Developing this project posed several challenges, mainly dealing with unsupported data types and creating a flexible user input system for column removal. One of the more challenging aspects was handling unsupported data types when saving to a SQL database table. API responses often contain complex data, such as lists or dictionaries, which cannot be stored directly. I had to inspect each column and convert these complex structures into strings to resolve the problems I was having with the code. While this solution worked, it added complexity to the code and reminded me to account for the tools' limitations.

Surprisingly, working with API integration was much easier than expected. Initially, I thought handling API responses, parsing complex JSON structures, and ensuring the data was in the correct format would be challenging. However, Python's “requests” library made receiving data straightforward. Once I understood the API’s structure and the data it returned, normalizing and storing it became a manageable process. Additionally, adding query parameters to the API call provided flexibility in controlling the results, such as limiting the number of records and filtering data by specific criteria.

A utility like this would be helpful for other data projects. I have read that many data scientists frequently deal with varied data sources, often needing to combine, clean, and store datasets in different formats. This tool’s ability to handle CSV and API data makes it adaptable to various scenarios. This utility provides a flexible and interactive way to process, clean, and store data, streamlining workflows in future projects. This project was a valuable learning experience, particularly in handling unsupported data types and building user-friendly input mechanisms. While some tasks were more straightforward than anticipated, others required deeper problem-solving. The utility’s adaptability to different data sources and formats will absolutely benefit future projects.