# Business Analytics Capstone

# Car Price Prediction

Submitted to: Mustafa Ahmed

Chik Hung, Tang (Ricky) 301243794

August 12, 2023

# Table of Contents

## Executive Summary

### 0.1 Executive Introduction

Every product or service comes with a price. Not only does it reflect the dynamics of supply and demand within a market, but also indicate the underlying relationship between prices and various external factors. The capability to precisely predict the price in the market allows corporations to differentiate from their competitors and dominate the market share. As Geely Auto aspires to penetrate the automotive market of the United States and Europe, an accurate price prediction model becomes a prerequisite for Geely to fine-tune the pricing strategies based on its business objectives.

### 0.2 Executive Objective

This project aims to equip Geely Auto with a state-of-the-art prediction model that suggests robust and precise pricing strategies. The goal is to optimize model performance to a high standard level, while keeping a minimum adjusted R-squared value of 0.8 to ensure reliable and accurate predictions in the future. Geely targets understanding the relationship between each car feature and its correlated prices. This goal will be accomplished by running various machine learning models to generate corresponding business insights driven by data.

### 0.3 Executive Model Description

Four machine learning models will be employed throughout this data analysis, including linear regression, decision trees, random forest, and gradient boosting. Each model possesses its own specialty to handle different scenarios and data patterns. Several modeling techniques will be implemented to enhance dataset interpretability, such as bootstrapping, feature engineering, dummy variables, data scaling, and data splitting. A thorough model comparison will be followed up at the end of the research to conclude actionable insights tailored to satisfy business needs.

### 0.4 Executive Recommendations

Tailored recommendations have been made to ensure Geely's sustained growth is consistently aligned with its business goals. Focusing promoting luxury and mid-range classes in foreign markets is expected to see potential growth in the long run, driven by appropriate pricing strategies from the machine learning model. Moreover, Geely should refrain from penetrating the automobile market with variables ranked at 0 importance, as such predictions could lead to inaccuracies and distorted results. By collectively guiding Geely Auto towards data-driven decision-making, it can position itself in diverse markets strategically and continue to succeed in the long term.

## Introduction

### 1.0 Background

The automotive industry is rapidly developing vehicles with advanced features, higher quality standards, and more robust performance. As Geely Auto is a well-established Chinese automobile brand and has a strong ambition to explore the US and European markets, accurately predicting vehicle prices becomes a prerequisite for this company to increase its competitiveness against various foreign competitors. This capstone project is tailor-made to predict car prices based on the importance of features. It aims to provide data-driven insights and visions to Geely Auto for adjusting its pricing strategies in the foreign market.

### 2.0 Problem Statement

Geely Auto is a Chinese automobile company aiming to set up its vehicle production line in the United States and Europe. To compete in the foreign market with an appropriate pricing strategy effectively, it is essential to comprehend the correlation between vehicle features and the manufacturer's suggested retail price (MSRP). By targeting this goal, the primary focus of this

capstone project lies in thoroughly studying the US and European automotive markets, assisting Geely Auto in developing efficient pricing and marketing decisions.

## 3.0 Objectives & Measurement

The primary objective is to develop various machine learning models for car price prediction that perform a minimum R-squared value of 0.8. This project aims to conduct a data-driven study using statistical analysis and compare learning models to identify the relationship and importance weights between car features and prices. Moreover, to further evaluate the performance and efficiency of the learning models, root mean squared error (RMSE) and mean absolute error (MAE) will be accounted in measuring the models' effectiveness. These metrics indicate the average differences between the predicted and actual prices, providing a quantitative summary to ensure that the results are well-aligned with the business goal.

## 4.0 Assumptions and Limitations

Data authenticity stands as the primary assumption to ensure the relevance and reliability of the vehicle dataset, providing value for Geely Auto's development of an efficient pricing strategy in the US and European markets, and allowing for fine-tuning of independent variables as needed. The assumption of consistency between predictor variables and the target variable also significantly prevents variations across different durations or periods. This mitigates time confusion and maintains the stability of various variables. Furthermore, this project assumes the dataset contains a high level of quality data with minimal missing values, typos, and outliers. This approach helps to thoroughly represent the relationship between different variables with minimal potential data biases. Moreover, the dataset's integrity must also be defined before any modeling processes. The number of data points consists of random samples with

representativeness within the population, encompassing a diverse range of observations from the larger population, even when dealing with a limited quantity of data points.

## Data Sources

### 5.0. Data Set Introduction

This dataset consists of 26 columns and 200 rows, indicating a myriad of car features and prices. It was conducted from multiple market surveys highlighting the relationship between the predictor and target variables. The dataset encompasses several data types, such as ‘int64’, ‘float64’, and ‘object’, representing numeric and categorial columns.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

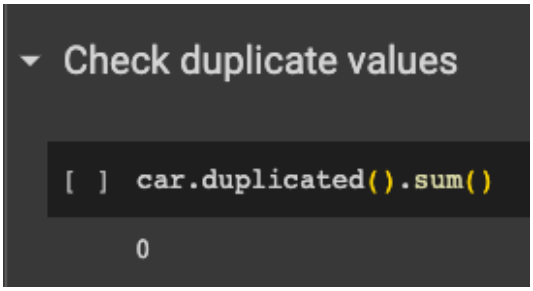
### 6.0. Exclusions

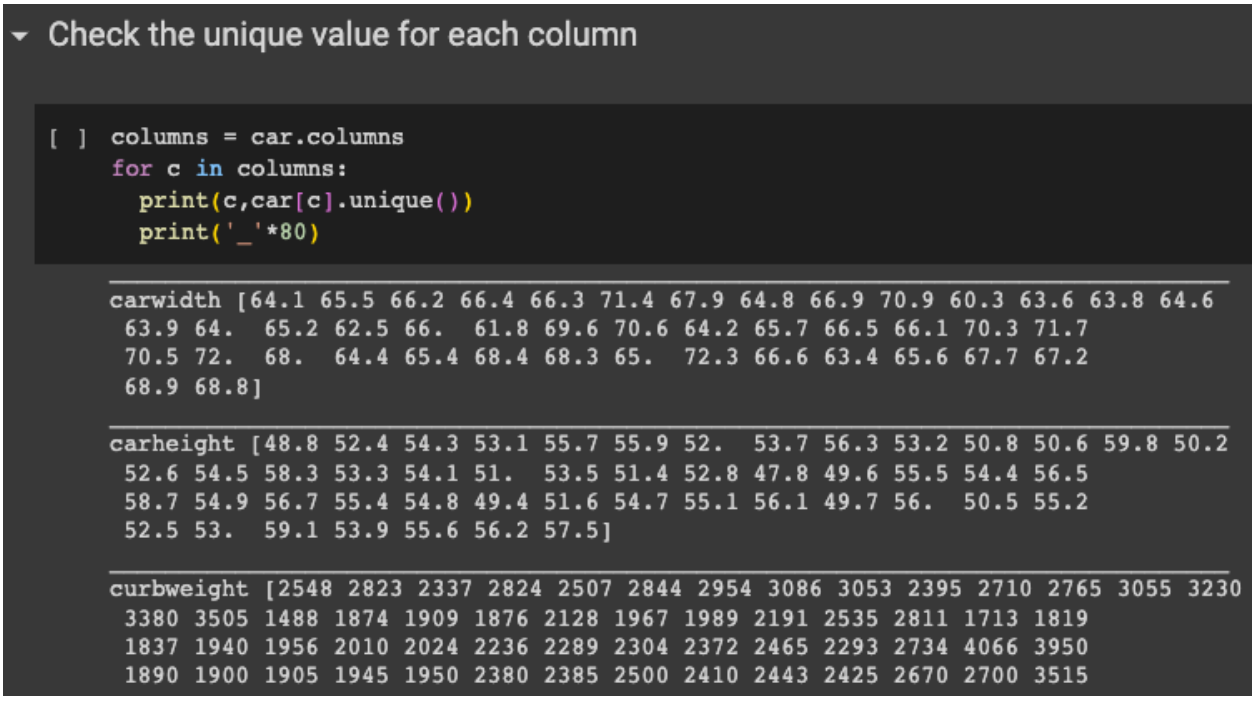
Exclusions were applied to ensure a high level of data quality and mitigate computational cost in our analysis. Anomalies, high cardinality columns, and data points unrelated to our

business goals were eliminated to enhance the integrity of our data exploration. Excluding irrelevant columns from the dataset leads to more reliable and accurate price predictions using various machine learning models. The excluded columns consist of door number, symboling, stroke, and compression ratio, as they lack a correlated relationship with the target variable.

6.1. Initial Data Cleansing or Preparation

Before performing data cleansing on the dataset, it is necessary to check each column's unique value to identify duplicated, missing values, typos, and outliers that may lead to distorted outcomes. The decision to apply data cleansing techniques later in the analysis will be based on the results from different value checks.



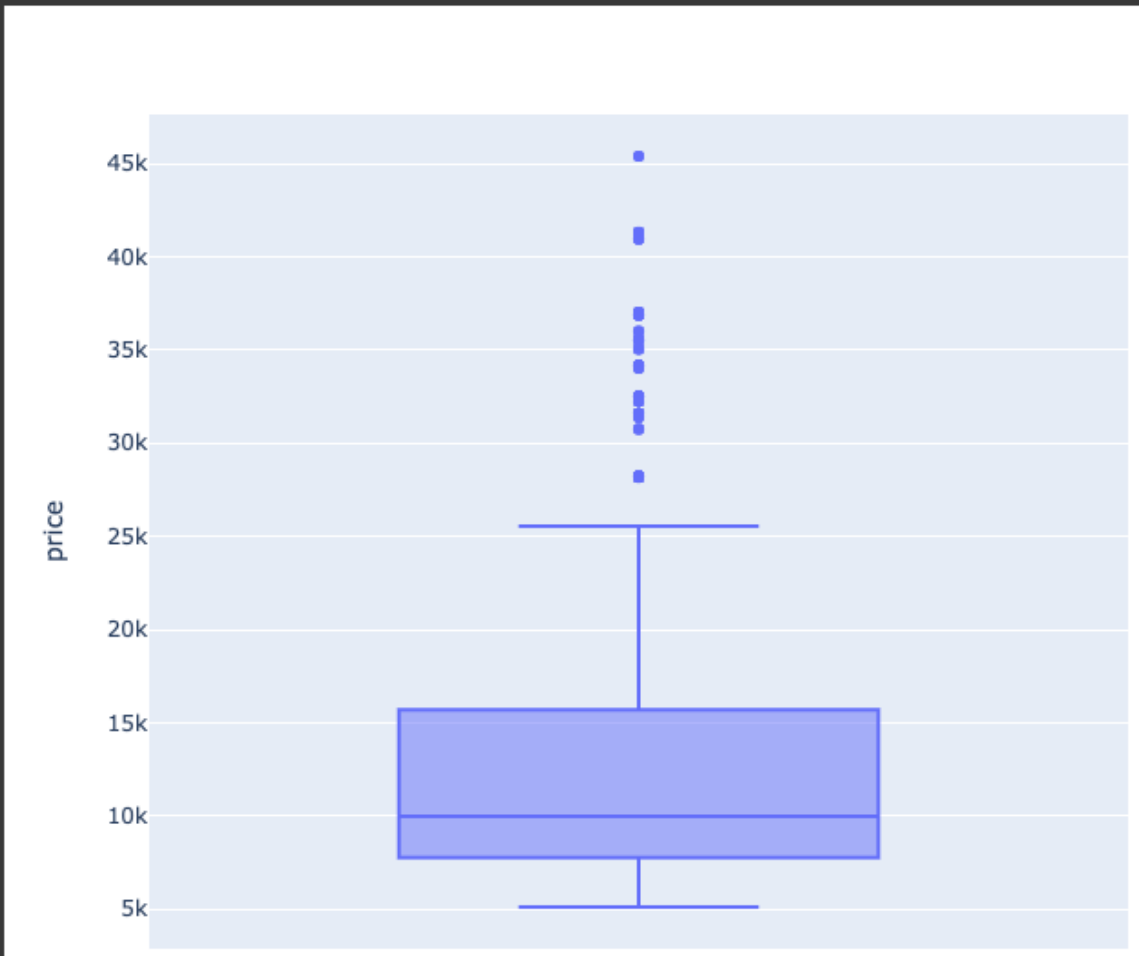
## Check null/missing values for each column

```
pd.DataFrame({'Columns':car.columns,'Missing Value T/F':car.columns.isna()})
#No missing values
```

| | Columns | Missing Value T/F |
|---|---|---|
| 0 | car_ID | False |
| 1 | symboling | False |
| 2 | CarName | False |
| 3 | fueltype | False |
| 4 | aspiration | False |
| 5 | doornumber | False |
| 6 | carbody | False |
| 7 | drivewheel | False |
| 8 | enginelocation | False |
| 9 | wheelbase | False |
| 10 | carlength | False |
| 11 | carwidth | False |
| 12 | carheight | False |
| 13 | curbweight | False |
| 14 | enginetype | False |
| 15 | cylindernumber | False |
| 16 | enginesize | False |
| 17 | fuelsystem | False |

## Box plot - Check outliers in our target variable (Price)

```
[ ] px.box(data_frame = car, y = 'price', width=700, height=600)
    #template = 'plotly_dark'
```



## 7.0. Data Dictionary

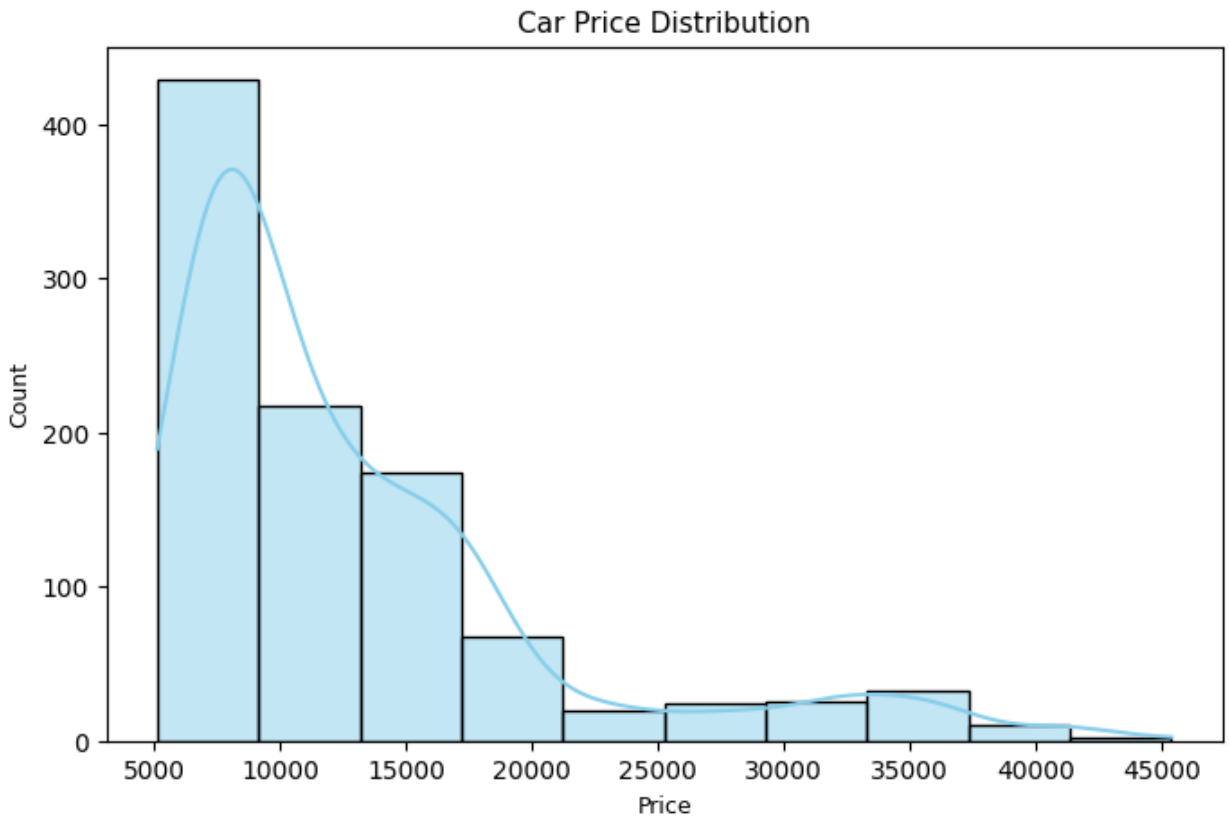
**Data Dictionary**

1. Car_ID > Unique id of each observation (*Interger*)
2. Symboling > Its assigned insurance risk rating, A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe (*Categorical*)
3. carCompany > Name of car company (*Categorical*)
4. fueltype > Car fuel type i.e gas or diesel (*Categorical*)
5. aspiration > Aspiration used in a car (*Categorical*)
6. doornumber > Number of doors in a car (*Categorical*)
7. carbody > body of car (*Categorical*)
8. drivewheel > type of drive wheel (*Categorical*)
9. enginelocation > Location of car engine (*Categorical*)
10. wheelbase > Wheelbase of car (*Numeric*)
11. carlength > Length of car (*Numeric*)
12. carwidth > Width of car (*Numeric*)
13. carheight > height of car (*Numeric*)
14. curbweight > The weight of a car without occupants or baggage (*Numeric*)
15. enginetype > Type of engine (*Categorical*)
16. cylindernumber > cylinder placed in the car (*Categorical*)
17. enginesize > Size of car (*Numeric*)
18. fuelsystem > Fuel system of car (*Categorical*)
19. boreratio > Boreratio of car (*Numeric*)
20. stroke > Stroke or volume inside the engine (*Numeric*)
21. compressionratio > compression ratio of car (*Numeric*)
22. horsepower > Horsepower (*Numeric*)
23. peakrpm > car peak rpm (*Numeric*)
24. citympg > Mileage in city (*Numeric*)
25. highwaympg > Mileage on highway (*Numeric*)
26. price (Dependent variable) > Price of car (*Numeric*)

# Data Exploration

## 8.0 Data Exploration Techniques

Data visualization will be implemented to ensure the dataset is adequately reprocessed and well-prepared for model training in the next step. Plotting diverse charts, such as scatter plots, bar charts, box plots, and distribution histograms, efficiently establishes a baseline understanding of variable distribution to discover the patterns between the predictor and target variables. Moreover, it also helps to uncover data trends and provides a quick overview of the variable's relationships within the dataset.

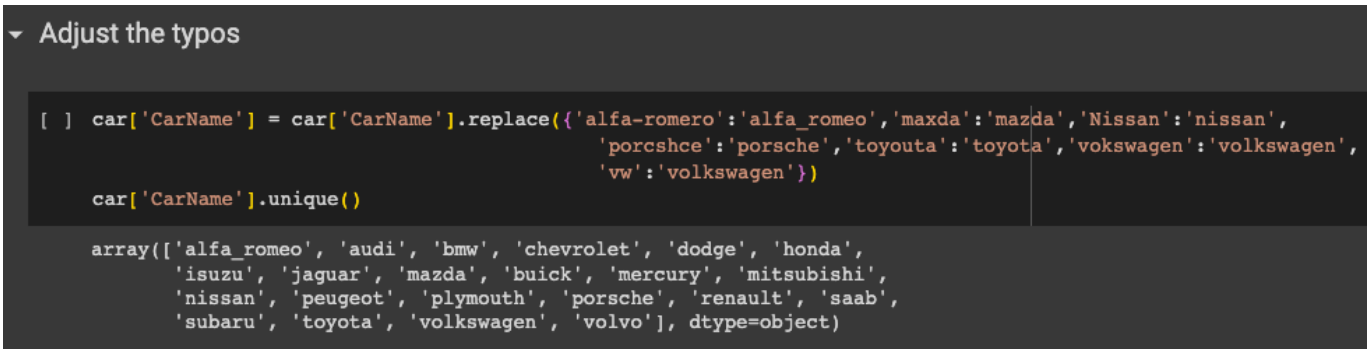



9.0 Data Cleansing

Data cleansing plays an essential role in mitigating inconsistent errors in data outcomes. Adjusting typos will be the primary task in this step because these incorrect data points yield meaningless values, potentially leading to abnormal or unreliable analytics results. The dataset contains several incorrect car brands, including alfa-romero, maxda, porcshce, toyouta, vokswagen, as well as other capitalized or abbreviated car brands with the same meanings, such as Nissan and vw. Adjustment codes will be applied to correct these typos to ensure data consistency for analytical purposes.

Moreover, no action is required to handle duplicated and missing values, as this dataset contains no repeated or null data points. This analysis retains the outliers to reflect real-life variability and preserve data accuracy. Although their removal could improve the data

interpretation and model's performance, it may also potentially lead to biased outcomes in this analysis.




10.0 Summary

Data exploration and cleansing are critical prerequisites to make sure the data points are accurate, reliable, and meaningful. They provide a robust groundwork for the subsequent analysis and model development by grasping the data variations. Furthermore, this process saves time to debug and minimize errors down the line because the raw data has been cleaned at this point. To develop well-trained prediction models, a clean dataset without missing values and typographical errors will remain the project's constant priority.

## Data Preparation and Feature Engineering

11.0. Data Preparation Needs

*1. Bootstrapping*

Bootstrapping ensures a mode's robustness by repeatedly resampling the data points to generate various samples within the dataset (Rosidi, 2023). This technique works particularly well with limited datasets because it increases the models' stability by introducing multiple bootstrap samples and enables them to train with additional observations. Besides, bootstrapping also mitigates the overfitting problem commonly encountered with a small dataset. Constructing

various training and validation data rows from the car dataset provides more data points to all machine learning models and guides them to pursue an optimal result effectively.

*2. Feature Engineering*

Feature engineering showcases the importance of having an extra derived variable, leading to a more straightforward interpretation and making the models more robust and representative of the dataset (Rosencranece, 2021). In this project, a new derived variable named 'CarClass' has been extracted from 'price' to establish three categories: Luxury, Mid-Range, and Affordable. These three different classes are determined based on the average price of each car brand. Vehicles ranging from $5,000 to $15,000 belong to the affordable group, those in the mid-range fall within the $15,000 to 25,000 range, and luxury vehicles are priced at $25,000 or above.

*3. Dummies Variables*

After reviewing the different unique values and data types in each column, the next step is to decide the most appropriate actions for refining the dataset. Numerous machine learning models exclusively require numeric columns and do not recognize categorical data as valid input (Brownlee, 2020). Thus, dummy variables are needed to convert categorical variables into numeric forms to meet model requirements. Another benefit of creating these variables is to enhance the flexibility in feature engineering, allowing more variations or ratios between car features and derived variables. This technique aids in exploring new data patterns or relationships that contribute to predicting car prices accurately.

*4. Data Splitting*

Moreover, data splitting also emphasizes on partitioning the car dataset into two subsets for training and testing purposes (Gillis, 2022). The selection of a 70% train and 30% test split

allows models to train on a more extensive training set. This concept is applicable and beneficial when tackling a relatively smaller dataset, as providing more training data points helps mitigate the risk of limited data for training. Besides, having a bigger training set introduces more diverse data and a more substantial regularization effect that ensures the reliability of different model outcomes.

11.1. Bootstrapping Process

Bootstrapping is a powerful and straightforward concept to extract the maximum details from the dataset. By bootstrapping the data rows from 200 to 1,000, this process allows plenty of models to perform training tasks without the limitations of insufficient data.

Bootstrapping

```
[ ] car = car.sample(1000,replace = True,ignore_index = True)
    car.head()
```

| | symboling | CarName | CarModel | fueltype | aspiration | doornumber | carbody | drivewheel | enginelocation | wheelbase |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | Volkswagen | Type | gas | std | four | sedan | fwd | front | 97.3 |
| 1 | 1 | Honda | Civic | gas | std | two | hatchback | fwd | front | 93.7 |
| 2 | 0 | Toyota | Corolla | gas | std | four | hatchback | fwd | front | 95.7 |
| 3 | 3 | Mazda | Glc | gas | std | two | hatchback | rwd | front | 95.3 |
| 4 | 2 | Chevrolet | Impala | gas | std | two | hatchback | fwd | front | 88.4 |

```
car.shape
```

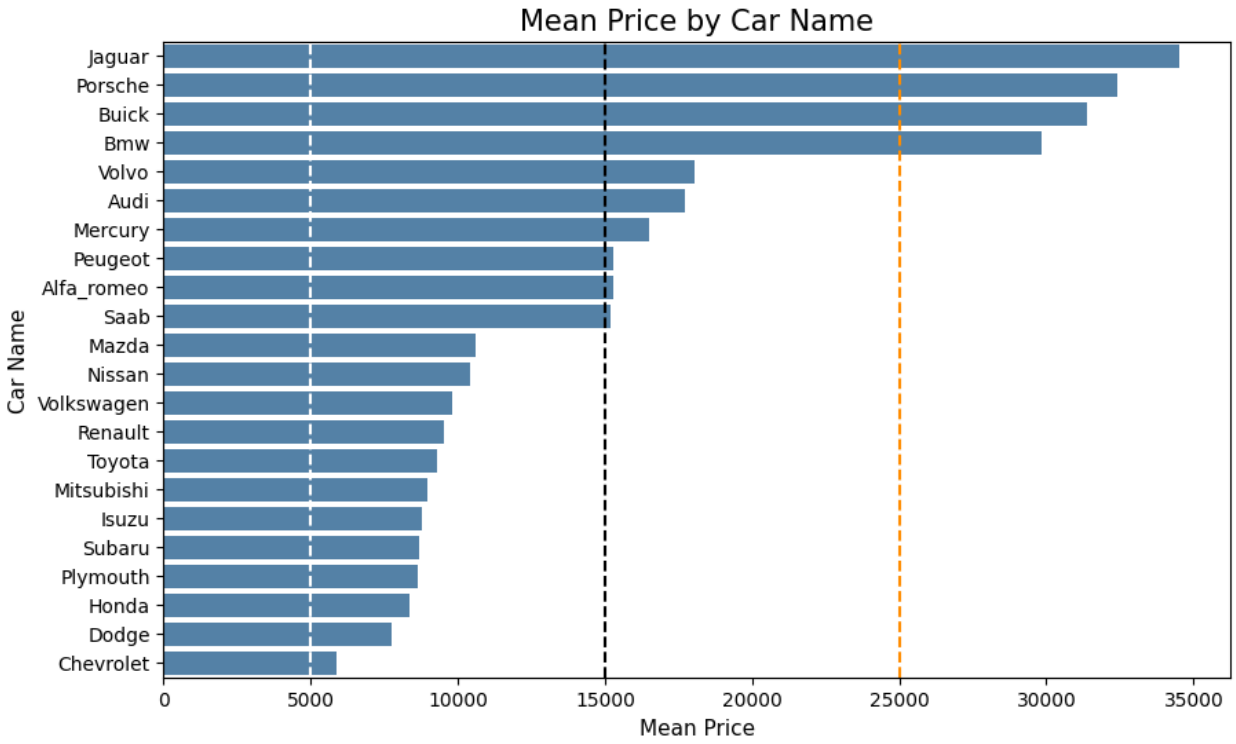
```
(1000, 26)
```

11.2. Feature Engineering Process

A horizontal bar chart has been created in Python to comprehend the price ranges of each car brand and highlight the appropriate cut-off price for each class. Following up by creating bins to classify car models into three categories and to understand their distribution through a histogram. Lastly, a comparison of the three classes with the total price of each car brand is presented side by side in Tableau, creating an interactive visualization interface and an enhanced

user experience.

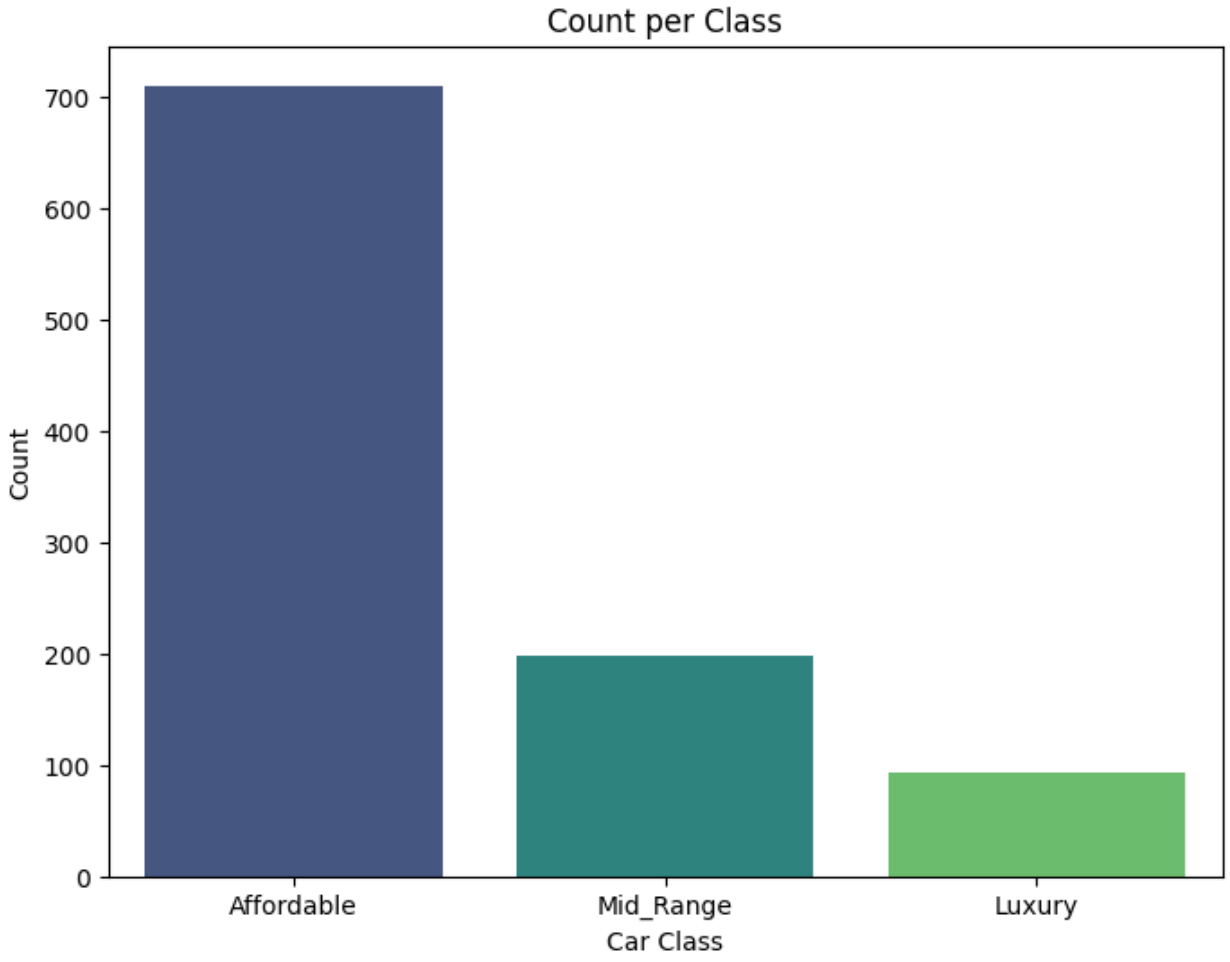
```
car_mean = car.groupby('CarName')['price'].mean().reset_index().sort_values(by='price', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=car_mean, x = 'price', y = 'CarName', color = 'steelblue',orient = 'h')
plt.xlabel('Mean Price', fontsize=11)
plt.ylabel('Car Name', fontsize=11)
plt.title('Mean Price by Car Name', fontsize=15)
plt.axvline(x = 5000, color = 'white', linestyle='dashed')
plt.axvline(x = 15000, color = 'black', linestyle='dashed')
plt.axvline(x = 25000, color = 'darkorange', linestyle='dashed')
plt.show()
```

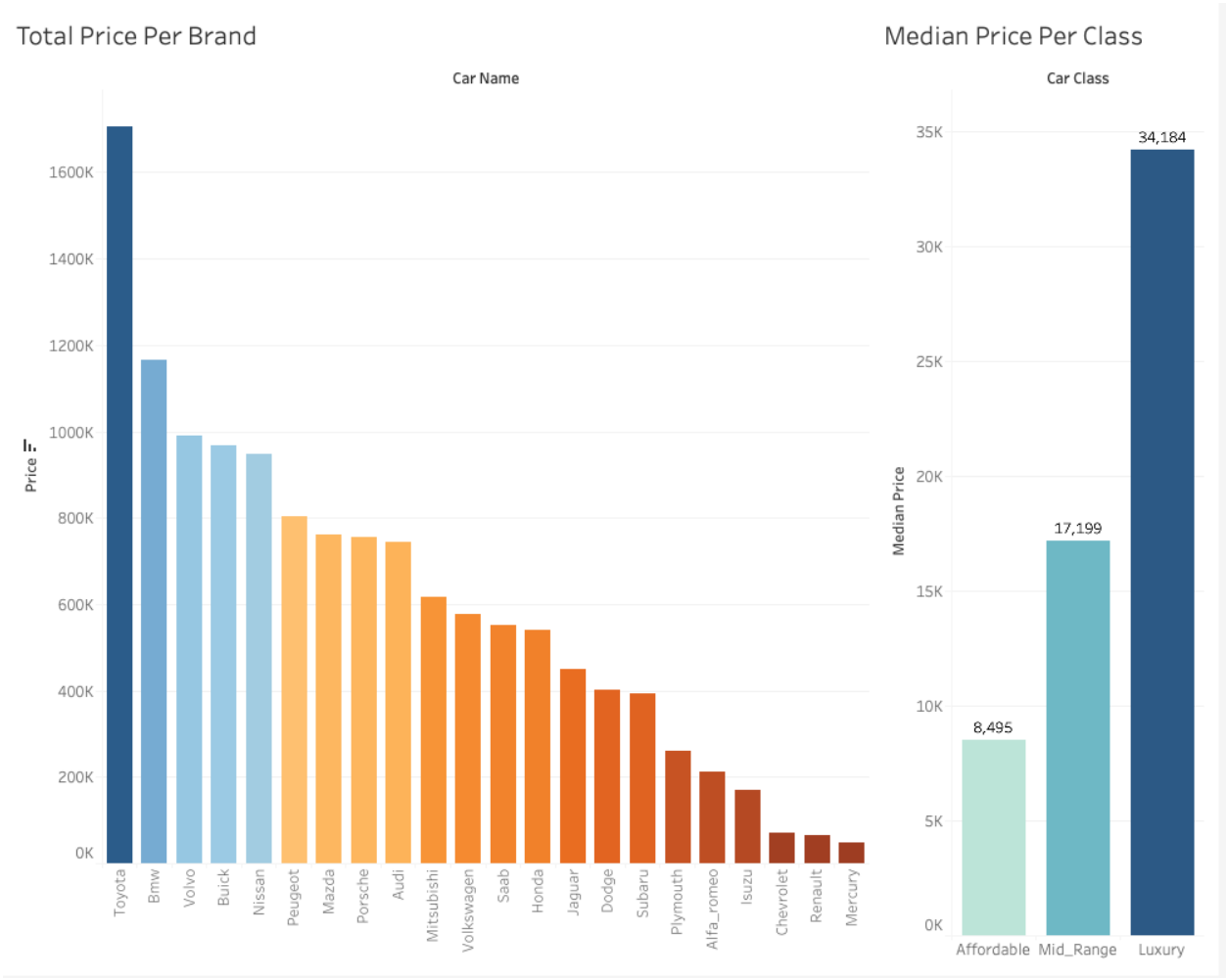



## Classify Car Models with Bins Variable

```
[ ] car_upperbound = car['price'].max()+1 #Avoid max value being left out
    classify_bins = [5000,15000,25000, car_upperbound]
    classify_labels = ['Affordable', 'Mid_Range', 'Luxury']
    car['CarClass'] = pd.cut(car['price'], bins = classify_bins, labels = classify_labels, right=False)
    car['CarClass'] = car['CarClass'].astype('object') #Change the data type from categorical to object
    car.head()
```

| citympg | highwaympg | price | CarClass |
|---|---|---|---|
| 27 | 34 | 8195.0 | Affordable |
| 30 | 34 | 7129.0 | Affordable |
| 30 | 37 | 7198.0 | Affordable |
| 16 | 23 | 15645.0 | Mid_Range |
| 47 | 53 | 5151.0 | Affordable |






## 12.0. Dummies Variables Process

In this step, the remaining categorical variables will be selected to create a predictor list, and the target variable price will be the outcome. Following up by combining new dummy variables with existing numeric car features to have a complete dataset ready for model training.

## Create Dummies Variables

```
predictors = ['fueltype','aspiration','carbody','drivewheel','enginelocation','enginetype',
              'cylindernumber','fuelsystem','CarClass']
outcome = 'price'
```

```
X = pd.get_dummies(car[predictors],drop_first = True)
y = car[outcome]
```

```
new_predictors = ['symboling','CarName','CarModel','doornumber','fueltype','aspiration','carbody','drivewheel',
                  'enginelocation','enginetype','cylindernumber','fuelsystem','CarClass','stroke','compressionratio']
car.drop(new_predictors, axis = 1, inplace = True) #Combine new dummies with existing numeric variables
car.head()
```

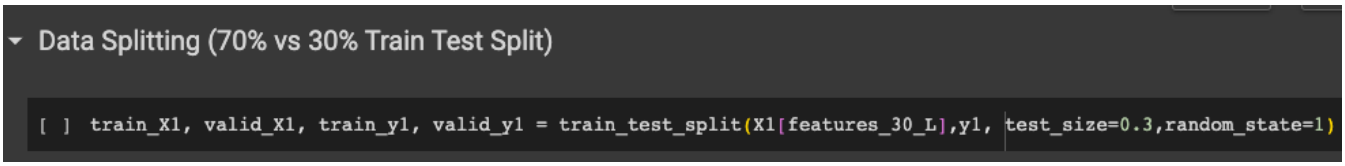
| | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | boreratio | horsepower | peakrpm | citympg | highwaympg | price | fueltype_gas | aspiration_turbo | carbody_hardtop | carbody_hatchback | carbody_sedan |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 97.3 | 171.7 | 65.5 | 55.7 | 2212 | 109 | 3.19 | 85 | 5250 | 27 | 34 | 8195.0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 93.7 | 150.0 | 64.0 | 52.6 | 1956 | 92 | 2.91 | 76 | 6000 | 30 | 34 | 7129.0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 95.7 | 166.3 | 64.4 | 52.8 | 2109 | 98 | 3.19 | 70 | 4800 | 30 | 37 | 7198.0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 95.3 | 169.0 | 65.7 | 49.6 | 2500 | 80 | 3.33 | 135 | 6000 | 16 | 23 | 15645.0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 88.4 | 141.1 | 60.3 | 53.2 | 1488 | 61 | 2.91 | 48 | 5100 | 47 | 53 | 5151.0 | 1 | 0 | 0 | 1 | 0 |

```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 42 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   wheelbase             1000 non-null   float64
 1   carlength             1000 non-null   float64
 2   carwidth              1000 non-null   float64
 3   carheight             1000 non-null   float64
 4   curbweight            1000 non-null   int64
 5   enginesize            1000 non-null   int64
 6   boreratio             1000 non-null   float64
 7   horsepower            1000 non-null   int64
 8   peakrpm               1000 non-null   int64
 9   citympg               1000 non-null   int64
 10  highwaympg            1000 non-null   int64
 11  price                 1000 non-null   float64
 12  fueltype_gas          1000 non-null   uint8
 13  aspiration_turbo      1000 non-null   uint8
 14  carbody_hardtop       1000 non-null   uint8
 15  carbody_hatchback     1000 non-null   uint8
 16  carbody_sedan         1000 non-null   uint8
 17  carbody_wagon         1000 non-null   uint8
 18  drivewheel_fwd        1000 non-null   uint8
 19  drivewheel_rwd        1000 non-null   uint8
 20  enginelocation_rear   1000 non-null   uint8
 21  enginetype_dohcv      1000 non-null   uint8
 22  enginetype_l          1000 non-null   uint8
 23  enginetype_ohc        1000 non-null   uint8
 24  enginetype_ohcf       1000 non-null   uint8
 25  enginetype_ohcv       1000 non-null   uint8
 26  enginetype_rotor      1000 non-null   uint8
 27  cylindernumber_five   1000 non-null   uint8
 28  cylindernumber_four   1000 non-null   uint8
 29  cylindernumber_six    1000 non-null   uint8
 30  cylindernumber_three  1000 non-null   uint8
 31  cylindernumber_twelve 1000 non-null   uint8
 32  cylindernumber_two    1000 non-null   uint8
 33  fuelsystem_2bbl       1000 non-null   uint8
```

12.1. Data Splitting Process

Every car feature, except for price, will be divided into 70% for training and 30% for validation sets. This division ensures that machine learning models have adequate data points to highlight essential information and increase the reliability of their model's performance.




## Model Exploration

13.0. Modeling Approach/Introduction

In this capstone project, four models will be deployed to encompass a diverse range of prediction outcomes and assess various model performances. These models include linear regression, decision trees, random forests, and gradient boosting. Each model possesses its own strengths and specialties in capturing essential insights from the data points, such as interpretability, predictive power, and robust outcomes. Furthermore, Recursive Feature Elimination (RFE) will also be introduced to each model to select exclusively 30 and 15 correlated variables from the car dataset, thereby enhancing the diversity of model performances.

14.0. Linear Regression

30 Selected Variables

To begin with, RFE will be implemented to select the top 30 variables. These features will then be used to fit the predictor data frame (X1) and outcome data series (y1) using the linear regression library.

```
[187] linear_m = LinearRegression()
      decision_t = DecisionTreeRegressor()
      random_f = RandomForestRegressor()
      gradient_b = GradientBoostingRegressor()
```

## 30 Selected Variables

```
[188] rfe_linear = RFE(estimator=linear_m, n_features_to_select=30)
      rfe_linear.fit(X1,y1)
```

RFE
estimator: LinearRegression
LinearRegression

```
[189] features_30_L = X1.columns[rfe_linear.support_]
      features_30_L

      Index(['carwidth', 'boreratio', 'highwaympg', 'fueltype_gas',
             'aspiration_turbo', 'carbody_hardtop', 'carbody_hatchback',
             'carbody_sedan', 'carbody_wagon', 'drivewheel_fwd', 'drivewheel_rwd',
             'enginelocation_rear', 'enginetype_dohcv', 'enginetype_l',
             'enginetype_ohc', 'enginetype_rotor', 'cylindernumber_five',
             'cylindernumber_four', 'cylindernumber_six', 'cylindernumber_three',
             'cylindernumber_twelve', 'cylindernumber_two', 'fuelsystem_2bbl',
             'fuelsystem_4bbl', 'fuelsystem_idi', 'fuelsystem_mfi',
             'fuelsystem_mpfi', 'fuelsystem_spfi', 'CarClass_Luxury',
             'CarClass_Mid_Range'],
            dtype='object')
```

After splitting the data into training and testing sets, data scaling will be exclusively applied to linear models to make sure all data points are on the same comparable scale. The choice is made based on the sensitivity of linear models between smaller and larger scale data. For instance, a vehicle with 1,000 horsepower would disproportionately impact the model due to its larger scale. Leaving the data unscaled before deploying linear models can result in biased coefficient estimates.

Conversely, the other three models are less sensitive to the scale of the car features. Their model algorithms are based on feature thresholds and complicated calculations without putting too much weight on the variable units. As a result, data scaling is less critical for these models due to their model theoretical nature than the linear regression model.

```
Data Splitting (70% vs 30% Train Test Split)

[190] train_X1, valid_X1, train_y1, valid_y1 = train_test_split(X1[features_30_L],y1, test_size=0.3,random_state=1)

Data Scaling - Standardization

[191] useful_num = list(train_X1.select_dtypes(include=['int64', 'float64']))
      scaler = StandardScaler()
      train_X1[useful_num] = scaler.fit_transform(train_X1[useful_num])
      valid_X1[useful_num] = scaler.fit_transform(valid_X1[useful_num])

[192] train_y1_np = np.array(train_y1).reshape(-1, 1)
      train_y1_minmax = scaler.fit_transform(train_y1_np)
      train_y1_scaled = pd.Series(train_y1_minmax.flatten())

[193] valid_y1_np = np.array(valid_y1).reshape(-1, 1)
      valid_y1_minmax = scaler.fit_transform(valid_y1_np)
      valid_y1_scaled = pd.Series(valid_y1_minmax.flatten())
```

After fitting the scaled training X1 and y1, the Ordinary Least Squares (OLS) regression result will be implemented to highlight various coefficients and p-values for all selected variables. The p-value plays an essential role in representing the statistical significance of a variable. A significance threshold of 0.05 will be the minimum acceptable range to ensure a high-quality model standard. To eliminate insignificant features, backward elimination will be applied to drop variables with the highest p-values one by one until all remaining variables fall within the acceptable range.

```
list_train_y1_scaled = list(train_y1_scaled)
X_train_ols = sm.add_constant(train_X1)
lm_ols = sm.OLS(list_train_y1_scaled,X_train_ols).fit()
print(lm_ols.summary()) #y = Price
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.966
Model:                            OLS   Adj. R-squared:                  0.965
Method:                 Least Squares   F-statistic:                     685.2
Date:                Tue, 15 Aug 2023   Prob (F-statistic):               0.00
Time:                        22:21:36   Log-Likelihood:                 192.40
No. Observations:                 700   AIC:                            -326.8
Df Residuals:                     671   BIC:                            -194.8
Df Model:                          28
Covariance Type:            nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                          0.3471      0.063      5.543      0.000       0.224       0.470
carwidth                       0.1404      0.019      7.402      0.000       0.103       0.178
boreratio                      0.1041      0.014      7.482      0.000       0.077       0.131
highwaympg                    -0.1140      0.019     -6.024      0.000      -0.151      -0.077
fueltype_gas                   0.1083      0.039      2.774      0.006       0.032       0.185
aspiration_turbo               0.1261      0.027      4.658      0.000       0.073       0.179
carbody_hardtop               -0.2356      0.061     -3.843      0.000      -0.356      -0.115
carbody_hatchback             -0.2872      0.049     -5.876      0.000      -0.383      -0.191
carbody_sedan                 -0.2133      0.047     -4.537      0.000      -0.306      -0.121
carbody_wagon                 -0.2615      0.051     -5.094      0.000      -0.362      -0.161
drivewheel_fwd                -0.1036      0.041     -2.548      0.011      -0.183      -0.024
drivewheel_rwd                -0.1356      0.047     -2.891      0.004      -0.228      -0.043
enginelocation_rear            0.4526      0.084      5.394      0.000       0.288       0.617
enginetype_dohcv              -0.5511      0.101     -5.430      0.000      -0.750      -0.352
enginetype_l                   0.1106      0.052      2.138      0.033       0.009       0.212
enginetype_ohc                 0.3224      0.028     11.393      0.000       0.267       0.378
enginetype_rotor              -0.2664      0.061     -4.360      0.000      -0.386      -0.146
cylindernumber_five           -0.9195      0.078    -11.739      0.000      -1.073      -0.766
cylindernumber_four           -0.8050      0.090     -8.962      0.000      -0.981      -0.629
cylindernumber_six            -0.2610      0.070     -3.747      0.000      -0.398      -0.124
cylindernumber_three          -0.2137      0.190     -1.123      0.262      -0.587       0.160
```

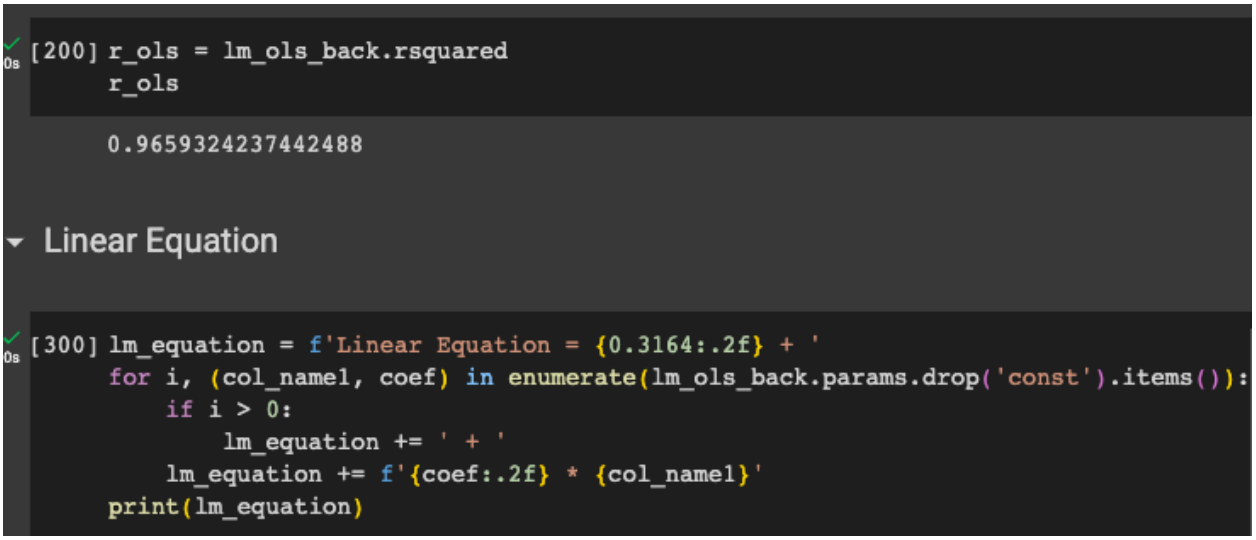
## Backward Elimination

- Create a loop to drop variables with the highest p-values one by one until all variables have p-values below 0.05

```
def backward_elimination(data, target, significance_level = 0.05):
    features = data.columns.tolist()
    while(len(features) > 0):
        features_with_constant = sm.add_constant(data[features])
        p_values_fit = sm.OLS(list(target), features_with_constant).fit().pvalues[1:]
        max_p_value = p_values_fit.max()
        if(max_p_value >= significance_level):
            excluded_feature = p_values_fit.idxmax()
            features.remove(excluded_feature)
        else:
            break

    return features

selected_features = backward_elimination(X_train_ols, train_y1_scaled, significance_level = 0.05)
print(selected_features)
```

```
['const', 'carwidth', 'boreratio', 'highwaympg', 'fueltype_gas', 'aspiration_turbo', 'carbody_hard
```

Following up by calculating the adjusted R-squared and demonstrating the linear equation to understand how the model generates price predictions based on varying coefficients. Moreover, the data needs to be unscaled to interpret regression summary and model comparison in later stages of the analysis. Lastly, the relationship between actual and predicted prices will be visualized to provide a clearer understanding of performance patterns and trends across different price ranges.



```
[200] r_ols = lm_ols_back.rsquared
      r_ols

      0.9659324237442488
```

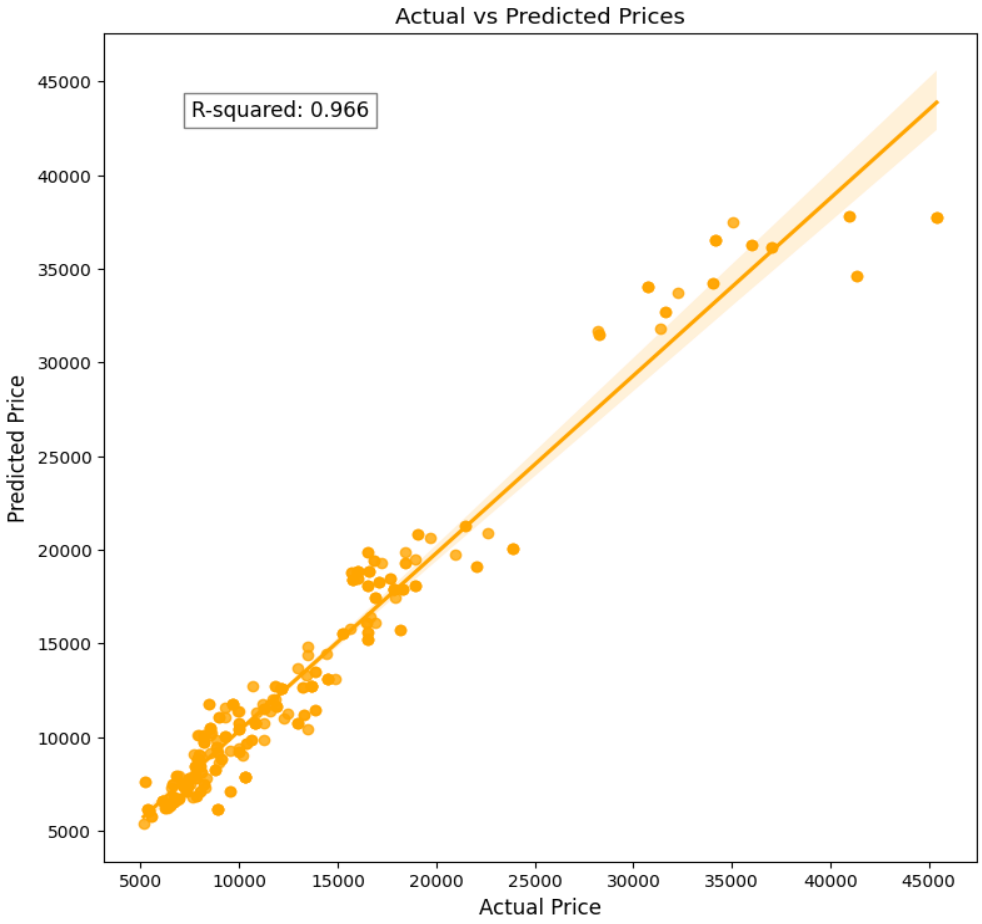
## Linear Equation

```
[300] lm_equation = f'Linear Equation = {0.3164:.2f} + '
      for i, (col_name1, coef) in enumerate(lm_ols_back.params.drop('const').items()):
          if i > 0:
              lm_equation += ' + '
          lm_equation += f'{coef:.2f} * {col_name1}'
      print(lm_equation)
```

Linear Equation = 0.32 + 0.15 * *carwidth* + 0.09 * *boreratio* + -0.12 * *highwaympg* + 0.09 * fueltype_gas + 0.12 * *aspiration_turbo* + -0.23 * *carbody_hardtop* + -0.28 * *carbody_hatchback* + -0.20 * *carbody_sedan* + -0.25 * *carbody_wagon* + -0.10 * *drivewheel_fwd* + -0.10 * *drivewheel_rwd* + 0.49 * *enginelocation_rear* + -0.53 * *enginetype_dohcv* + 0.29 * *enginetype_ohc* + -0.25 * *enginetype_rotor* + -0.87 * *cylindernumber_five* + -0.73 * *cylindernumber_four* + -0.24 * *cylindernumber_six* + -0.25 * *cylindernumber_two* + -0.07 * *fuelsystem_2bbl* + 0.27 * *fuelsystem_4bbl* + 0.23 * *fuelsystem_idi* + 0.06 * *fuelsystem_mpfi* + 1.97 * *CarClass_Luxury* + 0.59 * *CarClass_Mid_Range*

```python
#Unscale the number for better interpretation
y_train_pred_original = scaler.inverse_transform(y_train_pred.to_numpy().reshape(-1, 1))
y_train_pred_original

me_uc = (list(train_y1) - y_train_pred_original).mean()
mae_uc = mean_absolute_error(list(train_y1), y_train_pred_original)
rmse_uc = mean_squared_error(list(train_y1), y_train_pred_original, squared=False)
mpe_uc = ((list(train_y1) - y_train_pred_original) / list(train_y1)).mean()
mape_uc = (abs(list(train_y1) - y_train_pred_original) / list(train_y1)).mean()

print(f"Mean Error (ME): {me_uc:.3f}")
print(f'Root Mean Squared Error (RMSE): {rmse_uc:.3f}')
print(f'Mean Absolute Error (MAE): {mae_uc:.3f}')
print(f'Mean Percentage Error (MPE): {mpe_uc:.3f}')
print(f'Mean Absolute Percentage Error (MAPE): {mape_uc:.3f}')
```

```
Regression Summary - Scaled

Mean Error (ME): -0.000
Root Mean Squared Error (RMSE): 0.185
Mean Absolute Error (MAE): 0.130
Mean Percentage Error (MPE): -0.040
Mean Absolute Percentage Error (MAPE): 0.007

--------------------------------------------------
Regression Summary - Unscaled

Mean Error (ME): -355.952
Root Mean Squared Error (RMSE): 1535.472
Mean Absolute Error (MAE): 1125.645
Mean Percentage Error (MPE): -0.330
Mean Absolute Percentage Error (MAPE): 0.692
```



15 Selected Variables

As the modeling process closely mirrors the one with 30 variables, only the final outcome will be displayed to facilitate a more straightforward interpretation.

1. Select 15 variables with RFE

**15 Selected Variables**

- Linear Regression

```
[237] rfe_linear2 = RFE(estimator=linear_m, n_features_to_select=15)
      rfe_linear2.fit(X1,y1)
```

RFE
estimator: LinearRegression
LinearRegression

```
features_15_L = X1.columns[rfe_linear2.support_]
features_15_L
```

```
Index(['boreratio', 'enginelocation_rear', 'enginetype_dohcv', 'enginetype_l',
       'enginetype_ohc', 'enginetype_rotor', 'cylindernumber_five',
       'cylindernumber_four', 'cylindernumber_six', 'cylindernumber_three',
       'cylindernumber_two', 'fuelsystem_2bbl', 'fuelsystem_4bbl',
       'CarClass_Luxury', 'CarClass_Mid_Range'],
      dtype='object')
```

2. OLS Summary

```
X_train_ols_back2 = sm.add_constant(X_train_ols3[selected_features])
lm_ols_back2 = sm.OLS(list_train_y5_scaled,X_train_ols_back2).fit()
print(lm_ols_back2.summary())
```

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | y | R-squared: | 0.953 |
| Model: | OLS | Adj. R-squared: | 0.953 |
| Method: | Least Squares | F-statistic: | 1082. |
| Date: | Tue, 15 Aug 2023 | Prob (F-statistic): | 0.00 |
| Time: | 22:21:54 | Log-Likelihood: | 80.647 |
| No. Observations: | 700 | AIC: | -133.3 |
| Df Residuals: | 686 | BIC: | -69.58 |
| Df Model: | 13 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | 0.5906 | 0.074 | 7.958 | 0.000 | 0.445 | 0.736 |
| boreratio | 0.2125 | 0.012 | 18.462 | 0.000 | 0.190 | 0.235 |
| enginetype_dohcv | -0.9579 | 0.105 | -9.144 | 0.000 | -1.164 | -0.752 |
| enginetype_l | 0.3116 | 0.047 | 6.599 | 0.000 | 0.219 | 0.404 |
| enginetype_ohc | 0.3386 | 0.029 | 11.501 | 0.000 | 0.281 | 0.396 |
| enginetype_rotor | -0.4849 | 0.064 | -7.542 | 0.000 | -0.611 | -0.359 |
| cylindernumber_five | -0.9402 | 0.079 | -11.900 | 0.000 | -1.095 | -0.785 |
| cylindernumber_four | -1.2461 | 0.081 | -15.440 | 0.000 | -1.405 | -1.088 |
| cylindernumber_six | -0.5310 | 0.065 | -8.229 | 0.000 | -0.658 | -0.404 |
| cylindernumber_three | -1.3279 | 0.179 | -7.413 | 0.000 | -1.680 | -0.976 |
| cylindernumber_two | -0.4849 | 0.064 | -7.542 | 0.000 | -0.611 | -0.359 |
| fuelsystem_2bbl | -0.2696 | 0.021 | -13.024 | 0.000 | -0.310 | -0.229 |
| fuelsystem_4bbl | 0.2847 | 0.127 | 2.243 | 0.025 | 0.036 | 0.534 |
| CarClass_Luxury | 2.1298 | 0.056 | 37.928 | 0.000 | 2.020 | 2.240 |
| CarClass_Mid_Range | 0.6601 | 0.029 | 22.541 | 0.000 | 0.603 | 0.718 |

| | | | |
|---|---|---|---|
| Omnibus: | 86.356 | Durbin-Watson: | 1.964 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 177.614 |
| Skew: | 0.719 | Prob(JB): | 2.70e-39 |
| Kurtosis: | 5.005 | Cond. No. | 1.74e+16 |

3. Calculate Adjusted R-squared

```
r_ols2 = lm_ols_back2.rsquared
r_ols2
```

```
0.9534997376523686
```

4. Linear Equation

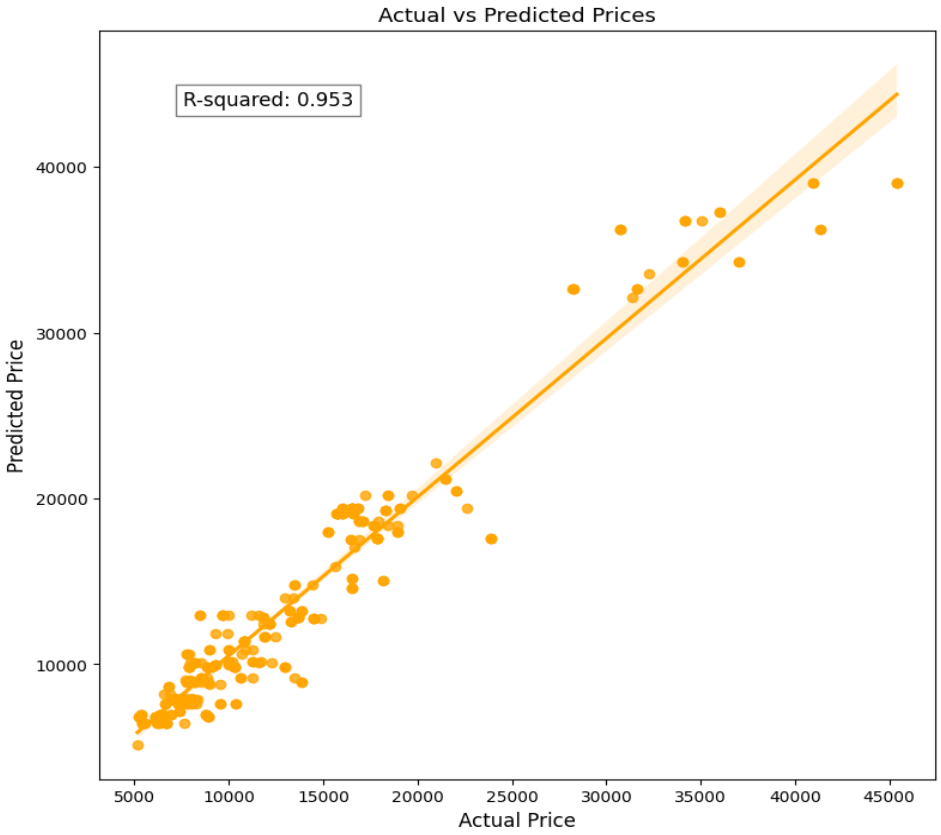
Linear Equation = 0.59 + 0.21 * boreratio + -0.96 * enginetype_dohcv + 0.31 * enginetype_l + 0.34 * enginetype_ohc + -0.48 * enginetype_rotor + -0.94 * cylindernumber_five + -1.25 * cylindernumber_four + -0.53 * cylindernumber_six + -1.33 * cylindernumber_three + -0.48 * cylindernumber_two + -0.27 * fuelsystem_2bbl + 0.28 * fuelsystem_4bbl + 2.13 * CarClass_Luxury + 0.66 * CarClass_Mid_Range

5. Linear Regression Summary (Scaled vs Unscaled)

```
print(f'Root Mean Squared Error (RMSE): {rmse2:.3f}')
print(f'Mean Absolute Error (MAE): {mae2:.3f}')
print(f'Mean Percentage Error (MPE): {mpe2:.3f}')
print(f'Mean Absolute Percentage Error (MAPE): {mape2:.3f}')
print('_'*50)
print('Regression Summary - Unscaled\n')

y_train_pred_original2 = scaler.inverse_transform(y_train_pred2.to_numpy().reshape(-1, 1))
y_train_pred_original2

me_uc2 = (list(train_y5) - y_train_pred_original2).mean()
mae_uc2 = mean_absolute_error(list(train_y5), y_train_pred_original2)
rmse_uc2 = mean_squared_error(list(train_y5), y_train_pred_original2, squared=False)
mpe_uc2 = ((list(train_y5) - y_train_pred_original2) / list(train_y5)).mean()
mape_uc2 = (abs(list(train_y5) - y_train_pred_original2) / list(train_y5)).mean()


print(f"Mean Error (ME): {me_uc2:.3f}")
print(f'Root Mean Squared Error (RMSE): {rmse_uc2:.3f}')
print(f'Mean Absolute Error (MAE): {mae_uc2:.3f}')
print(f'Mean Percentage Error (MPE): {mpe_uc2:.3f}')
print(f'Mean Absolute Percentage Error (MAPE): {mape_uc2:.3f}')
```

```
Regression Summary - Scaled

Mean Error (ME): -0.000
Root Mean Squared Error (RMSE): 0.216
Mean Absolute Error (MAE): 0.156
Mean Percentage Error (MPE): 0.076
Mean Absolute Percentage Error (MAPE): 0.147
__________________________________________________
Regression Summary - Unscaled

Mean Error (ME): -355.952
Root Mean Squared Error (RMSE): 1779.516
Mean Absolute Error (MAE): 1340.812
Mean Percentage Error (MPE): -0.330
Mean Absolute Percentage Error (MAPE): 0.682
```

6. Linear Curve (Actual vs Predicted Prices)




14.1. Decision Tree

30 Selected Variables

Beyond the 30 selected features and the data splitting into two sets, one major distinction between the decision tree and linear model is the grid search. A parameter grid will be applied, containing several possible hyperparameters. The algorithm will iteratively loop through various numbers to identify optimal parameters and estimators. For this specific model, the optimal parameters are determined to be 15 for max depth, 2 for minimum samples per leaf, and 5 for minimum samples per split. Moreover, 5 folds cross-validation will be used to train and evaluate the model 5 times. Visualizing the decision tree model helps to understand how the decision split is made based on different conditions. For example, the price prediction would be $17,355.27 if a

car's length is less than or equal to 188.3 and belongs to the mid-range class. Next, adjusted R-squared and regression statistics will be calculated for model comparison.

1. Select 30 Variables with RFE & Split the Data

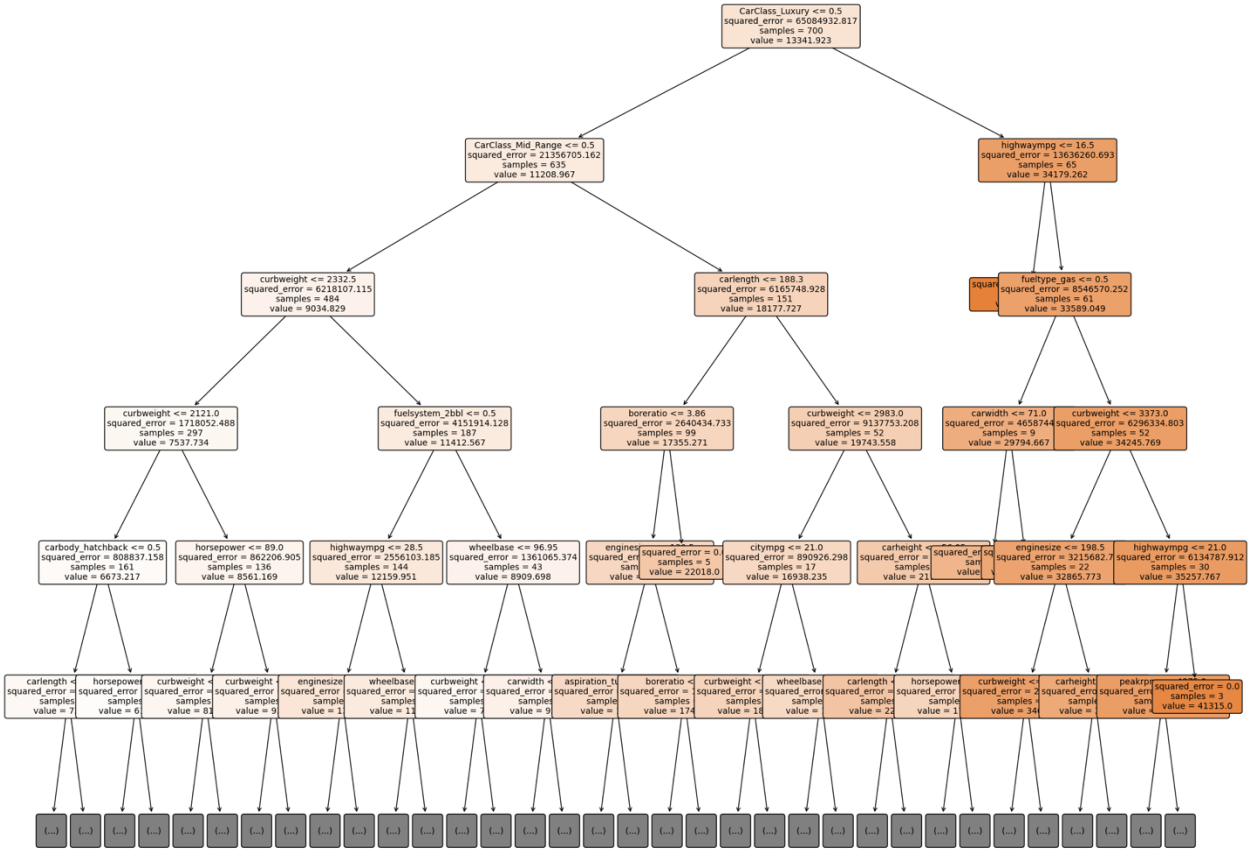
```
Decision Tree

[208] rfe_tree = RFE(estimator = decision_t, n_features_to_select=30)
      rfe_tree.fit(X1,y1)

      RFE
      estimator: DecisionTreeRegressor
      DecisionTreeRegressor

[209] features_30_D = X1.columns[rfe_tree.support_]
      features_30_D

      Index(['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
             'enginesize', 'boreratio', 'horsepower', 'peakrpm', 'citympg',
             'highwaympg', 'fueltype_gas', 'aspiration_turbo', 'carbody_hardtop',
             'carbody_hatchback', 'carbody_sedan', 'drivewheel_fwd',
             'drivewheel_rwd', 'enginetype_l', 'enginetype_ohc', 'enginetype_ohcf',
             'enginetype_rotor', 'cylindernumber_five', 'cylindernumber_four',
             'cylindernumber_two', 'fuelsystem_2bbl', 'fuelsystem_4bbl',
             'fuelsystem_mpfi', 'CarClass_Luxury', 'CarClass_Mid_Range'],
            dtype='object')

[210] train_X2, valid_X2, train_y2, valid_y2 = train_test_split(X1[features_30_D],y1, test_size = 0.3,random_state = 1)
```

2. Grid Search with Optimal Hyperparameters

```
param_grid_dt = {
    'max_depth': [15],
    'min_samples_split': [5],
    'min_samples_leaf': [2]
} #Optimal Hyperparameter
gridsearch_dt = GridSearchCV(DecisionTreeRegressor(random_state = 1),param_grid_dt, cv = 5)
gridsearch_dt.fit(train_X2,train_y2)

GridSearchCV
estimator: DecisionTreeRegressor
DecisionTreeRegressor

[212] best_params_dt = gridsearch_dt.best_params_
      best_estimator_dt = gridsearch_dt.best_estimator_

      print(best_params_dt)
      print(best_estimator_dt)

      {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 5}
      DecisionTreeRegressor(max_depth=15, min_samples_leaf=2, min_samples_split=5,
                            random_state=1)
```

3. Visualize Decision Tree Plot

```
plt.figure(figsize=(25, 20)) #To avoid the tree nodes crammed together, set the max_depth to 5 for better visualization
plot_tree(best_estimator_dt, feature_names = valid_X2.columns, filled = True, rounded = True,
          max_depth = 5, fontsize = 10)
plt.show()
```



4. Calculate Adjusted R-squared and Regression Summary

```
dt_summary = r2_score(valid_y2, best_estimator_dt.predict(valid_X2))
dt_summary
```

```
0.9977437553278784
```

```
[215] regressionSummary(valid_y2,best_estimator_dt.predict(valid_X2))
```

```
Regression statistics

                    Mean Error (ME) : -2.1364
     Root Mean Squared Error (RMSE) : 388.8609
          Mean Absolute Error (MAE) : 123.8295
        Mean Percentage Error (MPE) : -0.1010
Mean Absolute Percentage Error (MAPE) : 0.8291
```

15 Select Variables

1. Select 15 variables with RFE & Split the Data

Decision Tree

```
rfe_tree2 = RFE(estimator = decision_t, n_features_to_select=15)
rfe_tree2.fit(X1,y1)
```

```
RFE
estimator: DecisionTreeRegressor
DecisionTreeRegressor
```

```
[258] features_15_D = X1.columns[rfe_tree2.support_]
features_15_D
```

```
Index(['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
       'enginesize', 'boreratio', 'horsepower', 'peakrpm', 'highwaympg',
       'aspiration_turbo', 'carbody_hatchback', 'fuelsystem_2bbl',
       'CarClass_Luxury', 'CarClass_Mid_Range'],
      dtype='object')
```

```
[259] train_X6, valid_X6, train_y6, valid_y6 = train_test_split(X1[features_15_D],y1, test_size = 0.3,random_state = 1)
```

```
[260] gridsearch_dt2 = GridSearchCV(DecisionTreeRegressor(random_state = 1),param_grid_dt, cv = 5)
gridsearch_dt2.fit(train_X6,train_y6)
```

```
GridSearchCV
estimator: DecisionTreeRegressor
DecisionTreeRegressor
```

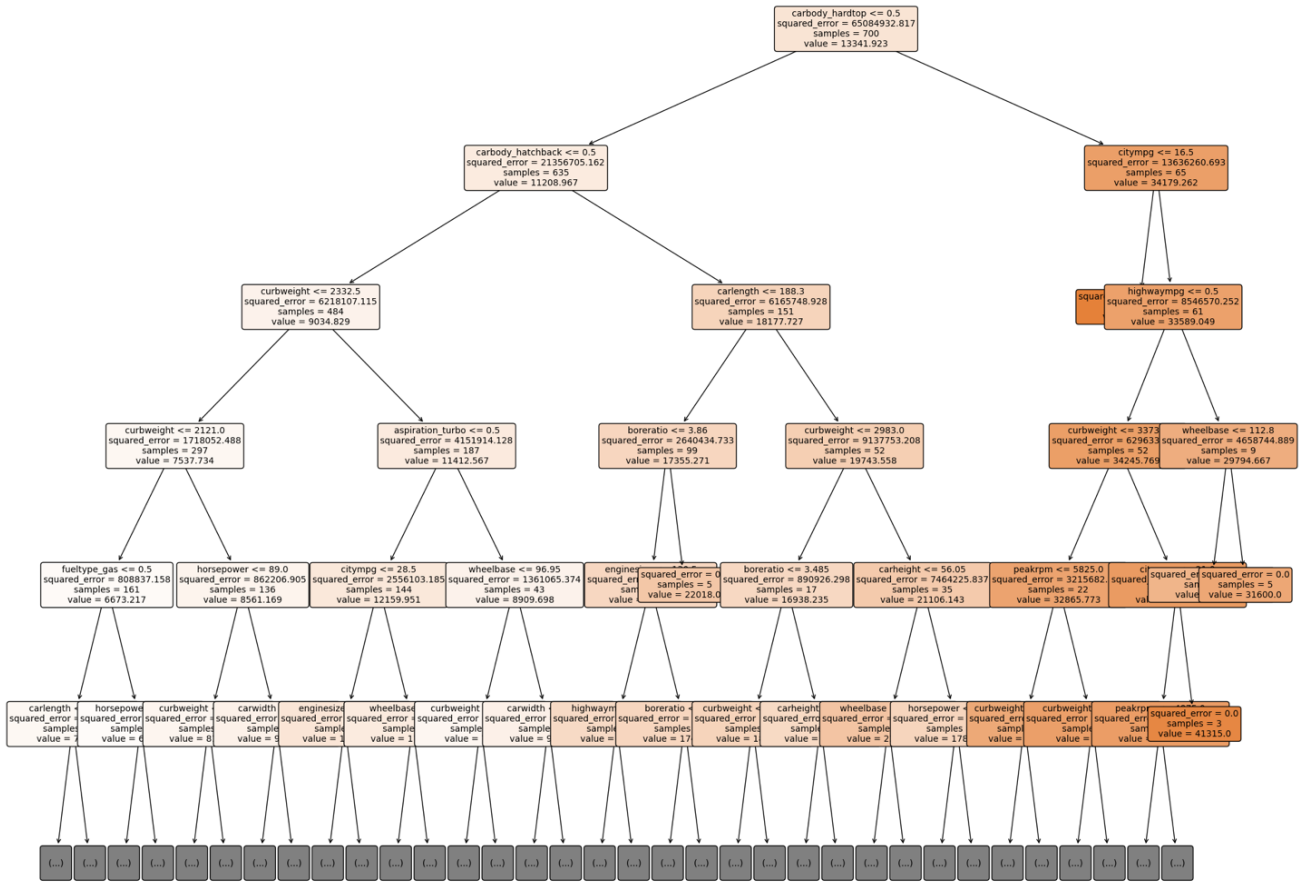
2. Grid Search with Optimal Hyperparameters

```
best_params_dt2 = gridsearch_dt2.best_params_
best_estimator_dt2 = gridsearch_dt2.best_estimator_

print(best_params_dt2)
print(best_estimator_dt2)
```

```
{'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 5}
DecisionTreeRegressor(max_depth=15, min_samples_leaf=2, min_samples_split=5,
                      random_state=1)
```

```
[262] plt.figure(figsize=(25, 20))
      plot_tree(best_estimator_dt2, feature_names = valid_X2.columns, filled = True, rounded = True,
                max_depth = 5, fontsize = 10)
      plt.show()
```

3. Visualize Decision Tree Plot



4. Calculate the Adjusted R-squared & Regression Summary

```
[263] dt_summary2 = r2_score(valid_y6, best_estimator_dt2.predict(valid_X6))
      dt_summary2

      0.9977497073492861

[264] regressionSummary(valid_y6, best_estimator_dt2.predict(valid_X6))

      Regression statistics

                          Mean Error (ME) : -3.8452
           Root Mean Squared Error (RMSE) : 388.3477
                Mean Absolute Error (MAE) : 123.8508
              Mean Percentage Error (MPE) : -0.1243
      Mean Absolute Percentage Error (MAPE) : 0.8295
```

15.0. Random Forest

30 Select Variables

Random forest is an advanced multiple decision tree approach to mitigate the impact of overfitting and improve accuracy in price prediction. With 30 selected features using RFE, a new parameter grid will be employed to fine-tune key hyperparameters on training and validation sets, including n_estimators, max_depth, min_samples_split, min_samples_leaf, and bootstrap. Similarly, this dataset will be split into 5 subsets using 5-fold cross-validation to enhance the model's reliability. The mean squared error will also serve as a parameter to fit the statistical requirements better. In this step, the optimal parameters are determined to be 100 for the number of estimators, 15 for max depth, 1 for minimum samples per leaf, 2 for minimum samples per split, and 'False' for bootstrap. The interpretation of a random forest resembles that of a decision tree. This will be followed by evaluating the corresponding adjusted R-squared and regression metrics to have a baseline understanding of the model's performance.

1. Select 30 Variables with RFE & Split the Data

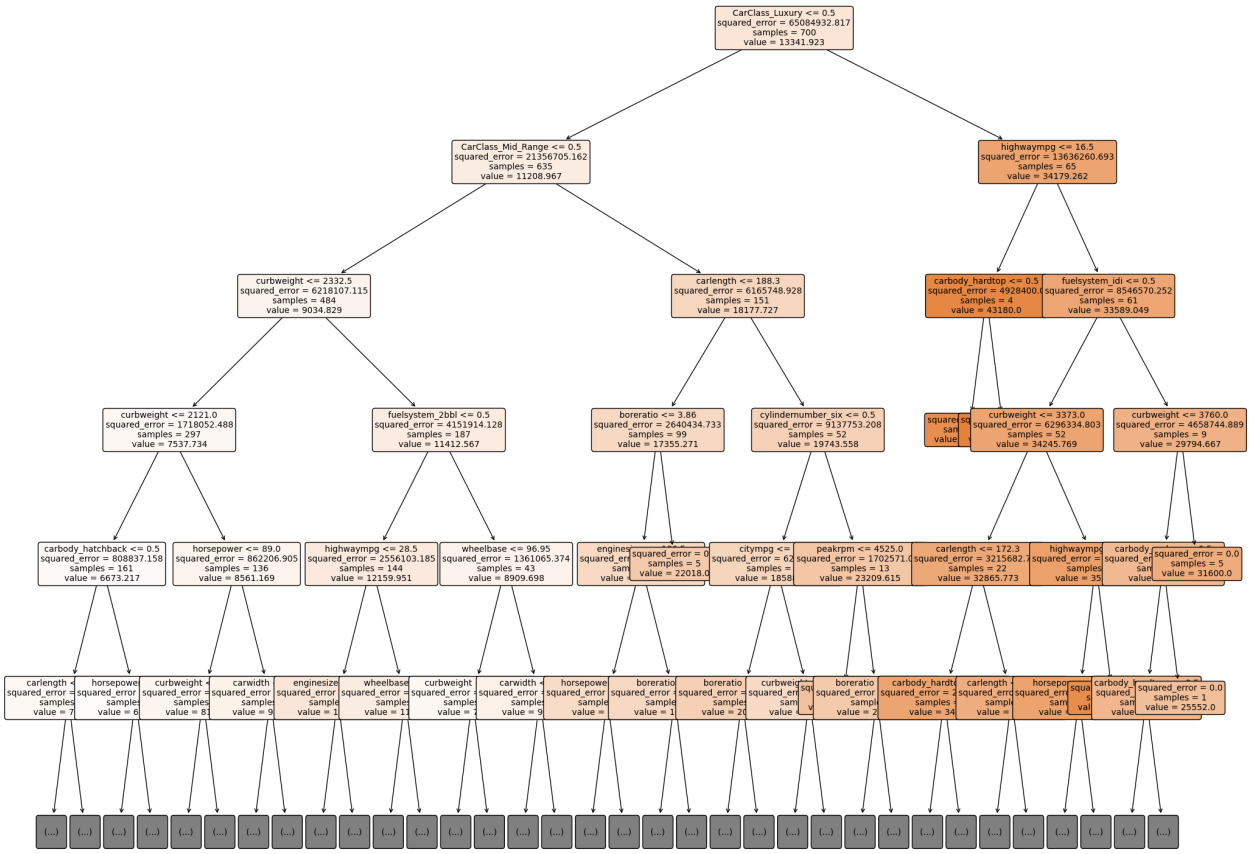
```
Random Forest Model

[218] rfe_rf = RFE(estimator = random_f, n_features_to_select=30)
      rfe_rf.fit(X1,y1)

RFE
estimator: RandomForestRegressor
RandomForestRegressor

[219] features_30_F = X1.columns[rfe_rf.support_]
      features_30_F

Index(['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
       'enginesize', 'boreratio', 'horsepower', 'peakrpm', 'citympg',
       'highwaympg', 'fueltype_gas', 'aspiration_turbo', 'carbody_hardtop',
       'carbody_hatchback', 'carbody_sedan', 'carbody_wagon', 'drivewheel_fwd',
       'enginelocation_rear', 'enginetype_l', 'enginetype_ohc',
       'enginetype_ohcf', 'cylindernumber_five', 'cylindernumber_four',
       'cylindernumber_six', 'fuelsystem_2bbl', 'fuelsystem_idi',
       'fuelsystem_mpfi', 'CarClass_Luxury', 'CarClass_Mid_Range'],
      dtype='object')

[220] train_X3, valid_X3, train_y3, valid_y3 = train_test_split(X1[features_30_F],y1, test_size = 0.3,random_state = 1)
```

2. Grid Search with Optimal Hyperparameters

```
param_grid_rf = {
    'n_estimators': [100],
    'max_depth': [15],
    'min_samples_split': [2],
    'min_samples_leaf': [1],
    'bootstrap': [False]
} #Fine tuned hyperparameter

gridsearch_rf = GridSearchCV(RandomForestRegressor(random_state = 1), param_grid_rf, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
gridsearch_rf.fit(train_X3, train_y3)

GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor

[222] best_params_rf = gridsearch_rf.best_params_
      best_estimator_rf = gridsearch_rf.best_estimator_

      print(best_params_rf)
      print(best_estimator_rf)

      {'bootstrap': False, 'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
      RandomForestRegressor(bootstrap=False, max_depth=15, random_state=1)
```

3. Visualize Random Forest Plot

```
plt.figure(figsize=(25, 20))
plot_tree(best_estimator_rf.estimators_[0], feature_names=valid_X3.columns, filled = True, rounded = True,
          max_depth = 5, fontsize = 10)
plt.show()
```




---

4. <u>Calculate Adjusted R-squared and Regression Summary</u>

```
[224] rf_summary = r2_score(valid_y3, best_estimator_rf.predict(valid_X3))
      rf_summary

      0.9991707282835413

[225] regressionSummary(valid_y3, best_estimator_rf.predict(valid_X3))

      Regression statistics

                          Mean Error (ME) : -10.8815
               Root Mean Squared Error (RMSE) : 235.7489
                    Mean Absolute Error (MAE) : 57.9902
                  Mean Percentage Error (MPE) : -0.1313
      Mean Absolute Percentage Error (MAPE) : 0.5364
```

## 15 Selected Variables

### 1. Select 15 Variables with RFE & Split the Data

Random Forest

```
[267] rfe_rf2 = RFE(estimator = random_f, n_features_to_select=15)
      rfe_rf2.fit(X1,y1)
```

RFE
estimator: RandomForestRegressor
RandomForestRegressor

```
[268] features_15_F = X1.columns[rfe_rf2.support_]
      features_15_F

      Index(['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight',
             'enginesize', 'boreratio', 'horsepower', 'peakrpm', 'citympg',
             'highwaympg', 'aspiration_turbo', 'carbody_hatchback',
             'CarClass_Luxury', 'CarClass_Mid_Range'],
            dtype='object')

[269] train_X7, valid_X7, train_y7, valid_y7 = train_test_split(X1[features_15_F],y1, test_size = 0.3,random_state = 1)
```

### 2. Grid Search with Optimal Hyperparameters

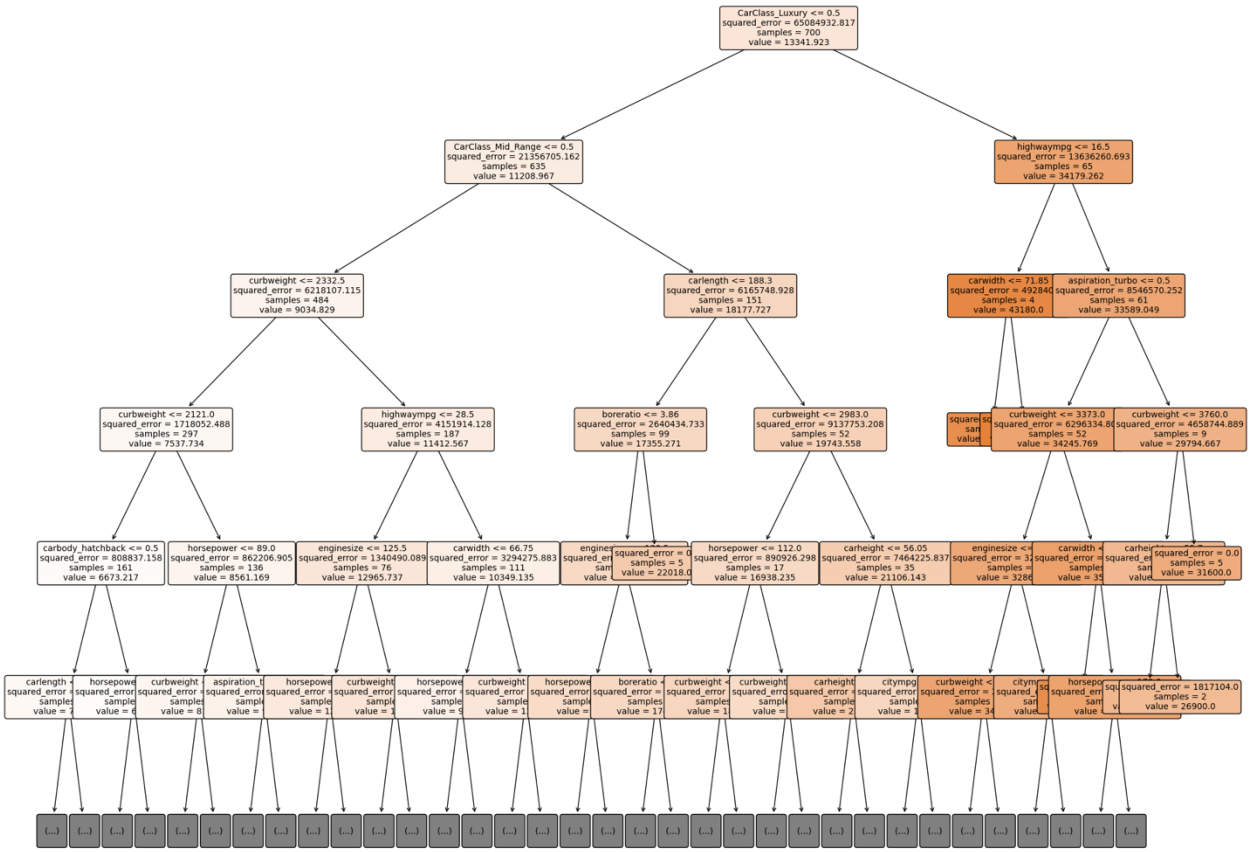
```
param_grid_rf2 = {
    'n_estimators': [50],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [2],
    'bootstrap': [False]
}

gridsearch_rf2 = GridSearchCV(RandomForestRegressor(random_state = 1), param_grid_rf2, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
gridsearch_rf2.fit(train_X7, train_y7)
```

GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor

```
[271] best_params_rf2 = gridsearch_rf2.best_params_
      best_estimator_rf2 = gridsearch_rf2.best_estimator_

      print(best_params_rf2)
      print(best_estimator_rf2)

      {'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
      RandomForestRegressor(bootstrap=False, min_samples_leaf=2, n_estimators=50,
                            random_state=1)
```

### 3. Visualize Random Forest Plot

```
plt.figure(figsize=(25, 20))
plot_tree(best_estimator_rf2.estimators_[0], feature_names = valid_X7.columns, filled = True, rounded = True,
          max_depth = 5, fontsize = 10)
plt.show()
```




---

4. Calculate Adjusted R-squared and Regression Summary

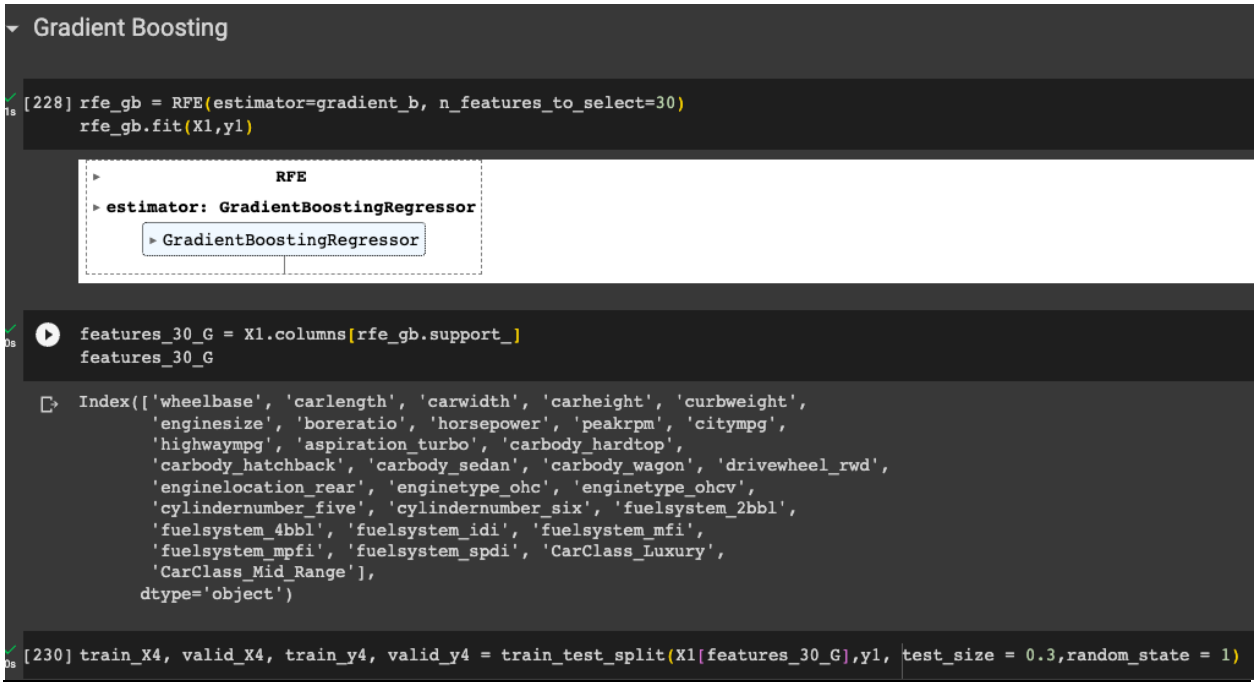
```
[273] rf_summary2 = r2_score(valid_y7, best_estimator_rf2.predict(valid_X7))
      rf_summary2

      0.9989031783084937

      regressionSummary(valid_y7, best_estimator_rf2.predict(valid_X7))

      Regression statistics

                        Mean Error (ME) : -7.4309
             Root Mean Squared Error (RMSE) : 271.1248
                  Mean Absolute Error (MAE) : 82.6473
                Mean Percentage Error (MPE) : -0.1036
      Mean Absolute Percentage Error (MAPE) : 0.6954
```

## 15.1. Gradient Boosting

### 30 Selected Variables

Gradient boosting refines its prediction accuracy and minimizes errors by iteratively combining weak learners together. Unlike decision trees and random forests, the grid search parameters include loss, max_depth, min_samples_leaf, min_samples_split, and n_estimators. Cross-validation and scoring will remain consistent with random forest, which is 5-fold validation and the mean squared error as an evaluation metric. For this model, the optimal parameters are identified as 'huber' for the loss function, 5 for max depth, 2 for minimum samples per leaf, 6 for minimum samples per split, and 75 for the number of estimators.

1. Select 30 Variables with RFE & Split the Data



2. Grid Search with Optimal Hyperparameters

```
[231] param_grid_gb = {
        'loss': ['huber'],
        'max_depth': [5],
        'min_samples_leaf': [2],
        'min_samples_split': [6],
        'n_estimators': [75],
    } #Optimal hyperparameter

    gridsearch_gb = GridSearchCV(GradientBoostingRegressor(random_state = 1), param_grid_gb, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
    gridsearch_gb.fit(train_X4, train_y4)
```

GridSearchCV
estimator: GradientBoostingRegressor
GradientBoostingRegressor

```
best_params_gb = gridsearch_gb.best_params_
best_estimator_gb = gridsearch_gb.best_estimator_

print(best_params_gb)
print(best_estimator_gb)
```

```
{'loss': 'huber', 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 6, 'n_estimators': 75}
GradientBoostingRegressor(loss='huber', max_depth=5, min_samples_leaf=2,
                          min_samples_split=6, n_estimators=75, random_state=1)
```

3. Calculate Adjusted R-squared and Regression Summary

```
[233] gb_summary = r2_score(valid_y4, best_estimator_gb.predict(valid_X4))
      gb_summary
```

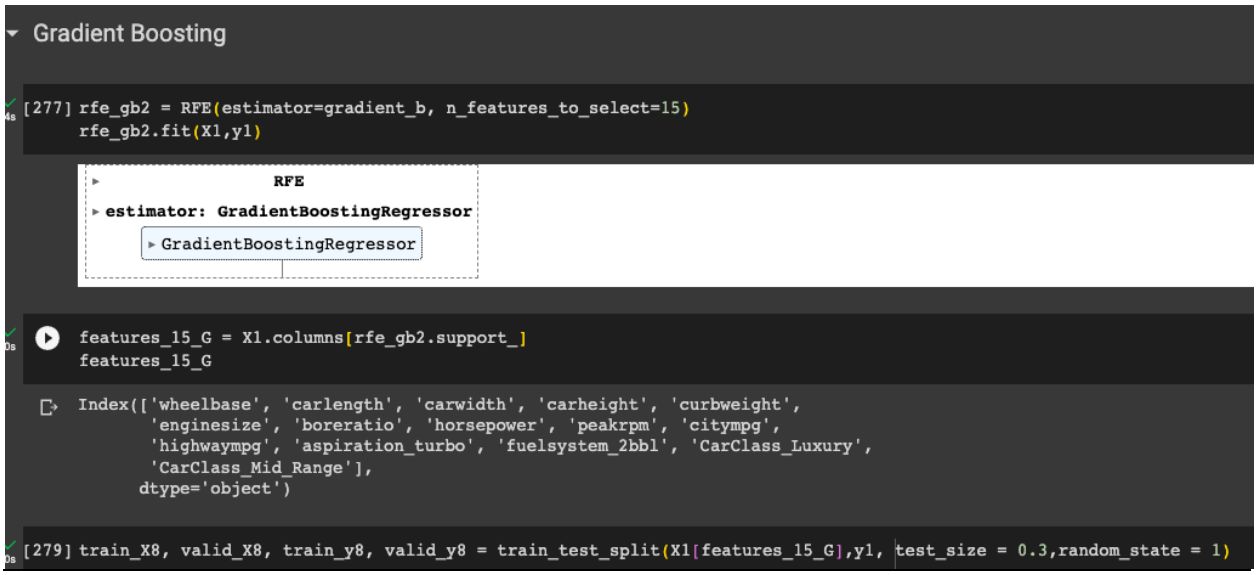
```
0.996295571127603
```

```
regressionSummary(valid_y4, best_estimator_gb.predict(valid_X4))
```

```
Regression statistics

                    Mean Error (ME) : 36.3114
         Root Mean Squared Error (RMSE) : 498.2667
              Mean Absolute Error (MAE) : 232.6177
            Mean Percentage Error (MPE) : -0.2370
Mean Absolute Percentage Error (MAPE) : 1.9435
```

## 15 Selected Variables

1. Select 15 Variables with RFE & Split the Data




2. Grid Search with Optimal Hyperparameters

3. Calculate Adjusted R-squared and Regression Summary

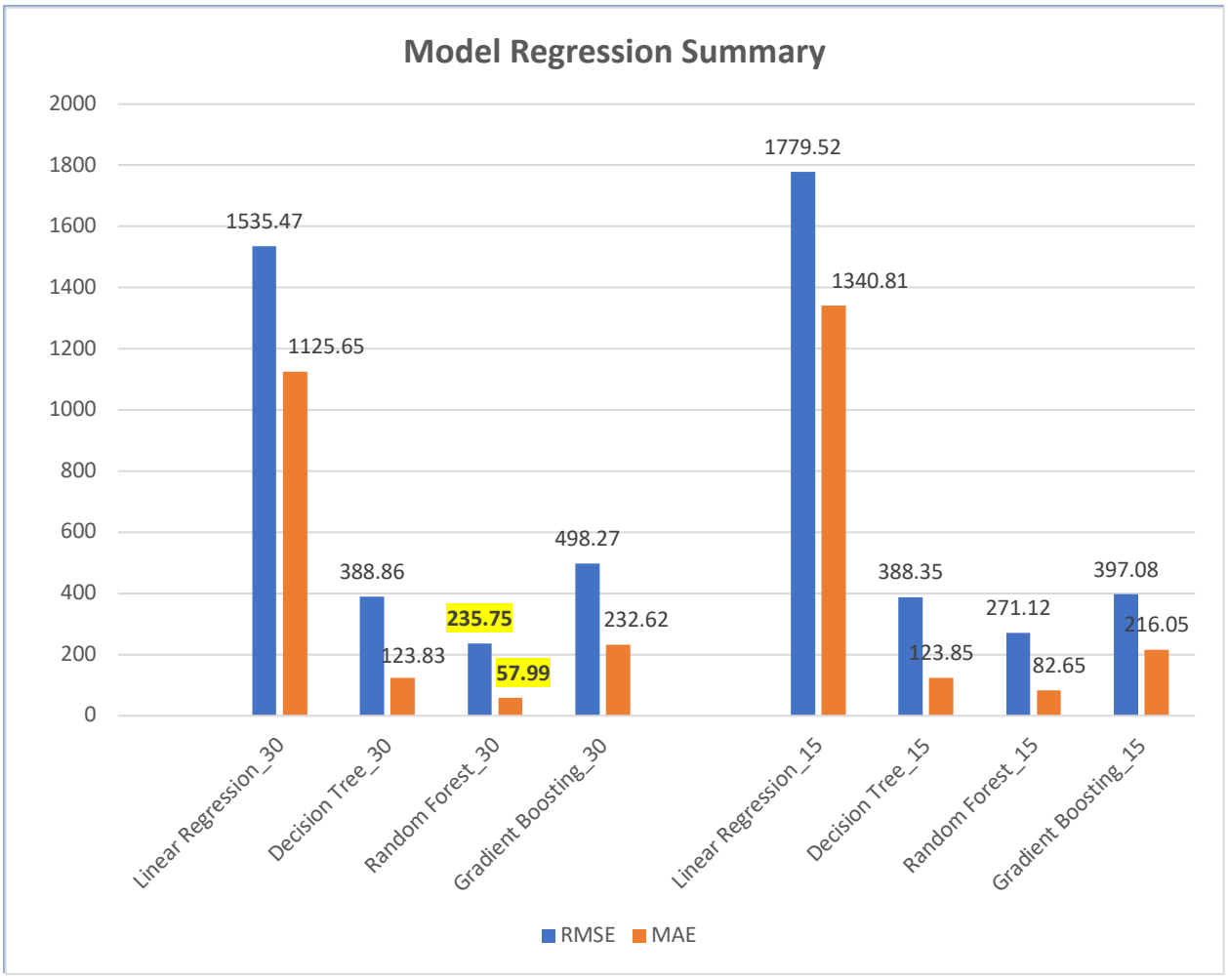
```
[282] gb_summary2 = r2_score(valid_y8, best_estimator_gb2.predict(valid_X8))
      gb_summary2

      0.9976473350109816

[283] regressionSummary(valid_y8, best_estimator_gb2.predict(valid_X8))

      Regression statistics

                          Mean Error (ME) : 22.4322
       Root Mean Squared Error (RMSE) : 397.0830
            Mean Absolute Error (MAE) : 216.0547
          Mean Percentage Error (MPE) : -0.1383
      Mean Absolute Percentage Error (MAPE) : 1.9999
```

16.0. Model Comparison

Since this project aims to focus exclusively on Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), the model comparison will be centered on identifying the model that achieved the lowest error. RMSE represents the average squared differences between the predicted and actual car prices from this model, while MAE highlights the average discrepancy between predicted and actual car prices. Both statistical metrics imply that lower values are preferred and signify outstanding performance. Among the models with both 30 and 15 variables, linear regression models exhibited the highest RMSE and MAE, followed by gradient boosting. The decision tree is the second-best model, with errors slightly higher than those of random forest. In this summary, the random forest model with 30 selected features performs best performance on car price predictions, featuring the lowest RMSE of 235.75 and MAE of 57.99.

```
reg_df = pd.DataFrame({'Model': model_list,'RMSE': rmse_list,'MAE':mae_list})
reg_df['RMSE'] = reg_df['RMSE'].apply(lambda x: '{:.2f}'.format(float(x)) if x != '-' else '-')
reg_df['MAE'] = reg_df['MAE'].apply(lambda x: '{:.2f}'.format(float(x)) if x != '-' else '-')
reg_df
```

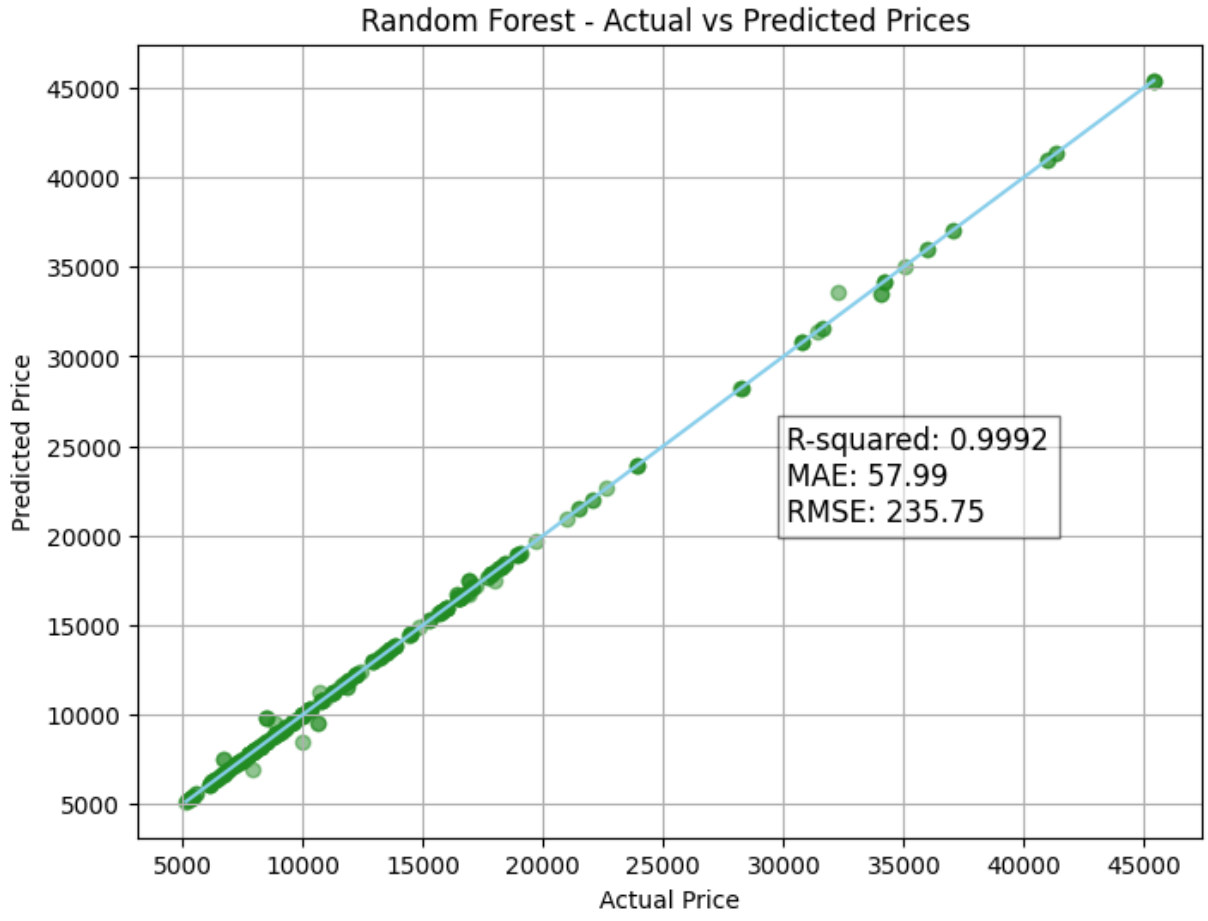
| | Model | RMSE | MAE |
|---|---|---|---|
| 0 | 30 Features | - | - |
| 1 | Linear Regression | 1535.47 | 1125.65 |
| 2 | Decision Tree | 388.86 | 123.83 |
| 3 | Random Forest | 235.75 | 57.99 |
| 4 | Gradient Boosting | 498.27 | 232.62 |
| 5 | 15 Features | - | - |
| 6 | Linear Regression | 1779.52 | 1340.81 |
| 7 | Decision Tree | 388.35 | 123.85 |
| 8 | Random Forest | 271.12 | 82.65 |
| 9 | Gradient Boosting | 397.08 | 216.05 |



## Model Recommendation

### 18.0 Model Selection

After a thorough model comparison, a random forest with 30 selected features is the best model for car price prediction due to its superior performance and robustness. Its RMSE highlights that the squared differences between the predicted and actual car prices from this model have an average error of 235.75 dollars, while the MAE showcases that the predicted car prices from this model are off by 57.99 dollars compared to the actual car prices on average. Furthermore, an adjusted R-squared value of 0.9992 indicates that the variables used in this final random forest model can explain approximately 99.92% of the variability in car prices. It proves its capability to pinpoint vital car features, capture fluctuated data patterns, and utilize them for precise price predictions with minimal error deviations.



19.0 Model Theory

Random forest employs an advanced tree-like structure that splits variables into multiple decision trees, with the goal of mitigating the risk of overfitting and high variance (Yiu, 2019). This algorithm makes the best use of bootstrapping by generating various subsets from the original car dataset, which are then used to train different decision trees. The algorithm's random selection also signifies the diversity in this dataset, reducing the correlation among car features. This randomness plays a vital role in maintaining data integrity without the concern of overusing repeated subsets in the long term.

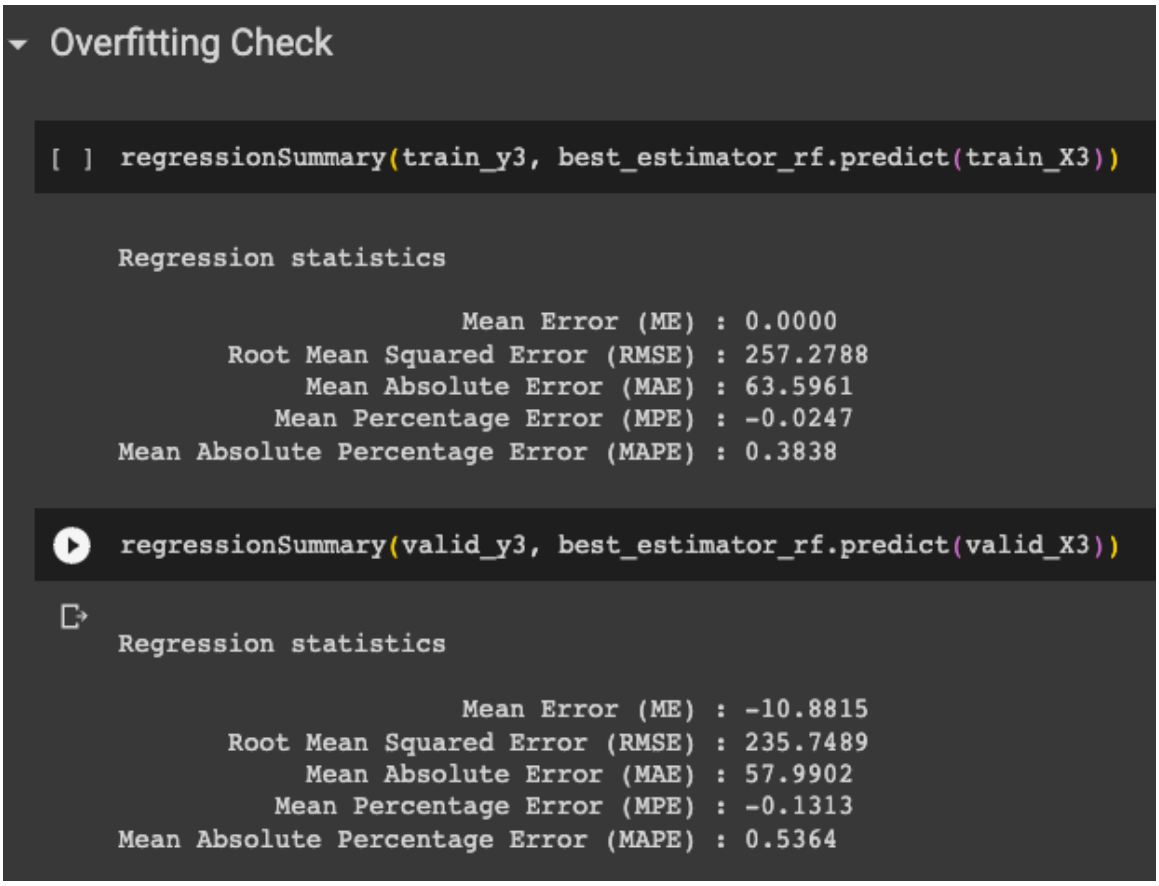
19.1 Model Assumptions and Limitations

Although random forest looks like a decent model in different aspects, its algorithm relies on the consistent relationship between the car features and prices. Every calculation or decision-making process is based on this assumption. However, once the dataset contains data points that deviate from this assumed consistency, the model performance to reflect the true relationship between the predictor features and target variable might be compromised.

Overfitting can be a limitation in this random forest model. When the model performs extremely well on a training set, it may struggle to accurately predict car prices from unseen data. As the learning model is already used to the training data patterns, improving the model performance with validation data becomes challenging, potentially leading to unreliable business insights.

However, there is a method to ensure the model avoids overfitting by examining the statistical summaries for both the training and validation sets. In this model, both the RMSE and MAE values in the validation regression summary are slightly lower than those in the training summary. The RMSE indicates a difference of $21.53 between training and validation sets, while

the MAE only shows a minor discrepancy of $5.61. This represents no strong indications of overfitting, as the difference between them is insignificant.

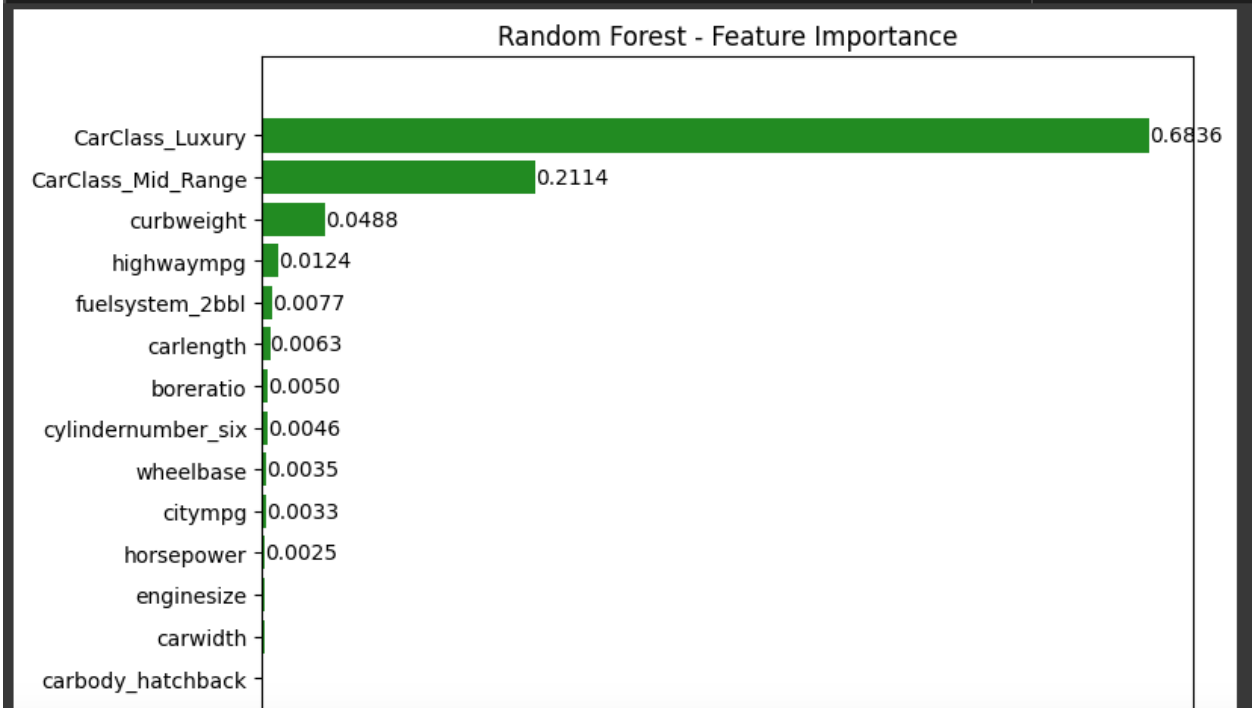



20.0 Model Sensitivity to Key Drivers

Feature importance puts a lot of emphasis on revealing the variables that achieve the highest contribution to predicting vehicle prices. Higher values mean a stronger influence on price impact. The top three variables are luxury, mid-range class, and curb weight, with the importance of 0.68, 0.21, and 0.05 respectively. Showcasing these variables are identified as key drivers and are more sensitive to precisely predicting vehicle prices. For instance, when the target is a luxury car, the likelihood of correctly predicting the car price is significantly higher than other conditions. As these car features strongly influence toward the prediction model, even

slight variations or fluctuations in these variables may result in apparent changes in the price prediction performance.

```
indices = var_imp.argsort()[::1]

plt.figure(figsize=(8, 10))
plt.title('Random Forest - Feature Importance')
bars = plt.barh(range(train_X3.shape[1]), var_imp[indices], align='center',color = 'forestgreen')
plt.yticks(range(train_X3.shape[1]), features_30_F[indices])
plt.xlabel('Importance')
plt.tight_layout()

for bar, importance in zip(bars[-11:], var_imp[indices][-11:]):
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f"{importance:.4f}",
             va='center', ha='left')
plt.show()
```




## Conclusion and Recommendations

### 22.0. Impacts on Business Problem (Scope of the recommended model)

Undoubtedly, the prominence of the top three crucial variables significantly impacts predicting car prices. Geely Auto should prioritize its pricing strategies based on the dominance variables that rank highest in importance. Tailoring the business goals to align with positively correlated features ensures an efficient approach to expanding market share in the US and

European markets. Resources should be allocated accordingly to those variables with high-importance rankings. The feature importance analysis provides a thorough indicator for Geely Auto to efficiently position itself in foreign markets and optimize its pricing techniques based on influential variables.

## 23.0. Recommended Next Steps

Converting the model findings into appropriate actions is a prerequisite to achieving the optimal outcome on car price prediction in foreign markets. Based on the analysis of the feature importance chart, Geely Auto should focus on the top three features to maximize the precision of predicting vehicle prices precisely.

Firstly, Geely should target foreign automotive markets by promoting their luxury and mid-range vehicles. These two variables have the highest importance, signifying a significant influence on the model performance. Secondly, the automotive company should increase the car weights because heavier vehicles correlate positively with accurate price predictions, whereas heavier cars are more likely to yield accurate predictions compared to lighter cars. Thirdly, by grouping the remaining features, fuel efficiency also plays an essential role in maintaining a high level of car price prediction accuracy. Fuel efficiency implies that two-barrel carburetors (2BBL) and highway miles per gallon (MPG) also have moderate importance in assisting the learning model to achieve a precise price prediction. Lastly, Geely Auto should avoid entering foreign markets with variables ranked at 0 importance, as price predictions in such cases may be uncontrollable and less efficient than those driven by high-ranking features. Therefore, the company should be cautious or reconsider its business plan when encountering vehicles equipped with 4 cylinders, a front-wheel drive system, and an L-shape engine type.

```
rf_plot = RandomForestRegressor(bootstrap = False, max_depth = 15, min_samples_leaf = 1,
                                min_samples_split = 2, n_estimators = 100, random_state = 1)
rf_plot.fit(train_X3,train_y3)
var_imp = rf_plot.feature_importances_
pd.DataFrame({'Features':features_30_F,'Importance':var_imp.round(4)}).sort_values(by='Importance', ascending=False)
```

| | Features | Importance |
|---|---|---|
| 28 | CarClass_Luxury | 0.6836 |
| 29 | CarClass_Mid_Range | 0.2114 |
| 4 | curbweight | 0.0488 |
| 10 | highwaympg | 0.0124 |
| 25 | fuelsystem_2bbl | 0.0077 |
| 1 | carlength | 0.0063 |
| 6 | boreratio | 0.0050 |
| 24 | cylindernumber_six | 0.0046 |
| 0 | wheelbase | 0.0035 |
| 9 | citympg | 0.0033 |
| 7 | horsepower | 0.0025 |
| 5 | enginesize | 0.0023 |
| 2 | carwidth | 0.0021 |
| 14 | carbody_hatchback | 0.0013 |
| 8 | peakrpm | 0.0011 |
| 3 | carheight | 0.0010 |
| 12 | aspiration_turbo | 0.0008 |
| 27 | fuelsystem_mpfi | 0.0004 |
| 15 | carbody_sedan | 0.0004 |
| 22 | cylindernumber_five | 0.0003 |
| 26 | fuelsystem_idi | 0.0003 |
| 13 | carbody_hardtop | 0.0003 |
| 11 | fueltype_gas | 0.0003 |
| 18 | enginelocation_rear | 0.0001 |
| 20 | enginetype_ohc | 0.0001 |
| 21 | enginetype_ohcf | 0.0001 |
| 16 | carbody_wagon | 0.0001 |
| 23 | cylindernumber_four | 0.0000 |
| 17 | drivewheel_fwd | 0.0000 |
| 19 | enginetype_l | 0.0000 |

## References

### 24.0 References

Brownlee, J. (2020, August 17). *Ordinal and one-hot encodings for Categorical Data*. MachineLearningMastery.com. https://machinelearningmastery.com/one-hot-encoding-for-categoricaldata/#:~:text=Many%20machine%20learning%20algorithms%20cannot,limitations%20on%20the%20algorithms%20themselves.

Gillis, A. S. (2022, April 15). *What is data splitting and why is it important?*. Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/data-splitting#:~:text=In%20machine%20learning%2C%20data%20splitting,to%20reliably%20fit%20additional%20data.

Rosencrance, L. (2021, January 4). *What is feature engineering for Machine Learning?*. Data Management. https://www.techtarget.com/searchdatamanagement/definition/feature-engineering#:~:text=Feature%20engineering%20can%20help%20data,create%20models%20with%20better%20accuracy.

Rosidi, N. (n.d.). *Machine learning: What is bootstrapping?*. KDnuggets. https://www.kdnuggets.com/2023/03/bootstrapping.html#:~:text=Bootstrapping%20is%20a%20resampling%20technique,same%20size%20as%20the%20original.

Yiu, T. (2021, September 29). *Understanding random forest*. Medium. https://towardsdatascience.com/understanding-random-forest-58381e0602d2