Data Cleaning for Informed Craigslist Car Purchases

Carolyn Mason

12/11/18

CSCI E-63 Big Data Analytics

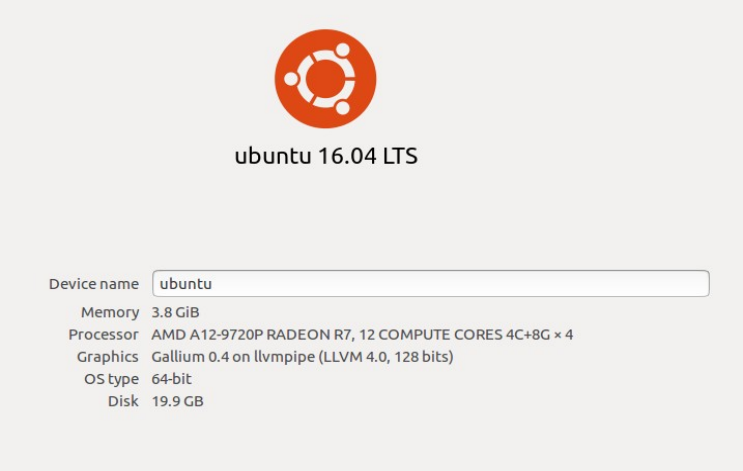
Problem:

People spend a large amount of time searching Craigslist for deals that best suite them. There are many options that can be difficult to sort through while making timely informed decisions. Sometimes the item will sell and the opportunity will be missed or users pass up a 'good-deal' without realizing it. There are tools that can aggregate Craigslist data which can help users sort through the listings and have a better understanding of the best listings out there. This project will take this set of data and use big data tools to make useful insights for user feedback.

Software:

This project demonstrates all examples using Python and Pyspark. The packages include sql, a scrapy webcrawler, and matplotlib.

The code was run on Linux 16.04 virtual machine.



Description of Data:

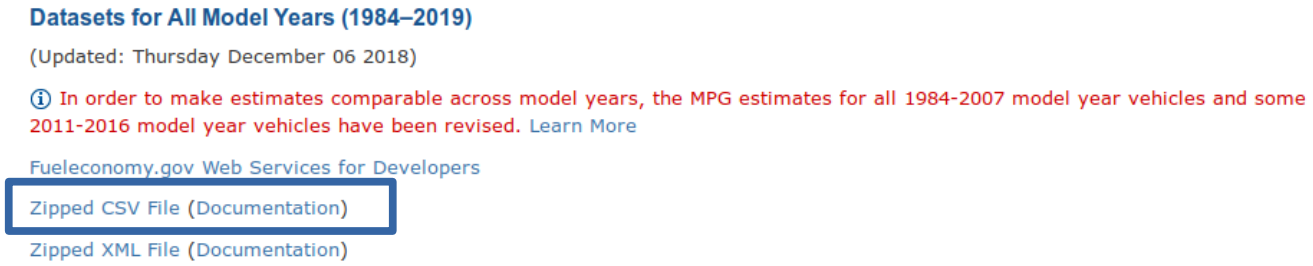
This project takes on two large sets of data. These include fuel economy listings from the US Department of Engery and aggregated Craigslist data. The fuel economy data is joined with the Craigslist data to provide more information for the user. The websites I used are listed below:

Fuel economy data:
https://www.fueleconomy.gov/feg/download.shtml

Craigslist:
https://newyork.craigslist.org/search/brk/cto

1) Fuel Economy Data:

On the US Department of energy website there are several data sets for vehicles ranging from 1984 to 2019. I scrolled down to the section ‘Datasets for All Model Years (1984-2019)’ and downloaded the ‘Zipped CSV File (Documentation)’. The raw data set contains a list of 40692 vehicles with 83 columns. Before this data is used, it will be cleaned and condensed.



A full list of the column headers shown below. The key columns are highlighted in yellow.

barrels08	engId,	pv2,	charge240b,
barrelsA08,	eng_dscr,	pv4,	c240bDscr,
charge120,	feScore,	range,	createdOn,
charge240,	fuelCost08,	rangeCity,	modifiedOn,
city08,	fuelCostA08,	rangeCityA,	startStop,
city08U,	fuelType,	rangeHwy,	phevCity,
cityA08,	fuelType1,	rangeHwyA,	phevHwy,
cityA08U,	ghgScore,	trany,	phevComb
cityCD,	ghgScoreA,	UCity,	
cityE,	highway08,	UCityA,	
cityUF,	highway08U,	UHighway,	
co2,	highwayA08,	UHighwayA,	
co2A,	highwayA08U,	VClass,	
co2TailpipeAGpm,	highwayCD,	year,	
co2TailpipeGpm,	highwayE,	youSaveSpend,	
comb08,	highwayUF,	guzzler,	
comb08U,	hlv,	trans_dscr,	
combA08,	hpv,	tCharger,	

combA08U,	id,	sCharger,
combE,	lv2,	atvType,
combinedCD,	lv4,	fuelType2,
combinedUF,	make,	rangeA,
cylinders,	model,	evMotor,
displ,	mpgData,	mfrCode,
drive,	phevBlended,	c240Dscr,

The key columns are: cylinders, displ, drive, fueltype, make, model, ucity, uhighway, trany, vclass, year. A short description and/or expected values are shown below. These were chosen, since they are helpful for a user choosing a used vehicle and are parameters that can help to match with Craiglist data.

cylinders: Number of cylinders in a car often between 4-12 are the central working part of a car engine

displ: Displacement is the total volume of the car's cylinders. This is how engine size is measured

drive: Awd, Fwd, Rwd, 2wd , 4wd

fueltype: gas, diesel, other

year: The data spans the years 1984-2019

make: This is the car manufacturer. A full list of all makes is here: bertone, london coach co inc, isis imports ltd, import trade services, lexus, jba motorcars, inc., merkur, gmc, yugo, lotus, saleen, asc incorporated, coda automotive, byd, smart, dodge, am general, aurora cars ltd, honda, aston martin, panther car company limited, srt, spyker, mercedes-benz, jaguar, saturn, geo, daewoo, lincoln, bill dovell motor car company, pas, inc, nissan, pas inc - gmc, roush performance, saab, fisker, vector, excalibur autos, volga associated automobile, bentley, texas coach company, toyota, import foreign auto sales inc, ruf automobile gmbh, volvo, general motors, pontiac, pagani, quantum technologies, panos, qvale, lambda control systems, mobility ventures llc, j.k. motors, s and s coach company e.p. dutton, hummer, eagle, morgan, isuzu, hyundai, volkswagen, mazda, wallace environmental, american motors corporation, consulier industries inc, rolls-royce, bugatti, fiat, acura, renault, jeep, daihatsu, mahindra, land rover, chrysler, infiniti, panoz auto-development, avanti motor corporation, subaru, buick, mercury, porsche, kenyon corporation of america, federal coach, cx automotive, environmental rsch and devp corp, red shift ltd., genesis, ccc engineering, mclaren automotive, evans automobiles, bitter gmbh and co. kg, ford, vpg, bmw, mitsubishi, vixen motor company, tvr engineering ltd, lamborghini, koenigsegg, pininfarina, suzuki, mcevoy motors, london taxi, plymouth, maybach, oldsmobile, goldacre, mini, tesla, ram, superior coaches div e.p. dutton, audi, maserati, shelby, chevrolet, e. p. dutton, inc., dabryan coach builders inc, sterling, grumman olson, dacia, scion, cadillac, saleen performance, kia, peugeot, alfa romeo, azure dynamics, laforza automobile inc, tecstar, lp, ferrari, karma, grumman allied industries, bmw alpina, autokraft limited

model: For each year car manufacturers will create a set of models. The model name will match a specific vehicle.

ucity: Miles per gallon for city driving

uhighway: Miles per gallon for highway driving

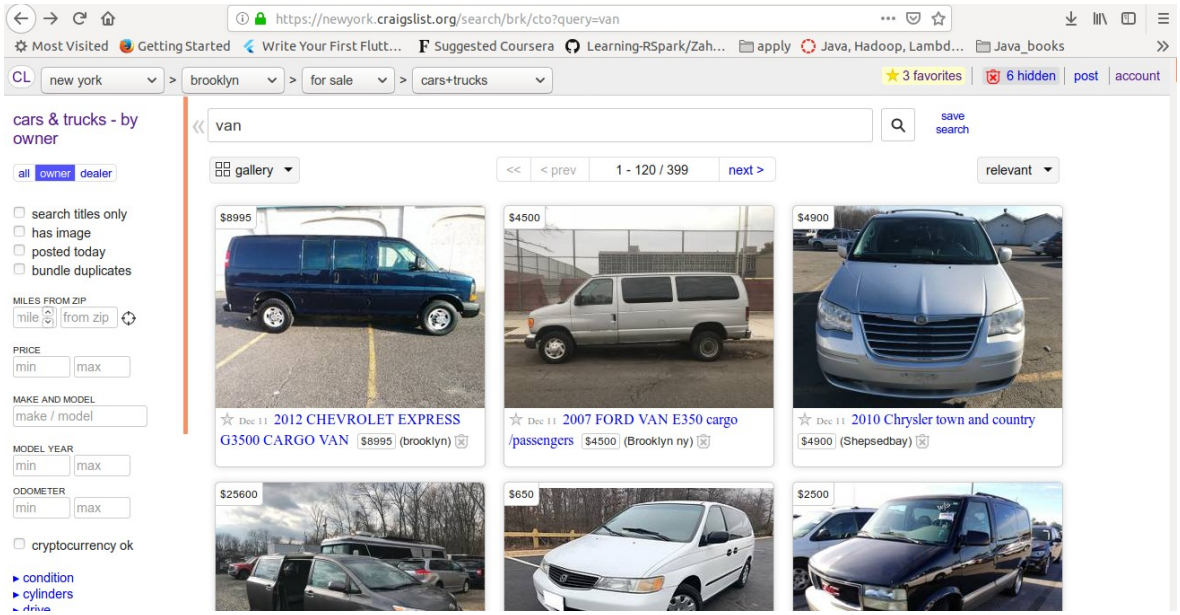
trany: Transmission: manual, automatic, or other

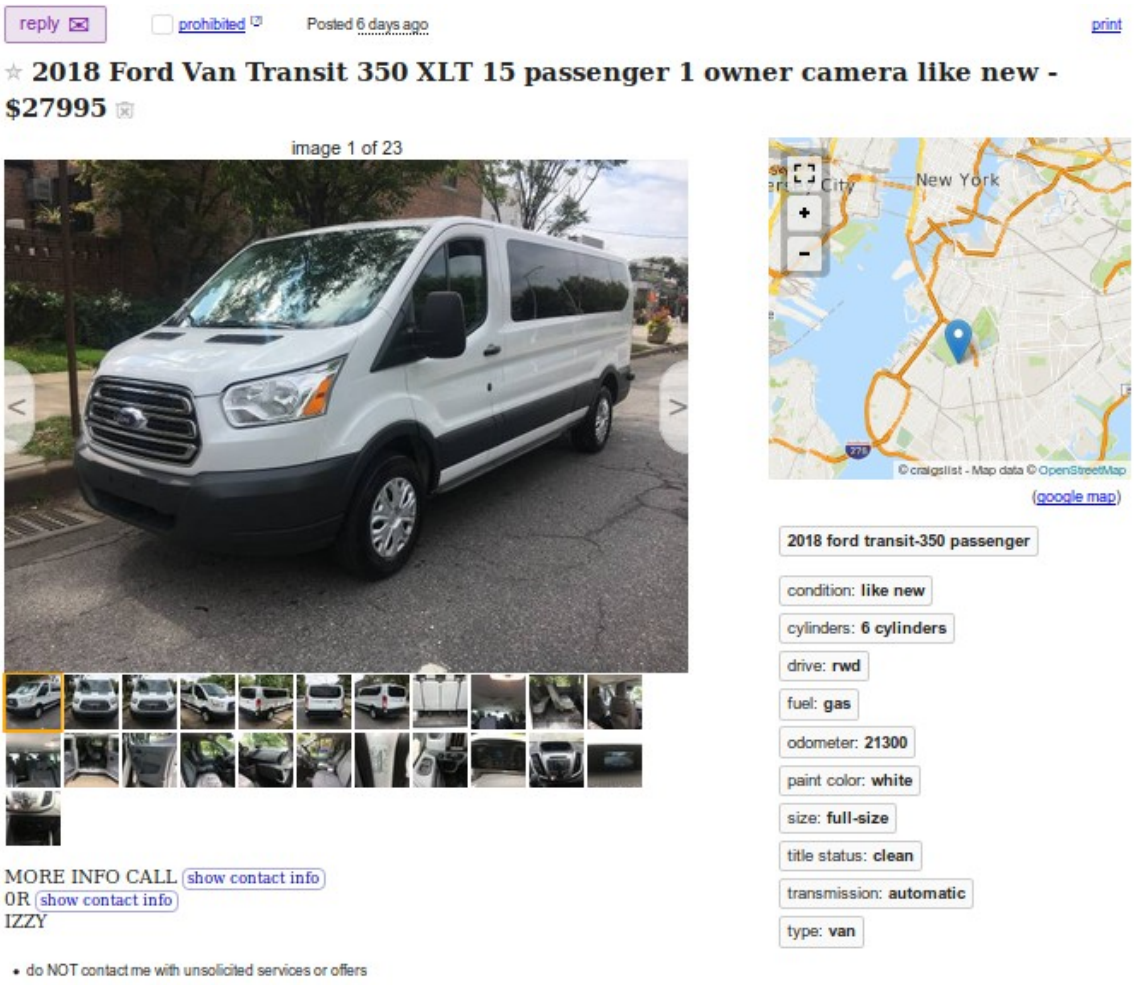
vclass: Vehicle class. A full list of classes are here: two seaters, compact cars, subcompact cars, small station wagons, midsize station wagons, midsize cars, minivan – 2wd, minivan – 4wd, small sport utility vehicle 2wd, small sport utility vehicle 4wd, sport utility vehicle – 4wd, sport utility vehicle – 2wd, standard pickup trucks 4wd, large cars, standard pickup trucks, standard pickup trucks 2wd, vans, vans cargo type, vans passenger type, special purpose vehicles, special purpose vehicle 2wd

In the next section, I will show the steps for pairing down and cleaning the fuel economy data.

2) Craigslist Data:

The next data set is taken from Craigslist. Craigslist is an American Classifieds site that has many categories of items sold. This project focuses on cars. A screen shot of Craigslist and an example of an add is shown below:





Information from each add is aggregated using a simple python scraper. I used the following tutorial to setup my initial scrapy code. The online example shows how to grab job listings and save the data to a csv file. The tutorial uses a spider, which will crawl the given website to extract data.

Initial scrapy setup:
https://python.gotrained.com/scrapy-tutorial-web-scraping-craigslist/

In order to install scrapy I followed the tutorial and ran these commands.

$ sudo pip install scrapy
$ scrapy startproject craigslist

I navigated into the spiders folder to create my own spider.

I expanded upon this code to search Craigslist for cars. I will step through my code in the next section.

The column names of the Craislist data are:
title, url, price, address, vin, odometer, condition, cylinders, drive, fuel, paint_color, size, title_status, transmission, type, year, make, model, description

Code:

There are three major parts to this tool: 1) Preparing the fuel economy data set, 2) Gathering and cleaning Craigslist data, 3) Joining the data sets for informed learning.

1) Preparing the Fuel Economy Data Set

I started with the csv I downloaded from (https://www.fueleconomy.gov/feg/download.shtml) and brought it into pyspark in order to clean up and pair down the code.

First import necessary functions and read in csv:

```
from pyspark.sql.functions import regexp_replace, col
import Pandas

# Read in csv
df_fuel = spark.read.format("csv").option("header", "true").option("inferSchema",
"true").load("fuel.csv")
```

Some of the data comes in a format that is non-standard and difficult to match. In order to match how data is stored in the Craigslist data file, I updated the transmission information.

```
# Re-format transmission data
df_fuel = df_fuel.withColumn('trany', regexp_replace('trany', 'Automatic.*',
'automatic'))
df_fuel = df_fuel.withColumn('trany', regexp_replace('trany', 'Manual.*',
'manual'))
```

Within the data set I found there were several occurrences of the same make, model, year, transmission, and number of cylinders. These data entries only differed by displacement and/or fuel economy. Since there was noway of gathering this information cleanly from Craigslist and the fuel economy was similar between ‘duplicate’ entries, I decided to remove these ‘duplicates’. I ordered the ‘duplicates’ by displacement, so the vehicles estimated fuel economy was not an over-estimate.

```
# Remove duplicates:
df_fuel.createOrReplaceTempView("fuel")
ans = spark.sql("SELECT * FROM (select cylinders,displ,drive,fuelType AS
fueltype,make,model,UCity AS ucity,UHighway AS uhighway,trany AS
transmission,VClass as vclass,year, row_number() OVER (PARTITION BY
make,model,year,trany,cylinders ORDER BY displ DESC) AS rn FROM fuel) AS dt WHERE
rn =1 ORDER BY make,model,year")
```

The next step is to save the edited data frame as a csv for later use.

```
# Create csv to read
ans.toPandas().to_csv('fuel_simple.csv', encoding='utf-8', index=False)
```

After the data is saved as a csv, it is VERY IMPORTANT that the user goes in and makes all items in the csv lowercase. I am currently doing this by hand. In future iterations, it might be cleaner if this step is done in the code.

A sample of the ‘simple-fuel.csv’ is shown below:

```
cylinders,displ,drive,fueltype,make,model,ucity,uhighway,transmission,vclass,year,rn
4,2.5,2-wheel drive,regular,am general,dj po vehicle 2wd,22,24,automatic,special purpose vehicle 2wd,1984,1
6,4.2,2-wheel drive,regular,am general,fj8c post office,16,18,automatic,special purpose vehicle 2wd,1984,1
4,2.5,rear-wheel drive,regular,am general,post office dj5 2wd,20,23.0769,automatic,special purpose vehicle 2wd,1985,1
6,4.2,rear-wheel drive,regular,am general,post office dj8 2wd,16,18,automatic,special purpose vehicle 2wd,1985,1
6,3.8,rear-wheel drive,premium,asc incorporated,gnx,17,29,automatic,midsize cars,1987,1
6,3,front-wheel drive,regular,acura,2.2cl/3.0cl,22,35.8974,automatic,subcompact cars,1997,1
4,2.2,front-wheel drive,regular,acura,2.2cl/3.0cl,28,39.7436,manual,subcompact cars,1997,1
4,2.2,front-wheel drive,regular,acura,2.2cl/3.0cl,25,37,automatic,subcompact cars,1997,1
4,2.3,front-wheel drive,regular,acura,2.3cl/3.0cl,24.5,37.9,automatic,subcompact cars,1998,1
6,3,front-wheel drive,regular,acura,2.3cl/3.0cl,21.7329,35.8658,automatic,subcompact cars,1998,1
4,2.3,front-wheel drive,regular,acura,2.3cl/3.0cl,27.1,40.2,manual,subcompact cars,1998,1
6,3,front-wheel drive,regular,acura,2.3cl/3.0cl,21.7454,35.8907,automatic,subcompact cars,1999,1
4,2.3,front-wheel drive,regular,acura,2.3cl/3.0cl,24.7,38,automatic,subcompact cars,1999,1
4,2.3,front-wheel drive,regular,acura,2.3cl/3.0cl,27,40.3,manual,subcompact cars,1999,1
5,2.5,front-wheel drive,premium,acura,2.5tl,22,32,automatic,compact cars,1995,1
6,3.2,front-wheel drive,premium,acura,2.5tl/3.2tl,21,30.7692,automatic,compact cars,1996,1
```

2) Gathering and Cleaning Craigslist Data

In this section I will review the code that I used to aggregate the Craigslist data. This piece of the code, ended up being fairly complex. In order to make a good match with the fuel economy data set, make, model, and year are required. Unfortunately make and model are not tags and much be sourced from the title, description, etc. The make was easy to search for, but a good amount of work went into finding the correct model. More improvements can be made in future code. A discussion of accuracy and lessons learned will be provided in a later section.

This section is one spider script called ‘vans.py’.

Start with the imports.

```
# Scrape
import scrapy
from scrapy import Request
import re, csv
```

This class holds the top-level information. This includes the *name* of the spider (it is important this matches the name of the python file), the allowed domains that the spider is allowed to scrape, and the start urls. The start url is the same as the Craigslist search url. I am running a search to look for vans that are priced greater than $1000. The min price helps cut-out listings that are not vehicles. The full list of Craigslist search options is listed below, so the query can easily be updated. Searches can also be done purely in pyspark.

```
class JobsSpider(scrapy.Spider):

    # Setup the search
    name = "vans"
    allowed_domains = ["craigslist.org"]

    # List of search options
    query = 'vans'
    search_distance=100
    postal=06033
    min_price=50
```

```
   max_price=10000
   auto_make_model='subaru'
   min_auto_year=1995
   max_auto_year=3000
   min_auto_miles=0
   max_auto_miles=15000
   condition=10  # 10=new,20=likenew,30=excellent,40=good,50=fair,60=salvage
   auto_cylinders=4
   auto_drivetrain=1 # 1=fwd, 2=rwd, 3=4wd
   auto_fuel_type=1 # 1=gas, 2=diesel, 3=hybrid, 4=electric, 5=other
   auto_size=3 # 1=compact, 2=full-size, 3=mid-size, 4=sub-compact
   auto_title_status=1 # 1=clean, 2=salvage, 3=rebuilt, 4=parts only, 5=lien,
6=missing
   auto_transmission=2 # 1=manual, 2=automatic, 3=other
   auto_bodytype=1 # 1=bus, 2=convertible, 3=coupe, 4=hatchback, 5=mini-van,
6=offroad,
                   # 7=pickup, 8=sedan, 9=truck, 10=SUV, 11=wagon, 12=van, 13=other

   # Craigslist start URL
   start_urls = ["https://newyork.craigslist.org/search/cto?
query=van&min_price=1000"]
```

Parse, is the predominant function of the spider. Here the urls will be iterated over. The line below that checks for 'button-next' will allow the code to search over multiple Craigslist pages.

```
   def parse(self, response):
      cars = response.xpath('//p[@class="result-info"]')

      # Print number of cars per page
      total = len(cars) + 1
      print(str(total)+' ..............')

      # Iterate over cars
      for car in cars:
         relative_url = car.xpath('a/@href').extract_first()
         absolute_url = response.urljoin(relative_url)
         title = car.xpath('a/text()').extract_first()
         address = car.xpath('span[@class="result-meta"]/span[@class="result-
hood"]/text()').extract_first("")[2:-1]

         yield Request(absolute_url, callback=self.parse_page, meta={'URL':
absolute_url, 'Title': title, 'Address':address})
      try:
         # See if there is a next page
         relative_next_url = response.xpath('//a[@class="button
next"]/@href').extract_first()
         absolute_next_url = "https://newyork.craigslist.org" + relative_next_url

         yield Request(absolute_next_url, callback=self.parse)
      except:
         pass
```

parse_page is the next major block of code. Here each listing is parsed, cleaned, and sent to be written to a csv. I included a good amount of logic for locating the pieces of information needed for a match. These include: make, model, year, transmission, and cylinders. Model proved to be difficult since there

is are many different models year to year with semi-unique names. The make and year possibilities are much smaller. The number of possible makes is 136 and the number of possible years is 34. I started this code by creating a dictionary of all models that pertain to a given make and given year. Once the make and year were determined, this made searching for the possible model much easier.

```
   def parse_page(self, response):

      # Needed lists
      all_makes= ['bertone', 'london coach co inc', 'isis imports ltd', 'import
trade services', 'lexus', 'jba motorcars, inc.', 'merkur', 'gmc', 'yugo', 'lotus',
'saleen', 'asc incorporated', 'coda automotive', 'byd', 'smart', 'dodge', 'am
general', 'aurora cars ltd', 'honda', 'aston martin', 'panther car company
limited', 'srt', 'spyker', 'mercedes-benz', 'jaguar', 'saturn', 'geo', 'daewoo',
'lincoln', 'bill dovell motor car company', 'pas, inc', 'nissan', 'pas inc - gmc',
'roush performance', 'saab', 'fisker', 'vector', 'excalibur autos', 'volga
associated automobile', 'bentley', 'texas coach company', 'toyota', 'import foreign
auto sales inc', 'ruf automobile gmbh', 'volvo', 'general motors', 'pontiac',
'pagani', 'quantum technologies', 'panos', 'qvale', 'lambda control systems',
'mobility ventures llc', 'j.k. motors', 's and s coach company  e.p. dutton',
'hummer', 'eagle', 'morgan', 'isuzu', 'hyundai', 'volkswagen', 'mazda', 'wallace
environmental', 'american motors corporation', 'consulier industries inc', 'rolls-
royce', 'bugatti', 'fiat', 'acura', 'renault', 'jeep', 'daihatsu', 'mahindra',
'land rover', 'chrysler', 'infiniti', 'panoz auto-development', 'avanti motor
corporation', 'subaru', 'buick', 'mercury', 'porsche', 'kenyon corporation of
america', 'federal coach', 'cx automotive', 'environmental rsch and devp corp',
'red shift ltd.', 'genesis', 'ccc engineering', 'mclaren automotive', 'evans
automobiles', 'bitter gmbh and co. kg', 'ford', 'vpg', 'bmw', 'mitsubishi', 'vixen
motor company', 'tvr engineering ltd', 'lamborghini', 'koenigsegg', 'pininfarina',
'suzuki', 'mcevoy motors', 'london taxi', 'plymouth', 'maybach', 'oldsmobile',
'goldacre', 'mini', 'tesla', 'ram', 'superior coaches div e.p. dutton', 'audi',
'maserati', 'shelby', 'chevrolet', 'e. p. dutton, inc.', 'dabryan coach builders
inc', 'sterling', 'grumman olson', 'dacia', 'scion', 'cadillac', 'saleen
performance', 'kia', 'peugeot', 'alfa romeo', 'azure dynamics', 'laforza automobile
inc', 'tecstar, lp', 'ferrari', 'karma', 'grumman allied industries', 'bmw alpina',
'autokraft limited', 'chevy'] # added chevy

      all_years = ['1985', '1986', '1987', '1989', '1990', '1991', '1992', '1993',
'1994', '1995', '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003',
'2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013',
'2014', '2015', '2016', '2017', '2018', '2019']

      # Make dictionary
      makes = {}
      with open('/home/carolyn/Documents/Classes/CSCI E-63 Big Data
Analytics/final_project/practice_joins/fuel_simple.csv') as csvfile:
         rows = csv.reader(csvfile, delimiter=',')
         # Dont read the header
         next(rows)
         for row in rows:
            #  0:cylinders, 1:displ, 2:drive, 3:fueltype, 4:make, 5:model, 6:ucity,
7:uhighway, 8:trany, 9:vclass, 10:year
            make_ = row[4].lower()
            year_ = int(row[10])
            model_ = row[5].lower()
            if make_ in makes:
```

```
            if year_ in makes[make_]:
               if model_ in makes[make_][year_]:
                  pass
               else:
                  makes[make_][year_].append(model_)
            else:
               makes[make_][year_] = [model_]
         else:
            makes[make_] = {year_ : [model_]}
```

This next section of text will search the listing for set tags that are called specifically by name. These include title, price, and description.

```
      # Get pieces
      url = response.meta.get('URL')
      title = response.meta.get('Title')
      address = response.meta.get('Address')

      # Get set tags: title, price, description. Clean up text
      title = response.xpath('//*[@ id =
"titletextonly"]/text()').extract_first().replace(',', '')
      price_init = response.xpath('//*[@class = "price"]/text()').extract_first()
      if(price_init):
         price = price_init.replace('$', '').replace(',', '')
      else:
         price = ''
      description = "".join(line for line in
response.xpath('//*[@id="postingbody"]/text()').extract()).replace(',',
'').replace('\n', '')
```

All other tags are listed under different attrgroups. In order to get this information from the listing, an array can be created for the tags and values by searching over the ‘attrgroup’. Some tags include empty spaces, so both the tags and values must be cleaned. Extra information can sometimes be included, so one must be careful to make sure the correct tag is with the correct value. The code below works through these steps and outputs a dictionary of attributes.

```
      # Get all vehicle information
      given_tags = response.xpath('//p[@class="attrgroup"]/span/text()').extract()
      given_values =
response.xpath('//p[@class="attrgroup"]/span/b/text()').extract()

      # Many times the car title is given as a tag_value, which throughs off the
dictionary. Check.
      given_tags_clean = filter(None,[x.replace('\n', '').replace(' ',
'').replace(',', '') for x in given_tags])
      given_values_clean = filter(None,[x.replace('\n', '').replace(',', '') for x
in given_values])
      # Make sure the tag placement is correct
      if ( len(given_values_clean) > len(given_tags_clean) or given_tags_clean[-1]
== "otherpostings"):
         dictionary = dict(zip(given_tags_clean, given_values_clean[1::]))
         title2 = given_values_clean[0]
      else:
         dictionary = dict(zip(given_tags_clean, given_values_clean))
```

```
        title2 = ''
```

I then search the dictionary to see which pieces of information the user provided. If a key piece of information is not there, additional work must be done.

```
      # VIN
      # odometer
      # name
      # condition
      # cylinders
      # drive
      # fuel
      # paint color
      # size
      # title status
      # transmission
      # type
      # model year
      vin = dictionary[u'VIN:'] if u'VIN:' in given_tags_clean else ''
      odometer = dictionary[u'odometer:'] if u'odometer:' in given_tags_clean else
''
      condition = dictionary[u'condition:'] if u'condition:' in given_tags_clean
else ''
      cylinders = dictionary[u'cylinders:'] if u'cylinders:' in given_tags_clean
else ''
      cylinders = re.sub("[^0-9]+","",cylinders)
      drive = dictionary[u'drive:'] if u'drive:' in given_tags_clean else ''
      fuel = dictionary[u'fuel:'] if u'fuel:' in given_tags_clean else ''
      paint_color = dictionary[u'paintcolor:'] if u'paintcolor:' in
given_tags_clean else ''
      size = dictionary[u'size:'] if u'size:' in given_tags_clean else ''
      title_status = dictionary[u'titlestatus:'] if u'titlestatus:' in
given_tags_clean else ''
      transmission = dictionary[u'transmission:'] if u'transmission:' in
given_tags_clean else ''
      vtype = dictionary[u'type:'] if u'type:' in given_tags_clean else ''
      year = dictionary[u'modelyear:'] if u'modelyear:' in given_tags_clean else ''
```

In order to fill in needed/ missing blanks, I use regular expressions search cleaned strings including titles, drive, and description. I use the make, year, and model arrays created above to run the searches.

```
      # Get make, model, and year are filled out. Important for join with fuel.csv
      # Check title, title2, drive and description
      search_all = title+title2+drive+description
      search_title = list(set([re.sub("[^0-9a-zA-Z- ]+","",x.lower()) for x in
title.split(' ')]))
      search_all_clean = re.sub("[^0-9a-zA-Z ]+","",search_all.lower())
      search_list = list(set([re.sub("[^0-9a-zA-Z ]+","",x.lower()) for x in
search_all.split(' ')]))
```

Since my search relies on how well Craigslist users entered data, I added in two levels of searching. First I check the title, then I check the title+drive+description. Sometimes users will add extraneous information into the description and list trades that they would like to make etc. We do not want the

code to match on this. The title tends to be more accurate, so I start there and continue on to the description if needed. This worked out really well for the make and year.

```
      # Make
      make_title = all_makes + search_title
      make_all = all_makes + search_list
      match_title = [x for x in make_title if make_title.count(x) > 1]
      match_all = [x for x in make_all if make_all.count(x) > 1]
      # Check the title first, becasuse ofter there is excess information in the
description
      if( len(match_title) > 0 ):
         make = match_title[0]
         if( make == 'chevy'):
            make = 'chevrolet'
      # Check all text
      elif( len(match_all) > 0 ):
         make = match_all[0]
         if( make == 'chevy'):
            make = 'chevrolet'
      else:
         make = None

      # Year
      if( year == ''):
         year_title = all_years + search_title
         year_all = all_years + search_list
         match_title = [x for x in year_title if year_title.count(x) > 1]
         match_all = [x for x in year_all if year_all.count(x) > 1]
         # Check the title first, because often there is excess information in the
description
         if( len(match_title) > 0 ):
            year = int(match_title[0])
         # Check all text
         elif( len(match_all) > 0 ):
            year = int(match_all[0])
         else:
            year = None
      else:
         # Year should be an int
         year = int(year.replace(' ', ''))
```

Finding the correct model, proved to be a lot more difficult. I felt this was important to spend time on to get accurate results- so I worked with three different methods. First I looped over all possible models for the given make/year (between 1 and 30 options) and looked for a direct match in the search string. If this failed I applied a second method which split up the model string and appended the drive type. Very often the model listed included a drive type and the first method missed the match. When the drive was added to the search, this provided many good matches. The last method implemented a smart-catch that counted the number of times words in the given model were found in the search text. If one model had more matches than all of the rest, then that match was the best! Often several matches tied or had low results. In this case, these matches were thrown out. It is important that only good matches are kept. This being said, this code is not an exact science. Many mis-matches can slide through. This is in-part due to bad input, but also in-part to how complicated some of these matches

are. More work is required here- but I did not want to de-focus the intent of the project. From test I ran- most matches are good.

```
      # Model
      # Instead of splitting all words... take out all spaces in both and search
for items in text string.
      model = None
      if( make != None and year != None ):

         # Use try/except incase a make/year has no models
         try:
            models = makes[make][year]
         except:
            models = []

         model_dict = dict.fromkeys(models, 0)
         myBreak = False

         # Loop over possible models for make/ year to find a match
         for m in models:
            # Method 1
            if ' '+m+' ' in search_all_clean:
               model = m
               #print("(1)")
               myBreak = True
               break
            else:
               # Method 2
               if( drive != ''):
                  #print("(2)")
                  # Check again. Big issue with models including drive in text:
                  sect = re.split('[^a-zA-Z0-9-/]', m)[0].lower()
                  result = re.search(sect,search_all_clean)
                  if result:
                     newSearch = result.group(0) + '.*' + drive
                     #print(newSearch)
                     # Check drive result
                     if re.search(newSearch, m):
                        model = m
                        myBreak = True
                        break
                     # If drive fwd or rwd try 2wd for more matches
                     if( drive == 'fwd' or drive == 'rwd'):
                        newSearch2 = result.group(0) + '.*' + '2wd'
                        if re.search(newSearch2, m):
                           model = m
                           myBreak = True
                           break
               # Method 3- smart catch
               model_sect = re.split('[^a-zA-Z0-9.!]', m)
               for sect in model_sect:
                  model_dict[m] += search_all_clean.count(' '+sect.lower()+' ')
```

```
         if(sum(model_dict.values()) > 0 and myBreak == False):
            array = model_dict.values()
            # Check if valid. If count <= 2 (since often forget to supply drive
type)
            if (array.count(max(array)) <= 2 and max(array) >= 1):
               model = max(model_dict.iterkeys(), key=lambda k: model_dict[k])
```

The last piece of information was to search for the number of cylinders. If not provided, I used a simple regex to find the value in the search string.

```
      # Cylinders
      if( cylinders == ''):
         result = re.search('.*(\d{1,2}).{0,3}cylinder',search_all_clean)
         result2 = re.search('.*cylinder.{0,3}(\d{1,2})\s',search_all_clean)
         result3= re.search('.*v(\d{1,2})\s',search_all_clean)
         if(result):
            cylinders = result.group(1)
         elif(result2):
            cylinders = result2.group(1)
         elif(result3):
            cylinders = result3.group(1)
```

Lastly, the data is sent off to the resulting csv.

```
      # Only keep good data for simplicity:
      # Check if successful with make, model, year, transmission, and cylinders
      if(make != None and model != None and year != None and transmission != '' and
cylinders != ''):
         yield{'url': url, 'title': title, 'price': price, 'address': address,
'vin': vin, 'odometer': odometer, 'condition': condition, 'cylinders': cylinders,
'drive': drive, 'fuel': fuel, 'paint_color': paint_color, 'size': size,
'title_status': title_status, 'transmission': transmission, 'type': vtype, 'year':
year, 'make': make, 'model': model, 'description': description}
      else:
         pass
```

Here is a sample of the vans.csv dataset:

```
1 title,url,price,address,vin,odometer,condition,cylinders,drive,fuel,paint_color,size,title_status,transmission,type,year,make,model,descriptio
2 Toyota Sienna 2005,https://newyork.craigslist.org/brk/cto/d/toyota-sienna-2005/6769714838.html,4650,Brooklyn Ny,,107657,like new,
  6,4wd,gas,grey,full-size,clean,automatic,mini-van,2005,toyota,sienna 4wd,        Hey I'm selling 2005 Toyota Sienna in perfect condition
  never been in accident runs great no engine or transmission light call me if you interested price negotiable
3 1998 Sienna Le,https://newyork.craigslist.org/brk/cto/d/1998-sienna-le/6769716627.html,1350,brooklyn,4t3zf13c9wu044407,180651,fair,
  6,fwd,gas,grey,,clean,automatic,mini-van,1998,toyota,sienna 2wd,        1998 Sienna Le.runs 100% good.dmv inspection system is ready.4 new
  tires.pls see pictures for dmges on right side doors.left front door handle.left rear bumper corner.mileage 180651.asking$1350. .
4 Honda Odyssey,https://newyork.craigslist.org/brk/cto/d/honda-odyssey/6769695062.html,5850,Ave Z & Ocean Ave,,86151,excellent,
  6,fwd,gas,blue,full-size,clean,automatic,mini-van,2007,honda,odyssey,        This is a very nice 2007 HONDA ODYSSEY! Has one previous owner
  all the options are there  including electric doors and sunroof. Good tires with 3 sets of keys. Kept in the garage All the time  runs
  smooth  no issues! Has the NY inspection till June 2019!Any questions please feel free to contact me at  ! Alex
5 2013 TOYOTA SIENNA LE MINIVAN 1 OWNER!ONLY 60K MILES!LIKE NEW!,https://newyork.craigslist.org/brk/cto/d/2013-toyota-sienna-le-minivan/
  6762941263.html,16000,new york,5TDKK3DC2DS387243,60596,,6,fwd,gas,black,mid-size,clean,automatic,van,2013,toyota,sienna 2wd,        2013
  TOYOTA SIENNA LE MINIVAN. 3.5L V6. 1 OWNER. ONLY 60K MILES. PRIVACY GLASS. ROOF RACK. GOOD TIRES. SEATS 8 PEOPLE. CLEAN CLOTH SEATS. 3 ROWS
  OF SEATS. BLUETOOTH. BACK UP CAMERA. CRUISE CONTROL. REMOTE START. POWER  DOORS MIRRORS WINDOWS SEAT.  AM/FM/CD.  AC & HEAT.  POWER SLIDING
  DOORS. 2 SETS OF KEYS. ALARM WITH REMOTE CONTROL. CALL ME AT 9l7-797-6699...9AM-8PM. TEXT ANY TIME.
6 2009 Honda Odyssey EX-L,https://newyork.craigslist.org/stn/cto/d/2009-honda-odyssey-ex/6768080211.html,6900,,,111000,excellent,
  6,fwd,gas,blue,full-size,clean,automatic,mini-van,2009,honda,odyssey,        Selling a 2009 Odyssey ex-L. Very well maintained and very
  clean. Heated leather seats backup camera third row with rear heat and A/C rear DVD entertainment. Pretty much every option but navigation.
  Brand new front tires and front end work done recently. No check engine light on the dash. Any questions please call or text anytime. No
  emails. Thanks
```

3) Joining the Data Sets for Informed Learning

After the first two data sets (‘fuel_simple.csv’ and ‘vans.csv’) are created, the files can be joined and conclusions can be drawn.

I start by importing the proper files and creating custom schema. The custom schema are important for column manipulation/ running statistical analysis. I also load the two csvs into data frames.

```
# Join the data sets
from pyspark.sql.types import *
from pyspark.sql.functions import expr, desc, col
from pyspark.sql.types import LongType, StringType, StructField, StructType,
BooleanType, ArrayType, IntegerType, FloatType

# Custom schemas
#cylinders,displ,drive,fueltype,make,model,ucity,uhighway,transmission,vclass,year,
rn
fields = [StructField("cylinders",FloatType(),True),
StructField("displ",FloatType(),True),
StructField("drive",StringType(),True),StructField("fueltype", StringType(),
True),StructField("make", StringType(),True),
StructField("model",StringType(),True), StructField("ucity",FloatType(),True),
StructField("uhighway",FloatType(),True),
StructField("transmission",StringType(),True),
StructField("vclass",StringType(),True), StructField("year",IntegerType(),True),
StructField("rn",IntegerType(),True)]
fuelSchema = StructType(fields)
#title,url,price,address,vin,odometer,condition,cylinders,drive,fuel,paint_color,si
ze,title_status,transmission,type,year,make,model,description
fields2 = [StructField("title",StringType(),True),
StructField("url",StringType(),True),
StructField("price",FloatType(),True),StructField("address", StringType(),
True),StructField("vin", StringType(),True),
StructField("odometer",FloatType(),True),
StructField("condition",StringType(),True),
StructField("cylinders",FloatType(),True), StructField("drive",StringType(),True),
StructField("fuel",StringType(),True),
StructField("paint_color",StringType(),True),
StructField("size",StringType(),True),
StructField("title_status",StringType(),True),
StructField("transmission",StringType(),True),StructField("type",StringType(),True)
, StructField("year",IntegerType(),True),StructField("make",StringType(),True),
StructField("model",StringType(),True),
StructField("description",StringType(),True)]
dataSchema = StructType(fields2)

# Loads csv's to data frames
df_fuel = spark.read.format("csv").option("header",
"true").schema(fuelSchema).load("fuel_simple.csv")
df_craigslist = spark.read.format("csv").option("header",
"true").schema(dataSchema).load("vans.csv")
```

Next, create tables and join on make, model, year, transmission, and cylinders. This join should be good, since a lot of cleaning took place in the two data sets.

```
# Create tables to query using SQL
df_fuel.createOrReplaceTempView("fuel")
```

```
df_craigslist.createOrReplaceTempView("data")

# Joins
#cylinders,displ,drive,fueltype,make,model,ucity,uhighway,transmission,vclass,year,
rn
#title,url,price,address,vin,odometer,condition,cylinders,drive,fuel,paint_color,si
ze,title_status,transmission,type,year,make,model,description
df_combined = spark.sql("SELECT
data.title,data.make,data.model,data.year,data.transmission,fuel.ucity,fuel.uhighwa
y,fuel.vclass,data.size,fuel.drive,fuel.fueltype,data.cylinders,data.fuel,fuel.displ
l,data.url,data.price,data.odometer,data.condition,data.address,data.vin,data.paint
_color,data.title_status,data.type,data.description FROM data LEFT OUTER JOIN fuel
ON fuel.make=data.make AND fuel.model=data.model AND
fuel.transmission=data.transmission AND fuel.year=data.year AND
fuel.cylinders=data.cylinders")
```

Run new queries for useful information! I cleaned the data some more so only utility vans popped up.

```
# Create table for sql queries
df_combined.createOrReplaceTempView("query")

# Return useful information
df_ans = spark.sql("SELECT make,model,year,price,ucity,uhighway,vclass,condition
FROM query WHERE uhighway IS NOT NULL AND (vclass LIKE '%van%' OR description LIKE
'%sprinter%') AND condition!='fair' AND vclass NOT LIKE '%minivan%' AND description
NOT LIKE '%mini%' ORDER BY uhighway DESC, price")

# Print results
df_ans.show(30,False)

# Same info- but with url
df_ans = spark.sql("SELECT url FROM query WHERE uhighway IS NOT NULL AND (vclass
LIKE '%van%' OR description LIKE '%sprinter%') AND condition!='fair' AND vclass NOT
LIKE '%minivan%' AND description NOT LIKE '%mini%' ORDER BY uhighway DESC, price")

# Print results
df_ans.show(30,False)
```

Demonstration (Results and Visualization):

The goal of this search was to find the most fuel efficient car. The search was then sorted by price. Since the urls, take up a lot of room, to matching url is in the second table.

Here is a table of the top 20 results:

```
>>> df_ans = spark.sql("SELECT make,model,year,price,ucity,uhighway,vclass,condition FROM query WHERE
uhighway IS NOT NULL ORDER BY uhighway DESC, price")
>>> df_ans.show(20,False)
+----------+-------+----+-------+-------+--------+------------+---------+
|make      |model  |year|price  |ucity  |uhighway|vclass      |condition|
+----------+-------+----+-------+-------+--------+------------+---------+
|honda     |insight|2000|1500.0 |68.1881|89.2029 |two seaters |fair     |
|honda     |insight|2000|2400.0 |68.1881|89.2029 |two seaters |excellent|
|toyota    |prius  |2016|21900.0|76.0467|71.5838 |midsize cars|like new |
|chevrolet |cruze  |2018|9890.0 |40.5   |70.2    |compact cars|like new |
|chevrolet |cruze  |2017|12690.0|40.5   |70.2    |compact cars|excellent|
```

```
|lexus     |ct 200h|2015|17900.0|72.0295|69.6895 |compact cars|like new |
|toyota    |prius  |2011|4900.0 |71.8162|69.5514 |midsize cars|good     |
|toyota    |prius  |2012|6499.0 |71.7588|69.5142 |midsize cars|like new |
|toyota    |prius  |2012|8500.0 |71.7588|69.5142 |midsize cars|good     |
|toyota    |prius  |2014|13900.0|71.651 |69.4488 |midsize cars|like new |
|chevrolet |cruze  |2014|6000.0 |34.8   |66.2994 |midsize cars|excellent|
|chevrolet |cruze  |2014|10000.0|34.8   |66.2994 |midsize cars|excellent|
|chevrolet |cruze  |2014|10000.0|34.8   |66.2994 |midsize cars|excellent|
|toyota    |prius  |2006|2200.0 |66.6   |64.8    |midsize cars|good     |
|toyota    |prius  |2005|3990.0 |66.6   |64.8    |midsize cars|like new |
|toyota    |prius  |2009|4200.0 |66.6   |64.8    |midsize cars|good     |
|toyota    |prius  |2008|4800.0 |66.6   |64.8    |midsize cars|like new |
|toyota    |prius  |2009|5500.0 |66.6   |64.8    |midsize cars|good     |
|bmw       |328d   |2014|13999.0|41.5772|64.7919 |compact cars|like new |
|mitsubishi|mirage |2015|4995.0 |49.4465|63.3897 |compact cars|excellent|
+----------+-------+----+-------+-------+--------+------------+---------+
only showing top 20 rows

>>> df_ans = spark.sql("SELECT url FROM query WHERE uhighway IS NOT NULL ORDER BY uhighway DESC,
price")>>> df_ans.show(20,False)
+-----------------------------------------------------------------------------------------+
|url                                                                                      |
+-----------------------------------------------------------------------------------------+
|https://newyork.craigslist.org/jsy/cto/d/2000-honda-insight/6771193156.html              |
|https://newyork.craigslist.org/lgi/cto/d/2000-honda-insight/6759610035.html              |
|https://phoenix.craigslist.org/evl/cto/d/2016-toyota-prius-touring/6754419444.html      |
|https://phoenix.craigslist.org/nph/cto/d/2018-chevy-cruze-ls/6756759665.html            |
|https://newyork.craigslist.org/stn/cto/d/2017-chevrolet-cruze-lt-great/6761191984.html|
|https://phoenix.craigslist.org/evl/cto/d/2015-lexus-ct200h-warranty/6753137983.html     |
|https://gulfport.craigslist.org/cto/d/2011-toyota-prius/6742823416.html                 |
|https://newyork.craigslist.org/brk/cto/d/2012-toyota-prius-iii-leather/6768312410.html|
|https://newyork.craigslist.org/brk/cto/d/2012-prius-for-sale/6752384858.html            |
|https://phoenix.craigslist.org/nph/cto/d/2014-toyota-prius-wagon/6762160316.html        |
|https://newyork.craigslist.org/jsy/cto/d/2014-chevy-cruze68kfree-temp/6770824008.html  |
|https://cosprings.craigslist.org/cto/d/2014-chevy-cruze/6769888826.html                 |
|https://denver.craigslist.org/cto/d/2014-chevy-cruze/6769889821.html                    |
|https://newyork.craigslist.org/lgi/cto/d/2006-toyota-prius-hybrid-179k/6768891950.html|
|https://newyork.craigslist.org/que/cto/d/2005-toyota-prius-hybrid/6751978060.html       |
|https://newyork.craigslist.org/lgi/cto/d/2009-toyota-prius/6763990165.html              |
|https://newyork.craigslist.org/brk/cto/d/toyota-prius-2008/6770886287.html              |
|https://newyork.craigslist.org/fct/cto/d/2009-toyota-prius/6768673255.html              |
|https://newyork.craigslist.org/brk/cto/d/2014-bmw-328d-xdrive/6765070181.html           |
|https://phoenix.craigslist.org/nph/cto/d/2015-mitsubishi-mirage/6770293393.html         |
+-----------------------------------------------------------------------------------------+
only showing top 20 rows
```

Here is the top result:

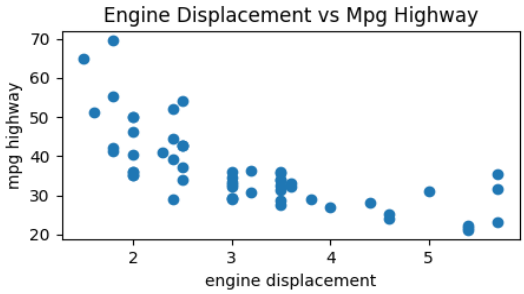
The goal of this particular search is to find the most fuel-efficient utility van. The search was then sorted by price. Since the urls, take up a lot of room, to matching url is in the second table.

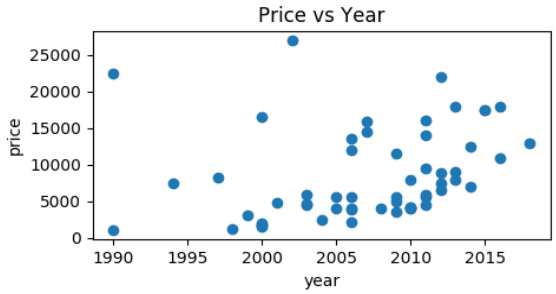
Here is a table of the top 20 results:

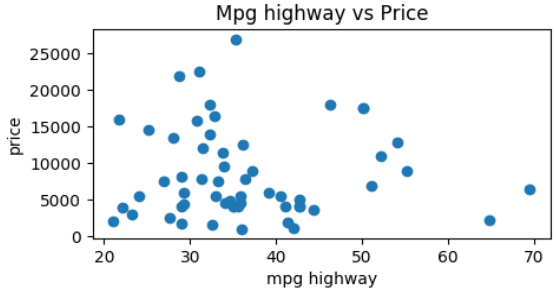
```
>>> df_ans = spark.sql("SELECT make,model,year,price,ucity,uhighway,vclass,condition FROM query WHERE
uhighway IS NOT NULL AND (vclass LIKE '%van%' OR description LIKE '%sprinter%') AND condition!='fair'
AND vclass NOT LIKE '%minivan%' AND description NOT LIKE '%mini%' ORDER BY uhighway DESC,
price").show(20,False)
+---------+-----------------------------------+----+-------+-------+--------+--------------------+---------+
|make     |model                              |year|price  |ucity   |uhighway|vclass              |condition|
+---------+-----------------------------------+----+-------+-------+--------+--------------------+---------+
|chevrolet|astro 2wd (cargo)                  |1992|1700.0 |18.8889|28.0    |vans                |excellent|
|chevrolet|astro awd (cargo)                  |2002|2900.0 |16.8   |25.5    |vans, cargo type    |excellent|
|chevrolet|express 1500/2500 2wd              |2004|8500.0 |17.0   |25.1    |vans, passenger type|good     |
|chevrolet|express 1500 2wd cargo             |2012|8995.0 |16.1   |24.3    |vans, cargo type    |good     |
|chevrolet|express 1500 2wd cargo             |2014|10800.0|16.1   |24.3    |vans, cargo type    |like new |
|chevrolet|express 1500 2wd cargo             |2012|12999.0|16.1   |24.3    |vans, cargo type    |good     |
|ford     |e250 econoline 2wd                 |2005|5500.0 |16.2   |24.1    |vans, cargo type    |good     |
|chevrolet|express 1500/2500 2wd              |1999|3000.0 |15.0971|23.1883 |vans, passenger type|good     |
|gmc      |savana 1500/2500 2wd (passenger)   |2000|11500.0|15.0   |23.1    |vans, passenger type|like new |
|gmc      |savana 1500  awd conversion (cargo)|2012|31995.0|15.8   |22.9    |vans, cargo type    |excellent|
|gmc      |savana 15/25 2wd conversion (cargo)|2003|3900.0 |15.2   |22.8    |vans, cargo type    |good     |
|chevrolet|express 1500 awd passenger         |2009|2500.0 |15.5   |22.4    |vans, passenger type|excellent|
|chevrolet|express 1500 2wd passenger         |2009|9600.0 |15.5   |22.3    |vans, passenger type|excellent|
|ford     |e150 econoline 2wd                 |2006|3900.0 |15.6   |22.2    |vans, cargo type    |excellent|
|ford     |e150 van ffv                       |2014|10995.0|15.1   |22.1    |vans, cargo type    |excellent|
|ford     |e150 van ffv                       |2013|12400.0|15.1   |22.1    |vans, cargo type    |like new |
|ford     |e250 van ffv                       |2012|6300.0 |15.0416|22.0323 |vans, cargo type    |excellent|
|ford     |e150 econoline 2wd                 |1995|3500.0 |14.0   |22.0    |vans                |good     |
|ford     |e150 van ffv                       |2011|16000.0|15.0   |21.8    |vans, cargo type    |like new |
|chevrolet|express 1500/2500 2wd              |2002|4500.0 |14.9   |21.7    |vans, passenger type|good     |
+---------+-----------------------------------+----+-------+-------+--------+--------------------+---------+
only showing top 20 rows

>>> df_ans = spark.sql("SELECT url FROM query WHERE uhighway IS NOT NULL AND (vclass LIKE '%van%' OR
description LIKE '%sprinter%') AND condition!='fair' AND vclass NOT LIKE '%minivan%' AND description
NOT LIKE '%mini%' ORDER BY uhighway DESC, price").show(20,False)
+-----------------------------------------------------------------------------------------+
|url                                                                                      |
+-----------------------------------------------------------------------------------------+
|https://seattle.craigslist.org/tac/cto/d/1992-chev-astro-van/6770311763.html             |
|https://newyork.craigslist.org/brk/cto/d/2002-chevrolet-astro/6771025824.html            |
|https://dothan.craigslist.org/cto/d/2004-chevrolet-express/6760727579.html               |
|https://newyork.craigslist.org/brk/cto/d/2012-chevrolet-express-g3500/6755026761.html    |
|https://newyork.craigslist.org/que/cto/d/2014-chevy-van-2500/6768996055.html             |
|https://newyork.craigslist.org/brk/cto/d/2012-chevy-express-lt3500/6762599292.html       |
|https://newyork.craigslist.org/lgi/cto/d/2005-ford-ek/6761206203.html                    |
|https://newyork.craigslist.org/stn/cto/d/1999-chevy-expresston-van/6756245121.html       |
|https://annarbor.craigslist.org/cto/d/gmc-pick-up-snow-plow/6755598263.html              |
|https://panamacity.craigslist.org/cto/d/2012-gmc-sirra-2500-hd-diesel/6745771162.html    |
|https://newyork.craigslist.org/lgi/cto/d/2003-gmc-3500-savana-cargo-van/6764026949.html  |
|https://newyork.craigslist.org/mnh/cto/d/2009-chevrolet-express-yf7/6768870177.html      |
|https://newyork.craigslist.org/stn/cto/d/2009-chevy-express-2500/6747826876.html         |
|https://newyork.craigslist.org/brk/cto/d/2006-ford-econolinepassenger/6770812061.html    |
|https://newyork.craigslist.org/que/cto/d/2014-ford-e150/6754851611.html                  |
|https://newyork.craigslist.org/brk/cto/d/2013-ford-e150-cargo-van/6763190512.html        |
|https://newyork.craigslist.org/que/cto/d/2012-cargo-ford-e250-van/6756066323.html        |
|https://newyork.craigslist.org/fct/cto/d/1995-ford-classic-conversion/6770768472.html    |
|https://newyork.craigslist.org/lgi/cto/d/2011-ford-e150-e85-flex-fuel/6764220601.html    |
|https://newyork.craigslist.org/que/cto/d/2002-chevy-express-3500-ext/6765329437.html     |
+-----------------------------------------------------------------------------------------+
only showing top 20 rows
```

Here is the top result!







Results:

The tool that I created can successfully search Craigslist for the vehicle that best matches their needs. This should decrease search time and make users feel more comfortable with their final decision It is also easy to see two like cars and compare the year, condition, and price. Since the data is user input and the code is not infallible it is not unlikely that errors will occur. More niche searches- like for vans may produce worse matches. Overall the code will produce useful user feedback and help view data trends.

Lessons Learned & Pros/Cons:

Running a more irregular search- like for vans can be more difficult. About half the data needed to be thrown out and many matches were wrong. Vans are more unique and may require better code insights and more fuel economy information.

From this project, I have a greater appreciation of what it takes to have good data. Data that is sourced from user input is rife- with irregularities. If data is missing points or contains bad information it can be difficult to root out and rooting out bad data can come at a cost. Below, I step through the metrics of running my example:

For one test case. When I only keep well formed data points that have: make, model, year, transmission, and cylinders I only get to keep:
700/1400

I looked deeper into this data set to see what could be improved. It is difficult to do anything without a make and a year.
96/1400 Craigslist adds did not include a make
170/1400 Craiglist adds did not include a year
21/1400 Craigslist adds did not include a make OR model (gives an idea of unique errors/overlap)

The next important piece of information is to find the model. Without the model we do not know anything about the vehicle. This is also important to join with the fuel database. I used a few methods to search the given Craiglist information for the model. With reasonable accuracy- where the make and

year are NOT null, 222 items have no reasonable match on a model. This data might be skewed since some models are not included in the fuel dataset- especially vans.
222/1400

Of the model data from the set above, the key makes that have missing models are listed below. All other makes have 5 or less missing models.
Gmc: 7/50 (7 missing gmcs out of the 50 provided)
honda: 27/218
ford: 95/332
chevrolet: 46/162

Ford and Chevrolet are difficult, since there are so many similarly named make and models. My code can a difficult time differentiating and picking the best match. There are also a good number of e350's and e450's missing from the fuel economy data set. This is important to note- but is not in the scope of this project to fix. Most of the issues with Honda were poor spelling. I got these spellings of Odyssey: odessy, Odysssey, Odessey, odisey. Since 80% of Honda's issues are from poor user input, these will be ignored for this project.

For a future time, more data could be collected on Ford and Chevrolet and added to the fuel economy data set:
https://www.fueleconomy.gov/feg/PowerSearch.do?action=noform&path=1&year1=1984&year2=2019&make=Ford&baseModel=E250%20Econoline&srchtyp=ymm&pageno=1&sortBy=Comb&tabView=0&rowLimit=200

When searching through initial results- I found that make, model, and year were not quite good enough to make a match. Also some matches were being displayed that did not make sense. One should be vary if a van is shown to get 50+mpg. Some of these poor results were from poor model matches. Others are from matches with the same make, model, year name that have very different expected mpg. In order to help weed out bad results, I also ran the join on number of cylinders and transmission. Every single Craigslist add included transmission information, but many were missing cylinder information. I ran code to search the description,but still 217/ 1400 include model information but NOT cylinders.
217/1400

These are just a few metrics/ considerations. This project is a success if the user wants Craigslist data matched with fuel economy. The user just has to realize that not all data is good and it is difficult to draw the intended conclusions from the data. I originally hoped to use machine learning to make guesses on un-matched Craigslist data, but feel there is not enough (or consistent) enough information supplied from Craigslist to be useful.

Future Work:

Get more niche searches working! I am interested in using my own code to search for a van. I would like the code to be able to produce good/ consistent results. There are other considerations like van-size that might be useful too.

It would be interesting to use Craigslist location search and make the same comparisons with vehicles around the United States. Some people might be willing to fly out to a location to see a vehicle they have been looking for.

Fuel economy data can be downloaded from the website- but it can also be imported to Python. I discovered this a little late in my project and think that would be a good addition. That would keep the fuel economy data up-to-date.

YouTube URLs:

2 minute: https://youtu.be/XTy6c-Z5twU
15 minute: https://youtu.be/CLf7mLvR4Hc

References:

Zoran's class notes

Fuel economy data: https://www.fueleconomy.gov/feg/download.shtml

Craigslist: https://newyork.craigslist.org/search/brk/cto

Craigslist Facts: https://expandedramblings.com/index.php/craigslist-statistics/

Initial scrapy setup: https://python.gotrained.com/scrapy-tutorial-web-scraping-craigslist/