ETL Project Report

Jason Cook

**Introduction**

This assignment required extracting data from an external source, transforming the data to meet specific requirements, and then loading the transformed data into a database for usage by others. The raw data came from a dataset published by the Centers for Dieses Control (CDC) containing information on deaths in the United States from 2005 – 2015.

The dataset contains two files for each year:

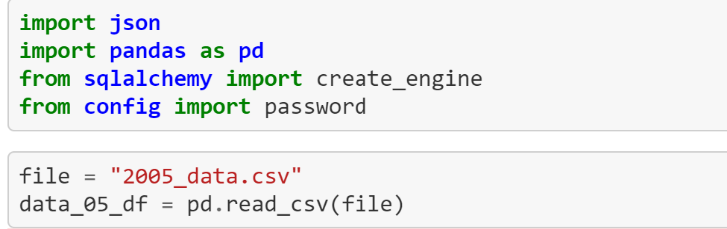
* The first file is a CSV with roughly 2.5 million rows as well as 77 columns with details about each death including the time, place, and manner of death as well as information about the decedent such as resident status, race, and education. The CSV file does not contain written descriptions of the information in each column; instead there are key values of different data types (integer, float, and object). I assume this is due to the robust nature of the data to minimize processing. These key values relate to descriptions that are found in the second file for each year.
* The second file is a json collection of dictionaries where each object in the collection is used to decode the values from the related column in the CSV file. Some of the keys are integers while others are objects.

The main purpose of the project is to select columns from the CSV file and then reference the corresponding json object that relates to the columns chosen so the actual descriptions are shown versus the key value placeholders. To do this we needed to read the data into a pandas dataframe and then export it to a mongo database so SQL queries can be run on the synthesized tables.

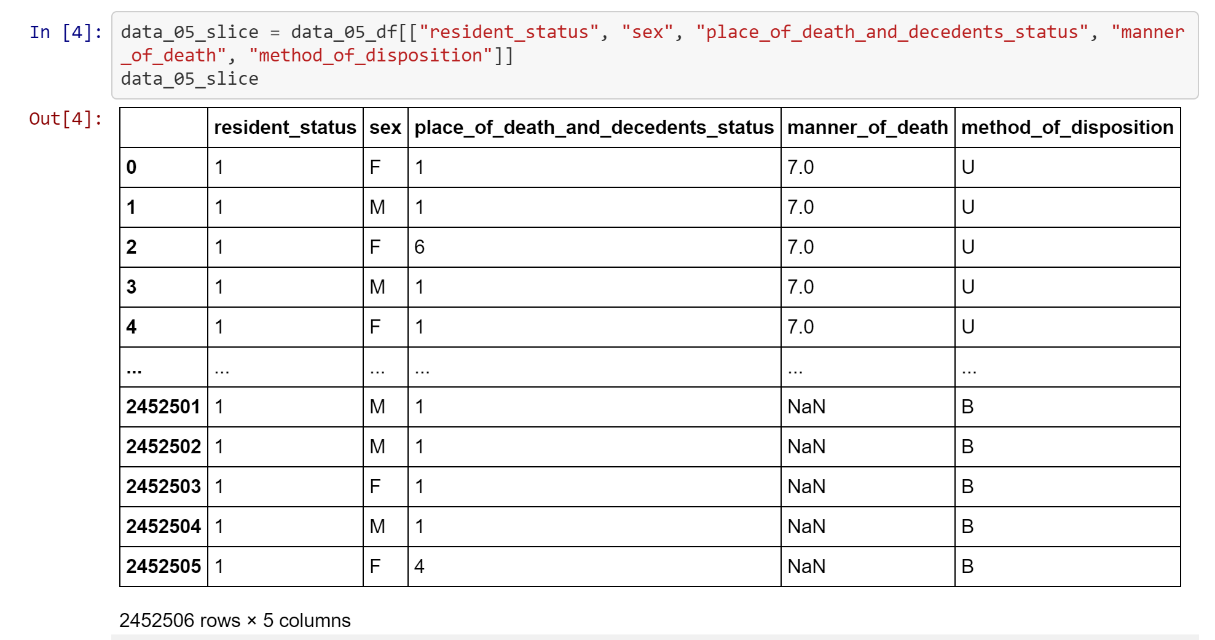
**ETL**

The extraction of the data was done via downloading from a published dataset via Kaggle.com. The dataset contained a total of 20 files (10 .json and 10 .csv), one set for each year. For this project I only needed to download 2 of the files for the year 2005.

The first step of the transformation phase of the project began with the CSV file. This is because the manipulations to the JSON file depend on what we do with the CSV. I began by importing conditionals and reading the csv file into a pandas DataFrame (df).

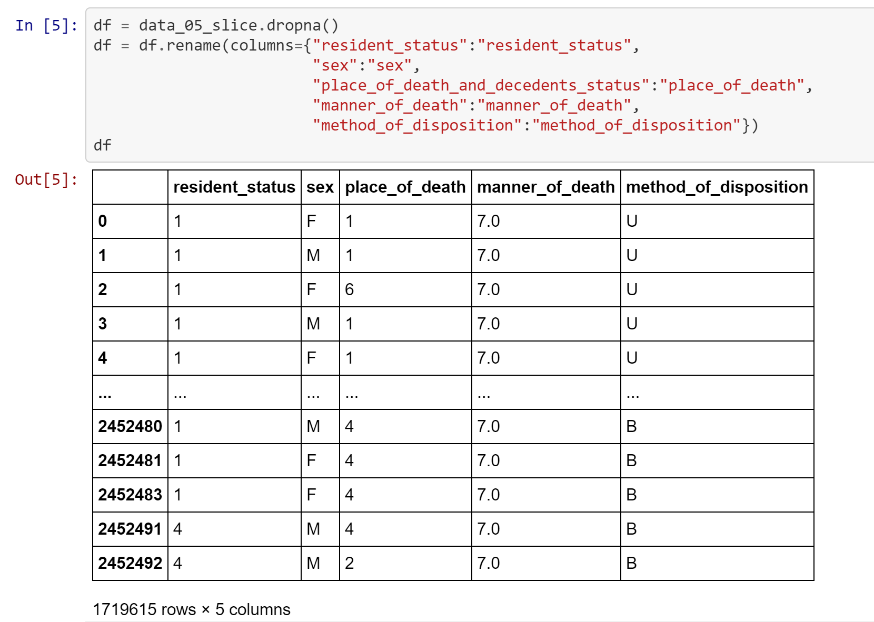


Then I was able to call the specific columns desired.

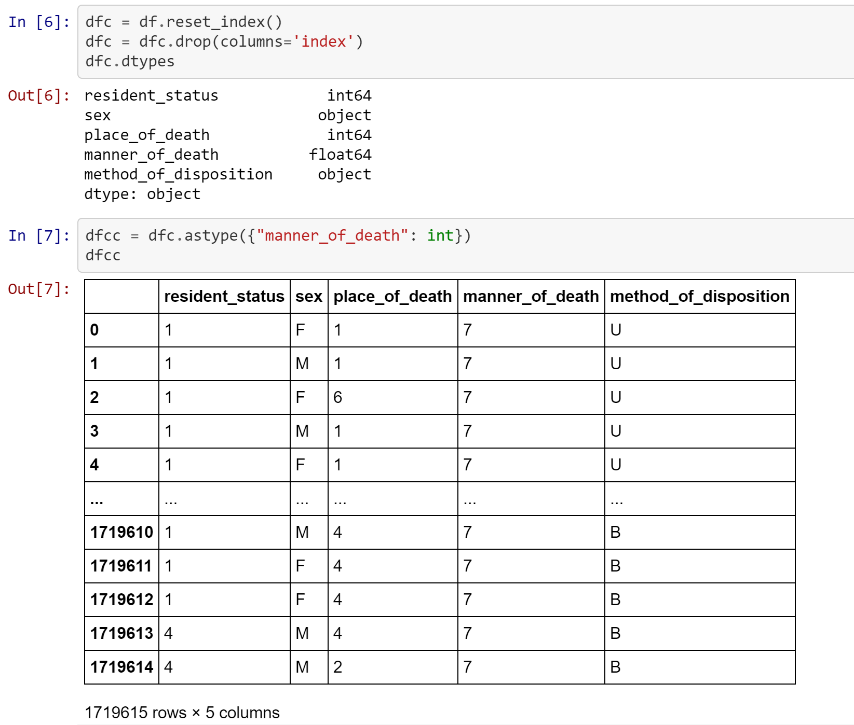


From here there was the immediate need to attend to a few minor cleaning tasks:

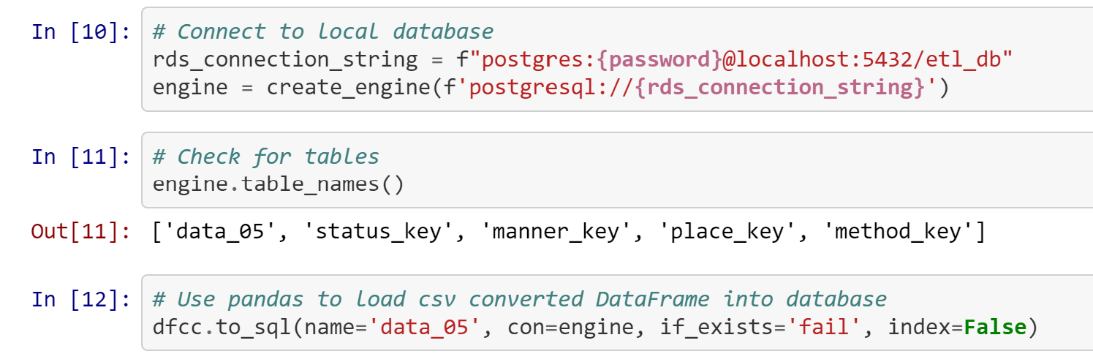
* Use the df.dropna() function to remove all rows with null values
* Rename the “place\_of\_death\_and\_decedents\_status” column to “place\_of\_death” for readability and efficiency



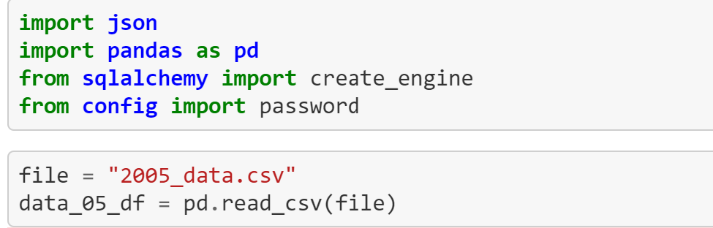
To finalize the df for exporting to my SQL database, I needed to change the datatype of the “manner\_of\_death” column to an integer to match that of the other columns with numeric datatypes. I also decided to re-index the data after eliminating around 750,000 entries.

****

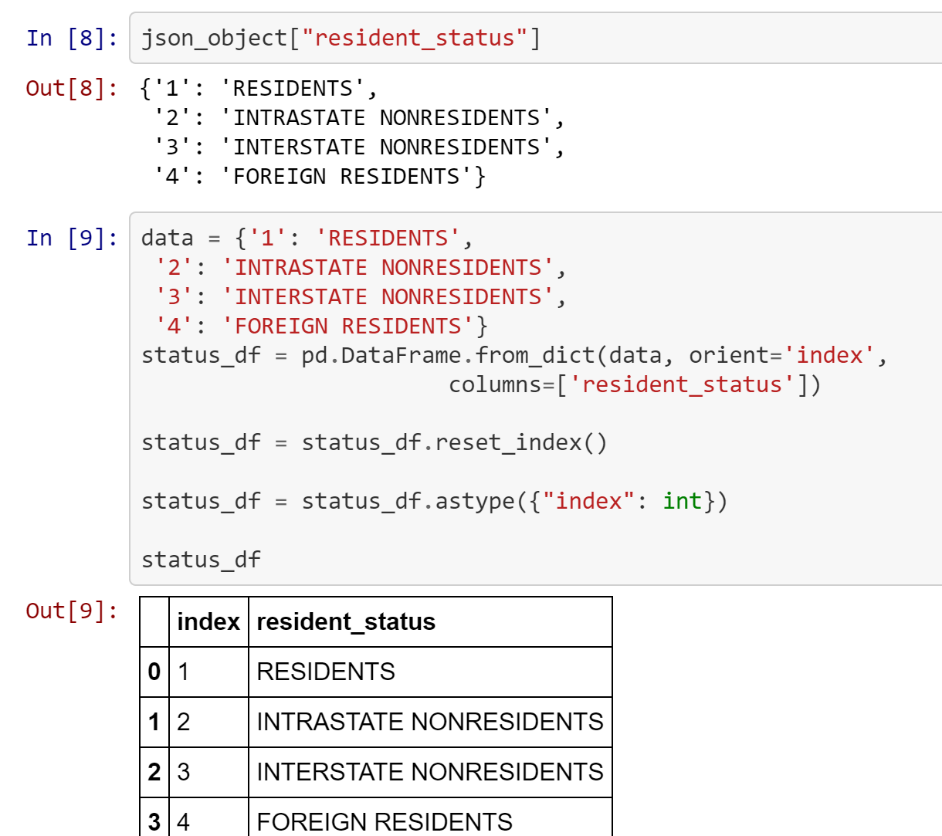
With the transformation complete, I was now able to load the df into my Mongo database as a new table.



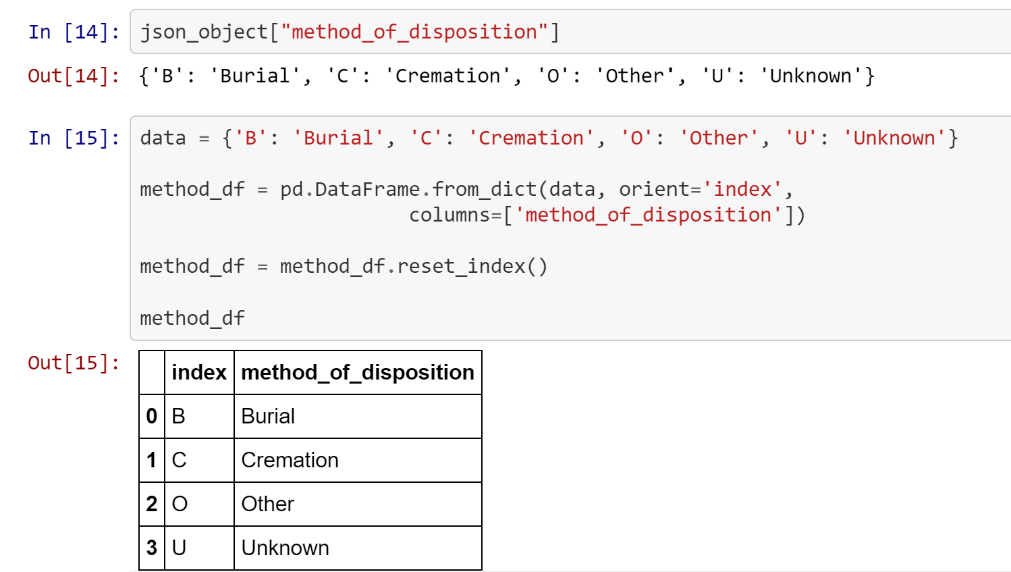
Now my attention was able to be focused on transforming the JSON file. I began with importing my conditionals and reading the file into a df.



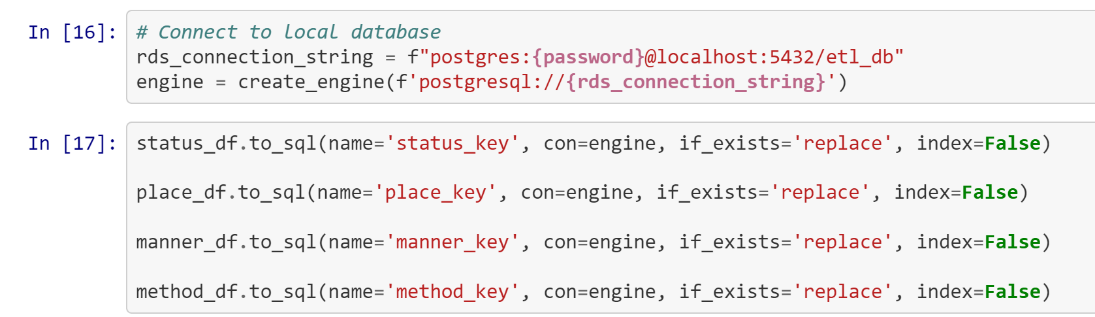
After the JSON file was loaded, I called the object in the collection that corresponded with each of the columns that I called from the CSV file. This returned a dictionary that I could immediately turn into a df with the DataFrame.from\_dict() function. This creates a one column df with the key values being the DataFrame’s index. I transformed the one column df into a two column df by resetting the index which creates a new column appropriately titled “index”. The only minor transformation needed here was to set the datatype of the new column to an integer to match that of the table created from the CSV file.



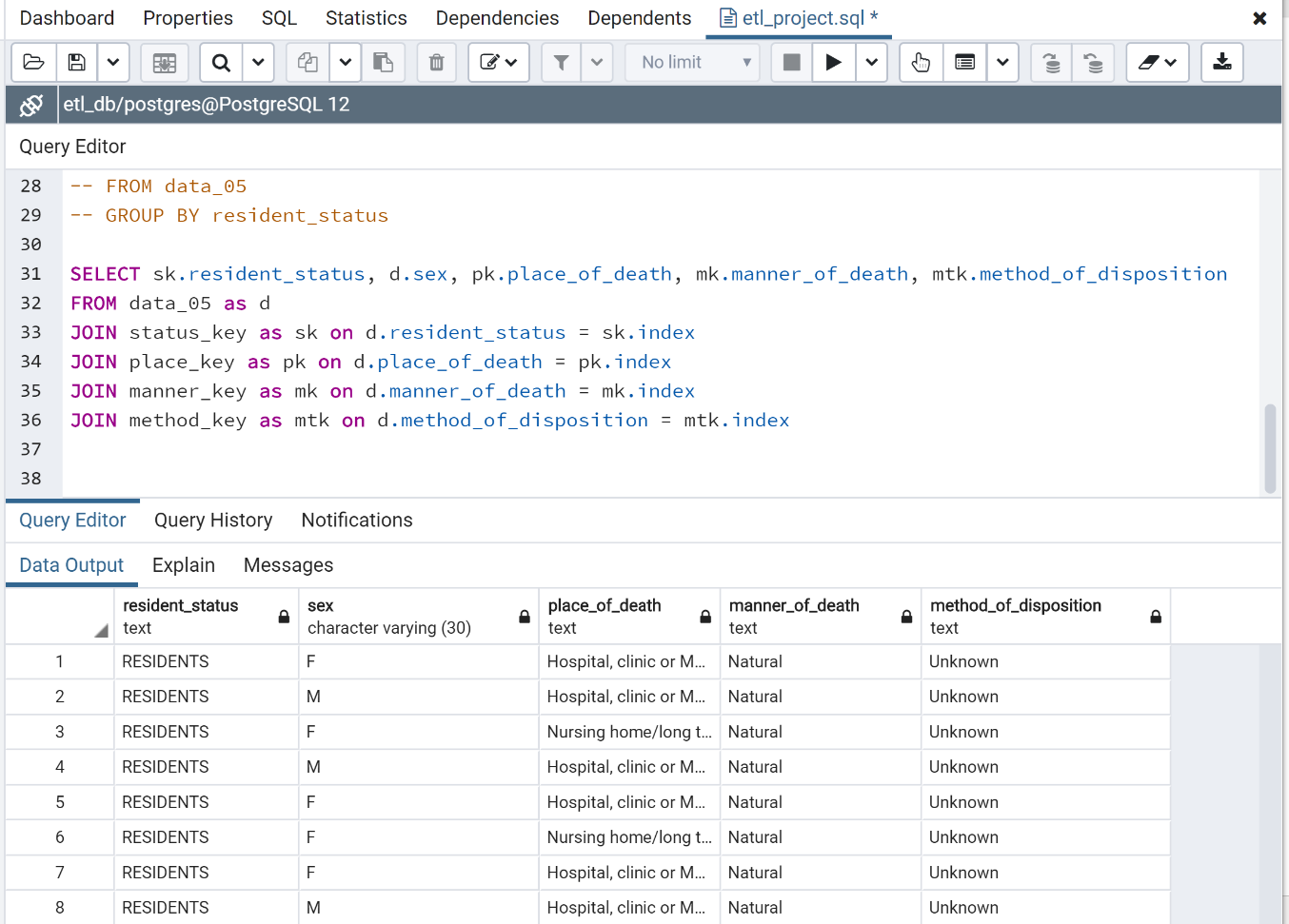
Note that this was not needed for one of the objects because the key was already an object datatype in python which corresponds to the varchar datatype in SQL.



Using this method for all of the required JSON objects I was ready to load my data into database tables for SQL queries.



Now my database had all of the tables that I needed to complete the original mission. I could now join the table with the data on deaths with the tables that contained their corresponding dictionary keys while only showing the descriptions of the data and not their code values that they originally contained.



Using this method, any particular data from the deaths table can be displayed in a view with the actual descriptions. From this point, all manner of queries can be run to aggregate or group the columns to glean useful insight from the data.