**Approach, Assumptions, and Coding Best Practices**

**Approach**

1. **OCR Extraction**: We use the Veryfi API to process PDF files and extract OCR text from them. The API client is configured with credentials, and the process\_document method is used to get the OCR text and other relevant data from each PDF file.
2. **Validation**: The is\_valid\_document function ensures that the extracted data adheres to specific formats:
   * **Vendor Name**: Should be at least one word.
   * **Vendor Address**: Should be at least one word.
   * **Invoice Number**: Should be numeric.
   * **Date**: Should be in YYYY-MM-DD format.
3. **Data Extraction**: The extract\_information function processes the OCR response to:
   * Extract and save the OCR text to a text file.
   * Extract specific data such as SKU, vendor name, address, and line items from the OCR response.
   * Save the extracted data into a JSON file.
4. **File Handling**: The script reads all files from a specified folder, processes them, and saves the results in designated folders for OCR text and extracted JSON data.

**Assumptions**

1. **OCR Extraction**: The OCR extraction from the Veryfi API is reliable and provides data in a structured format.
2. **Data Validation**: The provided regex patterns in is\_valid\_document are sufficient to validate the data formats. The vendor\_name and vendor\_address are assumed to be one or more words, which is a generic pattern suitable for most cases.
3. **SKU Extraction**: The SKU is assumed to be an 8-character alphanumeric string enclosed in parentheses. If the format is different, adjustments would be needed.
4. **Tax Rate**: It is assumed that tax rate data is not available in the provided OCR text based on the description.

**Coding Best Practices**

1. **Error Handling**: Added try-except blocks around file processing to handle and report errors gracefully.
2. **Folder Management**: Used os.makedirs to ensure the output folders exist before saving files.
3. **File Naming**: The script generates file names dynamically based on the original PDF file names, ensuring that the output files are correctly associated with their source files.
4. **Code Readability**: Comments are provided in uppercase to enhance readability and ensure clarity of the code’s purpose and functionality.
5. **String Formatting**: Used f-strings for formatting file names and paths, which is more readable and efficient compared to older string formatting methods.
6. **Validation and Extraction Separation**: Kept validation and data extraction separate to maintain code modularity and improve maintainability.

**Code Paradigm**

* **Imperative Programming**: The code follows an imperative programming paradigm where instructions are executed in a sequential manner to achieve the desired outcomes.

**Unit Tests**

* **Testing Functions**: The validation and extraction functions should be unit-tested to ensure they handle various edge cases and data formats correctly. Examples include:
  + **Testing is\_valid\_document**: Verify the function with various valid and invalid responses to ensure correct validation.
  + **Testing extract\_information**: Validate the function with sample OCR responses to check if the extraction logic is accurate and complete.