How the code works:

The Python script is designed to extract, standardize, and reconcile data from multiple source systems. It reads configurations from json file, processes system-specific data, and generates summary reports. Then reconciliation process compares extracted data against reference data to identify matches and mismatches based on defined rules.

**1. High-Level Workflow**

1. Load configuration settings from a JSON file.
2. Process raw data files for each system:
   * Standardize column names.
   * Format dates.
   * Aggregate data.
   * Save standardized summaries.
3. Perform reconciliation by comparing standardized summaries with reference files:
   * Calculate differences in charges, payments, and adjustments.
   * Identify matches or mismatches based on percentage thresholds.
4. Save final outputs:
   * Combined standardized summary file.
   * Combined reconciliation report.

## 1.1 Configuration Loading

**Function: load\_config(config\_path)**

* Loads configuration settings from a JSON file containing system-specific parameters to avoid hardcoded values.
* Returns a dictionary for further processing.

## 1.3 Data Standardization

**Function: process\_system\_data(system\_name, config, all\_standardized\_data)**

* Extracts system-specific configurations.
* Iterates through raw data files.
* Applies transformations:
  + Renames columns based on mappings.
  + Formats date columns.
  + Adds metadata columns.
  + Aggregates data.
* Saves the standardized summary to an Excel file.

**1.4 Reconciliation**

**Function: process\_excel\_files(summary\_df, file\_c, config)**

* Reads standardized summary and reference file.
* Merges datasets using a concatenated key (Concat\_Key).
* Calculates differences and percentage variations.
* Flags rows as "Match" or "Mismatch" based on percentage thresholds.

**1.5 Main Execution Flow**

**Function: main()**

1. Loads configuration settings.
2. Processes and standardizes data for each financial system.
3. Reconciles extracted summaries with reference datasets.
4. Saves final processed files and reconciliation reports.

**Detailed explanation:**

**2. JSON Configuration File**

**2.1 Purpose and Structure**

The JSON file serves as a dynamic configuration system, defining system-specific parameters to avoid hardcoded values in the script. This allows easy scalability and updates without modifying the script.

**2.2 Key Elements in JSON File**

Each system in systems (e.g., intergy, modmed, ecw, nextgen) has:

* **columns**: Maps original column names to standardized names.
* **staging\_folder**: Path to the folder where processed data will be saved.
* **output\_filename**: Name of the output summary file.
* **date\_format**: Format in which date values should be standardized.
* **aggregate\_functions**: Defines how numerical data should be aggregated (sum, average, etc.).
* **add\_columns**: Specifies additional metadata fields (e.g., source folder name).
* **filter\_conditions**: Conditions to filter data before processing.

**3.1 Importing Required Libraries**

import os

import json

import pandas as pd

* **os**: Used for handling file paths and directory operations.
* **json**: Enables reading and parsing JSON configuration files.
* **pandas**: Provides data processing and manipulation capabilities.

**3.2 Loading the Configuration File**

def load\_config(config\_path):

Defines a function load\_config that takes config\_path as an argument.

* Opens the JSON file in read mode ("r").
* Uses json.load(file) to parse JSON content into a Python dictionary.
* Returns the dictionary for further processing.

**3.3 Extracting Folder Structure Information**

def get\_correct\_parent\_folder(folder\_path):

**Splits folder\_path** using the system’s default separator (os.sep).

* **Extracts the parent folder** located 10 levels before the last directory.
* If the path has fewer than 10 parts, returns None to avoid an index error.

**3.4 Processing System Data**

def process\_system\_data(system\_name, config, all\_standardized\_data):

**Step-by-Step Breakdown:**

1. **Loads settings** for the given system\_name from config.
2. **Creates the staging folder** if it does not exist.
3. **Iterates over raw data files**, filtering only .xlsx,.csv,. tsv formats while skipping temporary system files (~$).
4. **Loads data into a Pandas DataFrame**.
5. **Standardizes column names** according to the mapping in columns.
6. **Formats date columns** (Date Post, Month) using date\_format.
7. **Adds metadata columns** (folder\_name, parent\_folder\_before\_last).
8. **Performs aggregations** based on aggregate\_functions.
9. **Saves the cleaned dataset** to output\_filename.

**3.5 Reconciling Processed Data**

def process\_excel\_files(summary\_df, file\_c, config):

**Reconciliation Steps:**

1. **Loads both files (summary\_df and file\_c)**.
2. **Determines the correct date column** (Date Post or Month).
3. **Generates a unique key (Concat\_Key)** for merging records.
4. **Performs an outer join** between summary\_df and file\_c.
5. **Computes differences**:
   * Payments
   * Charges
   * Adjustments
6. **Calculates percentage differences** to classify records as Match or Mismatch.
7. **Saves the reconciled output**.

**3.6 Main Execution Flow**

def main():

**Execution Flow:**

1. **Loads configuration settings.**
2. **Processes and standardizes each financial system’s data.**
3. **Reconciles extracted summaries with reference datasets.**
4. **Saves final processed files and reconciliation reports.**

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Ensures the script runs only when executed directly.

**4. Output Files**

* **Standardized Data Files**: Contains cleaned and formatted data.
* **Reconciled Data Files**: Highlights matches and mismatches.
* **Final Reports**: Summarized reconciliation results.

**5. Error Handling and Logging**

* Uses try-except to handle file errors.
* Skips files with incorrect formats.
* Displays meaningful error messages.

**6. Conclusion**

The script automates data reconciliation by:

* **Standardizing inputs dynamically**.
* **Performing aggregations and metadata tagging**.
* **Ensuring high accuracy reconciliation**.
* **Handling multiple source systems seamlessly**.