PROJECT 2: ETL

Group 2:

Ruby Sanchez

Vilma Diaz

Andrew Paton

Karina Horna

Introduction:

We've all heard that once you drive a vehicle off the dealership lot it depreciates drastically and immediately. We thought it was interesting that in 2021 the value of used vehicles are retaining their value and selling for higher prices than what was typical in prior years. This piqued our interest and thus we worked to gather historical vehicle data and the most current pricing data to build our database for future analysis.

The purpose of this project was to extract data from multiple data sources, transform the data into a usable and meaningful format, then load the data frames into tables within our database.

The team gathered historical data via Kaggle which was available in a csv file. Also, we built a web scrape program that gathered current data from truecar.com. With data from these different sources we determined that the transformation of the data would be most efficient using Pandas. The final step was to load the data to PostgreSQL where future analysis can be conducted.

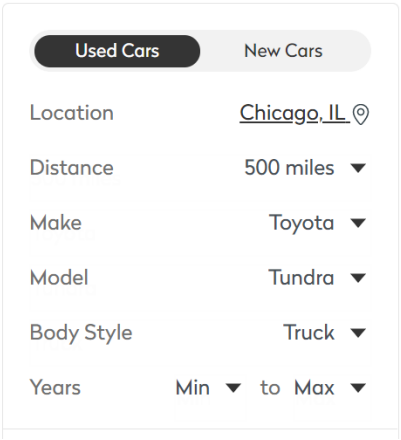
Within Github we have stored the following:

- Historical Data in a csv format (from Kaggle)
- CSV file generated from the web scrape program
- .ipynb file which includes the scrape program code and the ETL code
- .sql file which includes the queries used to establish the database tables

Extract:

Scrape Program

We used BeautifulSoup to scrape the data for trucks currently for sale within 500 miles of Chicago, Illinois. We chose this location to match the historical data set of truck sales in Illinois from 2018-2020.



To build the scrape we built a nested “for” loop to iterate saved data for each truck then changing to the next page of listings. We created an empty list to contain the data then assigning it as rows to a Pandas data frame.

Scrape Code:

Dependencies such as BeautifulSoup

```
# Dependencies
from bs4 import BeautifulSoup
from urllib.parse import urlparse
import urllib.request
import requests
import pandas as pd
import time
import re
import json
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import time
```

Web scrape Variables

We noticed each page URL had a page number, therefore allowing us to use a loop number to change the page and download all the trucks

```
# Scrape Program

base_url = 'https://www.truecar.com/used-cars-for-sale/listings/toyota/tundra/body-truck/location-houston-tx/?accidentHistory
tail = '&searchRadius=500'
page = 0
loops = 50

header = ['Brand','Model','Year','Trim','Drive','Description','Exterior Color','Interior Color','Miles','Price']
rows=[]
truck_data = pd.DataFrame(data=rows,columns=header)
#rowss= []
#rows1 = []
#trucks_data1 = pd.DataFrame(data=rows1,columns=header1)
```

Web Scrape Main For Loop

As mentioned, we changed the URL by the loop number and saved all the trucks listed on that page in "div_list"

```
for x in range(loops):
    page = x+1
    new_url = base_url+str(page)+tail
    html = requests.get(new_url)
    soup = BeautifulSoup(html.text,'lxml')
    #header = ['Brand','Model','Year','Trim','Drive','Description','Exterior Color','Interior Color','Miles','Price']

    div_list = soup.find_all('div', class_='card-content vehicle-card-body order-3')

    #rowss = []
    #rows = []
```

Web scrape Nested For Loop

The nested loop saves each truck's information by calling the loop number record from the "div_list". A try and except function was used in case information wasn't present.

```
for x in range(len(div_list)):

    # Start Row Addition----------------
    row = []

    # Call Truck HTML Data----------------
    truck = div_list[x]

    # Save Brand Data----------------
    try:
        brand = soup.find('span', class_='vehicle-header-make-model text-truncate').text
        num_brand= (len(brand))-(brand.find(" "))
        brand1 = brand[:-num_brand]
        row.append(brand1)
    except:
        row.append(np.nan)

    # Save Model Data----------------
    try:
        num_model = (brand.find(" "))+1
        model = brand[num_model:]
        row.append(model)
    except:
        row.append(np.nan)

    # Save Year Data----------------
    try:
        year = truck.find('span', class_='vehicle-card-year font-size-1').text
        row.append(year)
    except:
        row.append(np.nan)

    # Save Trim Data----------------
    try:
        desc = truck.find('div', class_='font-size-1 text-truncate').text
        num_trim = (len(desc))-(desc.find(" "))
        trim = desc[:-num_trim]
        row.append(trim)
    except:
        row.append(np.nan)

    # Save Drive Data----------------
    try:
        num_drive = (len(desc))-(desc.rfind(" "))-1
        drive = desc[-num_drive:]
        row.append(drive)
    except:
        row.append(np.nan)

    # Save Description Data----------------
    try:
        desc_mid1 = desc[(-num_trim)+1:]
        desc_mid2 = desc_mid1[:-num_drive]
        desc_mid3 = desc_mid2.strip()
        row.append(desc_mid2)
    except:
        row.append(np.nan)

    # Save Exterior Color Data----------------
    try:
        colors = truck.find('div',class_='vehicle-card-location font-size-1 margin-top-1 text-truncate').text
        num_col = (len(colors))-(colors.find(","))
        ext_col = colors[:-num_col]
        row.append(ext_col)
    except:
        row.append(np.nan)
```

Truck Historical Sales Data from Kaggle

We found historical truck data of all sales in Illinois from 2018-2020 to compare with current listings. The data was stored as a CSV within the same repository as our web scrape program. We read the file using pd.read then proceeded to the Transform stage.

Transform:

After acquiring the data from both our web scraper program and Kaggle, we further transformed it into similar tables layouts.

Web Scrape table example:

	Brand	Model	Year	Trim	Drive	Description	Exterior Color	Interior Color	Miles	Price
0	Ford	F-150	2015	XLT	4WD	SuperCrew 5.5' Box	Gray exterior	Black interior	38622	37499
1	Ford	F-150	2021		Sport		Black exterior	Black interior	3729	41590
2	Ford	F-150	2016	High	4WD	Country Crew Cab Short Box	Black exterior	Brown interior	69730	37958
3	Ford	F-150	2013	XLT	4WD	SuperCrew 5.5' Box	Black exterior	Gray interior	116165	19995

When reviewing the data, we found that some rows had missing values. This was evident by the various counts in values for the column headers.

	Brand	Model	Year	Trim	Drive	Description	Exterior Color	Interior Color	Miles	Price
count	9903	9903	9903	9903	9903	9903	9903	9903	9903	9891
unique	10	27	26	102	54	387	15	9	9435	4380
top	Ford	F-150	2019	XLT	4WD	SuperCrew 5.5' Box	White exterior	Black interior	17883	39998
freq	2973	2523	2366	1436	7534	1832	2696	6209	4	50

To remove incomplete data, we used “dropna”

```
# Drop Rows

truck_data.dropna(axis=0, how='any', inplace=True)
```

To trim down the data set to just relevant columns we created a data frame that included just the columns that we were interested in.

```
truck_data = truck_data[['Brand',
                         'Model',
                         'Year',
                         'Miles',
                         'Price'
                        ]]
```

:

	Brand	Model	Year	Miles	Price
0	Ford	F-150	2017	36956	47499
1	Ford	F-150	2012	60579	31990
2	Ford	F-150	2019	39058	39997
3	Ford	F-150	2013	116165	19995
4	Ford	F-150	2010	157719	13495

In order to prepare our data for the load phase, we set the index as our unique id.

```
# Set index
truck_data.set_index("id", inplace=True)

truck_data.head()
```

Additionally, we renamed columns in order to fit the format of our PostgreSQL database tables. This allowed us to complete the loading of the data into our database.

```
# Rename the column headers to match the format of the SQL table

truck_data = truck_data.rename(columns={'Brand':'brand',
                                        'Model':'model',
                                        'Year':'year',
                                        'Miles':'miles',
                                        'Price':'price',}
          )

truck_data.head()
```

Load:

The final database was created in PostgreSQL 14. Before the loading of the data could happen, we had to create a database and establish the tables in which the data would be stored.

Database name: vehicle_data

Table names: ['historical_data', 'current_data']

The next step was to establish the connection between our code used to extract and transform the data and the PostgreSQL database where we wanted the data stored. Once we made the connection and established that it was properly working, we were able to successfully load our data.

Potential Analysis:

Now that the database is created, we have the ability to query both the historical data and the current data for future analysis. We could use this data to understand the price escalation seen in today's auto industry. There are many current issues that we suspect are leading to an increase in used vehicle prices, such as supply chain shortages or inflation. Once we can complete a thorough analysis of the data in our database then we could make an informed decision about future analysis and areas to research.