ETL project Report

By
Shahmeer Amjad
Steven Wang
Ruolan Fan

PURPOSE:

The purpose of this report is to analyze ridesharing data, and vehicle prices data in the city of Boston, Massachusetts. We aimed to also incorporate weather data and essentially merge average weather data for the corresponding date of analysis for each dataset.

To conduct this analysis, we had to extract the data from various sources, clean it up and transform it into the required format. Once that was achieved, we had to establish a connection and load our final datasets on a PostgreSQL database.

Extract:

We obtained the rideshare data and weather data for the city of Boston from Kaggle. This dataset comprised of 10 variables of both numerical and categorical data, including a UNIX timestamp, which is essential for our analysis. From the same Kaggle project we obtained a weather csv file which included observations for the year 2018.

To obtain the available vehicles and their prices, for the city of Boston, we scraped data from craigslist and obtained data from vehicle ads from December 13th to December 16th inclusive.

Note: The information we scraped from craigslist was from December 13th to December 16th 2019.

Transformation:

The cab dataset had a total of 693,071 observations, and after removing observations with missing values we removed 55,095 observations. Additionally, we had to transform the timestamp from milliseconds to a date using the ‘strftime’ function and extracted a new column for ‘occurrence_hour’ from the timestamp.

Rideshare Data

For the weather data, we grouped observations by date, and took average values for the following columns: ‘temperature, rain, humidity’ to ensure data consistency when we merge weather data with the cab_data, and ‘vehichle_prices data’

In order to merge weather data with the cab data, we had to ensure that both ‘date’ columns had the same column name and were the same data type, specifically a datetime object. Once we converted the datatypes, we did an inner join on the ‘date’ variable to obtain the cabweather.csv

Web Scraping from Craigslist

Methodology:

Library: we have used the BeautifulSoup, requests, splinter and time library to obtain the data from Craigslist, Boston.

Target URL : *https://boston.craigslist.org/search/sss?query=cars*.

Process:

In order to acquire the requested variables, we use ‘requests’ to access the target url, and used ‘BeautifulSoup’ to acquire the variables’ value, and finally use ‘splinter’ for page flipping. The major factor to consider here was that we had to set up a wait time between the page flipping gap otherwise we kept receiving an error.

When we accessed the target URL, we got more than 3000 listing results for the car sales for the past week (specifically December 13th to December 16th 2019 inclusive). In order to obtain a consistent format for the ‘date’ variable we converted the timestamp which was ‘YY-MM-DD HH-MM’ to YY/MM/DD’ format.

We also had to change the data type of the price variable to numeric, removed the ‘$’ sign and further change it to a float data type. To maintain data consistency, we changed the year from 2019 to 2018 in order to match with the weather data. We then saved this dataset as carlisting.csv

As before, we merged the dataset with ‘avg_temp, avg_humidity, avg_rain’ for the specific date in question and did an inner join on the ‘date’ variable. Finally, we saved the csv as ‘car_listing_weather.csv’.

Additionally, we created a new dataframe from the ‘car_listing’ dataset and created a new dataframe which reflected totalnumber of car listings, average listing price and further merged with the weather data as we did before by doing an inner join, on the ‘date’ variable. We saved the file as ‘carlisting_weather_grouped.csv
Below is a snippet of our web scraping code:

Code Snippet for web scraping:

```
for x in range(24):

    html = browser.url
    response = requests.get(html)
    soup = bs(response.text, 'html.parser')

    results = soup.find_all('li', class_="result-row")

    for result in results:
        try:
        # Identify and return title of listing
            title = result.find('a', class_="result-title").text
        # Identify and return price of listing
            price = result.a.span.text
        # Identify and return link to listing
            link = result.a['href']
        # Identify and return the date
            date = result.time['datetime']

        # Print results only if title, price, and link are available
            if title and price and link and date:

                print('--------------')
                print(title)
                print(price)
                print(date)
                print(link)
            #Add the result to list
                titles.append(title)
                prices.append(price)
                dates.append(date)
                links.append(link)

        except AttributeError:
            print("no item found")
        except:
            print("unknown error")

    time.sleep(5)
    browser.click_link_by_text('next > ')
```

Loading:

Finally, once we obtained our final three datasets, we loaded them on PostgreSQL as three different tables in the schema. To do this we had to first read the csv files and establish an engine and use it to to connect to PostgreSQL server. We also included an ‘if exists’ append clause in the code, to ensure that in the future we can update the datatables from our csv, if there are new entries in the dataset.