Introduction

This write-up presents a comprehensive analysis and transformation of student enrollment data from the KoalaSis API. The goal of the project is to download, clean, process, and validate the data, creating a final dataset that includes relevant information about student demographics and enrollment dates. The resulting dataset will be used for various analytical purposes, such as identifying trends and patterns in student enrollment.

Data Overview

The data for this project is sourced from the KoalaSis API, which provides information about students, schools, and enrollments. The data is downloaded as JSON objects and converted to pandas DataFrames for further processing. The following data sources are used:

Students: Contains information about each student, including their unique ID, first name, last name, and gender.

Schools: Contains information about each school, including its unique ID, name, and start and end dates.

Enrollments: Contains information about each student's enrollment, including the student ID, school ID, entry date, and enrollment end date.

Methodology

The methodology employed in this project involves the following steps:

Download data from the KoalaSis API and save it as CSV files for further processing.

Read CSV files and create pandas DataFrames for students, schools, and enrollments.

Merge the DataFrames on appropriate keys and transform the data to include only necessary columns.

Apply transformations to selected fields, such as capitalizing names, formatting gender codes, and calculating exit/withdrawal dates.

Assign appropriate data types to each column and validate the final dataset for missing values and duplicates.

Save the transformed data to a CSV file for further analysis and visualization.

Code Explanation

The code implementation is divided into several functions, each with a specific purpose and functionality. These functions handle tasks such as data download, merging, transformation, validation, and saving. Key functions include:

download_data_to_csv: Downloads data from the KoalaSis API and saves it as CSV files.

read_csv_files: Reads CSV files and creates pandas DataFrames for students, schools, and enrollments.

merge_and_transform_data: Merges the DataFrames and applies transformations to selected fields.

validate_data: Validates the transformed data for missing values and duplicates.

save_transformed_data: Saves the transformed data as a CSV file for further analysis.

Throughout the code, conscious trade-offs and challenges have been addressed. For example, calculating the exit/withdrawal date for students required the use of the BDay class to account for business days, ensuring accurate results. Additionally, the code ensures proper handling of names with special characters and capitalization.

Results & Evaluation

The code successfully processes and transforms the data, resulting in a clean, well-structured dataset that includes relevant information about student demographics and enrollment dates. The final output contains the following columns:

SchoolId

NameOfInstitution

StudentUniqueId

LastSurname

FirstName

DisplayName

Gender

EntryDate

ExitWithdrawDate

Trade-offs & Challenges

Some trade-offs and challenges were encountered during the development of the code. For example, the decision to calculate exit/withdrawal dates using business days added complexity but ensured accurate results. Additionally, handling names with special characters, capitalization, and inconsistent formats required the use of custom functions and regular expressions, which may be less efficient for large datasets.

Moreover, we had to address the issues related to capitalizing names with special characters or apostrophes, which led to the creation of a custom function **capitalize_name_parts**. This function ensures proper capitalization of names, considering various edge cases and taking into account the possible presence of special characters.

Additionally, when working with date and time data, we encountered challenges with missing or inconsistent data types. To address this issue, we used **Union[str, pd.Timestamp]** in the **calculate_exit_withdraw_date** function, allowing it to accept both string and timestamp inputs. This approach enabled us to handle different input data types, perform the necessary date-time operations, and ensure that the function works seamlessly with the rest of the pipeline.

Future Improvements

There are several potential improvements and refinements that could be made to the code and methodology, including:

Enhancing the data validation process to include more comprehensive checks, such as outlier detection or inconsistencies in data relationships.

Implementing more efficient data processing methods, such as parallel processing or leveraging specialized libraries like Dask or Vaex, to handle large datasets.

Expanding the functionality of the code to support additional data sources, formats, and APIs, making it more versatile and adaptable to different use cases.

Introducing a user-friendly interface, such as a command-line interface (CLI) or a web application, to allow users without coding expertise to interact with the tool and perform data transformation tasks more easily.

Incorporating a data versioning system to track changes in the data and maintain a historical record of the transformations applied to the dataset.

Adding support for data visualization and reporting capabilities, enabling users to generate insights and visual representations of the transformed data directly from the tool.

Implementing a robust error handling and logging system to facilitate debugging and troubleshooting of the code.

Developing a comprehensive test suite to ensure code stability and correctness, allowing for continuous integration and deployment (CI/CD) practices.

Integrating the code into a larger data pipeline or a data platform, such as Apache Airflow, to automate the process of data extraction, transformation, and loading (ETL) for more complex workflows.

Enhancing code modularity and following best practices for code organization and documentation, making it easier for other developers to understand, maintain, and extend the codebase.