|  |  |
| --- | --- |
| Harshal Malve    **Vehicle Price Prediction and Visualization**  Introduction: As a mechanical engineer with over six years of domain experience, I embarked on a journey to explore data science and artificial intelligence. This capstone project, titled “Vehicle Price Prediction and Visualization,” combines my engineering background with newfound skills. By leveraging machine learning models and Power BI, I aim to predict premium car prices accurately and empower either buyer or sellers of the cars. | Abstract  The “Vehicle Price Prediction and Visualization” project aims to leverage machine learning techniques to predict the prices of premium cars. By combining data science methodologies with Power BI visualizations, this project addresses several key aspects. In the automotive industry, accurate price estimation is crucial for both buyers and sellers. Buyers need to make informed decisions, while sellers want to optimize their pricing strategies. Machine learning models provide a systematic approach to predict car prices based on relevant features, such as brand-model, mileage, engine specifications etc |

Index

[**Phase 1: Understanding data and requirements.** 3](#_Toc175309277)

[1.Business problem: 3](#_Toc175309278)

[2.Dataset: 3](#_Toc175309279)

[3.Key metric 3](#_Toc175309280)

[4.Real world challenges and constraints: 3](#_Toc175309281)

[5.Solutions to similar problems: 3](#_Toc175309282)

[6.References: 3](#_Toc175309283)

[**Phase 2 : EDA and Feature Extraction** 3](#_Toc175309284)

[1.Datasets and Libraries 3](#_Toc175309285)

[2.EDA 3](#_Toc175309286)

[3.Feature Extraction 3](#_Toc175309287)

[4.Feature transformation 3](#_Toc175309288)

[5.Data pre-processing 3](#_Toc175309289)

[6.Data processing 3](#_Toc175309290)

[**Phase 3 : Modelling and Error Analysis** 3](#_Toc175309291)

[1.Selection of model: 3](#_Toc175309292)

[2.Base line model: 3](#_Toc175309293)

[3. Model development: 3](#_Toc175309294)

[**Phase 4: Random forest regressor model** 3](#_Toc175309295)

[1. Data Preparation Steps for Random Forest Regressor: 3](#_Toc175309296)

[2. Random Forest Regressor: 3](#_Toc175309297)

[3.Model development: 3](#_Toc175309298)

[4. Model Evaluation: 3](#_Toc175309299)

[5. Hyper Tuned Model Evaluation: 3](#_Toc175309300)

[**Phase 5: Predicted car price – Power BI Analysis** 3](#_Toc175309301)

[1. Actual car price vs Predicted car price Analysis. 3](#_Toc175309302)

[**1.1 KPI - Key performance parameter of complete Data.** 3](#_Toc175309303)

[**1.2 Price by Manufacturer** 3](#_Toc175309304)

[**1.3 Actual vs Predicted - Price Comparison** 3](#_Toc175309305)

[**1.4 Car Price Summary - Actual vs. Predicted - scatter plot** 3](#_Toc175309306)

[**1.5 Discrepancy between the actual prices and the predicted prices of cars.** 3](#_Toc175309307)

[**1.6 discrepancies between actual and predicted car prices.** 3](#_Toc175309308)

[**1.7 The line chart illustrates three key trends related to car prices:** 3](#_Toc175309309)

[**Phase 6: Conclusion** 3](#_Toc175309310)

**Vehicle Price Prediction and Visualization**

# Phase 1: Understanding data and requirements.

[TOP](#TOC)

## 1.Business problem:

* 1. **Objectives:**

The primary objective of the project is to develop accurate machine learning models that predict the prices of premium cars. By analysing relevant features such as brand, model, mileage, and engine specifications, these models will provide reliable price estimates. Additionally, the project aims to enhance decision-making in the automotive industry by creating insightful Power BI visualizations based on the predicted prices.

* 1. **Challenges:**

Predicting car prices is challenging due to the significant impact of even slight variations in model features. Minor differences can lead to substantial price discrepancies. To address this, we require a robust machine learning model specifically designed for price prediction. This model should meticulously consider every aspect of car features and technical specifications. overcoming this challenge involves building an ML model that thoroughly analyzes and accounts for the nuances of each vehicle’s characteristics.

* 1. **Real World Impact:**

By accurately predicting car prices, potential buyers can make informed decisions. They’ll know whether a listed price is fair or inflated, allowing them to negotiate effectively. Car owners seeking to trade in their vehicles benefit from accurate price estimates. They can negotiate better trade-in values.

Car dealerships and sellers can optimize their inventory management. Accurate price predictions help them avoid overstocking or understocking specific models.

Insurance companies can estimate premiums more accurately based on predicted car values. This benefits both insurers and policyholders.

## 2.Dataset:

**2.1 Datasets:**

Source of the data set is mainly from United States considering Units and column header terminologies. Provided data sets consist of :-

1. imports\_85\_data.csv – the training and testing data set for Selected Premium car brands.

**2.2 Data Fields:**

Fields include:

1. Make

2. Fuel Type

3. Aspiration

4. Number of Doors

5. Body Style

6. Drive Wheels

7. Engine Location

8. Wheelbase (in inches)

9. Length (in inches)

10. Width (in inches)

11. Height (in inches)

12. Curb Weight (in pounds)

13. Engine Type - type of Engine like DOHC or OHCV.

14. Number of Cylinders

15. Engine Size (in cubic inches)

16. Fuel System

17. Bore Ratio.

18. Stroke - Length of piston travels from top to bottom or vice versa

19. Compression Ratio

20. Horsepower

21. Peak RPM

22. City MPG (Miles Per Gallon)

23. Highway MPG (Miles Per Gallon)

24. Price (in dollars)

25. Coumn\_Z – no information (relevance) found

26. col2

**2.3 Data Understanding & Tools:**

Data is provided by client/institute. Data was provided via cloud but due to low size of data set we Download it on local drive and used our own Computational resources rather than Cloude for cost saving purposes. For this particular instance we can use Pandas and Numpy libraries to process the data as we have data in CSV format. As data is product specific additional data can be acquired by having better product understanding of the same.

## 3.Key metric

**3.1 Business Metric:**

Considering this as a feature based problem the output would be considered as price prediction which means output (Y hat belongs to Real Data) there is a possibility of using multiple metrics to compare and understand the model performance.

**3.2 Available metrics:**

There are multiple metric to choose from:

1. R-Squared
2. Mean Absolute Error (MAE)
3. Mean Absolute Percentage Error
4. Mean Squared Error (MSE)
5. Root Mean Squared Error (RMSE)

**3.3 Metric selection and Reasoning:**

The metric which we will use for this Model will be Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R² score.

1. **Mean Absolute Error (MAE):**
   * **Definition:** MAE measures the average absolute difference between predicted car prices and actual prices.
   * **Reason:** Lower MAE indicates better model accuracy in price prediction.
2. **Root Mean Squared Error (RMSE):**

* **Definition:** RMSE represents the square root of the average squared differences between predicted and observed outcomes.
  + **Reason:** Lower RMSE values indicate better model accuracy in predicting car prices.

1. **R-squared (R²) Score:**
   * **Definition:** R² quantifies the proportion of variance in car prices explained by the model.
   * **Reason:** Higher R² indicates better fit to the data.

## 4.Real world challenges and constraints:

**4.1** **Constraints:**

**Data Availability:** As we know more the data better the prediction model. But provided data is bit smaller in the term of availability.

**Feature Engineering Complexity:** Creating relevant features from raw data. As we can se there are multiple columns in data with lots of categories. ML don’t understand alphabetical categories hence need encode them to binary which might increase more columns.

**Business Context Alignment:** we need make sure ML aligns with business goals. Models must address practical needs of Buyers or sellers.

**Ethical Considerations:** Avoiding biases and fairness issues. Biased models can lead to unfair predictions

**4.2** **Requirement for Solution:**

* Solution must include handling of the constraints i.e. manage the ‘curse of dimensionality’.
* High R2 score and balanced Train test efficiency rate is expected.
* As newly test data is not available will need to split train test data accordingly and use the same data to validate and analyse the Output.

## 5.Solutions to similar problems:

For predicting car prices based on feature-based data, several machine learning models have proven effective.

1. **Linear Regression:**
   * **How it Works:**
     + **Linear Regression models the relationship between features and car prices using a linear equation.**
     + **It estimates coefficients for each feature to predict prices.**
   * **Why Choose It:**
     + **Simple and interpretable.**
     + **Works well when features have a linear relationship with prices.**
     + **Good starting point for feature-based predictions.**
2. **Random Forest:**
   * **How it Works:**
     + **Random Forest creates an ensemble of decision trees during training. Each tree uses different subsets of features (such as make, model, year, and mileage) to predict car prices.**
     + **The final prediction is an average of predictions from all the trees.**
   * **Why Choose It:**
     + **Random Forest handles non-linear relationships well.**
     + **It’s robust against overfitting and works well with diverse features.**
     + **Interpretability is reasonable, making it suitable for practical applications.**
3. **XGBoost (Extreme Gradient Boosting):**
   * **How it Works:**
     + **XGBoost is an ensemble method that combines multiple decision trees.**
     + **It optimizes the sum of tree predictions, considering both bias and variance.**
   * **Why Choose It:**
     + **High accuracy due to gradient boosting.**
     + **Handles missing data and outliers effectively.**
     + **Widely used in competitions and real-world applications.**

## 6.References:

**1.** [Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/)

2. [GeeksforGeeks | A computer science portal for geeks](https://www.geeksforgeeks.org/)

3. [DigitalOcean | Cloud Infrastructure for Developers](https://www.digitalocean.com/)

4. [HatchJs Homepage - HatchJS.com](https://hatchjs.com/)

5. [Itsourcecode.com - Partner in Your Coding Journey!](https://itsourcecode.com/)

6. [Statistical Point | Online Statistics library | StatisticalPoint.com](https://statisticalpoint.com/)

7. [Machine Learning Blog | Data Basecamp](https://databasecamp.de/)

**(Reason to chose Random forest)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr#** | **Model** | **R2** | | **R2-Tuned** | | **MAE** | **MAE-Tuned** | **RMSE** | **RMSE-Tuned** |
| **Train** | **Test** | **Train** | **Test** | **Test** | **Test** | **Test** | **Test** |
| **1** | Linear Regression | 0.89 | 0.845 | 0.898 | 0.84539 | 0.0752 | 0.075 | 0.092 | 0.0924 |
|  |
| **2** | Decision Trees | 0.99 | 0.80 | NA | NA | 0.172 | NA | 0.238 | NA |  |
| **3** | Random Forest | 0.98 | 0.91 | 0.98 | 0.904 | 1681.34 | 1515.47 | 2495.72 | 2604.78 |  |
| **4** | Gradient Boosting |  | 0.896 |  | 0.927 | 1902.74 | 1421.78 | 2708.52 | 2261.90 |  |

# Phase 2 : EDA and Feature Extraction

[TOP](#TOC)

* **Exploratory Data Analysis (EDA):**

EDA involves analyzing datasets to understand their main characteristics. It helps summarize data features, detect patterns, and uncover relationships.

* + **Key Activities in EDA:**
    - Data visualization: Plotting histograms, scatter plots, and correlation matrices.
    - Summary statistics: Mean, median, standard deviation, etc.
    - Handling missing data: Impute or remove missing values.
    - Identifying outliers: Detect extreme data points.
  + **EDA Steps:**
    - Load your dataset
    - Explore distributions of features (mileage, year, fuel type).
    - Check correlations between features and target variable (price).
    - Visualize relationships (scatter plots, box plots).
* **Feature Extraction:**

It involves creating new features from existing data to improve model performance.

* **Encoding Categorical Variables:**
  + - Handling categorical variables: Encode them (one-hot encoding, label encoding).
* **Scaling and Normalization:**
  + Standardizing numerical features (e.g., mileage, engine size) ensures they have similar scales. Common methods include Min-Max scaling or std scaling.
  + Scaling prevents features with large values from dominating the model.

## 1.Datasets and Libraries

**1.1 Datasets :-**

Only one data set provided with all the Information.

* + - imports\_85\_data.csv

These datasets contains respective columns and the description are as follows –

1. Make - *Brand name of car (Car Manufacturer)*

2. Fuel Type - *Fuel for cars like Gas (petrol) and Disel*

3. Aspiration - *Addon feature for cars Power/Milage like std (naturally aspirated) and Turbo*

4. Number of Doors - *Two door or Four doors*

5. Body Style - *Like Convertible, Hatchback or Sedan etc*

6. Drive Wheels *- Like Rear wheel drive, Front wheel drive and All wheel drive*

7. Engine Location *- Like Front or Rear*

8. Wheelbase (in inches) - *The distance between the centres of the front and rear wheels.*

9. Length (in inches) - *The total distance between start point of car to end point of car.*

10. Width (in inches) - *Total width of car*

11. Height (in inches) - *Complete height of car*

12. Curb Weight (in pounds) - *The weight of the vehicle including a full tank of fuel and all standard equipment*

13. Engine Type - *Type of Engine like DOHC or OHCV*.

14. Number of Cylinders *- Number of Cylinders in engine like 3,4,6,8,12.*

15. Engine Size (in cubic inches) *- The volume of fuel and air that can be pushed through a car's cylinders*

16. Fuel System *- Fuel injecting system for engine.*

17. Bore Ratio – *Ratio of Hight and Diameter of engine bore.*

18. Stroke - *Length of piston travels from top to bottom or vice versa*

19. Compression Ratio -  *The volume of the cylinder and combustion chamber of engine*

20. Horsepower - *Power Output of engine*

21. Peak RPM - *Maximum rotation of engine per minutes.*

22. City MPG (Miles Per Gallon) - *Economy of car within city road conditions.*

23. Highway MPG (Miles Per Gallon) - *Economy of car within Highway Road conditions.*

24. Price (in dollars) - *Provided Car price*.

25. Coumn\_Z – *No information (relevance) found*

26. col2 – *No information (relevance) found*

**1.2** **Libraries :-**

We have used multiple libraries to perform EDA:-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

from sklearn.preprocessing import MinMaxScaler

from sklearn.impute import KNNImputer

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

from sklearn.linear\_model import Ridge

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import ExtraTreesRegressor

Description of these libraries are as follows: -

1. import pandas as pd: Imports the Pandas library for data manipulation and analysis, commonly used for handling data in tabular form (DataFrames).
2. import numpy as np: Imports the NumPy library, which provides support for numerical operations, arrays, and mathematical functions.
3. import matplotlib.pyplot as plt: Imports the Matplotlib library for creating visualizations, such as plots and charts.
4. import seaborn as sns: Imports the Seaborn library, which enhances Matplotlib’s visualizations with additional themes and statistical features.
5. %matplotlib inline: A Jupyter Notebook magic command that ensures Matplotlib plots are displayed directly in the notebook.
6. from sklearn.preprocessing import MinMaxScaler: Imports the MinMaxScaler class from Scikit-learn, used for feature scaling (normalization) between 0 and 1.
7. KNN Locates the k nearest data points (neighbours) based on similarity (usually Euclidean distance). - Checks their values for the missing feature. - Takes an average (or weighted average) of those values.

## 2.EDA

We will start with understanding of the data. First we need to Import data in Jupyter notebook.



After importing data we checked Data types. Data has float, object (Str) and int.

**2.1** **Finding NA and Null values :-**

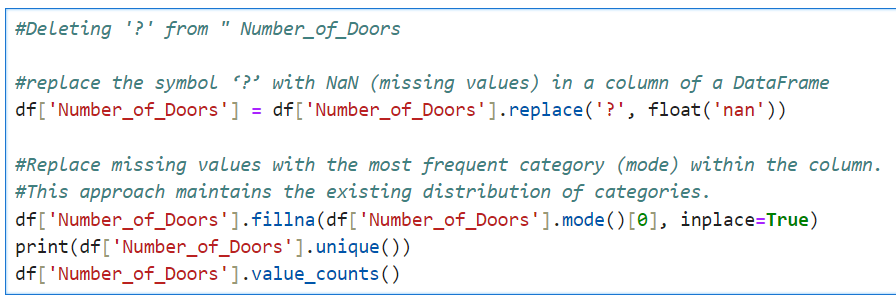
We found Null or NA values In columns col2, Price\_Dollars, Coumn\_Z, Bore\_Ratio, Stroke, Horsepower, Peak\_RPM.

* Filling the null values in the ‘Price\_Dollars’ column with the average cost of vehicles having a specific ‘Number\_of\_Cylinders’.
* Filling the Bore\_Ratio, Stroke, Horsepower with ‘KNNImputer’ (k-nearest neighbours) - if we take average of any parameters, it could be "Engine size". but we cannot take average parameters from Engine size as it is not categorical hence decided to go with KNN Imputation for Empty cells in Peak RPM and Horsepower.

**2.2 Finding “dirty” or “unstructured” data.**

When data contains symbols like ?, !, or @ in certain cells, it is often referred to as “dirty” or “unstructured” data.

* We check the Unique data in all the cells and Found ‘Number\_of\_Doors’ column has ‘?’ in it.
* First we replaced ‘?’ with NaN (missing values) in the column of data frame.
* Then we Replaced missing values with the most frequent category (mode) within the column.
* This approach maintains the existing distribution of categories.



**2.3 Unique values**

Analysing unique values for all the categorical columns 'Fuel\_Type', 'Number\_of\_Doors', 'Body\_Style', 'Drive\_Wheels', 'Engine\_Location', 'Engine\_Type', 'Number\_of\_Cylinders', 'Fuel\_System', 'Aspiration',

'Number\_of\_Cylinders'.

Following are the unique categorical values.

* Fuel\_Type - ['gas' 'diesel']
* Number\_of\_Doors - ['two' 'four']
* Body\_Style - ['convertible' 'hatchback' 'sedan' 'wagon' 'hardtop']
* Drive\_Wheels - ['rwd' 'fwd' '4wd']
* Engine\_Location - ['front' 'rear']
* Engine\_Type - ['dohc' 'ohcv' 'ohc' 'l' 'rotor' 'ohcf' 'dohcv']
* Number\_of\_Cylinders - ['four' 'six' 'five' 'three' 'twelve' 'two' 'eight']
* Fuel\_System - ['mpfi' '2bbl' 'mfi' '1bbl' 'spfi' '4bbl' 'idi' 'spdi']
* Aspiration - ['std' 'turbo']
* Number\_of\_Cylinders - ['four' 'six' 'five' 'three' 'twelve' 'two' 'eight']

## 3.Feature Extraction

**3.1 Encoding**

In our data preprocessing phase, we address columns containing alphabetical categorical values. Since machine learning algorithms operate on numerical data, we’ll encode these categorical values into digital representations. Common techniques include one-hot encoding, label encoding, frequency encoding. By doing so, we ensure compatibility with ML models and facilitate accurate predictions.

Data Encoding Strategy for Categorical Columns

1. Label Encoding**:**
   * We will apply label encoding to the following columns:
     + ‘Fuel\_Type’
     + ‘Aspiration’
     + ‘Number\_of\_Doors’
     + ‘Drive\_Wheels’
     + ‘Engine\_Location’
     + ‘Number\_of\_Cylinders’
   * Reason**:** Hot-not encoding (one-hot encoding) was causing the “curse of dimensionality,” resulting in too many columns. Label encoding assigns numeric values (0, 1, 2, etc.) based on the weightage of each category relative to car prices.
2. Frequency Encoding:
   * We will use frequency encoding for the following columns:
     + ‘Make\_Car\_Manufacturer’
     + ‘Body\_Style’
     + ‘Engine\_Type’
     + ‘Fuel\_System’
   * Reason**:** These columns have multiple categories. Frequency encoding assigns values based on the relative occurrence of each category, capturing their importance in the dataset.

By employing these encoding techniques, we ensure compatibility with machine learning models while preserving meaningful information from categorical features.

*Note - When applying frequency encoding to categorical columns, it generates a new column with encoded values based on category frequencies. Consequently, we should drop the original categorical column to avoid redundancy and maintain a concise feature set*.

**3.2 Finding corelation of data with output column**

**“**Price (in dollars)” with the help of ***correlation matrix*** function “df.corr()” and Heat map.

Removing least corelated columns from the data set to reduce dimensionality.

* 'Fuel\_Type',
* 'Aspiration',
* 'Number\_of\_Doors',
* 'Engine\_Location',
* 'Height\_inches',
* 'Stroke',
* 'Compression\_Ratio',
* 'Peak\_RPM',
* 'Coumn\_Z',
* 'Make\_Car\_Manufacturer\_FrequencyEncoded',
* 'Body\_Style\_FrequencyEncoded',
* 'Fuel\_System\_FrequencyEncoded'.

Above are least correlated columns with Car price hence dropping them.

Keeping ‘city\_MPG’, ‘Highway\_MPG’ - A correlation of -0.65/-0.68

Interpretation - When fuel efficiency (MPG) increases (meaning lower fuel consumption), car prices tend to be Lower- Most of the vehicles are performance based hence Higher the engine size lower the fuel eff. lower the fuel eff. means Higher the Car price. likewise, gas-guzzlers might be associated with Higher prices. Hence Keeping City\_Mpg, Highway\_MPG.

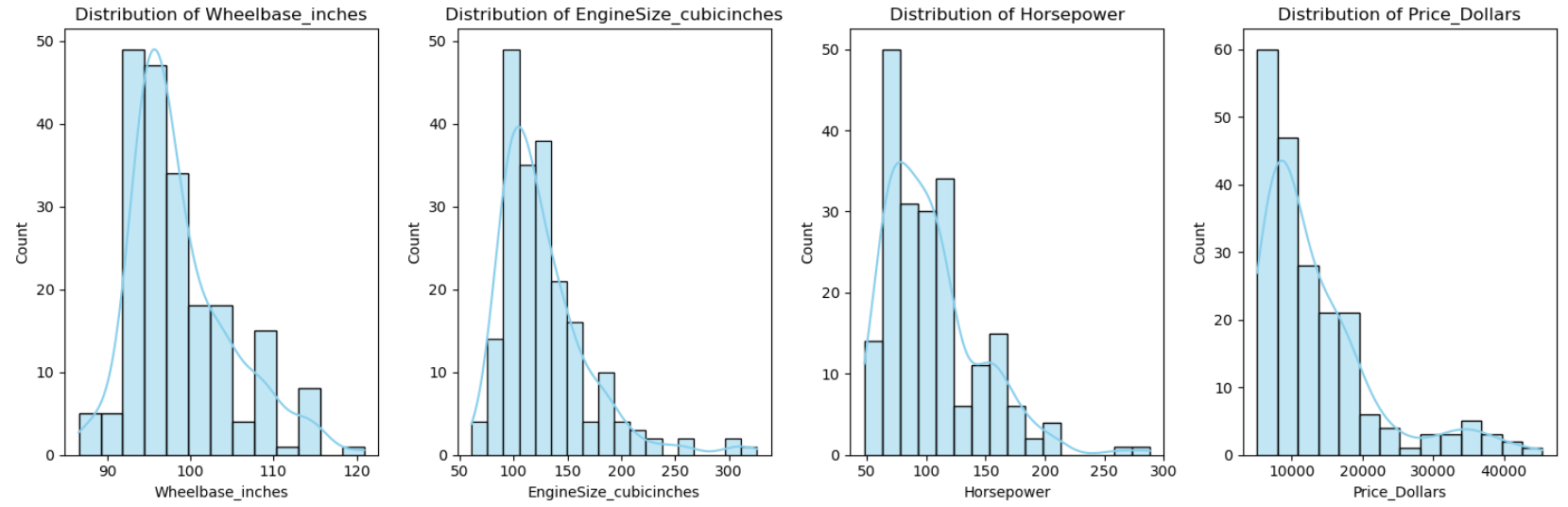
## 4.Feature transformation

**4.1Log-Transformation:**

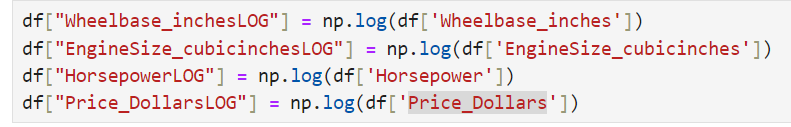
Log transformation is a feature transformation technique used in machine learning to modify the values of a numeric variable by taking the logarithm of each value.

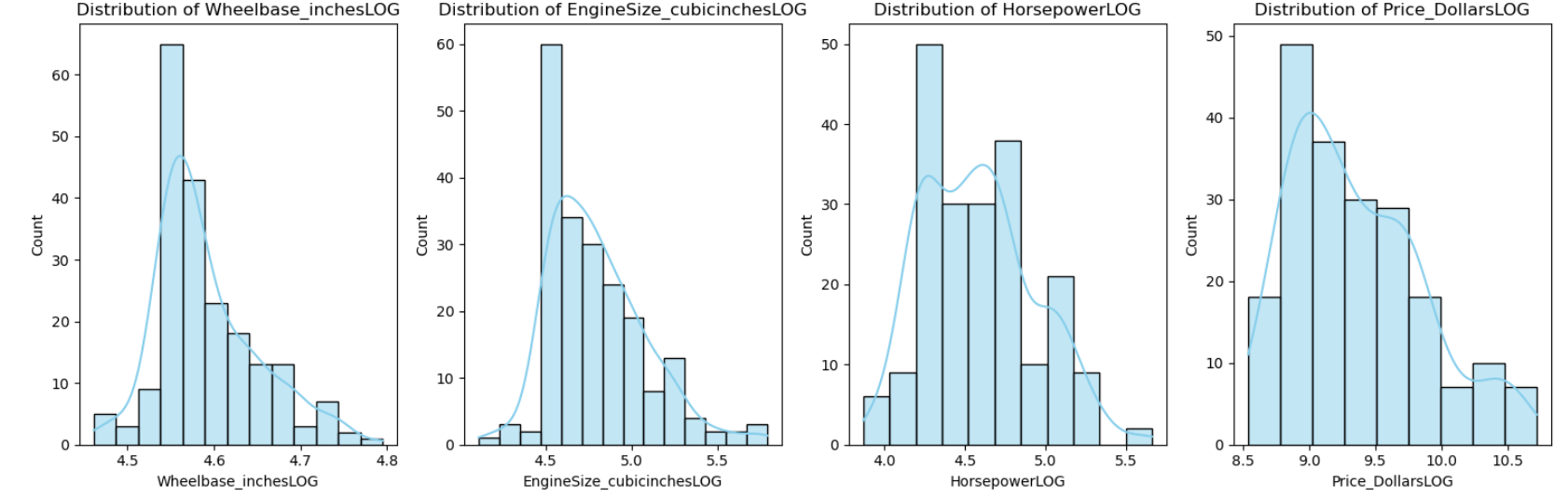
Log transformation can be used to normalize data that is not normally distributed. In many machine learning algorithms, having normally distributed data ensures better performance. By taking the log of the data, we transform it into a more normal distribution, making it easier to analyse and model**.**

By looking at distribution plots of ‘Wheelbase\_inches’, ‘EngineSize\_cubicinches’, ‘Horsepower’, Price\_Dollars are left skewd hence log transformation to make it as Gaussian as possible.



Before log transformation





After log transformation

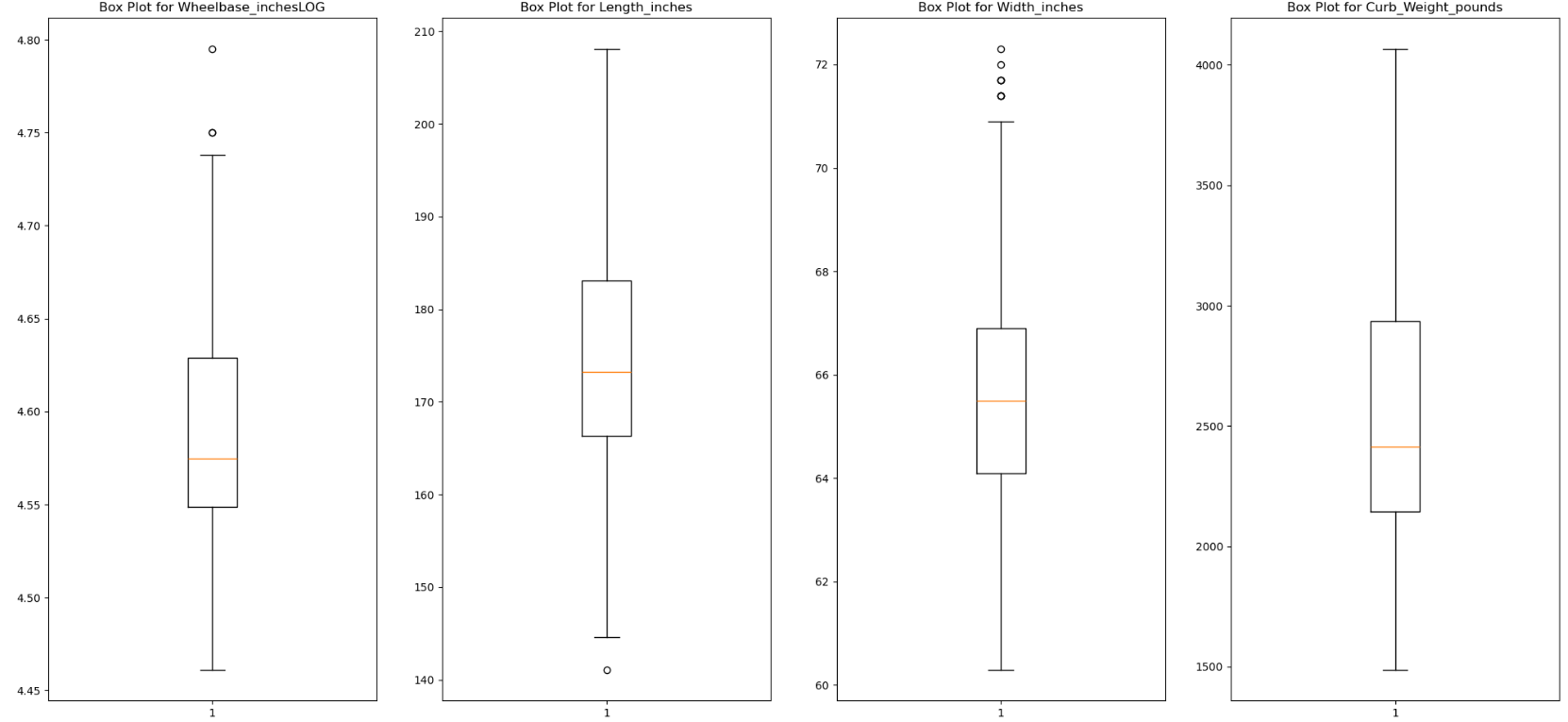
*Note - When applying Log transformation, it generates a new column with transformed values, we should drop the original categorical column to avoid redundancy and maintain a concise feature set*.

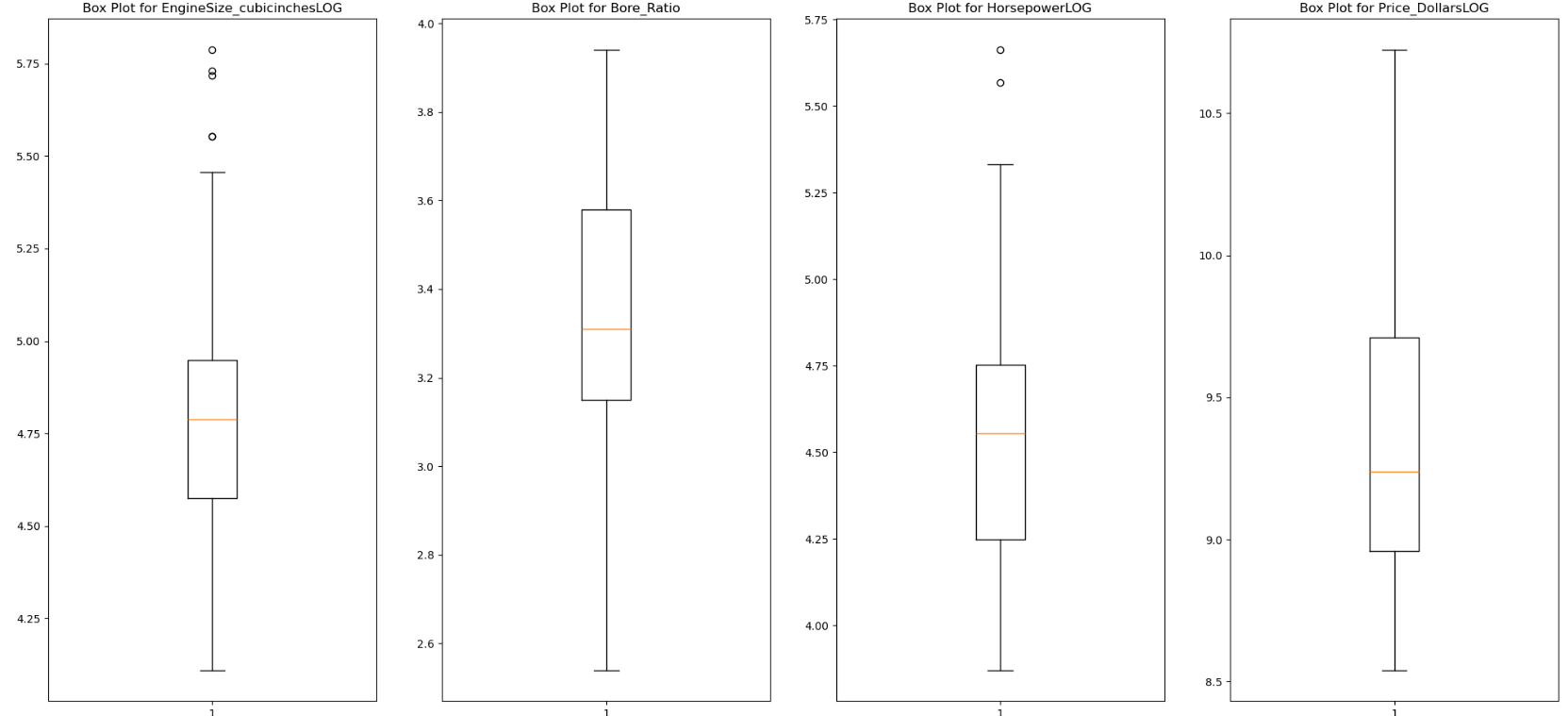
## 5.Data pre-processing

**5.1 Handling outliers:**

Handling outliers in machine learning models is an essential step for building accurate and robust models. Let’s explore some strategies for dealing with outliers:

* **Identifying Outliers**: The first step is to identify outliers using statistical methods (such as Z-score or IQR) or visualization techniques. These helps quantify how much a data point deviates from the rest of the data.
* We found outliers in following columns 'Wheelbase\_inchesLOG', 'Length\_inches', 'Width\_inches', 'Curb\_Weight\_pounds', 'EngineSize\_cubicinchesLOG', 'Bore\_Ratio', 'HorsepowerLOG', 'Price\_DollarsLOG'





Before handling outliers.

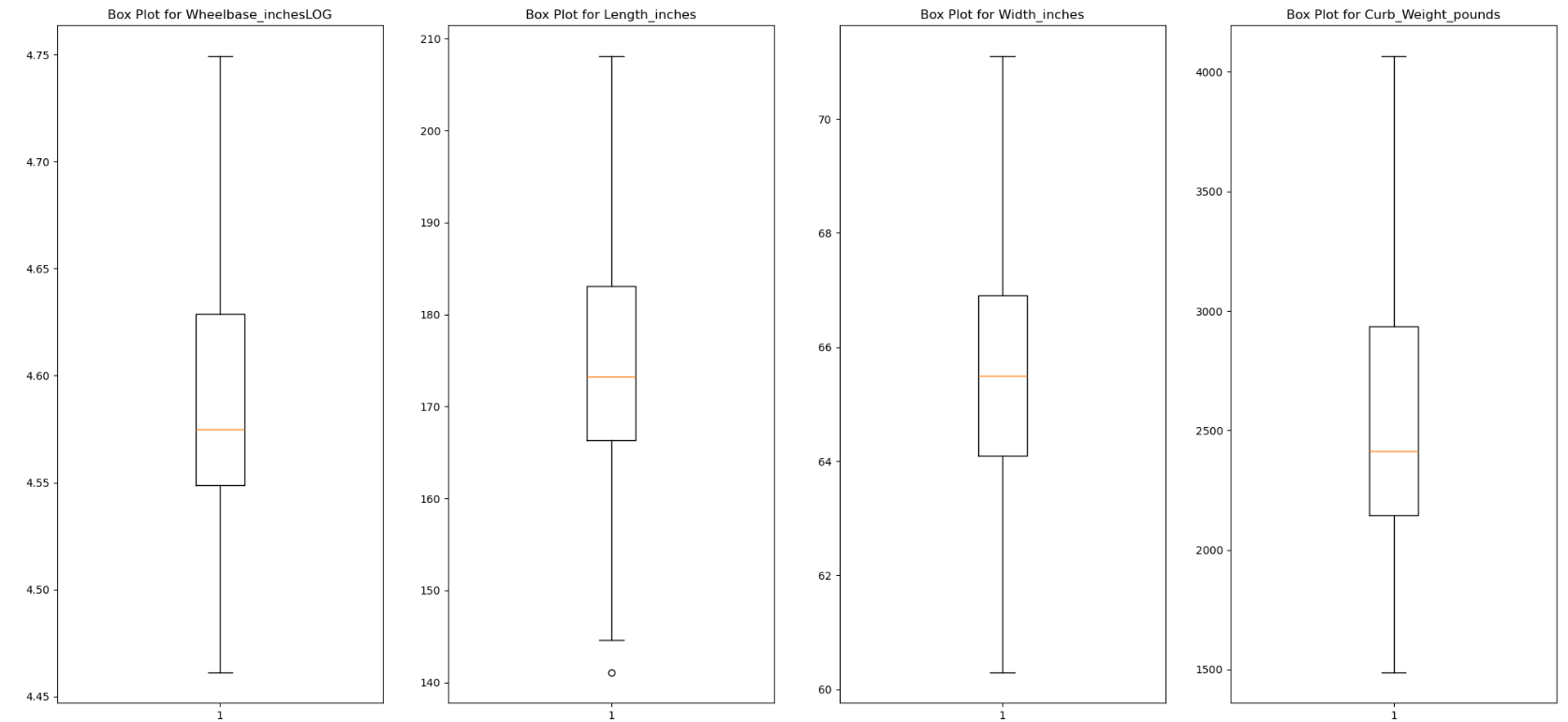
* In our analysis, we utilized box plots to identify outliers within the dataset. When dealing with outliers, we typically have two options: removal or transformation. Given our limited data availability, we opted for data transformation.

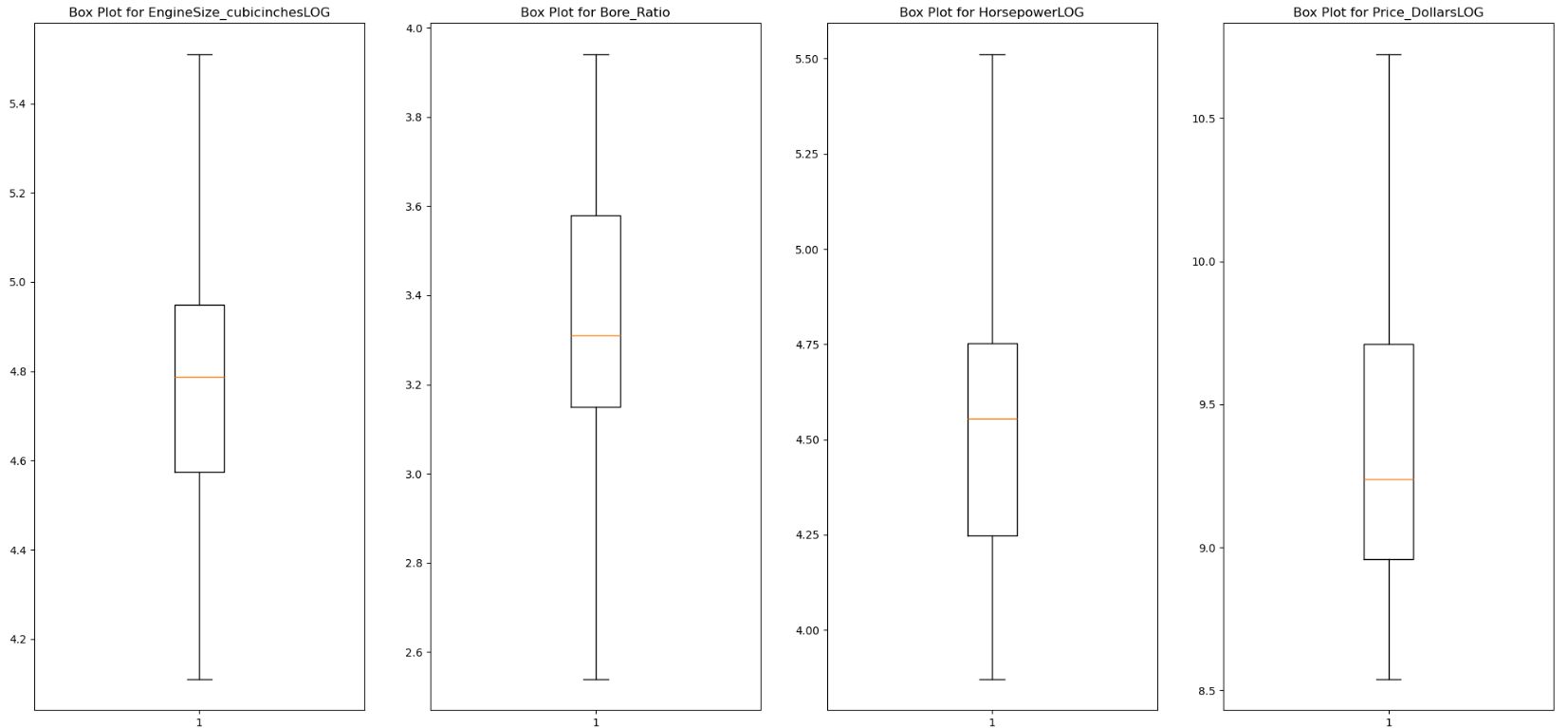
Specifically, we applied the **winsorization method** to address outliers. This involves replacing outlier values with the corresponding percentiles (25th and 75th) of the same column. By doing so, we mitigate the impact of extreme values while preserving the overall distribution of the data.

* Reasons to choose Winsorization - Winsorization helps avoid bias caused by extreme outliers when calculating metrics like mean and standard deviation.

It provides a more accurate view of the central tendency and spread of the data.

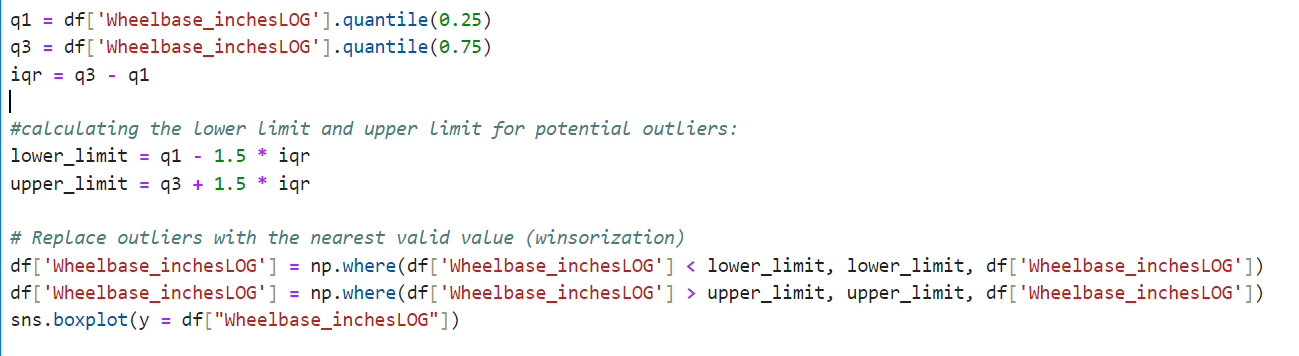
Winsorization retains extreme observations but replaces them with less extreme values





After handling outliers.

* We used IQR method to work on outliers.



* Interquartile Range (IQR):  The IQR represents the range of values within which the middle 50% of the data lies. It is the difference between the third quartile (Q3) and the first quartile (Q1).
* Q1 (First Quartile): The value below which 25% of the data lies.

Q3 (Third Quartile): The value below which 75% of the data lies.

* Formula:

IQR = Q3 - Q1

* Reason: IQR helps identify the spread of the central portion of the data, making it useful for detecting outliers and understanding variability.

## 6.Data processing

**6.1 Scaling:**

Input variables may have different units (e.g., feet, Inches, Miles, etc), resulting in different scales. Large input values can lead to unstable models with large weight values, affecting performance and generalization.

**6.2 Data Scaling Methods:**

* Standardization: Standardization rescales a dataset to have a mean of 0 and a standard deviation of 1.

 There is no specific upper or lower bound for the maximum and minimum values after standardization.

* Normalization: Normalization rescales a dataset so that each value falls between 0 and 1.

 A normalized dataset always has values within the [0, 1] range

* 1. **Selecting method.**
* As we can observe from above distribution chart most of the data is Gaussian/Normal hence choosing MinMax scaler It brings all features into a consistent scale.
* The main advantage of the MinMax Scaler is that it preserves the shape of the original distribution while bringing the values within a desired range and price prediction model is very sensitive towards data hence going towards Minmax scaling.



# Phase 3 : Modelling and Error Analysis

[TOP](#TOC)

## 1.Selection of model:

* Linear Regression
* Decision Tree
* Random Forest
* XGBoost (Extreme Gradient Boosting)

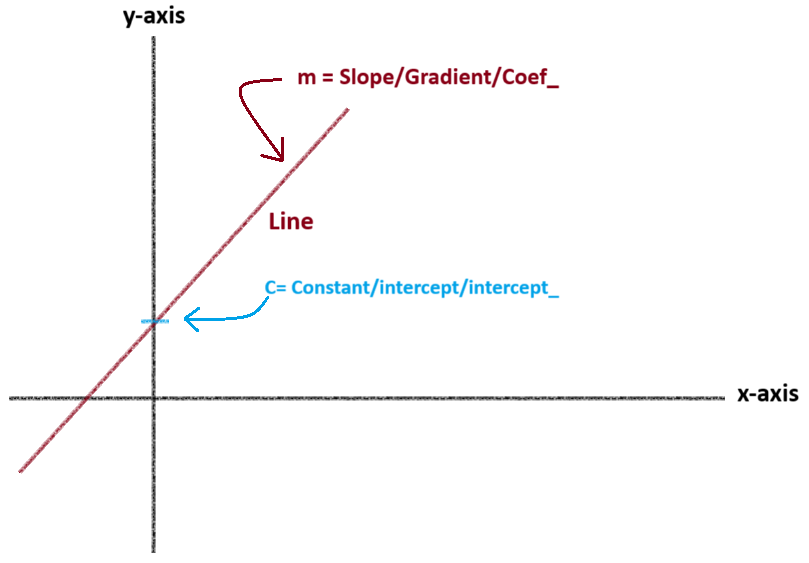
## 2.Base line model:

* We will be using **Linear Regression** as our first model. As we have optimized/processed the data accordingly and Linear Regression is a fundamental and widely used technique for predicting numerical values based on input features. We want to predict the price of a car based on relevant features (e.g., mileage, engine size, brand, etc.). Linear regression assumes a linear relationship between the input features (independent variables) and the target variable (price). We assess the model’s performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared (coefficient of determination). Once trained, the model can predict car prices for new data.

The model equation for simple linear regression is:

y=β0​+β1​⋅x+ϵ

(y=mx+c)



where:

* (y) is the predicted price.
* (β0 \beta\_0) is the intercept (constant term).
* (β1 \beta\_1) is the coefficient for the feature (x).
* (ϵ \epsilon) represents the error term (residuals).

In practice, we often use multiple features (e.g., mileage, engine size, brand) for better predictions:

(y=m**1**x**1**+ m**2**x**2** +…….+ m**n**x**n** +c)

Where:

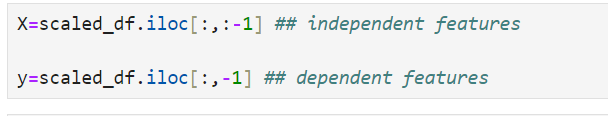
* (β1\beta\_1, \beta\_2, \....., \beta\_n) are the coefficients for each feature (x\_1, x\_2, \....., x\_n)

## 3. Model development:

**3.1 Data separation**

First we separate independent and dependent features means Input columns and Output column.

* **In our case all the features are Input columns.**
* 'Drive\_Wheels',
* 'Length\_inches',
* 'Width\_inches',
* 'Curb\_Weight\_pounds',
* 'Number\_of\_Cylinders',
* 'Bore\_Ratio',
* 'City\_MPG',
* 'Highway\_MPG',
* 'Engine\_Type\_FrequencyEncoded',
* 'Wheelbase\_inchesLOG',
* 'EngineSize\_cubicinchesLOG',
* 'HorsepowerLOG'.
* **And output column will be**
* 'Price\_DollarsLOG'

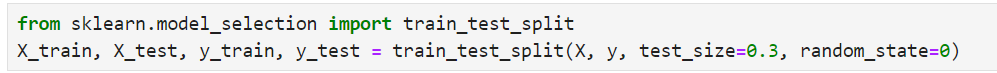


**‘.iloc[:,:-1]’** selects all rows and all columns except the last one from the DataFrame

**‘.iloc[:,-1]’** selects the last column from the DataFrame

**3.2 Train test split**

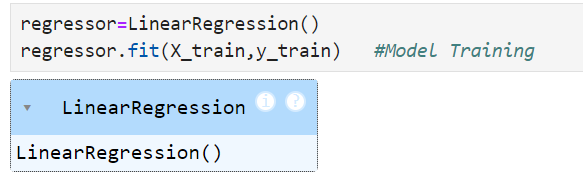
* When building a machine learning model, we divide our dataset into two subsets:
  + **Training set**: Used to train the model (learn patterns).
  + **Test set**: Used to evaluate how well the trained model generalizes to unseen data.
  + Randomly split the dataset into training and test subsets.
  + Typically, around 70-80% of the data is used for training, and the remaining 20-30% for testing.
  + In our case we have taken 30% for test size.
  + And kept random state = 0 (means it will take same train test portion every time)



**3.3 Model Fitting**

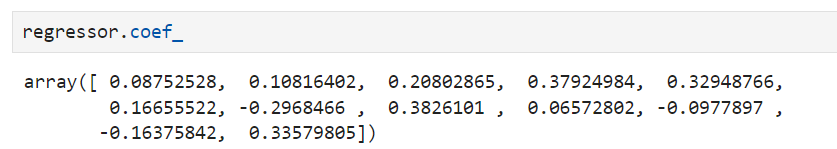
Once train test split is done we use the LinearRegression class from scikit-learn library. It’s time to fit the model meaning we are training the model.

* + - The LinearRegression class implements ordinary least squares (OLS) linear regression. It fits a linear model with coefficients (m = (m1,m2,…..,mn)) to minimize the residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation.
    - In simpler terms, it helps you find the best-fitting straight line through your data points.



**3.4 Coefficient**

after training the data we will need to find coefficient – Slope – gradient –(m) - coef\_

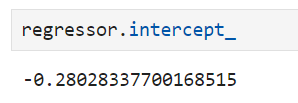


Following are the coefficients of all the lines (input columns)

* 0.08752528,
* 0.10816402,
* 0.20802865,
* 0.37924984,
* 0.32948766,
* 0.16655522,
* -0.2968466 ,
* 0.3826101 ,
* 0.06572802,
* -0.0977897 ,
* -0.16375842,
* 0.33579805

**3.5 Constant**

Next step will be finding Constant – Intercept – (c) – intercept\_



**3.6 Linear Regression Model Training**

let’s train the date of X\_train and X\_test with regressor class.

* prediction\_train:

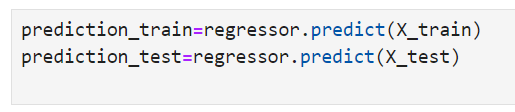
we are calling regressor.predict(X\_train), which means we are using your trained linear regression model to predict the target variable (y) for the training data.

The result stored in prediction\_train will be an array (or a pandas Series) containing the predicted values for the target variable based on the features in X\_train.

* prediction\_test:

Similarly, we are calling regressor.predict(X\_test), but this time on your test data.

The result stored in prediction\_test will be an array (or a pandas Series) containing the predicted values for the target variable based on the features in X\_test.



**3.7 R-squared (R²) on Train Data** – Once we are done with raining our data it’s time to check how well our regression model explains the variance in the target variable.

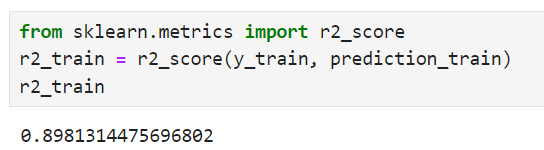
For that we are using R-squared (Coefficient of Determination).

value of R-squared ranges from 0 to 1:

* **0**: Our model explains none of the.
* **1**: Our model perfectly predicts the target variable.

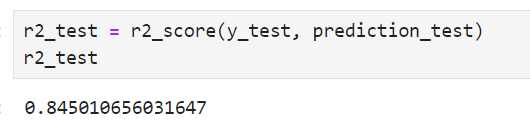
 calculated R-squared score for the training data is approximately **0.8981**

About 89.81% of the variance in the actual target values (from your training dataset) can be explained by your linear regression model.



**3.8 R-squared (R²) on Test Data** - The value we have calculated for r2\_test is approximately **0.8459**.

An R-squared of **0.8459** means that around **84.59%** of the variance in the actual target values (from your test dataset) can be explained by your linear regression model



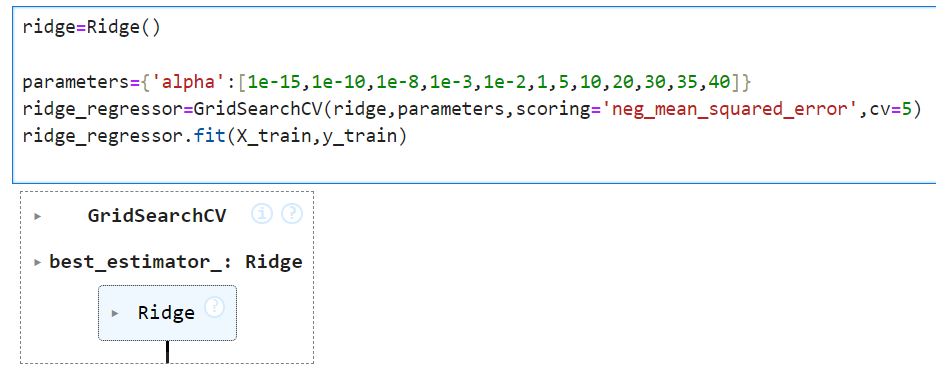
**3.9 MAE and RMSE of the model is** :

* MAE: 0.0752
* RMSE: 0.0925

**3.10 Hyper parameter Tuning.**

We have received pretty decent R2 scores let’s do hyper parameter tuning o improve it.

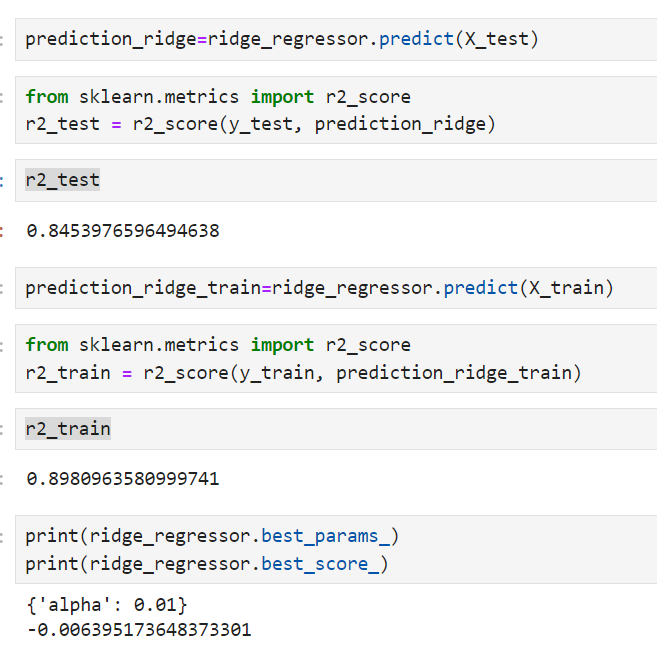
* The Ridge() class is instantiated, creating an instance of the Ridge regression model, which will be used for further analysis.
* The next lines involve hyperparameter tuning using grid search.
* A dictionary named parameters is defined with a single key 'alpha' and an array of values ranging from very small (e.g., 1e-15) to larger values (e.g., 40).
* The GridSearchCV class is used to perform an exhaustive search over the specified hyperparameter values.
* The ridge\_regressor instance is created with the following parameters:
  + Estimator: The Ridge regression model (ridge).
  + Parameters: The dictionary of hyperparameters (parameters).
  + Scoring: The metric used for evaluation (negative mean squared error).
  + Cross-validation: 5-fold cross-validation (cv=5).
* Finally, the fit(X\_train, y\_train) method is called on ridge\_regressor, which trains the model using the training data.
* Model Evaluation:
  + The grid search helps find the optimal value for the regularization parameter (alpha) by evaluating different combinations of hyperparameters.
  + The best estimator (model with the best hyperparameters) can be accessed using ridge\_regressor.best\_estimator\_



**3.11 As a result Ridge regressor with alfa** –

[1e-15,1e-10,1e-8,1e-3,1e- 2,1,5,10,20,30,35,40] did not changed much of the output. R2\_test turned out to be 0.845 and r2\_train is 0.898.

the best parameters found by the ridge regression model -0.00639 with 0.01 alfa.



**3.12 MAE and RMSE of the model with ridge regression is** :

* MAE: 0.07505
* RMSE: 0.0924

# Phase 4: Random forest regressor model

[TOP](#TOC)

## 1. Data Preparation Steps for Random Forest Regressor:

* I started with data from the base model.
* Then, followed similar preprocessing steps as you did for the Linear Regressor model, with a few exceptions.

1. **Common Steps (Same as Linear Regressor)**:
   * **Feature Engineering**: I performed feature engineering, likely including handling missing values, creating new features, and selecting relevant features.
   * **Encoding**: You encoded categorical variables to numerical representations (e.g., label, frequency).
2. **Exceptions (Not Applied for Random Forest)**:
   * **Min-Max Scaling**: I skipped Min-Max scaling because Random Forest models are not sensitive to feature scaling.
   * **Log Transformation**: Similarly, I didn’t apply log transformation because Random Forests don’t require it.
3. **Final Step**:
   * I kept the data prepared up to the encoding stage, as Random Forests do not need additional scaling or transformation.

## 2. Random Forest Regressor:

**Random Forest** is an ensemble learning technique based on decision trees. It combines multiple decision trees to create a robust and accurate model.

Each tree in the forest is trained on a random subset of the data, and their predictions are averaged or aggregated.

**2.1 Advantages -**

* High Accuracy: Random Forests offer better accuracy compared to single decision trees.
* Robustness: They handle outliers and noisy data well.
* Nonlinear Relationships: Random Forests can capture complex nonlinear relationships in the data.
* Feature Importance: They provide insights into feature importance, helping you understand which features impact predictions the most.

**2.2 Reason for model selection –**

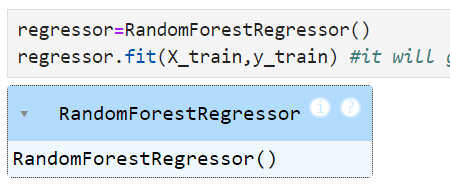
* **Robustness to Noise**: Car prices can be influenced by various factors (e.g., brand, mileage, year). Random Forests handle noisy data and outliers effectively.
* **Feature Importance**: You can identify which car features (e.g., engine size, brand reputation) contribute significantly to the price.
* **Nonlinear Effects**: Car prices often exhibit nonlinear relationships (e.g., luxury cars may have diminishing returns on price). Random Forests capture such complexities.
* **Ensemble Approach**: By combining multiple trees, Random Forests reduce overfitting and improve generalization.

In summary, Random Forest Regression is a powerful choice for car price prediction due to its robustness, ability to handle nonlinearities, and feature importance insights.

## 3.Model development:

*Note –  Prior to this stage, we have already completed the data preprocessing steps, including input-output separation and train-test data splitting. Therefore, we will skip those initial steps and proceed directly to the model training section.*

**3.1** Once train test split is done we use RandomForestRegressor class from scikit-learn library. It’s time to fit the model, meaning we are training the model.



* An instance of RandomForestRegressor is created. This regression model is commonly used for predicting numeric values (such as car prices) based on input features.
* Fitting the Model: The model is trained (or “fit”) using some data represented by ‘X’ and ‘y’. This is where the model learns from the data.
* Output Confirmation: The code prints a message confirming that the instance of RandomForestRegressor was created successfully.

## 4. Model Evaluation:

**4.1 Model Training**

let’s train the date of X\_train and X\_test with regressor class.

* prediction\_train:

we are calling regressor.predict(X\_train), which means we are using your trained linear regression model to predict the target variable (y) for the training data.

The result stored in prediction\_train will be an array (or a pandas Series) containing the predicted values for the target variable based on the features in X\_train.

* prediction\_test:

Similarly, we are calling regressor.predict(X\_test), but this time on your test data.

The result stored in prediction will be an array (or a pandas Series) containing the predicted values for the target variable based on the features in X\_test.

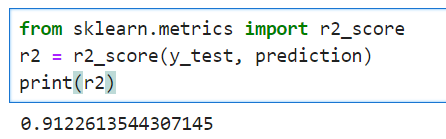


**

**4.2 R-squared (R²) on test Data**

Calculated R-squared score for the testing data is approximately **0.912**

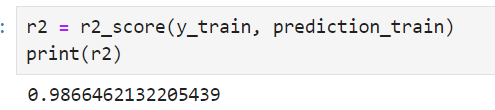
About 91.2% of the variance in the actual target values (from your testing dataset) can be explained by my regressor model.



**4.3 R-squared (R²) on Train Data**

Calculated R-squared score for the training data is approximately **0.986**

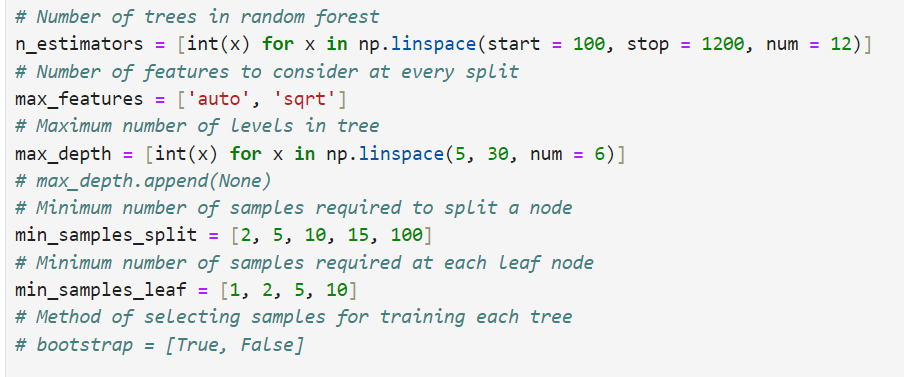
About 98.6% of the variance in the actual target values (from your training dataset) can be explained by your linear regression model.



**4.4 Hyper parameter Tuning**

We have received pretty decent R2 scores let’s do hyper parameter tuning to see if we can improve it.

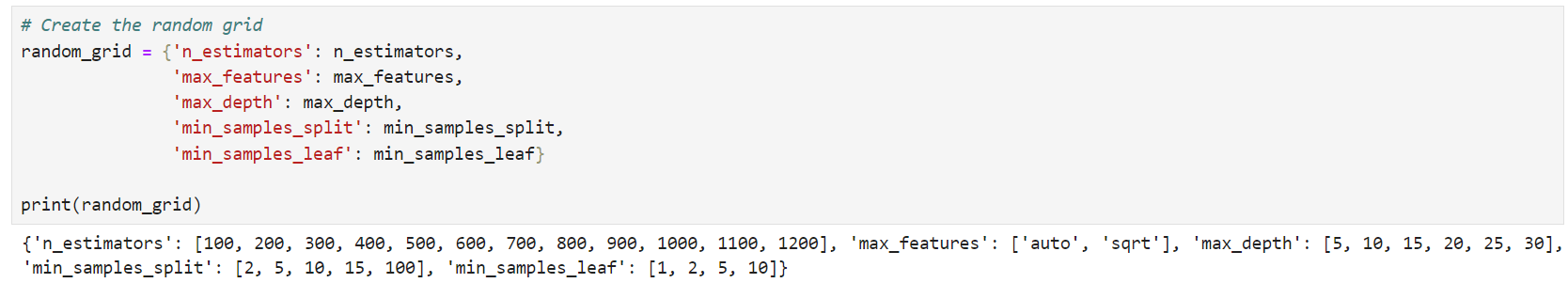
Let’s call ‘RandomForestRegressor()’ rf1 for hyper parameter tuning.



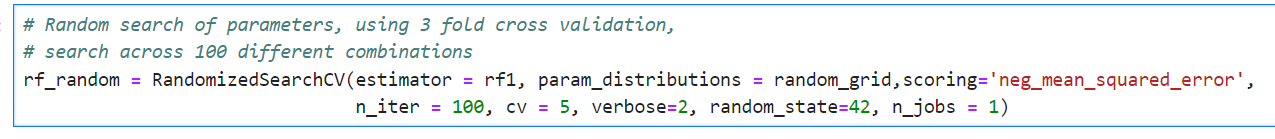
1. **Number of Trees (n\_estimators)**: The n\_estimators parameter determines how many decision trees are included in the random forest. In our case, it’s a list of values ranging from 100 to 1200, with 12 evenly spaced points.
2. **Number of Features (max\_features)**: This parameter controls the maximum number of features considered at each split. It’s set to either “auto” or “sqrt,” which means the algorithm will choose the square root of the total features.
3. **Maximum Depth (max\_depth)**: The max\_depth parameter specifies the maximum depth of each tree. I’ve defined a list of depth values from 5 to 30.
4. **Minimum Samples to Split (min\_samples\_split)**: This parameter sets the minimum number of samples required to split a node. I have provided a list of possible values (2, 5, 10, 15, and 100).
5. **Minimum Samples at Leaf Nodes (min\_samples\_leaf)**: The min\_samples\_leaf parameter determines the minimum number of samples required at each leaf node. I’ve specified values (1, 2, 5, and 10).
6. **Bootstrap Sampling (bootstrap)**: The bootstrap parameter controls whether bootstrapping (random sampling with replacement) is used when selecting samples for training each tree. It’s set to either True or False.

These settings play a crucial role in tuning the performance of your random forest model.

**4.5 created one dictionary named random\_grid**



**4.6** did Random search of parameters, using 3 fold cross validation, search across 100 different combinations.



1. **estimator**: This refers to the machine learning model (in your case, rf1) that you want to optimize using hyperparameter tuning. The RandomizedSearchCV will explore different hyperparameter settings for this model.
2. **param\_distributions**: It specifies the hyperparameter search space. You provide a dictionary (or list of dictionaries) with parameter names as keys and distributions (or lists) of possible parameter values. The search will sample from these distributions during optimization.
3. **scoring**: This determines the evaluation metric used during cross-validation. In your case, it’s set to 'neg\_mean\_squared\_error', which means the model’s performance will be evaluated based on the negative mean squared error (lower is better).
4. **n\_iter**: The number of random parameter settings to sample. In your example, it’s set to 100, meaning the search will try 100 different combinations of hyperparameters.
5. **cv**: The number of cross-validation folds. Here, it’s set to 5, indicating 5-fold cross-validation.
6. **verbose**: Controls the verbosity of the output during the search. A higher value (like 2) provides more detailed information.
7. **random\_state**: Ensures reproducibility by setting the random seed for sampling.
8. **n\_jobs**: Specifies the number of CPU cores to use for parallel computation. In your case, it’s set to 1 (single core).

**4.7** we use RandomForestRegressor class from scikit-learn library. It’s time to fit the tuned model, meaning we are training the tuned model.

* best\_params\_ = {'n\_estimators': 200,

'min\_samples\_split': 5,

'min\_samples\_leaf': 1,

'max\_features': 'sqrt',

'max\_depth': 15}

* best\_score\_ = -5993716.07

## 5. Hyper Tuned Model Evaluation:

**5.1 R-squared (R²) on Train Data**

calculated R-squared score for the training data is approximately **0.976**

About 97.6% of the variance in the actual target values (from your training dataset) can be explained by my regression model.

**5.2 R-squared (R²) on Test Data**

The value we have calculated for r2\_test is approximately **0.892**.An R-squared of **0.892**means that around **89.2%** of the variance in the actual target values (from your test dataset) can be explained by my regression model

**5.3 MAE and RMSE of the model is**

* MAE: 1618.631
* RMSE: 2757.04

# Phase 5: Predicted car price – Power BI Analysis

[TOP](#TOC)

## 1. Actual car price vs Predicted car price Analysis.

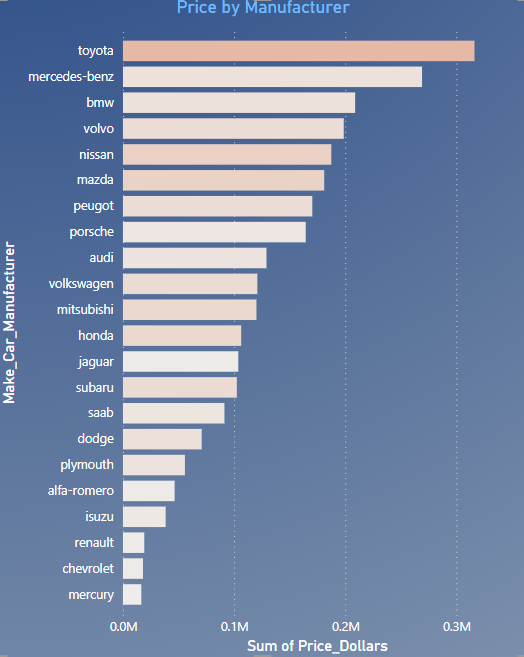
### **1.1 KPI - Key performance parameter of complete Data.**

* **Total Numbers of car manufacturers**- there are total 22 car Brands – manufactures Involved in this Data set.
* **Total Number of cars** – total 205 vehicles are there in data set.
* **Sum of all cars Price** –total Cost of all Cars in data set is $2.74 Million.
* **Predicted Price Total** - total predicted Cost of all Cars in data set is $2.69 Million
* **Price Difference (Min) –** The minimum predicted price difference is approximately $4.03 for one particular car. Showing the accuracy of the model
* **Price Difference (Avg)** - The average predicted price difference is around $230.
* **Price Difference (Max)** – the maximum predicted price difference in data set is approximately -$8640 meaning model predicted price was way lower than actual price

### **1.2 Price by Manufacturer**

* **Toyota**: Has the highest total pricing among car manufacturers.
* **Mercedes-Benz**: Follows closely in terms of total prices.
* **Mercury**: Has the Lowest total pricing among all the manufacturers.

This chart visually compares the total prices of cars from different manufacturers. Toyota stands out as having the highest overall pricing.



### **1.3 Actual vs Predicted - Price Comparison**

* **Alfa Romeo**: The actual price is $46,495, while the sum of predicted prices is $47,036.10.
* **Audi**: The actual price is $129,162.60, and the sum of predicted prices is $117,460.79.
* **BMW**: The actual price is $208,950, and the sum of predicted prices is $193,164.37.
* **Chevrolet**: The actual price is $18,021, and the sum of predicted prices is $18,356.25.
* **Dodge**: The actual price is $70,879, and the sum of predicted prices is $71,531.23.
* **Honda**: The actual price is $106,401, and the sum of predicted prices is $105,144.36.
* **Isuzu**: The actual price is $38,439.38, and the sum of predicted prices is $32,720.90.

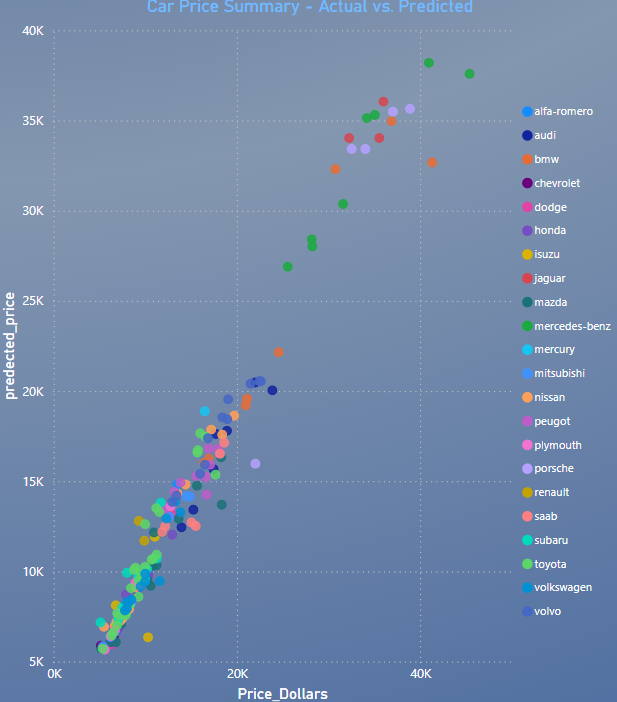
This chart visually compares actual market values with predictive modelling for vehicle pricing across different manufacturers.



### **1.4 Car Price Summary - Actual vs. Predicted - scatter plot**

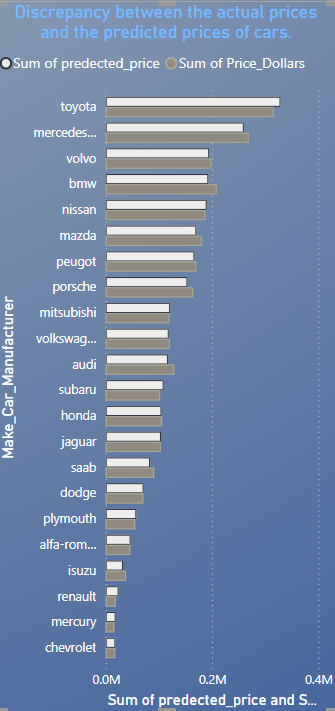
* The chart compares actual prices (on the horizontal axis) with predicted prices (on the vertical axis) for various car brands.
* Each coloured dot represents a specific car brand.
* Data points close to the diagonal line indicate accurate predictions, where actual and predicted prices match closely.
* Some brands show consistent predictions, while others have more variability.

This scatter plot helps visualize how well the prediction model performs across different car manufacturers in terms of pricing accuracy



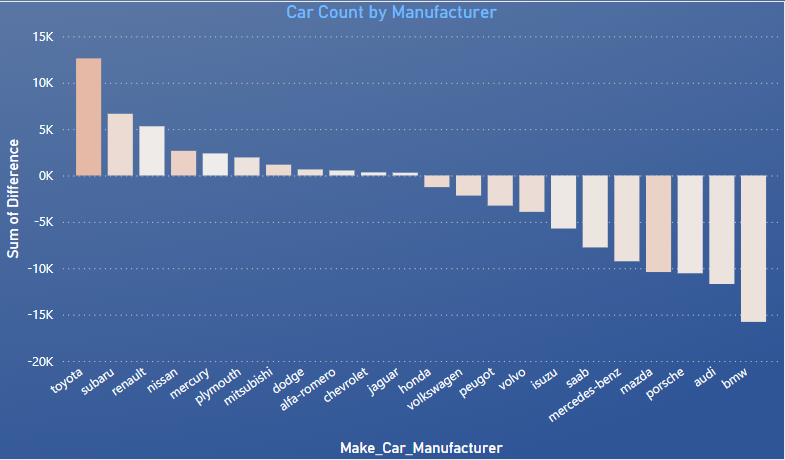
### **1.5 Discrepancy between the actual prices and the predicted prices of cars.**

The bar chart visually illustrates the discrepancy between the actual prices and the predicted prices of cars. Each data point on the plot represents a specific car, and the distance between the point and the diagonal line indicates how closely the predictions align with the real market values.



### **1.6 discrepancies between actual and predicted car prices.**

The bar chart visually represents the discrepancies between actual and predicted car prices. Each car brand is evaluated for the total error in price prediction. Notably, Toyota exhibits the highest positive difference, while BMW demonstrates the maximum negative difference. Interestingly, the price prediction for Jaguar is remarkably accurate, with minimal error.



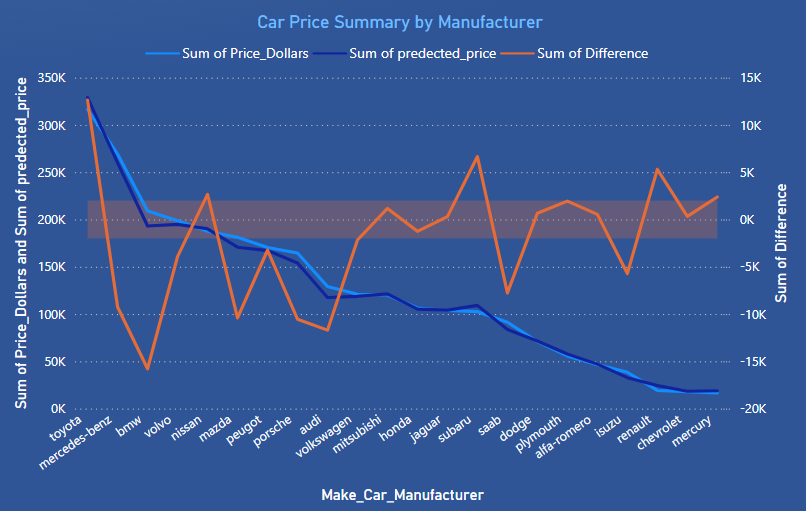
### **1.7 The line chart illustrates three key trends related to car prices:**

1. **Sum of Predicted Price (Dark Blue Line)**: This line represents the cumulative predicted prices for each car brand.
2. **Actual Car Price (Light Blue Line)**: The light blue line corresponds to the actual market prices of cars.
3. **Price Difference (Orange Line)**: The orange line depicts the difference between predicted and actual prices for each brand.

Observations:

* The dark blue and light blue lines move closely together, indicating accurate predictions.
* The orange line shows the maximum difference in car brands with the highest sample sizes.

Overall, the chart provides valuable insights into the accuracy of price predictions across different car manufacturers."



# Phase 6: Conclusion

[TOP](#TOC)

At the culmination of our project, I present my findings on car price prediction using machine learning. Here are the key takeaways:

1. **Linear Regressor Model:**
   * R2 (Test): 0.845
   * MAE (Test) (log-transformed): 0.0752
   * RMSE (Test) (log-transformed): 0.092
2. **Random Forest Regressor Model:**
   * R2 (Test): 0.91
   * MAE (Test): 1681.34
   * RMSE (Test): 2495.72

**Decision:** Based on superior performance metrics, we designate the Random Forest model as our optimal choice. Leveraging its predictions, we conducted an insightful actual versus predicted car price analysis using Power BI.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr#** | **Model** | **R2** | | **R2-Tuned** | | **MAE** | **MAE-Tuned** | **RMSE** | **RMSE-Tuned** |
| **Train** | **Test** | **Train** | **Test** | **Test** | **Test** | **Test** | **Test** |
| **1** | Linear Regression | 0.89 | 0.845 | 0.898 | 0.84539 | 0.0752 | 0.075 | 0.092 | 0.0924 |
|  |
| **2** | Decision Trees | 0.99 | 0.80 | NA | NA | 0.172 | NA | 0.238 | NA |  |
| **3** | Random Forest | 0.98 | 0.91 | 0.98 | 0.904 | 1681.34 | 1515.47 | 2495.72 | 2604.78 |  |
| **4** | Gradient Boosting |  | 0.896 |  | 0.927 | 1902.74 | 1421.78 | 2708.52 | 2261.90 |  |

**Thank you**

for joining me on this data-driven journey. If you have any further questions or wish to explore additional avenues, feel free to reach out.