**○ Why did you choose the technology stack that you did?**

I chose to use Python because it is the object-oriented language I am most familiar with. I went with using the Python library Pandas over PySpark to manipulate dataframes because it is what I am familiar with and has nearly identical functionality. As far as my choice of PostgreSQL over MariaDB and MySQL, it is the SQL dialect that I have worked with the most (multiple years of experience) and feel the most comfortable with -- for the purposes of this project, with its simplicity and relatively small scale, I do not think there were any significant advantages to using one SQL dialect and databases over another. Likewise, picking PostgreSQL necessitated my use of the Python library psycopg2, which is the most popular PostgreSQL database adapter Python library.

**○ How did determine what data type should be used for each column when evaluating the files?**

I first just loaded the CSV files as tables into a local database and used Datagrip to observe what each column's data looked like, including selecting for outliers from certain columns (e.g. finding the max length of a value in a column or, in the case of zip codes, finding zip codes that had non-numeric characters (a hyphen.)) Most of the columns were overtly string-formatted, which I chose to use varchar for. Columns that had numbers without decimals, I chose to use int for, as none of them exceeded int's max size of 2^32 - 1 (though phone numbers are close, so I opted to store all phone numbers in varchar, as they might include non-numeric characters sometimes anyway.) I chose to use float for any numbers that included decimals, which defaults to double precision in PostgreSQL. Then, there were notable exceptions:

As referenced earlier, while importing data to my local database tables, I was receiving an error that the Health Facility Information file could not import the zip code columns (Facility ZIP Code, Operator ZIP Code, and Cooperator ZIP Code) as int. I queried the columns for non-numeric characters and found multiple instances of a hyphenated zip code, so I chose to standardize all "zip code" columns across the three files as varchar for potential joins between the three tables. I also chose to use the date datatype for the columns Facility Open Date, in the Health Facility Information file, and Payment Date, in the Medicaid EHR Provider Payments file. This is important for business analysts, as it allows for querying for records by a range of dates, which is important for reporting purposes. I chose to use the float datatype for the column Payment Amount in the Medicaid EHR Provider Payments file; this required some transformation, as the values contained a dollar sign and commas, but transforming the data to float format is better than casting it as the datatype money in Postgres, which is localized and makes querying more difficult. And finally, I chose to use varchar for the Facility ID column in the Hospital Information file because, when loading the data from the corresponding Pandas dataframe to my created Postgres tables, I received an error that the value '49001F' could not be cast to int. Upon querying, multiple records in that column exist with alphabetic characters, so I just changed the create table query to use varchar instead of int for that column.

**○ What assumptions did you make, and how might those assumptions affect the project?**

The biggest assumption I made was that this project would not scale. That being said, I chose the fast execution helper execute\_batch to load data into the Postgres tables, rather than the easy way of using Panda's to\_sql function (which would create a table from a dataframe and load the data from the dataframe into it) because it is orders of magnitude faster at larger scales. For the scale of this assignment, it gives maybe a four-second faster runtime, most of which is due to loading the 40,000 row Medicaid EHR Provider Payments dataframe into Postgres. If this dataframe were double the size, it would jump to something like a nine-second faster runtime, and the runtime would be vastly quicker at even larger scales. I also assumed that the data cleaning involved in this assignment did not mean necessarily changing or normalizing the data. For example, the Health Facility Information file contains state names in full, whereas the Hospital Information file has state names as two-letter abbreviations — this should be normalized across tables to two-letter abbreviations, whether through a transform or a join to a reference table that relates state names to their two-letter abbreviations. I also assumed that the focus of this assignment was not on security. The database connection credentials would be obscured as secrets in a production script connecting to a non-local database. This could include parsing of a localized config file or utilizing a tool like a secrets manager. And finally, I made the assumption that the Facility ID columns, in the Health Facility Information and Hospital Information files, as well as the columns Provider NPI and Payee NPI in the Medicaid EHR Provider Payments file, should be indexed in the Postgres tables. This has the negative effect of increasing the runtime of an UPDATE, INSERT, or DELETE SQL statement on those columns, but with the trade-off of greatly increasing query runtime when these columns are used in WHERE statements (which I figured these specific columns would be -- e.g. someone searching for records relating to a specific facility in the health\_facility\_information or hospital\_information tables, or someone searching for payment records relating to a specific provider or payee in the medicaid\_ehr\_provider\_payments table.)

**○ Approximately how long did you spend working on this project?**

5.5 hours.

**○ What was the most difficult aspect of the project?**

I ran into a problem loading data from the dataframes into the created tables' int columns. I kept receiving the error "ERROR: integer out of range" initially, and was confused as I could not find any values in the local database tables I created that would fall outside of int's max size of 2^32 - 1, but I opted to go ahead and changed all Postgres tables' int columns to bigint. When I reran the data\_challenge.py script and received the error "ERROR: bigint out of range", I knew something else was going on (bigint's max size is incredibly large.) I know that Pandas dataframes store null or empty values as "NaN", but I thought upon loading to Postgres, Postgres would convert these values to nulls; as it turns out, with my first lucky go at trial-and-error, converting these NaN values to None values in the dataframe prior to loading the dataframes into Postgres gave me the null values I wanted in my database tables and the errors disappeared.

**○ What was the easiest aspect of the project?**

Using Pandas dataframes to manipulate data is a very familiar tactic to me, so the direction I feel like this project was geared toward (using PySpark, which is essentially just an alternative to Pandas in that they both use dataframe constructs) was very straightforward and obvious to me. Also, as Postgres is the SQL dialect I have worked with almost my entire career, and using the Python library psycopg2 to manipulate Postgres databases via Python is an industry-wide common technique, having the option to load data into a Postgres database as one of the three available options presented in the challenge made the challenge much more natural for me.