Vehicle Price Prediction (Autotrader)

Name: SACHIN IRAGOUDA PATIL

Subject: Machine Learning Concepts

Student ID: 22530172

Manchester Metropolitan University

Table of Content

Overview

In this assignment, we are working on the prediction of the car prices provided by the Auto Trade, British automotive business.
For this assignment, we are considering below features to predict the price of car:

- Mileage
- Year of Registration
- Vehicle Condition
- Fuel Type
- Body Type
- Standard Make

The target (independent variable) is “Price” of the car, which is provided in the dataset. In this report, we will analyse three regressor algorithms to decide the best suit for predicting the price of a vehicle.

Data Exploration

In this section, I will load the data and store it into a variable. And this variable can be used for further processing.

```
dataset_location = 'adverts.csv'
adverts = pd.read_csv(
    dataset_location
)
```

Now we have all the data stored in a variable called **“adverts”.**

I have used the method **“read_csv()”** from the **pandas** library, to read the data from the CSV file.

Now, I will look for the following things in the given dataset.

- Missing or Null values
- Outliers/Noise
- Features identification

Identification of Null values

Missing or null values present in the data could affect the accuracy of a machine learning model. To improve the accuracy of the model, first, we need to handle all the null values present in the data frame.

To handle null values, we can use two methods:

- Replace null values with the median value of the correspondent column (for numeric data only).
- Replace null values with the most commonly used data in that correspondent column (for string type of data).

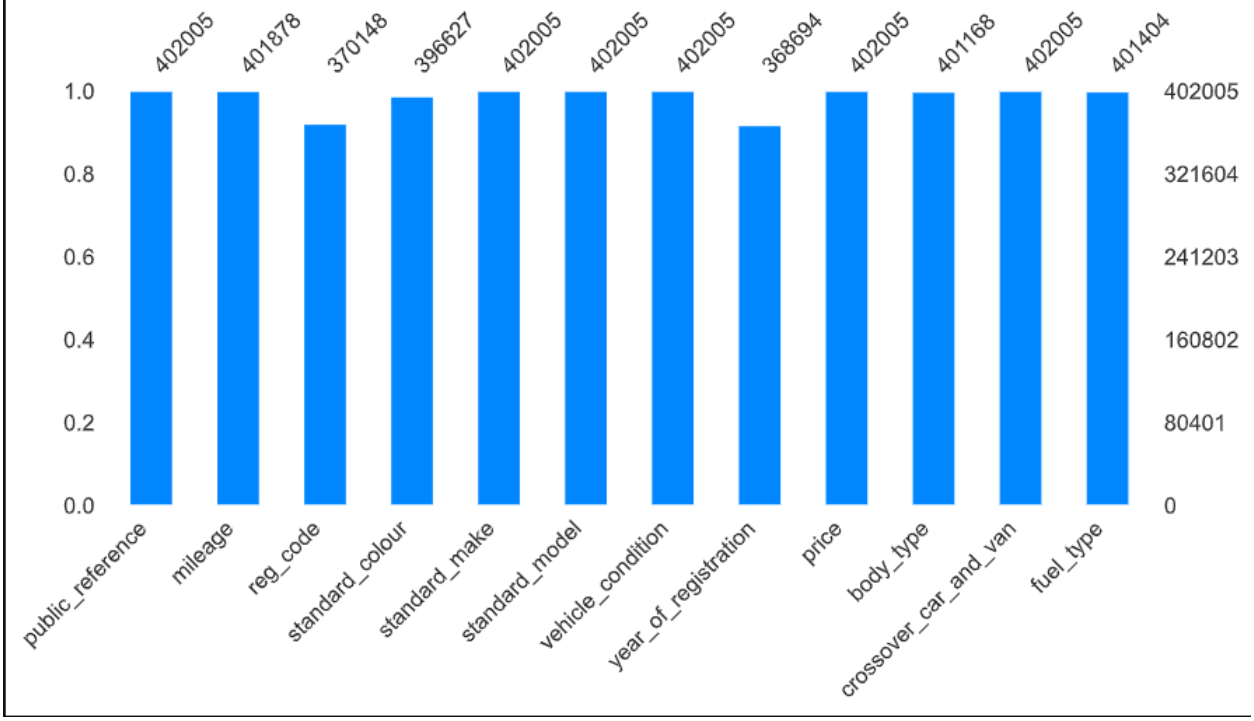
To get the count of null values in the data frame, I am using two methods, **isnull(), and sum()**. These two columns will give the total count as per the column. With the help of “**missingno”** library, we will plot the bar.

```
adverts.isnull().sum()
```

```
public_reference            0
mileage                   127
reg_code                31857
standard_colour          5378
standard_make               0
standard_model              0
vehicle_condition           0
year_of_registration    33311
price                       0
body_type                 837
crossover_car_and_van       0
fuel_type                 601
dtype: int64
```

```
msno.bar(adverts,color="dodgerblue", figsize=(10,4), fontsize=12)
```

```
<AxesSubplot: >
```



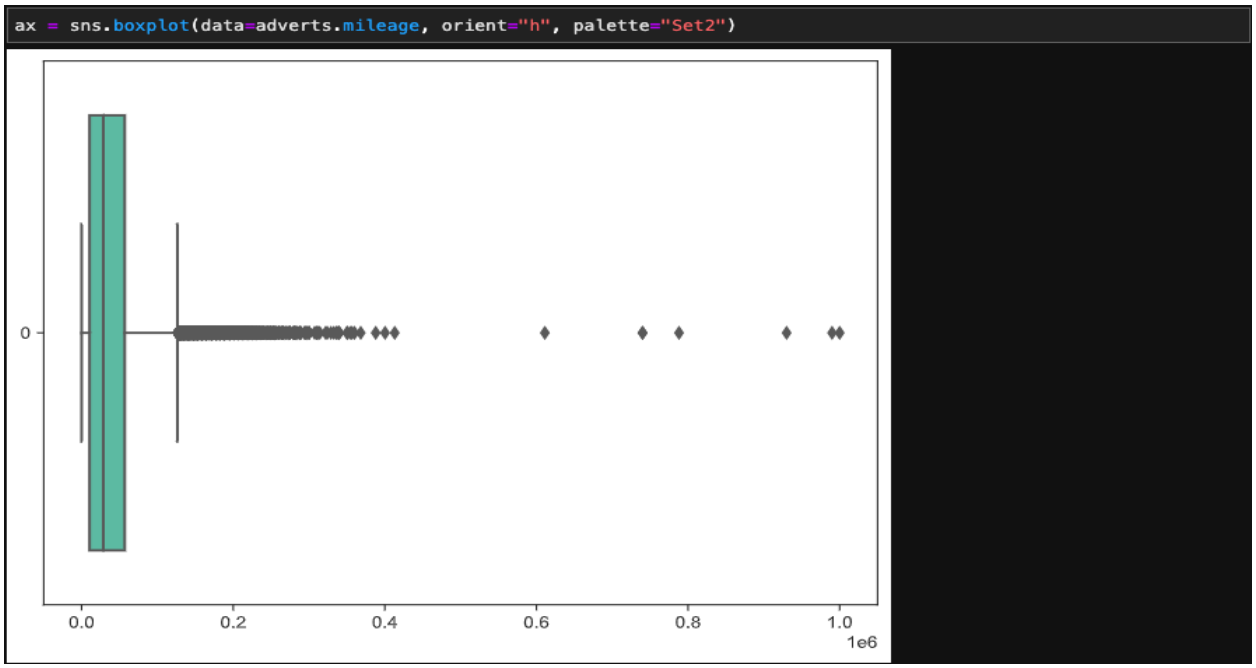
Identification of Outliers

Outliers are the data that are not linear in nature. Those could be a way above the average or a way below the average data. It could affect the overall prediction accuracy of the machine learning model. It is very important to handle these outliers before building the model for price prediction.

To handle outliers, we have two importance types of solutions:

- Z-Score
- IQR (Inter Quantile Range)

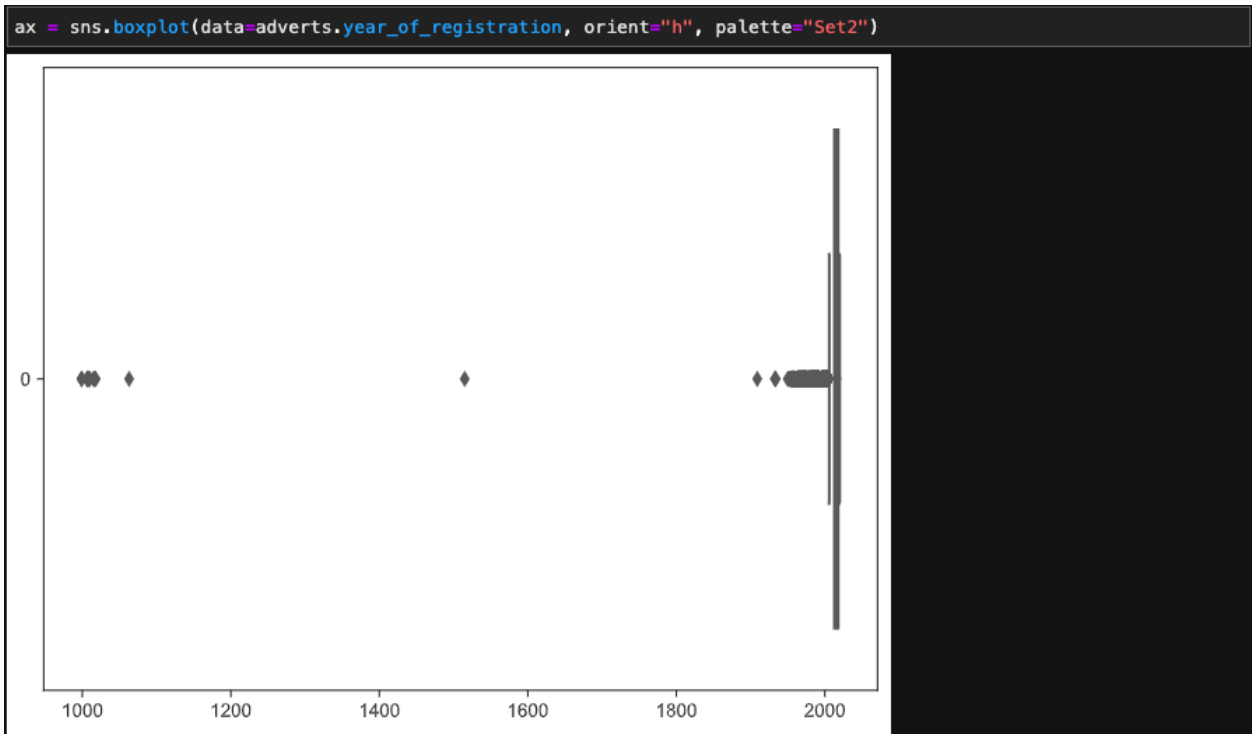
In this solution, I am using the IQR method. In this method, first we will calculate the first and third quarter to calculate the upper and lower range of the given data. This only works on the numerical data. We have **“mileage”, ”year_of_registration”, ”price”** columns, that contains the outliers.



Above boxplot shows the number of outliers present in the **mileage** column.

Likewise, We will also see the boxplot for **year_of_registration** column.

```
ax = sns.boxplot(data=adverts.year_of_registration, orient="h", palette="Set2")
```



Feature Analysis

In this section, we will findout the important features that are required for prediction of the target. We have mainly two types of features available in the dataset.

- Categorial features
- Quantitative features

Categorical features:

- standard_make
- standard_colour
- vehicle_condition
- body_type
- fuel_type

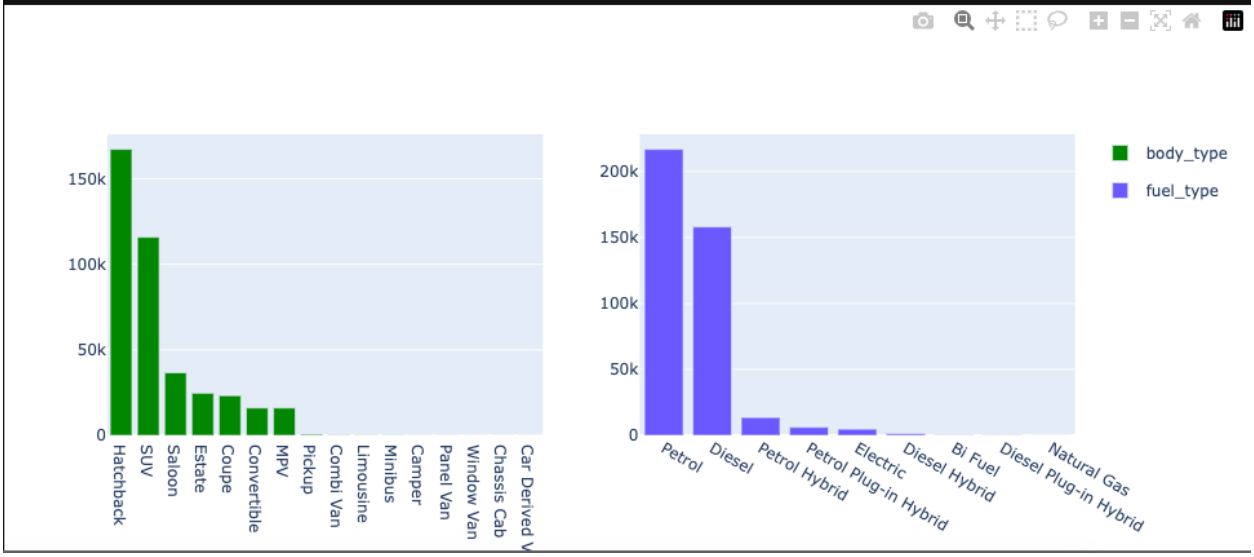
Quantitative features:

- mileage
- year_of_registration

```
features = adverts.columns
features
```

```
Index(['public_reference', 'mileage', 'reg_code', 'standard_colour',
       'standard_make', 'standard_model', 'vehicle_condition',
       'year_of_registration', 'price', 'body_type', 'crossover_car_and_van',
       'fuel_type'],
      dtype='object')
```

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=1, cols=2)
body_types = adverts['body_type'].value_counts()
fuel_types = adverts['fuel_type'].value_counts()
fig1=px.bar(body_types)
fig1.update_traces(marker_color='green')
fig2=px.bar(fuel_types)
fig.add_trace(fig1['data'][0], row=1, col=1)
fig.add_trace(fig2['data'][0], row=1, col=2)
```



Above charts shows us the distribution of categorical features. I considered two features, **body_type** and **fuel_type**. We can see the distribution in the above plots.

Data Processing

This step in the machine learning model building process is very crucial, because the way we process and clear the data, we will get the accuracy of the machine learning model. In this process, we clean the data, handle the outliers present in the data, we perform the required encodings for the data that is not in the numeric format.

We will preform the below operation in order to make sure that data is clean and outliers free before proceeding towards the model creation:

- Handling the null values
- Handling the Outliers
- Feature engineering

Handling Null values

As we have already discussed in the Data Exploration section, we have found the null values present in the given dataset.

```
print('Null Values in Mileage column: ',adverts.mileage.isnull().sum())
print('Null Values in Year of Registration column: ',adverts.year_of_registration.isnull().sum())
print('Null Values in Price column: ',adverts.price.isnull().sum())
print('Null Values in Body Type column: ',adverts.body_type.isnull().sum())
print('Null Values in Fuel Type column: ',adverts.fuel_type.isnull().sum())
```

```
Null Values in Mileage column:  127
Null Values in Year of Registration column:  33311
Null Values in Price column:  0
Null Values in Body Type column:  837
Null Values in Fuel Type column:  601
```

As we can see, null values present in the **mileage, year_of_registration, body_type and fuel_type** columns.

To get rid of this null values, I am using the method of replacing the null values with the median value of that correspondent column (for the numeric data only).

```
adverts['mileage'] = adverts['mileage'].fillna(adverts['mileage'].median())
adverts['year_of_registration']=adverts['year_of_registration'].fillna(adverts['year_of_registration'].median())
adverts['year_of_registration']=adverts['year_of_registration'].astype(int)
```

Simply replaced the null values with the median value of the correspondent column.

But, We cannot use this method for the data of string type. So we will find the most commonly used value of the corresponding column.

```
adverts.body_type.value_counts()
```

```
Hatchback          167315
SUV                115872
Saloon              36641
Estate              24692
Coupe               23258
Convertible         16038
MPV                 16026
Pickup                620
Combi Van             214
Limousine             159
Minibus               149
Camper                 77
Panel Van              61
Window Van             41
Chassis Cab             3
Car Derived Van         2
Name: body_type, dtype: int64
```

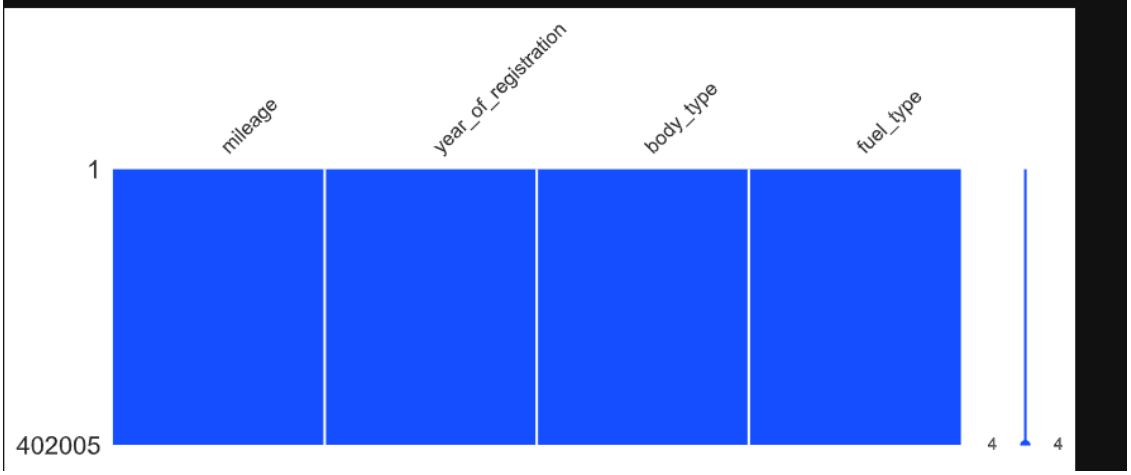
```
adverts.fuel_type.value_counts()
```

```
Petrol                  216929
Diesel                  158120
Petrol Hybrid            13602
Petrol Plug-in Hybrid     6160
Electric                  4783
Diesel Hybrid             1403
Bi Fuel                    221
Diesel Plug-in Hybrid      185
Natural Gas                  1
Name: fuel_type, dtype: int64
```

We have found the commonly used **body_type and fuel_type** values using the **"value_counts()"** method. It gives the total count of each type of categorical data present in the dataset.

```
msno.matrix(adverts[['mileage','year_of_registration','body_type','fuel_type']],
            color = (0, 0.4, 1), figsize=(10,3), fontsize=12)
```

```
<AxesSubplot: >
```



As we cas see, above chart clearly shows that there are no null values present in all above features.

Handling Outliers

As discussed in the Data Exploration section, data contains the outliers in **mileage, price and year_of_registration** columns. It is a crucial step to remove those outliers from the data in order to improve the accuracy of a machine learning model.

Here I am using the IQR (Inter-quantile range) method to eliminate the outliers.

Steps:

1. Find the first and third quarter for each column

```
q1_price,q3_price = np.percentile(prices,[25,75])
print('price range',q1_price,q3_price)
```

```
price range 7495.0 20000.0
```

```
q1_mileage,q3_mileage = np.percentile(mileage,[25,75])
print('mileage range',q1_mileage,q3_mileage)
```

```
mileage range 10487.0 56852.0
```

```
q1_yor,q3_yor = np.percentile(X.year_of_registration.values,[25,75])
print('year of registration range',q1_yor,q3_yor)
```

```
year of registration range 2014.0 2018.0
```

To find the q1 and q3, we will use the **percentile()** method of the **numpy** library. It takes two arguments, first is the column name (in the array format) and second is the range (here I am using 25 and 75).

```
# find IQR for Mileage
mileage_iqr = q3_mileage-q1_mileage
mileage_iqr
```

```
46365.0
```

```
# find IQR for Price
price_iqr = q3_price-q1_price
price_iqr
```

```
12505.0
```

```
# find IQR for Year of Registration
yor_iqr = q3_yor-q1_yor
yor_iqr
```

```
4.0
```

2. Find the Upper and lower limits

   Once we get the IQR value, now we will calculate the upper limit and the lower limit.

```
# finding lower and upper bound for Mileage
low_bound_mileage = q1_mileage-(1.5 * mileage_iqr)
upp_bound_mileage = q3_mileage+(1.5 * mileage_iqr)
print(low_bound_mileage, upp_bound_mileage)
```

```
-59060.5 126399.5
```

```
# finding lower and upper bound for Price
price_low_bound = q1_price-(1.5 * price_iqr)
price_upp_bound = q3_price+(1.5 * price_iqr)
print(low_bound, upp_bound)
```

```
-11262.5 38757.5
```

```
# finding lower and upper bound for YOR
yor_low_bound = q1_yor-(1.5 * yor_iqr)
yor_upp_bound = q3_yor+(1.5 * yor_iqr)
print(yor_low_bound, yor_upp_bound)
```

```
2008.0 2024.0
```

   Now we got the lower and upper limit for all the numerical features.

3. Trim the data

   Next step is to trim the data as per lower and upper limits

```
adverts = adverts.query("year_of_registration>=@yor_upp_bound")
```

```
adverts = adverts.query("price <= @price_upp_bound")
```

```
print('Min Price',adverts.price.min())
print('Max Price',adverts.price.max())
```

```
Min Price 250
Max Price 38755
```

```
adverts = adverts.query("mileage <= @upp_bound_mileage")
```

```
print('Min mileage: ',adverts.mileage.min())
print('Max mileage: ',adverts.mileage.max())
```

```
Min mileage:  0.0
Max mileage:  126365.0
```

   Here I am using the **query()** method to trim the data. I am passing the query to trim the data with respect to lower and upper limit.

Feature Engineering

This is one of the most important process in the machine learning model creation. This process includes the encoding of the features which are of string type. Since string type of data is not acceptable as input for a machine learning algorithm, It is important to convert the string values into appropriate numeric value.

Here I am using two types of encoding:

- Ordinal encoding
- Target encoding

Firstly, using ordinal encoding for converting the **vehicle_condition, body_type and fuel_type** columns (which contains the string data) into numeric data.

```
oe = OrdinalEncoder()
adverts.vehicle_condition=oe.fit_transform(adverts[["vehicle_condition"]]).astype(int)
adverts.crossover_car_and_van=oe.fit_transform(adverts[["crossover_car_and_van"]]).astype(int)

adverts.body_type=oe.fit_transform(adverts[["body_type"]]).astype(int)
adverts.fuel_type=oe.fit_transform(adverts[["fuel_type"]]).astype(int)
```

Ordinal encoding performs the rank based encoding. It gives the rank to a string type of data.

E.g.: USED: 1, NEW: 0

Now for column **standard_make**, since the brand values of different vehicle is different, we cannot use the normal ordinal encoding, because it gives rank randomly to the data which is not a proper way when it comes to **standard_make**.

For this column I am using the target encoding, which calculates the mean price of each brand, and replace that mean price with the brand name. In this way, we will have the proper data for the column **standard_make**.

```
tenc=ce.TargetEncoder()
standard_make_1=tenc.fit_transform(adverts['standard_make'],adverts['price'])
standard_model_1=tenc.fit_transform(adverts['standard_model'],adverts['price'])

adverts = standard_model_1.join(adverts.drop('standard_model',axis = 1))
adverts = standard_make_1.join(adverts.drop('standard_make',axis = 1))
adverts.head(3)
```

Now after encoding, we have prepared with the data to create a machine learning model. Now next step is to create a machine learning model.

Model Creation and Performing Prediction

Now, we are done with the data cleaning, encoding and pre-processing. And also performed the transformation and fitting the data (Features data).

Now next step is to predict the prices with the help of test data, which we have divided from the main dataset using train_test_split class.

Here, we are using 3 algorithms to predict the prices of the car

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor

Firstly, need to split the data into train and test, so that we can have some data to test our model accuracy.

Here I am dividing the data into 75% data for training the model and 25% data for testing purpose.

```
#Split data into test and train
X_train, X_test, y_train, y_test=train_test_split(features,target, test_size=0.25, random_state=48)
```

```
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

Here I am performing the scaling to improve the accuracy of machine learning model. Object of **StandardScaler()** function will help to fit and transform the training data.

Linear Regression

This is one of the most common regression algorithm to make a prediction on the basis of provided set of features.

```
# Linear Regression
lr= LinearRegression()
lr.fit(X_train, y_train)
ypred = lr.predict(X_test)
lr_r2_Score = (r2_score(y_test, ypred)*100).round(2)
lr_mean_abs_err = mean_absolute_error(y_test, ypred)
lr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, ypred))
print('r2 score: ',lr_r2_Score ,'%')
print('mean absolute error: ',lr_mean_abs_err)
print('mean squared error: ',lr_mean_sqr_err)
```

```
r2 score:  79.05 %
mean absolute error:  2791.7150174515973
mean squared error:  3737.347419986021
```

In linear regression, we use train data to fit the model. Usign the **LinearRegression** class, to use the **predict()** method which predicts the price.

Here I am also calculating the mean squared error and mean absolute error, which is not up to the mark. So we can say that this is not the best regressor for the prediction.

Decision Tree Regressor

This regressor builds the regressor in the for of trees. It basically breaks the given data into smaller trees. The final result will be a tree with **decision nodes and leaf nodes**.

```
# Decision Tree Regressor
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)
yhat = dt.predict(X_test)
dtr_r2_Score = (r2_score(y_test, yhat)*100).round(2)
dtr_mean_abs_err = mean_absolute_error(y_test, yhat)
dtr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, yhat))
print('r2 score: ',dtr_r2_Score ,'%')
print('mean absolute error: ',dtr_mean_abs_err)
print('mean squared error: ',dtr_mean_sqr_err)
```

```
r2 score:  88.8 %
mean absolute error:  1800.137376754181
mean squared error:  2732.476975425621 3
```

The r2 score for decision tree regressor is better as compared with the r2 score of a linear regression model. And also the mean absolute error and mean squared error is low as compared with the linear regression model.

Random Forest Regression

This is another very popular regression method of supervised learning that uses the ensemble learning for regression.

```
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
regression = RandomForestRegressor(n_estimators=100,random_state=0)
regression.fit(X_train,y_train)
y_pred = regression.predict(X_test)
rfr_r2_score = r2_score(y_test, y_pred)*100
rfr_r2_score=rfr_r2_score.round(2)
rfr_r2_score
rfr_mean_abs_err = mean_absolute_error(y_test, y_pred)
rfr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, y_pred))
print('r2 score: ',rfr_r2_score ,'%')
print('mean absolute error: ',rfr_mean_abs_err)
print('mean squared error: ',rfr_mean_sqr_err)
```

```
r2 score:  92.38 %
mean absolute error:  1512.126864825055
mean squared error:  2254.2230183967313
```

As we can see in the above image, I am using **RandomForestRegressor** class to predict the price of a vehicle.

RandomForestRegressor takes two arguments. First is the number of estimators, and second is the random state.

The r2 score of this regressor is much better that rest two regressors. And also the mean absolute error and mean squared error is low as compared with other two regressors.

So we can say that, with given data, Random forest regression is the best algorithm to make a model to predict the price of a vehicle.

Model Analysis

In this section, we will analyse the result of all the models. Below table shows the overall accuracy of each model along with their mean absolute error and mean squared error.

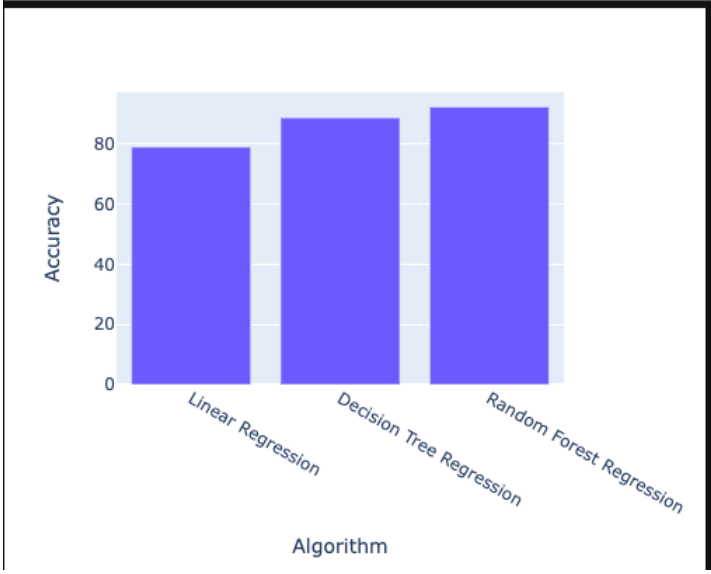
```
result = pd.DataFrame({
'Algorithm': ['Linear Regression','Decision Tree Regression','Random Forest Regression'],
'Accuracy':[lr_r2_Score,dtr_r2_Score,rfr_r2_score],
'Mean Absolute Error':[lr_mean_abs_err,dtr_mean_abs_err,dtr_mean_sqr_err],
'Mean Squared Errir':[lr_mean_sqr_err,rfr_mean_sqr_err,rfr_mean_sqr_err]
})
```

```
result
```

	Algorithm	Accuracy	Mean Absolute Error	Mean Squared Errir
0	Linear Regression	79.03	2793.185315	3739.210803
1	Decision Tree Regression	88.78	1799.551652	2254.808406
2	Random Forest Regression	92.37	2734.478408	2254.808406

Visualization of above table:

```
fig = px.bar(result, x='Algorithm', y='Accuracy',height=400, width=500)
fig.show()
```



Above chart shows us the final scores of all three model along with the mean absolute error and mean squared errors.

Feature Importance

Depending on how crucial they are for predicting the outcome, it ranks the input features. The features' score will increase according to how accurately they predict the result. It can be used to address issues including classification and regression.

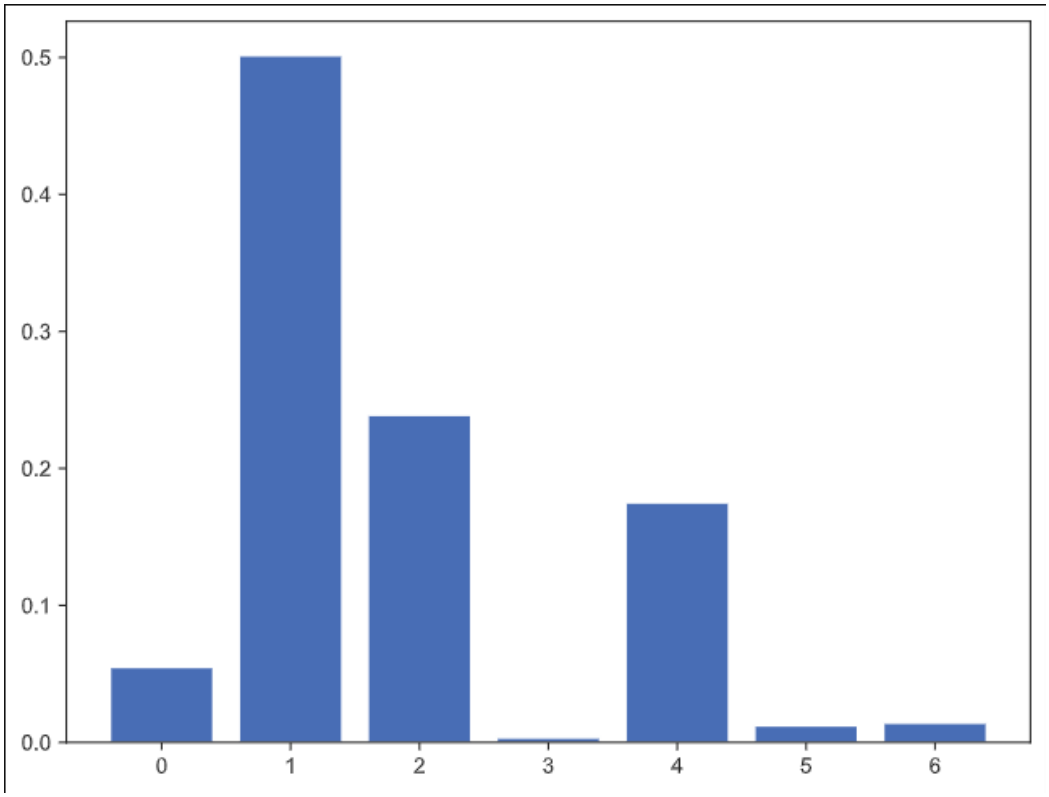
Firstly, we will see the feature importance for **decision tree regressor**

```
importance = dt.feature_importances_
importance
```

```
array([0.05488311, 0.50136899, 0.23895449, 0.00339036, 0.17501839,
       0.01221908, 0.0141656 ])
```

```
for i,v in enumerate(importance):
    print('Feature: {}, Score: {}'.format(i,v))
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Feature: 0, Score: 0.05488310708426268
Feature: 1, Score: 0.5013689855440775
Feature: 2, Score: 0.23895448573397746
Feature: 3, Score: 0.0033903590646417177
Feature: 4, Score: 0.17501838541093936
Feature: 5, Score: 0.01221908013183956
Feature: 6, Score: 0.014165597030261797
```



As we can see in the above bar chart, feature no.1, 2 and 4 quite important as compared to others.

Random Forest Regressor

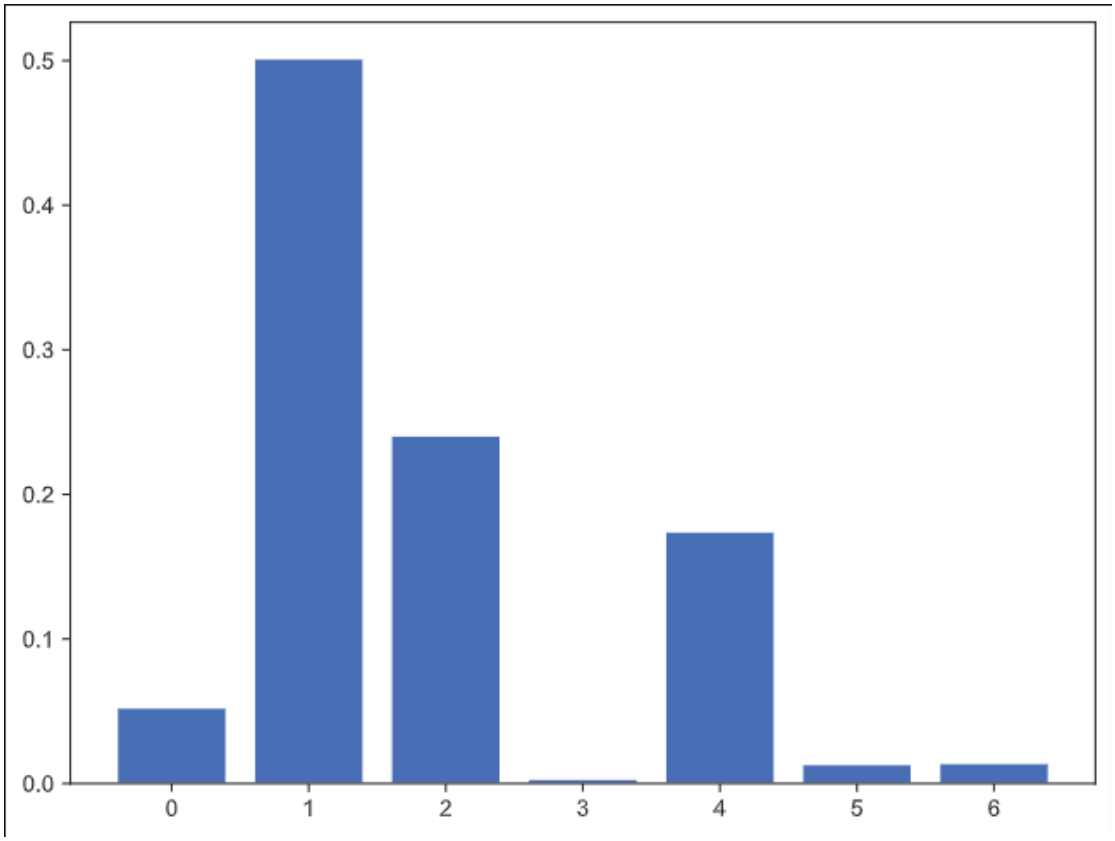
Below is the feature importance array for random forest regressor.

```
regression.feature_importances_
```

```
array([0.05254355, 0.50152191, 0.24074547, 0.00327856, 0.17434767,
       0.01340609, 0.01415674])
```

```
for i,v in enumerate(regression.feature_importances_):
    print('Feature: {}, Score: {}'.format(i,v))
plt.bar([x for x in range(len(regression.feature_importances_))], regression.feature_importances_)
plt.show()
```

```
Feature: 0, Score: 0.052543553620518046
Feature: 1, Score: 0.5015219089766918
Feature: 2, Score: 0.2407454686018319
Feature: 3, Score: 0.0032785613062360753
Feature: 4, Score: 0.1743476736767348
Feature: 5, Score: 0.013406088892563423
Feature: 6, Score: 0.01415674492542411
```



Conclusion

In this project, I have created three machine learning models to predict the price of a vehicle. Based on the final result including the r2 scores and mean absolute and mean squared error we can conclude that Decision tree and Random forest regressors are suitable to create the model.