Car Price Prediction

Submitted by:

Chethan B. K.

ACKNOWLEDGMENT

First of all I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity.

The Data was collected from the below websites

- https://www.carwale.com/
- https://www.cardekho.com/

Most of the concepts used to predict the Micro-Credit loan defaulters are learned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/
- Stack-overflow
- https://imbalanced-learn.org/stable/

INTRODUCTION

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper

The market value is based on a number of factors, including demand, supply, options, and incentives. The market value of a vehicle usually falls somewhere between the sticker price and the invoice price. Because the market value is an average, some people will pay more than that amount, while others will pay less.

A car's value is determined by many factors: the popularity of the make and model of your car, vehicle specifications, trim levels, physical appearance, mileage, consistent maintenance and working condition.

Using this as a base, I have collected the data from few websites. The data was collected for the car body types like sedan, SUV/MUV, hatchback, minivan, coupe/convertibles

Once the data is collected, the data will be cleaned and pre-processed with all the necessary tools and the same will be used to build machine learning models in order to predict the price of the same.

Analytical Problem Framing

The dataset has around 6500 rows and 13 columns. Using this dataset we will be training the Machine Learning models on 70% of the data and the models will be tested on 30% data.

Since we have removed the null values from the dataset during the data collection stage, we can expect outliers and un-realistic values for certain variables.

Below are the definition for each variable available on the dataset

Price	Listed Price of the car
year	Year of manufacturing of the car
fuel	Type of fuel used in the car
trans	Type of transmission in a car (Automatic/Manual)
mile	Number of Kilometres driven
color	Color of the car
owner	Number of previous owner for the car
engine	Engine size of the car in CC (Cubic Centimetre)
seating	Seating capacity of the car
location	Car's location
Body	Type of car (ex. SUV, sedane, hatchback)
make	Brand name of the car
model	Car's model/variant

Importing the necessary libraries and looking at the glimpse of the data

```
import pandas as pd
import numpy as np
from category_encoders import BinaryEncoder
from sklearn.preprocessing import power_transform, OrdinalEncoder
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
data = pd.read_excel(r'C:\Users\LENOVO\Documents\Post Graduation\internship\car-data collection\data files\cars.xlsx')
data.head()
```

	price	year	fuel	trans	mile	color	owner	engine	seating	location	Body	make	model
0	422000.0	2019	Petrol	Manual	24230	White	1st Owner	998	5	Ahmedabad	hatchback	Maruti Suzuki	S-Presso
1	585000.0	2018	Diesel	Manual	31694	White	1st Owner	1248	5	Ahmedabad	hatchback	Maruti Suzuki	Swift
2	610000.0	2017	Petrol	Manual	49072	White	1st Owner	1197	5	Ahmedabad	hatchback	Maruti Suzuki	Baleno
3	399000.0	2013	Diesel	Manual	72301	White	1st Owner	1248	5	Ahmedabad	hatchback	Maruti Suzuki	Swift
4	600000.0	2019	Petrol	Manual	21060	White	1st Owner	1197	5	Ahmedabad	hatchback	Maruti Suzuki	Swift

Pre-Processing:

Before we can proceed with the analysis, we have to check for the unique values in the categorical columns, because there is a chance of repeated categories in a dataset.

```
data['year'].unique()
```

```
array([2019, 2018, 2017, 2013, 2016, 2014, 2020, 2011, 2015, 2012, 2009,
       2010, 2008, 2021, 2003, 2006, 2007, 2005, 2004, 1998, 2002, 1991,
       1996, 1995, 1994, 2001, 2000, 1999], dtype=int64)
```

```
data['fuel'].value_counts()
```

```
Diesel                  3262
Petrol                  2783
Diesel + Diesel          227
Petrol + Petrol          193
Petrol + CNG              23
Petrol + Cng              21
LPG + Lpg                 19
CNG + Cng                 12
Diesel + CNG              11
Petrol + LPG              10
CNG + CNG                  5
Hybrid                     5
LPG + LPG                  4
Petrol + Diesel            4
Petrol + Lpg               4
CNG                        2
LPG + Petrol+Lpg           2
Hybrid + Hybrid(Ele        1
Diesel + Petrol            1
CNG + Petrol+Cng           1
Hybrid + Petrol            1
Name: fuel, dtype: int64
```

From the above image, I can see that there are few repeated categories in the '**fuel'** column, which needs to be handled before we can proceed with further steps.

I'm using numpy to handle the repeated categories in the '**fuel'** column.

```
data['fuel'] = np.where(data['fuel'].isin(['Hybrid + Hybrid(Ele','Hybrid + Petrol']),'Hybrid', data['fuel'])
```

```
data['fuel'] = np.where(data['fuel'].isin(['CNG + Petrol+Cng','Diesel + Petrol','LPG + Petrol+Lpg','Petrol + Diesel']),'Petrol', data
['fuel'])
```

```
data['fuel'] = np.where(data['fuel']=='Petrol + Lpg','Petrol + LPG',
                    np.where(data['fuel'].isin(['CNG + CNG','CNG + Cng']),'CNG',data['fuel']))
```

```
data['fuel'] = np.where(data['fuel'].isin(['LPG + LPG','LPG + Lpg']),'LPG',
                        np.where(data['fuel']=='Petrol + Cng','Petrol + CNG',
                                 np.where(data['fuel']=='Diesel + Diesel','Diesel',
                                          np.where(data['fuel']=='Petrol + Petrol','Petrol',data['fuel']))))
```

```
data['fuel'] = np.where(data['fuel']=='Petrol + LPG','LPG',
                        np.where(data['fuel']=='Diesel + CNG','CNG', data['fuel']))
```

Likewise, I'm repeating the same method for the 'owner' column, where there are repeated categories

```
data['owner'].unique()
```

```
array(['1st Owner', '2nd Owner', '3rd Owner', 'First', 'Second',
       'UnRegistered Car', 'Third', 'Fourth', '4 or More'], dtype=object)
```

```
data['owner'] = np.where(data['owner']=='1st Owner','First', np.where(data['owner']=='2nd Owner','Second',
                                                                      np.where(data['owner']=='3rd Owner','Third',
                                                                               np.where(data['owner']=='4 or More','Fourth',data['owner
'])))))
```

Now all the repeated categories within a feature has been handled. Let's check for the data types of the available feature

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6591 entries, 0 to 6590
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   price     6591 non-null   float64
 1   year      6591 non-null   int64  
 2   fuel      6591 non-null   object 
 3   trans     6591 non-null   object 
 4   mile      6591 non-null   int64  
 5   color     6591 non-null   object 
 6   owner     6591 non-null   object 
 7   engine    6591 non-null   int64  
 8   seating   6591 non-null   int64  
 9   location  6591 non-null   object 
 10  Body      6591 non-null   object 
 11  make      6591 non-null   object 
 12  model     6591 non-null   object 
dtypes: float64(1), int64(4), object(8)
memory usage: 669.5+ KB
```

I can see that the features like price, year, mile, engine and seating are of numerical type data. Further the rest of the features are of categorical type data.

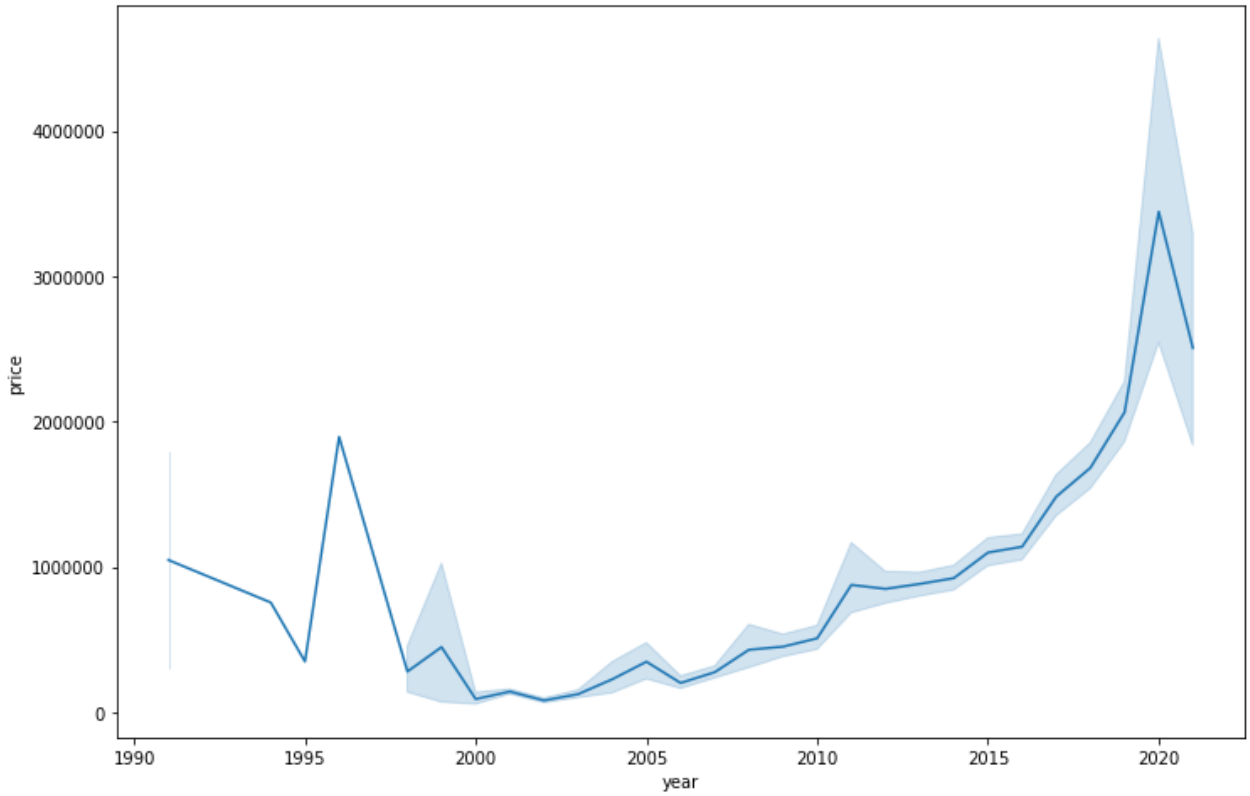
Now that we have done the pre-processing part. We can explore the data and its relationship with the target variable.

Firstly we'll visualize the relationship between the dependent variable and independent variables. I'm using seaborn library to visualize the same

- year v/s price

```
plt.figure(figsize = (12,8))
sns.lineplot(x = 'year', y = 'price', data = data, palette = 'Blues')
```

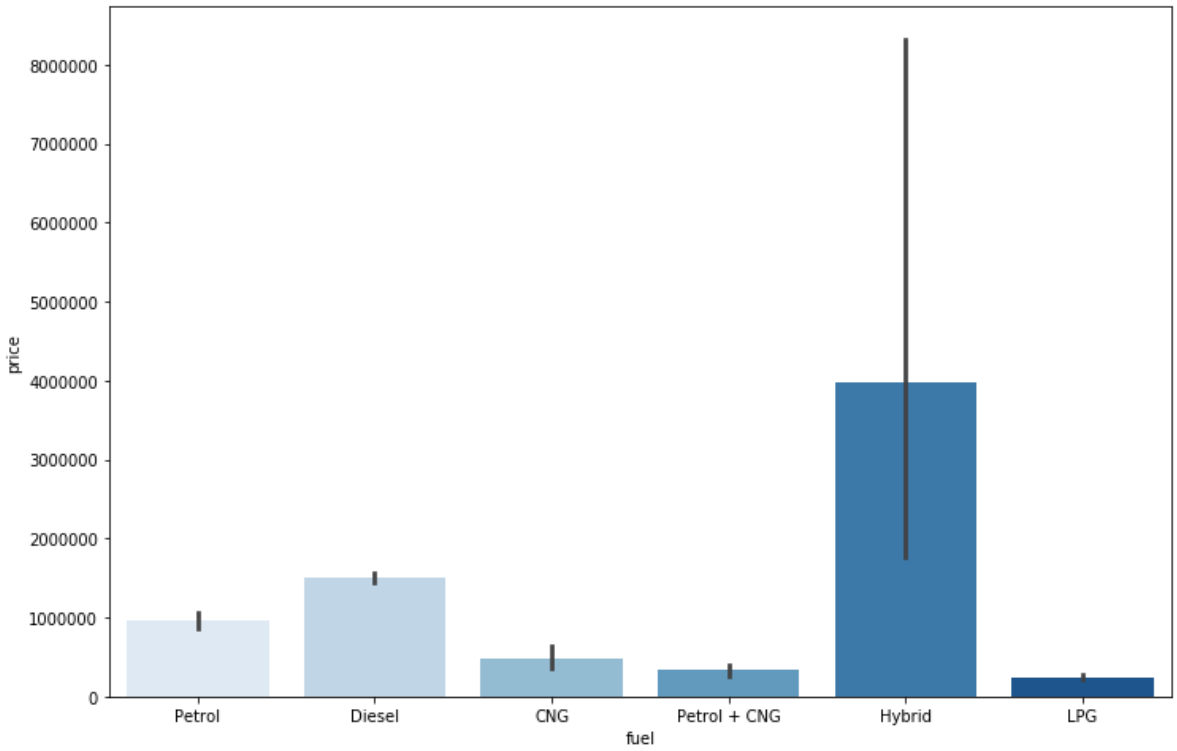
```
<matplotlib.axes._subplots.AxesSubplot at 0x1df153714e0>
```



I can say that there is a positive relationship between manufacturing year and the list price. Newer cars were sold at higher prices

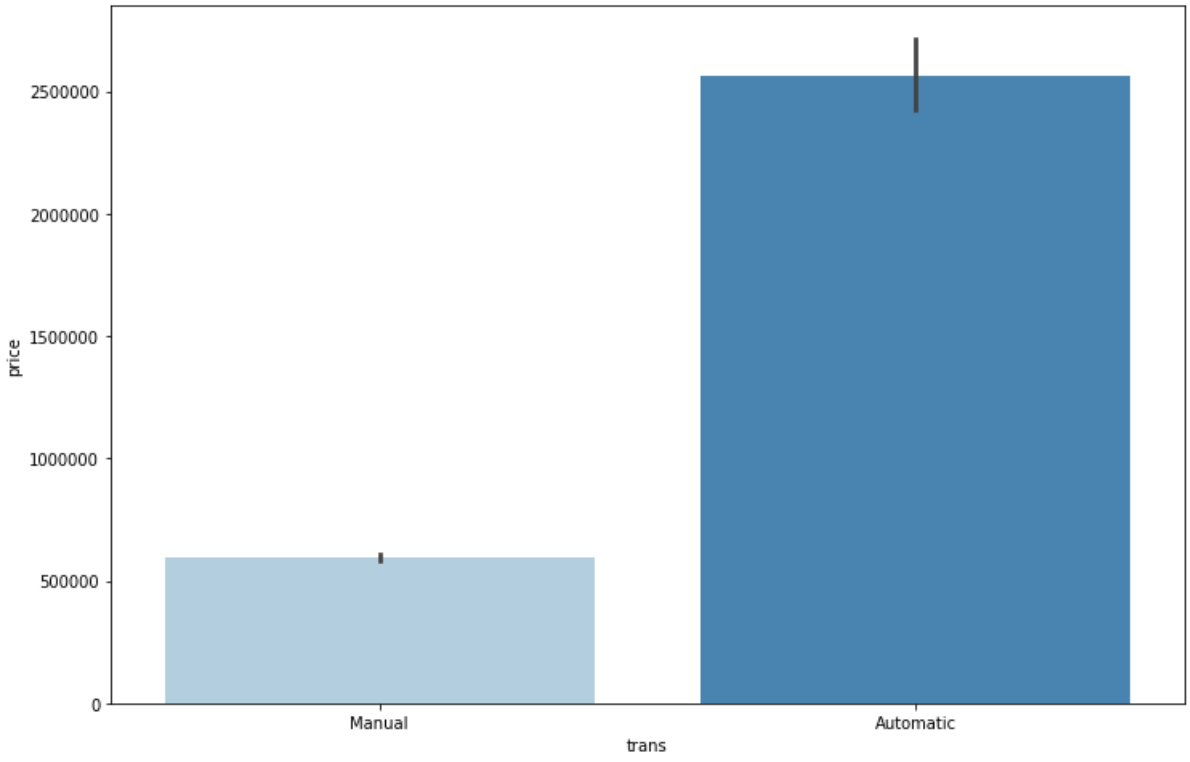
We can clearly see the increased trend for the cars manufactured after the year 2000.

- fuel v/s price

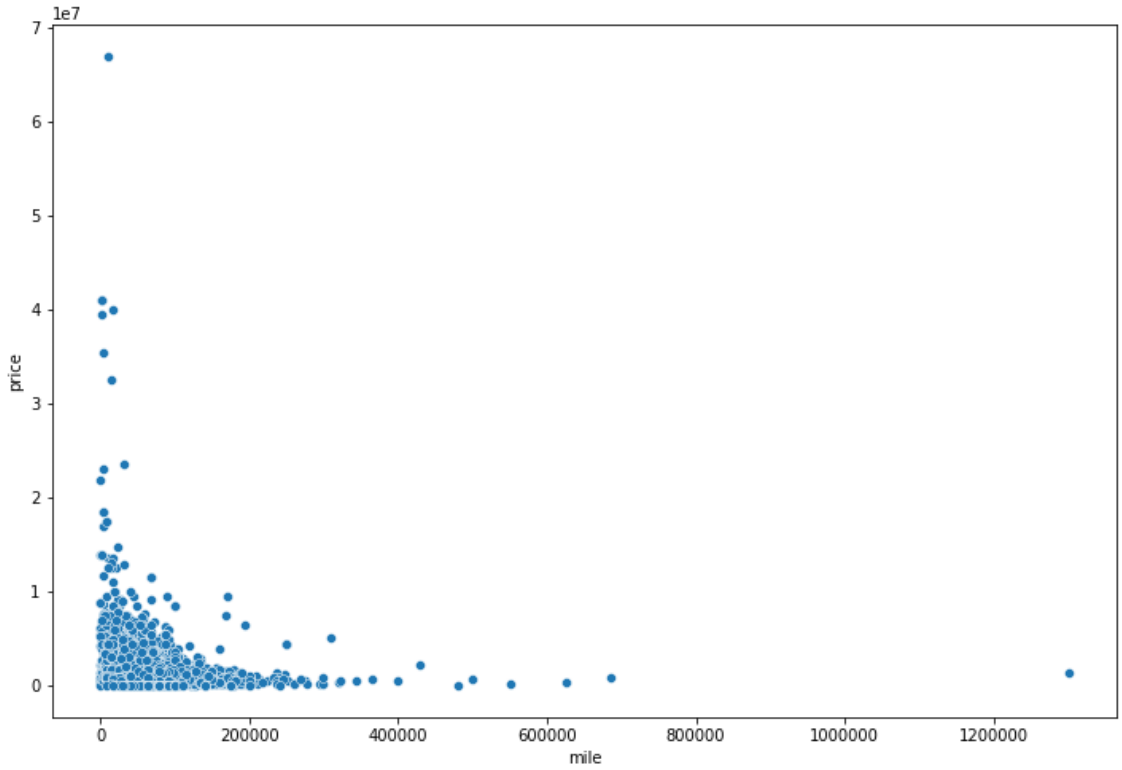


Hybrid cars were sold at higher prices followed by cars that used diesel as fuel when compared to others.

- trans v/s price

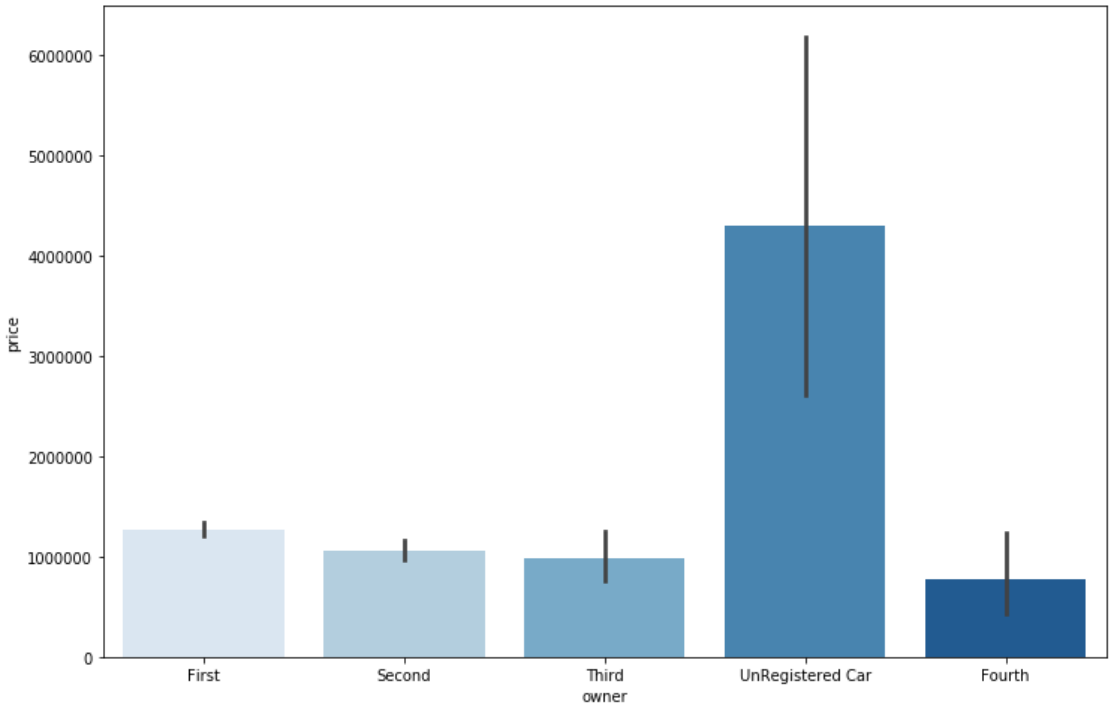


- mile v/s price



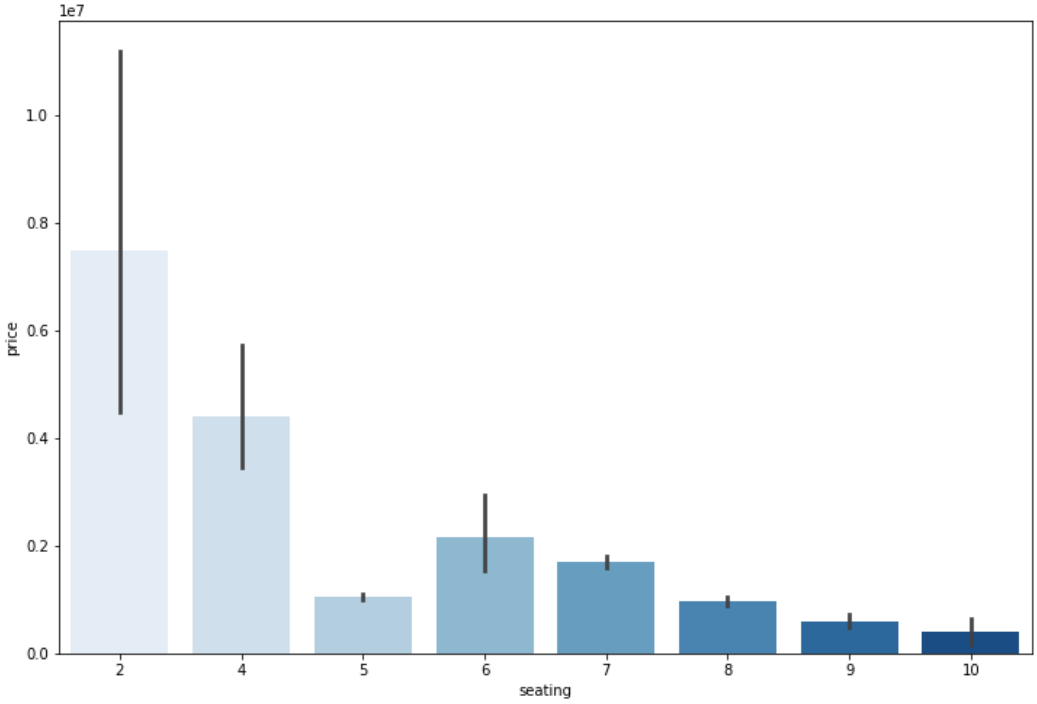
The price of the car decreases with the increased Kilometres driven for a used car

- owner v/s price



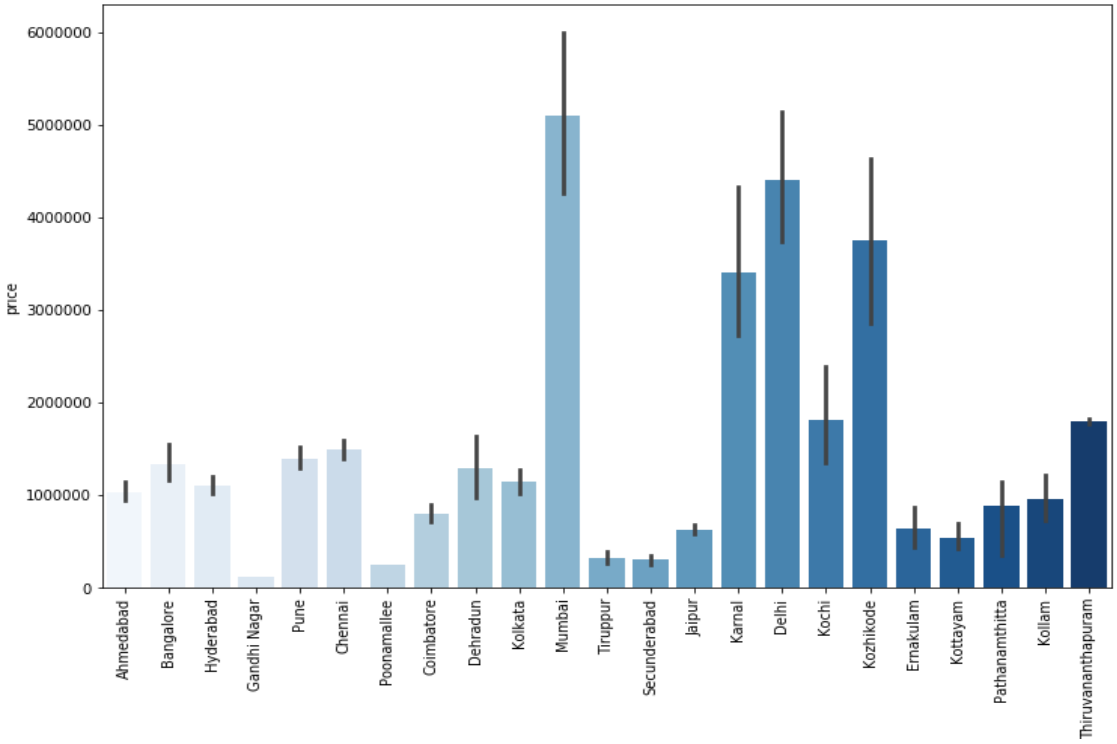
I can say that the unregistered cars were sold at higher prices followed by the cars with single owners.

- seating v/s price



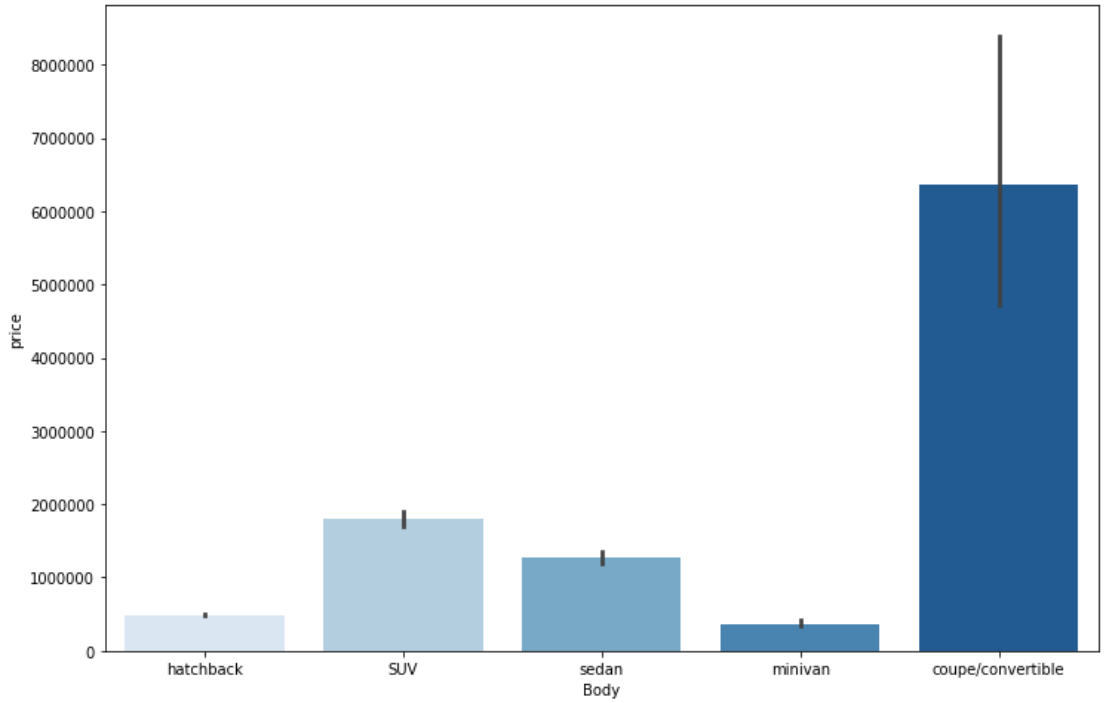
Upon reviewing the above analysis, I can say that the cars with less number of seats are sold at higher prices because most of the 2 seater cars are coupe/convertible or sports car.

- location v/s price



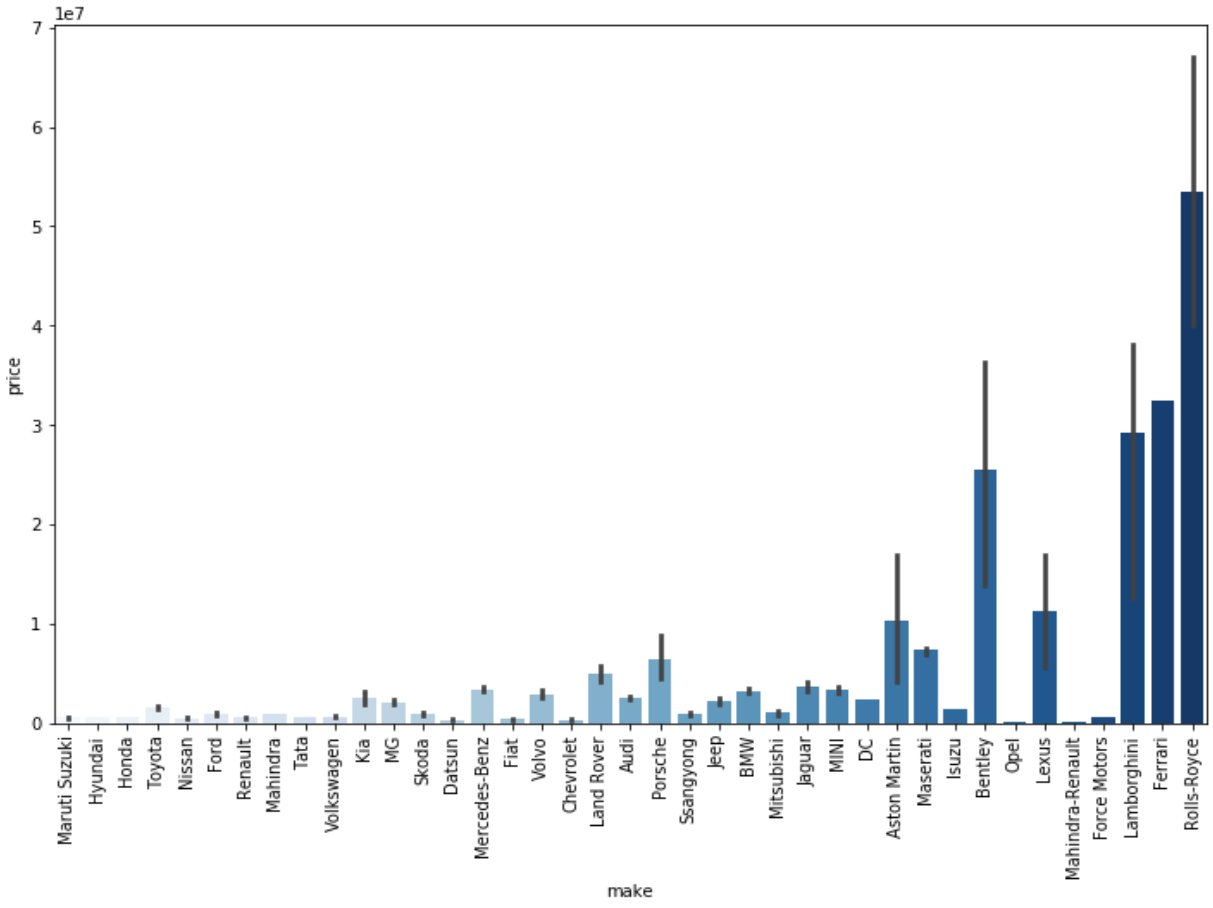
Cars at Mumbai, Delhi, kozhikode were sold at higher prices when compared to other cities

- body v/s price



From the above visual, I can see that coupe/convertible cars were sold at higher prices, followed by SUV, sedan and hatchback

- make v/s price



Before we can proceed for further analysis, I'm encoding all the categorical data using binary encoder

```
be = BinaryEncoder()
enc_data = be.fit_transform(data[['fuel','trans','color','owner','location','Body','make','model']], return_df = True)
```

Now that we have encoded the categorical data, I'm checking for correlation of the variables with the target

```
corr_data = dataset.corr()
corr_data['price'].sort_values(ascending = False)
```

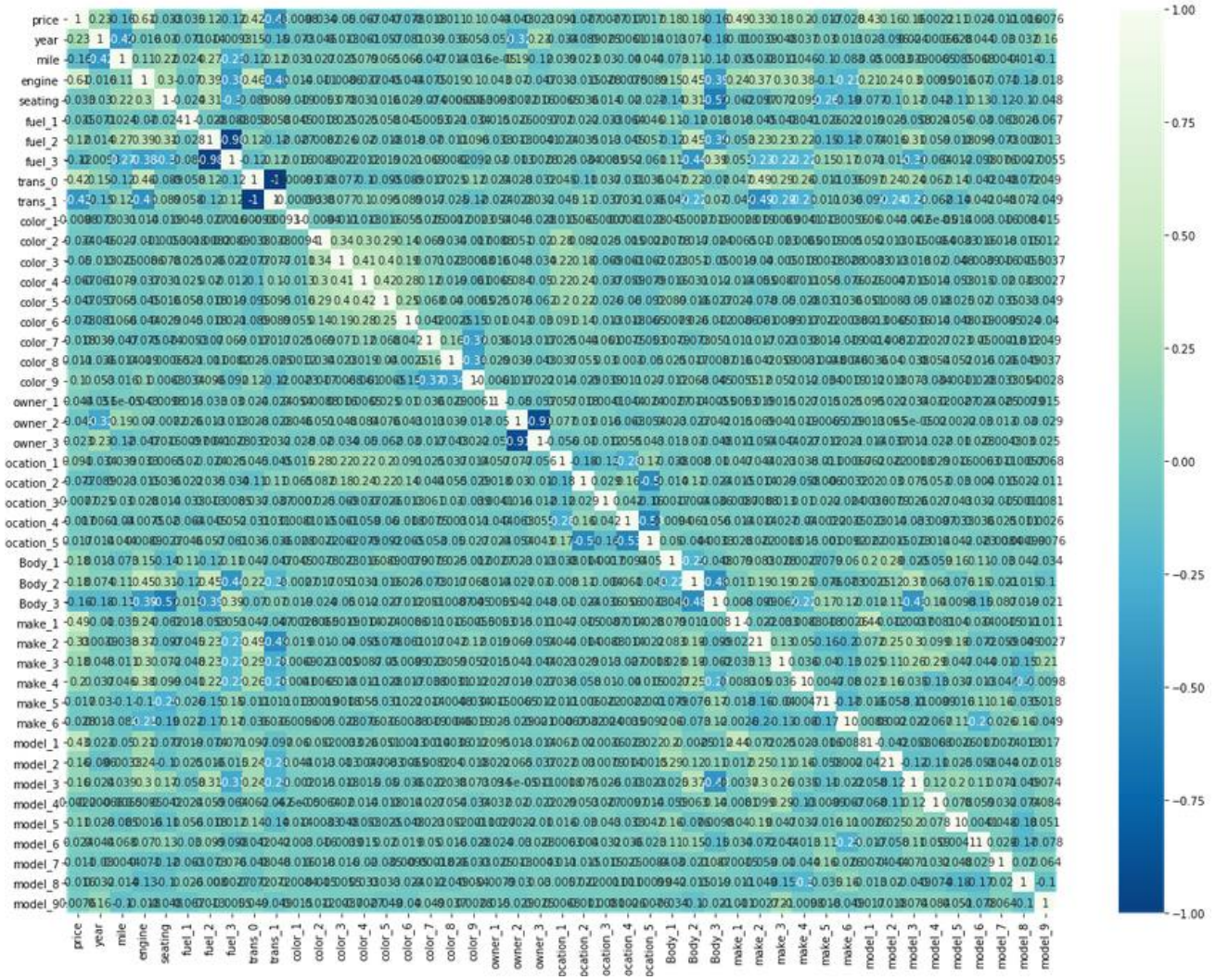
```
price          1.000000
engine         0.609159
make_1         0.493910
model_1        0.425324
trans_0        0.424015
make_2         0.328619
year           0.233362
make_4         0.196502
Body_2         0.181374
make_3         0.180000
Body_1         0.176818
model_2        0.164647
model_3        0.161236
fuel_2         0.120989
model_5        0.108403
color_9        0.104163
location_1     0.090874
owner_1        0.044015
model_6        0.023657
owner_3        0.022921
location_5     0.017361
color_8        0.011384
model_9        0.007610
model_4        0.002250
location_3    -0.007708
color_1       -0.009791
model_7       -0.010631
model_8       -0.015875
make_5        -0.017248
location_4    -0.017389
color_7       -0.017824
make_6        -0.027944
seating       -0.033450
color_2       -0.033903
fuel_1        -0.035002
owner_2       -0.042974
color_5       -0.046950
color_3       -0.049752
color_4       -0.067164
location_2    -0.077073
color_6       -0.078280
fuel_3        -0.118597
mile          -0.162956
Body_3        -0.164614
trans_1       -0.424015
fuel_0              NaN
color_0             NaN
owner_0             NaN
location_0          NaN
Body_0              NaN
make_0              NaN
model_0             NaN
Name: price, dtype: float64
```

We can see from this correlation coefficient table, that the highly correlated variables are engine, make and transmission (automatic or manual).

Further there are certain variables that do not show any correlation with the target. Hence removing these variables as they will not be useful in sale price prediction.

```
dataset = dataset.drop(columns = ['fuel_0','color_0','owner_0','location_0','Body_0','make_0','model_0'])
```

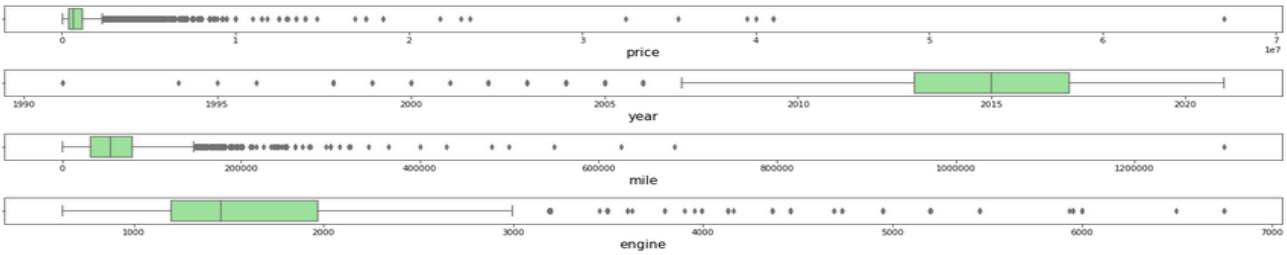
Now we can proceed further in checking for multi-collinearity in the dataset. In order to achieve that we are plotting a heatmap using the correlation table



Upon reviewing, I can see that there are few independent variables which are highly correlated with each other. However, I'm not removing them at this stage because multi-collinearity in a dataset will not affect the prediction in any manner.

Now I'm proceeding with further analysis and checking for outliers in the continuous data variables within the dataset using boxplot

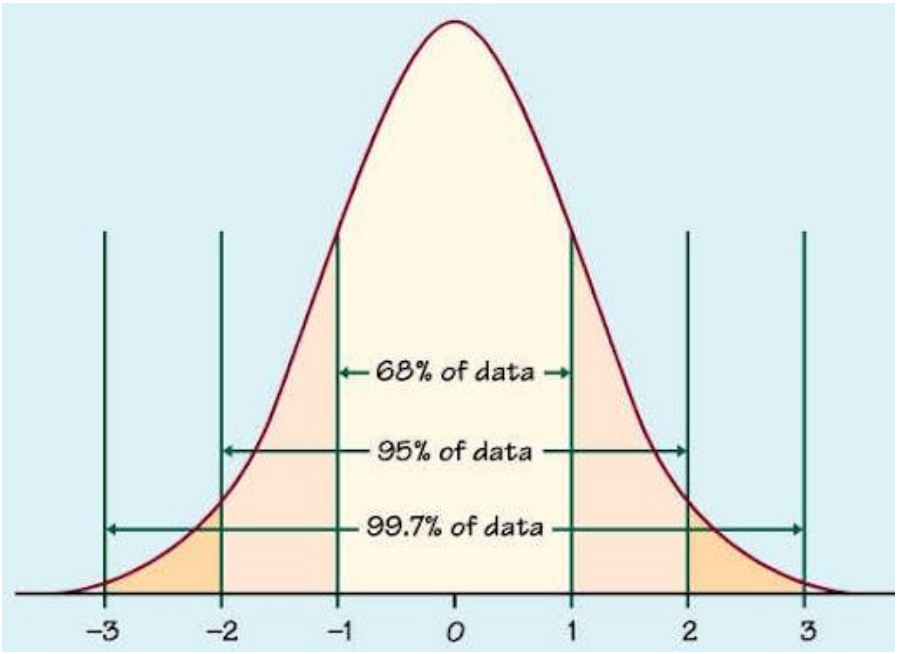
```
plt.figure(figsize = (20,60))
pltnum = 1

for i in dataset:
    if pltnum <=50:
        plt.subplot(50,1,pltnum)
        sns.boxplot(dataset[i],color = 'lightgreen')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```



I can see that there are lot of outliers in the dataset. Hence I'm proceeding with handling the outliers with the z-score method.

It is assumed that close to 99.7% data lies between -3 to +3 standard deviation. We can consider the remaining data (0.03) to be outlier.

Therefore using the z-score method, I'm taking the data within the range of -2.1 to +2.1 standard deviation to control the outlier.



```
from scipy.stats import zscore
z = np.abs(zscore(dataset[['mile','engine']]))
z.head()
```

	mile	engine
0	0.807877	0.977306
1	0.636684	0.586068
2	0.238105	0.665880
3	0.294672	0.586068
4	0.880584	0.665880

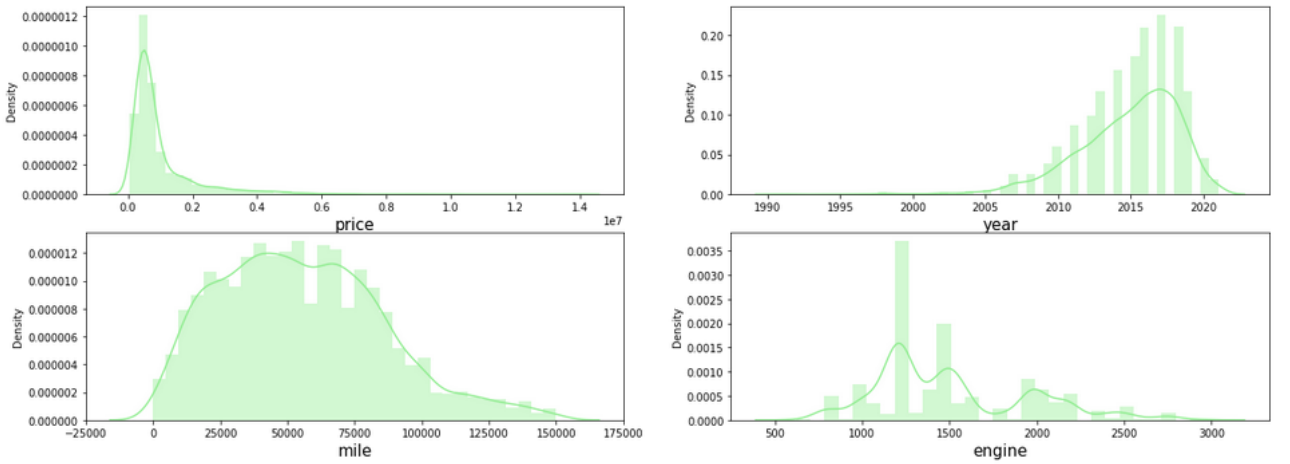
```
new_data = dataset[(z<2.1).all(axis = 1)]
print(dataset.shape)
print(new_data.shape)
```

```
(6591, 45)
(6067, 45)
```

Post outlier removal I can see that the “new_data” dataset contains 6067 rows out of the actual dataset consisting of 6591 rows. If we are to proceed with outlier then there should be only 7 to 8 % data loss and we are losing close to 8% data loss therefore we are proceeding with the outlier removal.

Now let’s check the data distribution to understand the data.

```
plt.figure(figsize = (20,15))
pltnum = 1

for i in new_data:
    if pltnum <=4:
        plt.subplot(4,2,pltnum)
        sns.distplot(new_data[i],color = 'lightgreen')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.show()
```



From the above distribution, we can clearly see that continuous data variables does contain outliers.

In order to be a good dataset, we assume that all the continuous variables follow normal distribution. However I can see that the continuous columns are skewed and we will have to control skewness to make data follow normal distribution.

Skewness coefficient:

```
x.skew()
```

```
year          -1.086255
mile           0.512053
engine         0.794256
seating        2.140643
fuel_1         8.537383
fuel_2        -0.078868
fuel_3         0.087463
trans_0        0.898188
trans_1       -0.898188
color_1       21.538891
color_2        5.367602
color_3        4.318395
color_4        3.622595
color_5        2.930888
color_6        1.800759
color_7        0.709119
color_8        0.267804
color_9       -0.285444
owner_1       10.362014
owner_2        1.561464
owner_3       -1.692777
location_1     3.255366
location_2     0.979982
location_3     0.691081
location_4    -0.641657
location_5     0.296332
Body_1         7.156380
Body_2        -0.492743
Body_3        -0.819980
make_1        31.759509
make_2         2.224353
make_3         1.097921
make_4         1.020534
make_5         0.004946
make_6        -0.053106
model_1       15.485565
model_2        1.876763
model_3        0.986368
model_4        0.456943
model_5        0.342906
model_6       -0.085479
model_7        0.053766
model_8       -0.115931
model_9        0.113280
dtype: float64
```

It is assumed that the skewness co-efficient within the range of -0.5 to +0.5 is acceptable. Proceeding with the same assumption to get the skewness under control.

In order to perform the same we are using power transformation using cube-root transformation on the entire dataset excluding the target variable.

Once performed, most of the skewness are under control except few and we are proceeding with the model building assuming that outliers is not the cause of the skewness in the dataset.

Skewness co-efficient post transformation:

```
x = np.cbrt(x)
```

```
x.skew()
```

```
year          -1.091542
mile          -0.606210
engine         0.401242
seating        1.770393
fuel_1         8.537383
fuel_2        -0.078868
fuel_3         0.087463
trans_0        0.898188
trans_1       -0.898188
color_1       21.538891
color_2        5.367602
color_3        4.318395
color_4        3.622595
color_5        2.930888
color_6        1.800759
color_7        0.709119
color_8        0.267804
color_9       -0.285444
owner_1       10.362014
owner_2        1.561464
owner_3       -1.692777
location_1     3.255366
location_2     0.979982
location_3     0.691081
location_4    -0.641657
location_5     0.296332
Body_1         7.156380
Body_2        -0.492743
Body_3        -0.819980
make_1        31.759509
make_2         2.224353
make_3         1.097921
make_4         1.020534
make_5         0.004946
make_6        -0.053106
model_1       15.485565
model_2        1.876763
model_3        0.986368
model_4        0.456943
model_5        0.342906
model_6       -0.085479
model_7        0.053766
model_8       -0.115931
model_9        0.113280
dtype: float64
```

Assumptions:

1. Although there is skewness present in some of the variables post transformation, we assume that this is not because of the presence of outliers, however it is the shape of the data itself
2. We assume that all the variables follow normal distribution
3. The multi-collinearity in the dataset will not affect the prediction.

Hardware and Software Requirements and Tools Used:

- Python version 3
- Jupyter interactive notebook
- Windows 10 professional
- Sci-kit learn Library
- Sci-py Library
- Seaborn Library
- Matplotlib Library
- Intel-core i3 processor
- 4GB RAM and 500 MB ROM
- Snipping tool

Model/s Development and Evaluation

Further, before build the model we will have to split the data to test and train. The best possible way to split the data is by finding the best random state to split and the benefit is that we can control over fitting up to certain extent before even building the model.

We are trying to match the R2 score of the training data set and the test dataset, which ever split (**random state**) satisfies the condition (**r2 score of training dataset = r2 score of testing dataset**). We'll take the same random state to split the dataset and build the model.

We are using a simple for loop to achieve the same.

```
from sklearn.linear_model import LinearRegression
rs = 0
for i in range(0,2000):
    x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = i, test_size = 0.3)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    tr_score = lr.score(x_train,y_train)
    ts_score = lr.score(x_test,y_test)
    if round(tr_score*100,1) == round(ts_score*100,1):
        if i> rs:
            rs = i
print('the best random state is', rs)
```

```
the best random state is 1975
```

Now, I can say that the best random state for the split is 1975 and we will be splitting the dataset 70% train and 30% test with the random state 1975.

I'm testing the results with the below algorithms.

1. Linear Regression
2. Random Forest Regressor
3. Extra Trees Regressor
4. XG Boost Regressor

In order to test the model, I'm using r2 score and RMSE (Root Mean Squared Error), further in order to verify the model's fit, I'm using cross val score to identify the best model.

Model 1: Linear Regression

The first Machine Learning model I'm using to predict the Sale price is Linear Regression, this gives us with better understanding of the dataset and it's a simple model to build.

```
lin = LinearRegression()
lin.fit(x_train,y_train)
lin_pred = lin.predict(x_test)
lin_score = lin.score(x_test,y_test)
lin_score
```

```
0.6497054776469693
```

```
from sklearn.metrics import mean_squared_error
lin_rmse = np.sqrt(mean_squared_error(y_test,lin_pred))
print('RMSE for Linear Regression: ', lin_rmse)
```

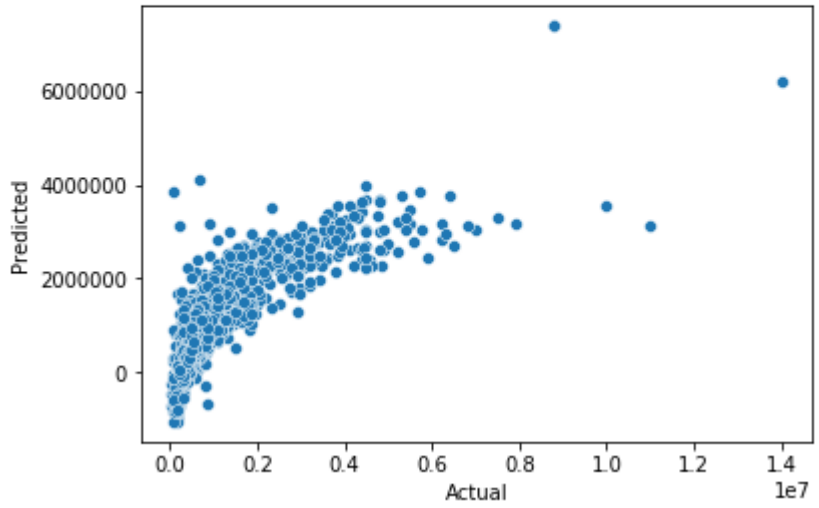
```
RMSE for Linear Regression:  692726.9642558749
```

Using the Linear Regression, we were able to get the r2 score of 0.64 and the RMSE of 692726.96

Let's visualize the Actual v/s Predicted values

```
plt.figure()
sns.scatterplot(y_test, lin_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
Text(0, 0.5, 'Predicted')
```



Further, I'm verifying the fit using cross_val_score with cross validation of 5 and check for overfitting.

```
cv = cross_val_score(lin,x,y,scoring = 'r2', cv = 5)
cv = cv.mean()
cv
```

```
0.5986767679991206
```

Model 2: Random Forest Regressor

Here, I'm using ensemble techniques to predict the outcome and I'm using Random Forest Regressor.

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr_pred = rfr.predict(x_test)
rfr_score = rfr.score(x_test,y_test)
print('The r2 for the Random Forest Regression is' , rfr_score)
```

```
The r2 for the Random Forest Regression is 0.9268305365406163
```

```
rfr_rmse = np.sqrt(mean_squared_error(y_test,rfr_pred))
print('RMSE for Random Forest Regression: ', rfr_rmse)
```

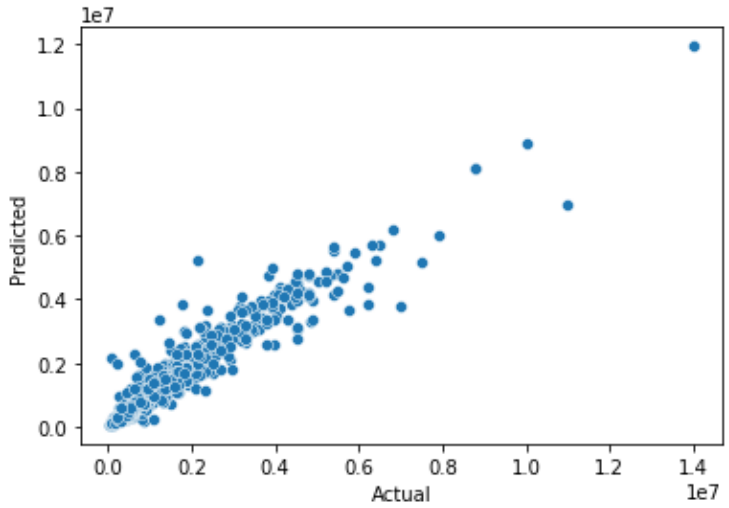
```
RMSE for Random Forest Regression:  316599.59495493484
```

Here I can see that the Random Forest Regressor, is predicting the Sale Price with r2-score of 0.93 and the RMSE of 316600

Let's visualize the Actual v/s Predicted values

```
plt.figure()
sns.scatterplot(y_test, rfr_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
Text(0, 0.5, 'Predicted')
```



Further, I'm verifying the fit using cross_val_score with cross validation of 5 and check for overfitting.

```
cv1 = cross_val_score(rfr,x,y,scoring = 'r2', cv = 5)
cv1 = cv1.mean()
cv1
```

```
0.8255794683669093
```

Model 3: Extra Trees Regressor

The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression

```
from sklearn.ensemble import ExtraTreesRegressor
et = ExtraTreesRegressor()
et.fit(x_train,y_train)
et_pred = et.predict(x_test)
et_score = et.score(x_test,y_test)
print('The r2 for the Extra Trees Regression is' , et_score)
```

```
The r2 for the Extra Trees Regression is 0.9413955228687255
```

```
et_rmse = np.sqrt(mean_squared_error(y_test,et_pred))
print('RMSE for Extra Trees Regression: ', et_rmse)
```

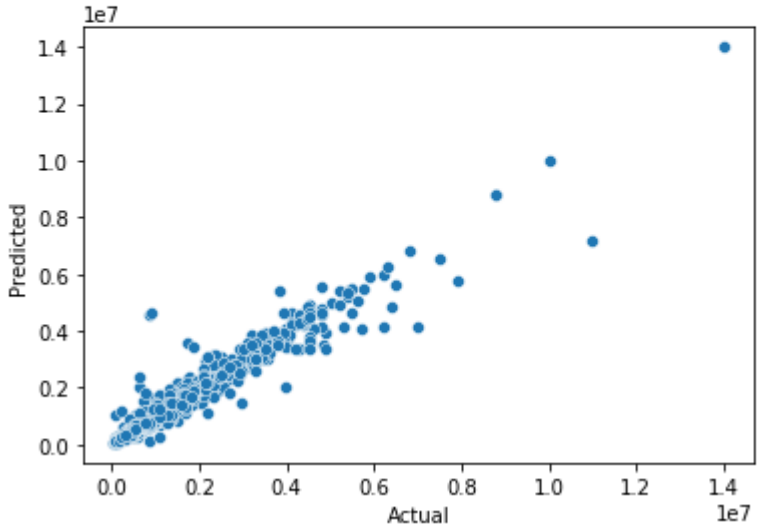
```
RMSE for Extra Trees Regression:  283341.9102025325
```

Here I can see that the Extra Trees Regressor is predicting the sale price with r2-score of 0.94 and the RMSE of 283342

Let's visualize the Actual v/s Predicted values

```
plt.figure()
sns.scatterplot(y_test, et_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
Text(0, 0.5, 'Predicted')
```



Further, I'm verifying the fit using cross_val_score with cross validation of 5 and check for overfitting.

```
cv2 = cross_val_score(et,x,y,scoring = 'r2', cv = 5)
cv2 = cv2.mean()
cv2
```

```
0.8731298419181431
```

Model 4: XG Boost Regressor

Extreme Gradient Boosting Algorithm. Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modelling problems. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models.

```
from xgboost import XGBRegressor
xgb = XGBRegressor()
xgb.fit(x_train,y_train)
xgb_pred = xgb.predict(x_test)
xgb_score = xgb.score(x_test,y_test)
print('The r2 for the XGB Regression is' , xgb_score)
```

```
The r2 for the XGB Regression is 0.9474470378211802
```

```
xgb_rmse = np.sqrt(mean_squared_error(y_test,xgb_pred))
print('RMSE for XG Boost Regression: ', xgb_rmse)
```

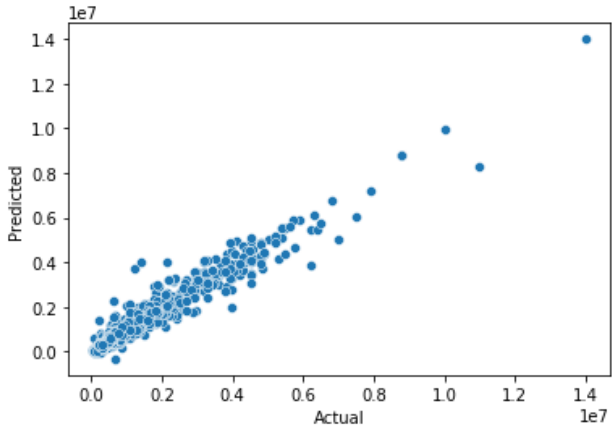
```
RMSE for XG Boost Regression:  268314.42389258975
```

Here I can see that the XG Boost Regressor, is predicting the sale price with F1-score of 0.95 and the RMSE of 268314.

Let's visualize the Actual v/s Predicted values

```
plt.figure()
sns.scatterplot(y_test, xgb_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
Text(0, 0.5, 'Predicted')
```



Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv3 = cross_val_score(xgb,x,y,scoring = 'r2', cv = 5)
cv3 = cv3.mean()
cv3
```

```
0.846205796177989
```

Finding the best model by subtracting the model's r2 score with the cross validation r2 score.

```
ml = [lin_score, rfr_score,et_score, xgb_score]
cv = [cv,cv1,cv2,cv3]
rmse = [lin_rmse, rfr_rmse, et_rmse, xgb_rmse]
model = pd.DataFrame({})
model['ml'] = ml
model['cv'] =cv
model['rmse'] = rmse
model['diff'] = model['ml'] - model['cv']
model
```

	ml	cv	rmse	diff
0	0.649705	0.598677	692726.964256	0.051029
1	0.926831	0.825579	316599.594955	0.101251
2	0.941396	0.873130	283341.910203	0.068266
3	0.947447	0.846206	268314.423893	0.101241

Here I can see that the Extra Trees model has highest accuracy and the highest r2-score. Further the model is also not overfitting.

Therefore, Extra Trees is the best Machine Learning model for the dataset. Therefore proceeding with the Hyper Parameter Tuning.

```
params = {'n_estimators':[100,200,300,400],
          'max_depth':[3,5,7,9],
          'min_samples_split':[2,3,4,5,6],
          'bootstrap':[True,False]}
```

```
gcv = GridSearchCV(ExtraTreesRegressor(),params, cv = 5, n_jobs = -1)
gcv.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True, False], 'max_depth': [3, 5, 7, 9],
                         'min_samples_split': [2, 3, 4, 5, 6],
                         'n_estimators': [100, 200, 300, 400]})
```

```
gcv.best_params_
```

```
{'bootstrap': False,
 'max_depth': 9,
 'min_samples_split': 2,
 'n_estimators': 200}
```

Final Model

```
final = ExtraTreesRegressor(max_depth =9, bootstrap = False, min_samples_split = 2, n_estimators = 200)
final.fit(x_train,y_train)
fn_pred = final.predict(x_test)
fn_score = final.score(x_test,y_test)
fn_score
```

```
0.9248627498703242
```

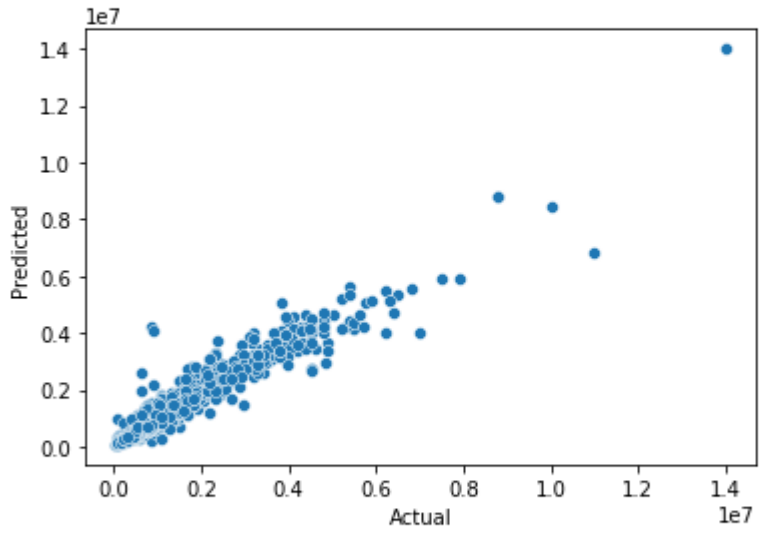
```
fn_rmse = np.sqrt(mean_squared_error(y_test,fn_pred))
print('RMSE for Hyper Tuned Extra Trees Regression: ', fn_rmse)
```

```
RMSE for Hyper Tuned Extra Trees Regression:  320828.59401267767
```

Performing the hyper parameter tuning doesn't improve the scores, the Key Metric used to finalize the model was RMSE and R2-Score. And the Extra Trees is the best model at predicting the sale price of the used cars.

```
plt.figure()
sns.scatterplot(y_test, fn_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
Text(0, 0.5, 'Predicted')
```




```
import joblib
joblib.dump(final,'car_price.pkl')
```

```
['car_price.pkl']
```

CONCLUSION

We have successfully built a model using multiple models and found that the Extra Trees Regressor model.

Below are the details of the model's metrics predicting the dataset

- R2- score of 0.92
- RMSE of 320828

You can view the same from the visualizations on the correlation of independent variable over dependent variable (target)

As we can see from the boxplot, I couldn't remove all the outliers yet since the data is expensive, I have to proceed with the dataset with outliers

Further, I couldn't get skewness under control for few variables through couple of transformation techniques, yet I have proceeded with building the model.

Looking at the heat map for correlation, I could see there were few independent variables which were correlated with each other, yet I have not removed any variable based on their correlation because multi-collinearity will not affect prediction.

Limitations of this work and Scope for Future Work.

1. Due to the presence of lot of outlier, we are unsure whether the model is going to perform well to a completely new dataset.
2. During data-collection, there are certain websites that do not provide the necessary information on the used car due to which the data collected was limited to ~6000 data points.

Other than these above limitations, I couldn't find more scope for improvement.