PREDICTING USED CARS PRICES

CONTENTS

1. INTRODUCTION:

The world's largest collection of used vehicles for sale resides on Craigslist, presenting a unique challenge to collect and analyze this vast dataset. This project aims to address the challenge of predicting used car prices using a supervised machine learning approach. Predicting the price of pre-owned vehicles is a crucial task for both sellers and buyers in the used car market. The complexity lies in the varying factors influencing a used car's value, including manufacturer, model, production year, condition, fuel type, transmission, and other attributes. This report explores the background, motivation, and data associated with the project.

2. BACKGROUND:

Craigslist serves as a vast repository of used vehicle listings, providing a rich source of data for understanding pricing patterns in the used car market. The challenge is to synthesize this wealth of information into a predictive model that can assist car dealerships and online marketplaces in optimizing their inventory and pricing strategies.

3. MOTIVATION:

The motivation behind this project arises from the inherent complexity of pricing used vehicles. The value of a used car is influenced by a myriad of factors, making it a challenging task for both sellers and buyers to determine a fair price. Car dealerships and online marketplaces can benefit from accurate price predictions to optimize their inventory. By ensuring the right mix of vehicles at prices aligned with market demand, businesses can maximize profitability and competitive advantage. Sellers can also utilize predicted prices to develop effective pricing strategies, attracting potential buyers while optimizing revenue.

4. DATA:

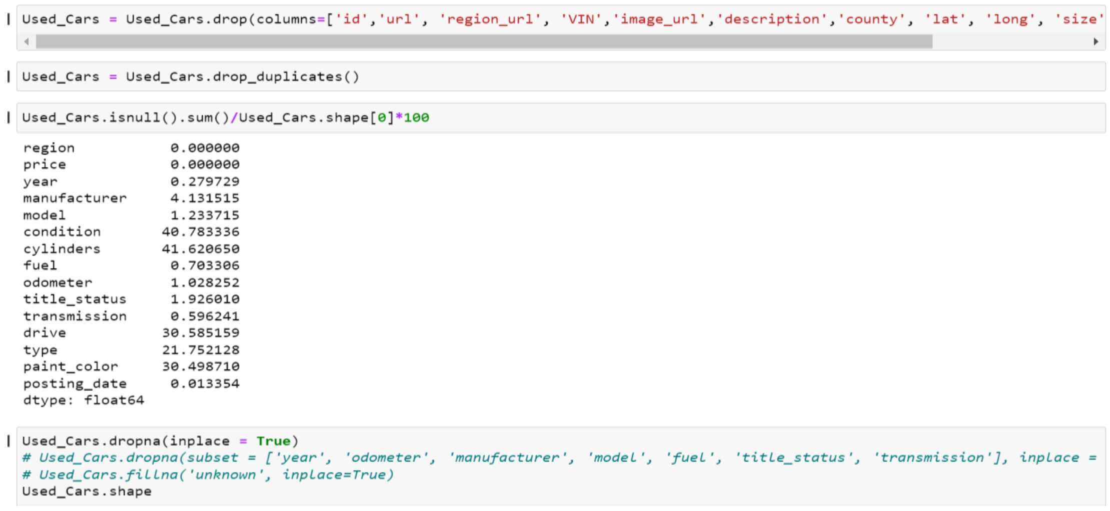
The dataset used for this project is a robust collection of used vehicle listings from Craigslist within the United States. It comprises 426,880 rows and 26 columns, including key information such as below: (each block represents a column)

ID: Entry ID	URL: Listing URL	Region: Craigslist region	Price: Entry price	Year: Entry year	Image_url: Image URL
Model: Model of the vehicle	Condition: Condition of the vehicle	Cylinders: Number of cylinders	Fuel: Fuel type	Odometer: Miles traveled by the vehicle	Drive: Type of drive

State: State of the listing	Lat: Latitude of the listing	Long: Longitude of the listing	Posting_date: Timestamp	County: Useless column left in by mistake	Description: Listed description of the vehicle
Size: Size of the vehicle	Transmission: Transmission type	VIN: Vehicle identification number	Type: Generic type of vehicle	Paint_color: Color of the vehicle	

5. Data Cleaning and Preprocessing

Reviewed and dropped irrelevant columns such as 'id', 'url', and 'VIN' to focus on essential features. Additionally, removed duplicate entries to ensure data integrity.
Handling Missing Values: Checked for missing values and opted to drop rows with any missing data, ensuring a clean dataset for subsequent analysis. After data cleaning, have 79991 rows and 15 columns.



```
Used_Cars = Used_Cars.drop(columns=['id','url', 'region_url', 'VIN','image_url','description','county', 'lat', 'long', 'size'
```

```
Used_Cars = Used_Cars.drop_duplicates()
```

```
Used_Cars.isnull().sum()/Used_Cars.shape[0]*100
```

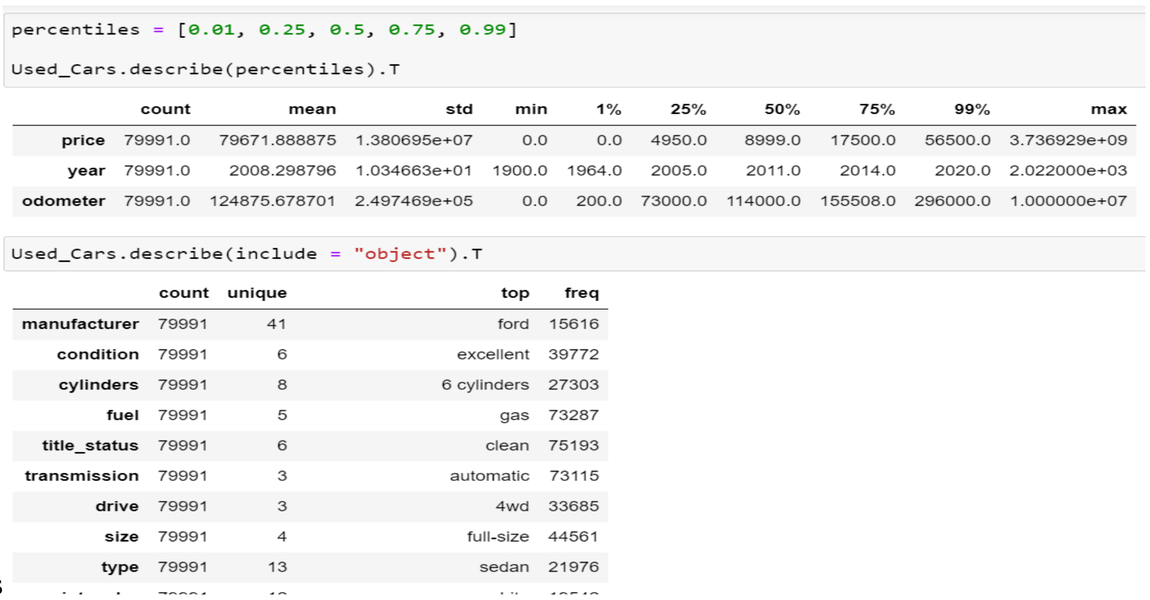
```
region           0.000000
price            0.000000
year             0.279729
manufacturer     4.131515
model            1.233715
condition       40.783336
cylinders       41.620650
fuel             0.703306
odometer         1.028252
title_status     1.926010
transmission     0.596241
drive           30.585159
type            21.752128
paint_color     30.498710
posting_date     0.013354
dtype: float64
```

```
Used_Cars.dropna(inplace = True)
# Used_Cars.dropna(subset = ['year', 'odometer', 'manufacturer', 'model', 'fuel', 'title_status', 'transmission'], inplace =
# Used_Cars.fillna('unknown', inplace=True)
Used_Cars.shape
```

fig 1.1 data cleaning

5.1.Descriptive Statistics:

For the numerical features in the dataset, executed a statistical analysis to derive descriptive statistics focusing on specific percentiles of interest. The chosen percentiles, namely 1st percentile (0.01), 25th percentile (Q1 or 0.25), median (50th percentile or 0.5), 75th percentile (Q3 or 0.75), and 99th percentile (0.99), provide insights into various aspects of the data distribution.



```
percentiles = [0.01, 0.25, 0.5, 0.75, 0.99]
Used_Cars.describe(percentiles).T
```

	count	mean	std	min	1%	25%	50%	75%	99%	max
price	79991.0	79671.888875	1.380695e+07	0.0	0.0	4950.0	8999.0	17500.0	56500.0	3.736929e+09
year	79991.0	2008.298796	1.034663e+01	1900.0	1964.0	2005.0	2011.0	2014.0	2020.0	2.022000e+03
odometer	79991.0	124875.678701	2.497469e+05	0.0	200.0	73000.0	114000.0	155508.0	296000.0	1.000000e+07

```
Used_Cars.describe(include = "object").T
```

	count	unique	top	freq
manufacturer	79991	41	ford	15616
condition	79991	6	excellent	39772
cylinders	79991	8	6 cylinders	27303
fuel	79991	5	gas	73287
title_status	79991	6	clean	75193
transmission	79991	3	automatic	73115
drive	79991	3	4wd	33685
size	79991	4	full-size	44561
type	79991	13	sedan	21976

us

fig 1.2 descriptive statistics

The resulting statistics encompass measures such as mean, standard deviation, minimum, maximum, and quartiles. Furthermore, the specified percentiles aid in identifying potential outliers or extreme values at critical points in the distribution. This analysis is crucial in gaining a nuanced understanding of the central tendency, spread, and distribution characteristics of the numerical features

6. EDA (Exploratory Data Analysis)

6.1 Outlier Removal

Outliers in the 'price' and 'odometer' columns underwent identification and subsequent removal through the utilization of the Interquartile Range (IQR) method. This meticulous step was taken to bolster the resilience of the predictive model by eliminating extreme values that could potentially skew the results.

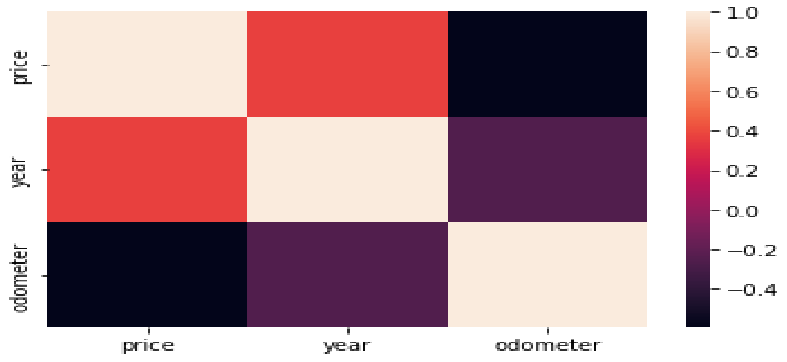


fig 1.4 heat map

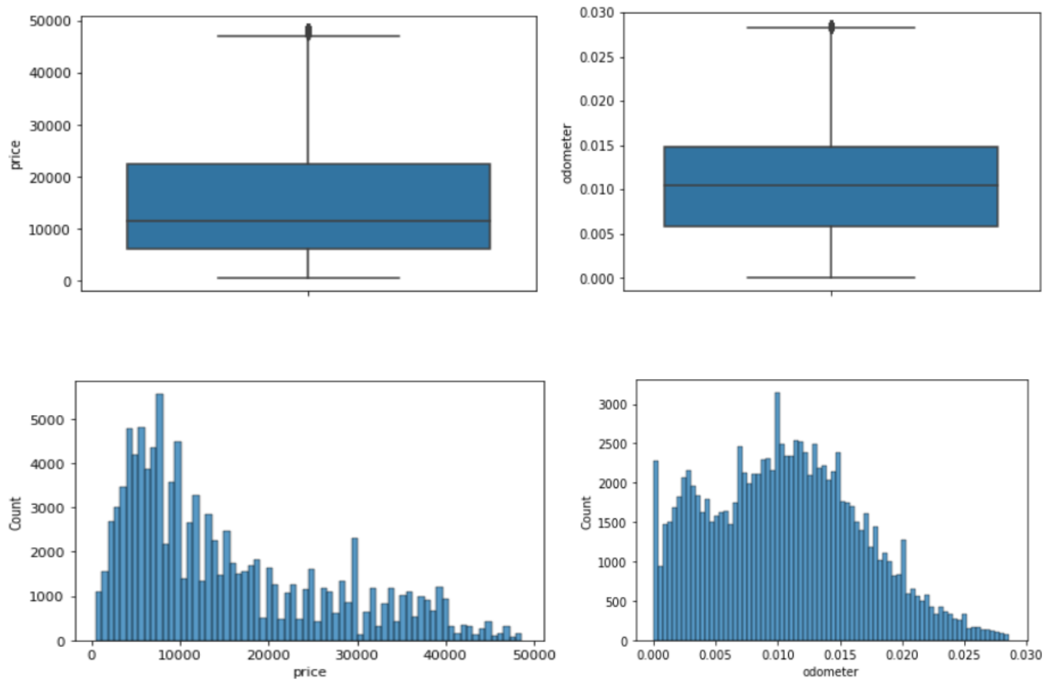


fig 1.3 box plots of price and odometer

6.2 Feature Engineering

In the pursuit of a more enriched dataset, embarked on feature engineering endeavors. The 'car_age' feature was ingeniously crafted by leveraging the 'posting_date' and 'year' columns. Furthermore, the 'posting_date' column underwent a granular transformation, being split into 'posting_date' and 'posting_time.' The 'condition' column, representing the state of the car, was artfully encoded using Ordinal Encoding, thereby transforming it into a numerical format conducive to the analytical needs.

6.3 Encoding Categorical Variables

To render the categorical variables machine-learning-friendly, a judicious application of one-hot encoding was administered. This transformation was applied to variables such as 'region,' 'manufacturer,' 'model,' 'cylinders,' 'fuel,' 'title_status,' 'transmission,' 'drive,' 'type,' and 'paint_color,' imparting numerical representation while preserving the inherent insights contained in these categorical descriptors.

6.4 In-depth Exploratory Data Analysis

A comprehensive Exploratory Data Analysis unfolded, yielding profound insights into the relationships between diverse features and the target variable, 'price.' The canvas of visualizations included dynamic portrayals through boxplots, histograms, and pair plots. These visual tools were instrumental in unraveling the intricate distributions and correlations latent within the dataset.

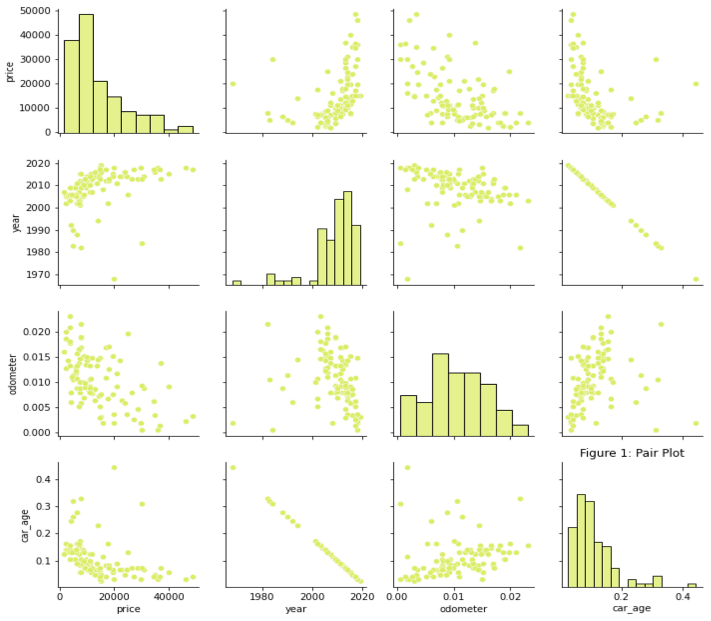


fig 1.5 pair plots

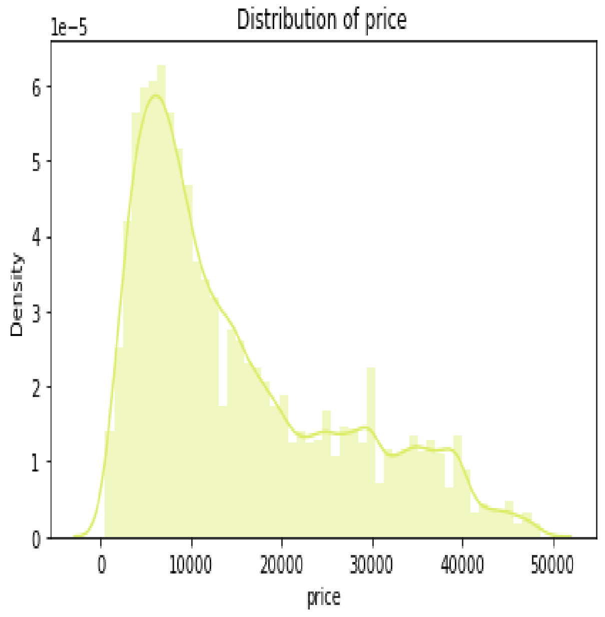


fig 1.6 Distribution of price

6.5 Feature Importance Evaluation

Diligent evaluation of the importance of different features in predicting used car prices was a pivotal aspect of the analysis. This involved the creation of illustrative bar plots, providing a visual narrative of the impactful variables. Special emphasis was placed on the influence of fuel type, with an additional exploration into the combined effect of fuel type and condition on car prices. These analyses equipped with a nuanced understanding of the variables steering the pricing dynamics within the dataset.

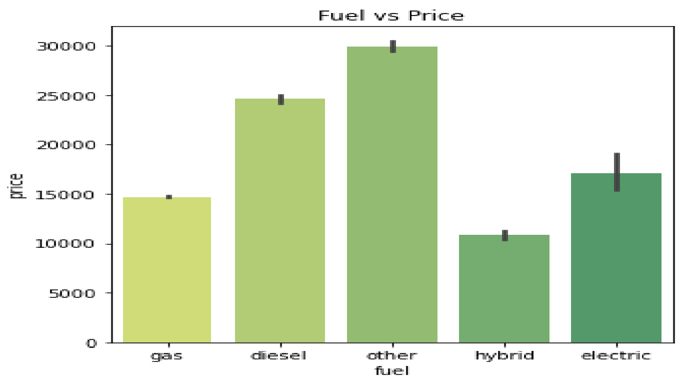


fig 1.7 Fuel vs Price

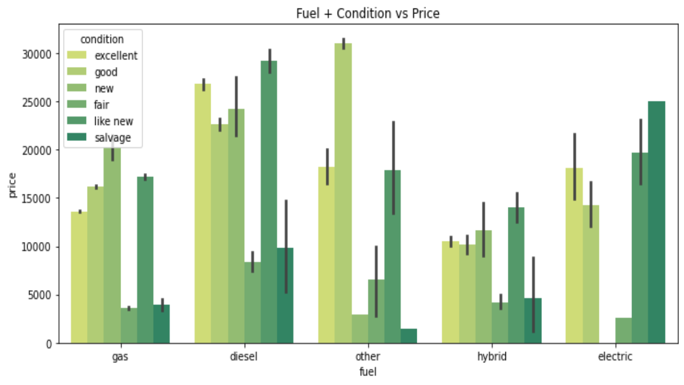


fig 1.8 Fuel + Condition vs Price

These meticulous steps within the realm of Exploratory Data Analysis not only fortified the dataset but also laid the groundwork for informed modeling and predictive endeavors in the subsequent phases of the project.

7. MODELS AND PERFORMANCE EVALUATION:

7.1 LINEAR REGRESSION:

Linear regression is a statistical method used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. The goal is to find the best-fitting line that minimizes the sum of the squared differences between the observed values and the values predicted by the linear model.
In this model, used Scikit-Learn for the analysis. Next, splitted the data into 70% training and 30% testing.
To summarize, the MAE values for both training and test sets are relatively high, indicating that, on average, there is a considerable absolute difference between the actual and predicted values. This suggests that the model might not be capturing the variability in the data well. The R-Score values (0.7218 for training and 0.71575 for test) suggest that the model explains a moderate percentage of the variance in the data. While these values are not extremely high, they indicate a reasonable fit to the data. The similarity between training and test MAE values suggests that the model is not suffering from significant overfitting or underfitting, and it is generalizing reasonably well to unseen data. To conclude, the model seems to provide a moderate fit to the data, but there might be room for improvement in terms of reducing errors.

7.2 POLYNOMIAL REGRESSION:

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an *nth-degree* polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y | x).
Here, trained a polynomial regression model with grid search and 5-fold cross-validation.
To summarize, the MAE values for both training and test sets are relatively high, indicating that, on average, there is a considerable absolute difference between the actual and predicted values. This suggests that the linear regression model might not be capturing the variability in the data well. The test R-Score of 0.71575 suggests that the linear regression model explains a moderate percentage of the variance in the test data. The best estimator polynomial degree is 1, which corresponds to a linear regression model. In summary, the linear regression model provides a reasonable fit to the data.

7.3 LASSO:

LASSO regression, also known as L1 regularization, is a popular technique used in statistical modeling and machine learning to estimate the relationships between variables and make predictions. LASSO stands for Least Absolute Shrinkage and Selection Operator.
Based on the obtained metrics, the training R-squared is 0.7219, and the test R-squared is 0.7157. The R-squared values, while not extremely high, indicate that the Lasso Regression model captures a substantial portion of the variance in the target variable on both training and test datasets. The MAE on the training data is 4377.54, representing the average absolute difference between the predicted and actual values on the training set. The MAE on the test data is 4403.84, indicating a similar level of average absolute error on the test set. The MAE values are relatively consistent between training and test sets, suggesting that the model generalizes well and its performance is comparable to new, unseen data.

Overall, the Lasso Regression model seems to be performing adequately, with a consistent level of performance on both the training and test datasets.

7.4 RIDGE REGRESSION:

Ridge regression is a model-tuning method that is used to analyze any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large. This results in predicted values being far away from the actual values.

The choice of the optimal value of the hyperparameter, which is alpha, is 0.001. This value controls the strength of the regularization term added to the linear regression cost function.

To summarize, the MAE values for both training and test sets are relatively high, indicating that, on average, there is a considerable absolute difference between the actual and predicted values. This suggests that the Ridge Regression model might not be capturing the variability in the data well. The R-Score values (0.7218 for training and 0.71569 for test) suggest that the model explains a moderate percentage of the variance in the data. The best hyperparameter tuning results suggest that a small amount of regularization (alpha = 0.001) is beneficial for improving model performance. Ridge Regression introduces a penalty term to prevent overfitting, and the chosen alpha value balances regularization and fitting the data.

7.5 KNN:

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier that uses proximity to make classifications or predictions about the grouping of an individual data point.

To summarize, this model has a relatively low Mean Absolute Error (MAE) on the training set, indicating that, on average, the predictions are close to the actual values during training. The MAE on the test set is slightly higher, but the model still performs reasonably well on unseen data. The R-Score values (0.85820 for training and 0.7756 for test) suggest that the model explains a significant portion of the variance in the data. The best parameter, k=5, and the associated best score of 0.7701 indicate that the model performs well with this specific choice of hyperparameter. Overall, the model seems to generalize well to new data, and the selected hyperparameter (k=5) appears to be suitable.

7.6 RANDOM FOREST:

Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time.

The model shows a very low MAE on the training set, indicating that, on average, the predictions are very close to the actual values during training. The MAE on the test set is higher than the training MAE but still relatively low, indicating good generalization to unseen data. The R-Score values (0.98844 for training and 0.9181 for test) suggest that the model explains a very high percentage of the variance in the data. The high R-Score on the test set indicates that the model is likely not overfitting and is performing well on new, unseen data. However, the test R-squared is slightly lower than the training R-squared, which is expected but the two values are reasonably close. It's a good sign when the R-squared values are high for both training and test datasets, as it indicates a good balance between model complexity and generalization to new data. Overall, based on the obtained metrics, it seems like the Random Forest model is performing exceptionally well and appears to be well-fitted to the data, showing high explanatory power both on the data it was trained on and on new, unseen data.

7.7 DECISION TREE:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. It has a hierarchical tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

From the output obtained, the MAE values for both training and test sets are relatively high, indicating that, on average, there is a considerable absolute difference between the actual and predicted values. This suggests that the Decision Tree model might not be capturing the variability in the data well. The R-Score values (0.7874 for training and 0.7811 for test) suggest that the Decision Tree model explains a substantial percentage of the variance in the data. This model has good predictive power, as indicated by the high R-Scores. The chosen hyperparameter values (max depth of 7 and min samples split of 40) indicate a relatively shallow tree with a conservative approach to splitting nodes. This can help prevent overfitting and improve the model's generalization to new, unseen data.

7.8 ADABOOST REGRESSION:

AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique used as an Ensemble Method in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances.

This model’s MAE values for both training and test sets are relatively low, indicating that, on average, there is a smaller absolute difference between the actual and predicted values. This suggests that the AdaBoost Regression model is performing well in terms of minimizing prediction errors. The R-Score values (0.8734 for training and 0.8539 for test) suggest that the model explains a substantial percentage of the variance in the data. The model has good predictive power, as indicated by the high R-Scores.

8. MODELS COMPARISON:

8.1 Error Analysis of algorithm :

SR NO	**MODEL NAME**	**MAE TRAIN**	**MAE TEST**
1	Linear Regression	4377.681	4403.96

2	Polynomial Regression	4377.681	4403.96
3	LASSO Regression	4377.54	4403.841
4	Ridge Regression	4377.569	4403.873
5	KNN	2683.882	3400.458
6	Random Forest Regression	690.731	1829.567
7	Decision Tree	3700.22	3718.884
8	AdaBoost	2986.25	3121.513

From the above table, can comprehend that MAE – Mean Absolute Error is higher in the Linear, Polynomial, Lasso, Ridge regression. Algorithms like KNN, Decision tree and AdaBoost have average MAE scores. Random Forest has the lowest MAE test score. Therefore, the Random forest algorithm is best suitable for this kind of data.

8.2 Accuracy comparison of algorithms:

SR NO	**MODEL NAME**	**TRAINING**	**TEST ACCURACY**
1	Linear Regression	72.1	71.5
2	Polynomial Regression	72.1	71.5
3	LASSO Regression	72.1	71.5
4	Ridge Regression	72.1	71.5

5	KNN	85.8	77.5
6	Random Forest Regression	98.8	91.8
7	Decision Tree	78.7	78.1
8	AdaBoost	87.3	85.3

Random Forest performs the best on the chosen dataset. After that, AdaBoost, Decision tree and KNN give moderate results. Linear, Polynomial, LASSO and Ridge regression produce poor results.

9. CONCLUSION:

In today's competitive business landscape, numerous companies are vying to establish themselves as industry leaders by offering top-notch products and expanding their customer base. The automotive industry, in particular, is progressively enhancing its capabilities and incorporating new technologies to meet evolving market demands. Determining an appropriate price tag for a car based on its features poses a significant challenge for manufacturers. The alignment of car quality and features plays a pivotal role in capturing customer interest and driving sales.

To address this challenge, have developed several machine learning models to predict the probable price of a car based on its features. My approach includes multiple linear regression, lasso, ridge regression, decision tree, and random forest classifier models. Through a comprehensive comparison, have identified the random forest classifier as the most effective model, boasting an impressive accuracy score of 91.8%.

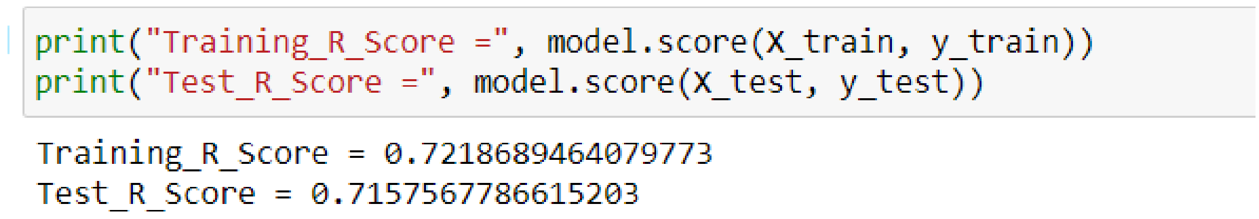
The dataset used for this analysis was sourced from Kaggle, comprising 16 variables where "msrp" serves as the dependent variable, and the remaining 15 act as predictors. With a dataset size of 79991 samples, the model has been trained effectively. Looking ahead, as technology continues to advance, experimenting with larger datasets incorporating additional parameters is advisable. Exploring other machine learning and deep learning techniques, including neural networks, holds promise for further enhancing the accuracy of car price predictions in the future.

10. APPENDIX:

Below are the python code for all the models:

LINEAR REGRESSION:

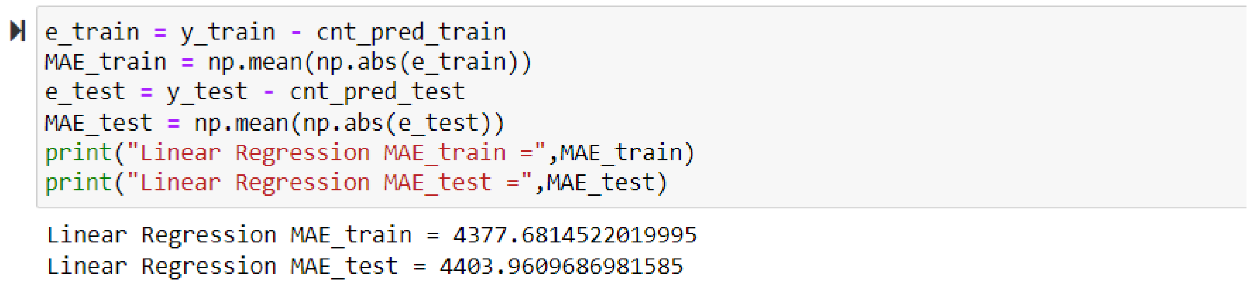
The R- squared Testing and Training data Accuracy obtained are :



```
print("Training_R_Score =", model.score(X_train, y_train))
print("Test_R_Score =", model.score(X_test, y_test))
```

```
Training_R_Score = 0.7218689464079773
Test_R_Score = 0.7157567786615203
```

The performance of the model is evaluated using Mean Absolute Error (MAE) on the training and testing data set whose values are :

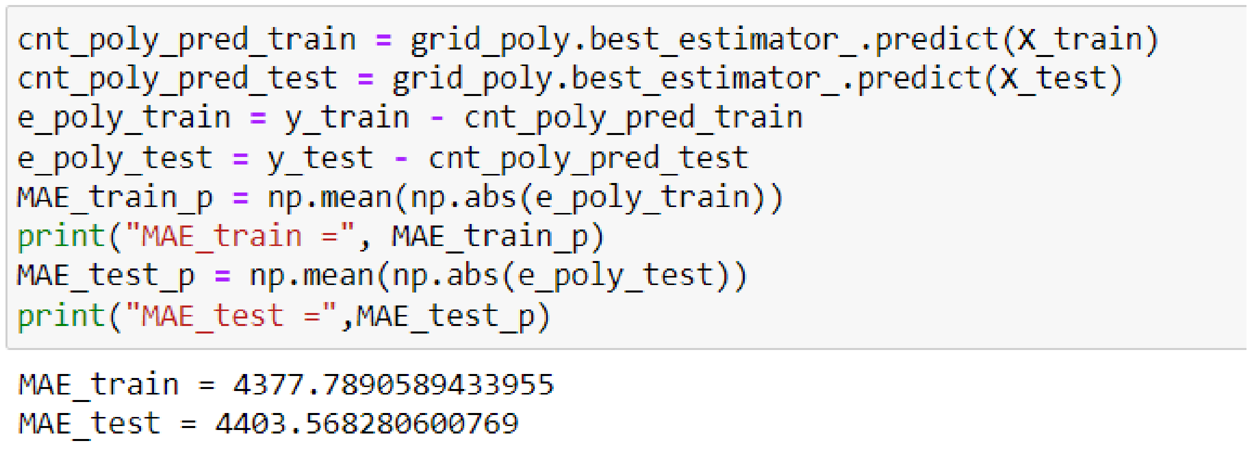


```
e_train = y_train - cnt_pred_train
MAE_train = np.mean(np.abs(e_train))
e_test = y_test - cnt_pred_test
MAE_test = np.mean(np.abs(e_test))
print("Linear Regression MAE_train =",MAE_train)
print("Linear Regression MAE_test =",MAE_test)
```

```
Linear Regression MAE_train = 4377.6814522019995
Linear Regression MAE_test = 4403.9609686981585
```

POLYNOMIAL REGRESSION:

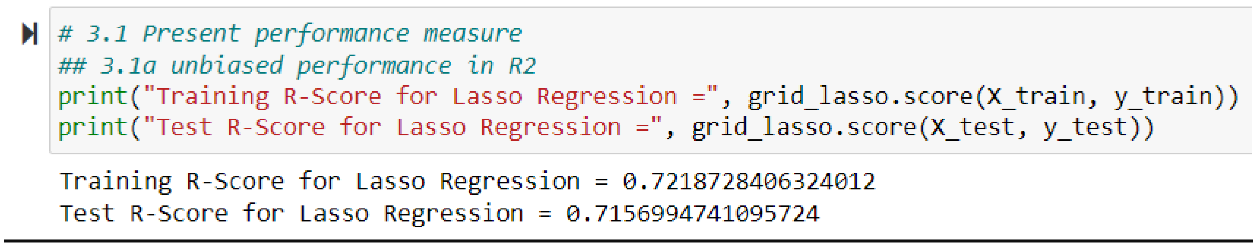
The performance of the model is evaluated using Mean Absolute Error (MAE) on the training and testing data set whose values are :



```
cnt_poly_pred_train = grid_poly.best_estimator_.predict(X_train)
cnt_poly_pred_test = grid_poly.best_estimator_.predict(X_test)
e_poly_train = y_train - cnt_poly_pred_train
e_poly_test = y_test - cnt_poly_pred_test
MAE_train_p = np.mean(np.abs(e_poly_train))
print("MAE_train =", MAE_train_p)
MAE_test_p = np.mean(np.abs(e_poly_test))
print("MAE_test =",MAE_test_p)
```

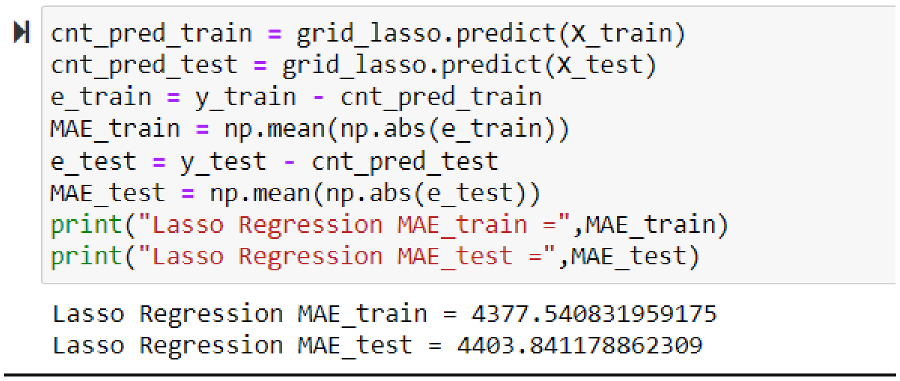
```
MAE_train = 4377.7890589433955
MAE_test = 4403.568280600769
```

LASSO:



```
# 3.1 Present performance measure
## 3.1a unbiased performance in R2
print("Training R-Score for Lasso Regression =", grid_lasso.score(X_train, y_train))
print("Test R-Score for Lasso Regression =", grid_lasso.score(X_test, y_test))
```

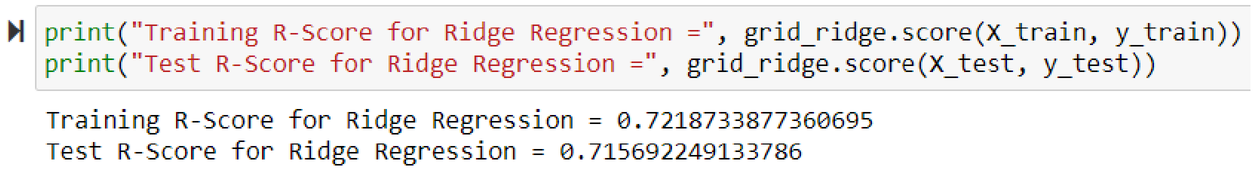
```
Training R-Score for Lasso Regression = 0.7218728406324012
Test R-Score for Lasso Regression = 0.7156994741095724
```



```
cnt_pred_train = grid_lasso.predict(X_train)
cnt_pred_test = grid_lasso.predict(X_test)
e_train = y_train - cnt_pred_train
MAE_train = np.mean(np.abs(e_train))
e_test = y_test - cnt_pred_test
MAE_test = np.mean(np.abs(e_test))
print("Lasso Regression MAE_train =",MAE_train)
print("Lasso Regression MAE_test =",MAE_test)
```

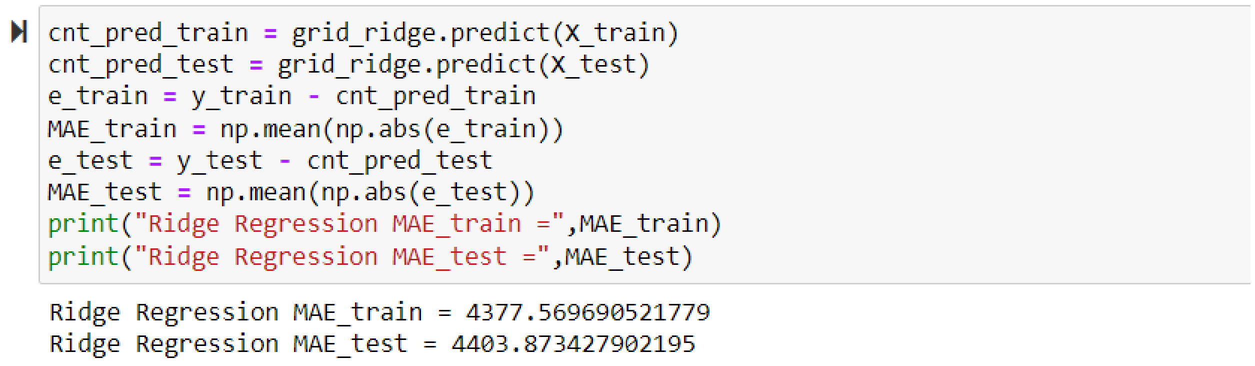
```
Lasso Regression MAE_train = 4377.540831959175
Lasso Regression MAE_test = 4403.841178862309
```

RIDGE REGRESSION:



```
print("Training R-Score for Ridge Regression =", grid_ridge.score(X_train, y_train))
print("Test R-Score for Ridge Regression =", grid_ridge.score(X_test, y_test))
```

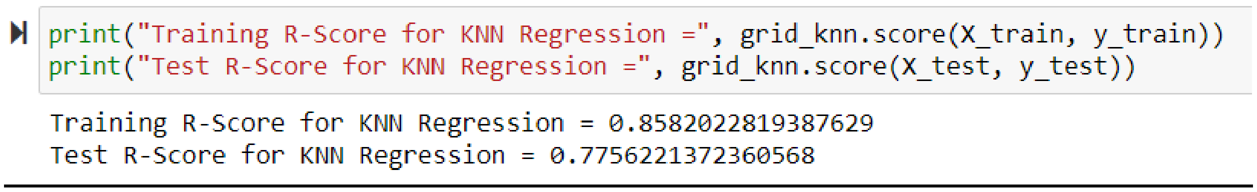
```
Training R-Score for Ridge Regression = 0.7218733877360695
Test R-Score for Ridge Regression = 0.715692249133786
```



```
cnt_pred_train = grid_ridge.predict(X_train)
cnt_pred_test = grid_ridge.predict(X_test)
e_train = y_train - cnt_pred_train
MAE_train = np.mean(np.abs(e_train))
e_test = y_test - cnt_pred_test
MAE_test = np.mean(np.abs(e_test))
print("Ridge Regression MAE_train =",MAE_train)
print("Ridge Regression MAE_test =",MAE_test)
```

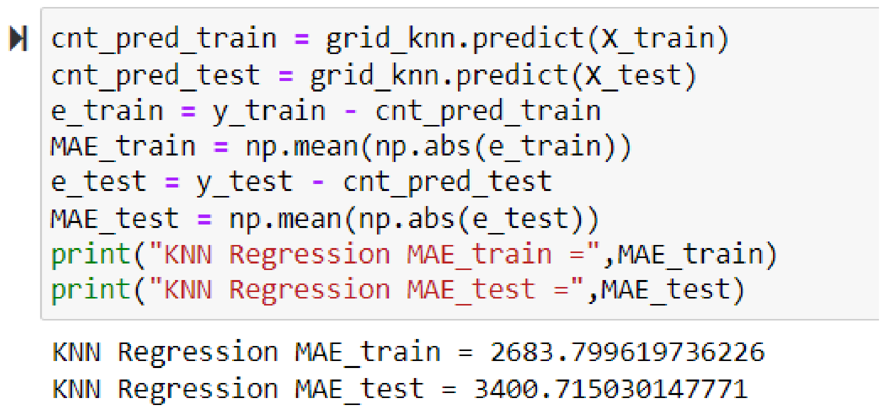
```
Ridge Regression MAE_train = 4377.569690521779
Ridge Regression MAE_test = 4403.873427902195
```

KNN:



```
print("Training R-Score for KNN Regression =", grid_knn.score(X_train, y_train))
print("Test R-Score for KNN Regression =", grid_knn.score(X_test, y_test))
```

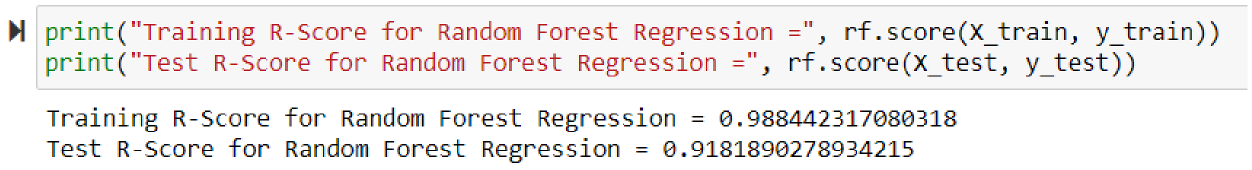
```
Training R-Score for KNN Regression = 0.8582022819387629
Test R-Score for KNN Regression = 0.7756221372360568
```



```
cnt_pred_train = grid_knn.predict(X_train)
cnt_pred_test = grid_knn.predict(X_test)
e_train = y_train - cnt_pred_train
MAE_train = np.mean(np.abs(e_train))
e_test = y_test - cnt_pred_test
MAE_test = np.mean(np.abs(e_test))
print("KNN Regression MAE_train =",MAE_train)
print("KNN Regression MAE_test =",MAE_test)
```

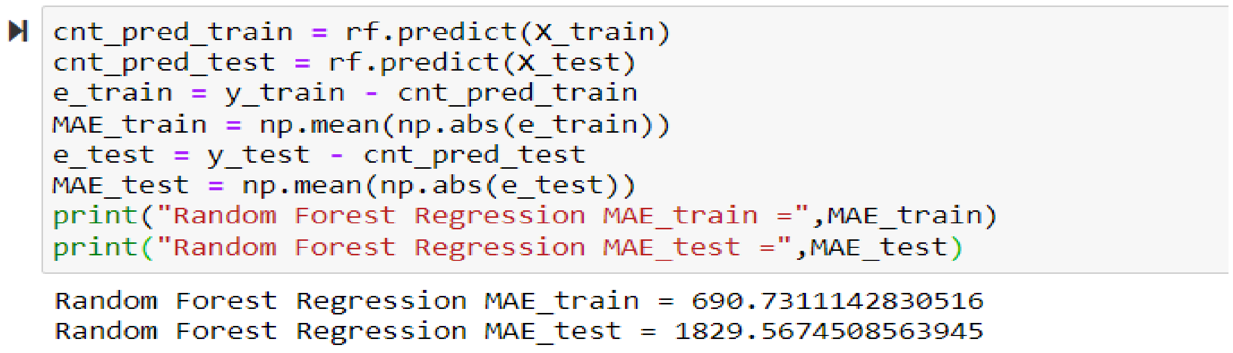
```
KNN Regression MAE_train = 2683.799619736226
KNN Regression MAE_test = 3400.715030147771
```

RANDOM FOREST:



```
print("Training R-Score for Random Forest Regression =", rf.score(X_train, y_train))
print("Test R-Score for Random Forest Regression =", rf.score(X_test, y_test))
```

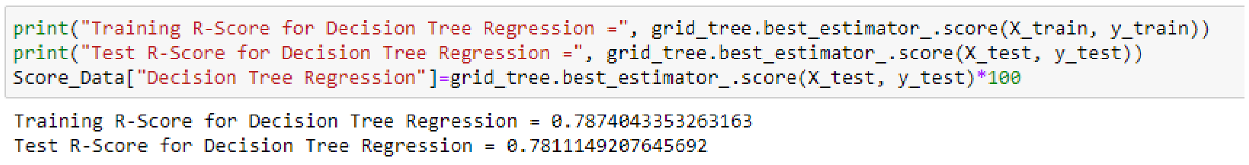
```
Training R-Score for Random Forest Regression = 0.988442317080318
Test R-Score for Random Forest Regression = 0.9181890278934215
```



```
cnt_pred_train = rf.predict(X_train)
cnt_pred_test = rf.predict(X_test)
e_train = y_train - cnt_pred_train
MAE_train = np.mean(np.abs(e_train))
e_test = y_test - cnt_pred_test
MAE_test = np.mean(np.abs(e_test))
print("Random Forest Regression MAE_train =",MAE_train)
print("Random Forest Regression MAE_test =",MAE_test)
```

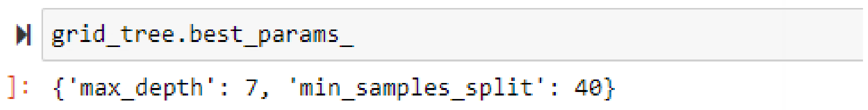
```
Random Forest Regression MAE_train = 690.7311142830516
Random Forest Regression MAE_test = 1829.5674508563945
```

DECISION TREE:



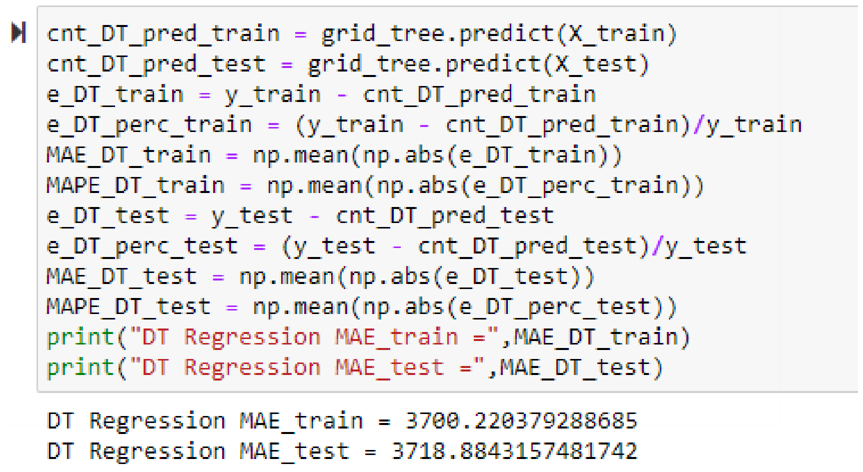
```
print("Training R-Score for Decision Tree Regression =", grid_tree.best_estimator_.score(X_train, y_train))
print("Test R-Score for Decision Tree Regression =", grid_tree.best_estimator_.score(X_test, y_test))
Score_Data["Decision Tree Regression"]=grid_tree.best_estimator_.score(X_test, y_test)*100
```

```
Training R-Score for Decision Tree Regression = 0.7874043353263163
Test R-Score for Decision Tree Regression = 0.7811149207645692
```



```
grid_tree.best_params_
```

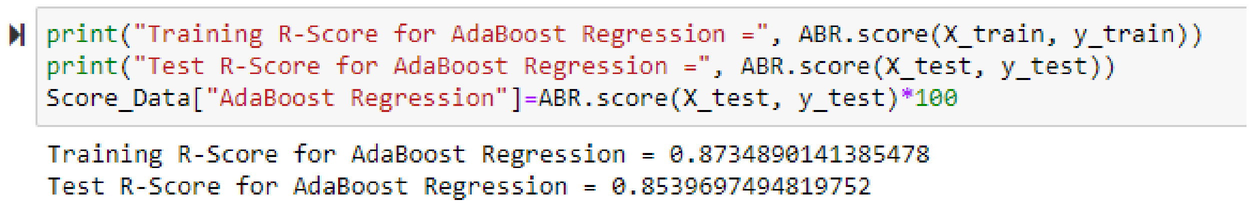
```
]: {'max_depth': 7, 'min_samples_split': 40}
```



```
cnt_DT_pred_train = grid_tree.predict(X_train)
cnt_DT_pred_test = grid_tree.predict(X_test)
e_DT_train = y_train - cnt_DT_pred_train
e_DT_perc_train = (y_train - cnt_DT_pred_train)/y_train
MAE_DT_train = np.mean(np.abs(e_DT_train))
MAPE_DT_train = np.mean(np.abs(e_DT_perc_train))
e_DT_test = y_test - cnt_DT_pred_test
e_DT_perc_test = (y_test - cnt_DT_pred_test)/y_test
MAE_DT_test = np.mean(np.abs(e_DT_test))
MAPE_DT_test = np.mean(np.abs(e_DT_perc_test))
print("DT Regression MAE_train =",MAE_DT_train)
print("DT Regression MAE_test =",MAE_DT_test)
```

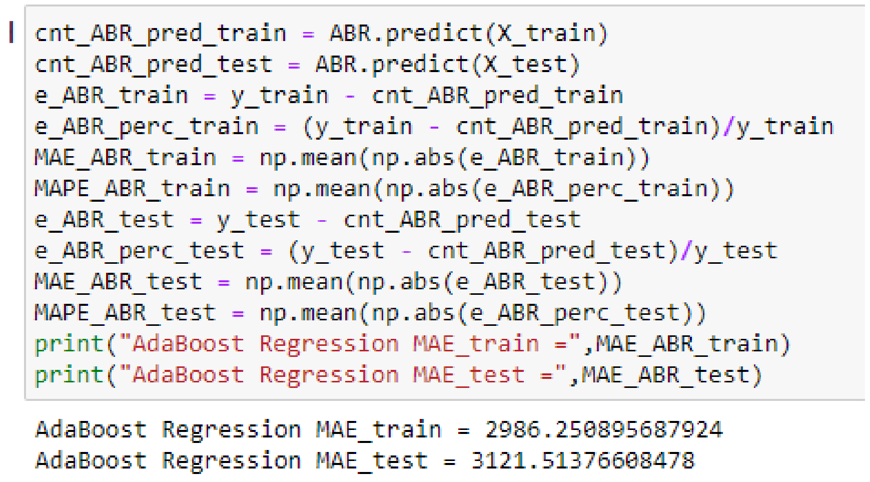
```
DT Regression MAE_train = 3700.220379288685
DT Regression MAE_test = 3718.8843157481742
```

ADABOOST REGRESSION:



```
print("Training R-Score for AdaBoost Regression =", ABR.score(X_train, y_train))
print("Test R-Score for AdaBoost Regression =", ABR.score(X_test, y_test))
Score_Data["AdaBoost Regression"]=ABR.score(X_test, y_test)*100
```

```
Training R-Score for AdaBoost Regression = 0.8734890141385478
Test R-Score for AdaBoost Regression = 0.8539697494819752
```



```
cnt_ABR_pred_train = ABR.predict(X_train)
cnt_ABR_pred_test = ABR.predict(X_test)
e_ABR_train = y_train - cnt_ABR_pred_train
e_ABR_perc_train = (y_train - cnt_ABR_pred_train)/y_train
MAE_ABR_train = np.mean(np.abs(e_ABR_train))
MAPE_ABR_train = np.mean(np.abs(e_ABR_perc_train))
e_ABR_test = y_test - cnt_ABR_pred_test
e_ABR_perc_test = (y_test - cnt_ABR_pred_test)/y_test
MAE_ABR_test = np.mean(np.abs(e_ABR_test))
MAPE_ABR_test = np.mean(np.abs(e_ABR_perc_test))
print("AdaBoost Regression MAE_train =",MAE_ABR_train)
print("AdaBoost Regression MAE_test =",MAE_ABR_test)
```

```
AdaBoost Regression MAE_train = 2986.250895687924
AdaBoost Regression MAE_test = 3121.51376608478
```