Extraction, Transformation, and Load (ETL) Specification

Used Car Emissions

Table of Contents

[Revision History 3](#_Toc16941354)

[1. Introduction 4](#_Toc16941355)

[1.1 Summary 4](#_Toc16941356)

[1.2 Scope of Initiative 4](#_Toc16941357)

[1.3 Resources 5](#_Toc16941358)

[2. Transformation Component Documentation 6](#_Toc16941359)

[2.1 Data Import/Extract 6](#_Toc16941360)

[2.2 ETL Data Flow 6](#_Toc16941361)

[3. Exhibit 1 10](#_Toc16941362)

[4. Exhibit 2 13](#_Toc16941363)

# Introduction

The purpose of this Extraction, Transformation, and Load (ETL) Specification Document is to capture details that pertain specifically to ETL development to be used by the developer if and when this process is replicated.

## Summary

Address in the section:

* Potentially the business owner (The Client) of this project is an independent automobile service center that specializes in smog testing and makes the necessary repairs for those cars that fail the emission test (smog check) or are likely to fail soon.
* The target auto repair centers are independently owned repairs shops that used car owners are likely to take their cars to for emissions compliancy and related repairs and services.
* By analyzing the current mix of used cars listed for sale and their CO2 emission rate (CO2\_tailpipe\_Gpm), we would connect the auto centers who would proactively market their services to car owners who are likely to bring their purchased car to auto repair centers so they can get their cars registered.
* We have not interviewed our target market to determine how they want to receive the information and at what granularity? So until that is determined, we will provide a list of used cars currently on the market sorted in descending order by CO2\_tailpipe\_Gpm with VIN information, so they can decide if they want to contact the seller and pay him/her to advertise their auto center once they know who the buyer is.
* Similarly, we need to perform focus groups and determine how often the data base is updated. Since this information can be obtained in real-time, we would discuss the business model with interested parties based on their capacity and ability to reach used car owners.
* This is the first time we are integrating the used car data and emission data to analyze and prepare reportsfor our target market.

## Scope of Initiative

Address in this section:

* The information from the sources is integrated periodically (frequency is to be determined). The summary report is generated (or updated) after each run. The report is presented to our client(s) either via electronic file or on a web portal, where each paying client will access the information using their log-in credentials.
* Our expected business objective is to create a recurring revenue business service (subscription service) by providing real time on-line service to paying customers.
* To provide this service we need real-time access to [**Million Used Car Listings**](https://www.kaggle.com/jpayne/852k-used-car-listings) and extract the information at least once a day. Real-time access to the other data source, [**Fuel Economy Web Services**](https://www.fueleconomy.gov/feg/ws/index.shtml) is not required at this time.

## Resources

1. As stated earlier, the Business Clients are owners of the auto repair centers who need to bring in more business and are willing to proactively market their services to people who are most likely to need their services soon; i.e., people who would buy a used car.
2. The following are the systems needed:
   * Python Environment
   * Pandas Libraries and Jupyter Notebook
   * PostgreSQL and PgAdmin 4
3. The data sources that are needed for this ETL are:
   * true\_car\_listings.csv from [**Million Used Car Listings**](https://www.kaggle.com/jpayne/852k-used-car-listings)
   * vehicles-2.csv from [**Fuel Economy Web Services**](https://www.fueleconomy.gov/feg/ws/index.shtml)
4. In addition to the above data sources, a tertiary source is needed to transform the following values of the data in true\_car\_listings.csv, by using the VIN decoder API json to get accurate ‘make’ , ‘model’, ‘year’ for each listing in the database. The VIN decoder API json is:
   * [https://vpic.nhtsa.dot.gov/api/vehicles/decodevin/**{VIN}**?format=json](https://vpic.nhtsa.dot.gov/api/vehicles/decodevin/%7bVIN%7d?format=json)

where {VIN} is fetched from the Vin column of true\_car\_listings.csv

# Transformation Component Documentation

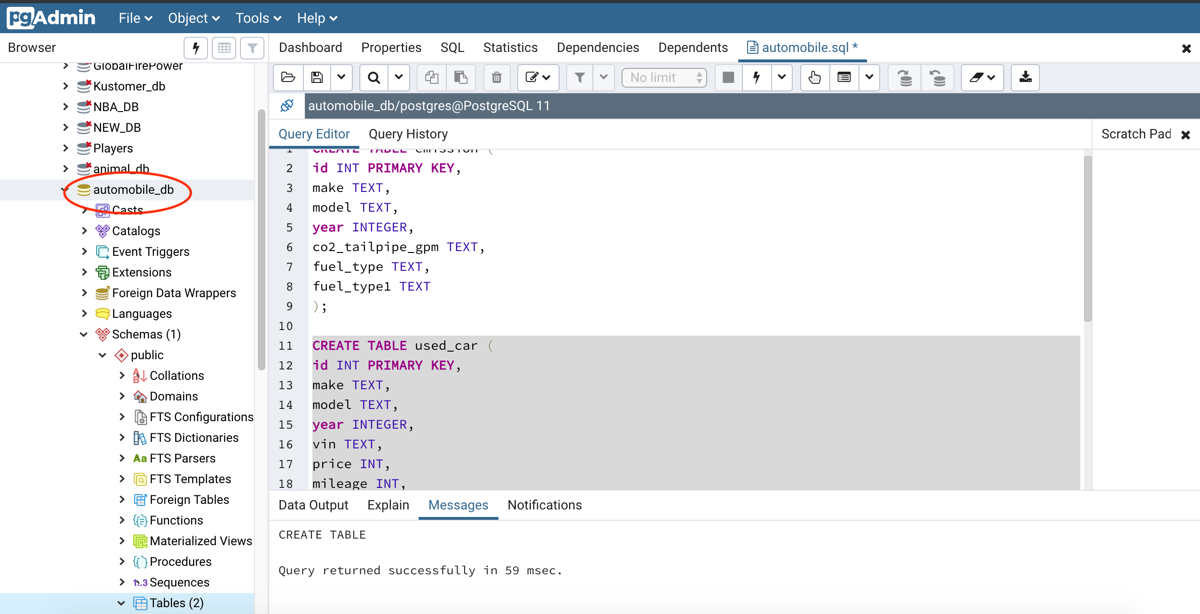
## Data Import/Extract

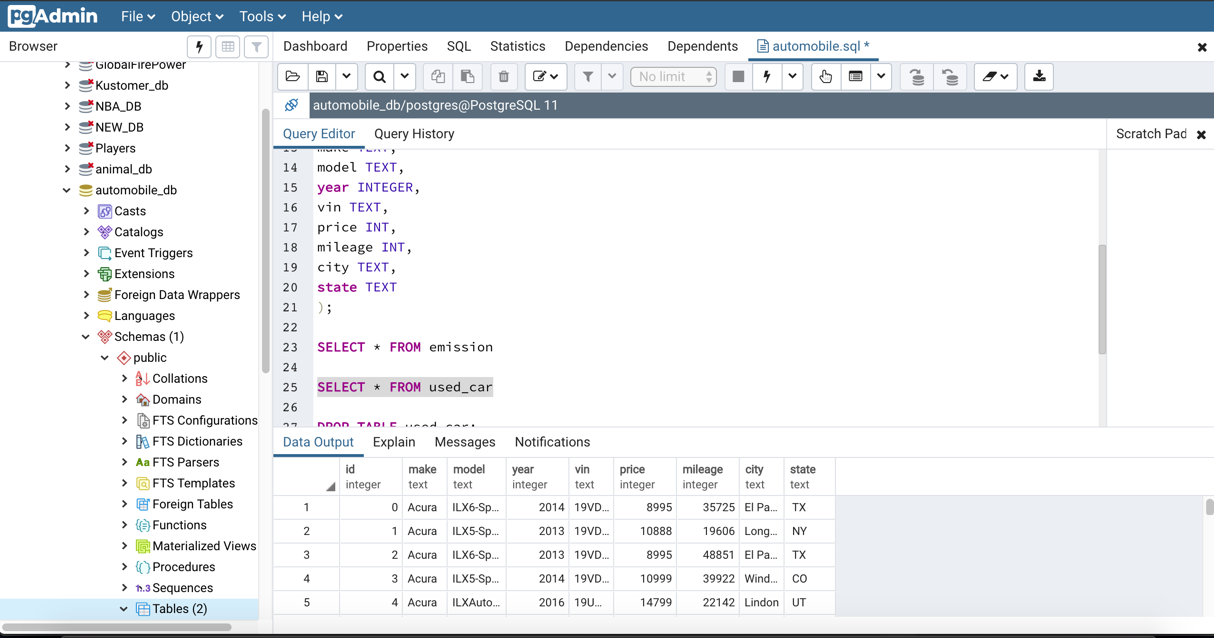
* The csv files identified in section 1.3 are imported to Pandas Jupyter Notebook used\_car\_emissions.ipynb. The expressions to do this are stated in lines 2 and 3 in Exhibit 1.
* To get better results, we recommend running VINdecoder.ipynb to get accurate ‘make’ , ‘model’, ‘year’ for each listing in the true\_car\_listings.csv, as this information is manually entered by the seller of the car, and they may be incomplete or inaccurate.
* PLEASE NOTE, the VINdecoder.ipynb is currently set to run the first 10 records in the true\_car\_listings.csv as there are over 1 million records, which would put exorbitant load on the hardware and software resources. It is best to hand over this part of the ETL to a data administrator.
* If the Vindecoder is used, then in used\_car\_emission.ipynb import VIndecoded\_true\_car\_listings.csv instead of true\_car\_listings.csv in used\_car\_emissions.ipynb
* Regardless of which of the above paths are taken, The SQL table names are:
  + emission
  + used\_car

## ETL Data Flow

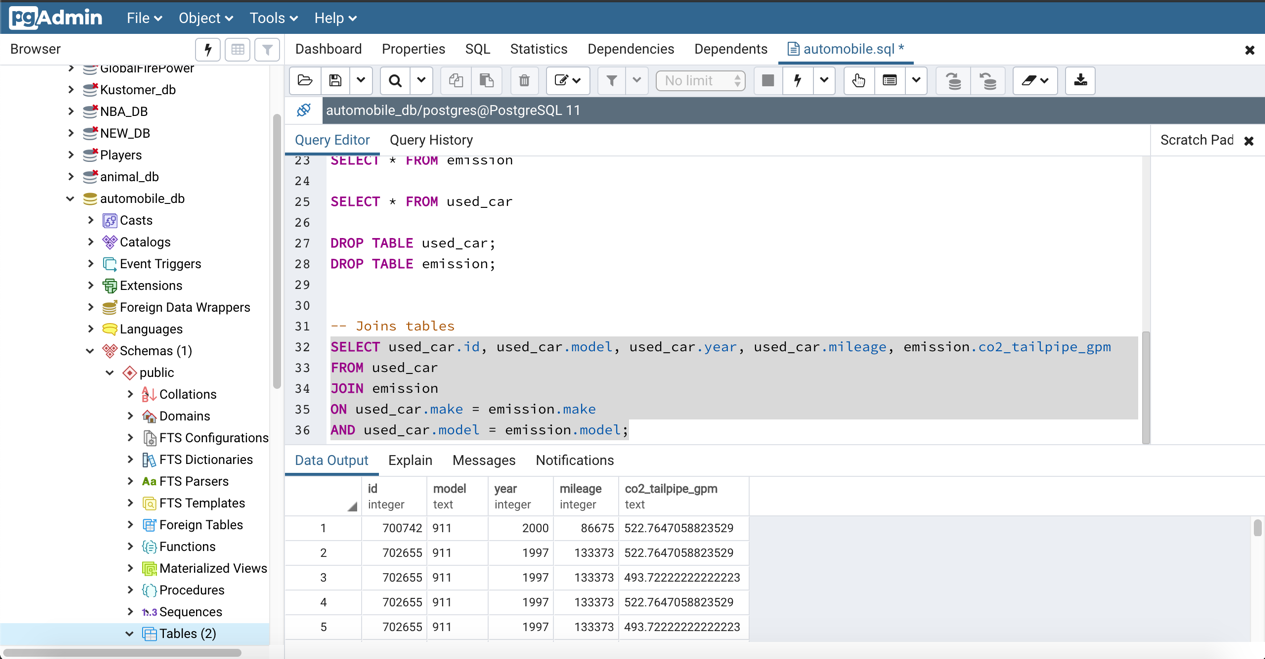
* Direct access is achieved by downloading the CSV files from their respective URL:
  + true\_car\_listings.csv from [**Fuel Economy Web Services**](https://www.fueleconomy.gov/feg/ws/index.shtml)
  + vehicles-2.csv from [**Million Used Car Listings**](https://www.kaggle.com/jpayne/852k-used-car-listings)
* The data from each source is extracted using Pandas Jupyter Notebook. The python program that is ran to perform the ETL is called:
  + used\_car\_emissions.py or used\_car\_emissions.ipynb

The expressions are documented in Exhibit 1:

* These are the transformed information (pandas data frames) that are transferred to PostgreSQL as stated in lines 13 and 14 in Exhibit 1.
  + vehicle\_transformed
  + used\_car\_transformed2
* The username and password for PostgreSQL must be entered in the python script (See line 11 in Exhibit 1)
* After step 11 in Exhibit 1 is concluded, the following must take place:
  + In PostgreSQL, a new database called automobile\_db shall be created,
  + and two tables are created where the column names match the column names of the data frames that were procured in the transformation process.
  + Next the information from each of the designated data frames in pandas are loaded to the SQL tables that were created above. This is illustrated in lines 13 and 14 of Exhibit 1.
  + Once the tables are loaded, we can make sure that the data is loaded by selecting to view all rows of the table as illustrated in the screen shot below.



* + Finally we conducted a merge between two tables on ‘make’ and ‘model’ to get the first actionable query result as illustrated below:



# Exhibit 1

Python Expressions for Extracting, Transforming, and Loading the data:

1. ​Import Dependencies

import pandas as pd

from sqlalchemy import create\_engine

​

**Extraction**

​​

1. Import the CSV file from the first data source and extract it into DataFrame

vehicle\_file = "Resources/vehicles-2.csv"

vehicle\_df = pd.read\_csv(vehicle\_file)

vehicle\_df.head()

​

​

1. Import the CSV file from the second data source and extract it into DataFrame

used\_car\_file = "Resources/true\_car\_listings.csv"

used\_car\_df = pd.read\_csv(used\_car\_file)

used\_car\_df.head()

​

​

**Transformation**

1. Begin the transformation process by creatign a subset data frame with ONLY the desired columns.

vehicle\_col = ['make', 'model', 'year','co2TailpipeGpm','fuelType', 'fuelType1']

vehicle\_transformation\_df = vehicle\_df[vehicle\_col]

vehicle\_transformation\_df.head()

​

1. ​ rename the dataframe columns to make them more SQL friendly

vehicle\_transformed = vehicle\_transformation\_df.rename(columns={

'co2TailpipeGpm': 'co2\_tailpipe\_gpm',

'fuelType': 'fuel\_type',

'fuelType1':'fuel\_type1'

})

vehicle\_transformed.head()

​

1. add a distinct id column that would map to the id column of its respective SQL table

vehicle\_transformed['id'] = vehicle\_transformed.index

vehicle\_transformed.head()

​

1. ​Similarly, repeat the transformation process for the data frame from the second source by

#creating a subset data frame with ONLY the desired columns.

used\_car\_col = ['Year', 'Make', 'Model','Price','Mileage','City','State','Vin']

used\_car\_transformation\_df = used\_car\_df[used\_car\_col]

​

1. ​rename the dataframe columns to make them more SQL friendly

used\_car\_transformed = used\_car\_transformation\_df.rename(columns={

'Year': 'year',

'Make': 'make',

'Model': 'model',

'Price':'price',

'Mileage':'mileage',

'City':'city',

'State':'state',

'Vin':'vin'

})

​

used\_car\_transformed.head()

​

1. Add a unique id column

used\_car\_transformed=used\_car\_transformed.reset\_index(drop=False)

used\_car\_transformed.head()

​

1. rename the index column to id as it would map to the id column of its respective SQL table ​

used\_car\_transformed2 = used\_car\_transformed.rename(columns={'index': 'id'})

used\_car\_transformed2.head()

​

​

**Load**

1. ​create a connection pandas and to the SQL environment - in this case PostgreSQL

rds\_connection\_string = "username:password@localhost:5432/automobile\_db"

engine = create\_engine(f'postgresql://{rds\_connection\_string}')

​

​

1. Once the tables are created on the PostgreSQL side, verify the connection is working by recalling

#the table names that were created in PostGreSQL inside the automobile\_db database.​

engine.table\_names()

​

​​

1. ​Load the information from the dataframe to its respective table in postgreSQL

vehicle\_transformed.to\_sql(name='emission', con=engine, if\_exists='append', index=False)

​

1. ​Repeat the above for the second dataframe by Loading the information from the dataframe to its respective table in postgreSQL

used\_car\_transformed2.to\_sql(name='used\_car', con=engine, if\_exists='append', index=False)

​

​

# Exhibit 2

import requests

from pprint import pprint

import pandas as pd

from sqlalchemy import create\_engine

import numpy as np

import requests

import time

​

​​

​#url for vinDecoder API

url = "https://vpic.nhtsa.dot.gov/api/vehicles/decodevin/5UXWX7C5\*BA?format=json"

​

​​

​# test the url for vinDecoder API with different VIN

car = "https://vpic.nhtsa.dot.gov/api/vehicles/decodevin/WBAUP7C56CVP22994?format=json"

​

​​

​# json query

autos = requests.get(car).json()

autos

​

​

​# queries to get make, model, and year from the json

make = autos['Results'][6]['Value']

model = autos['Results'][8]['Value']

year = autos['Results'][9]['Value']

print(make)

print(model)

print(year)

​

​

​#import csv with the VIN listings

vehicle\_file = "Resources/true\_car\_listings.csv"

vehicle\_df = pd.read\_csv(vehicle\_file)

vehicle\_df.head()

​

​

​

test = requests.get(query).json()

test

​

​

​

​#truncated file with 10 records to test if the API calls work

short\_vehicle\_df = vehicle\_df[:10]

short\_vehicle\_df

​

​

# create three lists to store retrieved data  
make = []

model = []

year = []

# loop through VIN’s to and call the code for API queries to get the results and append it to 3 lists.

for i in short\_vehicle\_df['Vin']:

query = f"{url2}{i}?format=json"

response = requests.get(query).json()

make.append(response['Results'][6]['Value'])

model.append(response['Results'][8]['Value'])

year.append(response['Results'][9]['Value'])

​

​

​# select required columns from the csv file

new\_vehicle\_df = short\_vehicle\_df[['Price','City','State','Mileage','Vin']]

new\_vehicle\_df.head()

​

​

​

​# append retrieved data

new\_vehicle\_df['make'] = make

new\_vehicle\_df['model'] = model

new\_vehicle\_df['year'] = year

​

​

​# check new dataframe

new\_vehicle\_df.head()

​

​

​# create new csv file with appended data

new\_vehicle\_df.to\_csv("VINdecoded\_true\_car\_listings.csv", index = False)

​

​

​

​

​

​