**Group 1 ETL Project Report Q&A**

* What data sources we chose, and why, and the ETL steps:

The group decided to combine average auto insurance, by location, with weather data. The first step was to collect average auto insurance data. Six tables, which included average auto insurance rates, were scraped from CarInsurance.com and AutoInsurance.org. After importing the tables as dataframes they were combined into a master table. In order to be utilized, the dataframe needed to be cleaned of duplicate and missing data. For example, some of the rows did not include ‘state’ data. To fill in the missing data, two additional sources were used:

* + - Zip-Codes API (<https://api.zip-codes.com>)
    - State Abbreviations Code Table (<http://app02.clerk.org/menu/ccis/Help/CCIS%20Codes/state_codes.html>).

The Zip-Codes API was used to find the state abbreviation for the zip codes that were missing a state. The State Abbreviations Code Table was used to change the state abbreviations to the full state name. The next step was to extract data for each zip code from Open Weather Map (<https://openweathermap.org/>). In this step we extracted a list of zip codes for the areas where we had auto insurance data and then we ran an API call requesting weather data for each zip code.

* Explanation of the types of transformations performed, and why:

When cleaning up all the tables we needed to replace some of the index columns, originally some of the index columns used the zip codes values. The zip codes were removed from the index and replaced with a list we created by appending values incrementing by one. This was a critical step since there were still some duplicate zip codes after running a drop\_duplicates function on the dataframe. The remaining duplicates were not deleted because they only shared the same zip code but the other columns were different. Having an index with values incrementing by one allowed us to run a for loop to find the remaining duplicate data.

Before the data could be imported into PostgreSQL, we needed to convert the Average Auto Insurance Price column from an object and into an integer. In order to do this, we needed to remove the $ and commas from the Average Auto Insurance Price column.

After we introduced our weather data. We went through some data cleaning steps where we dropped any instances where we did not have weather data (null values) for our dataframe.

* Why you chose the type of final database:

As a group we felt much more comfortable with what has been learned to date, the types of exercises, and language syntax regarding PostgreSQL over Mongo. It also makes sense in light of our data being quite structured and numerical. If the data was more unstructured and fractured, a non-SQL database would’ve made more sense.

* Schema of the tables/collections in the final database:
  + “app.quickdatabasediaphragms” was utilized to create the ERD. It is located in the group repository for viewing at: /Resources/Auto\_Insurance\_Weather\_data\_ER\_Diaphram.PNG
* Flask API:

The Flask API was first designed and utilized with .csv data that was scraped and cleaned (via the process described above). A home route was arranged and explains that the user can search the data by inputting: Zip code, City, or State – by the URL’s shown. They can also display all the data, if so chosen. Once the data was fully cleaned and we had a working flask app – a PostgreSQL database was manufactured to arrange the back end. A copy of the functioning flask app was created (app\_SQL.py) as the new working copy that creates the engine which pulls from Postgres. We were fortunate enough to have Emeka in our group, who had previously setup an AWS Postgres server account. Utilizing the correct login credentials, we were able to access this server and forgo a build of the server locally on each of our machines. There is a transformation step within the flask app: a join utilizing sqlalchemy in line 48 allows a single dataframe to be searched by the user.

* Hypothetical use cases for our database:

This data could be used by insurance companies and/or customers to determine if weather affects insurance rates and evaluate their business strategy. For example, an insurance company may use the average auto insurance rate to determine if they are charging too much or not enough in areas with a lot of rain. If their rates are too high, this may explain why their business is low in these areas.

From the customer side: Maintaining car insurance is not only a necessity for property protection, but it’s also the law! How is one to know if her current rate is a fair one? Many companies will provide quotes, but that often comes with baggage such as speaking with nagging representatives, giving up personal information, and of course the endless barrage of advertisements that result in a simple inquiry with one of these firms. Our application can be queried hassle-free. It can return car insurance rates by City, State, or Zip Code. It also displays the weather in these areas to the car insurance rates themselves as a duel function: One can trend this over analyze if certain weather patterns correlate (obviously inversely) to car insurance rates, or it serves as a simple function of displaying this data as a courtesy. The first of these we find quite interesting. This data could be collected and visualized overtime as means of research as well. One would tend to assume that the harsher the weather to an area, the higher the car insurance premiums would run.

Further work could be done on the project to pull in other pieces of information such as the locations of Auto dealers in the zip code. Locations of Auto mechanic shops in the zip code of choice to create a one stop shop to get all your automotive information for a desired zip code.