AI/ML Developer Assignment - ZenStatement

Objective:

You are tasked with building an intelligent agent framework that can process financial data from multiple sources and make decisions based on predefined categories and resolutions. The goal is to categorize financial discrepancies, handle them appropriately, and automate actions such as uploading files or sending emails based on specific use cases.

Assignment Overview:

Problem Statement:

ZenStatement's platform ingests financial data from multiple sources (such as SFTP, APIs, and manual uploads) and performs data reconciliation. Your task is to build a simple agentic framework that will:

- Understand the financial discrepancies from the uploaded file and categorize it
- It will pick up only one category, and upload the respective data into a certain path
- Once the data is uploaded, there will be a resolution which the agent/assistant needs to understand and act accordingly

For this, use any publicly available LLM + Assistant combination to prompt tune and consolidate the information.

Data:

You will be provided with a synthetic dataset containing transaction records from various sources. The dataset includes:

- **Attributes:** Order ID, amount, date, payment method, status, and others.
- **Statuses:** Processed data with various sub-statuses, including a focus on "Not Found Sys B."
- **Comments:** An additional column in the second dataset with plain English responses regarding the resolution of issues for specific order IDs.

Requirements:

1. **Preprocessing and Categorization:**
    a. Preprocess the data, handle any missing or invalid data, and categorize the records according to sub-statuses.
    b. Specifically, focus on the **"Not Found Sys B"** category and prepare a CSV file containing **order_id**, **amount**, and **date**.
2. **File Upload:**
    a. Once you have the categorized data, upload it to a specified folder or use an email for the upload mechanism (whichever is more feasible for you).
3. **Resolution Handling:**
    a. After the upload, process a second dataset (from another folder), which will contain **comments** for each order ID. The comments provide details on the resolution status.
    b. For each order ID in this dataset, perform the following:
        i. **Use Case 1:** If the case is resolved, upload the data to a third folder or send it via email.
        ii. **Use Case 2:** If the case is unresolved, generate a summary explaining why it is unresolved.
        iii. **Use Case 3:** For unresolved cases, suggest **next steps** to resolve the issue.
        iv. **Use Case 4:** Lastly, for the resolved cases, identify the pattern, so that it can be closed internally if possible, without raising it to support

Integration:

- The solution should be **deployable** in an environment that supports Python (e.g., AWS Lambda, Azure, or similar cloud platforms).
- Ensure the framework can handle different types of financial data formats and integrate smoothly into ZenStatement's existing infrastructure.

Submission Instructions:

- **Code and Documentation:** Submit a GitHub repository (or other version control system of your choice) with the following:
    - Code: All code should be well-organized and documented. Include clear instructions for how to set up the environment and run the models.
    - README: The README file should describe:
        - The problem and your approach.
        - How to set up the environment and run the code.
        - Explanation of the model choice and any preprocessing steps.
        - How to evaluate the model.
- **Environment Setup:**
    - Specify any libraries, frameworks, or specific configurations needed to run your solution.
    - If possible, provide Docker support for containerizing the environment.
- **Testing Setup:**
    - Include a brief explanation of how you've tested the solution and how we can test it on our side.

Testing and Evaluation:

We will evaluate your submission based on the following criteria:

- Accuracy of the Solution: We will test the model against a set of validation data not included in the original dataset.
- Model Performance: Does your model show consistent results across different test cases and edge cases?
- Code Quality and Readability:
    - How well-structured and maintainable is the code?
    - Is the code properly documented and easy to follow?
- Scalability:
    - How well can your solution handle larger datasets (we may test this with a larger dataset that is similar to the one you have used)?
    - Does the solution integrate easily into an existing cloud-based infrastructure (AWS, Azure, etc.)?
- Explainability and Transparency:
    - Are the decisions made by your model explainable?
- Creativity and Approach:
    - While we have outlined the basic requirements, we encourage creativity. If you have additional ideas or approaches that can improve the model's performance or usability, feel free to include them.