**ETL Report: California Asthma data and EPA Air Quality Data**

Darrell Gerber  
February 7, 2022

**Introduction**

Set the stage. Introduce the problem that you are trying to solve. Identify sources of data. Describe why the data needs to be transformed.

**Data Sources**

* **Asthma Emergency Department Visit Rates** 
  + California Department of Public Health. (2019). Asthma ED Visit Rates by County (November 10, 2021). Retrieved from <https://data.chhs.ca.gov/dataset/asthma-emergency-department-visit-rates>. Accessed February 3, 2022.
  + Geographical Location: California by County and by zip code
  + Time Granularity: County - Annual from 2015 to 2019, Zip code – Annual from 2013 to 2019 (note, treat 2013 to 2015 as a separate dataset from 2016 to 2019 due to a change in coding)
  + Terms of Use: <https://data.chhs.ca.gov/pages/terms>
* **Air Data: Air Quality Data Collected at Outdoor Monitors Across the US** 
  + US Environmental Protection Agency. Air Quality System Data Mart [internet database] available via <https://www.epa.gov/outdoor-air-quality-data>. Accessed February 3, 2022.
  + Geographical Location: 2,498 AQS sites in 2020
  + Time Granularity: 1957 to 2021 for AQS. Daily, quarterly, and annual summaries. Sample durations from 3 minutes to 3 months.
  + Details: This API is the primary place to obtain row-level data from the EPA's Air Quality System (AQS) database.

**Extraction**

* **Asthma Emergency Department Visit Rates**
  + Go to <https://data.chhs.ca.gov/dataset/asthma-emergency-department-visit-rates>.
  + Download the dataset “Asthma ED Visit Rates by County” as a CSV.
  + Load the dataset to the Jadr-asdfasf datalake
  + In a databrick, load the CSV file into a dataframe
* **Air Data: Air Quality Data Collected at Outdoor Monitors Across the US** 
  + Refer to <https://aqs.epa.gov/aqsweb/documents/data_api.html> for instructions on the EPA Air Quality System (AQS) API.
  + Signup for an access key if needed
  + Download the annual summary AQS data for counties in California from 2015 to 2019
    - Retrieve the list of counties in California with data. The accessing URL is of the form, [https://aqs.epa.gov/data/api/list/countiesByState?email={email}&key={key}&state=06](https://aqs.epa.gov/data/api/list/countiesByState?email=%7bemail%7d&key=%7bkey%7d&state=06) (the code for California is ‘06’)
    - Retrieve the data for the selected contaminants for each county in California for the desired years. Data for no more than one year at a time can be retrieved from the API. The selected contaminants are: Lead (TSP) LC, Carbon monoxide, Sulfur dioxide, Nitrogen dioxide (NO2), Ozone, PM10 Total 0-10um STP, Lead PM10 LC FRM/FEM, and PM2.5 - Local Conditions. The accessing URL is of the form, https://aqs.epa.gov/data/api/annualData/byCounty?email={email}&key={key}&param=14129,42401,42602,44201,81102,85129,88101&bdate={bdate}&edate={edate}&state=06&county={county}

Where did you get the data from? How did you get the data? What format is the extracted data? What steps were taken to extract the data? Be sure to number steps when the order matters.

**Transformation**

* **Asthma Emergency Department Visit Rates**

1. There are null values and values of ‘0’ in the ‘NUMBER OF ED VISITS’ and ‘AGE-ADJUSTED ED VISIT RATE’ columns. Remove those rows.
2. Remove the commas in the ‘NUMBER OF ED VISITS’ column
3. Convert the ‘NUMBER OF ED VISITS’ column to type ‘integer’
4. Drop the ‘COMMENT’ column
5. Filter to extract only the ‘Total Population’ data from the ‘STRATA’ column
6. Drop the ‘STRATA’, ‘STRATE NAME’, and ‘AGE GROUP’ columns
7. Add a ‘STATE’ column with ‘CA’

* **Air Data: Air Quality Data Collected at Outdoor Monitors Across the US**

1. Add a ‘STATE’ column with ‘CA’
2. Drop the ‘Lead PM10 LC FRM/FEM’ column due to a lack of data for most counties.
3. The dataset contains many different measurements and metrics for each contaminant. Select to keep only the following:
   * + Methods
       - 'Hi-Vol - ICAP SPECTRA (ICP-MS); 0.45M HNO3 Boil30 min',
       - 'INSTRUMENTAL - GAS PHASE CHEMILUMINESCENCE',
       - 'INSTRUMENTAL - CHEMILUMINESCENCE',
       - 'INSTRUMENTAL - ULTRA VIOLET ABSORPTION',
       - 'INSTRUMENTAL - ULTRA VIOLET',
       - 'Andersen RAAS2.5-300 PM2.5 SEQ w/WINS - GRAVIMETRIC',
       - 'R & P Model 2025 PM-2.5 Sequential Air Sampler w/VSCC - Gravimetric',
       - 'Multiple Methods Used', and
       - 'INSTRUMENTAL - Pulsed Fluorescent 43C-TLE/43i-TLE'.
     + Metrics
       - 'Daily Maximum 1-hour average',
       - 'Daily maxima of observed hourly values (between 9:00 AM and 8:00 PM)',
       - 'Daily Mean',
       - 'Daily maximum 1-hour average', and
       - 'Observed Values'
     + NOTE: Be aware that capitalization matters for the names above. Also, ‘Observed Values’ is the desired metric only for ‘Lead (TSP) LC’, but not for ‘Sulfur dioxide.’
4. The dataset includes multiple readings for each county. Reduce the dataset to a county average by taking the arithmetic mean of each numeric measurement. An example snippet of PySpark code is:

from pyspark.sql import functions as F

AQIDFAgg = AQIDF.groupBy( 'state', county','year','parameter').pivot('parameter').agg(F.mean('arithmetic\_mean'), F.mean('first\_max\_value'), F.mean('ninety\_ninth\_percentile'), F.mean('standard\_deviation'), F.mean('second\_max\_value'), F.first('method'), F.first('metric\_used'), F.first('units\_of\_measure'))

The groupBy() function sets our aggregation to be by the counties, years, and parameter. The pivot() command changes the parameters from a row for each parameter to all possible ‘parameter’ values as columns. The agg() function does an aggregation on multiple columns at once. The F.first() function returns the first value found in each aggregation for the string columns.

1. Rename the columns with a ‘.’ in them to remove the period. This caused problems with later function calls in PySpark.
2. The pivot() function moved the parameters to the columns (which we need for machine learning since they will be features). However, there remains a row for each measurement with ‘null’ values in the columns not associated with that measurement. There are duplicate rows for each county/year combination. Reduce these to a single row per county/year with all of the measurement values in the appropriate column. An example snippet of PySpark code is:

AQIDFAgg2 = AQIDFAgg.groupBy('state','county','year').agg(F.first('Lead (TSP) LC\_avg(arithmetic\_mean)', ignorenulls=True).alias(LEAD\_MEAN),

F.first('Lead (TSP) LC\_avg(first\_max\_value)', ignorenulls=True).alias(LEAD\_1STMAX), F.first('Lead (TSP) LC\_avg(ninety\_ninth\_percentile)', ignorenulls=True).alias(LEAD\_99PERC), F.first('Lead (TSP) LC\_avg(standard\_deviation)', ignorenulls=True).alias(LEAD\_STD),…

(Repeated for each column in the dataframe)

The groupBy() this time is just by state, county, and year. The aggregation is to retrieve the first value found while ignoring the null values. NOTE: some cells will still have ‘null’ values because all measurement parameters aren’t available for every county in every year. The alias() function renames the columns to a shorter name.

Did you use all of the data you extracted as-is? Did you remove columns? Did you change columns’ names? Did you change your column formats? What steps were taken to get the data in a form that you could use it? Be sure to number steps when the order matters.

Since your end goal will be to load your data in SQL Server, include table mappings that identify the source data and its destination.

**Load**

* Refer to the ERD for the SQL database for the final database structure.
* The Asthma dataframe and air quality dataframe are stored in several linked tables in a SQL database. There are several key steps loading this data into the database:
  + Linking Asthma and County tables

1. Load the current County and State data sets from SQL
2. Do a left join to combine them.
3. Add an index column to the Asthma dataframe called ASTHMA\_ID.
4. Create a temporary table with a column for the ASTHMA\_ID and the corresponding COUNTY\_ID matching on county and state.
5. Do a left outer join on the asthma dataframe to add the COUNTY\_ID to the asthma dataframe.
6. Check for any null values in the COUNTY\_ID column where there isn’t a county currently in the table.
   * 1. Create a dataframe of the new counties missing from the County table in the SQL database. Append them to the SQL database skipping over “California” as this is the statewide data and not a county.
   * Linking Air Quality and Method, Metric, and Units tables
7. Go through all of the Method columns in the air quality dataframe and collect all distinct values for method names.
8. Append the dataframe of distinct method names to the Method table in the SQL database. If the write fails because a value is already there, read the current table and only append any values not already in the database.
9. Load the current Method dataset from SQL to get the current METHOD\_IDs.
10. Add an index column to the air quality dataset call AQ\_ID.
11. Create temporary tables with a column for the AQ\_ID and the corresponding METHOD\_ID matching on each method name for lead, NO2, ozone, PM10, PM2.5, and SO2.
12. Do a left outer join on the air quality dataframe to each temporary table to add the method IDs to the air quality dataframe.
13. Repeat the same process for the Metric names (storing in the Metric table in the SQL database) and Units names (storing in the Unit table in the SQL database)
    * Load the Air Quality datasets to the SQL database
14. Read the current AirQualityDataCounty table from the SQL database.
15. Reorder the air quality dataframe so the column order matches the schema of the table in the SQL database. Rename columns, if needed, to match the SQL schema.
16. Set the schema of the air quality dataframe to be the same as the schema of the table loaded from the SQL database. Note: You may also need to first cast some of the ID numbers previously added to the air quality dataframe back to integer in case they were added in a different type.
17. Append the air quality dataframe to the AirQualityDataCounty table in the SQL database. If the write fails because a value is already there, read the current table and only append any values not already in the database.
    * Linking Asthma and Air Quality datasets
18. Create a column with unique indices for the asthma dataframe (ASTHMA\_ID), if they aren’t already there.
19. Read the current AirQualityDataCounty table from the SQL database.
20. Create a temporary table with a column of the ASTHMA\_ID s and a column of the corresponding AQ\_ID for the row in the air quality dataframe with matching county and year.
21. Do a left outer join on the asthma dataframe with the temporary table on the ASTHMA\_ID to add the AQ\_ID to the asthma dataframe. The left outer join is necessary to account for the counties/years that don’t have air quality data available.
22. Read the current CAAsthmaData table from the SQL database.
23. Reorder the asthma dataframe so the column order matches the schema of the table in the SQL database. Rename columns, if needed, to match the SQL schema.
24. Set the schema of the asthma dataframe to be the same as the schema of the table loaded from the SQL database. Note: You may also need to first cast some of the ID numbers previously added to the asthma dataframe back to integer in case they were added in a different type.
25. Append the asthma dataframe to the CAAsthmaData table in the SQL database. If the write fails because a value is already there, read the current table and only append any values not already in the database.

**Conclusion**

To ensure the accuracy and viability of our analysis, we have performed three separate ETL processes to integrate and relate each of our individual datasets. In addition to cleaning and transforming the data, we have also established a relational database structure and have stored the cleaned data for later reference. By doing so, we will be better able to explore the relationships that exist between asthma-incidence rates, regional air quality, and industry activity. Further, storing our data in a normalized SQL database ensures that it will be accessible and consistent across all platforms and services that we end up using for our analysis.