ReadMe (Practical Application 11.1 - Car Model Prediction)

Dmitri Kazanksi

Table of Contents:

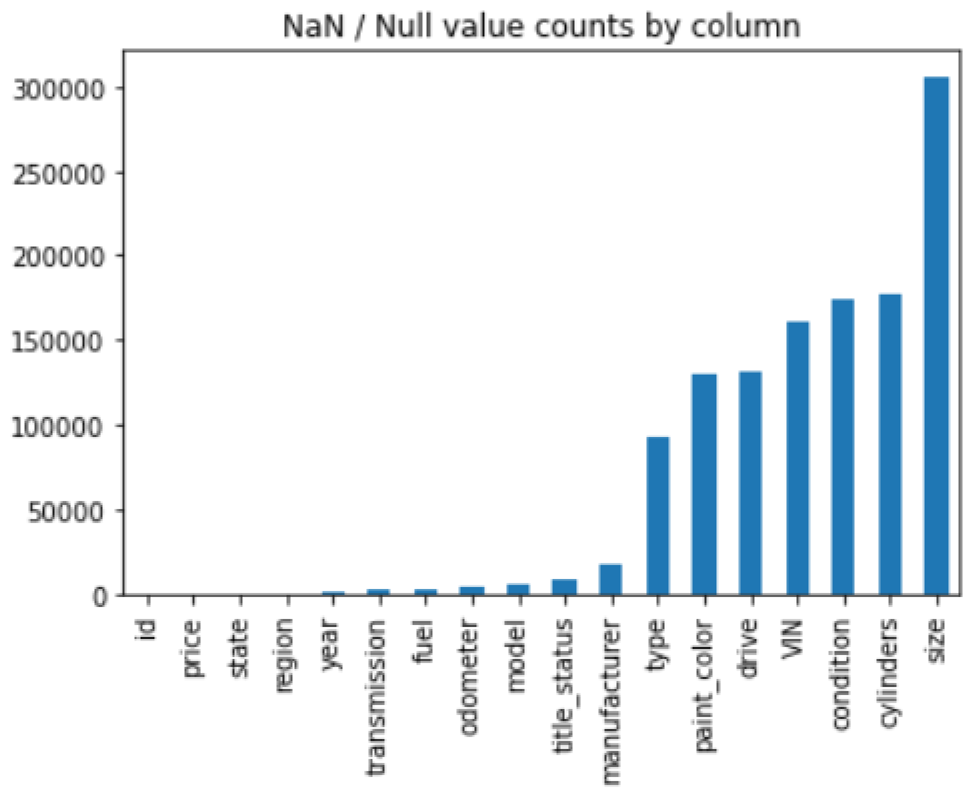
Data Understanding - Summary

To gain an understanding of the data, I followed the following steps:

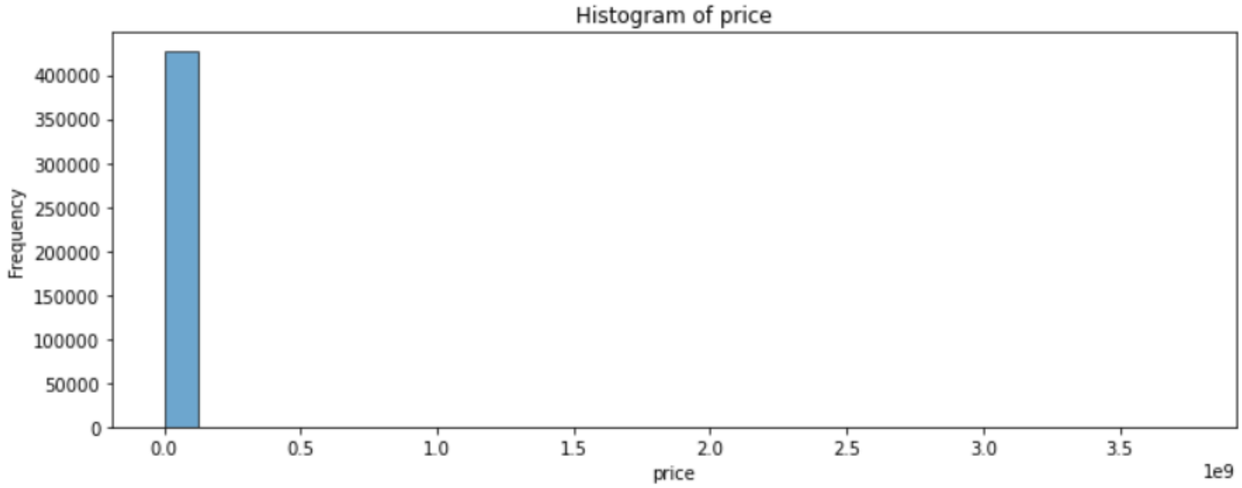
1. Converted the CSV into a Pandas DataFrame
2. Reviewed the DataFrame Sample or Head to get an idea about the possible values of each field.
3. Review Shape and Info to understand the counts and the shape of the data
4. Using the described method, look at the summary statistics for numerical values
5. Counted the null values per field and per record to figure out how to treat them
6. Reviewed duplicate records
7. Reviewed bogus records, such as records that were clearly incorrect.
8. Review possible values for each field to see if they can be combined or cleaned up.
9. Review the null (NaN) values and determine what to do with them.
10. Analyze the distribution of each feature
11. Analyze the impact of each feature on price
12. Produced a Correlation Matrix to check for Multicollinearity and to understand the impact of the numerical variables on Price.

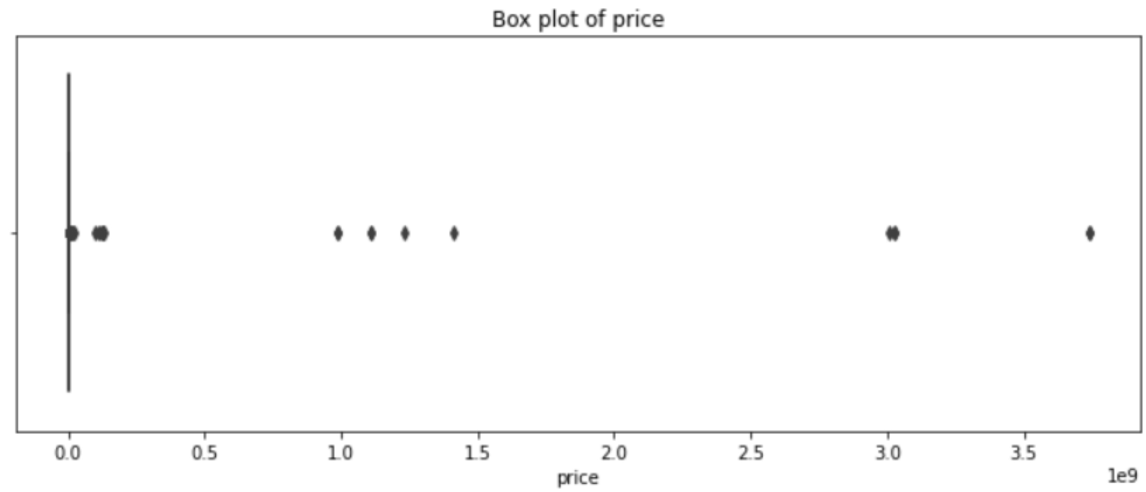
The data appeared to be dirty and needed extensive cleanup and transformation. Here are some of the issues with the data:

- About 35% of records had duplicate VIN numbers that are supposed to be unique to a car. This indicates that many cars were listed multiple times.
- There were a lot of NaN values. The graph below shows NaN counts by feature:



- The data set contained a lot of bogus data, such as:
    - Cars with the prices in billions of dollars. These were regular cars, such as Fords and Hondas–and not any rare “hypercars.” For example, this was the initial box plot and histogram of car prices:





- Cars with prices of $0 or $1, which also were good, fairly new cars that should have had prices of thousands of USD.

Data Transformation - Step 1: Clean-up

Before transforming the data for modeling, I had to do a basic cleanup that included the following:

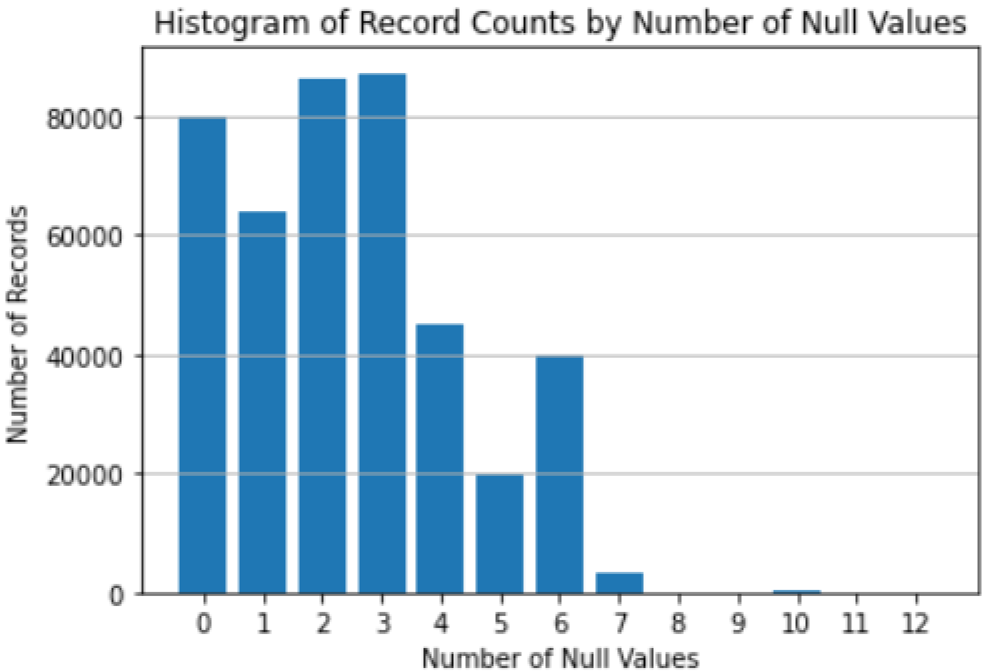
1. Deleting records with duplicate VIN numbers (except when the VIN numbers were set to 0 or were missing). That eliminated 35% of records.
2. Treating the null (NaN) values. Typically, there are two approaches to treating the records with NaNs:
    - Delete records that contain NaNs
    - Impute the NaNs with Median (for numerical) and Mode (for categorical)

The disadvantage of the first method is that we might lose a lot, if not all, of records. The disadvantage of the 2nd method is that we introduce bogus information. For example, if the manufacturer is unknown, we might end up replacing the data with "Ford" (if it is the most common manufacturer in the US--even though Ford has a market share that is far from the majority (only 16%).

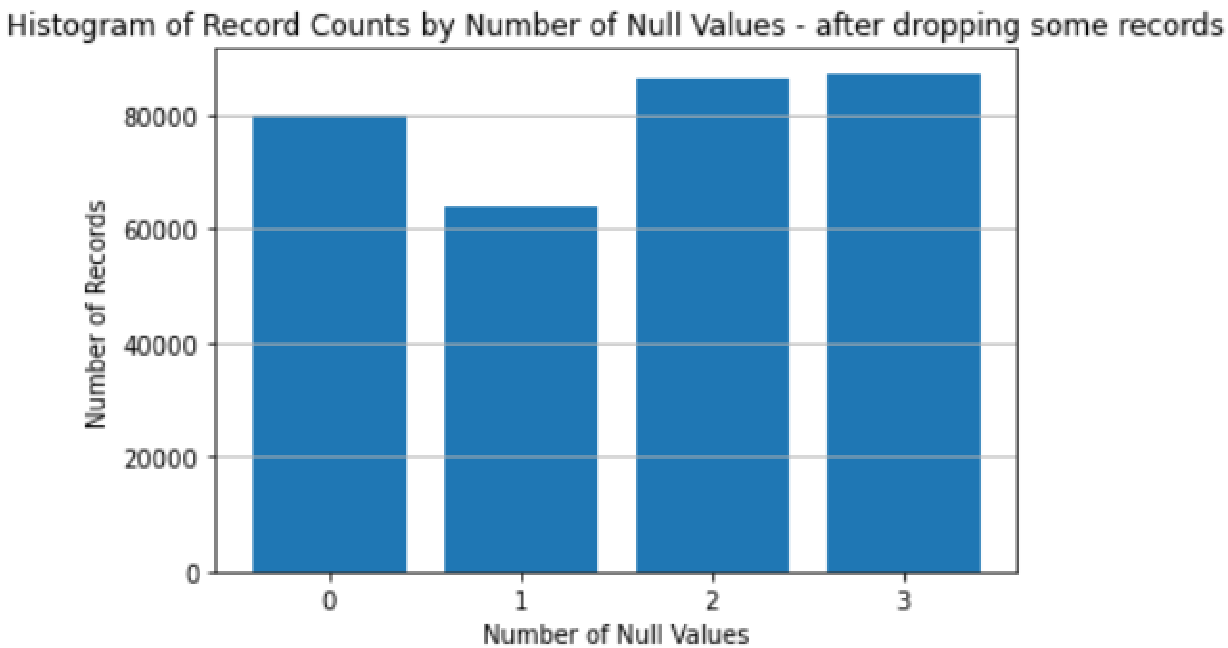
I ended up using a combination approach, where:

A. I counted the number of NaNs per record and deleted records that contained more than 3 NaNs.
B. For the records that contained three or fewer NaNs, I ended up imputing the NaNs with Median and Mode for Numerical and Categorical features, respectively.

This was the initial record counts by the number of NaN:



This was the histogram after deleting the records with too many NaNs:



Data Transformation - Step 2: Preparation for Modeling

Having treated the NaNs, I proceed with the following transformations:

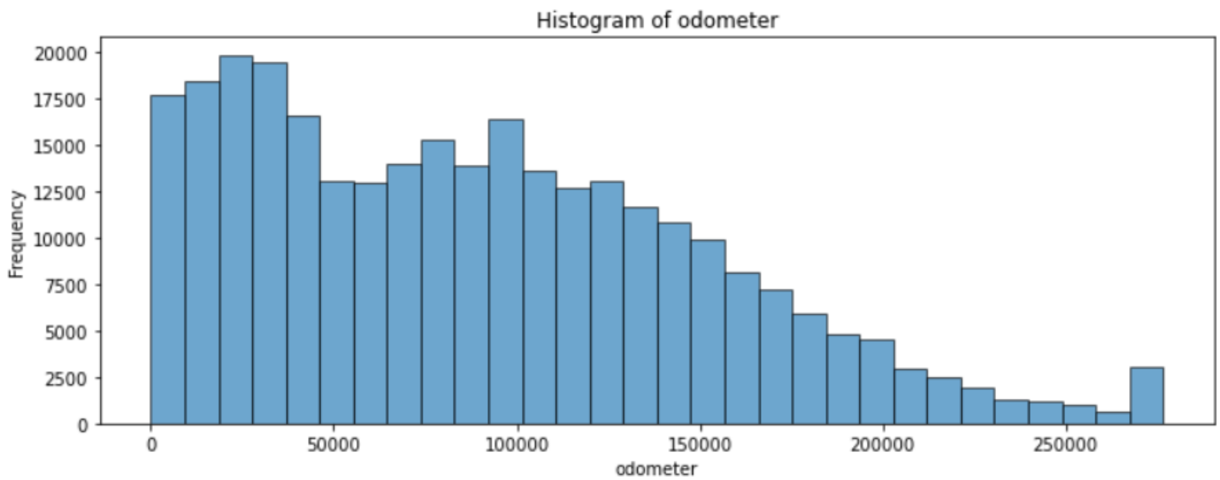
1. Transformed price into LogPrice–since price distribution was highly skewed
2. Treated the outliers of Price and other numerical variables by setting them to 1.5* Upper bound or -1.5* Lower bound
3. Transformed “Object” data type into “Category.”
4. Transformed “Year” and “Odometer” values from Float to Integer
5. Consolidated some of the values of categorical variables, which were too close. For example, “lien” and “clean” titles were essentially identical.
6. Converted some of the categorical values to numerical, whenever appropriate. For example, Cylinders were numerical variables expressed in words “4-cylinder, 6-cylinder, etc.)
7. Converted ordinal variables to their numerical equivalent. These included:
    a. Condition (from “salvage” to “new”)
    b. Title Status (from missing or parts only all the way to “clean”)
8. Renamed variables to use shorter names
9. Applied on-hot encoding for categorical variables that could not be transformed into numerical. Those included:
    a. Transmission
    b. Manufacturer
    c. Fuel Type

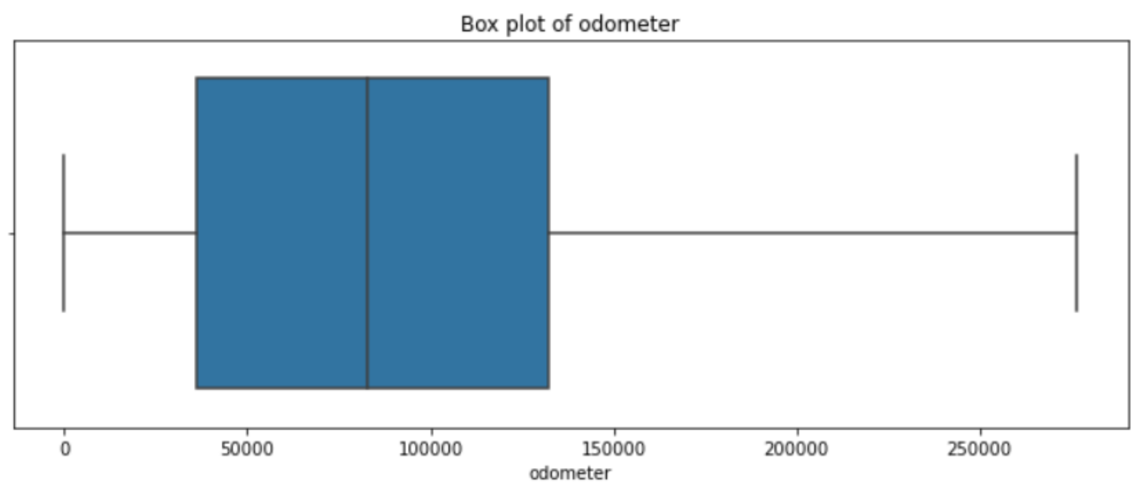
10. For one-hot encoded variables, delete one of the variables in each group so it can be used as a “default.” For example, I deleted the “Acura” manufacturer so that when every other manufacturer value is 0, the manufacturer is “Acura.” That helped me reduce the number of variables and prevent redundancy.

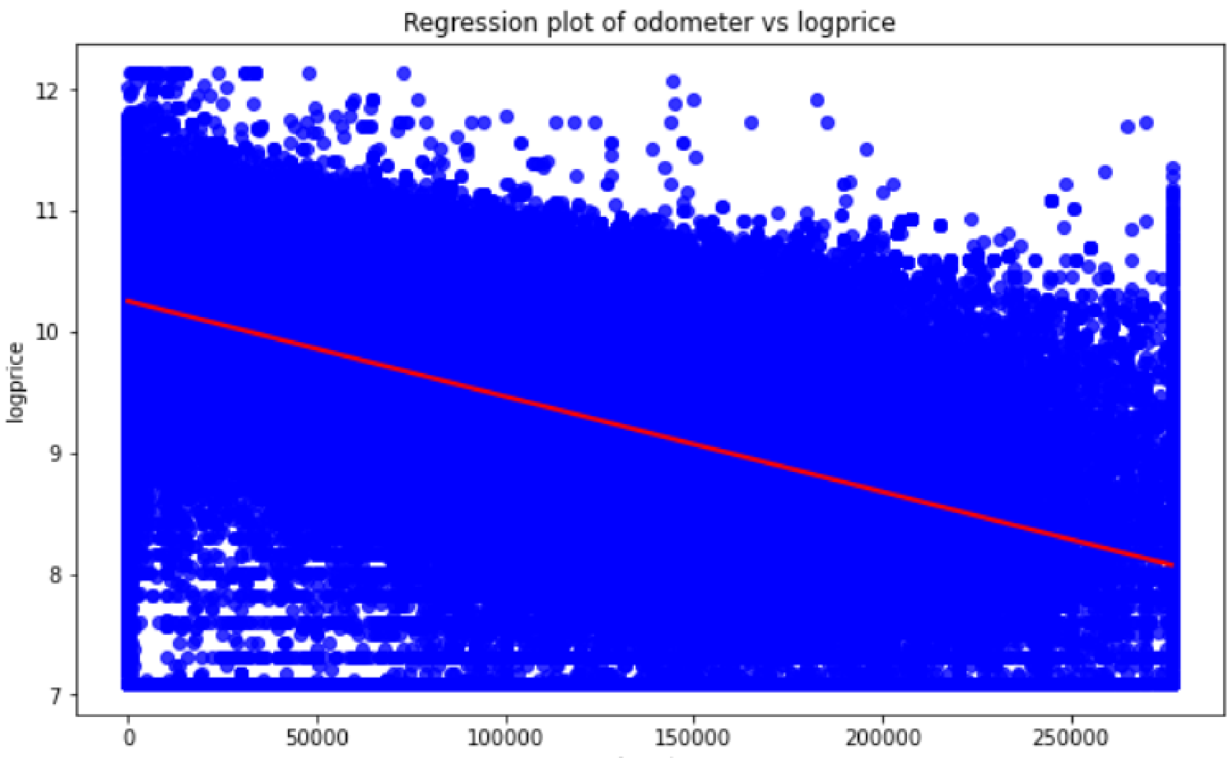
Exploratory Data Analysis - Summary

To understand what variables are important and how they impact the lotgprice, I produced the following visualizations.

For all numerical (including the transformed to numerical) variables, I produced a Histogram, Box Plot, and the Reg plot. For example, for Odometer, I produced:

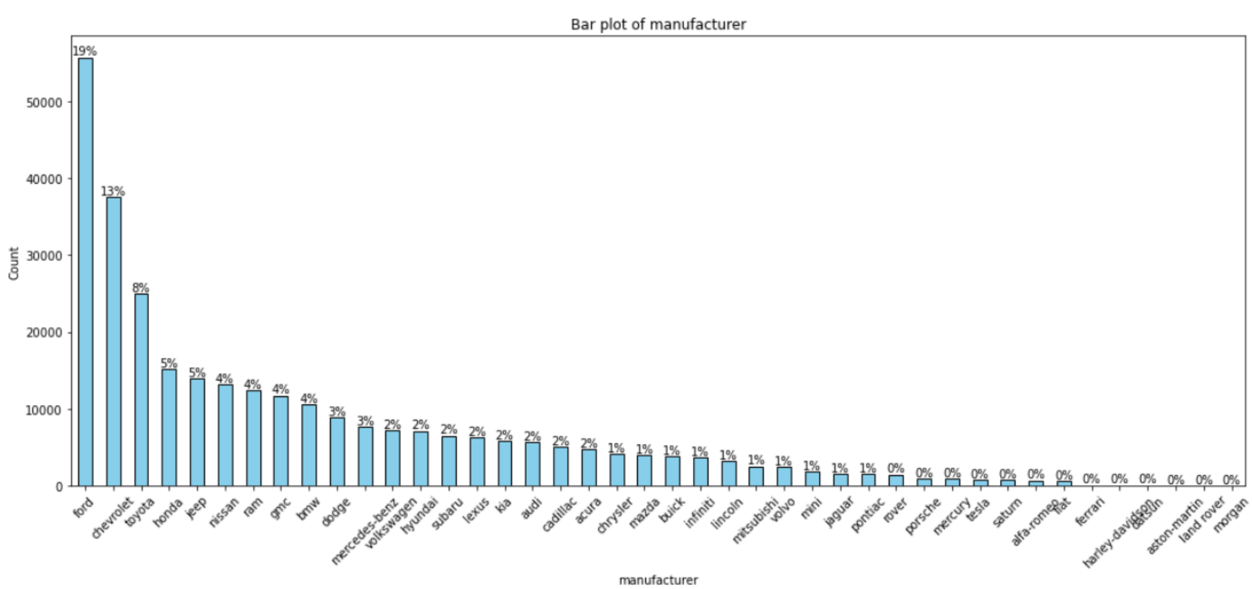


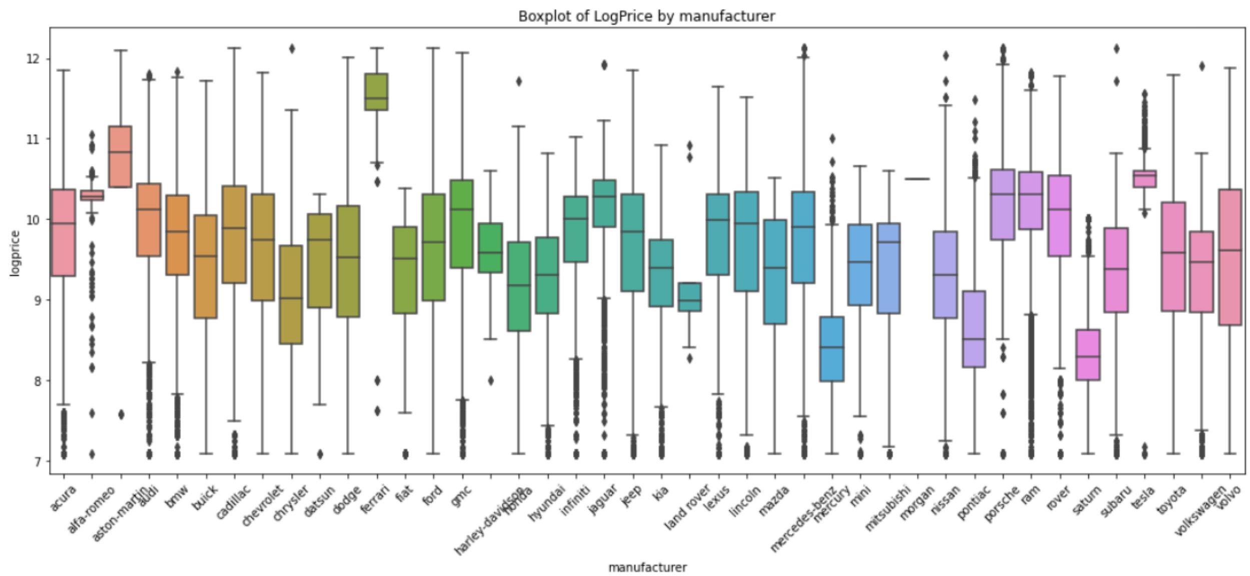




This gave me an understanding of how each numerical variable is distributed and how it impacts LogPrice. For example, higher odometer readings reduce the price, which makes sense.

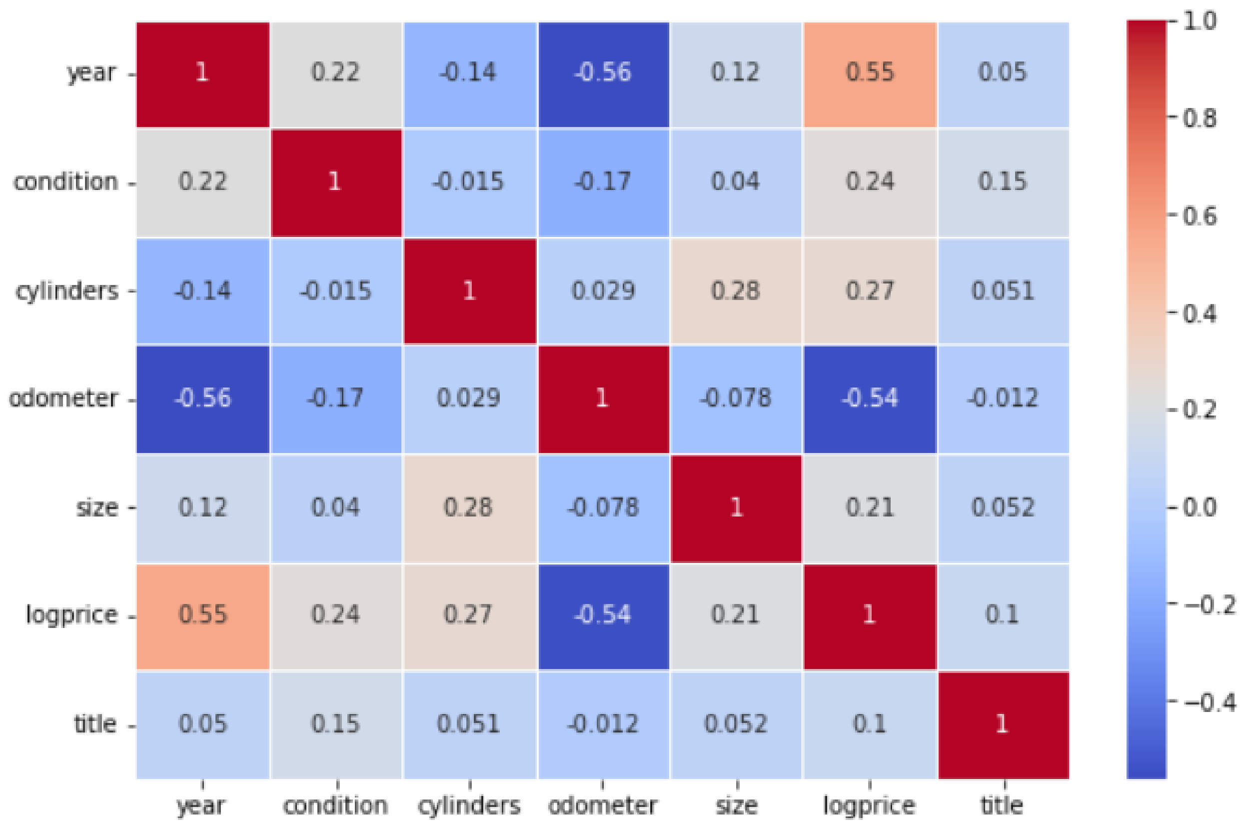
For each Categorical variable, I produced BarPlot and Box Plot. For example, for Manufacturer, I produced:





This gave me an understanding of the value counts for each value of the categorical variables and their impact on LogPrice.

Finally, I produced a Correlation Table for all Numerical (including the transformed to numerical) variables:



- Based on the analysis, the most important numerical predictors of the price were:
    - Year (the newer, the higher the price)
    - Odometer (the more miles, the lower the logprice.
- Cylinders, Condition, and Size appear to have a smaller impact on the price
- Categorical variables that impacted the price were:
    - Manufacturer
    - Title Status
    - Transmission
    - Type
    - Size
    - Drive
- State and Color do not appear to have a strong influence on the price. So, I ended up excluding them from the modeling exercises below.

Modeling-Summary

To establish a baseline, I defined an "Average" model (logprice = average log price for all cars). The model ad the following error statistics:

- Baseline (Average) RMSE: 0.904892361538039
- Baseline (Average) RMA: 0.8548331118511051

After that, I created three models, as shown below:

Model1: Ridge Regression with Standard Scaler and Grid Search

The best Alpha parameter came to 42.919

The model produced the following performance:

- Ridge Train RMSE: 0.567
- Ridge Test RMSE: 0.568

Model 2: OLS Regression with Sequential Feature Selector

This model was very slow to calculate, as I would expect. It produced the following performance:

- Sequential Selector Train RMSE: 0.624
- Sequential Selector Test RMSE: 0.624

Model 3: Lasso with Polynomial Features and Standard Scaler

I had to limit the polynomial degree to 2 because to calculate 3, I would need 92GB of RAM, and I only had 64.

This model did not do very well

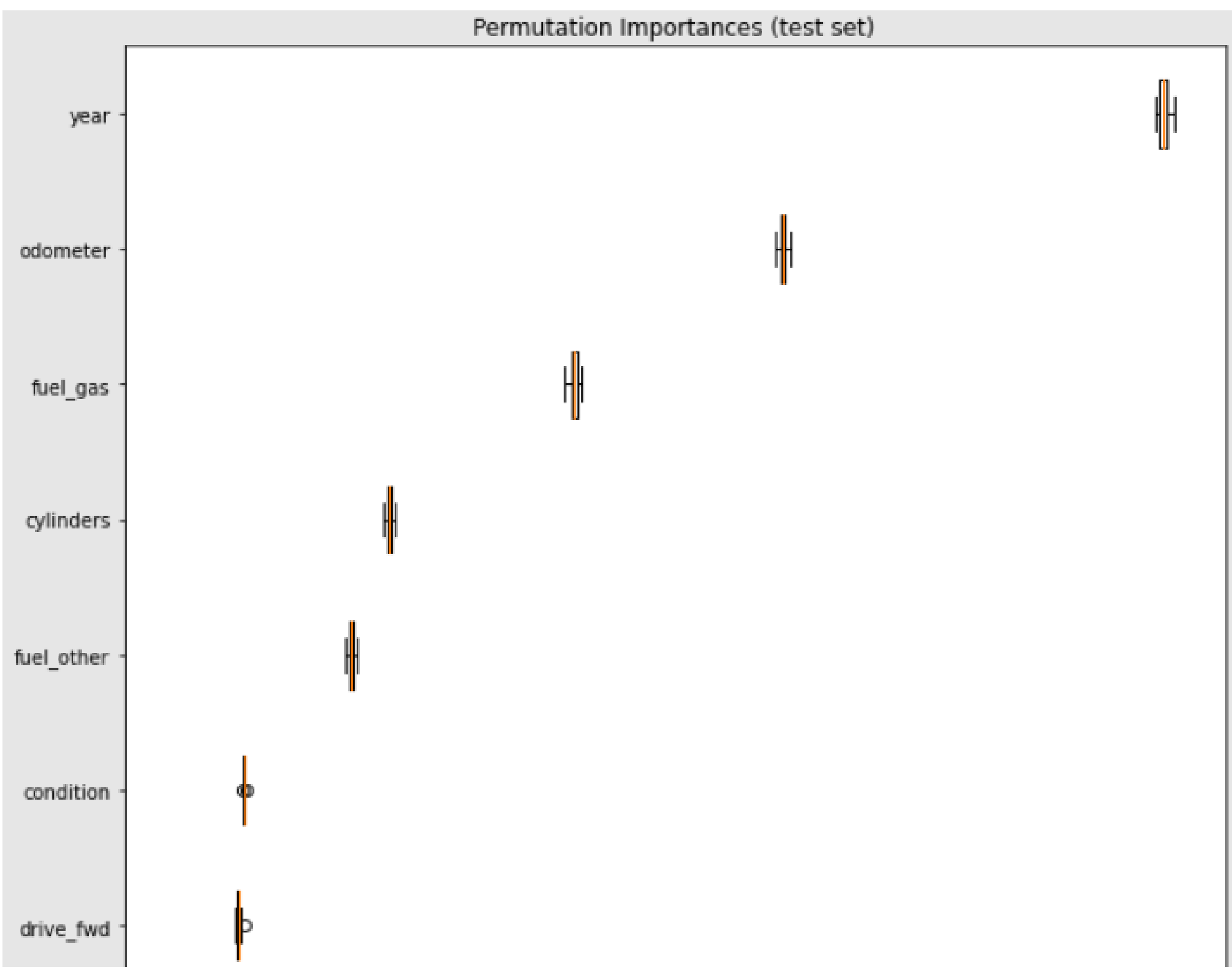
- Lasso Train RMSE: 0.907
- Lasso Test RMSE: 0.907

For each of the three models, Test RMSE was similar to Train RMSE, which indicated that we successfully prevented overfitting.

Based on the Test RMSE result, I recommended using the Ridge model.

Feature Importance

I calculated the Feature Importance for each model and created a Permutation Importance Plot. For the best model (Ridge), the Plot looked like this (showing top features):



All three models, as well as the EDA, pointed to the importance of the following features:

- Year (the newer, the higher the price)
- Condition (the better, the higher the price)
- Cylinders (the more, the higher the price)
- Odometer (the lower the mileage, the higher the price)
- Type of fuel ("other," such as electric, was best)
- Make (manufacturer): Some are more expensive, and others are cheap.

Model Interpretation and Deployment

One of the good things about Regression models, such as Ridge, is that they are easy to interpret and deploy. For the best model (Ridge), I produced the following list of coefficients for each of the features used in the model.

0 year: 0.35610637052782124
1 condition: 0.0912967714130221
2 cylinders: 0.16541548610613835
3 odometer: -0.2790798011474413
4 size: 0.015642057688281526
5 title: 0.036332807009326104
6 make_alfa-romeo: 0.005646105599343679
7 make_aston-martin: 0.0013736777203190004
8 make_audi: 0.01659017461510657
9 make_bmw: -0.004601693351688086
10 make_buick: -0.016008386695129908
11 make_cadillac: -0.005491009478797403
12 make_chevrolet: -0.02766695269708579
13 make_chrysler: -0.036302269323723234
14 make_datsun: 0.008588211677951237
15 make_dodge: -0.0367277313073569
16 make_ferrari: 0.021126315287522183
17 make_fiat: -0.02186316219744158
18 make_ford: -0.03248561025612152
19 make_gmc: -0.0077563957254419705
20 make_harley-davidson: -0.0005576007446695917
21 make_honda: -0.006544167883423729
22 make_hyundai: -0.03569456449196106
23 make_infiniti: -0.000869641604046785
24 make_jaguar: -0.003815578496814698
25 make_jeep: -0.009698266964836689
26 make_kia: -0.036648988215852996
27 make_land rover: 0.0001815190900496453
28 make_lexus: 0.03238017816257291
29 make_lincoln: 0.0012670955806414882

30 make_mazda: -0.022982787111301816
31 make_mercedes-benz: 0.006492125408135606
32 make_mercury: -0.02598096021286774
33 make_mini: -0.00707288147090442
34 make_mitsubishi: -0.033139800001109
35 make_morgan: 0.002996337988204665
36 make_nissan: -0.045088127302030136
37 make_pontiac: -0.011479822966360484
38 make_porsche: 0.026696471118585852
39 make_ram: -0.01748370240635555
40 make_rover: 0.009893361851987826
41 make_saturn: -0.023302569245781942
42 make_subaru: -0.013519459112599223
43 make_tesla: 0.023356012268510376
44 make_toyota: 0.02503193365889552
45 make_volkswagen: -0.03564909753565984
46 make_volvo: -0.0011267852299690602
47 fuel_electric: -0.04542964095801256
48 fuel_gas: -0.2259952439117115
49 fuel_hybrid: -0.06910967622470997
50 fuel_other: -0.1516596277068553
51 tr_manual: 0.04682881447609341
52 tr_other: 0.03270026881464663
53 drive_fwd: -0.09031948362234074
54 drive_rwd: 0.0241878599509725
55 type_bus: -0.013221051321766935
56 type_convertible: 0.03201291471377096
57 type_coupe: 0.01839553230984127
58 type_hatchback: -0.0365712537227965
59 type_mini-van: -0.01386177505851913
60 type_offroad: 0.015477643806543344
61 type_other: 0.030898098542237906
62 type_pickup: 0.08028954870630044
63 type_sedan: -0.057000932558437224
64 type_truck: 0.05897950976961168
65 type_van: 0.012812703284642528
66 type_wagon: -0.004829400141209407

The model is very easy to interpret and easy to use.

- Positive coefficients increase the log price when the feature is present.
- Negative coefficients decrease the log price when the feature is present.

When deploying the model, one would need to follow these steps:

1. Get the value of each feature

2. Multiply the value by the corresponding coefficient, as listed above
3. Sum up the results. This will give you the estimated log price of the car.
4. Take an exponent of that sum to arrive at the estimated price in USD.
5.

Please note that Many of these features will be equal to 0 most of the time. For example, Only one of the "Make" features will equal 1 and the rest to 0.

Finally, please note that the following values were deliberately excluded from the features for modeling purposes:

- make_acura
- fuel_diesel
- tr_automatic
- drive_4wd
- type_SUV

One can think of them as “default” values. When the car falls under each of these values, all other values from the same group become equal to 0. For example, when the car make is "Acura," you need to set the values of all other makes to 0.

The calculations should be easy to automate, for example, via a JavaScript file on a web page or via a mobile app.

Overall, we have a useful model that is easy to interpret, makes sense, and is easy to deploy.

Thanks!