Task Overview

To evaluate your expertise in Python programming, natural language processing, and your ability to simulate integration with Large Language Models (LLMs), we have prepared a task for you. This task should take approximately **one hour** to complete. While you are free to utilize code generation tools or any Python libraries, the task requires thoughtful implementation that cannot be fully automated.

Task Description

Your task is to create a Python script that processes a sample regulatory text file and extracts key business requirements from it. The script should simulate integration with an LLM to summarize each section of the regulatory text. Due to time and resource constraints, you will mock the LLM responses.

Specific Requirements

1. **Python Script:**
    - **Input Processing:**
        - Read a text file (`regulations.txt`) containing sample regulatory text. You can create this file with at least three sections or paragraphs of regulatory content.
    - **Text Segmentation:**
        - Split the text into individual sections or paragraphs for processing.
    - **Simulated LLM Integration:**
        - Implement a function `simulate_llm_summary(text_section)` that simulates an LLM summarizing a section of text.
        - Since actual LLM integration may require API keys and internet access, mock this function to return a generated summary. For example, you can reverse the words, select key sentences, or use any simple text manipulation to simulate a summary.
    - **Requirement Extraction:**
        - For each section, use the `simulate_llm_summary` function to obtain a summary of key business requirements.
    - **Output Generation:**
        - Store the original text and its corresponding summary in a structured format (e.g., JSON or CSV file).
        - The output file should include:
            - Section number
            - Original text
            - Summarized requirements
            - Any other relevant metadata (optional)
2. **Code Organization and Documentation:**
    - **Modular Code:**
        - Organize your code using functions and, if appropriate, classes to promote readability and reusability.
    - **Documentation:**

        - Include docstrings for all functions and classes.
        - Comment your code where necessary to explain non-obvious parts.
    - **Error Handling:**
        - Implement basic error handling to manage exceptions that may occur during file I/O or data processing.

3. **Deliverables:**

    - **Python Script (`extract_requirements.py`):**
        - The main script containing your implementation.
    - **Sample Input File (`regulations.txt`):**
        - The sample regulatory text file you used for the task.
    - **Output File (`extracted_requirements.json` or `extracted_requirements.csv`):**
        - The file containing the extracted summaries.
    - **README.md:**
        - Instructions on how to run your script.
        - Any assumptions or design decisions you made.
        - Explanation of how you simulated the LLM integration.
        - Dependencies and how to install them.

Constraints

- **Python Version:**
    - Use Python 3.6 or higher.
- **Libraries:**
    - You may use standard Python libraries and open-source packages available via `pip`.
    - Do not use any external APIs that require an API key or internet access.
- **Execution:**
    - The script should be executable from the command line.
    - Ensure that all dependencies can be installed easily (preferably via a `requirements.txt` file).

Submission Instructions

- **Code Submission:**

    - Submit your code as a public Git repository (e.g., GitHub, GitLab).
    - Ensure the repository contains all the deliverables specified above.

- **Repository Structure:**

    - Organize your repository for clarity. For example:

```
├── README.md
├── extract_requirements.py
├── regulations.txt
├── extracted_requirements.json
├── requirements.txt
```

- **Documentation:**
    - Make sure your README file provides clear instructions on how to set up and run your script.
- **Optional Enhancements:**
    - If you have extra time, you may implement additional features such as:
        - More sophisticated text summarization in the `simulate_llm_summary` function.
        - Unit tests for your functions.
        - Use of NLP libraries like NLTK or spaCy for text processing.

Evaluation Criteria

- **Code Quality:**
    - Readability, organization, and adherence to Python best practices.
- **Functionality:**
    - Correctness of the script in processing the input file and generating the expected output.
- **Understanding of Python:**
    - Effective use of language features and standard libraries.
- **Documentation:**
    - Clarity of the README and usefulness of code comments and docstrings.
- **Problem-Solving Approach:**
    - Thoughtfulness in simulating the LLM integration and handling of the task requirements.

Note

Before you begin, if you have any questions or need clarifications, feel free to reach out. Good luck, and we look forward to reviewing your work!