Description of my approach in the Data Annotations Engineer Technical test

Candidate: Juan Pablo Montoya Ortega

February 9, 2025

Contents

1 Introduction

This document explains the design and implementation of an ETL (Extract-Transform-Load) pipeline for invoice data extraction. The solution uses the Veryfi API to process PDF documents, extract OCR text, transform raw text into structured JSON data, and finally load the results into designated folders.

2 Overall Approach and Code Paradigm

The solution follows a classic ETL approach:

a) **Extract:** This phase is handled by the `extract.py` module, which communicates with the Veryfi API. The key function `extract_veryfi_json` accepts a PDF file path and a pre-configured API client to retrieve the raw JSON response.

b) **Transform:** The transformation phase is the most elaborate part. The `transform.py` module is responsible for:

- Cleaning the OCR text to remove unwanted characters and headers/footers.
- Using regular expressions to locate and extract important data fields (such as vendor details, invoice dates, and line items).
- Reconstructing tabular data using Python's string manipulation and Pandas DataFrames.

c) **Load:** The `load.py` module writes the output data as JSON files into specified directories. It saves both the raw JSON from Veryfi and the processed, transformed invoice data.

3 Module Details and Code Nuances

3.1 Extraction Module (`extract.py`)

The extraction module is straightforward yet crucial. The function:

```
def extract_veryfi_json(file_path, client):
    """
    Processes the document using the Veryfi API and returns the
   resulting JSON.
    """
    json_result = client.process_document(file_path)
    return json_result
```

is responsible for sending the PDF to the Veryfi API. By isolating this logic, we can easily update the API interaction in one place if needed.

3.2 Transformation Module (`transform.py`)

The transformation module is the core of the pipeline. It contains multiple functions that work together to clean and parse the OCR text.

3.2.1 Cleaning the OCR Text

The function `clean_ocr_text` performs several operations:

- It replaces form feed characters (`\f`) with newline characters (`\n`):

```
text = text.replace("\f", "\n")

```

- It checks if the text contains the expected header, such as:

```
if not re.search(r'Description\s+Quantity\s+Rate\s+Amount', text,
    flags=re.IGNORECASE):

```

- It removes unwanted header or footer lines using:

```
text = re.sub(r'^\s*(Invoice|Page\s+\d+\s+of\s+\d+).*$', '', text,
    flags=re.MULTILINE)

```

  This regular expression matches lines that start with "Invoice" or "Page X of Y", and removes them to ensure that only the essential data remains.

3.2.2 Extracting Key Invoice Fields

The function `extract_invoice_data` is responsible for extracting important details from the OCR text:

- **Vendor Name:** It uses the following regex to extract the vendor information:

```
vendor_match = re.search(r'Please make payments to:\s*(.+)',
    ocr_text)

```

- **Vendor Address:** The address is extracted with:

```
vendor_address_match = re.search(r"\n([\w\s,]+ \d{5}(?:-\d{4})?)\
    nPO Box", ocr_text)

```

  This pattern captures a sequence of word characters, spaces, or commas followed by a five-digit ZIP code (with an optional four-digit extension) before encountering the string "PO Box".

- **Invoice Number and Dates:** A more complex regular expression is used:

```
invoice_match = re.search(
    r'Invoice Date Due Date\tInvoice No\.\n\t(\d{2}/\d{2}/\d{2})\t
    (\d{2}/\d{2}/\d{2})\t([\d]+)\n\n([^\n]+)',
    ocr_text
)

```

  This regex carefully matches the format of dates and invoice number in the OCR text.

3.2.3 Processing Tabular Data

After cleaning the text, the module identifies the section containing itemized line data. This is done by:

- Locating the start of the table:

```
start_pattern = r"^\s*Description\s+Quantity\s+Rate\s+Amount\s*$"

```

- Locating the end of the table:

```
end_pattern = r"^\tTotal\s+USD"

```

The code then slices the OCR text between these two patterns and processes each line by splitting on the tab character (`"\t"`). It uses Pandas to convert the list of rows into a DataFrame for easier manipulation and type conversion (e.g., converting strings to floats for numerical values).

Additionally, functions like `is_valid_row` and `clean_description` further ensure that the extracted rows are valid. For example:

```
if re.fullmatch(r'w-\d+', desc):
    return False
```

This line filters out any row that consists solely of a pattern like "w-300387", which is not useful data.

3.3 Loading Module (`load.py`)

The loading module is responsible for writing the JSON data to disk. It includes:

- A generic `save_json` function that accepts the data and a file path, writing the JSON with proper formatting.
- Two specialized functions: `save_veryfi_json` and `save_processed_json`, which determine the correct output directory and filename before calling `save_json`.

4 Assumptions and Considerations

The pipeline was built under several assumptions:

- **Input Documents:** All PDF files are located in a directory named `Documents` at the root of the project.
- **Consistent OCR Format:** The OCR text provided by the Veryfi API is assumed to follow a predictable format with clearly defined sections such as the header (`Description Quantity Rate Amount`) and footer (e.g., `Total USD`).
- **External Configuration:** API credentials and other configuration parameters are stored in a separate JSON file (`src/static/credentials.json`) to avoid hardcoding sensitive information in the code.
- **Error Handling:** If the OCR text does not conform to the expected format (for example, if the necessary headers are missing), the program raises a `ValueError` and skips processing that document.

5 Coding Best Practices

In developing this solution, I have applied several best practices:

- **Modularity:** Each phase of the ETL pipeline is separated into its own module (`extract.py`, `transform.py`, and `load.py`), which makes the code easier to understand, maintain, and extend.
- **Clear and Descriptive Naming:** Functions and variables are named clearly to reflect their purpose. For example, `clean_ocr_text` and `extract_invoice_data` provide immediate insight into what they do.
- **Robust Regular Expressions:** The use of regular expressions is central to the transformation phase. Detailed patterns, such as

```
r'^\s*(Invoice|Page\s+\d+\s+of\s+\d+).*$'
```

  allow for flexible and powerful matching of unwanted text segments.
- **Separation of Configuration:** Sensitive information such as API credentials is stored in an external file, reducing the risk of accidental exposure.
- **Error Handling and Logging:** The code gracefully handles errors (e.g., by raising exceptions when the OCR format is invalid) and provides informative print statements to help trace the processing flow.
- **Use of External Libraries:** Utilizing well-established libraries like `pandas` for data manipulation and Python's built-in `re` module for regular expressions contributes to code reliability and performance.

6 Unit Testing Overview

The unit tests for the invoice transformation process ensure that the conversion from OCR data to the final JSON output is both accurate and consistent with the original Veryfi data. These tests focus on several key aspects:

- **Field Consistency:** The tests verify that critical fields such as the invoice number, client name (bill-to), and vendor name remain identical between the processed JSON and the original Veryfi JSON.
- **Date Normalization:** Given the differences in date formats—one as MM/DD/YY and the other as YYYY-MM-DD HH:MM:SS—the tests convert both into comparable date objects to ensure that the dates match.
- **Line Item Verification:** They check that the number of line items extracted from the OCR matches the number of items in the Veryfi output. Additionally, the tests compare the total amounts (with a tolerance for minor rounding differences) to ensure that no data is lost or erroneously altered during transformation.
- **Robustness and Error Reporting:** Any discrepancies, such as missing files or parsing errors, are flagged with clear error messages. This facilitates rapid debugging and enhances the overall reliability of the ETL process.

By automating these validations with Python's `unittest` framework, the testing suite offers continuous assurance that the transformation logic works as intended—even as the data formats or processing algorithms evolve. This rigorous testing is essential for maintaining the integrity and reliability of the invoice processing system.

7 Conclusion

To summarize, I have designed this ETL pipeline with careful attention to reliably extract, transform, and load invoice data. I focused on organizing the code into clear, distinct modules to improve maintainability and scalability, and I incorporated detailed regular expressions and robust error handling to ensure that only high-quality data is processed. I hope this approach, based on established coding best practices, serves as a solid foundation for future development and potential system integrations. I am eager to receive feedback and continue learning as I work on further improvements.