Driven by Data: Predicting Used Car Prices with Advanced Machine Learning

Maxwell Yang (my37), Lucas Duffy (ld14), Kelly Lee (c129)

2024-12-12

Project Description and Abstract

The goal of this project was to predict car prices using a dataset called used_cars. It contained data on several thousand cars, each represented by key variables describing its make, model, condition, and other features. The dataset went through a lot of preprocessing and cleaning, from handling missing values, consolidating categorical levels, scaring various numerical variables, and removing outliers. There were also key transformations including converting long color descriptions into simpler colors and splitting columns into multiple columns to ensure that the data was well prepared.
Clustering techniques, including Hierarchical Clustering, K-means, and Spectral Clustering, were applied to explore the dataset and identify patterns among the cars. These methods highlighted the features that most influence similarity and provided insights to inform predictive modeling. To predict car prices, five regression models were implemented: Lasso Regression, Support Vector Machine (SVM) with an RBF kernel, Random Forest, Gradient Boosting, and K-Nearest Neighbors (KNN). Each model was tuned through cross-validation, with RMSE serving as the primary evaluation metric. Random Forest highlighted car models and mileage as key predictors but struggled with overfitting. Lasso Regression demonstrated strong generalizability, identifying engine size, manual transmission, and green exterior color as influential predictors. SVM and Gradient Boosting provided nuanced insights, with mileage, model year, and accident history consistently impacting price. KNN regression, though simple, offered competitive performance but explained less variance.
In our analysis, we incorporated unique methods beyond those reviewed in the literature, particularly in data preprocessing and model tuning. While the literature emphasized traditional supervised learning techniques like Random Forest and Lasso regression, we extended these approaches by introducing additional dimensionality reduction steps, such as consolidating categorical levels and removing rare categories to address overfitting risks in high-dimensional data. Furthermore, for Random Forest, we used 5-fold cross-validation for hyperparameter tuning instead of a fixed split approach, ensuring more robust parameter selection. For gradient boosting, we employed early stopping based on validation performance, a technique not mentioned in the reviewed studies, which helped mitigate overfitting while maintaining model complexity. AI tools were employed for code optimization, grammar correction, and model evaluation, adhering to academic integrity guidelines. This multi-method approach highlights the strengths and tradeoffs of predictive models in real-world applications.

Literature Review

With the increased price of new cars, many car buyers are turning to used cars. New car prices are fixed by governments to prevent price gouging by manufacturers and dealerships, but used cars have little to no regulations on sale price. Different used car marketplaces have different metrics to determine if a listing is a “good deal” or not. For the two research papers I’ve read, the two regression models they used to predict used car prices are random forests and Lasso regression.

The random forest model only focused on listings from private owners and ignored dealership listings. To make analysis relevant, they also filtered the release year from 1863 to 2017. The variables they focused on directly were price, kilometers driven, brand, type of vehicle, age of vehicle, fuel type, is automatic, damage repaired, and horsepower. They fitted a random forest model by using the Grid Search Algorithm, a systematic method that tunes parameters such as mtry, nodesize, and numtrees to find the best performing parameters on the dataset. Eventually, 500 was the desired numtrees and mtry was every variable. The data was split into a 70:20:10 split, randomly chosen, for training data, testing data, and cross-validation data respectively. Training error was significantly lower than testing error, which means there was some overfitting in the model. Overall, it was concluded that the most relevant features were price, kilometers driven, brand, and vehicle type.
The second research paper that uses Lasso regression has similar data cleaning processes and removed the same irrelevant variables that were inconclusive and had many missing data values. The data was split 70:30 into training data and testing data, randomly sampled. They used Lasso regression with 10-fold cross validation to find the optimal subset of attributes. Including automatically generated dummy variables, there were 67 total estimates. The attributes included were mileage, model of car, trim, make, car type, cylinder type, has cruise control, has speakers, has leather seats, and fuel tank capacity. Since Lasso relies on the training set to find the best attributes, there is potential for underfitting in this model.
Both research papers used valid supervised learning techniques to determine used car prices, and both models had their benefits and pitfalls, with random forests overfitting the data and Lasso regression underfitting the data. The datasets used by the two research papers are also different, so it adds a layer of complexity when comparing the results of the two models. We will aim to use these models in the model building section of the report, taking into account the results from the research papers. We may use additional methods such as glmnet (combining Lasso and Ridge) and boosting for random forests to further improve the models and gain more conclusive results. Although not covered, the use of unsupervised models also provide benefits such as feature engineering and outlier detection.

Mukkesh, G., & Bhupathi, M. (2020). Used cars price prediction using supervised learning techniques. Retrieved from https://www.researchgate.net/

Roy, R., & Khatua, S. (2019). Price prediction of used cars using machine learning algorithms. In J. K. Mandal, D. Bhattacharya, & S. Pal (Eds.), Emerging Technologies in Data Mining and Information Security (pp. 301–310). Springer. https://doi.org/10.1007/978-3-030-03402-3_28/

Data Processing and Summary Statistics

```
## [1] 4009
```

```
## [1] 3964
```

For the first steps of data processing, we noticed that there were many data entries that were blank or "-", so I changed it to NA. We also found that it is impossible to predict fuel type if we do not know any information of the engine, if both engine and fuel type were missing we dropped the row. It did not impact the data that much however, because we only dropped 45 rows out of 4009 rows.

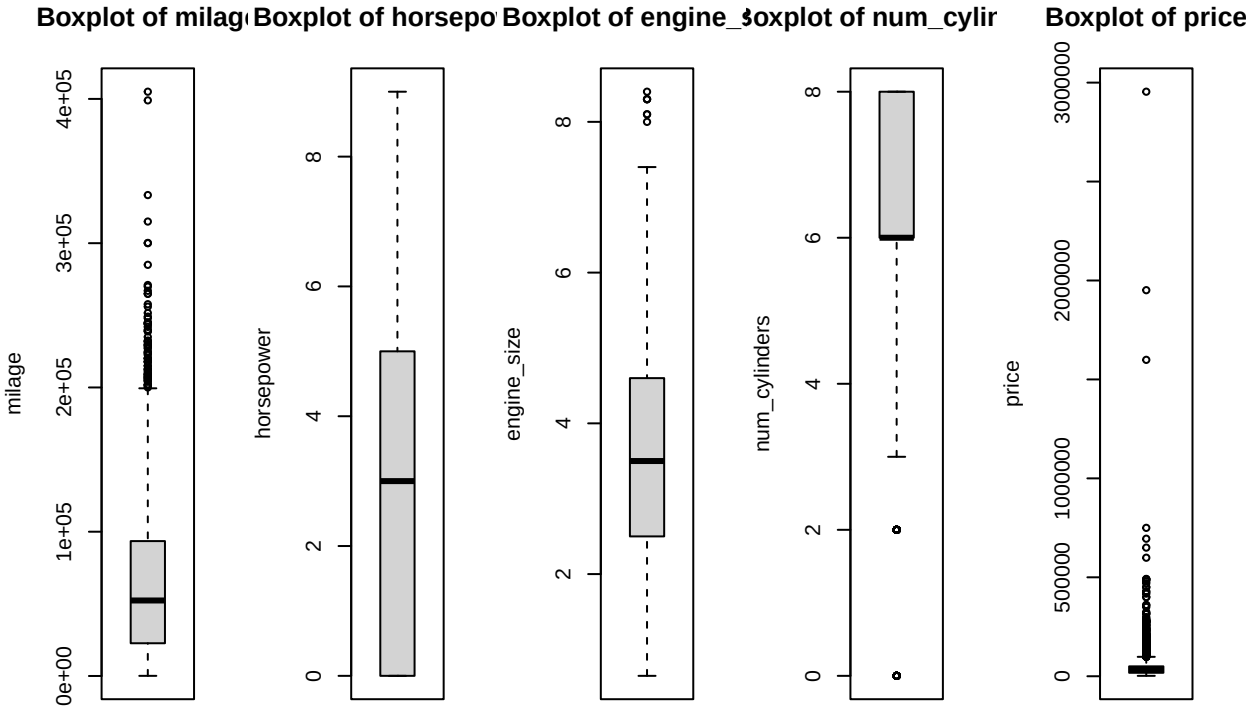
```
## Warning: NAs introduced by coercion
## Warning: NAs introduced by coercion
## Warning: NAs introduced by coercion
```

Next, we split the engine data and created three new columns, horsepower, engine_size, and num_cylinders.

Because dropping all the rows with NA for horsepower, enginesize, num_cylinders, and price would be dropping too many rows, I replaced all the NA in the numerical columns with the overall median. We chose median because the initial data had a lot of outliers and the mean is skewed.

Next, we went on to clean the categorical variables. We decided that not reporting clean_title should be left as unknown, as the only other category type is "Yes". However, we cannot definetly say that it would be "No" so we would leave it as "Unknown". For accident, we assume that if it was left blank, it would be "None-reported". Finally, if the fuel type was left blank, we checked using the engine data that it was an Electric vehicle, hence we updated it to "Electric."

Lastly, there are many cars that do not have their exterior and/or interior color specified. We change NA to "not specified" for now to prevent any errors while model building. There are also cars with transmission missing, which we also replace with not specified.




```
##      milage         horsepower     engine_size    num_cylinders
##  Min.   :   100   Min.   :0.000   Min.   :0.650   Min.   :0.000
##  1st Qu.: 22689   1st Qu.:0.000   1st Qu.:2.500   1st Qu.:6.000
##  Median : 52338   Median :3.000   Median :3.500   Median :6.000
##  Mean   : 64357   Mean   :3.053   Mean   :3.676   Mean   :6.048
##  3rd Qu.: 93500   3rd Qu.:5.000   3rd Qu.:4.600   3rd Qu.:8.000
##  Max.   :405000   Max.   :9.000   Max.   :8.400   Max.   :8.000
##      price
##  Min.   :   2000
##  1st Qu.:  17500
##  Median :  31450
##  Mean   :  44653
##  3rd Qu.:  49998
##  Max.   :2954083
```

```
## [1] 0.06281534
```

We see that about 6% of the data are considered outliers. This was expected, especially for the price variable, where cars have a huge range of prices, and luxury cars are less frequent but very expensive. After running some models on the data, we realized that some columns were too impactful, such as brand_Lamborghini, since there are only a couple rows of lamborghinis. We decided to remove these rows

instead of imputating them with quantile values to prevent tampering with the data. Since the outliers only accounted for 6% of the data, we decided that removing them would help us interpret our models better and not change the underlying trend in our data. We also decided to remove the "model" column from our data, since a lot of its information can be extracted from other columns such as engine_size and brand.

```
##      brand model_year milage     fuel_type
## 1     Ford       2013  51000 E85 Flex Fuel
## 2  Hyundai       2021  34742      Gasoline
## 3    Lexus       2022  22372      Gasoline
## 4 INFINITI       2015  88900        Hybrid
## 5     Audi       2021   9835      Gasoline
## 6    Acura       2016 136397      Gasoline
##                                                engine      transmission
## 1 300.0HP 3.7L V6 Cylinder Engine Flex Fuel Capability       6-Speed A/T
## 2                                  3.8L V6 24V GDI DOHC 8-Speed Automatic
## 3                                       3.5 Liter DOHC         Automatic
## 4  354.0HP 3.5L V6 Cylinder Engine Gas/Electric Hybrid       7-Speed A/T
## 5                            2.0L I4 16V GDI DOHC Turbo 8-Speed Automatic
## 6                                           2.4 Liter                 F
##                  ext_col int_col                              accident
## 1                  Black   Black At least 1 accident or damage reported
## 2        Moonlight Cloud    Gray At least 1 accident or damage reported
## 3                   Blue   Black                          None reported
## 4                  Black   Black                          None reported
## 5 Glacier White Metallic   Black                          None reported
## 6                 Silver  Ebony.                          None reported
##   clean_title price horsepower engine_size num_cylinders
## 1         Yes 10300          0         3.7             6
## 2         Yes 38005          3         3.8             6
## 3     Unknown 54598          3         3.5             6
## 4         Yes 15500          4         3.5             6
## 5     Unknown 34999          3         2.0             6
## 6     Unknown 14798          3         2.4             6
```

One particular issue that occurs is the number of levels in exterior color and interior color of the car. Many entries include more descriptive versions of colors, such as "light blue" and "blue". To reduce dimensionality of data, we combined every basic color with a descriptor into just the basic color. So for example, all instances of "light blue" was changed to blue.

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Similarly, transmission also has a problem of too many levels. Transmissions are abbreviated as at (automatic) or mt (manual) in some rows but fully included in other rows. We also generalized the speeds of all

transmission, so we end up only having 4 levels: automatic, manual, both, or not specified. This reduces again reduces dimensionality in our data.

```
threshold <- 50
for (col in names(df)) {
  if (!(col %in% c("model", "model_year", "milage", "price", "horsepower", "engine_size", "num_cylinders
    freq_table <- table(df[[col]])
    rare_levels <- names(freq_table[freq_table < threshold])
    df[[col]] <- factor(ifelse(df[[col]] %in% rare_levels, "Other", as.character(df[[col]]))) }}
```

Lastly, we do a final sweep to reduce dimensionality by combining "rare" levels into an "Other" level. There are many instances where a model or color only shows up once, which could skew the data and produce inconclusive results. Every level with less than 50 observations was classified to the "other" level.

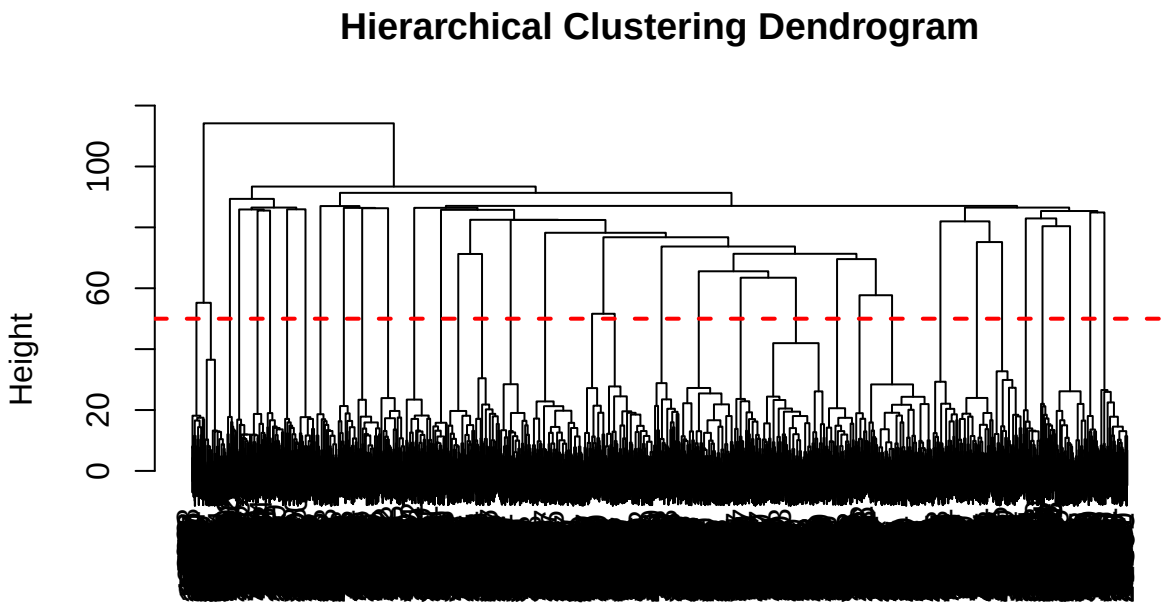
Unsupervised Learning

```
library(dplyr)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
df_encoded <- df %>%mutate(across(where(is.factor), as.character)) %>% dummyVars(~ ., data = ., fullRank
df_encoded <- as.data.frame(df_encoded)
df_features <- df_encoded %>%select(-price)
numeric_columns <- sapply(df_features, is.numeric)
df_features[numeric_columns] <- scale(df_features[numeric_columns])
distance_matrix <- dist(df_features, method = "euclidean")
hc <- hclust(distance_matrix, method = "ward.D2")
plot(hc, main = "Hierarchical Clustering Dendrogram", xlab = "", sub = "")
abline(h = 50, col = "red", lty = 2, lwd = 2)
```

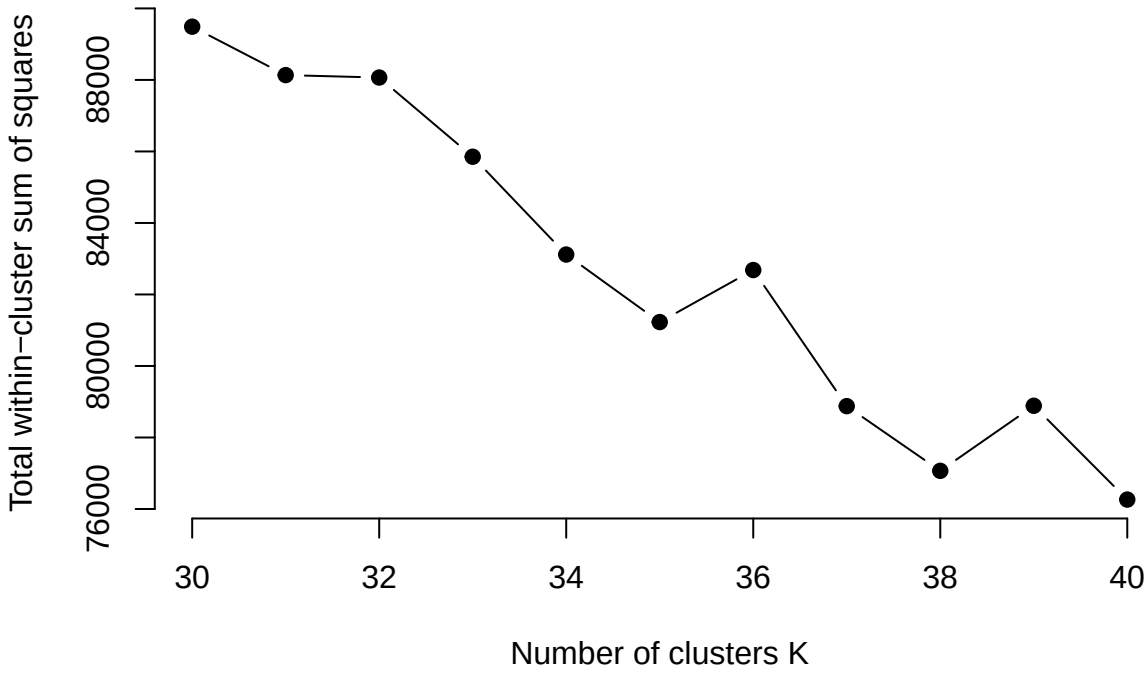


```r
k <- 35
clusters <- cutree(hc, k = k)
df_hclust <- df %>% mutate(cluster = clusters)
summary_by_cluster <- df_hclust %>%group_by(cluster) %>%summarize(avg_price = mean(price, na.rm = TRUE)
print(summary_by_cluster)
```

```
## # A tibble: 35 x 3
##    cluster avg_price count
##      <int>     <dbl> <int>
##  1       1    22639.    97
##  2       2    18895.    70
##  3       3    30661.   105
##  4       4    22823.    58
##  5       5    30022.   115
##  6       6    41833.   107
##  7       7    41408.    57
##  8       8    48803.    85
##  9       9    42809.   105
## 10      10    27535.   138
## # i 25 more rows
```

The first unsupervised clustering algorithm is hierarchical clustering using hclust. We implement a one-hot encoding method to convert the categorical variables to numerical so they can be included in the process. This one-hot encoded data will be used for the remainder of the clustering portion. We isolate our outcome, price, from our features and begin clustering. Now, in order to determine the optimal number of clusters we plot complete linkage individually and draw a horizontal line through the middle of the largest vertical difference between nodes the number of vertical lines it passes through is the number of clusters. It is determined to be occurring around a height value of roughly 50, drawing the line through the dendrogram gives 35 clusters. After the observations are split into clusters a summary is provided that allows us to determine if clusters are associated with price. With many differing average prices it is clear that the clustering was able to successfully use features to isolate on the outcome variable. Beyond the differing average prices a quick glance indicate that the clusters also tell us information about the clusters, for example cluster 1 is completely filled with e85 Gasoline cars, which indicates that unique features like that one could often group together in clusters, this phenomenon is also seen with other unique traits like "both" transmission. These clusters therefore allow for an individual to assess how certain traits affect price and leverage their own preferences to maximize their consumer surplus.

```r
scaled_data <- scale(df_features)
k_values<- 30:40
wss <- numeric(length(k_values))
set.seed(42)
for (i in seq_along(k_values)) {
  k <- k_values[i]
  kmeans_result <-kmeans(scaled_data, centers = k, nstart = 25)
  wss[i] <- kmeans_result$tot.withinss
}
plot(k_values, wss, type = "b", pch = 19, frame = FALSE, xlab = "Number of clusters K",
ylab = "Total within-cluster sum of squares")
abline(v = which.min(wss), col = "red", lty = 2)
```




```
scaled_data <- scale(df_features)
k <- 35
set.seed(42)
kmeans_result <- kmeans(scaled_data, centers = k, nstart = 25)
df_kmeans <- df %>% mutate(cluster = kmeans_result$cluster)
summary_by_cluster_kmeans <- df_kmeans %>%group_by(cluster) %>%summarize(avg_price = mean(price, na.rm =
print(summary_by_cluster_kmeans)
```

```
## # A tibble: 35 x 3
##    cluster avg_price count
##      <int>     <dbl> <int>
##  1       1    31367.   325
##  2       2    40803.    42
##  3       3    44148     35
##  4       4    45859.   105
##  5       5    15676.    70
##  6       6    59025.    52
##  7       7    21334.    62
##  8       8    20994.    57
##  9       9    18895.    70
## 10      10    26099.    77
## # i 25 more rows
```

```
anova_result <- aov(price ~ cluster, data = df_kmeans)
summary(anova_result)
```

```
##               Df    Sum Sq   Mean Sq F value  Pr(>F)
## cluster        1 3.813e+09 3.813e+09   8.519 0.00354 **
## Residuals   3595 1.609e+12 4.475e+08
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
```

The next clustering method is kmeans. We decided to use the elbow method to determine the number of clusters for the kmeans approach. The bend occurred at 35 therefore 35 clusters. The anova test results helped confirm that there was a distinction between the clusters. The clusters are similar to hierarchical in the fact that certain unique features end up in the same cluster, but in this instance there is a little less of that occurrence. The fact that features drive the clustering could allow a consumer to use clusters to determine what impacts the price of used cars. There exists a distinction by price between the clusters, and price was not included in the clustering to avoid over fit, so this observed distinction of price is a result of clustering selection.

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(cluster)
library(dplyr)
```

```
data_matrix <- as.matrix(df_features)
best_k <- 30
final_spectral_result <- specc(data_matrix, centers = best_k)
df_spec <- df %>%mutate(cluster = final_spectral_result)
summary_by_cluster_spec <- df_spec %>%group_by(cluster) %>%summarize(avg_price = mean(price, na.rm = TRU
print(summary_by_cluster_spec)
```

```
## # A tibble: 30 x 3
##    cluster avg_price count
##    <specc>     <dbl> <int>
##  1  1         42809.   105
##  2  2         49040.    33
##  3  3         43625.    86
##  4  4         32244.   123
##  5  5         27303.    49
##  6  6         22727.    59
##  7  7         22823.    58
##  8  8         27492.    67
##  9  9         34287.    96
## 10 10         47682.    58
## # i 20 more rows
```

```
anova_result_spec <- aov(price ~ as.factor(cluster), data = df_spec)
summary(anova_result_spec)
```

```
##                      Df    Sum Sq   Mean Sq F value Pr(>F)
## as.factor(cluster)   29 1.997e+11 6.887e+09   17.39 <2e-16 ***
## Residuals          3567 1.413e+12 3.961e+08
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#Add cluster info to df so it can be included in supervised regression
df$spec_clust <- as.factor(final_spectral_result)
df$kmeans_clust <- as.factor(kmeans_result$cluster)
df$hc_clust <- as.factor(clusters)
df$spec_clust <- as.numeric(as.character(df$spec_clust))
df$kmeans_clust <- as.numeric(as.character(df$kmeans_clust))
df$hc_clust <- as.numeric(as.character(df$hc_clust))
```

For spectral clustering we decided to use the silhouette method to determine the optimal number of clusters, over a range of 30:40. The method took an extremely long time to execute, so based on previous clustering methods we decided to use a cluster size. The clusters produced by spectral are significantly more intertwined this could be a indication of spectral's graph based structure that allows it to handle demonstrability more effectively. The anova test indicates that the clusters are significantly different, and the summary of avg_price in the clusters indicate a difference between clusters for our outcome variable.

Overall, the clustering algorithms allow us to identify similar car profiles in the data, which indicates to us cars that would affect a potential supervised model in a similar manner. Additionally, it isolates some features and intertwines others, which may indicate which features are most unique or "distant" from others, like for example e85 gasoline is typically isolated in its own cluster while "both" transmission is often spread across many clusters. This observation tells us that e85 cars are typically extremely unique and are often aligned along other features.

We utilized thee clusters created by the algorithms in the supervised predictive models in order to catch unseen trends and improve the feature selection.

Another use case for clusters in supervised models is using models to predict within each cluster, which could control for similarities and help isolate and identify features that affect prices.

Prediction Models

```r
set.seed(1)
n = nrow(df)
train_index <- sample(seq_len(n), size = floor(0.8 * n))
train_data <- df[train_index, ]
remaining <- df[-train_index, ]
test_index <- sample(seq_len(nrow(remaining)), size = floor(0.2 * n))
test_data <- remaining[test_index, ]
```

We partition our data into training and testing data, using an 80-20 split. We make sure to sample randomly to avoid bias.

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

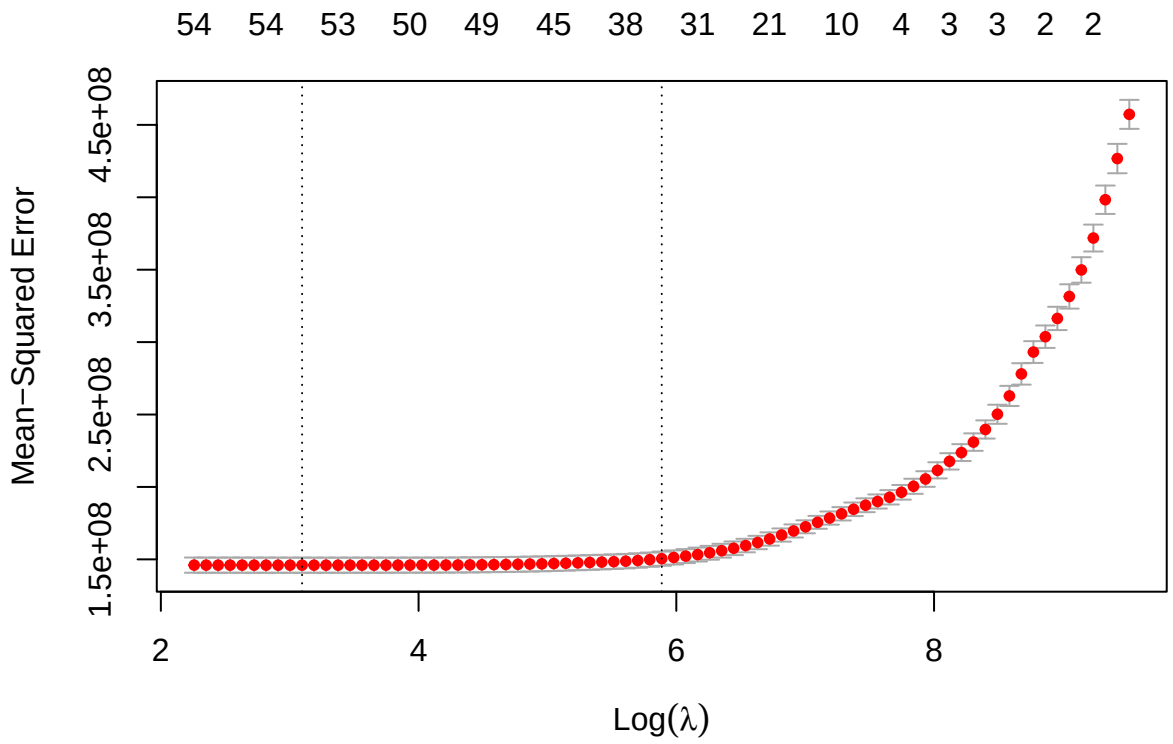
```
x_train <- model.matrix(price ~ ., data = train_data)[, -1]
y_train <- train_data$price
x_test <- model.matrix(price ~ ., data = test_data)[, -1]
y_test <- test_data$price
test_indices <- which(rownames(orig_df) %in% rownames(x_test))
```

Since the split of training and testing data is random, there are levels that are present in the training data and not present in the testing data. We apply the process of harmonization, which is including all levels for each categorical variable, in the testing and training data to prevent errors.

```
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10)
best_lambda_lasso <- cv_lasso$lambda.min
head(sort(as.matrix(coef(cv_lasso, s = "lambda.min"))[-1, , drop = TRUE], decreasing = TRUE), 5)
```

```
##       brandPorsche          brandLand           brandBMW brandMercedes-Benz
##          21926.395          11069.567           7712.194           7430.668
##        engine_size
##          6803.056
```

```
plot(cv_lasso)
```




```
lasso_preds <- predict(cv_lasso, s = "lambda.min", newx = x_test)
lasso_preds_1se <- predict(cv_lasso, s = "lambda.1se", newx = x_test)
lasso_mse <- mean((lasso_preds - y_test)^2)
lasso_mse_1se <- mean((lasso_preds_1se - y_test)^2)
y_mean = mean(y_test)
r_squared <- 1 - (sum((y_test - lasso_preds)^2) / sum((y_test - y_mean)^2))
r_squared_1se <- 1 - (sum((y_test - lasso_preds_1se)^2) / sum((y_test - y_mean)^2))
```

```
## lambda.min:  22.10851
```

```
## lambda.1se:  360.3136
```

```
## Test RMSE for Lasso Regression using lambda.min: 10657.71
```

```
## Test RMSE for Lasso Regression using lambda.1se: 10811.37
```

```
## R-squared for Lasso Regression using lambda.min: 0.7208227
```

```
## R-squared for Lasso Regression using lambda.1se: 0.7127143
```

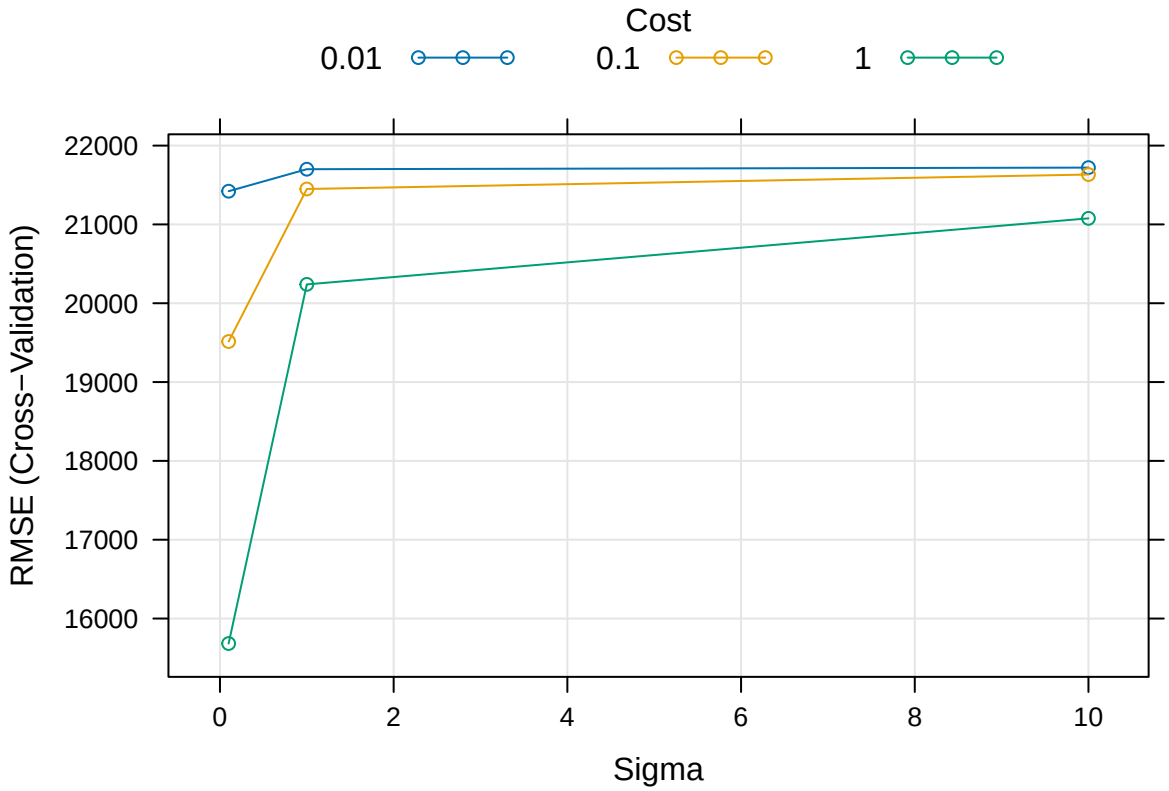
Looking at the results of the Lasso regression, we used 10-fold cross validation to tune and find the optimal lambda value. This means that we divided the training data into 10 subsets, where the model uses 9 subsets for training and the last subset for validation. This process is repeated 10 times, for each subset. The general goal of cross-validation is to prevent overfitting by ensuring the model can be generalized across different subsets of data. While choosing between lambda.min and lambda.1se, it appears the RMSE and R^2 is lower for lambda.min, so that is the preferred lambda value. This is because lambda.min focuses on minimizing RMSE whilst lambda.1se focuses on simplfying the model. We also want to tune to parameter alpha, which determines which type of penalty. The alpha value with the lowest RMSE was 1, which happens to be Lasso regression. RMSE calculates the error between the predicted value and testing value. It is in the same units as the testing set, and gives us a quantitative measure of how far away we are from the average. However, it is sensitive to outliers and can be inflated. R^2 explains how much variance is explained by the model and scale-invariant, however it can be inflated due to overfitting. In this context, RMSE is the superior model evaluation criteria, since we are interested in how much money we are off by and outliers are removed before running our regression. After looking at the coefficients for each model, it appears that the most influential predictors are brand porsche, brand land rover, and exterior color green. This makes some logical sense, as porsche and land rover are luxury cars, so name value alone makes the price more. A "green" exterior color also is a more unique color, so it would probably be on luxury cars or sports cars, making this particular color result in more expensive cars.

```
library(e1071)
library(caret)
library(kernlab)
```

```
set.seed(1)
svm_model <- svm(x_train, y_train, kernel = "radial", cost = 1, gamma = 1/ncol(x_train), scale = FALSE)
svm_preds <- predict(svm_model, x_test)
mse <- mean((svm_preds - y_test)^2)
tune_grid <- expand.grid(sigma = c(0.1, 1, 10), C = c(0.01, 0.1, 1))
control <- trainControl(method = "cv", number = 5)
svm_tuned <- train( x_train, y_train,method = "svmRadial",tuneGrid = tune_grid,trControl = control,metri
print(svm_tuned$bestTune)
```

```
##   sigma C
## 3   0.1 1
```

```
svm_final <- svm_tuned$finalModel
svm_final_preds <- predict(svm_final, x_test)
final_mse <- mean((svm_final_preds - y_test)^2)
r_squared <- 1 - ( sum((y_test - svm_final_preds)^2) / sum((y_test - mean(y_test))^2))
plot(svm_tuned)
```




```
varImp(svm_tuned)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 55)
##
##                          Overall
## milage                   100.000
## model_year                77.779
## accidentNone reported     12.999
## engine_size               12.696
## num_cylinders              7.931
## clean_titleYes             6.448
## fuel_typeElectric          6.181
## fuel_typeGasoline          5.973
## int_colgray                5.549
## brandPorsche               3.912
## fuel_typeHybrid            3.201
## int_colblack               2.708
## brandTesla                 2.108
## brandHyundai               2.011
## hc_clust                   1.983
## fuel_typeE85 Flex Fuel     1.803
## int_colred                 1.685
## ext_colsilver              1.582
## brandNissan                1.553
## transmissionManual         1.543
```

```
## Test RMSE for SVM: 20255.25
```

```
## Test RMSE for Tuned SVM: 13834.1
```

```
## R^2: 0.5296144
```

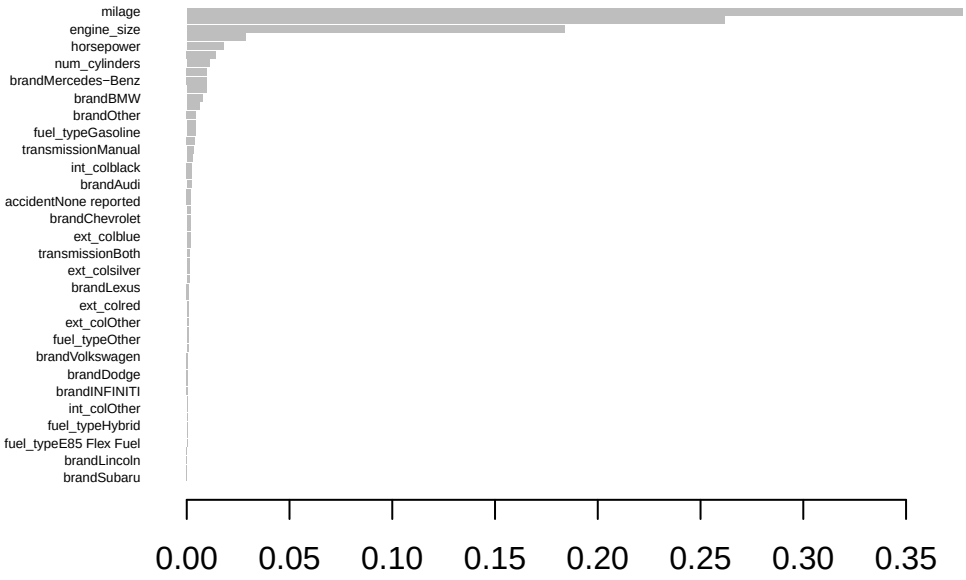
Looking at the results of using SVM as another regression method, we decide to use a SVM with the RBF (radial basis function) kernel. The parameters we needed to tune were Sigma, to control the width of the RBF kernel, and C (cost), that balances bias-variance tradeoff to achieve desired complexity but not overfitting. We first create a SVM model with C = 1 and gamma = 1/# of columns. This provides a baseline to compare our tuned svm model later. For our tuned svm model, we select from a variety of parameters for sigma and cost. We determine the optimal value for sigma is 0.1 and C = 1. We used 5-fold cross-validation to tune these parameters. To evaluate between the parameter values, we used RMSE, since we want to reduce the testing error, and R^2, to obtain the variance explained by each predictor. Based on the advantages of drawbacks of these metrics mentioned earlier, it is also more appropriate to RMSE in this case, since it has the same units as price, so it is easier to interpret. To identify the most impactful predictors, we use varImp to determine the relative performance of each predictor variable. Looking at the performance of each predictor, it appears that the most impactful predictors are milage, model_year, and no accident reported. This makes sense, as milage determines how realiable a car is, a later model_year means the car is newer, and no accident reported ensures the car's integrity is good.

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
dtrain <- xgb.DMatrix(data = as.matrix(x_train), label = y_train)
dtest <- xgb.DMatrix(data = as.matrix(x_test), label = y_test)
params <- list(objective = "reg:squarederror",eval_metric = "rmse",max_depth = 5,eta = 0.1,nthread = 2)
model <- xgb.train(params = params, data = dtrain,nrounds = 1000,watchlist = list(train = dtrain, test =
best_iter <- model$best_iteration
predictions <- predict(model, dtest, iteration_range = best_iter)
rmse <- sqrt(mean((predictions - y_test)^2))
r_squared <- 1 - ( sum((y_test - predictions)^2) / sum((y_test - mean(y_test))^2))
xgb.plot.importance(xgb.importance(model = model))
```



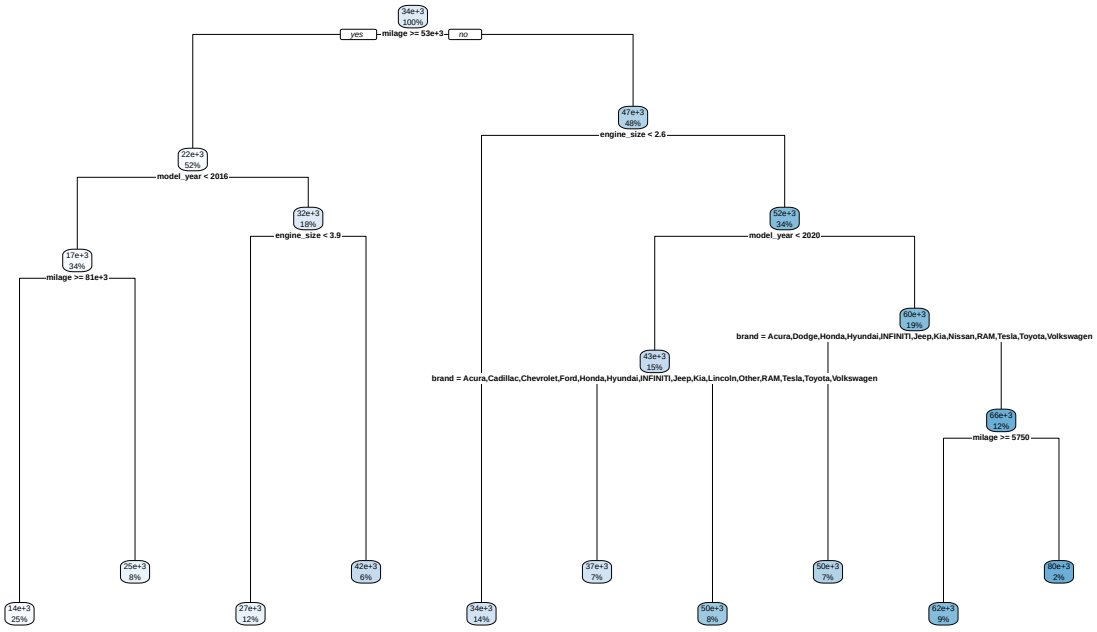

```
## Test RMSE: 8336.055
```

```
## R^2: 0.8292057
```

The next supervised machine learning model we tried to fit was a gradient boosting linear regression. Gradient boosting involves combining multiple weak learner models, in this case, decision trees. A learning rate is scaled to ensure that updates are gradual. There are many tuned parameters for this model, but the ones we found to have impact on the MSE were max_depth (the depth of the decision trees), nrounds (number of boosting rounds to perform), and learning rate. We first tried to implement a grid search to determine the optimal values for these parameters, but due to the complexity of this model, it took too much time. We manually tested values for all the parameters, with max_depth from 1 to 10, nrounds from 50 to 10000, and learning rate from between 0.01 and 1 manually. We end up with the optimal values being 5, 1000, and 0.1, respectively, those values had the lowest RMSE and R^2. As mentioned earlier, RMSE is the more appropriate performance metric, since our data is all the same scale and we want to have it in the same unit as price. Looking at the most optimal model and the importance plot, it appears that mileage, other brand, and Porsche. This makes logical sense, as mileage determines how reliable a car is, and other brand consists of brands with less than 50 observations, probably making them rarer and more sought after cars. The sample logic applies for Porsche, as it is a luxury car brand that has high prices on its models.

```
set.seed(123)
tree_model <- rpart( price ~ .,  data = train_data, method = "anova",control = rpart.control(cp = 0.01))
rpart.plot(tree_model, main = "Decision Tree for Car Price Prediction")
```

Decision Tree for Car Price Prediction




```
tree_predictions <- predict(tree_model, newdata = test_data)
mae <- mean(abs(tree_predictions - y_test))
rmse <- sqrt(mean((tree_predictions - y_test)^2))
r2 <- 1 - sum((tree_predictions - y_test)^2) / sum((y_test - mean(y_test))^2)
```

```
## Mean Absolute Error (MAE): 9678.591
```

```
## Root Mean Square Error (RMSE): 12803.94
```

```
## R-squared (R2): 0.5970607
```

The decision tree regression model's MAE was 9678.591, making the the average error in predicting car prices. The Root Mean Square Error (RMSE) was 12803.94, which is the overall magnitude of prediction errors and showing sensitivity to larger deviations. The R-squared (R^2) value of 0.597 suggests that the model explains approximately 59.7% of the variance in car prices, demonstrating reasonable predictive power and model strength but leaving room for improvement compared to more complex models. In the end, we prefer RMSE for the same aforementioned reasons as we want a metric in the same units as price. Key predictors identified by the decision tree included mileage, model year, fuel type, engine size, and transmission type. Mileage was a significantly influence in prices, with higher mileage cars typically valued lower, while newer model years commanded higher prices. Fuel type and engine size further distinguished cars by performance and efficiency, with certain types such as electric or larger engines increasing prices. Transmission type also played a role, particularly in specific markets like sports cars. Overall, the decision tree was a good model in providing insights into the factors driving car prices but was less effective in capturing complex relationships compared to advanced models like Gradient Boosting.

```
library(caret)
```

```r
set.seed(123)
preProc <- preProcess(x_train, method = c("center", "scale"))
newx_train <- predict(preProc, x_train)
newx_test <- predict(preProc, x_test)
tune_grid <- expand.grid(k = seq(3, 21, by = 2))
knn_model <- caret::train(x = newx_train,y = y_train,method = "knn",tuneGrid = tune_grid,trControl = tra
best_k <- knn_model$bestTune$k
knn_predictions <- predict(knn_model, newdata = newx_test)
mae <- mean(abs(knn_predictions - y_test))
rmse <- sqrt(mean((knn_predictions - y_test)^2))
r2 <- 1 - sum((knn_predictions - y_test)^2) / sum((y_test - mean(y_test))^2)
```

```
## Best k: 9
```

```
## Mean Absolute Error (MAE): 11050.63
```

```
## Root Mean Square Error (RMSE): 14560.42
```

```
## R-squared (R2): 0.4789249
```

KNN achieved a Mean Absolute Error (MAE) of 10,997 a Root Mean Square Error (RMSE) of 14,553, and an r-squared of 0.479 with optimal k = 7, which proves to explain a decent portion of the variance in car prices. Again, we prefer RMSE to keep the same units. While KNN provided competitive results, its simplicity limited its capacity to capture higher-dimensional relationships compared to more complex models. Despite this, it effectively identified relationships in the data, with mileage and model year being particularly influential in predicting car prices. We tuned k using bestTune to find the k with the lowest RMSE. For the K-Nearest Neighbors (KNN) regression model, the Euclidean distance metric was used to calculate the similarity between data points. Since KNN is a distance-based algorithm, it requires all input features to be numeric.

Open-Ended Question

```
##           brand             model model_year milage fuel_type
## 3832 Volkswagen Jetta 1.4T R-Line       2019  55234  Gasoline
## 731         BMW           M6 Base       2015  37000  Gasoline
## 2915       Land     Rover LR2 HSE       2010 100472  Gasoline
##                                                 engine
## 3832                          1.4L I4 16V GDI DOHC Turbo
## 731           560.0HP 4.4L 8 Cylinder Engine Gasoline Fuel
## 2915 230.0HP 3.2L Straight 6 Cylinder Engine Gasoline Fuel
##                        transmission               ext_col      int_col
## 3832              8-Speed Automatic Platinum Gray Metallic Black / Gray
## 731  Transmission w/Dual Shift Mode                 Silver       Silver
## 2915                    6-Speed A/T                   Gray        Beige
##           accident clean_title price horsepower engine_size num_cylinders
## 3832 None reported     Unknown 19290          3         1.4             6
## 731  None reported         Yes 45950          0         4.4             8
## 2915 None reported         Yes  9975          0         3.2             6
```

```r
three_cars[,"milage"] <- 0
three_cars[,"accidentNone reported"] <- 1
three_cars[,"clean_titleYes"] <- 1
dthree <- xgb.DMatrix(as.matrix(three_cars))
predicted_prices <- predict(model, dthree, iteration_range = best_iter)
true_data = c(24140, 118795, 36350)
rmse <- sqrt(mean((true_data - predicted_prices)^2))
```

```
## RMSE: 30340.2
```

```
## Predicted prices for the randomly selected cars with mileage set to 0:
```

```
## [1] 35181.89 68522.09 46946.91
```

For the open ended question of trying to predict car prices as if they were new, we decided to modify each row to be the following: set mileage to 0, since new cars cannot be driven yet, set accident to None since a new car cannot be in an accident, and set clean title to Yes since the car cannot be damaged. This may lead to inconclusive results, since older cars tend to have more miles, so setting mileage to 0 would make the model overestimate the "new" price of the car. I then searched on the internet for the actual prices of the new cars. I took the MSRP price for when the car was released, disregarding price depreciation. This is because our model is not a time series model and therefore cannot determine price depreciation. We used our gradient boosting model, since it had the lowest RMSE to predict the prices of the randomly selected cars and ended up with an RMSE of around 30000. Since we are restricted to only sampling three cars, there is a lot of variability within the results. For example, one of the cars selected was a BMW M6, and since gradient boosting applies some regularization to shrink parameters, the model may fail to recognize that BMW (a luxury car brand) would inflate the sale price, hence why my model underpredicted.