Technical Document on Financial Data Extraction Pipeline

This document details the technical design, implementation, and evaluation of a financial data extraction pipeline developed for scraping quarterly report PDFs. The pipeline employs modular agents, regular expressions, prompt engineering, and integration with OpenAI's API to extract key financial figures from income statements.

---

1. Overview

The objective of the pipeline is to extract financial metrics exclusively from the most recent 3-month period in Group or Consolidated Income Statements. The system is designed to:

- Scrape PDF documents to extract text.
- Identify candidate sections that contain income statements.
- Prioritize specific statement types (Group or Consolidated) using rule-based keyword matching.
- Use a language model to extract structured financial data.
- Aggregate results into a consolidated CSV output for further analysis.

---

2. Architecture & Components

2.1. Modular Agent Design

The pipeline is built using a modular agent architecture with the following components:

- **PDFScraperAgent:**
  Uses `pdfminer.high_level.extract_text` to extract text content from PDF files.
- **CompanyNameExtractionAgent:**
  Extracts the company name from the first page using a regex pattern that targets names

ending with "PLC".

- **SectionSelectorAgent:**
  Splits the PDF text into pages, then selects candidate pages based on targeted keywords such as "Consolidated Income Statement" or "Statement of Profit or Loss".
  - **Technique:** Embedding-based similarity is computed using OpenAI embeddings and cosine similarity to rank candidate sections.
  - **Regex Enhancements:** Uses a regular expression that captures both “3 months ended …” and “03 months to …” patterns.
- **DataExtractionAgent:**
  Uses prompt engineering via a Jinja template to instruct OpenAI’s API (e.g., GPT-3.5-turbo) to extract specific financial metrics.
  - **Prompt Guidance:** The prompt explicitly instructs the model to extract only the latest 3-month period, with real examples of valid vs. invalid headings.
  - **Output Format:** The expected JSON output is defined, including keys like `CompanyName`, `Revenue`, `COGS`, `GrossProfit`, and `PeriodStartEnd`.
- **DataAggregatorAgent:**
  Aggregates the extracted data from multiple PDFs into a pandas DataFrame, then outputs the final results to a CSV file.

2.2. Pipeline Flow

1. **Environment and Configuration Setup:**
   Environment variables are loaded and configuration is read from a YAML file.
2. **File Identification:**
   The system scans a designated folder for PDF files.
3. **Text Extraction:**
   The `PDFScraperAgent` processes each PDF to extract text.
4. **Section and Data Extraction:**
   - The `SectionSelectorAgent` isolates candidate sections by filtering pages based on keywords and regex checks for 3-month column patterns.

    - The `DataExtractionAgent` uses a detailed prompt (via Jinja template) to instruct the language model to extract data from the most recent 3-month column only.

5. **Data Aggregation:**
   The `DataAggregatorAgent` consolidates all results into a CSV file for downstream analysis.

---

3. Techniques and Approaches

3.1. Regular Expressions and Keyword Filtering

- **Keyword Matching:**
  The system searches for key phrases like "Consolidated Income Statement" and "Statement of Profit or Loss" to prioritize sections.

- **Regex Patterns:**

    - The pattern `(?:\b0?3\s*months\s*(?:ended|to)\s*\d{1,2}\s*\w+\s*\d{4})` is used to detect valid 3-month headings.

    - This pattern captures variations like “3 months ended …”, “03 months ended …”, “3 months to …”, etc.

3.2. Embedding-Based Similarity

- **Embedding Generation:**
  Uses OpenAI embeddings to generate vector representations for each page of the PDF.

- **Cosine Similarity:**
  Pages are ranked based on the similarity between the query (constructed from the target keywords) and each page’s embedding.

3.3. Prompt Engineering via Jinja Templates

- **Prompt Structure:**
  The prompt includes explicit instructions, examples of valid headings (e.g., "3 months

ended 31 Dec 2024"), and an example JSON structure.

- **Guidance for Data Extraction:**
  The prompt directs the model to ignore 6, 9, or 12-month columns and ambiguous lines, ensuring that only the latest 3-month data is processed.

- **Separation of Concerns:**
  By using a Jinja template, prompt modifications can be made without altering code logic.

3.4. Error Handling and Rate-Limiting

- **Retry Logic:**
  The DataExtractionAgent implements a retry mechanism for handling API rate limits, waiting for a calculated time before retrying the request.

- **Logging:**
  Each step logs actions and errors to facilitate debugging and performance monitoring.

---

4. Approach and Assumptions

- **Assumptions:**
  - The PDF's first page typically contains the company name, ending with "PLC."
  - The targeted income statements are clearly labeled with keywords such as "Consolidated Income Statement" or "Statement of Profit or Loss."
  - The valid 3-month period columns include headings like "3 months ended …" or "03 months to …" and are present in at least one page.
  - The language model (OpenAI API) understands and follows the prompt instructions accurately.
- **Approach:**
  - The pipeline uses a hybrid of rule-based filtering (regex and keywords) and LLM-based extraction.
  - Embedding similarity is used to rank candidate pages for section selection.

    - A structured prompt via a Jinja template provides explicit guidance for data extraction.

---

5. Limitations and Handling Data Inconsistencies

5.1. Limitations

- **Extraction Accuracy:**
  Some extracted values may be incorrect or inconsistent due to:
    - Variations in PDF formatting and layout.
    - Incomplete or ambiguous text extraction.
    - Limitations in the language model's interpretation of the prompt.
- **Dependence on Prompt Quality:**
  The accuracy of extracted data relies heavily on the quality of the prompt. Misinterpretation of headings (e.g., confusing a 9-month column for a 3-month column) can lead to errors.
- **Static Rules and Heuristics:**
  Hard-coded regex and keyword lists may not cover all possible variations in real-world financial statements.

5.2. Handling Data Inconsistencies

- **Post-Extraction Validation:**
  Implement additional validation steps to verify that extracted dates and financial figures match the expected formats.
- **Fallback Mechanisms:**
  If the LLM output is ambiguous or missing, consider flagging the extraction result for manual review.
- **Training a Neural Network:**
  One solution to improve accuracy would be to train a dedicated neural network on a large dataset of real examples. This network could learn to:

    - Identify and classify different income statement sections.
    - Extract the correct 3-month period data reliably.
- **Section-Wise Breaking:**
  The pipeline can be enhanced by breaking down pages sectionwise, particularly by leveraging “Statement of Profit or Loss” (SOPL) sections to isolate the part containing the 3-month data. This would allow more precise extraction by limiting the context given to the LLM.

Below is the updated documentation section with explicit equations based on the code (no changes to the .py file have been made):

Updated Documentation: Calculation Equations & Final Results

6.2. Final Results

- **Output Format:**
  The final output is provided as a structured JSON for each PDF and an aggregated CSV file containing the following fields:
    - **CompanyName**
    - **Revenue, COGS, GrossProfit, Other Operating Income, Distribution Costs, Administrative Expenses, Other Operating Expense, NetIncome**
    - **PeriodStartEnd** (representing the date range for the latest 3-month period)
    - **FileName**
- **Calculation Equations:**
  The pipeline computes additional metrics from the raw scraped data using the following equations:
    - **operating_expenses**
      Equation:
      Operating_expenses = DistributionCosts+AdministrativeExpenses + OtherOperatingExpense
      Explanation: This sums the three expense components to yield the total

operating expenses.

- **operating_income**
  Equation:
  Operating_income = GrossProfit+OtherOperatingIncome - operating_expenses
  Explanation: Operating income is calculated by adding Gross Profit and Other Operating Income, then subtracting the total operating expenses.

  This calculation is done by running the preprocess.py, which generates a csv to apply the prediction model and a json to create the react-dashboard

- **Observed Issues:**
  - Some financial figures may not be accurately extracted due to variations in PDF layouts and ambiguities in the text.
  - In certain cases, the model might misidentify columns or return data from older periods rather than the most recent 3-month period.
- **Recommendations for Improvement:**
  - **Neural Network Training:** Develop and train a dedicated neural network using a large dataset of real quarterly reports to enhance extraction accuracy.
  - **Enhanced Section Segmentation:** Improve segmentation by dividing pages into smaller sections using markers such as “Statement of Profit or Loss” (SOPL) to better isolate the region containing the desired 3-month data.
  - **Dynamic Regex and Configurations:** Regularly update and refine regex patterns and keyword lists based on new examples and user feedback to better capture variations in report formats.

---

7. Conclusion

This document has detailed the architecture, methods, and challenges of our financial data extraction pipeline. The system extracts key financial metrics from PDFs using a combination of rule-based methods (including regex and keyword filtering) and LLM-based extraction with a structured prompt. Although the current implementation demonstrates a viable approach, improvements—such as advanced machine learning techniques and more refined section segmentation—are recommended for higher accuracy and robustness. Future work should focus on these enhancements, as well as on continuously updating extraction rules based on real-world examples.

This concludes the updated documentation on the financial data extraction pipeline, including the calculation equations and final output format.