Financial Document Fraud Analyzer Documentation

---

1. User Documentation

1.1. Introduction

The Financial Document Fraud Analyzer is a comprehensive, end-to-end application built to analyze and detect potential fraudulent activity in various types of financial documents. These documents include invoices, receipts, bank statements, and loan applications. The system combines multiple state-of-the-art technologies: Optical Character Recognition (OCR), machine learning models, rule-based heuristics, explainability modules (e.g., SHAP), and natural language summarization. It offers a friendly user interface via a Streamlit web app and alerting via Slack notifications.

This tool is particularly useful for compliance teams, finance auditors, business analysts, and data scientists looking for an automated, scalable fraud detection system that supports both individual document and batch analysis.

1.2. Installation

1. **Clone the Repository**

   ```
   git clone https://github.com/Shivasairam1706/Financial-fraud-analyzer.git
   cd Financial-fraud-analyzer
   ```

2. **Set Up Virtual Environment**

   ```
   python3 -m venv venv
   source venv/bin/activate    # On Windows: venv\Scripts\activate
   ```

3. **Install Dependencies**

   ```
   pip install -r requirements.txt
   ```

   This will install all required Python packages including OCR libraries, machine learning frameworks, Streamlit, and testing tools.

4. **Configuration**

   - Copy `config.py.template` to `config.py` if it exists.
   - Update API keys for external services (e.g., GPT-4, Slack).
   - Configure thresholds, file paths, and runtime settings in `config.py`.

1.3. Quick Start

1. **Generate Synthetic Data (Optional)**
   Useful for testing purposes and model validation.

```
python synthetic_data_generator.py
```

This will create artificial financial documents with controllable noise and fraud patterns.

2. **Launch Streamlit Dashboard**

```
streamlit run streamlit_app.py
```

   - Navigate to `http://localhost:8501` in your browser.
   - Ensure firewall rules allow access to the local port.

3. **Single Document Analysis**
   - Upload a PDF or image of a financial document.
   - The system extracts key fields and calculates fraud scores.
   - Visual explanations (e.g., SHAP plots) help understand the decision.
4. **Batch Document Analysis**
   - Upload a ZIP archive of documents.
   - Analyze them in one go.
   - Download a consolidated fraud analysis report (CSV or JSON).
5. **Slack Alerts**
   - Set the `SLACK_WEBHOOK_URL` in your `config.py`.
   - Fraud alerts will automatically notify your team in the specified Slack channel.

1.4. Usage Commands

These commands can be executed via the terminal for CLI-based workflows.

- **Parse a Single Document:**

```
python document_parser.py --input path/to/doc.pdf --output parsed.json
```

- **Run Fraud Detection on Parsed Data:**

```
python fraud_detector.py --input parsed.json --model
models/fraud_model.pkl --output results.json
```

- **Generate Explainable Report:**

```
python explainer.py --input results.json --output report.html
```

This outputs an HTML-based report summarizing why a document is considered fraudulent or not.

1.5. Troubleshooting & FAQs

- **OCR Errors:**
   - Ensure that the document is not skewed or too blurry.

    - Try using a different OCR engine (e.g., Tesseract or Google Vision).
- **Missing or Broken Dependencies:**
    - Double-check your Python environment.
    - Run `pip install -r requirements.txt` again.
- **Slack Alerts Not Delivered:**
    - Ensure internet access is available.
    - Check that your Slack webhook is still active.
    - Validate the webhook URL in `config.py`.
- **UI Not Loading:**
    - Confirm Streamlit is installed.
    - Check port conflicts or firewall settings.
- **Low Model Performance:**
    - Train the model with your specific document types for better accuracy.

---

2. Developer Documentation

2.1. Project Architecture

```
flowchart TD
    A[Upload Document] --> B[DocumentParser]
    B --> C[FeatureEngineering]
    C --> D{FraudDetector}
    D -->|IsolationForest| E[AnomalyScores]
    D -->|RuleBased| F[RuleScores]
    E & F --> G[Decision]
    G --> H[Explainer]
    H --> I[ReportGenerator]
    I --> J[StreamlitUI]
    G --> K[SlackNotifier]
```

This architecture diagram summarizes the data flow from document ingestion to fraud notification. The modular design allows for easy customization and scaling.

2.2. Module Overview

- **synthetic_data_generator.py**:
    - Generates fake invoice/financial records for benchmarking.
    - Customizable parameters include noise level, field variability, and fraud ratio.
- **document_parser.py**:
    - Handles OCR using EasyOCR.

    - Applies regex and NLP techniques to extract fields like invoice ID, vendor name, amounts, and dates.
- **fraud_detector.py**:
    - Contains both ML-based and rule-based detectors.
    - Supports IsolationForest and custom heuristic rules.
    - Output includes anomaly scores and rule violations.
- **explainer.py**:
    - Uses SHAP, LIME, and GPT-4 for explanation.
    - Supports both local and global model explainability.
- **streamlit_app.py**:
    - Renders a clean and interactive dashboard.
    - Tabs include: Single File Upload, Batch Analysis, and Visual Summaries.
- **slack_bot.py**:
    - Pushes alerts with fraud scores and summaries to Slack channels.
- **auto_responder.py**:
    - Optional Slack integration to listen for user actions and trigger further workflows.
- **config.py**:
    - Single source for global settings like API keys, detection thresholds, and model paths.
- **utils.py**:
    - Utility functions for file I/O, data formatting, logging, and common operations.

2.3. Development Setup

1. **Branching Strategy**
    - `main` is the stable, production-ready branch.
    - Feature branches follow naming convention: `feature/<feature_name>`.
    - Pull requests must include passing unit and integration tests.
2. **Testing Suite**
    - Unit tests available in the `tests/` directory.
    - Use `pytest` to execute tests:

```
pytest --maxfail=1 --disable-warnings -q
```

    - Includes tests for parsing, model prediction, and integration flows.
3. **Coding Standards**

- PEP8 compliance.
- Use type annotations and Google-style docstrings.
- Modular functions and logging via `logging.getLogger(__name__)`.

2.4. Extending the System

- **Add New Fraud Rules**:
    1. Create a new class inheriting from `RuleBasedDetector`.
    2. Define the `apply_rule()` method.
    3. Register the rule in the detection factory.
- **Support More Document Types**:
    1. Extend parsing logic in `document_parser.py`.
    2. Add annotated samples in `data/sample_documents/`.
- **Custom Model Training**:
    1. Use your labeled dataset.
    2. Train a new IsolationForest or XGBoost model.
    3. Save as `.pkl` and update model path in `config.py`.
- **Deployment via Docker**:

```
docker build -t fraud-analyzer ./docker
docker run -p 8501:8501 fraud-analyzer
```

    1. Customize Dockerfile for environment-specific dependencies.
- **CI/CD Integration**:
    1. Use GitHub Actions or Jenkins for automated testing and deployment.
    2. Include linter and test steps in your pipeline.

2.5. Contribution Guidelines

1. Fork this repository into your GitHub account.
2. Clone your fork and create a feature branch.
3. Make changes with descriptive commits.
4. Write appropriate tests and ensure they pass.
5. Open a Pull Request with clear description and context.
6. Tag relevant maintainers for review.

All contributors must follow the code of conduct and maintain professional collaboration ethics.

End of Documentation