ETL Report: Jadr Health Insights

Capstone Project

Abbey Guilliat, Darrell Gerber, Regina Huber, Justin Bartell
February 10, 2022

Introduction

For our capstone project, we are interested in exploring how asthma incidence rates may be impacted by environmental factors, specifically focusing on regional air quality and quantifiable measures of local industrial activity. Our goal is to develop a machine learning model that will take real-time air quality data to predict the number of asthma-induced emergency room visits for any pre-selected county. Additionally, we hope to explore the correlation between various industries and asthma incidence rates by considering industry size and revenue outputs as measures of industry activity.

We have collected three primary data sources to perform this analysis: the California Department of Public Health's Asthma Emergency Department Visit Rates, the US Environmental Protection Agency's National Air Quality Data API, and the 2012 US Census Survey of Business Owners Industry Dataset. The available asthma incidence data are annual aggregates. Therefore, while we intended to develop an ML model to make daily predictions of asthma-induced ED visits, we decided to design our analysis for an annual study. Additionally, we focus on the county level, and our ML model trains using California asthma-incidence data. Finally, we assume that the relationship between air quality and asthma-incidence rates in California are comparable national. Therefore, the model predicts annual asthma-incidence rates in countries throughout the United States, given their average yearly air quality measurements.

The cleaning and transformation of this data are crucial in ensuring the accuracy and viability of our results and analysis. These steps are vital as we are not working with ideal data and treat this exploration as a proof-of-concept. Additionally, to perform our analysis, we must connect the three disparate datasets to observe and establish the relationships between them. Further, by defining a consistent database structure, we hope to make our data more accessible and usable across various platforms and services.

PART ONE: Census Data

Data Sources:

2012 US Census Survey of Business Owners

SB1200CSA05 - Statistics for All US Firms by Industry, Gender, and Receipts Size of Firm for the US and States: 2012. (2015, December 15). Retrieved February 4, 2022, from https://data.census.gov/cedsci/table?q=SB1200CSA05&tid=SBOCS2012.SB1200CSA05.

- Contains information on number of business firms, sales and receipts of business firms, number of employees for business firms, and business firm owner gender demographics
- Data is aggregated by US counties and states and can be filtered to show firm totals or totals by specific industries

Extract:

Data was collected from https://data.census.gov/cedsci/ and we specifically extracted the data set titled: SB1200CSA05 - *Statistics for All US Firms by Industry, Gender, and Receipts Size of Firm for the US and States: 2012*. The data was downloaded as a Zip file containing a data CSV, a metadata CSV, and an HTML text file. For our analysis, we have primarily utilized the data CSV but have also referenced the metadata CSV to better understand the information contained within the data columns. After downloading the data we:

1. Uploaded it to our shared project data lake
2. Created a data brick specifically for cleaning/transforming the census data
3. Established a mount point connecting to our data lake
4. Read the data CSV into a spark data frame

Transform:

1. Read data into Pandas data frame using .toPandas()
2. Drop top row to remove duplicate header values
3. Filter data frame to get firm totals only by industry, not demographic breakdowns by sex

```
df = df[df['SEX'] == '001']
```

   a. Don't forget to reset index after this step!
4. Drop columns unnecessary for analysis:

```
['GEO_ID','SEX','SEX_LABEL','RCPNOPD_S','FIRMNOPD_S','PAYANN_S','E
MP_S','RCPPDEMP_S','FIRMPDEMP_S','RCPALL_S','FIRMALL_S','NAICS2012
','NAICS2012_F','RCPSZFI_LABEL','RCPSZFI','PAYANN','index','GEO_ID
_F']
```

5. Split ['NAME'] column into ['COUNTY'] and ['STATE'] columns

```
df.NAME.str.split(',',expand=True)
```

   a. Drop `['NAME']` and reorder columns so that `['COUNTY']` and `['STATE']` are first columns after index

    b. Order:
    `[['COUNTY','STATE','YEAR','NAICS2012_LABEL','FIRMALL','RCPALL','FIRMPDEMP','RCPPDEMP','EMP','FIRMNOPD','RCPNOPD']]`
6. Drop duplicate and blank rows from data frame
7. Replace 'S' fields with value of 0 for columns `['RCPALL']`, `['RCPPDEMP']`, and `['RCPNOPD']`
8. In `['EMP']` column, replace non-castable int types to the median value of the respective range
    a. i.e. `df['EMP'].replace('100 to 249 employees', '175', inplace=True)`
9. Cast `['FIRMALL']`, `['FIRMPDDEMP']`, `['FIRMNOPD']`, `['RCPALL']`, `['RCPPDEMP']`, `['RCPNOPD']`, and `['EMP'] .astype('int')` and then confirm proper data types
10. Replace fields equal to 0 with `np.nan`
11. Reset index
12. Iterate through `['COUNTY']` column to remove the last 7 characters from each field (i.e. ' County') so we are left with only the name of each county
13. Drop any extra index columns if necessary

Load:

1. Create new data frame for state information
    a. Contains one column for state name + one column for state abbreviation
    b. Make sure to drop duplicate values if necessary (51x2)
2. Write state data frame to SQL table `dbo.State` in order to assign unique identifying key
    a. To prevent adding duplicates to SQL database, first load in existing data from `dbo.State` and read this into a spark data frame
    b. Set this loaded dataframe to equal only `[['STATE_ABBR','STATE_NAME']]` columns
    c. Convert/ensure that your state data frame is a spark data frame
    d. Use the `.subtract(loaded_df)` function on your original state data frame
    e. Write the result of the previous step into the dbo.State table
3. Read data from SQL `dbo.State` table back into data brick and create a for loop to iterate through main data frame
    a. Currently, main data frame contains state name information
    b. In loop, match state name values and convert main data frame fields to corresponding `STATE_IDs`
4. Repeat steps 1-3 for county information
    a. In step 3 loop, be sure to match both `COUNTY_NAME` and `STATE_ID` information before converting main data frame county name information to corresponding `COUNTY_IDs`
5. Drop the `['STATE']` column from main data frame
6. We now have a structure that almost matches the `dbo.CensusIndustryData` table we devised in our SQL database ERD
    a. Including unique `COUNTY_ID` foreign keys that connect to both the `dbo.County` and `dbo.State` tables (!!)
    b. Convert main data frame into a spark data frame to prepare for SQL loading
7. Write main data frame to the SQL `dbo.CensusIndustryData` table

8. Congratulations! You've successfully extracted, transformed, and loaded the data into the SQL database.

PART TWO: Asthma and Air Quality Data

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 10, 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.

Extract:

- **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 (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}

Transform:

- **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' values 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.' Removed the rows with 'Sulfur dioxide' and 'Observed Values'.
    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('arithme
tic_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.

5) Rename the columns with a '.' in them to remove the period. This caused problems with later function calls in PySpark.
6) 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.

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.
    i) 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
    1) Go through all of the Method columns in the air quality dataframe and collect all distinct values for method names.
    2) 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.
    3) Load the current Method dataset from SQL to get the current `METHOD_ID`s.
    4) Add an index column to the air quality dataset call `AQ_ID`.
    5) 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.
    6) Do a left outer join on the air quality dataframe to each temporary table to add the method IDs to the air quality dataframe.
    7) 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
    1) Read the current `AirQualityDataCounty` table from the SQL database.
    2) 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.
    3) 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.
    4) 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
    1) Create a column with unique indices for the asthma dataframe (`ASTHMA_ID`), if they aren't already there.
    2) Read the current `AirQualityDataCounty` table from the SQL database.
    3) 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.

4) 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.
5) Read the current `CAAsthmaData` table from the SQL database.
6) 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.
7) 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.
8) 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.

Congratulations! You've successfully extracted, transformed, and loaded the data into the SQL database.

PART THREE: Air Quality Data – Kafka Producer and Consumer

Data Sources:

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 10, 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.

Extract:

Kafka Producer:

- **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 all counties for 2021
    - Retrieve a list of the state with data available. The accessing URL is of the form, https://aqs.epa.gov/data/api/list/states?email={email}&key={key}. Loop through all of these states doing for each:
      - Retrieve the list of counties in that state with data. The accessing URL is of the form, https://aqs.epa.gov/data/api/list/countiesByState?email={email}&key={key}&state={state}. Loop through all of these counties doing for each:
        - Retrieve the data for the selected contaminants for 2021. 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=20210101&edate=20211231&state={state}&county={county}

Kafka Consumer:

- **Air Data: Air Quality Data Collected at Outdoor Monitors Across the US**
  - Connect to the Kafka server
  - Gather all data on the `'jadr-AQI'` topic.
  - Removed duplicate rows using the `.distinct()` function.

Transform:

Kafka Producer:

No transformation done on the data in the Kafka producer. The data is fed straight into the data stream as pulled from the EPA API without modification, one row at a time. See Load below.

Kafka Consumer:

- **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'` values 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.'` Removed the rows with `'Sulfur dioxide'` and `'Observed Values'`.
  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('arithme
tic_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.

5) Rename the columns with a '.' in them to remove the period. This caused problems with later function calls in PySpark.
6) 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.

Load:

Kafka Producer:

- Send each response to a request from the EPA API directly to the Kafka server under the topic, `'jadr-AQI'`. Responses are for each request of data for a county.

Kafka Consumer:

Consumer to data lake

- Write the cleaned air quality data to the data lake as a JSON file.

Data lake to SQL

1) Read in the JSON file from the data lake.
2) Remove duplicate rows using the `.distinct()` function.
3) Replace all `null` values with "".
4) Load in the `Method`, `Metric`, and `Unit` tables from the SQL database.
   a) Replace all `null` values in the `_NAME` column of each with "".

5) Load the current `State` table from SQL
6) Do a leftouter join of the air quality dataframe to the State dataframe matching on the `STATE_NAME`.
    a) Check if there are any null values in the `STATE_ID` column. These are states not already loaded in the `State` table on the SQL database.
    b) Append the missing states to the `State` table.
    c) Load the `State` table from the SQL database again.
    d) Do another left-outer join between the air quality dataframe and the State dataframe matching on the `STATE_NAME`.
7) Drop the `'state'`, `'STATE_ABBR'`, and `'STATE_NAME'` columns.
8) Load the current `County` table from the SQL database.
9) Do a left-outer join between the air quality dataframe and the County dataframe matching on the `COUNTY_NAME` and `STATE_ID`.
    a) Check if there are any null values in the `COUNTY_ID` column. These are counties not already loaded in the `County` table on the SQL database.
    b) Append the missing states to the `County` table.
    c) Load the `County` table from the SQL database again.
    d) Do another left-outer join between the air quality dataframe and the County dataframe matching on the `COUNTY_NAME` and `STATE_ID`.
10) Drop the `'county`, `'STATE_ID`, and `'COUNTY_NAME'` columns.
11) Loop through the `'Lead'`, `'NO2'`, `'Ozone'`, `'PM10'`, and `'SO2'` pollutants .
    a) Fill in all null values in the air quality `Method` column with "".
    b) Do a left-outer join between the air quality dataframe and the Method dataframe matching on the `Method` column and the `METHOD_NAME`.
    c) If there are null values in the `METHOD_ID` column there are methods in the dataframe not yet loaded in the `Method` table on the SQL database.
        i) Filter the rows with `null` values
        ii) Select just the `Method` column
        iii) Rename the `Method` column to `METHOD_NAME`
        iv) Get the distinct values on that column.
        v) Replace the "" values with `None` to add the `nulls` back.
        vi) Append the new method values to the `Method` table in the SQL database.
        vii) Do a left-outer join between the air quality dataframe and the Method dataframe matching on the `Method` column and the `METHOD_NAME`.
    d) Drop the `METHOD_NAME`, and `Method` columns. Rename the `METHOD_ID` column to `{pollutant}_METHOD_ID`.
12) Repeat the previous step for the `Metric` and `Units` columns and tables.
13) Load the `AirQualityDataCounty` table from the SQL database.
14) Drop the `AQ_ID` from the dataframe.
15) Reorder the columns in air quality dataframe we have been working with match the column order in schema of the `AirQualityDataCounty` dataframe.
16) Set the schema of the air quality dataframe to the same schema as the `AirQualityDataCounty` dataframe.
17) Call the `dropDuplicates()` function on the air quality dataframe after subtracting the loaded `AirQualityDataCounty`. This will remove any air quality data already in the SQL database. E.g. `df_toload = df.dropDuplicates(subset=["COUNTY_ID","YEAR"]).subtract(df_loaded).`

18) Append the data not already in the SQL database to the `AirQualityDataCounty` table.

Congratulations! You've successfully extracted, transformed, and loaded the data into the SQL 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 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 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 we use for our analysis.