```
# Import necessary libraries
import json
from collections import defaultdict
from typing import Dict, List, Tuple, Any, Union
import os
import pandas as pd
from datetime import datetime
def iterate_json_files(directory: str):
    """
    Generator function to iterate over JSON files in a given directory.
    Args:
    directory (str): Path to the directory containing JSON files.
    Yields:
    tuple: A tuple containing the filename and full file path for each JSON file.
    """
    for filename in os.listdir(directory):
        if filename.endswith('.json'):
            file_path = os.path.join(directory, filename)
            yield (filename, file_path)
def flatten_json(data: Union[Dict[str, Any], List[Any]], prefix: str = '') -> Dict[str,
Any]:
    """
    Recursively flatten a nested JSON structure into a flat dictionary.
    Args:
    data (Union[Dict[str, Any], List[Any]]): The JSON data to flatten.
    prefix (str): The current key prefix for nested structures.
    Returns:
    Dict[str, Any]: A flattened dictionary representation of the JSON data.
    """
    flattened = {}
    if isinstance(data, dict):
        for key, value in data.items():
            new_key = f"{prefix}.{key}" if prefix else key
            if isinstance(value, dict):
                flattened.update(flatten_json(value, new_key))
            elif isinstance(value, list):
                flattened[new_key] = f"Array[{len(value)}]"
            else:
                flattened[new_key] = value
    elif isinstance(data, list):
        for i, item in enumerate(data):
            new_key = f"{prefix}[{i}]" if prefix else f"[{i}]"
            if isinstance(item, dict):
                flattened.update(flatten_json(item, new_key))
            elif isinstance(item, list):
                flattened[new_key] = f"Array[{len(item)}]"
            else:
                flattened[new_key] = item
    else:
        flattened[prefix] = data
    return flattened
def analyze_json_files(directory: str):
    """
    Analyze JSON files in the given directory to extract various statistics and structures.
    Args:
    directory (str): Path to the directory containing JSON files.
    Returns:
    tuple: A tuple containing various analysis results:
        - field_frequency: Dictionary of field occurrences across all files.
        - missing_fields: Dictionary of files with their missing fields.
        - extra_fields: Dictionary of files with their extra fields.
        - file_structures: Dictionary grouping files by their structure.
        - file_data: Dictionary containing detailed data for each file group.
    """
    field_frequency = defaultdict(int)
    missing_fields = defaultdict(list)
    extra_fields = defaultdict(list)
    all_fields = set()
    file_structures = defaultdict(list)
    file_data = defaultdict(list)
    # Iterate through all JSON files in the directory
    for file_name, file_path in iterate_json_files(directory):
        with open(file_path, 'r') as f:
            data = json.load(f)
        # Flatten the JSON structure
        flattened_data = flatten_json(data)
        fields = set(flattened_data.keys())
        all_fields.update(fields)
        # Count field occurrences
        for field in fields:
            field_frequency[field] += 1
        # Group files by their structure
        structure_key = tuple(sorted(fields))
        file_structures[structure_key].append(file_name)
        file_data[structure_key].append((file_name, file_path, flattened_data))
    # Calculate total number of files and identify common fields
    total_files = sum(len(files) for files in file_structures.values())
    common_fields = set(field for field, count in field_frequency.items() if count ==
    total_files)
    # Identify missing and extra fields for each file
    for structure, files in file_structures.items():
        missing = common_fields - set(structure)
        extra = set(structure) - common_fields
        for file in files:
            if missing:
                missing_fields[file].extend(missing)
            if extra:
                extra_fields[file].extend(extra)
    return field_frequency, missing_fields, extra_fields, file_structures, file_data
def generate_summary_report(field_frequency: Dict[str, int],
                            missing_fields: Dict[str, List[str]],
                            extra_fields: Dict[str, List[str]],
                            file_structures: Dict[Tuple[str], List[str]]) -> str:
    """
    Generate a summary report based on the analysis results.
    Args:
    field_frequency (Dict[str, int]): Dictionary of field occurrences.
    missing_fields (Dict[str, List[str]]): Dictionary of files with their missing fields.
    extra_fields (Dict[str, List[str]]): Dictionary of files with their extra fields.
    file_structures (Dict[Tuple[str], List[str]]): Dictionary grouping files by their
    structure.
    Returns:
    str: A formatted string containing the summary report.
    """
    total_files = sum(len(files) for files in file_structures.values())
    report = f"JSON Files Analysis Report\n"
    report += f"Total files analyzed: {total_files}\n\n"
    # Add field frequency information to the report
    report += "Field Frequency:\n"
    for field, count in sorted(field_frequency.items(), key=lambda x: x[1], reverse=True):
        report += f" {field}: {count} ({count/total_files*100:.2f}%)\n"
    # Add missing fields information to the report
    report += "\nMissing Fields:\n"
    for file, fields in missing_fields.items():
        report += f" {file}: {', '.join(fields)}\n"
    # Add extra fields information to the report
    report += "\nExtra Fields:\n"
    for file, fields in extra_fields.items():
        report += f" {file}: {', '.join(fields)}\n"
    # Add file grouping information to the report
    report += "\nFile Grouping:\n"
    for i, (structure, files) in enumerate(file_structures.items(), 1):
        report += f" Group {i} ({len(files)} files):\n"
        report += f" Fields: {', '.join(structure)}\n"
        report += f" Files: {', '.join(files[:5])}{'...' if len(files) > 5 else ''}\n"
    return report
def create_dataframes(file_data: Dict[Tuple[str], List[Tuple[str, str, Dict]]]) ->
Dict[int, pd.DataFrame]:
    """
    Create pandas DataFrames for each group of files with the same structure.
    Args:
    file_data (Dict[Tuple[str], List[Tuple[str, str, Dict]]]): Dictionary containing
    detailed data for each file group.
    Returns:
    Dict[int, pd.DataFrame]: A dictionary of DataFrames, keyed by group number.
    """
    dataframes = {}
    for i, (structure, files) in enumerate(file_data.items(), 1):
        df_data = []
        for file_name, file_path, data in files:
            row = {
                'file_name': file_name,
                'file_path': file_path,
                'field_names': ', '.join(sorted(data.keys())),
                **data
            }
            df_data.append(row)
        df = pd.DataFrame(df_data)
        # Reorder columns to put field_names after file_path
        columns = ['file_name', 'file_path', 'field_names'] + [col for col in df.columns if col
        not in ['file_name', 'file_path', 'field_names']]
        df = df[columns]
        dataframes[i] = df
    return dataframes
def main(directory: str, report_directory: str):
    """
    Main function to orchestrate the JSON file analysis process.
    Args:
    directory (str): Path to the directory containing JSON files to analyze.
    report_directory (str): Path to the directory where reports and DataFrames should be
    saved.
    Returns:
    Dict[int, pd.DataFrame]: A dictionary of DataFrames, keyed by group number.
    """
    # Analyze JSON files
    field_frequency, missing_fields, extra_fields, file_structures, file_data =
    analyze_json_files(directory)
    # Generate and print the summary report
    report = generate_summary_report(field_frequency, missing_fields, extra_fields,
    file_structures)
    print(report)
    # Create the reports directory if it doesn't exist
    os.makedirs(report_directory, exist_ok=True)
    # Generate timestamp for the report file name
    timestamp = datetime.now().strftime("%Y%m%d-%H%M%S")
    # Save the report to a file in the specified directory with timestamp
    report_file_name = f'{timestamp}_json_analysis_report.txt'
    report_file_path = os.path.join(report_directory, report_file_name)
    with open(report_file_path, 'w') as f:
        f.write(report)
    print(f"Report saved to: {report_file_path}")
    # Create and save dataframes
    dataframes = create_dataframes(file_data)
    for group_number, df in dataframes.items():
        df_file_path = os.path.join(report_directory, f'group_{group_number}_dataframe.csv')
        df.to_csv(df_file_path, index=False)
        print(f"Group {group_number} DataFrame saved to: {df_file_path}")
    return dataframes # Return the dataframes for potential future use
if __name__ == "__main__":
    # Specify the directory containing the JSON files
    json_directory =
    r"C:\Users\kroy2\Documents\python\projects\json_processor\json_test_files2"
    # Specify the directory where the report and dataframes should be saved
    report_directory = r"C:\Users\kroy2\Documents\python\projects\json_processor\reports"
    # Call the main function with the specified directories
    dataframes = main(json_directory, report_directory)
```