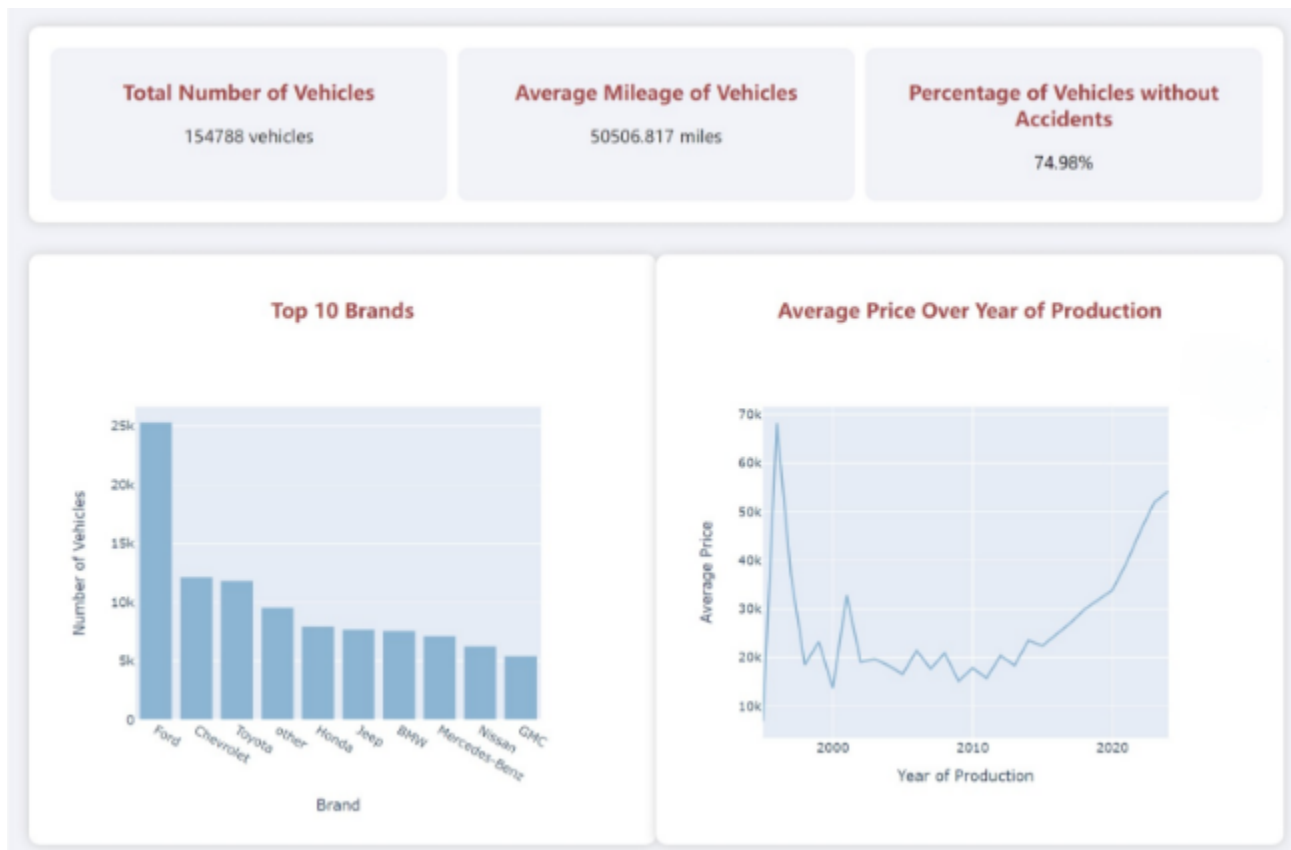


Data 602 Final Project

Analysis and Prediction of Used Vehicle Price

December 12 2023



DATA 602: Principles of Data Science

Professor Mohammad Taghi Hajiaghayi

University of Maryland College Park

Author:

Muyan Cheng

Qiao Qin

Ke Xu

1 Topic Discussion

After getting the Final Project Description and getting a good understanding of what the task is. It took us multiple meeting to come to an agreement on what our topic is going to be. We went through topics like Gold Price Analysis; Currency Exchange Rate analysis and Crime Rate Analysis. But ultimately, we landed on used car price analysis. As in our opinion, used car price are not very dependent on uncontrollable factors like world events and political policies as other topics are and therefore are more predictable than the other topics we have previously picked.

2 Introduction

After deciding on what our topic is, we got to looking for our data set right away. Unfortunately, there is no ready to go data set for us to use except for one API that was locked behind a pay wall. So we decided to scrape our own data, which would also give us toe right to choose what piece of data we deemed meaningful and important for our analysis.

After going through the used-car-selling websites, we decided that cars.com would be a good source to get our data from. After all, the website is called Cars.com. All jokes aside, it has been rated 9.0 out of 10.0 in terms of websites to buy or sell cars by MarketWatch, making it reasonably reliable in our opinions. Plus, it provides a lot of information for all the vehicles listed in a structured form, giving us a lot of information to train our model with.

3 Data Source and Data Collection

When we opened up cars.com, the website prompts you for more information on the car you are looking for, like the make(brand), model and your location. Since we are looking for a set of generalized information about all types of cars, we searched for cars of all brand and all model in a 10 mile area center from college park. And all the listing we got are in the form of small information card, here is an example(Figure 1):

Please click to see the example of the search result page

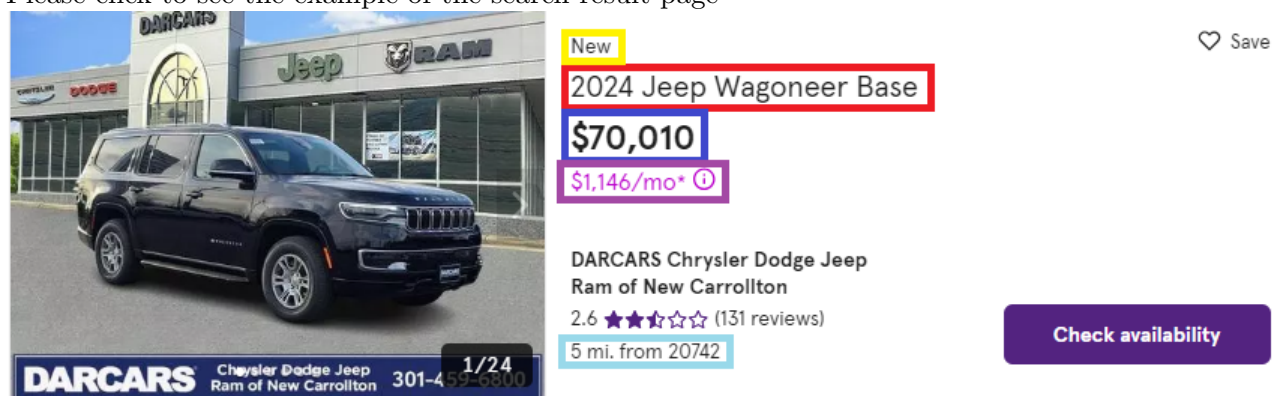


Figure 1: small info card

Circled in the example we provided above are all the information we can obtain from the listing, five piece in total and only three piece of them are useful. This is clearly unacceptable, so we opened the actual page of the listing and try to gather more information there. After going through the listing page, we managed to find two tables containing information that we deemed to useful. The

listing page in its entirety is too long for us to show here, so we will include an example of the useful tables:

Please click to see the example

As shown in the example, there are 12 piece of information in 3 sections that we have deemed to be useful. But after looking through more listings, we realized that the vehicle history section actually contains 5 piece of information rather than 4. For the sake of accuracy, we collected all 5 pieces. Totaling 13 pieces of information for each listing.

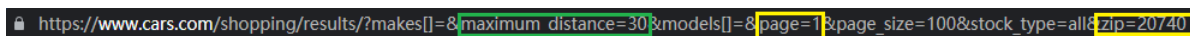
After finding the useful information, our next task is to scrape the selected data for a large number of listings. So logically, we brought our attention to the URL of the listing page, where we got all of our information, here is the example(Figure 2):



<https://www.cars.com/vehicledetail/b9a49fe9-647a-4844-8375-b1183e7f20e3>

Figure 2: Vehicle Detail page URL

As I have circled in the example above, Cars.com uses a unique ID for each of their listings, which we can use to scrape the data we need and We can get a list of listing IDs from the search result page. The search result displays 100 pages of results with 100 listings on each page for any zip code we have selected. After looking at the URL of the search result(Figure 3), we realized that we can go through all the pages by just changing the page number in the URL, we can also change the zip code and maximum distance to search for cars in different areas by changing the zip code and maximum distance part of the URL.



[https://www.cars.com/shopping/results/?makes\[\]=8&maximum_distance=30&models\[\]=8&page=1&page_size=100&stock_type=all&zip=20740](https://www.cars.com/shopping/results/?makes[]=8&maximum_distance=30&models[]=8&page=1&page_size=100&stock_type=all&zip=20740)

Figure 3: Search Result page URL

We got our list of zip codes from www.ciclt.net. A web-based tools to improve membership management, event registration and political involvement and advocacy. We believe that if www.ciclt.net is good enough for political involvement, it should be sufficient for our analysis.

Because of the nature of the used vehicle market, the price of most vehicle is not updated daily, and only a small amount of them are updated weekly(around 25% by our estimation). Making scrape our data based on a pre-existing set of zip code infeasible, so we decided to scrape our data from different states every week. We understand that this is not exactly what the instruction asked us to do, but since we didn't restrict our data source to any brand or models, we believe our way of obtaining data would be acceptable for this project.

With the help of Request and urlopen from urllib along with soup from BeautifulSoup, we were able to get a list of zip codes of the states we have selected. We then searched for listing in a 10 mile radius of every one of those zip codes and record them. After removing the duplicate IDs from our record, the IDs were fed into another Request algorithm to collect the data we need and store them in a Pandas Data Frame. With each row containing the information we have collected for each vehicle. A few thing we thought was worth mentioning about the data collection process was:

- As we mentioned before, many listings didn't have all 13 piece of data we wanted, so we filled in the missing ones with an empty string and we will decide what to do with them later.

- We found out we can not scrape the whole page and filter them later, since the Ram needed to store all those pages are too much. Forcing us to filter as we scrape
- Even though we have chosen 10 miles as the radius of our search, there is still a lot of duplicate IDs (roughly 50% of all IDs collected were duplicates), there are not as many cars listed on Cars.com as we initially thought.

4 Initial Observation

Now that we have stored all of our data in the Pandas Data Frame(Figure 4), we can start analyzing what we have obtained.

	new/used	name	milage	price	id	fuel consumption	drive mode	fuel type	accident history	clean title	one owner?	personal use	factory recall
0	Used	2016 Jaguar F-TYPE S	25,279 mi	\$38,400	4f40a2dd-4080-4f32-84bf-c3bce4c0a7ca	19-27	Rear-wheel Drive	Gasoline	At least 1 accident or damage reported	NaN	No	No	NaN
1	NaN	NaN	NaN	NaN	8b1ef161-c7b0-4ae4-b5a6-8441b9510584	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Used	2022 Nissan Pathfinder SL	28,800 mi	\$32,994	1b6dae71-3160-4c99-8477-edda938c424f	21-27	Four-wheel Drive	Gasoline	At least 1 accident or damage reported	NaN	Yes	Yes	At least 1 open recall reported
3	Used	2021 Jeep Grand Cherokee SRT	35,230 mi	\$62,500	fa45d263-0171-4222-808f-c2b85c38774	13-19	Four-wheel Drive	Gasoline	None reported	NaN	No	Yes	NaN
4	Used	2022 Cadillac CT4-V V-Series	14,975 mi	\$44,995	3a98e7d4-71cd-4a88-87f1-578f4137d119	--	All-wheel Drive	Gasoline	None reported	NaN	Yes	Yes	NaN
...
154783	Acura Certified	2023 Acura RDX A-Spec	4,966 mi	\$43,370	79a45d1f-1fac-477a-a9f4-d9733382296d	22-27	Front-wheel Drive	Gasoline	None reported	NaN	No	No	NaN
154784	Used	2023 Land Rover Defender 75th Limited Edition	190 mi	\$86,998	2096e10d-8e93-4994-b4b6-00b019603bb	18-23	Four-wheel Drive	Gasoline	None reported	NaN	Yes	Yes	NaN
154785	Used	2015 Hyundai Santa Fe Limited	98,361 mi	\$14,599	246d0cac-b693-4f5a-ac66-514b5926813	18-25	Front-wheel Drive	Gasoline	None reported	NaN	No	Yes	NaN
154786	Used	2016 Lexus NX 200t Base	48,736 mi	\$30,775	0466e59e-6ca5-40f0-a51c-1666585db15c	22-28	All-wheel Drive	Gasoline	None reported	Yes	No	Yes	NaN
154787	Used	2023 Kia EV6 Wind	498 mi	\$38,488	dfa5e68f-883f-4767-bce4-932aa111be41	NaN	NaN	NaN	None reported	NaN	Yes	Yes	NaN

154788 rows x 13 columns

Figure 4: An example of the dataframe we constructed with our data

Right off the bat, we realized there are rows containing only empty strings like the second row. this happens because either the listing didn't contain enough data(less than 6/13) or the vehicle was taken off the website, likely because it is sold.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 154788 entries, 0 to 154787
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  ---                ---
0   new/used              143495 non-null object
1   name                  143495 non-null object
2   milage                143495 non-null object
3   price                 143495 non-null object
4   id                    154788 non-null object
5   fuel consumption      137772 non-null object
6   drive mode            137772 non-null object
7   fuel type             137772 non-null object
8   accident history      141468 non-null object
9   clean title           30661 non-null object
10  one owner?            141455 non-null object
11  personal use          141468 non-null object
12  factory recall        21019 non-null object
dtypes: object(13)
```

Figure 5: how much non-null object we have in each column

After removing all the meaningless rows from our data frame, we moved our attention towards unacceptable columns. By using the .info() function of our data frame, we were able to see how many non-null object does each column contain(Figure 5). We then removed the columns that doesn't contain enough data. In this case, we removed 'clean title' and 'factory recall'

5 Feature Engineering

The first step of Feature engineering for us is to fill in all the missing data. We filled all the integer based column with the mean of the column Because filling it with 0 would influence the result too much. We filled the string based column with the mode of the all the element in that column, because leaving it empty is simply not an option. Almost all the data we currently have are categorical strings, which can not be used in its current form. Therefore, we decided to encode them as new features to train our models with, we used one-hot method to accomplish this transformation. Not only because one hot is commonly used in machine learning, but also because the added efficiency might be needed consider the size of our data set. Here are short explanations on what we did and what we learned.

5.1 new/used

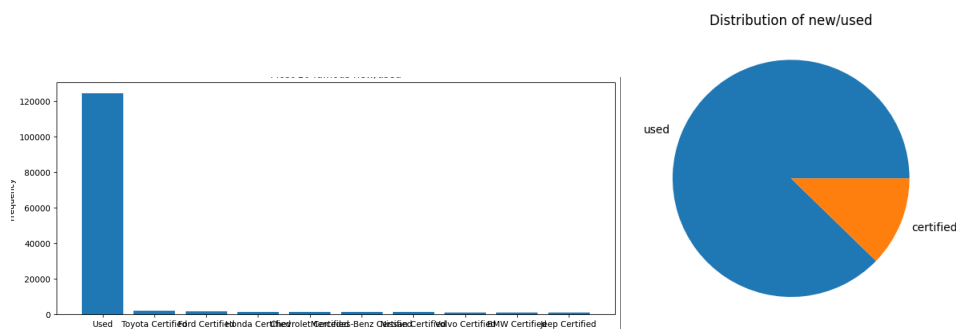


Figure 6: how much data belongs to each group

As Figure 6 suggested:most of our data came from used vehicles, and the rest are brand certified. It is unreasonable to encode the new/used column in its current form especially with one-hot method. So we decided to group all the brand certified vehicle into one group named 'certified'. Making the new/used column binary and easy to work with(Figure 7). The pie chart on the right displays the distribution of our data after our conversion, around 87% of our data came from used cars and only 12% of the data came from certified vehicle. we then plotted the average price of all used cars and certified cars to find out the level of influence new/used has on the price the vehicle. As the Figure 8 suggested, the influence is not very sever.

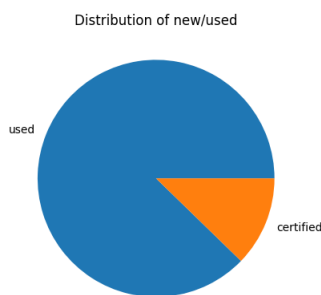


Figure 7: distribution of the two group

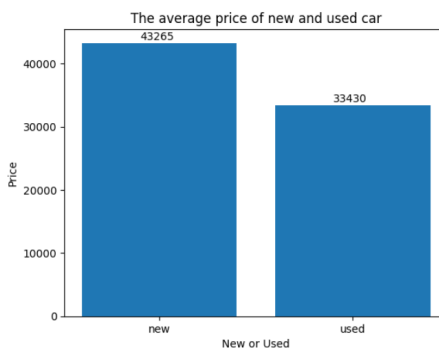


Figure 8: Average price of the two group

5.2 brand

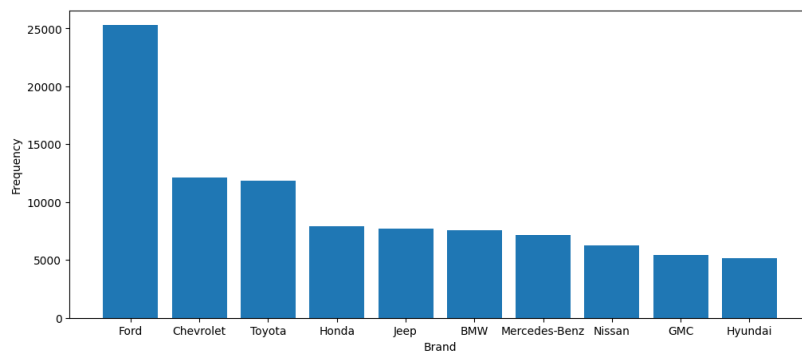


Figure 9: Number of data belong to each brand

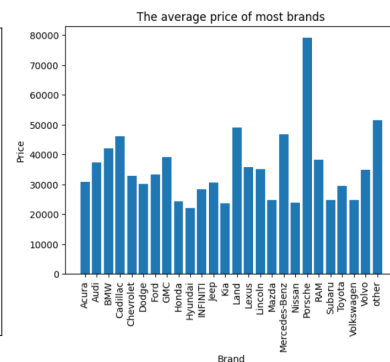


Figure 10: average price of each brand

It was the same story with data distribution in brand as Figure 9 suggested. Where the majority of the data belong to a few groups. In fact, with the help of Pandas Data Frame functions, we were able to find out that 93.83% of all data belong to 25 brands, so we decided to keep those 25 brand and set everything else as 'other'. All three of us thought most of our samples would belong to either Toyota or Honda, but none of us expected Ford to take the crown.

we also plotted the average price of each brand to find the level of influence(Figure 10). Compared to new/used, brand definitely has a greater influence on the price of the vehicle. As the average price of different brand vary wildly.

5.3 millage and price

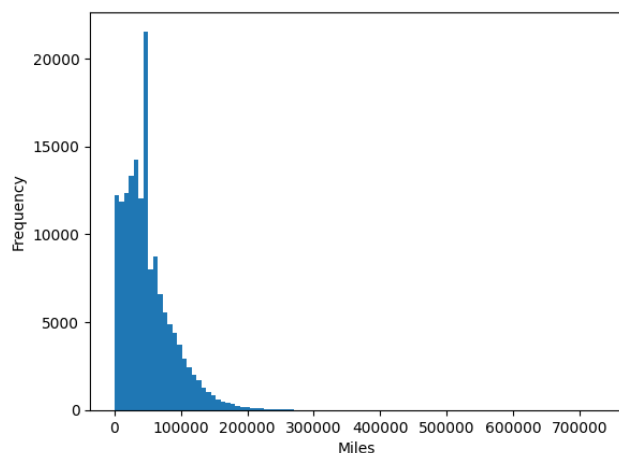


Figure 11: Number of data belong to each brand

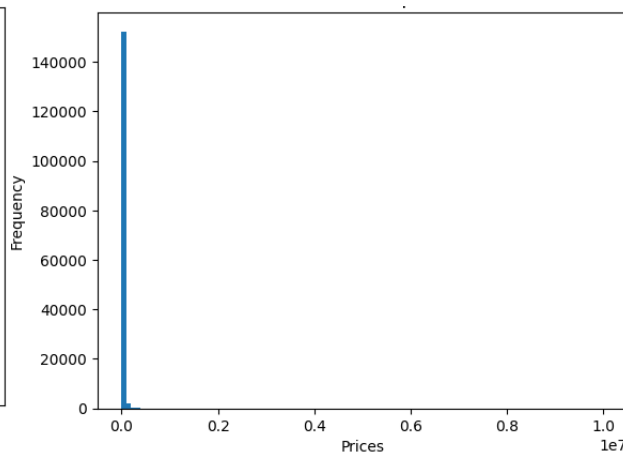


Figure 12: average price of each brand

We noticed from Figure 11 and Figure 12 that the price was very concentrated at around 20,000 dollars, and the mileage was concentrated around 55,000 miles. Making us believe that on average, cars.com would be a good place to shop for second hand vehicles.

5.4 drive mode

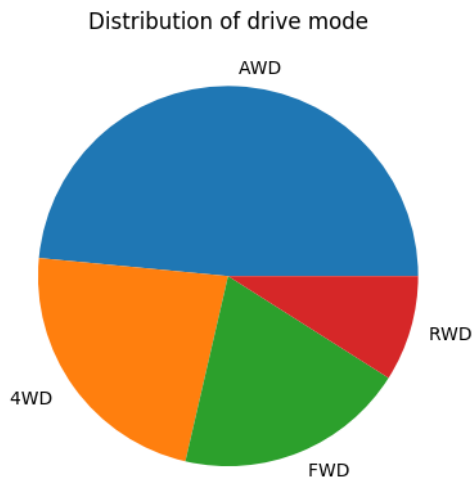


Figure 13: Pie graph of each drive mode

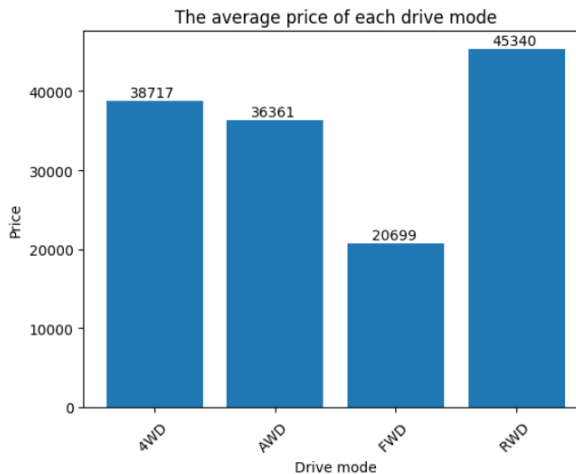
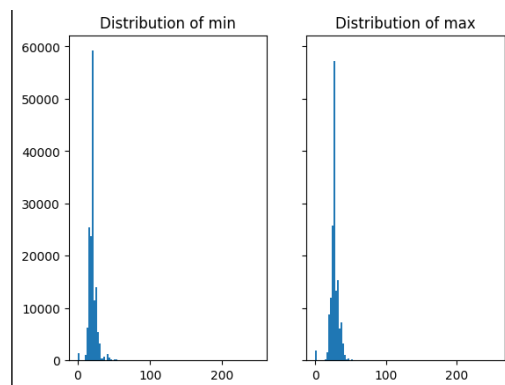


Figure 14: Average price of each drive mode

We didn't need to change anything about our drive mode column, the data were evenly distributed into four groups(Figure 13), perfect for the one-hot encoding method. It's interesting that almost half of all vehicle we collected data from is All Wheel Drive(AWD). Because AWD vehicles are generally more expensive, not only to purchase, but also to maintain. Especially compare to Front Wheel drive(FWD) and Rear Wheel drive(RWD) vehicle, which only totaled a bit more than 25% surprisingly.

As Figure 14 suggest, Four Wheel Drive(4WD) and All Wheel Drive(AWD) are similar when it comes to price, but Front Wheel Drive(FWD) and Rear Wheel Drive(RWD) influence the price of the vehicle severely.

5.5 fuel consumption



We can tell from Figure 15 the above plot the fuel consumption number is again, relative concentrated at around 20.85 for the min and 27.21 for the max. This consumption number is extremely inefficient, however, considering a large number of our data came from US-made vehicles, these number makes perfect sense.

Figure 15: Number of data belong to each brand

5.6 fuel type

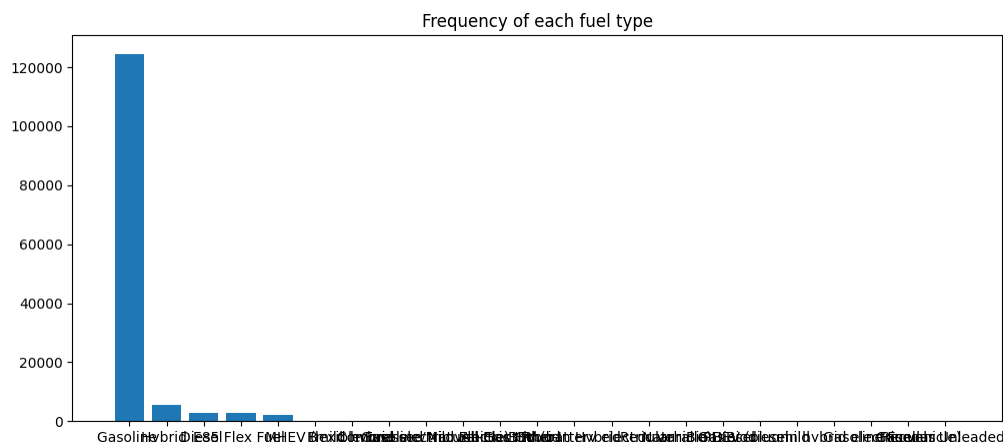


Figure 16: number of data belong to each fuel type

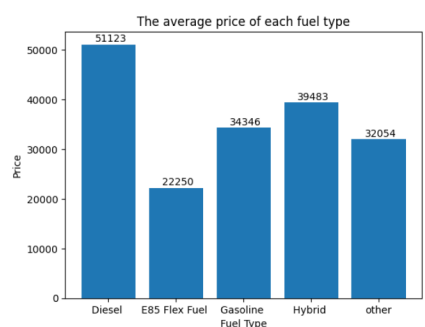


Figure 17: Average price of each fuel type

of our sample consumes gasoline or are hybrid). I don't believe there is enough evidence to draw any conclusion.

Once again, we encountered an column that need to be converted. We kept the first four group and grouped up everything else into the fifth group. This is still rather inefficient, but this is the most reasonable way of grouping to maximize accuracy. As Figure 16 suggest, over 80%(120,000/150,000) of our data came from gasoline vehicles. Not a surprise as our data came most from used vehicles and alternative fuel vehicle has only been introduced rather recently. Even though according to Figure 17, the fuel type a vehicle consumes has a rather noticeable amount of influence on the vehicle price. But based on our sample distribution(almost all

5.7 accident history

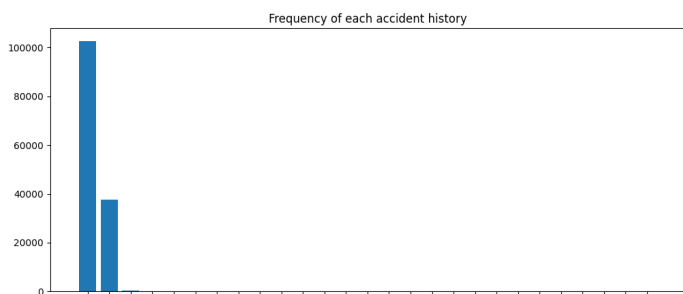


Figure 18: number of data had accident and not

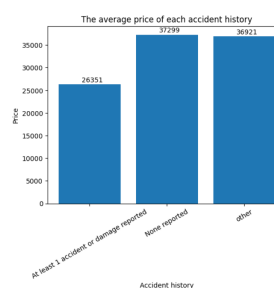


Figure 19: average price based on accident history

In Figure 18, the first column represent 'no accident reported' and the second column represent 'at least one accident reported', with the barely third column representing all the other data grouped into one named 'other'. We can see that even though most of the vehicle are reported to

be accident free, the amount of vehicle that has at least been through one accident is higher than we expected. more than 25% of our data came from vehicle that has at least one accident. As Expected, whether a vehicle has been through an accident influence the vehicle price a lot.(Figure 19)

5.8 number of owners and personal use

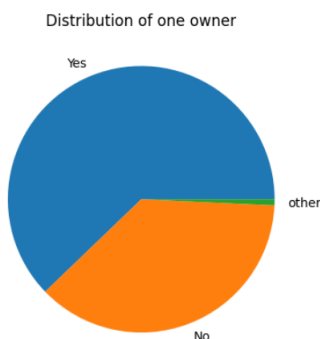
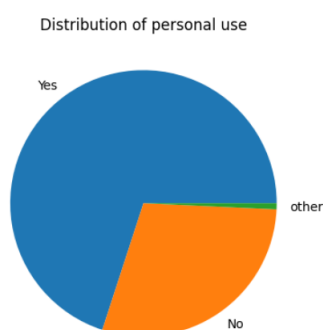


Figure 20: Pie of personal use Figure 21: pie of one owner ve- Figure 22: average price of vehi-
 cile cle grouped by ownership num-
 ber

As shown by Figure 20 and Figure 21, the two columns are very similar in the seance that they both consists of three group: 'Yes', 'No' and 'others'. Even the distribution of the three group are rather similar.

This plot was a little surprising, since we thought whether a vehicle has only had one owner would influence the vehicle price a lot. But based on Figure 22, whether a vehicle have only had one owner wouldn't influence the vehicle price as much as we expected.

6 Model Selection

Since we have chose to encode our data with the one-hot method, the features we have obtained are purely binary. Leading us to test our training data with the linear regression model. However, the linear Regression model gave us prediction with very high training error and after closer inspection, we concluded that the linear regression model has an underfitting problem. In another word, we need a more complex model. We then tested the gradient boosting model with our training data. The training error was a lot lower as we expected, but the testing error was too high for our comfort. We believe the gradient boosting model has a overfitting problem. We understood that obtaining more data might fix the overfitting problem, but based on our data size, we believed it's difficult for us to obtain enough data to fix our problem. After two failed attempts, we decided to test our training data with a neural network to see how the model performs.

7 Model Construction

We used nn; optim and utils from Pytorch to create our neural network, we had 5 layers with different function like Dropout, Batchnormld weaved between the layers. Here is a walk through and sketch(Figure 23) of our network:

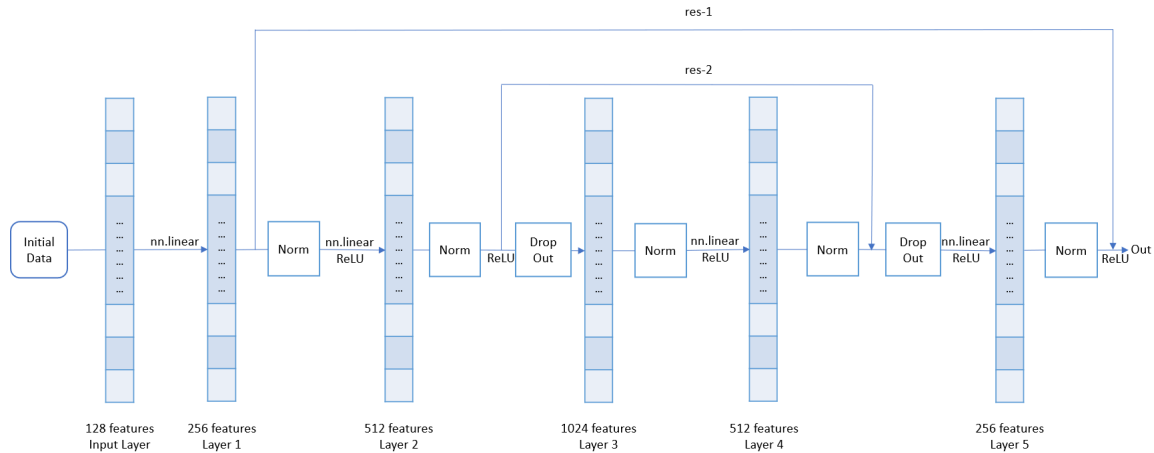


Figure 23: A rough sketch of our network

- Before we begun, we used optim.AdamW with $lr = 0.02$ to create an optimizer for our model. 0.02 was chosen because it's neither too big that it will cause our model to skip over the optimal solution, nor is it too small that it will take forever to train our model with our data size.
- Consider the hardware we are running this network on, we didn't divide our data set into batches. In another word, we are training our network with all the data we have every time. This might be time consuming, but it will produce a better estimator so we believed it's worth it.
- with the help of nn.linear, we transform our features into an array and feed it into the first layer of our network. We would save result our first layer produces as residual 1 for later use. We will use Resnet in our network to minimize the loss of accuracy. Eventhough our network is not very deep, having some extra measure wouldn't hurt.
- we would then use nn.BatchNorm1d to try to minimize the influence of any large number on our result.
- After all of our data is normalized, we would use nn.ReLU as our activation function to push our result into the the second layer.
- For the result from the second layer, we would normaliz our result again with nn.BatchNorm1d() before saving it as the second residual for our Resnet operation.
- Before we feed our normalized result from layer 2 into layer 3 to continue training our network, we used nn.Dropout() to drop a certain percentage of the said result to prevent overfitting.
- All the preparation is done, we push the result from layer 2 into layer 3 with nn.ReLU()

- We would normalize our result from layer 3 and directly push our result into layer 4, again with `nn.ReLU()`
- After obtaining our result from layer 4, we normalized the result before bringing in residual 2, then using `nn.Dropout()` and push what is left into layer 5
- We would repeat what we did after layer 4, normalize our data before bringing in residual 1. Then use `nn.ReLU()` for the last time to obtain our prediction.

8 Model Training

We decided to feed our data into our neural network 2000 times and record the training and testing loss every 50 times to visualize the improvement. Here is what we got(Figure 24 and Figure 25):

```
epoch:0 training loss = 9.484848022460938,test loss = 9.5321626663208
epoch:50 training loss = 9.215303421020508,test loss = 9.290000915527344
epoch:100 training loss = 9.195377349853516,test loss = 9.269281387329102
epoch:150 training loss = 9.184779167175293,test loss = 9.259315490722656
epoch:200 training loss = 9.175662994384766,test loss = 9.25273609161377
epoch:250 training loss = 9.160301208496094,test loss = 9.225996017456055
epoch:300 training loss = 9.14815902709961,test loss = 9.189255714416504
epoch:350 training loss = 9.113619804382324,test loss = 9.188393592834473
epoch:400 training loss = 9.037347793579102,test loss = 9.213704109191895
epoch:450 training loss = 8.976736068725586,test loss = 9.162004470825195
epoch:500 training loss = 8.900739669799805,test loss = 8.939314842224121
epoch:550 training loss = 8.844852447509766,test loss = 9.079914093017578
epoch:600 training loss = 8.801857948303223,test loss = 8.959667205810547
epoch:650 training loss = 8.76110553741455,test loss = 8.927393913269043
epoch:700 training loss = 8.729384422302246,test loss = 8.741996765136719
epoch:750 training loss = 8.713932991027832,test loss = 8.728058815002441
epoch:800 training loss = 8.703980445861816,test loss = 8.715592384338379
epoch:850 training loss = 8.69427490234375,test loss = 8.749910354614258
epoch:900 training loss = 8.684149742126465,test loss = 8.69658660886719
epoch:950 training loss = 8.678755760192871,test loss = 8.723184585571289
epoch:1000 training loss = 8.676798820495605,test loss = 8.704527854919434
epoch:1050 training loss = 8.666694641113281,test loss = 8.640482902526855
epoch:1100 training loss = 8.666211128234863,test loss = 8.69034481048584
epoch:1150 training loss = 8.657840728759766,test loss = 8.637208938598633
epoch:1200 training loss = 8.65395450592041,test loss = 8.617786407470703
epoch:1250 training loss = 8.656709671020508,test loss = 8.599568367004395
epoch:1300 training loss = 8.649298667907715,test loss = 8.5921049118042
epoch:1350 training loss = 8.648763656616211,test loss = 8.669276237487793
epoch:1400 training loss = 8.641443252563477,test loss = 8.65392780303955
epoch:1450 training loss = 8.638916015625,test loss = 8.660441398620605
epoch:1500 training loss = 8.645833015441895,test loss = 8.651449203491211
epoch:1550 training loss = 8.638946533203125,test loss = 8.636441230773926
epoch:1600 training loss = 8.642093658447266,test loss = 8.674224853515625
epoch:1650 training loss = 8.635700225830078,test loss = 8.629197120666504
epoch:1700 training loss = 8.62874698638916,test loss = 8.658665657043457
epoch:1750 training loss = 8.63154125213623,test loss = 8.615553855895996
epoch:1800 training loss = 8.628732681274414,test loss = 8.711620330810547
epoch:1850 training loss = 8.62224006652832,test loss = 8.656779289245605
epoch:1900 training loss = 8.622858047485352,test loss = 8.653131484985352
epoch:1950 training loss = 8.623217582702637,test loss = 8.634129524230957
```

Figure 24: Training progress

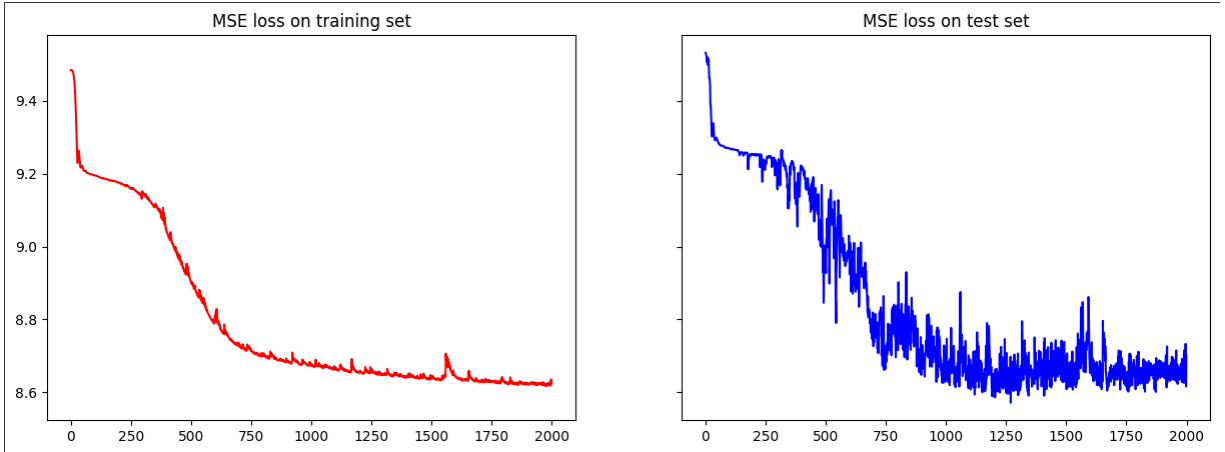


Figure 25: A Visualization of our progress

As shown in the example above, both the training and testing loss stagnates after around 1250 training cycles. Our best training error is around 8.622 and best testing error is around 8.592

9 Visualization

We created a dashboard to present our results. We used the dash library in python to make this dashboard, Dash is a Python library that helps us create interactive web applications. It uses modern UI elements, Python code and Plotly.js charts to create various types of web applications. The image below(Figure 26) shows the dashboard we have created:

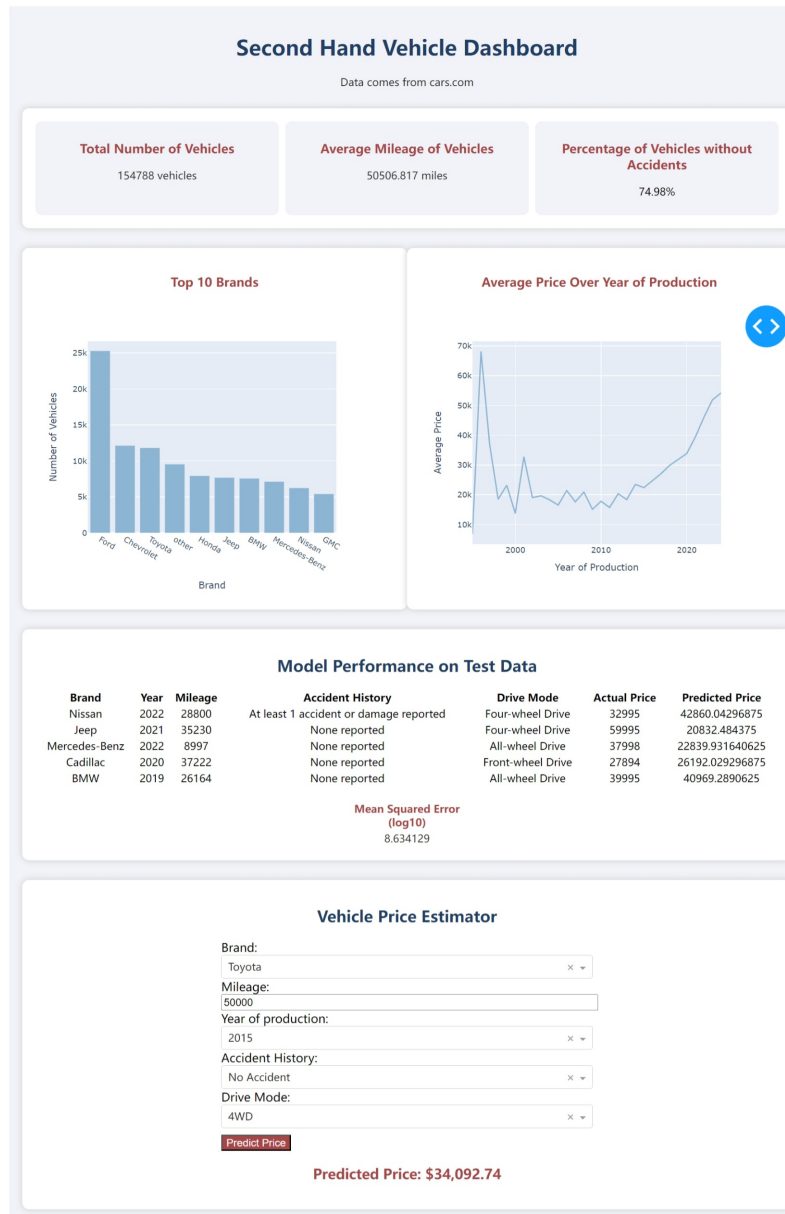


Figure 26: A peek at our web page

Our dashboard is divided into four parts:

The first part is the key data from our data set used in training our model, including the number of total INSTANCES in the training set, the average mileage of the vehicles and the percentage of vehicles with no accident history, which reflect a basic picture of our training data(Figure 27).

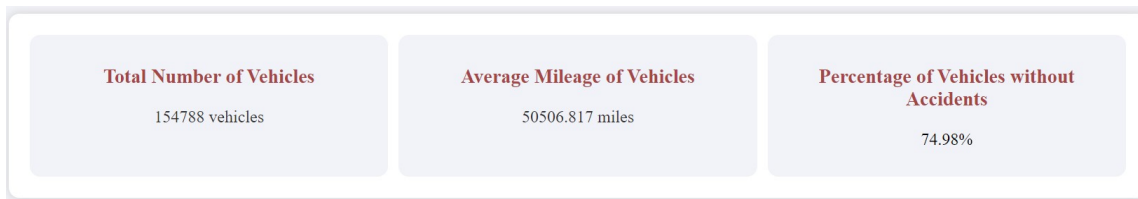


Figure 27: Key Datas

The second part is composed of two plots, the first plot is the top ten brands in terms of the number of vehicles in our training data, which can be seen that the number of Fords is the highest; the second plot is the relationship between the year of production of vehicles and the average price, which shows in general, used cars gets more expensive as time goes on. It shows that the newer the vehicle is, more expensive it is.

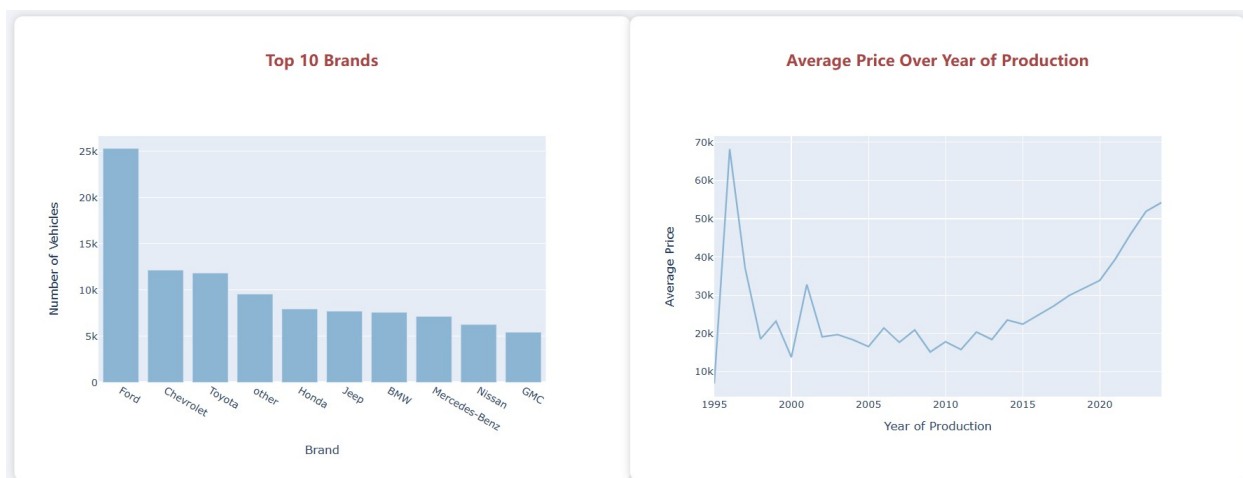


Figure 28: Example Graphs

The third part shows an example of prediction made by our model, and the table shows the most important features of the first five instances, the actual price and the predicted price, and outputs the mean-square error of the model's prediction at the end.

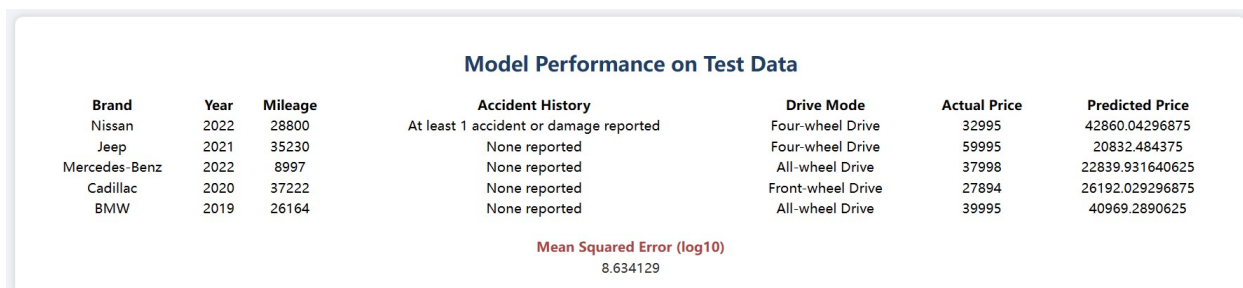


Figure 29: Test Performance

The fourth part is an interactive price predictor, where you can input five parameters: make(brand) of the vehicle, mileage, year of production, accident history, and driving mode, and click "predict price" to return a predicted price.

Vehicle Price Estimator

Brand: Toyota

Mileage: 50000

Year of production: 2015

Accident History: No Accident

Drive Mode: 4WD

Predict Price

Predicted Price: \$34,092.74

Figure 30: Estimation page

For example, if we want to predict the price of a 2020 BMW 4WD Model with 100,000 miles and no accident history, we will enter the information as shown in the graph. Click "predict price" and the predicted price will be returned as \$30,550.46.

Vehicle Price Estimator

Brand: BMW

Mileage: 100000

Year of production: 2020

Accident History: No Accident

Drive Mode: 4WD

Predict Price

Predicted Price: \$30,550.46

Figure 31: Example prediction

In this section, in order to make the interface interactive, we use the callback function, which is called by clicking on "predict price" after entering the brand, mileage, year of manufacture, accident history and driving mode. The callback function reads in the input data, and since our model has up to 11 features, we choose to fill in the rest of the non-input features with their means or plurals, and then put them into the model for prediction, and output the final result.

When we run the program locally, it will start a local server listening on the specified port 8050, and we can access the dashboard directly by typing `http://127.0.0.1:8050/` or `http://localhost:8050/` in the local browser.

Here is the link to our GitHub Project, where you can find all the related code:
<https://github.com/MuyanCheng/DATA602-Final-Project>

Appendix A: Useful Search Result Page

Please click to return

New and used vehicles for sale

Sort by
Best match

The best cars of 2023!
See our experts' top picks

BEST CARS OF 2023 ★

Never miss a car!
Get email alerts on this search.

Email

Subscribe

By subscribing, you agree to our [Privacy Notice](#) and [Terms of Service](#).

8,629 matches Save search

Basics

Search within
10 miles

ZIP
20740

New/used
New & Used

Make
All makes

Price & payment **NEW**

Full Price

Monthly

Min price
Lowest

Max price
Highest

Deal rating

☐ **Great Deal** (740)


☐ **Good Deal** (1,618)

☐ **Fair Deal** (443)

Min year

Max year

LEXUS | DARCARS LEXUS OF SILVER SPRING



(301) 680-0400

New


2023 Lexus RX 350 Luxury

\$61,695 MSRP \$64,195

\$1,829/mo* ⓘ

DARCARS Lexus Of Silver Spring
4.7 ★★★★★ (1,760 reviews)
📍 Silver Spring, MD (5 mi.)

Check availability



Mercedes-Benz of Silver Spring 1/22

New

2024 Mercedes-Benz GLS 450 4MATIC


\$95,115

\$2,819/mo* ⓘ

Hot Car

Mercedes-Benz of Silver Spring
4.9 ★★★★★ (2,008 reviews)
📍 Silver Spring, MD (6 mi.)

Check availability



New

2024 Mercedes-Benz GLE 350 Base 4MATIC

\$70,245

\$2,082/mo* ⓘ

Mercedes-Benz of Silver Spring
4.9 ★★★★★ (2,008 reviews)

Check availability

17

Appendix B: Useful Information from the listing page

Please click to return

Used

2015 BMW 750 750Li

83,437 mi.

\$17,977

\$1,018 price drop

\$294/mo* ⓘ

🔗 Good Deal | \$2,475 under

Home Delivery

Virtual Appointments

Basics

Exterior color	Space Gray Metallic
Interior color	Black
Drivetrain	Rear-wheel Drive
MPG	16–25 ⓘ
Fuel type	Gasoline
Transmission	Automatic
Engine	4.4L V-8 gasoline direct injection, DOHC, Double VANOS variable
VIN	WBAYE8C56FD780871
Stock #	21966
Mileage	83,437 mi.

Vehicle history by AutoCheck

Get the AutoCheck Report 🔗

Accidents or damage	None reported
Clean title	Yes
1-owner vehicle	Yes
Personal use only	No