Problem statement:

This is a mock problem that I used to guide my data science lifecycle for this project. I want to give those looking to sell their used cars the opportunity to get an idea of how much their car is worth based on a lot of data from a car sales website. The user will interact with a web app hosted through AWS to input his or her car's features and will then receive an estimate for how much the model thinks the car is worth.

Description of the dataset, how you obtained, cleaned, and wrangled it

I'm using Carfax data that I scraped online. The data is divided into 3 areas: San Francisco, New York, Dallas, with a 1000 mile radius outside of each city. This way, most of America by land area is covered with my search. The data contains the most popular car manufacturers such as Toyota, Honda, Audi, Ford, as well as others.The data I gathered consists of ads available on the website in the Winter of 2020.

Duplicates:

In total, the data consists of over 50,000 data points.There are a lot of duplicate ads, which is expected as my search areas overlap. Not so many different ads for the same car, though. After keeping only the first of duplicate ads and the first of duplicate cars, I ended up with just under 36,000 unique car postings.

Nulls:

The subModel feature had 35,893 missing values, and the badge feature had 5,082 missing values. Because there was no information that I was likely to gather from mostly missing columns, I deleted the subModel feature. For reference subModel of a car is a way of delimiting the car beyond just make and model; thus - subModel. Badge only had around 5000 missing values, so I will likely impute them later. It's also possible that this variable will have multicollinearity with other variables, so I might delete it later.

Where are the cars?

https://public.tableau.com/views/Capstone2EDA/DualMapDash?:embed=yes&:device=desktop&:display_count=y&publish=yes&:origin=viz_share_link&:showVizHome=no

The map visualization above shows circles, corresponding to the locations of cars sold as well as amount of cars sold (by the size of the circle). The states are also colored to be a darker shade of blue for states where relatively more cars were sold.

A few interesting patterns emerge from this viz. The first is that a lot of cars are sold in California and Texas. In these states there are big hubs of activity centered around the large cities such as San Francisco, Los Angeles, and Dallas. Interestingly, outside of those hubs, there is not too much

activity. On the East Coast on the other hand, it appears as though the hubs aren't as large, and the activity is spread out more evenly across smaller communities. This is shown by the fact that on the East coast, broadly speaking, the dots are smaller but more spread out, whereas on the West Coast, the dots are closer together and larger in size.

In this report, there aren't ads that come from North Dakota, South Dakota, or Nebraska on the continental US; and Hawaii and Alaska are not represented either.

Given this broad overview of the data at hand, I will dive into the deeper analysis, shown below.

https://public.tableau.com/views/Capstone2EDA/DimensionToggleSheet?:embed=yes&:device=desktop&:display_count=y&publish=yes&:origin=viz_share_link&:showVizHome=no

The main takeaway I have from the above visualization is that the data are not qually distributed among the different dimensions, meaning for example there are more Kias than Nissans. This means that down the line, I will have to pay closer attention to models that perform well when data is equally distributed.

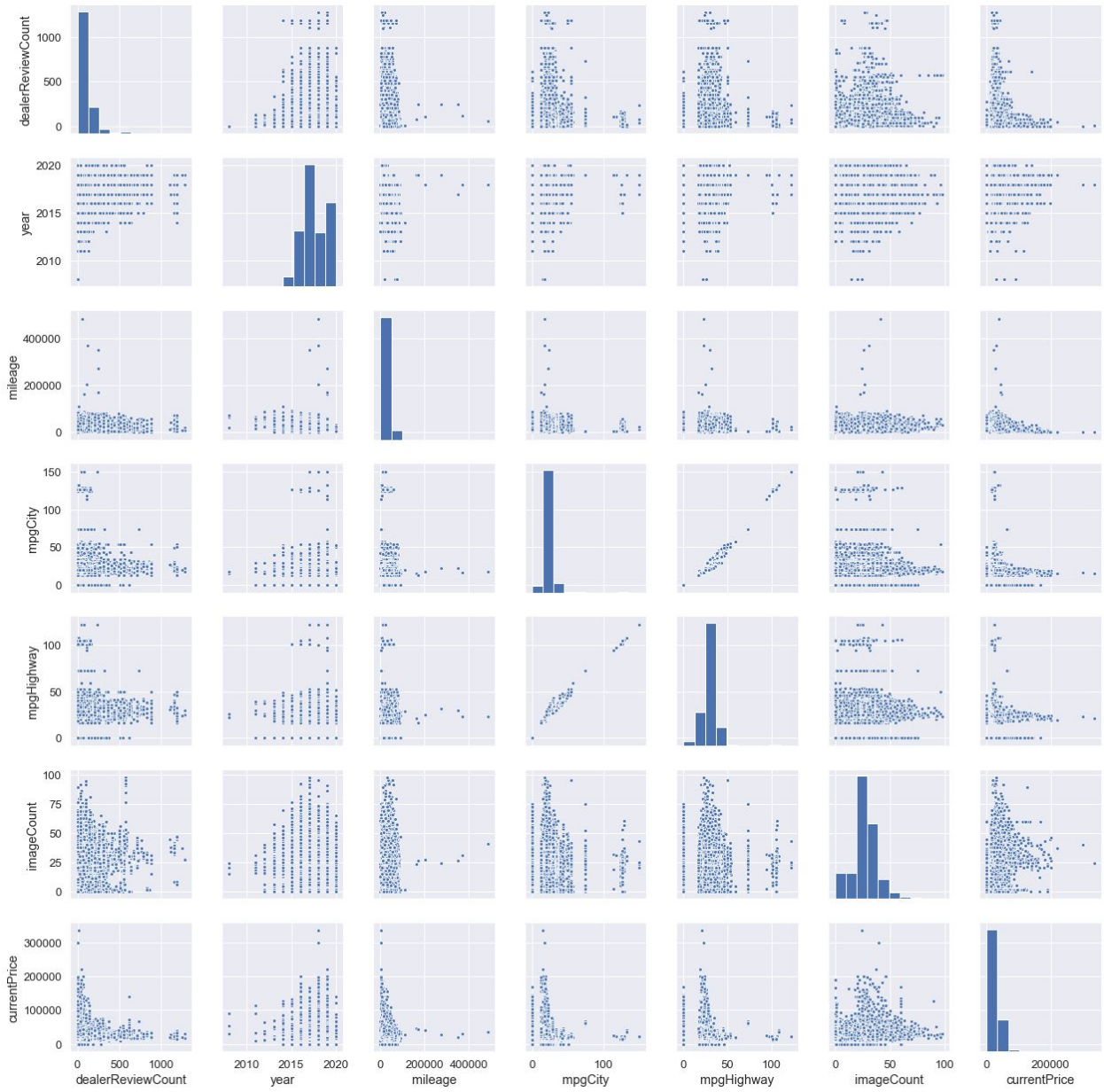
Prices for each region

https://public.tableau.com/views/Capstone2EDA/PriceMapDash?:embed=yes&:device=desktop&:display_count=y&publish=yes&:origin=viz_share_link&:showVizHome=no

This visualization makes it clear that the highest median price for a car is in the North-West (as deliminated on the map above) and the lowest in the South-West. The range of values is not too great, however. The minimum is 23,900 USD and the maximum is 25,881 USD. It's important to note, however, the differences in the number of car listings in each region. This may play a role in skewing the data slightly. But, as mentioned before, it's clear that the differences between the median prices of cars in each region are not great, so I doubt this will play a role in modeling.

I will be looking at currentPrice as the label. There are two: listPrice and currentPrice. The website allows its users to change the list price after posting the ad. This way, if there is a price change, the new one will be more reflective of the intended value of the car. Plus there aren't any differences.

Numerical Variables



This plot is very large and overwhelming, but I'm only really looking for multicollinearity between features that may mess up a lot of future modeling. There looks to be a high correlation between mpdCity and mpgHighway, something that I will have to account for in the future. It's possible that keeping just one will yield better results.

A few other things stand out too. For example There is a cluster of cars on the high end of mgpCity and mpgHighway. These are likely hybrid cars that will have to be accounted for, as they are outliers. Analogously, there are cars that have 0 mgpCity and 0 mpgHighway readings. At first glance, these are likely electric cars, but I will doublecheck.

Mileage, currentPrice, and dealerReviewCount also have pretty heavy outliers that may need to be accounted for.

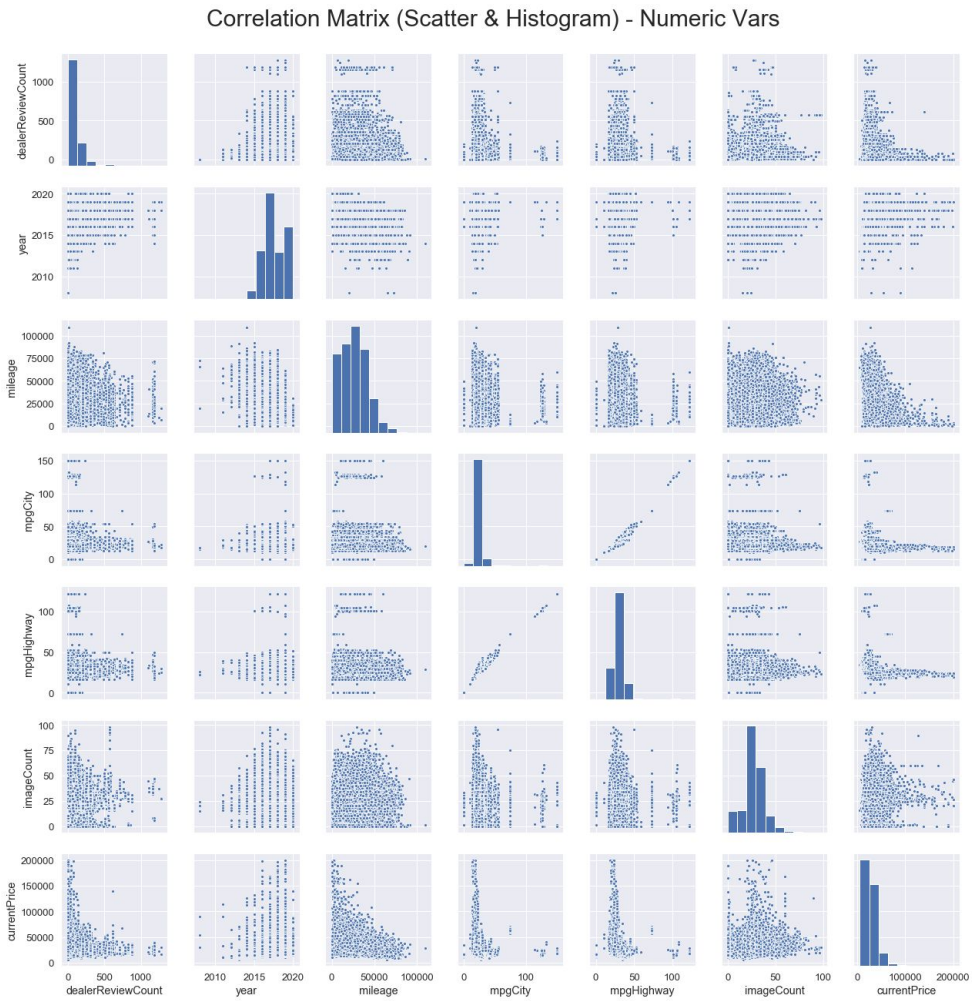
There are 813 cars that have mpgCity and mpgHighway listed as 0. Some of them are not electric which implies that the zeros were put in by mistake. To remedy this, I will check the data for any other cars of the same make, bodytype, model, and year to see if there are any of the same cars with mpgCity input. If there are, I will use that value. If not, I will look in other years, then in other models. Ultimately if there aren't any analogous cars, I will impute the values myself. Luckily I only had to input 10 or so values myself.

CurrentPrice == 0

There are also 272 cars in the data where their currentPrice is zero. This is problematic because currentPrice is the target variable. Normally I would impute this, but given that it's a target variable, using ML to impute the value and the using the imputed value as a target might introduce a lot of variability. To avoid this, I will remove them. I will later use them to test the frontend of my webapp that predicts these prices.

Obvious outliers

There are ten cars that have a mileage of over 150,000 miles and a price of over $201,000. These cars are by nature influential and may harm my models because the bulk of the data is well inside of this range. For this reason, I will remove them. I feel comfortable doing this because there are only 10 data points here and they are not useful in predicting cars at a much lower price range, such as $20,000.



Now with the big influential points removed, it's easier to look at the trends between the numeric variables. To start, there are some visible clusters of cars that may be easy to explain. For example mpgCity and mpgHighway have high end clusters, likely as a result of hybrid cars. There are also some cars from the year 2008, which might be influential. there also appears to be a cluster of dealerships with a very high amount of reviews, likely as a result of them being very good or very bad.

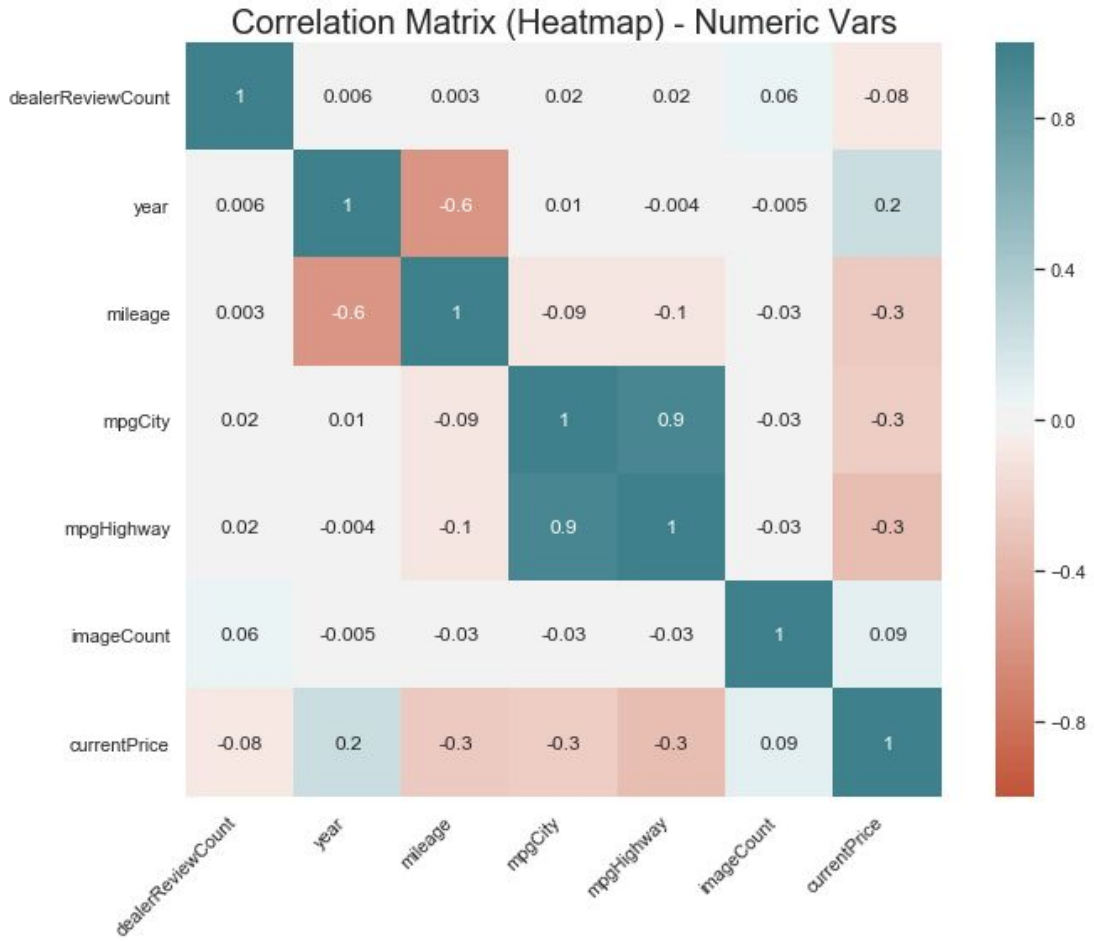
Suspicious dealerships

Wow, all the dealers with over 1000 reviews seem to only have a 5 star rating. They must be very good, have programs that incentivize giving them a 5 star review, like a rebate or discount, or are faking their reviews.

After a review of a sample of these dealerships, the reviews seemed genuine and the dealerships seemed to simply provide good service to their customers. It's also possible that they asked for reviews and had rebates, but that is unconfirmed.

Some other patterns stand out too. For example as mileage goes up, currentPrice seems to go down exponentially, somewhat expectedly. Note that there is a wide variation of prices at low mileages, but at high mileages, prices are exclusively low.

The other variables don't show very high correlations between each other and the target variable. This will be further investigated below with the same correlation plot, but one that better shows the correlations.



The graph above confirms some of the patterns that emerged from the correlation matrix. mpgCity and mpgHighway have a correlation coefficient of 0.9 which can be devastating for some modeling techniques such as linear regression. Year and mileage are negatively correlated at -0.6, as expected. All other correlations are fairly small and are unlikely to cause trouble in modeling. There is also a relatively high positive correlation between year and currentPrice of 0.2, which makes sense, as newer cars would cost more.

For more info on Price, see the dashboard below.

https://public.tableau.com/views/Capstone2EDA/DimensionToggleSheet3_1?:embed=yes&:device=desktop&:display_count=y&publish=yes&:origin=viz_share_link&:showVizHome=no

Machine Learning (ML) Analysis and Results

I tried the following ML models: Linear Regression - Lasso, Ridge, and Elastic Net, Random Forest (RF), and tuned XGBoost models. I also used GridSearchCV with 3 folds to tune the models. Each step provided me with a better fitting model, the procedure and results of which I discuss below. Additionally, I compare the models to each other at the end of this section using Root Mean Squared Error as the metric.

Initial Considerations

Because the data is a mix of categorical and numeric variables, which will need to be turned into dummy variables, using one-hot encoding. Additionally, to improve the performance of the linear-based models, I will scale the data to be centered at 0 with a standard deviation of 1.

Another consideration is small cell size. I tackled this with two methods: 1) by getting rid of columns that won't be useful for users when predicting the price of their own car (e.g. subTrim, subModel, etc.), and 2) by getting rid of any groups that have less than 3 members.

Initial Train/Test Split

I created a train/test split for the data with a test size of 30%, meaning I used 70% of my data to train and tune my models with Cross-Validation (CV) splits and held 30% of the data untouched to then give a final test of the model.

Models:

Lasso Linear Regression

I did a comprehensive GridSearch with 5 CV folds of the parameter sequence and found the optimal alpha value of 0.1.

Lasso Linear Regression Results

Tuned Lasso RMSE on holdout set: 6221.3054

Ridge Linear Regression

I did a comprehensive GridSearch with 5 CV folds of the parameter sequence and found the optimal alpha value of 0.2941.

Ridge Linear Regression Results

Tuned Lasso RMSE on holdout set: 6301.3197

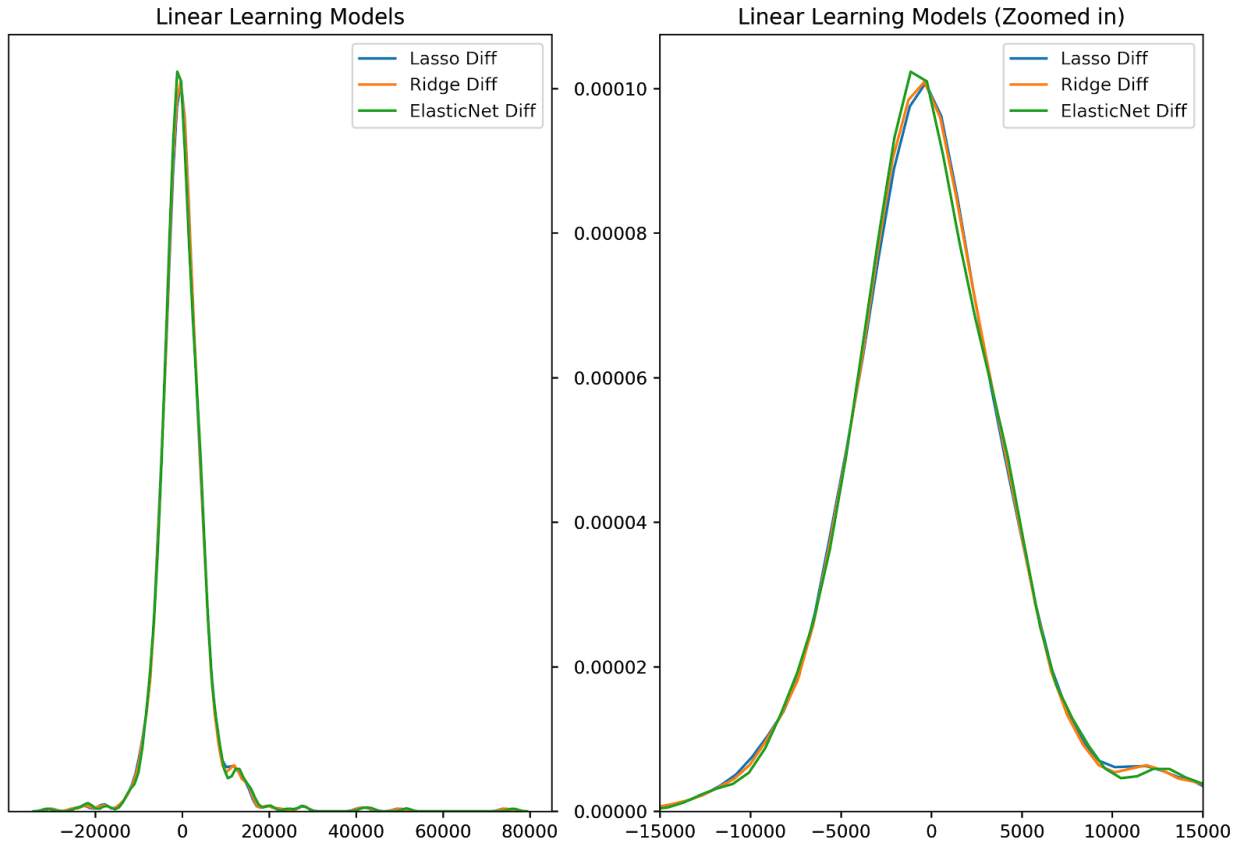
Elastic Net

I did a comprehensive GridSearch with 5 CV folds of the parameter sequence and found the optimal alpha value of 0.2941.

Elastic Net Results

Tuned Lasso RMSE on holdout set: 6228.0565

Linear Model Comparison:



To visually compare the predictive ability of the models, I've graphed the KDE plots of the difference between the observed values and predicted values of the holdout set for each type of linear model. What is obvious to me is that no one of the models is much better than the others, so it might be necessary to do more modeling to come up with a better prediction.

Random Forest Classifier

The next model I tried was the Random Forest (RF) Classifier. Now that I have introduced the general concepts of parameter tuning through GridSearch with 5 CV folds, I won't spend much time talking about them here. I built a parameter grid of the following parameters:

```
params_rf = {'bootstrap': [True, False],
             'max_depth': [10, 20, 50, 100],
             'max_features': ['auto', 'sqrt'],
             'min_samples_leaf': [1, 2, 4],
             'min_samples_split': [2, 5, 10]
```

}

For each parameter, I picked a variety of values, all of which are available in the Jupyter notebook for this project. Because of the large amount of possible parameters, I used SearchCV to save some time but still come up with a usable result.

RF Classifier Results

After all the tuning, the RMSE on the holdout set was 6276.4488

XGBoost

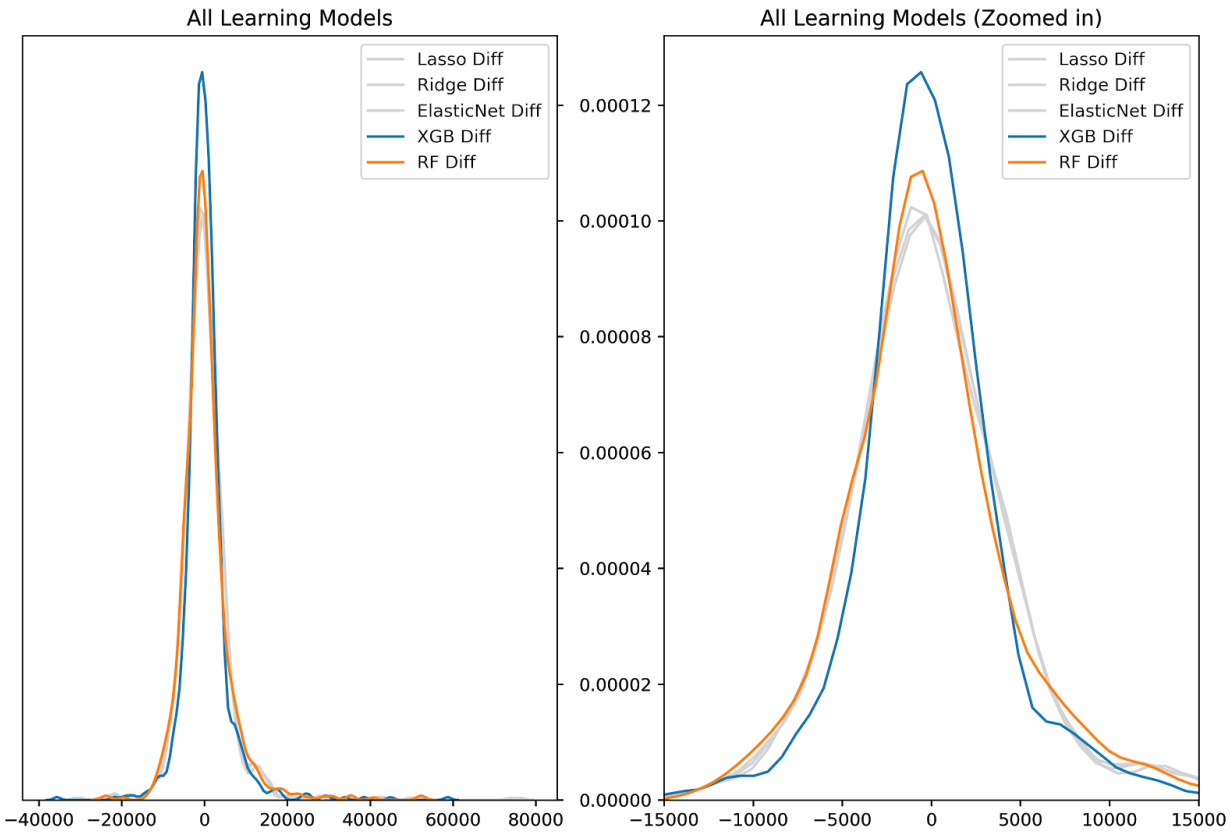
The final model I tried was the XGBoost model, a model with many parameters requiring the most tuning out of any model I tried. Because of the large amount of parameters, I tuned them one to three at a time, got the results and tuned the next batch of parameters. The final tuned model had the following parameters:

```
params_xgb =  {'colsample_bytree': 0.8,
               'gamma': 0,
               'learning_rate': 0.2,
               'max_depth': 8,
               'min_child_weight': 1,
               'reg_alpha': 97,
               'reg_lambda': 0.9,
               'subsample': 0.9
              }
```

XGBoost Results

The best XGBoost model, whose parameters are shown above produces an RMSE score of 5623.6894, the best so far.

Comparing All Models



The figure above shows the KDE curves of all the models I tried in this section. The gray lines show the linear based models. It's interesting to note that a RF model did not outperform the Elastic Net and Lasso Linear Regressions.This confirms that there is no ready-made machine learning solution that will fit any data. XGBoost still outperformed all other models, which is also shown on the KDE plot above. This will be the model that I will use for future predictions.

Project Conclusion

In this project I explored, analyzed, and visualized car sales ad data in order to build a regressor that can tell a user how much to sell their car for based on the relevant cars sold in the US. After looking into the data and training several models, I delivered my best tuned XGBoost model as my final classifier to be deployed in real time to ask for user's car information and provide a prediction of the price.