Medicare Part D-ETL

By Thusneem Mohamed Ismail, Eric Hagee, Godday Ogbo, and Timothy Chien

Project Description

This project is an Extraction, Transformation, and Loading (ETL) project that uses Python as the main programming language, and thus uses packages such as Pandas (for data reading/cleaning/transformation) and PyMySQL (to load to a SQL database).

This project uses two sets of data about prescription drugs paid by Medicare and/or Medicaid. Specifically, one set of data lists costs (including all sources) paid by Medicare recipients for prescription drugs. Another set is a list maintained by Medicaid as a guide for drug acquisition price unit costs. We attempt to relate these two tables with data from the web about out-of-pocket costs for Medicare recipients to create a number of tables that could be used to look at drug pricing.

Extraction

Out-of-pocket co-payments for Medicare Part D recipients do not follow a particularly simple formula. There are 4 possible basic cost shares that depend on the total cost of the prescriptions purchased through the program for the calendar year. One of the most alarming cost shares occurs in the coverage gap, also known as the “donut hole”, where the cost of the drug falls mostly on the insured with little or possibly no coverage from the insurer. On the other hand, despite being called catastrophic coverage, the cost share during that period is minimal and the burden falls mostly on the insurer. One theoretical scenario is a patient who is on a single prescription throughout the year. Would this patient end the year in the ”donut hole” or reach catastrophic coverage, for instance?

Another potential point of interest is actual acquisition costs of drugs (i.e. are providers upcharging for certain drugs more than others). Considering potential markups for cheaper vs. more expensive drugs could be an item of interest.

We essentially needed three sources of data to consider these questions. The first, contains aggregate costs (i.e. insurance and copayments) paid for prescription drugs used by Medicare Part D recipients in the year 2016. It also contains some information about drugs, such as whether the drug is an opioid or an antibiotic for instance. This was retrieved in csv format from Kaggle, at https://www.kaggle.com/cms/cms-part-d-prescriber-summary-reports-2013-2016#part-d-prescriber-national-summary-report-calendar-year-2016.csv. The data came to Kaggle from the Center for Medicare and Medicaid Services, and is thus reported by the government.

As well, we also retrieved the National Average Drug Acquisition Cost dataset from Medicaid.gov, the website for the Center for Medicaid and Children's Health Insurance Program (CHIP). This dataset has updates on the average unit cost for pharmacies to obtain the drug. The data can be exported from https://data.medicaid.gov/Drug-Pricing-and-Payment/NADAC-National-Average-Drug-Acquisition-Cost-/a4y5-998d/data.

In both of the above cases, we used Pandas to read the csv's from disk.

We obtained information on the various Medicare Part-D coverage cut-offs for different years from Q1 Group LLC (https://q1medicare.com/PartD-The-MedicarePartDOutlookAllYears.php), a group that gives information about Medicare. The data here was in tables on the web, so we used Pandas from_html function to scrape them.

Transformation

Prescriber Summary (Aggregate Cost)
For this table, we first renamed some columns, as they had unnecessary spaces on the ends of the column names. Then, we explored some of the values for the flags (ex. antibiotic). We removed one (GH65 tag) and turned the others from strings to booleans. Also, to later bin the data (e.g. donut-hole drug), we need to calculate the average yearly cost for the drug. To do so, we first calculate the average cost for a 30-day fill, and then assume twelve fills in a year (30 does not divide into 365 evenly, but we estimate here).

NADAC
In this case, the column names seem to be formatted appropriately, but the table is large as is (over 500 GB) and we don't need a lot of these columns, so they are dropped.

Then, as each drug will have a large number of updates for different dates, we really only want the most recent update. To do this, we need to sort by date after reformatting the dates to a more sortable format. We then drop duplicate names, keeping only the last date.

This particular data source is rather messy in its naming practice. For example, sometimes abbreviations for the same drug can be input differently at different dates. As well, all the names have dosages that are not in the names in the first table. This will make joining difficult. So, we attempt to split the name from the dosage. This is done by separating out any numbers, special characters, or keywords (such as "tablet"). We then save the names and dosages in separate columns. Note that this transformation actually occurs in the workflow before the duplicate entries are dropped.

The resulting simplification of the dataset is saved as a csv file for possible future use.

Coverage Table

For this table, the pandas.from_html function returns a list of any tables that are scraped. As this was the first table, we saved it as a data frame and gave it an appropriate name.

While the data comes as one html table from the website, the html table of interest is better thought of as several different tables, so we slice appropriate parts as separate data frames that can be referenced based on the type of beneficiary receiving benefits.

Binning

We used information from the Q1 data source (the web source) as bins to label the Aggregate Cost dataset, in particular the 2016 and 2019 thresholds. Assuming that the patient is paying for the prescription for the course of the calendar year, this process identifies the risk that the single drug would place a beneficiary into a particular cost sharing period.

Index for Joins

It was impractical to join the Prescriber Summary and NADAC tables as there was not a convenient common key to join on. The most logical choice was to attempt to join on the drug name. However, even after parsing the drug names for dosage to try to improve the matches, only about half of the NADAC table was able to find a match in the Prescriber Summary table.

In the interest of presenting as much data as possible, we decided to keep the two tables separate. So after separating names from dosages in each table (see above), we then saved a key of the match based on the other table's index (note that -1 indicates no match).

Column Names

To load to a SQL database, we had to format the column names, removing spaces in favor of underscores and make everything lower case.

Loading

The data was loaded to a MySQL database using the Pandas to_sql function. Five dataframes on different benefits and beneficiaries was loaded from the Coverage table along with the transformed Prescriber Summary and simplified NADAC tables for a total of 7. It does require that one run "CREATE DATABASE partD_rx_db;" prior to running the python code. In order to set up the connection the user should also create a "keys.py" file which assigns values to the "mysql" as the address to the desired server to store the data, "user" as the user name and "pw" as password.

Conclusion

This resulting tables should allow a user to estimate the risk of a particular prescription of putting one into a cost share portion of Medicare Part D based on the drug name. At the same time it should allow them to see what the average pharmacy acquisition costs. The main limitations were messy data (inconsistent names) and no convenient drug ID for matching.