Car Price Prediction using Regression Model

Abstract

Within the recent decade, car production has progressively expanded. This has given rise to the used automotive market, which has grown into a successful industry in its own right. Using Machine Learning Algorithms such as Linear Regression, Random Forest Regression, Gradient Booster, and MLP Regression, we will aim to build a statistical model that can anticipate the price of a used vehicle based on past customer data and a certain variety of features. We also analyse the estimation accuracy, and it is discovered that Gradient Booster performs the best.

Background

There are three papers related to car price prediction using machine learning algorithms. "An Empirical Study of Car Price Prediction using Regression Analysis" by V. Vinothina and K. Sarukesi (2018) compares Linear Regression, Decision Tree, and Random Forest algorithms for car price prediction and concludes that Random Forest has the best performance. "Prediction of Car Prices using Artificial Neural Networks" by H. A. D. U. Dissanayake and P. G. Gamage (2020) proposes an Artificial Neural Network (ANN) model for car price prediction and compares it with Linear Regression and Random Forest, with the ANN model performing better. "Comparative Analysis of Machine Learning Techniques for Car Price Prediction" by Amjad Ali and Muhammad Arshad Islam (2019) compares six supervised learning algorithms, including Linear Regression, Decision Tree, Random Forest, Gradient Boosting, Support Vector Machine, and Artificial Neural Networks, and finds that Random Forest has the best performance.

Exploratory Data Analysis

The used car data used in this research were collected from www.kaggle.com under the public domain license.

The dataset has 2059 records and 20 characteristics, according to the shape method's analysis of the data's structure. The info() method is used to get a concise summary of the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2059 entries, 0 to 2058
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Make                2059 non-null   object
 1   Model               2059 non-null   object
 2   Price               2059 non-null   int64
 3   Year                2059 non-null   int64
 4   Kilometer           2059 non-null   int64
 5   Fuel Type           2059 non-null   object
 6   Transmission        2059 non-null   object
 7   Location            2059 non-null   object
 8   Color               2059 non-null   object
 9   Owner               2059 non-null   object
 10  Seller Type         2059 non-null   object
 11  Engine              1979 non-null   object
 12  Max Power           1979 non-null   object
 13  Max Torque          1979 non-null   object
 14  Drivetrain          1923 non-null   object
 15  Length              1995 non-null   float64
 16  Width               1995 non-null   float64
 17  Height              1995 non-null   float64
 18  Seating Capacity    1995 non-null   float64
 19  Fuel Tank Capacity  1946 non-null   float64
```

Fig 1. Summary provided of data

Examining the "Non-Null Count" column of the result, it is clear that there are several features with null values. Thus, the data must be cleaned before visualisation , as the pictorial representation might be ambiguous with null values.

Data Cleaning

To eliminate redundancy owing to case sensitivity, all features are changed to lower case as the initial step. The following code is used to perform the task.

```
# to change all the attributes to lowercase
for x in cars_df.columns:
    if type(cars_df.loc[1,x]) is str:
        cars_df[x]=cars_df[x].str.lower()
```

Fig 2. Converting all features to lower case

By using the Missingno library, a bar graph is generated that shows the distribution of NaN values for each feature.

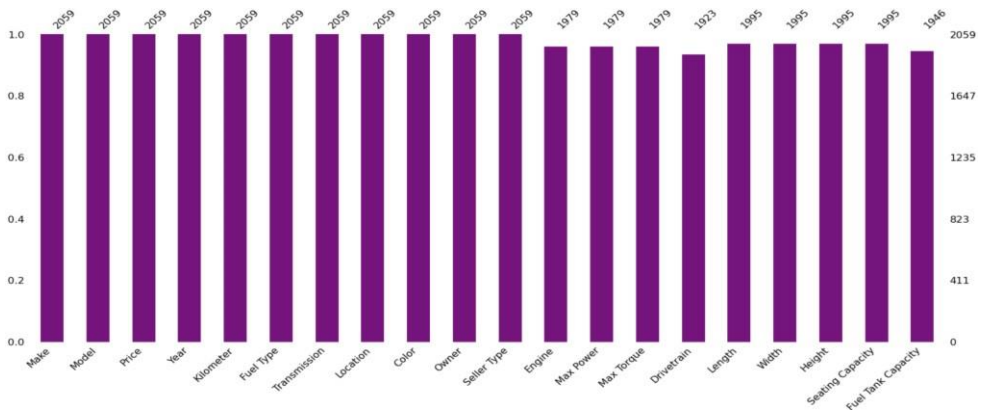

Fig 3. Visualising Null Values

Null values can occasionally be represented by other characters like "?," "-," or just a blank space. To determine if the dataset contains such undesirable characters, following piece of code is run.

```
# to check unwanted characters representing null vales in the dataset
for x in cars_df.columns:
    print(x,':',cars_df[(cars_df[x]=='?') | (cars_df[x]=='') | (cars_df[x]=='-') ][x].count())
```

Fig 4. To check for Ambiguous Characters

Finally, by combining the isnull() and sum() procedures, the total number of nulls in each column is determined.

```
Make                   0
Model                  0
Price                  0
Year                   0
Kilometer              0
Fuel Type              0
Transmission           0
Location               0
Color                  0
Owner                  0
Seller Type            0
Engine                80
Max Power             80
Max Torque            80
Drivetrain           136
Length                64
Width                 64
Height                64
Seating Capacity      64
Fuel Tank Capacity   113
dtype: int64
```

Fig 5. No. of Nulls per Feature

In order to deal with missing values, there are several methods that may be used, including deletion, imputation, prediction, assigning a specific category, and expert knowledge. If the missing numbers are insignificant, deletion can be used. Imputation can improve data retention, but it might add bias if the imputed numbers are wrong. Prediction may be a useful method if the missing data is linked to other elements in the dataset. Expert knowledge can also be utilised to fill in missing data. The approaches take into account the datatype of missing value.

i. A threshold limit is established and any record with more than three null values are deleted.

```
# dropping records with more than 3 null values
cars_df.dropna(axis=0, how='any', thresh=17, subset=None, inplace=True)
```

Fig 6. Threshold elimination

ii. ‘Drivetrain’ has three distinct categories, with FWD being the most prevalent, accounting for 1323 of the 1916 records. Since there are only 63 missing values in the feature, the null values are replaced with the majority class without introducing bias into the dataset. iii.

```
# replacing null values in Drivetrain as FWD
cars_df['Drivetrain'].fillna('fwd',inplace= True)
```

Fig 7. Replacing by majority class

iv. Mean imputation is the most often utilised approach for numerical data types. However, outliners have a significant impact on mean hence, outliners must be checked.

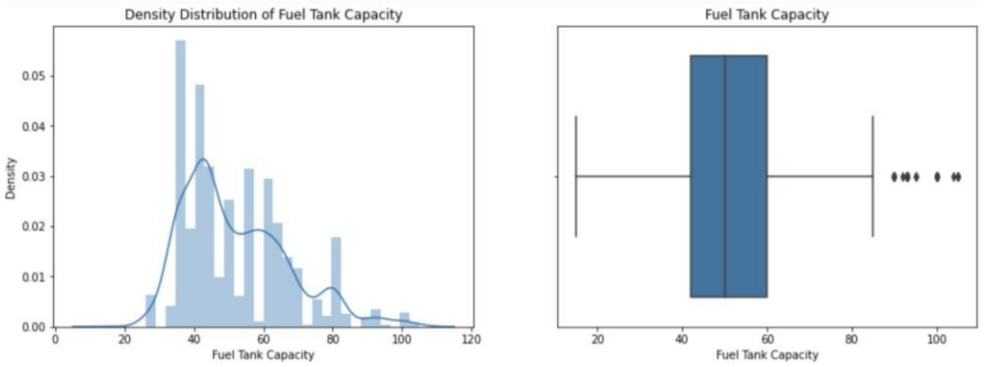

Fig 8. Detecting outliers the feature

The graphs above show that there are few outliners in the feature, however after cross-referencing with resources, it is discovered that the data is valid. Due to high variation, imputation will be an inefficient procedure. Dropping the records with missing values as there are only 42 records with null values.

Duplicates can consume unnecessary storage and also skew data so, data is examined for duplicate records.

Data Visualisation

Visualisation data distribution is essential as it reflects the underlying patterns in the data. When a feature has a high number of different values, expressing everything in a single graph can become complicated. Hence, bar graphs are plotted for features with fewer than 35 unique values.

```
# Distribution graphs of column data
def dataDistPerColumn(cars_df, nGraphShown, nGraphPerRow):
    nunique = cars_df.nunique()
    data = cars_df[[col for col in cars_df if nunique[col] > 1 and nunique[col] < 35]]
    nRow, nCol = data.shape
    columnNames = list(data)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80)
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(3, nGraphPerRow, i + 1)
        columnDf = data.iloc[:, i]
        valueCounts = columnDf.value_counts()
        sns.set(rc={'figure.figsize':(8,6)})
        ax = sns.barplot(x=data.iloc[:, i].unique(), y=data.iloc[:, i].value_counts(),data=cars_df)
        if data.iloc[:, i].nunique() < 10:
            for p in ax.patches:
             ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.55, p.get_height()+0.09), rotation=0, ha='center')

        ax.set( ylabel='Number of Cars')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()

dataDistPerColumn(cars_df,9, 3)
```

Fig 9. Code to visualise features less than 35 unique values

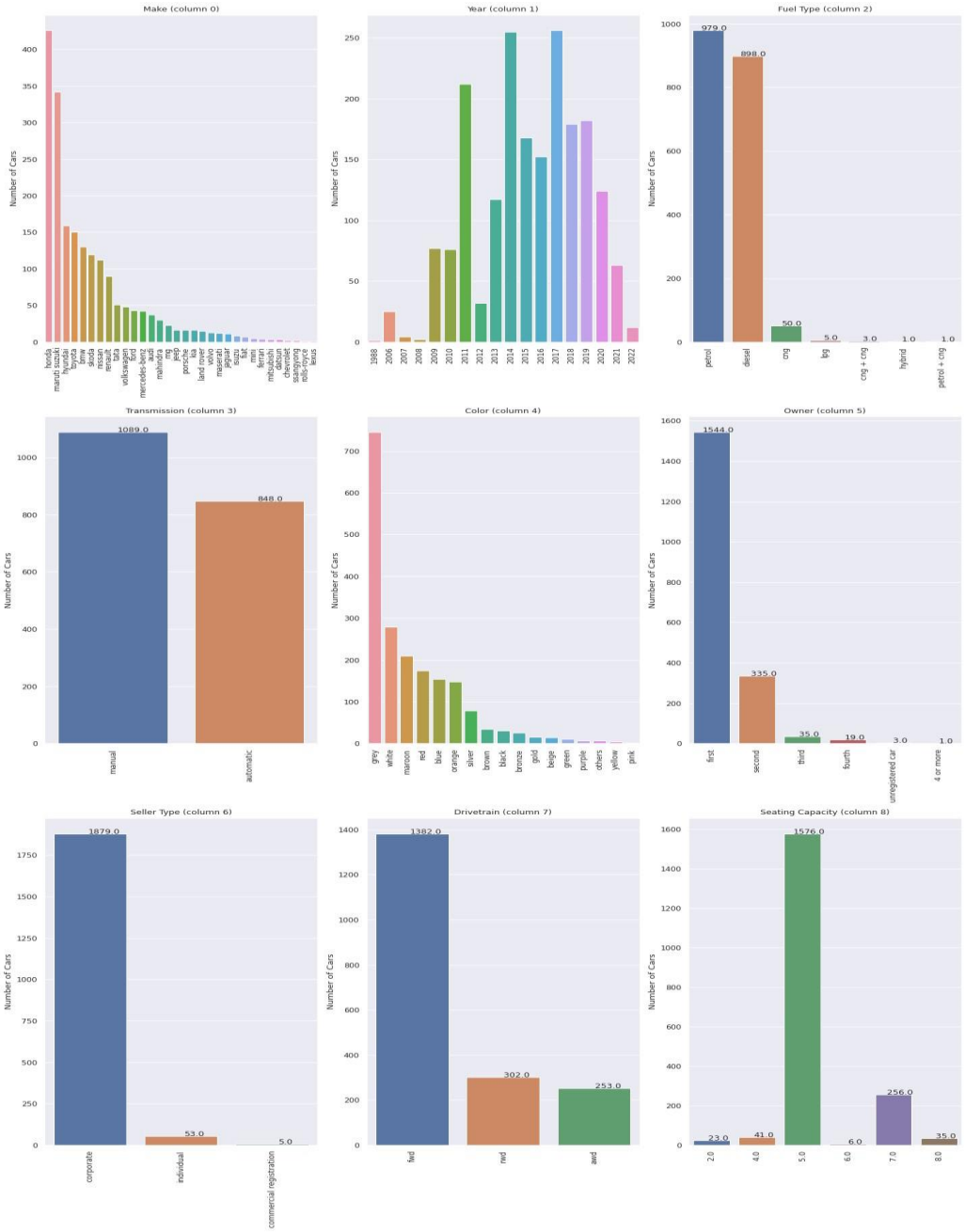

Fig 10. Plots for each feature

Creating a word cloud for ‘Make’ to visualise manufacturing companies recorded.

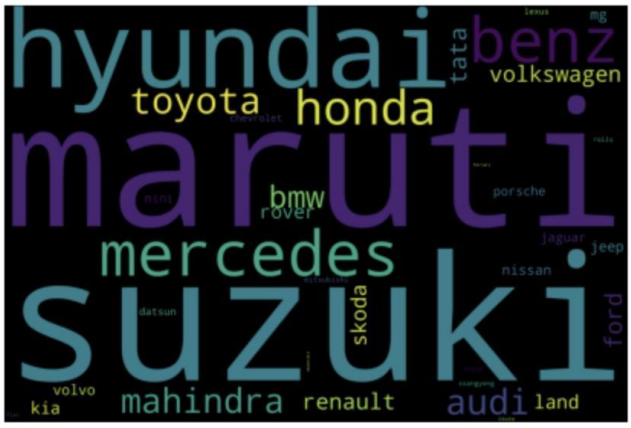

Fig 11. Wordcloud of Manufacturers

To analysing models manufactured per company, ‘Model’ is grouped based on ‘Make’ to find different elements of the feature and a bar graph is plotted to visualise the number of cars manufactured per company. It is as follows.

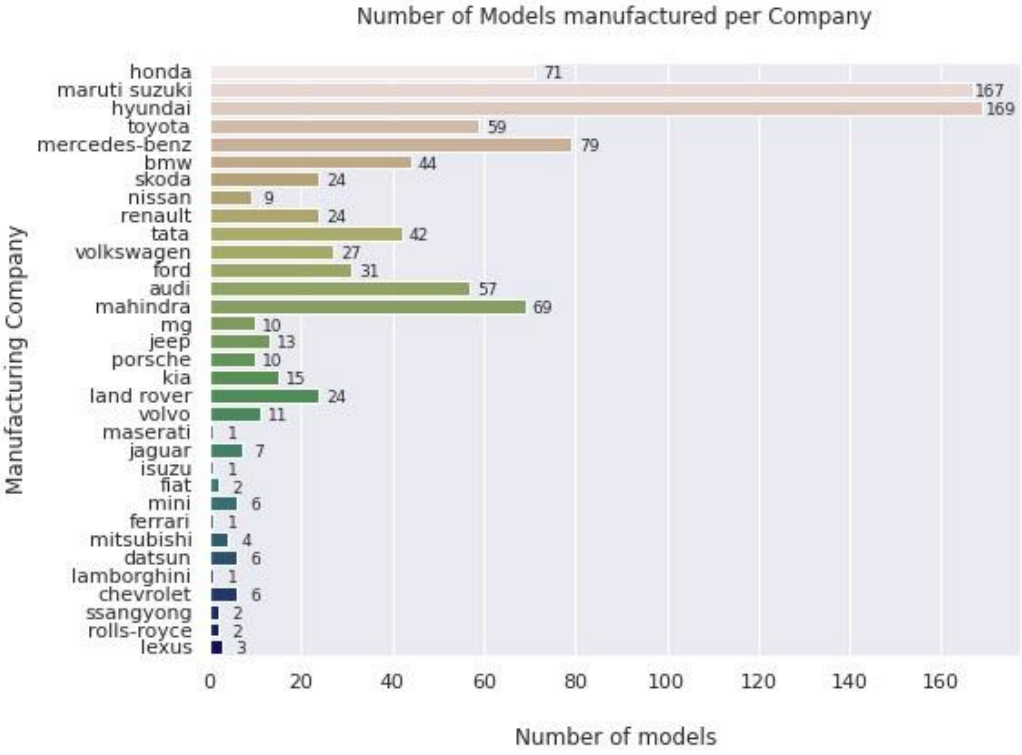

Fig 12. Plot of Cars manufactured per company

Unique() as well as value_counts() is used to analyse features over 35 unique values.

A line graph is generated to illustrate the trend of the target variable over years.

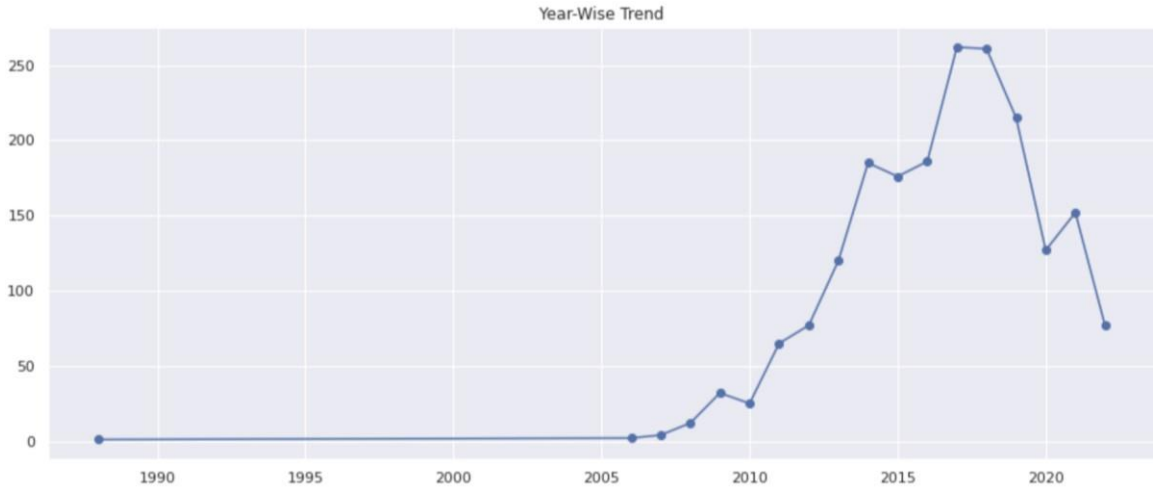

Fig 13. Buying pattern over years

The describe() method is used to do a static examination of all numerical features.

	Price	Year	Kilometer	Length	Width	Height	Seating Capacity	Fuel Tank Capacity
count	1.937000e+03	1937.000000	1.937000e+03	1937.000000	1937.000000	1937.000000	1937.000000	1937.000000
mean	1.679054e+06	2016.552917	5.383137e+04	4272.762003	1764.260196	1590.402168	5.299948	52.018224
std	2.398959e+06	3.247919	5.852369e+04	440.329314	133.208559	135.041507	0.813982	15.109569
min	4.900000e+04	1988.000000	0.000000e+00	3099.000000	1475.000000	1213.000000	2.000000	15.000000
25%	4.849990e+05	2014.000000	2.873200e+04	3985.000000	1695.000000	1485.000000	5.000000	42.000000
50%	8.199990e+05	2017.000000	4.958000e+04	4355.000000	1770.000000	1545.000000	5.000000	50.000000
75%	1.850000e+06	2019.000000	7.200000e+04	4600.000000	1831.000000	1675.000000	5.000000	60.000000
max	3.500000e+07	2022.000000	2.000000e+06	5569.000000	2220.000000	1995.000000	8.000000	105.000000

Fig 14. Statistical summary of data

Outliner Detection

Outliers are observations that are significantly different from the bulk of the other observations in a dataset. Consequently, for each feature, a boxplot is plotted to locate outliers in the data.

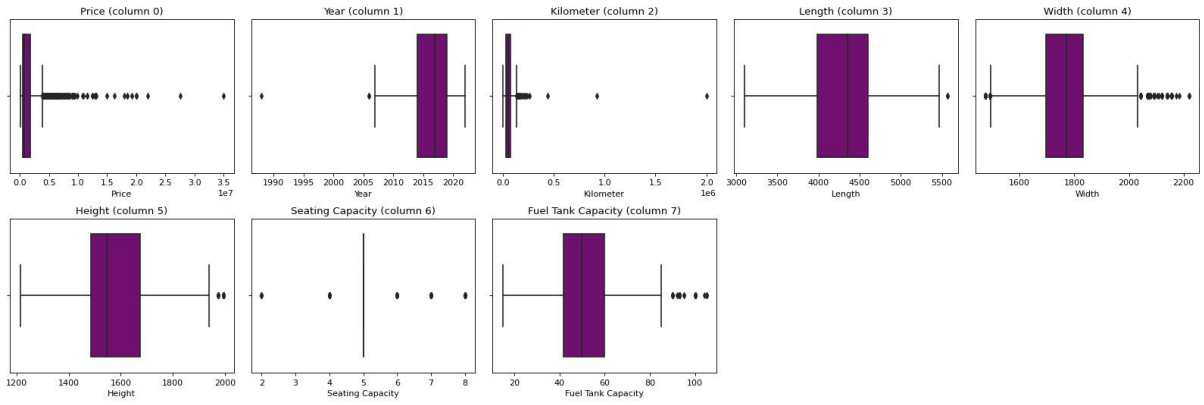

Fig 15. Boxplot per feature

The plots show that there are outliers in multiple features however, before removing or changing the records, it is critical to analyse the outliers. As a result, all entries identified as outliers are crosschecked using domain expertise. It is determined that the records provided are of high-priced automobiles. Since there aren't many such records of high-priced automobiles and the price range differs significantly from other low-priced cars, they are emphasised as outliers. Nonetheless, because the records are legitimate, removing them would be inappropriate.

Data Transformation

Data transformation is done to prepare the data for analysis.

The feature 'Engine' has an undesirable character, 'cc,' and since it's a string, it will be of little use to a regression model in this form. Thus, the numerical value is extracted while the remainder is ignored. Similarly, using the code below, relevant numerical values are retrieved from the 'max power' and 'max torque' attributes.

```
# extract the numerical values using str.extract()
cars_df["Engine"] = (cars_df["Engine"].str.extract(r'(\d+)')).astype(int)
# extract the numerical values using str.extract()
cars_df["Max Power"] = (cars_df["Max Power"].str.extract(r'(\d+)')).astype(int)
# extract the numerical values using str.extract()
cars_df["Max Torque"] = (cars_df["Max Torque"].str.extract(r'(\d+)')).astype(int)
```

Fig 16. Extracting numerical values from string Further, calculating ‘Age’ column based on the ‘Year’ feature as it will be more useful for a regression model.

```
# calculating age of the cars
cars_df["Age"]=cars_df["Year"].apply(lambda x: 2023-x)
```

Fig 17. Adding ‘Age’ feature

Feature Selection

Feature selection aims to enhance the performance of a model by selecting the best features by dropping irrelevant, redundant, or noisy features.

The pair plot is the most frequent method for determining feature dependence. It shows the correlation between two attributes in a dataset. It is a scatterplot matrix, with each plot displaying the connection between two features. The following is the pair plot generated for our dataset.

Fig 18. Pair plot for the dataset

Pair plot gives the relation between numerical features. For analysing the relationship between categorical features and target variable, each categorical feature is analysed based by plotting a scatter plot for each feature. The resultant plots are as follows.

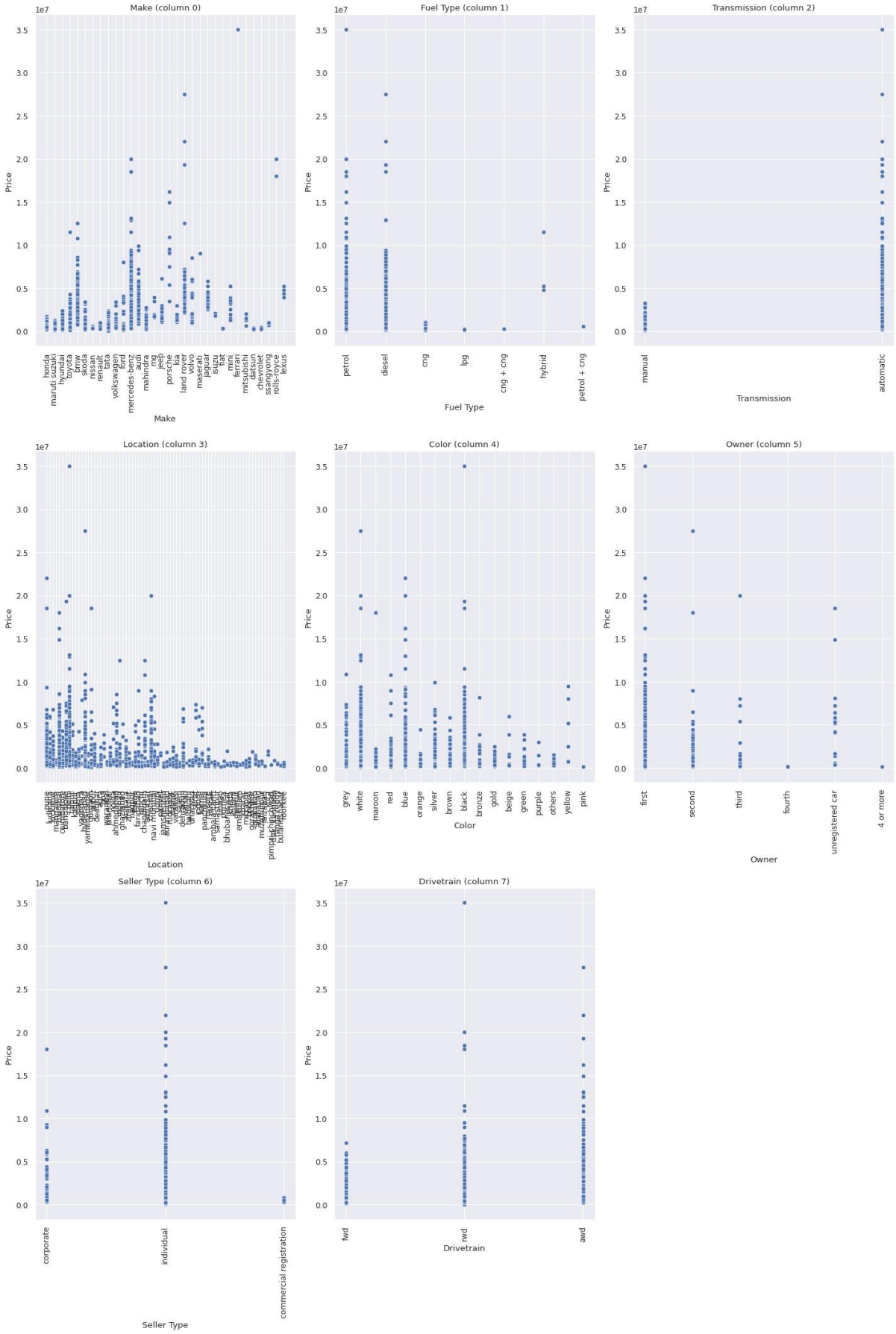

Fig 19. Relationship between categorical features and Price

The features model, make, location, and colour do not exhibit significant variance with regard to the target variable according to the observations from the pair plot and the preceding scatter plots. Moreover, the age feature more effectively conveys the same information as year. Thus, dropping these features will reduce the computational load without affecting the performance of the models. This helps preventing overfitting and making the model more generalized.

```
# dropping model name
cars_df.drop(["Model"], axis = 1, inplace = True)

# dropping location feature
cars_df.drop(["Location"], axis = 1, inplace = True)

# dropping feature color
cars_df.drop(["Color"], axis = 1, inplace = True)

# dropping feature make
cars_df.drop(["Make"], axis = 1, inplace = True)

# dropping feature year
cars_df.drop(["Year"], axis = 1, inplace = True)
```

Fig 20. Droping mentioned features

The string characteristics 'Fuel Type', 'Seller Type', 'Transmission' ,'Owner,' and 'Drivetrain' cannot be utilised for regression since regression models require numerical data to generate predictions. Thus it is necessary to convert these features into numbers. This can be accomplished by converting category data into numerical data using encoding techniques as one-hot encoding, label encoding, or manual encoding. Manual and label encoding may not work well for features with a large number of unique values because they can establish an ordinal relationship between categories that does not always exist. Hence performing one-hot encoding for features more than 3 unique values.

```
# mannual encoding
cars_df.replace({'Transmission':{'manual':0,'automatic':1}}, inplace=True)

# label encoding
cars_df['Seller Type'] = LabelEncoder().fit_transform(cars_df['Seller Type'])

# label encoding
cars_df['Drivetrain'] = LabelEncoder().fit_transform(cars_df['Drivetrain'])


# one hot encoding using get_dummies function
cars_df=pd.get_dummies(cars_df,columns=['Fuel Type','Owner'],drop_first=True)
```

Fig 21. Encoding categorical features

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1937 entries, 0 to 2057
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Price                   1937 non-null   int64
 1   Kilometer               1937 non-null   int64
 2   Transmission            1937 non-null   int64
 3   Seller Type             1937 non-null   int64
 4   Engine                  1937 non-null   int64
 5   Max Power               1937 non-null   int64
 6   Max Torque              1937 non-null   int64
 7   Drivetrain              1937 non-null   int64
 8   Length                  1937 non-null   float64
 9   Width                   1937 non-null   float64
 10  Height                  1937 non-null   float64
 11  Seating Capacity        1937 non-null   float64
 12  Fuel Tank Capacity      1937 non-null   float64
 13  Age                     1937 non-null   int64
 14  Fuel Type_cng + cng     1937 non-null   uint8
 15  Fuel Type_diesel        1937 non-null   uint8
 16  Fuel Type_hybrid        1937 non-null   uint8
 17  Fuel Type_lpg           1937 non-null   uint8
 18  Fuel Type_petrol        1937 non-null   uint8
 19  Fuel Type_petrol + cng  1937 non-null   uint8
 20  Owner_first             1937 non-null   uint8
 21  Owner_fourth            1937 non-null   uint8
 22  Owner_second            1937 non-null   uint8
 23  Owner_third             1937 non-null   uint8
 24  Owner_unregistered car  1937 non-null   uint8
dtypes: float64(5), int64(9), uint8(11)
memory usage: 312.3 KB
```

Fig 22. Structure of data after encoding

To avoid overfitting, the data must first be divided into train and test sets before proceeding with feature selection. Feature selection is performed on training data, and the same procedures are performed on test data without learning from it again. Here, the data is divided into two categories: train data (80%), and test data (20%).

The correlation of characteristics is a different approach to feature selection. A statistical metric called Pearson correlation, which ranges from -1 to +1, measures the linear connection between two variables. This measure is used to identify and remove highly correlated features from a dataset. We initially determine the correlation matrix for each feature in the dataset to find highly associated features.

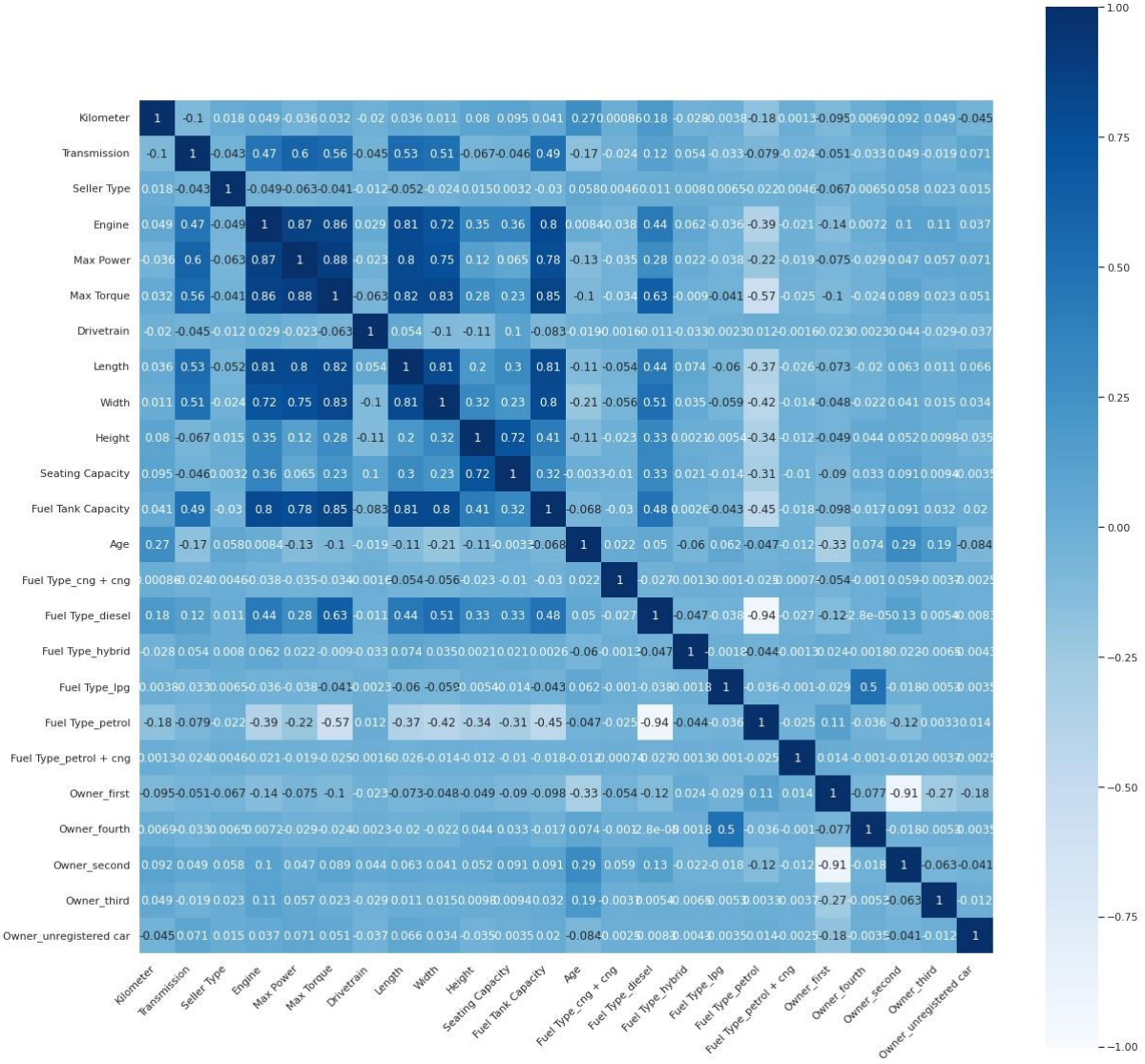

Fig 23. Correlation matrix of Dataset

Further, creating a function to recognize and store features in the dataset whose correlation is above a specified threshold limit. The above mentioned is called and the training dataset without target variable(X_train) and a threshold of 0.85 is passed to the function. The function returns a list of features with correlation more than the threshold specified. This function considers absolute values as features with value close or equal to -1 means they convey the same information just in opposite direction. Therefore correlation values such as -90 and 90 is treated as the same.

```
# selecting highly correlated columns and remove the first feature that is highly correlated with any othre feature

def corr_check(data,thresh):
  col_cor = set()
  cor_matrix=cars_df.corr()
  for i in range(len(cor_matrix.columns)):
      for j in range(i):
          if abs(cor_matrix.iloc[i,j]) > thresh:
              colname = cor_matrix.columns[i]
              col_cor.add(colname)
  return col_cor

corr_features=corr_check(X_train,0.85)
```

Fig 24. Identifying highly correlated features

The highly correlated features are dropped from both train and test data. The following are the new shapes of X_train and X_test.

```
# shape of train and test data after dropping feature based on correlation
print('Shape of X train: ',X_train.shape)
print('Shape of X test: ',X_test.shape)
```

```
Shape of X train:  (1355, 19)
Shape of X test:  (582, 19)
```

Fig 25. New shapes of train and test data

Identifying feature importance using a random forest regressor is a technique for determining which features in a dataset have the most influence on the target variable. The random forest regressor is a machine learning technique that builds numerous decision trees from random selections of input attributes and then combines their predictions to get a final output. The importance of a feature is estimated as the decrease in impurity when the feature is utilised to divide the data in the decision tree.

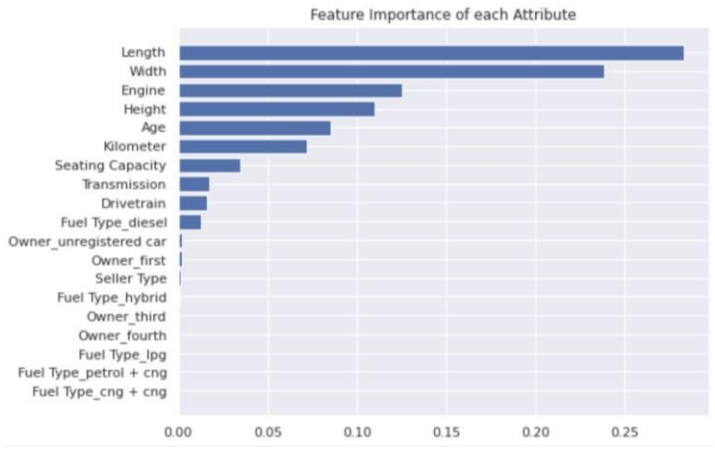

Fig 26. Feature importance per attribute

```
# shape of train and test data after dropping feature based on feature importance
print('Shape of X train: ',X_train.shape)
print('Shape of X test: ',X_test.shape)
```

```
Shape of X train:  (1355, 9)
Shape of X test:  (582, 9)
```

Fig 27. New shapes of train and test data

The bottom 10 features based on the feature importance are dropped from both train and test data. According to feature importance, the worst 10 features are eliminated from the train and test datasets.

Mutual information may be utilised to identify which features are more informative or predictive of the target variable. ‘mutual_info_regression is a function used to calculate the mutual information between a set of features (X_train) and a target variable (y_train).

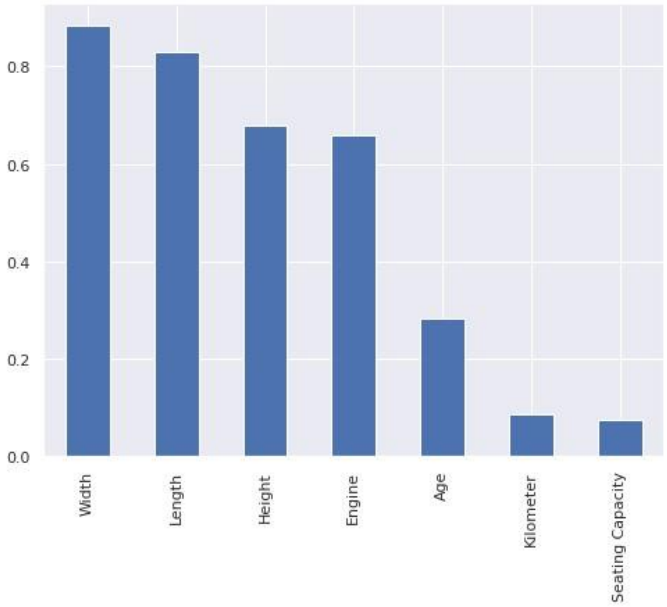

Fig 28. Mutual Information per attribute

No feature is discarded at this step since there are only 7 features left in the dataset and each feature contributes to the prediction of the target variable. Removing more features may result in data loss.

Model Building

Cross-validation is a statistical approach that divides available data into two groups. While the training set is used to train the model, the validation set is used to assess its performance. This procedure is repeated numerous times, using different subsets for training and validation. The results are then averaged to obtain an estimate of the model's performance. This increases the accuracy and dependability of the model's predictions by providing more accurate approximation of the model's real performance on unseen data. Thus, cross validation technique is employed for in the project to boost the model performance. Here, three models have been developed using this method. i. Linear Regression Model

Linear regression models the relationship between a dependent variable and one or more independent variables assuming a linear relationship between them, to make predictions based on the values of the independent variables. The model is trained for 10 folds and later tested on the remaining 20% of test data. The statistics of module evaluation are as below.

Fig 29. Bar chart of R2 score of each iteration

```
R2 score on the test set: 0.7295134802238409
Mean Squared Error on the test set: 204857078442.65234
Mean Absolute Error on the test set: 323129.37097357796
```

Fig 30. Model Evaluation Scores

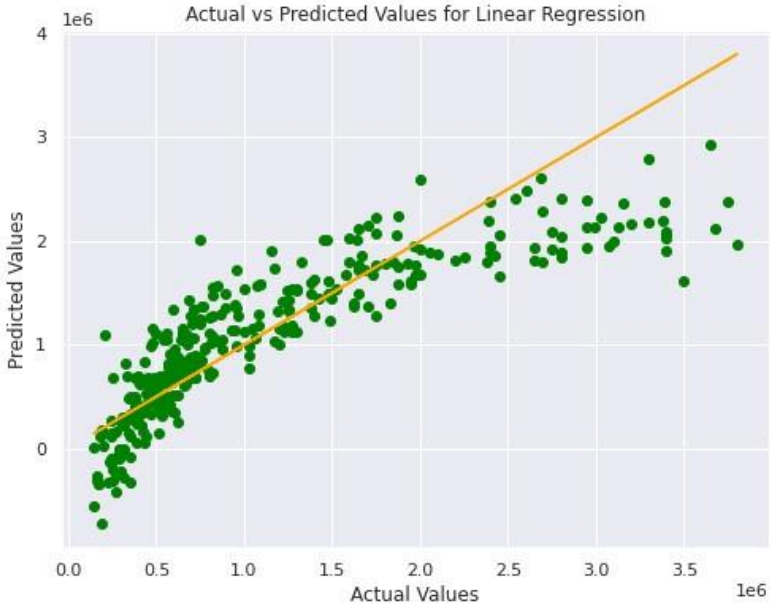

Fig 31. Visualising variance between actual and predicted values

It is noted that the model performs well for low price ranges but predicts high costs poorly. This is due to the fact that the data consists only few records with high prices.

ii. Random Forest Regressor

Random Forest Regressor is known to be a robust regression model that can handle outliers quite well. This is because Random Forest Regressor is an ensemble model that combines multiple decision trees to make predictions. To avoid overfitting of the model, cross validation is performed on the training set and finally the results are tested on unseen test set. The statistics of module evaluation are as below.

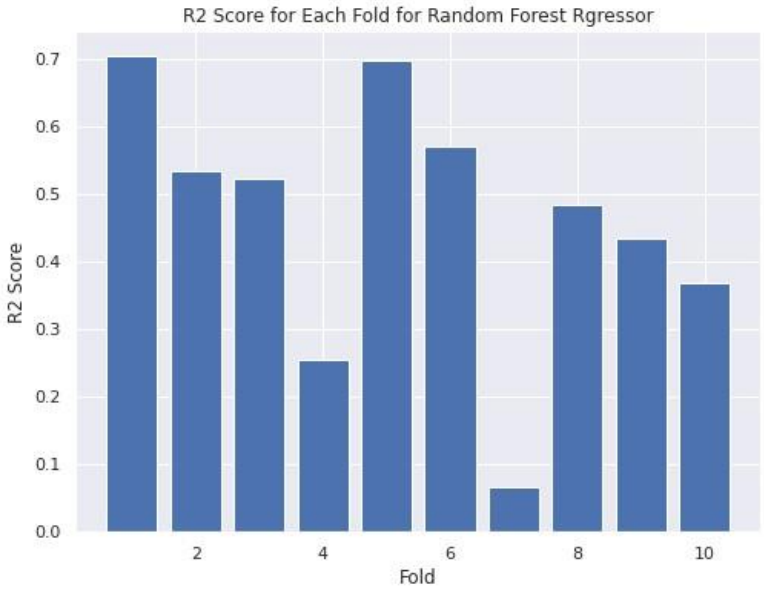

Fig 26. Bar chart of R2 score of each iteration

```
R2 score on the test set: 0.9249165376068673
Test set Mean Absolute Error: 307079964830.1805
Test set Mean Absolute Error: 256927.93424275893
```

Fig 27. Model Evaluation Scores

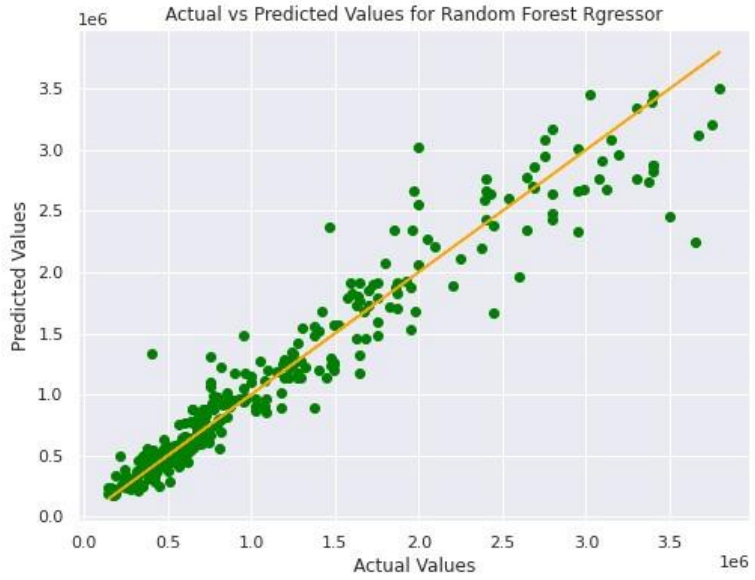

Fig 32. Visualising variance between actual and predicted values

iii. Gradient Booster Regressor

Gradient boosting is a machine learning algorithm that combines weak prediction models, often decision trees, to create more accurate predictions. The technique operates by adding new models in a sequential fashion that focus on the flaws of prior models, with the objective of lowering the overall prediction error. Gradient boosting trains models on the residual errors of prior models, which teaches them to identify the difference between the real value and the predictions of past models. Cross validation is done to enhance the performance of model. The statistics of module evaluation are as below.

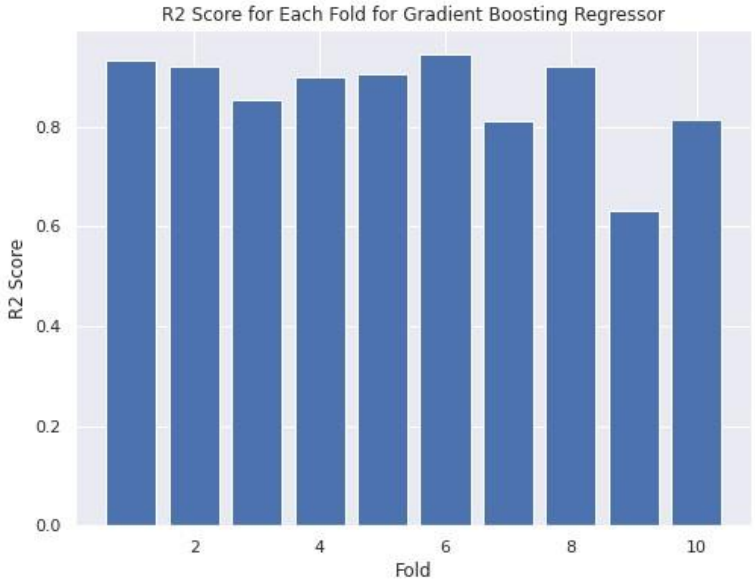

Fig 33. Bar chart of R2 score of each iteration

```
R2 score on the test set: 0.930322824300511
Test set Mean Squared Error: 284969072833.0389
Test set Mean Absolute Error: 282105.27546096296
```

Fig 34. Model Evaluation Scores

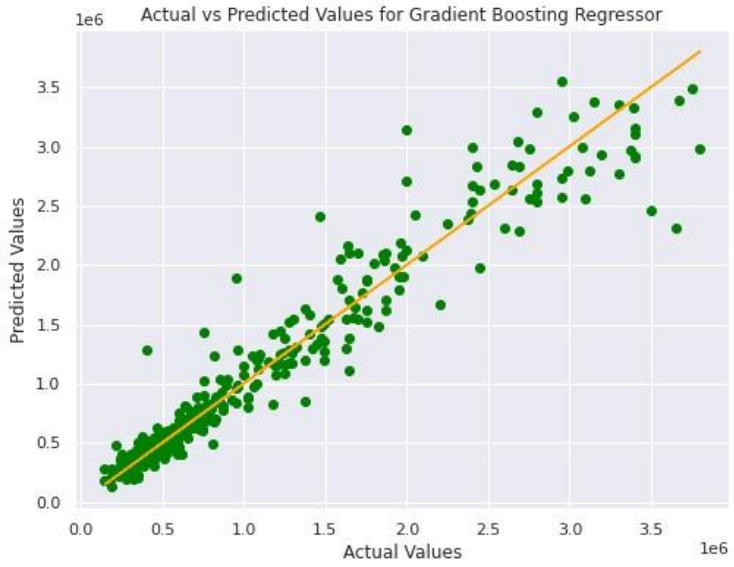

Fig 35. Visualising variance between actual and predicted values

iv. Artificial Neural Network

Artificial neural networks (ANNs) may be trained on input-output pairs to understand the underlying mapping between them and then utilised for regression tasks. One or more hidden layers of neurons in the network design generally change the input data to generate the output. ANNs may learn to approximate extremely nonlinear functions it's main advantage being that it can work with a diverse set of input and output data. ANNs are sensitive to input feature scale, a features with wider range of values may dominate the training and bias the predictions. Scaling prevents such unnecessary bias and aids the optimization algorithm in converging to a minimum of the cost function as rapidly as possible. This enhances the model's ability to generalise to new data and prediction accurately. The network architecture is defined, including the number of layers, the number of neurons in each layer, and the activation functions. Cross validation is not performed in this case as it requires high computational power. The statistics of module evaluation are as below.

```
R2 score on the test set: 0.806174761723165
Test set Mean Squared Error: 146796491280.60956
Test set Mean Absolute Error: 288422.4630229563
```

Fig 36. Model Evaluation Scores

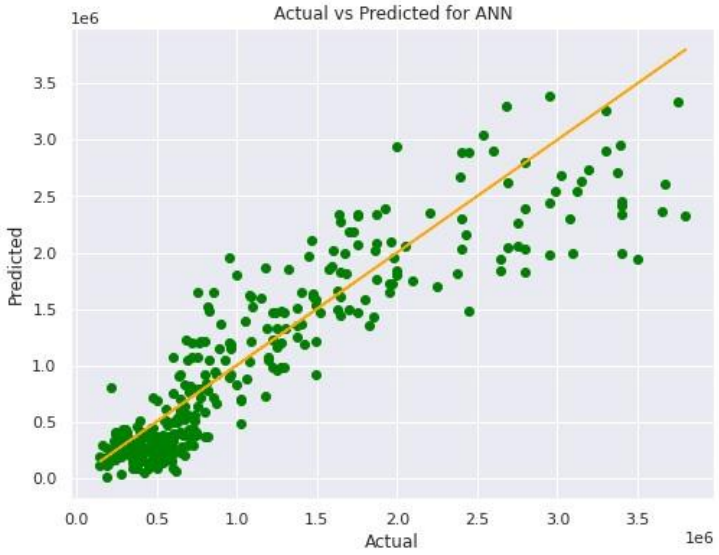

Fig 37. Visualising the variance between actual and predicted values

There was a performance decrease of linear regression model due to high variation in prices and low number of records for high range cars, thus dividing data based on priced to test the difference in performance. Although the division was aimed to improve the performance of linear regression model, there is no significant improvement in the predictions. Rather, the model performed extremely poorly in case of high ranged cars.

Another way of splitting the data set could be by using clustering algorithms probably kmeans clustering as it is distance based algorithm and gives freedom to choose the number of clusters to be formed. This will be a more robust method of splitting the data and thus can be used even if the records vary as you do not need to hardcode the condition of splitting the records. The issue of imbalanced data can be resolved by an alternative method of using SMOTE which creates dummy records of minority class to balance the data.

Results

During the cross validation process, models gives out multiple outputs thus analysing the performance of each model using boxplots.

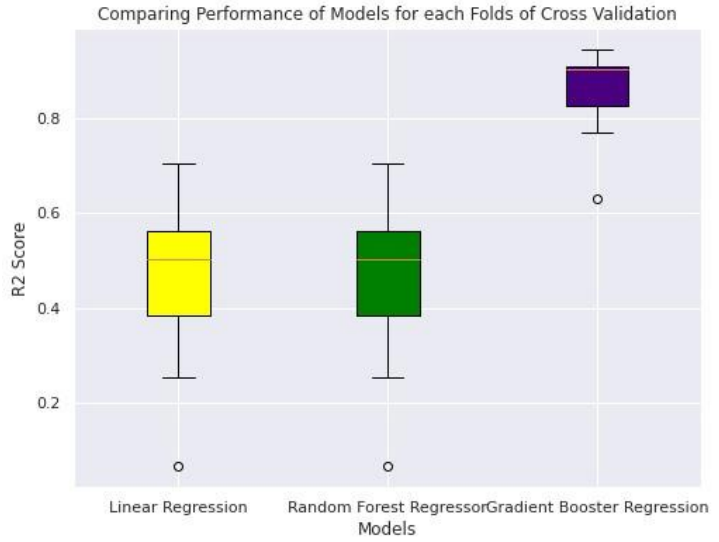

Fig 38. Performance of models during Cross Validation

The Gradient Booster Regressor outperforms all other models. The model is consistent and performs well with this dataset. The performance with test sets are visualised below.

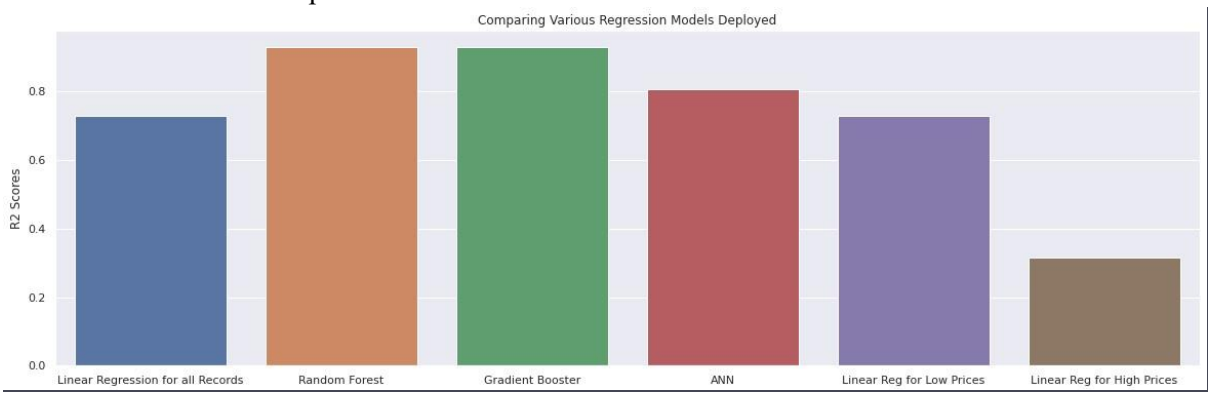

Fig.39 Comparing all Regression Models

It can be noted that Gradient Booster continues to be the best performer even with the test data indicating it is not overfitted. Random forest although gives average results with train data performs well with test data.

Hyper Parameter Tuning

Hyperparameter tuning is a strategy for determining the optimal set of hyperparameters for a machine learning model by exhaustively searching through a defined range of hyperparameters using crossvalidation. GridSearchCV may be a computationally expensive procedure, particularly for models with a high number of hyperparameters or huge datasets thus, demonstrating this method only to identify best hyperparameters for Random Forest Regressor as we see it performs average for train data but gives good results during testing phase.

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best R2 Score on Validation Set: 0.8122467924760919
Test set R2 Score: 0.927861794370295
Test set Mean Squared Error: 295034311706.2785
```

Fig 40. Optimum Parameters returned by GridSearchCV

Conclusion

In conclusion, the Gradient Booster Algorithm beats all other models in this situation; nevertheless, there are various methods presented in the study that may be employed to increase the performance of other models. **References**

Ali, A. and Islam, M.A., 2019. A comparison of machine learning techniques for car price prediction. International Journal of Advanced Computer Science and Applications, 10(7), pp.57-63.

Vinothina, V. and Sarukesi, K., 2018. An empirical study of car price prediction using regression analysis. International Journal of Engineering and Technology, 7(4), pp.208-212.

Dissanayake, H.A.D.U. and Gamage, P.G., 2020. Prediction of Car Prices using Artificial Neural Networks. International Journal of Advances in Scientific Research and Engineering, 6(1), pp.61-65.