ETL Project Final Report

* The sources of data that you will extract from.
* The type of transformation needed for this data (cleaning, joining, filtering, aggregating, etc).
* The type of final production database to load the data into (relational or non-relational).
* The final tables or collections that will be used in the production database.

Extract: your original data sources and how the data was formatted (CSV, JSON, MySQL, etc).

Transform: what data cleaning or transformation was required.

Load: the final database, tables/collections, and why this was chosen.

Sources of Data:

The sources of data included the following sources:

1. The Census
   1. URL: <https://factfinder.census.gov/faces/nav/jsf/pages/searchresults.xhtml?refresh=t&keepList=t>
   2. Data extracted from census:
      1. Annual State Population Estimate
      2. Estimated Number without Health Insurance
      3. Rate of Uninsured= Number without Health/Population
      4. For the years 2009-2017.
2. OASIS--stands for Online Analytical Statistical Information System; Is a Georgia State Public Health Database.
   1. URL: <https://oasis.state.ga.us/oasis/webquery/qryER.aspx>
   2. Data extracted from OASIS:
      1. Emergency Room (ER) visitation rates for Georgia for the years 2009-2017.
3. AHRQ—stands for American Healthcare Research and Quality; Is a Healthcare Quality Organization,
   1. URL: <https://nhqr.net.ahrq.gov/inhqrdr/data/query>
   2. Data extracted from AHRQ:
      1. Pneumonia rates for Ohio, Oregon, and Georgia, for the years 2011 to 2015.

Transformation

In discussing our data transformations, I want to start by discussing the end goal of our data as this might explain the steps we took during the data transformations. We wanted data from both Medicaid expansion states and non Medicaid expansion states. We got this via two Medicaid expansion states of Ohio and Oregon. We used Georgia as our one non Medicaid expansion state. We wanted to compare these states pre and post Affordable care act implementation, hence we have the years of 2009 to 2017. More specifically, we wanted to compare health insurance rates. We derived the health insurance rate by dividing the number of uninsured by the population. We wanted to compare non Medicaid expansion states to Medicaid expansion states. Furthermore, we wanted to try and compare health outcomes to health insurance rates. To look at health outcomes, we used rates of pneumonia from AHRQ and ER visitation rates from Georgia.

In regards to transformations, we were given the population for a state for a given year. However, in regards to the uninsured we found counts for various age demographics for both male and female and needed to sum the various counts to get a total uninsured count. Then we divided the uninsured count totals by the population to get an uninsured rate. There were many other transformations done to aid in the review of the data like transposing the data (switching columns and rows) and or changing column headers.

Furthermore, we creatively utilized lists and dictionaries in our loops to loop through states, years, and dataframes. In one case, via a loop, we stored dataframes in a dictionary and were able to simply call the dataframes stored in the dictionary via theDictionary[theDf] syntax.

In regards to cleaning data, we had several non-numeric values or missing values and something described as data statistically unreliable or DSU data in the AHRQ data. Thinking that we did not want this type of data saved in the database we first tried to set all of these cells to null or empty string. However, when we attempted to save the data to a mysql database via the use df.to\_sql we received an error. To resolve this error we converted all DSU, empty string, and or NaNs to NaN, which the df.to\_sql was then able to handle. The interesting thing about this is the code behind df.to\_sql then converted the NaNs to nulls in the mysql database.

In regards to joining data, we used a more surgical approach. We used loops and a df.loc[row][col] syntax to extract and load into a summary df. Via this method we were able join on an appropriate state and year data.

For filtering data, specifically for finding the number of uninsured to sum, we used df[col].str.contains syntax to filter on the rows we needed to sum. Here is an example of this type of code:

theUninsured=theDFs[theYear][(theDFs[theYear].index.str.contains('Estimate')) & \

(theDFs[theYear].index.str.contains('No health insurance coverage'))][f'{state}'].sum()

Loading

In preparation for loading the summarized data into the our final data store, a mysql database, we stored the data into a summary dataframe.

Todo List:

All:

* Review requirements to insure nothing missed
* Review code
  + Do one data verification
    - Eg look at data from web site and insure it correlates with final summary df

Cuong

* Do data check on ahrq and oasis data
* First draft of table code

Ted

* Go ahead and submit link
* Create mysql insertion code
* Write 1st draft of 2 sections of paper.
  + 1)sources of data 2)transformation

Jenz

* Data check on census data
* Review teds paper parts and edit as needed
* Do final data part of paper
  + Eg. Using mysql, table code, insertion code
* Send final paper to team for review by 4pm tomorrow