Predicting Rideshare Pricing in Boston

Rio Wombacher
Brown DSI
https://github.com/rio-wombacher/rideshare_price_predictor.git

1. Introduction

With the variability of pricing for rideshare apps due to surge pricing, we want to identify which features have the greatest impact on price and be able to predict the price of a ride given current circumstances. Data from 2017 shows that in Boston alone, almost 100,000 rides a day were completed via Uber and Lyft[1], a number that is likely to have grown years since. I was motivated to look into this data, because, like many Americans, I have used both Lyft and Uber before, and at times their pricing model can seem random and exhibit large fluctuations in short periods of time.

Our data is taken from the Kaggle dataset “Uber & Lyft Cab prices” which gathers data about Uber and Lyft prices in Boston from November 26th to December 18th 2018. Uber and Lyft do not publicize their data, so the data was collected via Uber and Lyft API queries and corresponding weather conditions. Essentially, each “ride” in our dataset is the equivalent of going on your phone and checking what the price would be at that time for your specific route.

We are trying to build a regression model because our target variable, price, is continuous in nature. The dataset contains features about the ride itself (distance, source, destination, etc.), along with information on the type of car chosen for the ride, the time of the ride, and some brief weather information for the day. In total, we began with 693,071 “rides” and 15 features.

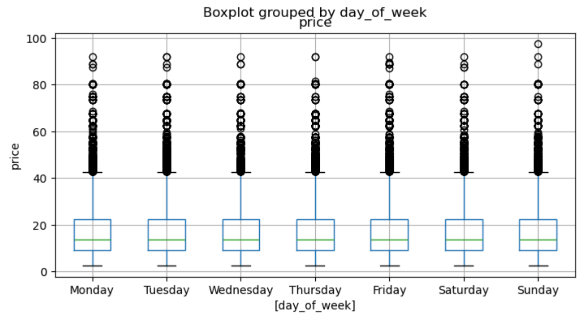
2. Exploratory Data Analysis

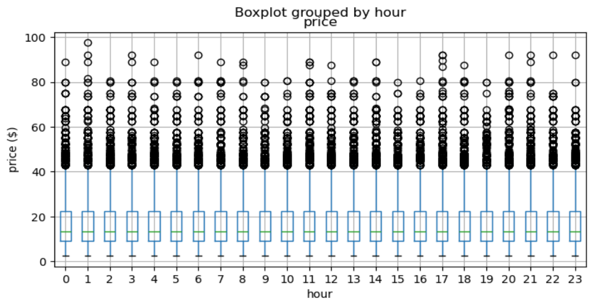
To start the exploratory data analysis (EDA) process, I first wanted to clean the data as much as possible. I removed the columns “timezone”, “id”, “timestamp”, and “product_id”. The time zone column was removed since all the data is from Boston, meaning all rides will obviously be in the same time zone. The time stamp and product id features were both variables that had more descriptive counterparts, so they were not needed. For example, we already had the “datetime” column which told us the month, day, day of the week, and hour of the ride, so there was no need to keep the timestamp column which was less descriptive. Similarly, product id was redundant and served the same purpose as the “name” feature (later renamed to “car_type”) such that they both described the type of car ordered for the ride. Finally, the id column was just a unique identifier for each ride, which offered no predictive power so it could be removed.

The next step was identifying null values and duplicates and deciding what to do with them. Thankfully, the only observations here with null values were related to taxi cab rides through Uber, which were not relevant here, so those rows were removed. Duplicate rows were also removed, since they were corresponding to the same exact ride at the same time, so we were able to keep just one instance of these duplicates. After these updates, the dataset contained 637,036 rows and 11 features.

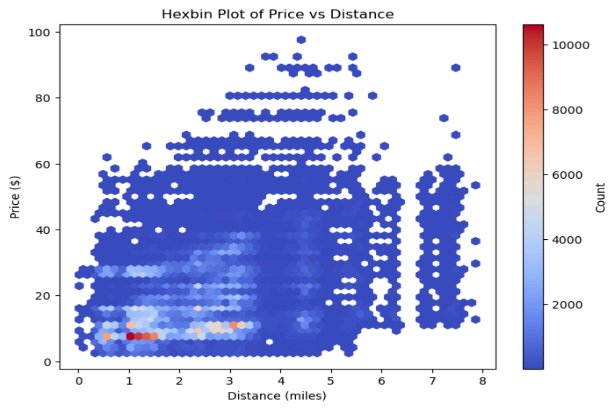
In the final step of our cleaning process, I renamed the feature "name" to "car_type" which more accurately describes what it represents. I also extracted the day of the week from the datetime feature so we would be able to see if that variable plays a role in predicting the price, and dropped the datetime column afterwards, since all its information was now captured in separate individual features.

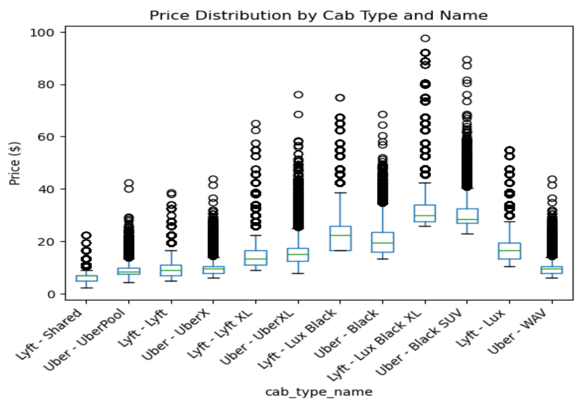
Moving on to the EDA, I first checked the distributions of all my variables with respect to my target variable, price. One thing that surprised me was the lack of correlation between the day of the week or the hour of the ride and the price. One possible explanation for this is that prices are driven by the supply of drivers and the demand from passengers. At times where demand increases (rush hour, closing time for the bars, etc.) the supply of drivers also increases, as more drivers will become available at periods of high potential business.



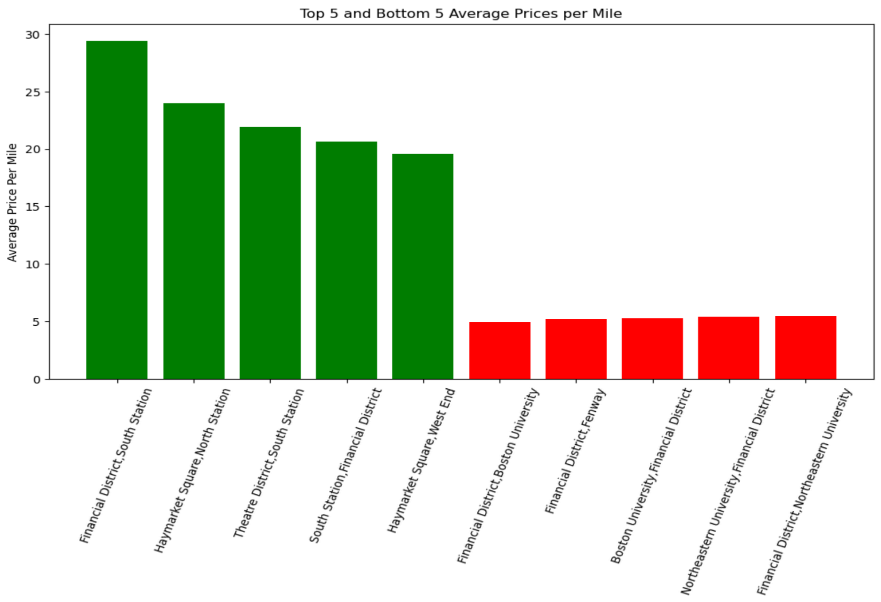


Two variables that did show strong correlations with price were distance, and the car type selected for the ride. To analyze the correlation between distance and price I created a hexbin plot, which is a combination of a scatter plot and a 2D histogram, as the data was particularly dense. This allowed me to visualize not only the relationship between those two variables, but also the distribution of the rides. For the distribution of car types, I used boxplots to compare prices for different car types on the same graph, which made it easier to see how prices vary based on the type of car that was ordered. One important aspect to note is Uber and Lyft have different names for the same type of vehicles. For example, "Uberpool" and "Lyft-shared" are the same concept and have very similar distributions, but the differing names in the car type feature make it seem as though they represent two distinct cars categories.





Finally, to analyze if "source" or "destination" had any significant impact on the price of the ride. To visualize this, I plotted the top 5 most expensive rides based on the source and destination and the cheapest 5 rides. I also normalized the distances, to easily compare the average price per mile of these trips. Although it may seem like going to or from the Financial District is the cheapest ride, it is also responsible for the most expensive price-per-mile ride. Further examining the data, I found that the 5 cheapest rides were, on average, the longest trips in the dataset, while the top 5 most expensive rides tended to be among the shortest. This suggests that the base cost of calling a ride comprises the majority of the price, while the actual cost per mile decreases as the trip distance increases.



3. Methods

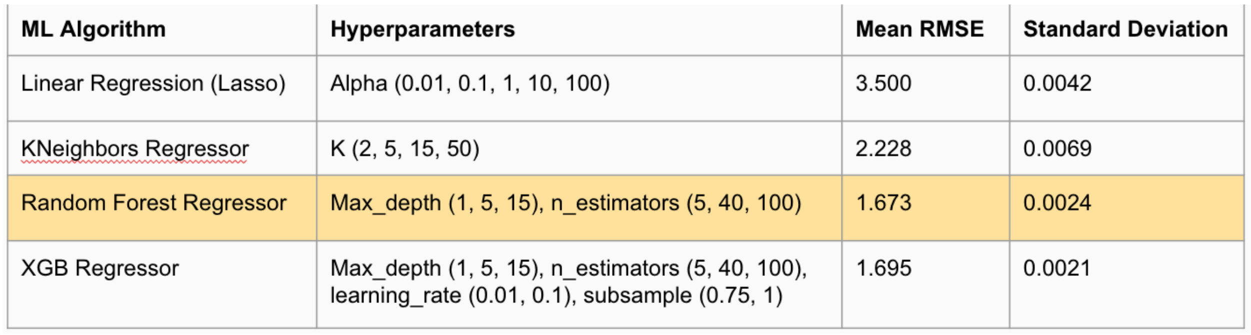
The dataset is very large and doesn't have any sort of group structure so the data was split using a basic train-test split. The training set was 60% of the data, while the remaining 40% of the data was split in half to create the validation and testing sets. This gave us a 60-20-20 split of our data for our three sets.

To preprocess the data, the first step was to manually standardize the "car_type" feature, so that similar car types were given the same name to keep consistency. For example, "UberX" and "Lyft" were the names of both companies' most basic vehicles, so for the car_type feature both were renamed to 'Basic'. Next, I used sklearn's ColumnTransformer to preprocess the data with the following preprocessors:

Preprocessor	Features
OrdinalEncoder	'car_type', 'day_of_week', 'hour'
OneHotEncoder	'source', 'destination', 'cab_type', 'short_summary'
StandardScaler	'distance', 'temperature', 'surge_multiplier'

ML Pipeline

With the data now preprocessed, I created 4 machine learning models, each trained on 3 random states. Only 3 random states were needed here because of the large size of the dataset, so the impact of randomness on our models is very low. For each random state, each model was trained by looping through the hyperparameter grids seen below, and the model with the lowest RMSE validation score would be saved, giving us 12 saved models. For each model, the RMSE on the test set was calculated and averaged across all three random states to determine the model's mean RMSE. The standard deviation of the RMSE scores helped assess the randomness associated with each model due to the random state chosen.



ML Algorithm	Hyperparameters	Mean RMSE	Standard Deviation
Linear Regression (Lasso)	Alpha (0.01, 0.1, 1, 10, 100)	3.500	0.0042
KNeighbors Regressor	K (2, 5, 15, 50)	2.228	0.0069
Random Forest Regressor	Max_depth (1, 5, 15), n_estimators (5, 40, 100)	1.673	0.0024
XGB Regressor	Max_depth (1, 5, 15), n_estimators (5, 40, 100), learning_rate (0.01, 0.1), subsample (0.75, 1)	1.695	0.0021

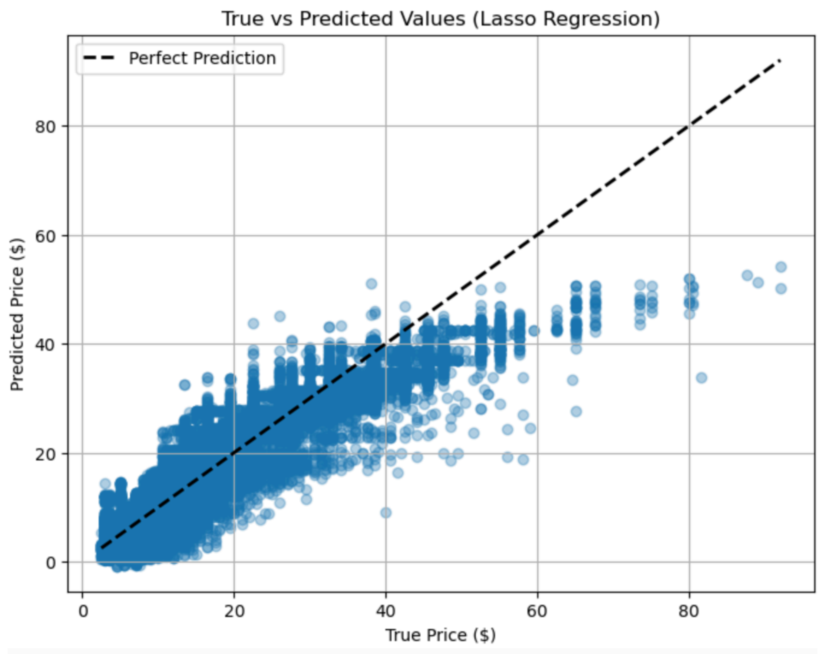
The best performing model based on the RMSE score is the Random Forest Regressor with a mean RMSE of 1.673.

4. Results

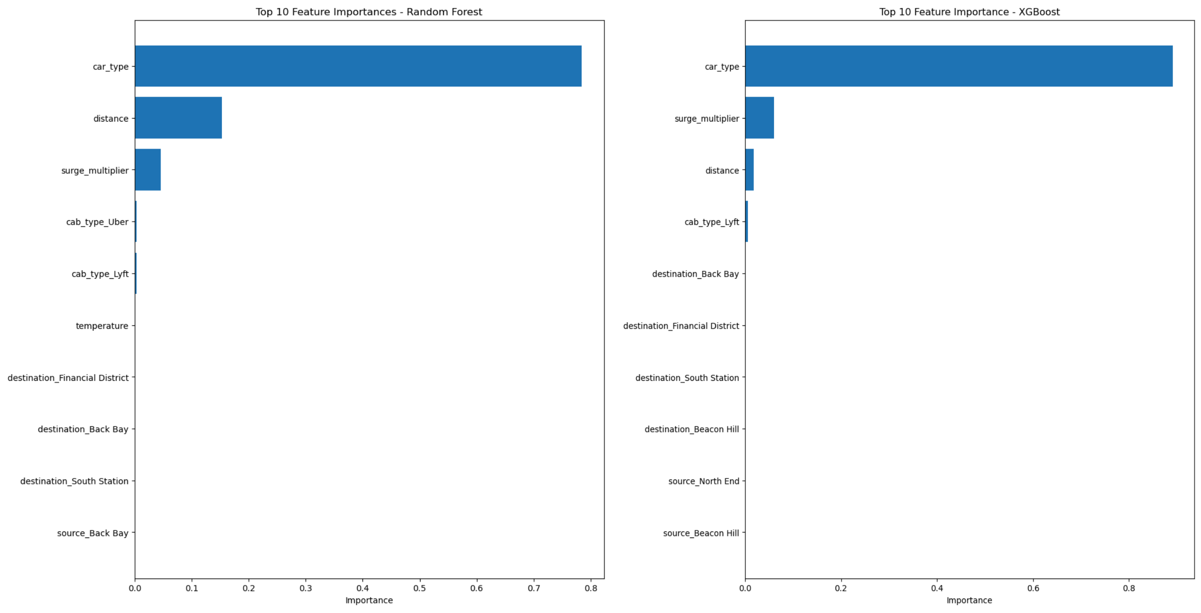
To evaluate the performance of the models, the mean RMSE scores were compared to the baseline RMSE. The baseline score was calculated by taking the mean of the target variable for each random state and using it as the predicted value for every observation in the test set. These three RMSE scores were then averaged to produce the baseline RMSE. As shown below, the standard deviations above the baseline RMSE are quite large due to the low standard deviation within our model scores, which is a result of the dataset's size.

ML Algorithm	Mean RMSE	STD	Baseline RMSE	Baseline STD	STD above Baseline
Linear Regression (Lasso)	3.500	0.0042	9.330	0.0031	1866.44
KNeighbors Regressor	2.228	0.0069	9.330	0.0031	2273.50
Random Forest Regressor	1.673	0.0024	9.330	0.0031	2451.25
XGB Regressor	1.695	0.0021	9.330	0.0031	2444.24

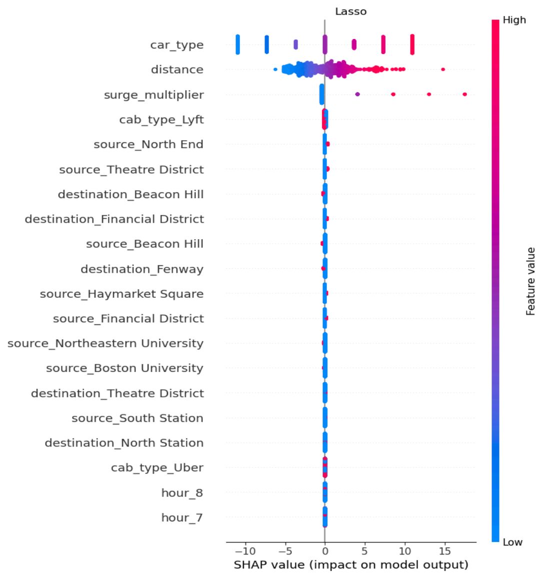
It is also useful to visualize the predicted prices against the true prices to see how our model is performing with respect to the different price tiers. As we can see in the graph below, the Lasso model does fairly well in predicting the prices for cheaper rides, but when the prices get higher and become outliers the predictions become worse.



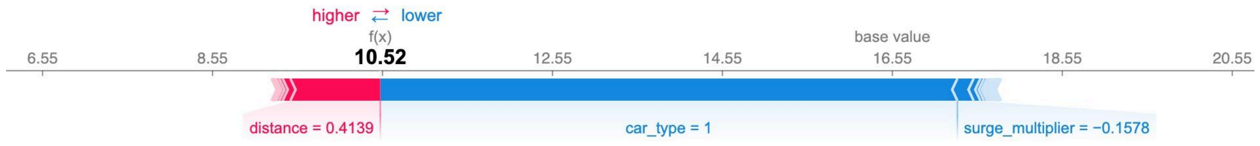
To analyze which features are the most important in our best performing models, XGB Regressor and Random Forest Regressor, I examined the feature importance values generated by their respective functions . The graphs show that for both models, car type is by far the most influential feature in predicting price, followed by distance and the surge multiplier. This is no surprise to have these three features as our most important, as our EDA led us to believe this would be the case. However, It was surprising to see how much higher the car type feature importance was than the rest of the variables, I would expect distance to be slightly closer.



For our Lasso model, the SHAP feature importance plot was used to visualize the contribution of each feature to the model. Meanwhile, the KNeighbors model relies on the closest K data points to the point of interest to make predictions, so no direct feature importance values are available. We can see here that the SHAP plot of our Lasso model tells the same story as our other two feature importance plots: car type is the most influential feature followed by distance and surge multiplier.



To gain a deeper understanding of how these model features look when applied to an individual point, we can use a SHAP force plot to visualize local feature importance. This plot shows us for index= 32 how our feature values impact the predicted price for our Lasso model. It is evident here that the car type pushes the prediction down dramatically from the base value of 16.55, and with some push and pull from all the other features the predicted price will be 10.52.



5. Outlook

One significant challenge in building this predictive model was the lack of publicly available data from Uber and Lyft. This is like trying to figure out what a puzzle is supposed to look like without all the pieces. If that data was available I would be able to build a more comprehensive model by incorporating features that would more than likely play a role in price setting, such as demand. Demand in particular is an intriguing feature because I would be interested to see how it plays a role in feature importance alongside our other variables.

In terms of the model itself, with more computing power I would like to expand the parameter grid used for each model and see if that can improve performance. Another avenue to explore would be building individual models for each car type, because we know how big of a role that plays in determining price. Obviously ordering a luxury car will be more expensive than a basic car, but it seems to be hiding the nuances other variables may have when it comes to determining pricing. For instance, if a model was built on just basic cars alone, would distance dominate the model's feature importance like car type currently does, or would other variables become more important predictors?

Finally, I would explore the possibility of improving model performance by using only the most predictive features. By building a model with a streamlined set of variables, we could reduce computational overhead without sacrificing—and potentially even improving—our evaluation score. This approach would not only enhance efficiency but also clarify the essential drivers of pricing, offering a more focused view on the problem.

6. References

[1]https://www.massconvention.com/about-us/news-item/there-were-nearly-100000-uber-and-lyft-rides-per-day-in-boston-last-year