Veryfi technical Test: Invoice Data Extraction Script

Prepared by: Andrea P. Gomez
Application: Data Annotations Engineer
Date: April 15, 2025

Context and Use Case

In an industrial document processing pipeline, raw text extracted from invoices via Optical Character Recognition (OCR) often needs to be converted into structured metadata and tables. This script focuses on processing a specific invoice template from **Switch Ltd.** and is intended to be part of a modular system that extracts relevant information for automated financial workflows.

Code Paradigm

The script follows a **procedural programming** paradigm using standard Python libraries such as re, os and json. Most importantly, it uses Veryfi's Python API to retrieve the Client's information in an OCR text format. It is structured into discrete steps that reflect a logical flow from input preprocessing to data cleaning. Functions are modular and easily extendable to accommodate other invoice formats.

Pipeline Steps

1. **Text Cleaning**
    a. Removes excessive whitespace, line breaks, and empty lines.
    b. Filters out irrelevant or redundant text using simple rules.
2. **Header Extraction**
    a. Uses regular expressions to extract key fields such as:
        i. Vendor name
        ii. Invoice number
        iii. Invoice date
        iv. Billing and shipping addresses
3. **Table Isolation**
    a. Identifies the table section of the invoice using keyword anchors such as `"QUANTITY"`, `"UNIT PRICE"`, and `"AMOUNT"`.
    b. Extracts lines between start and end markers.
4. **Line Parsing**
    a. Tokenizes each row.
    b. Applies logic to correct misaligned columns.
    c. Filters out headers and footers that may repeat or interfere.
5. **Data Formatting**
    a. Converts extracted values into appropriate types (`float`, `str`, etc.).
    b. Loads clean data into a `json` file for downstream use.

Unit Tests

1. **Component-Level Testing on a Representative File**
   Each section of the extraction pipeline was tested individually to ensure robust performance across multiple documents. Specifically, the fields (`vendor name, vendor address,bill to name, invoice number, date, sku, description, quantity, tax rate, price, total`) These tests confirmed that each regular expression accurately extracted the correct field from a known-good file.
2. **Batch Testing Across All Invoices**
   The script was then tested in a loop across multiple invoices from the same source ("Switch"), ensuring that the regex patterns consistently extracted all fields from the entire dataset. When inconsistencies were found, the relevant expressions were analyzed and corrected to generalize across all valid cases.
3. **Negative Testing with Other Document Formats**
   To confirm format specificity, the script was run on several documents **not** matching the "Switch" invoice structure. The script correctly ignored or failed gracefully on these documents, demonstrating that it is robustly tailored to only parse files of the expected format.

Assumptions

- The input strictly follows the **Switch Ltd.** invoice layout, with consistent structure and labeling across documents.
- All invoices come from the same vendor, **Switch Ltd.**, and no attempt is made to generalize the approach beyond this format.
- The `"amount"` field in the line items is assumed to represent the **price** of the product (i.e., unit price).
- The **total** listed for each line item is considered the final **total of the invoice** for summation validation.
- All table data is aggregated into a single dictionary per invoice.
- The `sku` is assigned based on the row index of the item in the table to provide unique identifiers within the document context.
- Any documents that do not match the Switch invoice format are explicitly excluded to ensure structural consistency and accurate parsing.

Coding Best Practices

The script was written following strong engineering principles to maximize **readability**, **maintainability**, and **scalability** for industry applications:

- **Modular Function Design**
  Each operation is encapsulated in distinct functions to support debugging, unit testing, and future extensions.

- **Clear Variable Naming**
  Names are intentionally descriptive and consistent, adhering to Python conventions for clarity and professional development standards.
- **Use of Built-in Libraries**
  Only essential and widely supported libraries (`re`, `json`, `os`) are used to ensure high portability and minimal environment setup.
- **Inline Comments & Docstrings**
  Each function includes docstrings detailing inputs, outputs, and functionality. Inline comments provide additional explanations for key lines of logic.
- **Thorough Annotation**
  The code is thoroughly annotated to ensure other developers — including collaborators unfamiliar with the script — can clearly follow and understand the structure, flow, and logic of each component.
- **JSON Output for Structured Storage**
  All extracted data is stored as Python dictionaries and serialized into JSON files. This structure facilitates straightforward integration with databases, APIs, or downstream processing tools.
- **Error Tolerance and Validation Checks**
  The code incorporates validation steps and exception handling to catch and manage irregularities in input formats, missing fields, or corrupted data.
- **Regex Pattern Isolation**
  All regular expressions are modular and labeled according to the specific information they extract, making it easy to update them if the invoice format evolves.
- **File-Agnostic Parsing Logic**
  The code dynamically processes different files based on folder scanning and generalizes all logic to work on any file with the correct structure — supporting scalability and automation.

Limitations

- Parsing is tightly coupled to the structure of Switch Ltd. invoices and is not compatible with alternative formats.
- The approach assumes clean and correctly aligned table structures; irregularities may break parsing.
- Logging and exception handling mechanisms are limited in the current version.

Conclusion and Recommendations

This script demonstrates an effective and modular approach to structured data extraction from OCR-processed invoices in a production-like environment. It is well-suited for integration into downstream automation systems requiring structured metadata and financial data. Future iterations could benefit from:

- Expanding support to multiple vendor formats using layout detection, fuzzy matching or machine learning.
- Improved error handling and debugging support.
- Configurability via templates or schema definitions to increase adaptabilit