Matthew Padgett

ETL Project

Sources:

<https://www.kaggle.com/the-guardian/olympic-games?select=winter.csv>

* Data contains a country dictionary as well as medal winners from all Olympics games from 1896-2014.

Olympic Games Data Report

Extract:

Browsed through Kaggle.com to find any specific data that sounded fun and interesting. Discovered Olympics data that spanned over almost 200 years and looked at both the summer and winter Olympics. The data was specific to the medals won throughout these games and the recipients. The data came in the form of Comma Separated Values files where there was One CSV for the dictionary of countries, one for the summer Olympics, and one for the winter Olympics. Once the CSVs were downloaded, we placed them into a Resources folder located within an ETL Project directory. With the files now available locally, we could now pull them into a Jupyter Notebook to start to Transform the data. In our Jupyter Notebook, we navigated to each CSV through file paths that were defined individually and then read the CSVs into a Data Frame.

Transform:

Using the .read\_csv function, the CSV files were read into individual data frames. From here we could begin to transform the data starting with the dictionary of countries. This data set includes information like the country’s name, population, and GDP per Capita. However, the only information that is used throughout the medal data is the country codes, so we decided we only wanted to use the dictionary of countries as a way to cross reference the country’s name to their code. To do this, we simply created a new variable to represent the cleaned data frame and then specified the two columns we wanted to see which were ‘Country’ and ‘Code’.

Next, we moved on to the summer and winter data set where there was a lot more data to look at. Both datasets included the year the games took place as well as the city where they were held. The rest of the data was specific to medals that were won such as the athlete’s name, the medal level, the event, the athlete’s gender, and the overall sport. We wanted to narrow down the data to only look at a specific part, and we decided on only looking at gold medal winners that were from the United States. Using our dictionary of countries, we used the .loc function to locate the row where the ‘Country’ == “United States” which gave us the country code. Once we knew that, we could use the .loc function again except this time we used it on the summer data to only show rows where the ‘Country’ == “USA” and the ‘Medal’ == “Gold”. Because some of the columns referred to information that was very broad like the City and Discipline, there was a lot of repeating information for these columns. We didn’t feel like this was as important, so we filtered the data set to only show us the Year, Athlete, Country, Gender, Sport, and Event. This process was repeated on the winter data set.

One thing to note during this transformation process is that we considered looking at only a specific year of the data set. For example, using a year variable with the .loc function, we could specify to only look at the 2000 Olympic games. However, we decided that we wanted to take a step back and gather data that was just a little broader that could then be filtered once in the database if needed.

Now we had a transformed summer data set and a transformed winter data set, but we wanted to combine these two to make just one combined data set. In order to accomplish this, the .append function was used to append the winter data set to the summer data set. Originally, we thought it would be some type of join, but using pandas we were able to simplify the solution. An aspect of transforming the data that we didn't realize until we got to the Extract portion of the project was that the columns needed to match the name of the columns in our PostgreSGL database in order to have the correct relationship. We then went back and used the .rename function to rename the columns in both the countries and combined games data frames.

Load:

In order to load the data into a database, we created a local connection to the PostgreSQL database within the Jupyter Notebook using sqlalchemy. In PostgreSQL itself, we created the olympics\_db database as well as the ‘country\_info’ and ‘usa\_golds’ tables. We created the tables so that the ‘code’ was a Primary Key in the ‘country\_info’ table and also a Foriegn Key in the ‘usa\_golds’ table. Using the .to\_sql function, we could insert the data in our pandas dataframes into the tables of our database. We were able to see the inserted data using a SELECT statement in both the Jupyter Notebook and the PostgreSQL server.