CSED226 Intro. to Data Analysis : Assn1 Report

September 12, 2025

Student 20240505 Hyunseong Kong

1. Overview

The first assignment is to implement a Python program that processes given data.
The main goal is to reshape the schema into a specific format while properly handling edge cases.

2. Differences

2-1. Implementation

The main distinction lies in code structure. The solution code explicitly defines the tidy row model and organizes core features into separate functions, making it more extensible. My code, on the other hand, is written as a single block. Details are summarized below:

Aspect	Submitted (`20240505.py`)	Solution (`practice1_solution.py`)
Code Flow	One large script	Modular, with separate functions
CSV Handling	`csv.reader`	`csv.DictWriter`
Edge Cases	Not fully handled	Result directory creation Use Optional for None

THe submitted code relies on `csv.reader` with index-based access (rows[0], rows[4:]). So it may breaks if column order changes. Also, missing values are handled ad hoc with inline conditionals. On the other hand, the solution code uses `csv.DictReader`, accessing data by column name. Thus, the solution code is robust against formatting inconsistencies. In addition, my submitted code reformatted the data in real time during the CSV reading loop, so in the saving stage it only needed to write the processed results. The solution code, on the other hand, performed additional time-related computations at the saving stage. When storing parquet data, the solution code processed the data by recording it into a list while writing to CSV. In my case, I handled the storage simultaneously while reading from the CSV.

2-2. Data Handling

The solution clearly defined the data schema using `@dataclass`. This wrapper automatically provides useful methods for data management, making it convenient to work with. Furthermore, it explicitly defined the type of `count` as `Optional`, thereby handling `None` values properly. Furthermore, the solution code explicitly specifies all data types during both the reading and saving steps, which helps prevent potential errors. This is not possible in the submitted code, as it omits some of the dataset's types.
Also, the solution code used an iterator when reading CSV files and returned the rows as dictionaries. Since an iterator generates elements on demand (streaming), it has the advantage of consuming very little memory. In contrast, lists eagerly store all data in memory. Once it stored, it provides fast access and efficient for repeated use. However, in my submitted code, the data was read only once and saved once, so this advantage was not effectively utilized. Furthermore, although the test dataset was relatively small, when working with very large datasets that cannot fit entirely into memory, my submitted approach would not be feasible.

3. Lessons Learned

Through this assignment, I learned the importance of modular code design, proper use of CSV utilities, and systematic handling of exceptions and edge cases. The most significant lesson I learned is to use `yield` rather than simply saving the entire dataset into a Python list. This judgment stemmed from the fact that Python lists cannot be used in a big data. However, since iterators also incur some overhead when processing `yield`, it is important to select the appropriate method according to the characteristics of the given dataset.