state:
        st.session_state.newBalanceOrig = 0.0
    if "amount" not in st.session_state:
        st.session_state.amount = 0.0
    if "prediction_message" not in st.session_state:
        st.session_state.prediction_message = ""
    # this function returns the prediction on a single transaction
    st.write("Here you can check if a transaction is fraud or not.")
    st.header("📈Single Set Prediction")

    if st.button("Clear inputs"):
        st.session_state.update({"oldBalanceOrg": 0.0, "newBalanceOrig": 0.0,
"amount": 0.0,"type": "Select", "prediction_message": ""})
        st.experimental_rerun()

    type = st.selectbox('Transaction Type', ["Select","PAYMENT", "TRANSFER",
"CASH OUT", "DEBIT", "CASH IN"], key='type')
    amount = st.number_input(
        "Amount", min_value=0.0, key="amount", value=st.session_state.amount
    )
    oldBalanceOrg = st.number_input(
            "Old balance at origin",
            min_value=0.0,
            key="oldBalanceOrg",
            value=st.session_state.oldBalanceOrg,
    )
    newBalanceOrig = st.number_input(
            "New balance at origin",
            min_value=0.0,
            key="newBalanceOrig",
            value=st.session_state.newBalanceOrig,
    )

    if st.button("Predict"):
        if type=="Select" and amount == 0.0 and oldBalanceOrg == 0.0 and
newBalanceOrig == 0.0:
            st.warning("Please fill all the required fields.")
        else:
            isFraud = predict_transaction(type, amount, oldBalanceOrg,
newBalanceOrig)
```

```
            # Store result in session state
            st.session_state.prediction_result = isFraud
            st.session_state.prediction_history.append({
                "Transaction Type": type,
                "Amount": amount,
                "Old Balance Origin": oldBalanceOrg,
                "New Balance Origin": newBalanceOrig,
                "Prediction": "Fraudulent" if isFraud == 1 else "Non-Fraudulent",
                "Transaction ID": "",
                "Account ID": ""
            })

        # Display prediction result and report section
    if st.session_state.prediction_result is not None:
        if st.session_state.prediction_result == 1:
            st.error("Prediction: " + "⚠️ The transaction is predicted to be
fraudulent.")
            report_expander = st.expander("Report Fraudulent Transaction")
            with report_expander:
                transaction_id = st.text_input("Enter Transaction ID", "")
                account_id = st.text_input("Enter Account ID", "")
                if transaction_id and account_id:
                    confirm_report = st.button("Confirm Report")
                    if confirm_report:
                        st.session_state.reported_transactions.append({
                            "Transaction ID": transaction_id,
                            "Account ID": account_id,
                            "Transaction Type": type,
                            "Amount": amount,
                            "Old Balance Origin": oldBalanceOrg,
                            "New Balance Origin": newBalanceOrig,
                            "Prediction": "Fraudulent"
                        })
                        st.success(f"Transaction {transaction_id} reported.")
                        # Update the prediction history with the reported
transaction details
                        st.session_state.prediction_history[-1]["Transaction ID"]
= transaction_id
                        st.session_state.prediction_history[-1]["Account ID"] =
account_id
        else:
            st.success("Prediction: " + "✅ The transaction is predicted to be
non-fraudulent.")
```

Batch Transaction Prediction Code:

```
def batch_predict():
    # valid attributes for the model and scaler
    columns = ['type', 'amount', 'oldbalanceOrg', 'newbalanceOrig']
    # mapping prediction
    output = {0: 'Not Fraud', 1: 'Fraud'}

    # this function runs predicition on every record in the dataset
    def predict_batch(data):
        return model.predict(data)

    # streamlit UI
    header = st.container()

    with header:
        st.title("📊Batch File Prediction")
        st.write("Here you can check multiple transactions from a single CSV
file.")

    with st.expander("Upload a .csv file to predict transactions",
expanded=True):
        uploaded_file = st.file_uploader("Choose file:", type="csv")

        if uploaded_file:
            df = pd.read_csv(uploaded_file)
            st.dataframe(df.head())
            st.write(f"There are {df.shape[0]} records and {df.shape[1]}
attributes.")
            if set(df.columns) != set(columns):
                st.error(f"File columns do not match the expected columns.
Expected columns: {columns}.")

        if st.button("Predict file"):
            if uploaded_file:
                new_df = df[columns]
                new_df['type'] = df['type'].replace(type_dict)
                new_df[columns] = scaler.transform(new_df)
                predictions = predict_batch(new_df)
                df['isFraud'] = predictions
                df['isFraud'] = df['isFraud'].replace(output)
                counts = df['isFraud'].value_counts()
                st.success(f"File has {counts[0]} legitimate transactions ✅ and
{counts[1]} fraud transactions ⚠️.")
```

```
                st.download_button(label="Download as CSV",
data=df.to_csv(index=False), file_name='predicted_transactions.csv',
mime='text/csv')
            else:
                st.warning("Please upload the file")
def predict_history():
```

Appendix B:

Initial dataset source:
https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection