Capstone Project 1: Data Wrangling

https://github.com/mdubow/Springboard-Data-Science-Career-Track/blob/master/Capstone-Project-1/Data-Wrangling/Capstone%201%20Data%20Wrangling.ipynb

For my project I primarily work with two dataframes, both .csv files, which are referred to here as the Prescription dataset and the Overdose dataset. Both of these require some data cleaning in order to be properly used, although the steps involved are quite different. After cleaning, these dataframes will be joined together.

The Prescription dataset (gathered by the Washington Post and found on kaggle.com) is massive (> 200 GB), so the file has to be read in to Python in chunks which are subsequently processed, saved as separate files, and merged back together. The original .csv file has 42 columns, although only 7 columns are relevant to this project. Most of these are categorical variables based on region (zip code, state, city, county), one categorical variable for drug type, one column which holds timestamps for the transaction date, and our primary numerical variable - quantity. To select these columns, I started by creating a list of the column names; next, I called the .loc method on the Prescription dataframe and passed the column list as an argument.

A preliminary examination of this selected dataframe (using the method .describe to look at descriptive statistics) shows an outlier for our quantity column. Although the mean value for this column is around 3 and the median is 1, we see a max value of 5160. However, I believe this value is feasible - the vast majority of buyers of opioid-based drugs are individual users (hence the quantity of 1), but there are also wholesale resellers who could potentially purchase such large amounts as this. Furthermore, we are less concerned with average values for this data and more interested in summing over our categorical variables, so this outlier does not negatively impact our ability to understand or use the dataset.

A more difficult problem is presented in our zip code and timestamp columns. The .dtypes method shows that both columns have data stored as integers. However, these values should absolutely not be understood as integers; many zip codes begin with ‘0’ and any date before October begins with ‘0’, but storing any value as an integer will cause the front zeroes to be dropped. These zip codes are considered structured data, as the front digits represent different regions.

As a result, the dataset has numerous zip code values with only 3 or 4 digits and most timestamp values are missing a front ‘0’. First, I converted these columns to strings, as they should be. Next, I created a ‘for’ loop and iterated over the given column using the ‘enumerate’ function. This allowed me to use two iterators, one for the index (‘i’) created by ‘enumerate’ and the other for the value (‘zip_code’ and ‘date’, respectively). Within the loop, I specified that, if the value at index ‘i’ is not the proper number of digits (5 for zip codes and 8 for dates), the value should be replaced with a concatenated string of a front ‘0’ and the original value. Additionally, I used the Pandas method .to_datetime to convert the timestamp column from strings to datetime objects, which are easier to operate on. From these datetime objects I was able to extract the year, which will be used later to merge with the Overdose dataset

Checking for missing values using the .isnull shows that there are 21 rows in this chunk of 1,000,000 rows with missing values in the county column. Initially, I thought these rows might correspond to independent cities outside of counties, of which there are 41 in the United States. This assumption quickly proved incorrect - none of the cities with null county values are independent. Furthermore, independent cities in the dataset do hold a county value, e.g. the rows for transactions in Baltimore, Maryland have a county value of ‘Baltimore City’. This shows that there is no consistent pattern to the missing values, and as they are statistically insignificant (0.00002% of total values), I determined it was best to drop these values.

I needed to aggregate my dataset in some way so that I could match it with the Overdose dataset. To find the prescription rate, I knew I would need to find the total number of pills. Therefore, I grouped my data by the state and year columns, then summed over the quantity column. In my .groupby function, I set the as_index parameter to false so that the state and year would be separate columns in the new dataframe, and not a multi-index for the quantity sums. This yielded a dataframe of 380 rows, corresponding to entries for the 50 states (plus D.C.) over the years 2006 to 2012. The number of rows, 380, includes some entries for US territories which will be discarded later when merging with the Overdose dataset.

Our Overdose dataset is, fortunately, much cleaner. The original .csv file, found on data.world, contains data from 1999 to 2016. Before downloading the full file, I used a SQL query to select data between 2006 and 2012 so that it would correspond to the dates from the Prescription dataset. A quick look at the actual data showed that it contained some missing values, which it stored as either ‘Unreliable’ or ‘Suppressed’. My first step here was to use the .replace function to change the ‘Suppressed’ values to ‘NaN’. I noticed that these missing values never occurred in the first row for any state, only in the middle or at the end. Therefore, it seemed appropriate to use a backfill to replace these NaN’s with the values from the previous year. The value ‘Unreliable’ was stored exclusively in the crude rate column. This value can be easily recalculated, so I will create a new column to hold overdose rates without any null values.

After looking at the new Overdose dataset, it became clear that one more step is required to clean the Prescription dataset. This Prescription dataset uses abbreviations for the states, but the Overdose set uses full state names. To deal with this discrepancy, I created a dictionary with the abbreviations as keys and the full names as values, then used the .map function to rename the states in the Prescription dataset. I also changed the name of state column to match the Overdose dataset using the .rename function.

Now that our datasets are clean, they are ready to be joined. To do so, I used the .merge function and chose an inner join because I only wanted rows that exist in both datasets (as mentioned before, this is where the rows for non-state territories are dropped). The dataframes were merged on the state and year columns, which we know exist in both datasets. Finally, I created a new column for the prescription rate, found by dividing the quantity by the population, and a column for overdose rate, calculated similarly. This rate is per 100,000 people in order to keep our numbers on a reasonable scale. We now have a clean dataset with roughly 300 rows, ready to be explored further.