SECOND HAND CARS RESELLING PRICE ANALYSIS

Name - T Y Frugtniet

Student No - COADDS-21.2P -006

Table of content

Summary

Data related to second hand cars reselling prices was downloaded from kaggle.com.
Data is loaded and explored to understand the shape and size, looking for null values and dropping them as they were less than 5% of the total number.
After treating the missing values, distributions of the variables are plotted and treated for outliers.
Then variables are explored by visualizing the features against each other.
As the distributions are found to be exponential, log normal transformation is applied to obtain the linear relationship between the features.
Further, data is checked for the presence of multicollinearity and removed to avoid misleading predictions.
After such preprocessing, data was scaled and splitted into test and train data followed by the creation of a linear regression model and predicted the value and its statistics

SECOND HAND CARS RESELLING PRICE ANALYSIS

1. Introduction

The analysis on second hand cars reselling price is done based on a data set downloaded from Kaggle,com which has a shape of (4345,9) i.e there are around 4345 rows and 9 columns in the data set. The analysis is done on 7 car types, identifying the correlation between price (dependent variable) and other independent variables such as Mileage, EngineV, Year of manufacturing.
Linear Regression model is used in the analysis to identify and predict the values.

2. Implementation

2.1 Libraries

```
[ ]  import numpy as np # fundamental package for scientific computation in python
     import pandas as pd #package used for working with data sets
     import matplotlib.pyplot as plt #library for data visualization in forms of plots,graphs and charts
     import seaborn as sns #library used to visualize random distribution
     import statsmodels.api as sm #libarary used to work with statistical models
     import os  #importing python library
     sns.set()  #imporing customized seaborn themes

     from math import * #importing all the classes and functions in maths library
     import warnings  #to alert the user about some conitions in the program
     warnings.filterwarnings('ignore')
```

2.2 Reading and Understanding Raw data

2.2.1 Importing the CSV file

```
[ ] data = pd.read_csv('/content/drive/MyDrive/Machine Learning/Regression/Second Hand Car Analysis/Second hand cars reselling price.csv')
```

2.2.2 Understanding the data

The raw data set has 4345 rows of data under 9 headings. From initial reading and understanding following is observed;

A. Brand : BMW cars are more expensive than Toyota.
B. Mileage : The greater the milage the expensive the car
C. EngineV : The greater the engine volume the more expensive the car. Sports cars are more expensive than family cars
D. Year : The older the car cheaper the price

```
[ ] data.head()
```

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	Model
0	BMW	4200.0	sedan	277	2.0	Petrol	yes	1991	320
1	Mercedes-Benz	7900.0	van	427	2.9	Diesel	yes	1999	Sprinter 212
2	Mercedes-Benz	13300.0	sedan	358	5.0	Gas	yes	2003	S 500
3	Audi	23000.0	crossover	240	4.2	Petrol	yes	2007	Q7
4	Toyota	18300.0	crossover	120	2.0	Petrol	yes	2011	Rav 4

```
[ ] data.tail()
```

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	Model
4340	Mercedes-Benz	125000.0	sedan	9	3.0	Diesel	yes	2014	S 350
4341	BMW	6500.0	sedan	1	3.5	Petrol	yes	1999	535
4342	BMW	8000.0	sedan	194	2.0	Petrol	yes	1985	520
4343	Toyota	14200.0	sedan	31	NaN	Petrol	yes	2014	Corolla
4344	Volkswagen	13500.0	van	124	2.0	Diesel	yes	2013	T5 (Transporter)

```
data.shape
```

```
(4345, 9)
```

2.3 Preprocessing data

Before continuing with the rest of the process it is important to preprocess the data to obtain a more accurate output. Identifying whether there any null values and treating them accordigly is done in this step.

```
data.describe(include="all") #calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame
```

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	Model
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.000000	4345
unique	7	NaN	6	NaN	NaN	4	2	NaN	312
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN	E-Class
freq	936	NaN	1649	NaN	NaN	2019	3947	NaN	199
mean	NaN	19418.746935	NaN	161.237284	2.790734	NaN	NaN	2006.550058	NaN
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.719097	NaN
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.000000	NaN
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.000000	NaN
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.000000	NaN
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.000000	NaN
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.000000	NaN

```
data.isnull().sum() #finding how many null values in the data set
```

```
Brand             0
Price           172
Body              0
Mileage           0
EngineV         150
Engine Type       0
Registration      0
Year              0
Model             0
dtype: int64
```

```
[ ] data1 = data.drop(['Model'], axis = 1)  # there are too many repitions but lesser value addition to the model
```

```
[ ] data_no_mv =  data1.dropna(axis=0) #This is to drop null values ;to drop coloumns axis=1, to drop raws axis=0
```

```
data1.head() # when a data drop is done always print the head to confirm
```

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
0	BMW	4200.0	sedan	277	2.0	Petrol	yes	1991
1	Mercedes-Benz	7900.0	van	427	2.9	Diesel	yes	1999
2	Mercedes-Benz	13300.0	sedan	358	5.0	Gas	yes	2003
3	Audi	23000.0	crossover	240	4.2	Petrol	yes	2007
4	Toyota	18300.0	crossover	120	2.0	Petrol	yes	2011

```
data1.describe(include="all")
```

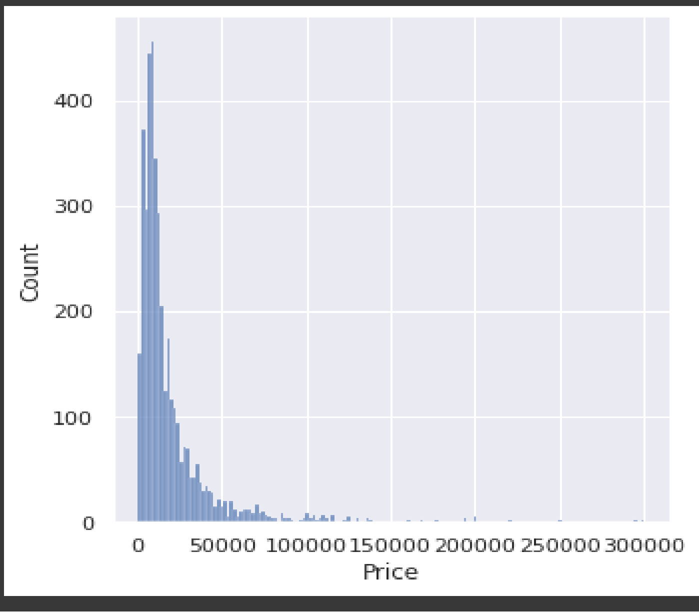
	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	4345	4173.000000	4345	4345.000000	4195.000000	4345	4345	4345.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	936	NaN	1649	NaN	NaN	2019	3947	NaN
mean	NaN	19418.746935	NaN	161.237284	2.790734	NaN	NaN	2006.550058
std	NaN	25584.242620	NaN	105.705797	5.066437	NaN	NaN	6.719097
min	NaN	600.000000	NaN	0.000000	0.600000	NaN	NaN	1969.000000
25%	NaN	6999.000000	NaN	86.000000	1.800000	NaN	NaN	2003.000000
50%	NaN	11500.000000	NaN	155.000000	2.200000	NaN	NaN	2008.000000
75%	NaN	21700.000000	NaN	230.000000	3.000000	NaN	NaN	2012.000000
max	NaN	300000.000000	NaN	980.000000	99.990000	NaN	NaN	2016.000000

2.4 Dealing with Outliers

Once the missing values are treated, it is required to address the outliers in the data which affects the accuracy of the output.
Outliers are the exceptions in the dataset. To understand the behviour both dependent and independent variables are plotted.

2.4.1 Dependent Variable - Price

```
sns.displot(data_no_mv['Price'])
plt.show()
```



It can be observed that the prices are distributed with positive skewness. To address the matter and to remove outliers quantile method is used.

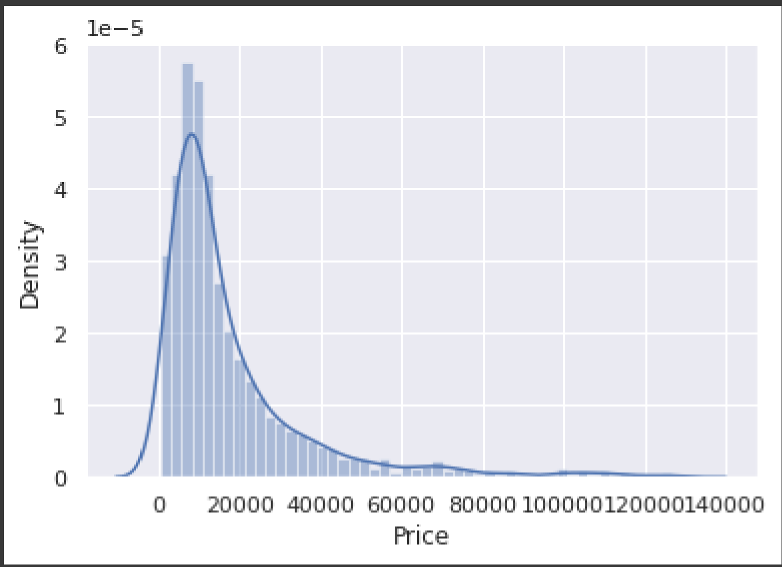
Considering 99 percentile values of the prices to address the matter.

```
q = data_no_mv['Price'].quantile(0.99)
data_1 = data_no_mv[data_no_mv['Price'] < q]
data_1.describe()
```

	Price	Mileage	EngineV	Year
count	3984.000000	3984.000000	3984.000000	3984.000000
mean	17837.117460	165.116466	2.743770	2006.292922
std	18976.268315	102.766126	4.956057	6.672745
min	600.000000	0.000000	0.600000	1969.000000
25%	6980.000000	93.000000	1.800000	2002.750000
50%	11400.000000	160.000000	2.200000	2007.000000
75%	21000.000000	230.000000	3.000000	2011.000000
max	129222.000000	980.000000	99.990000	2016.000000

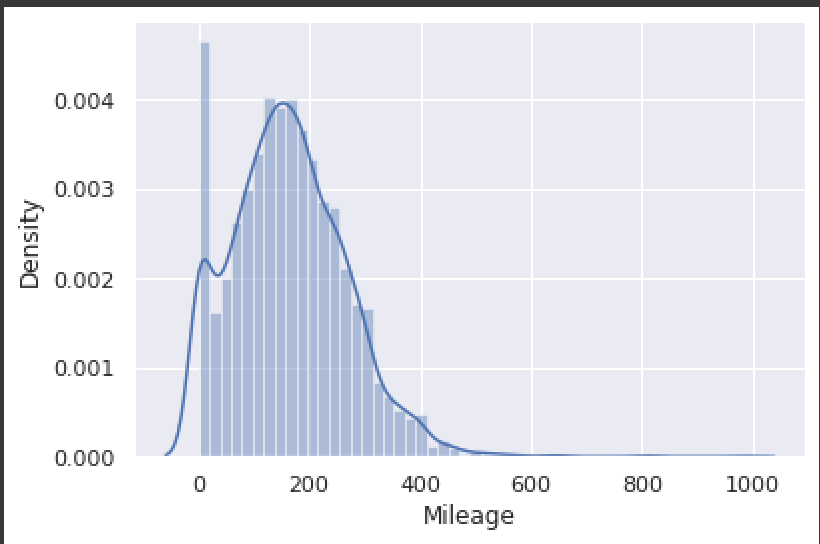
Plotting the new data after applying 99 percentiles, it is observed that the skewness is still there, which cannot be ignored.

```
sns.distplot(data_1['Price'])
plt.show()
```



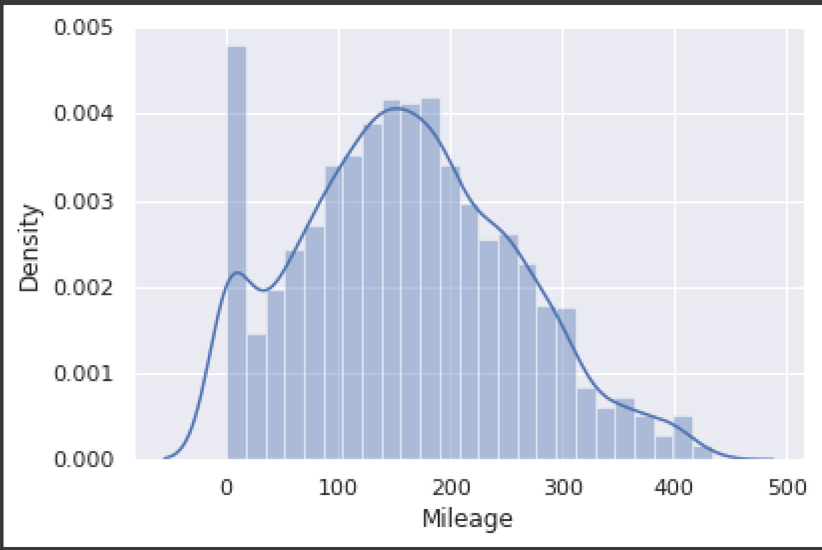
2.4.2 Independent variable 1 - Mileage

```
sns.distplot(data_no_mv['Mileage'])
plt.show()
```



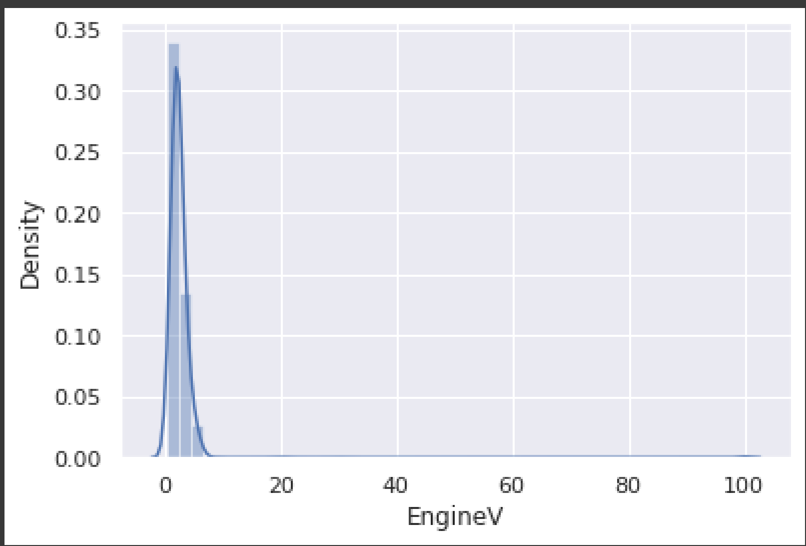
Considering 99 quantile for mileage data

```
q = data_no_mv['Mileage'].quantile(0.99)
data_2 = data_1[data_1['Mileage'] < q]
sns.distplot(data_2['Mileage'])
plt.show()
```



2.4.3 Independent variable 2 - EngineV

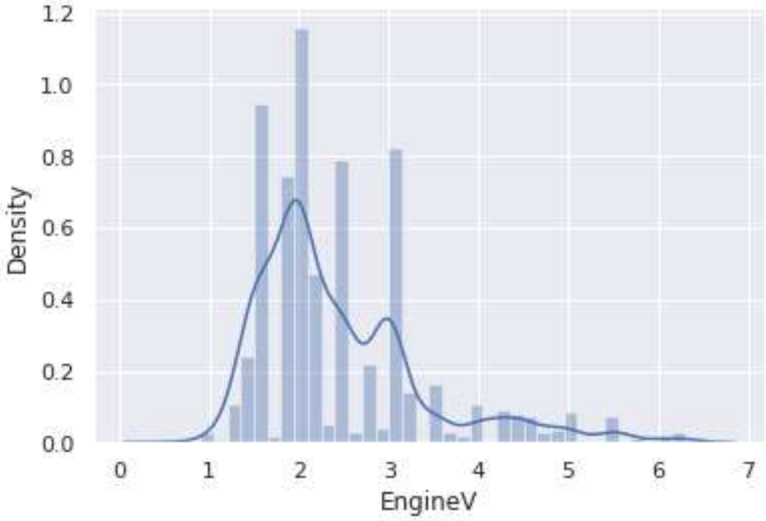
```
sns.distplot(data_no_mv['EngineV'])
plt.show()
```



There is a huge outlier in the EngineV. When observing the data manually, there is a value of 99.99 where as the others are in the range of 0.6 to 6.5
To deal with this, maximum value can be set to 6.5

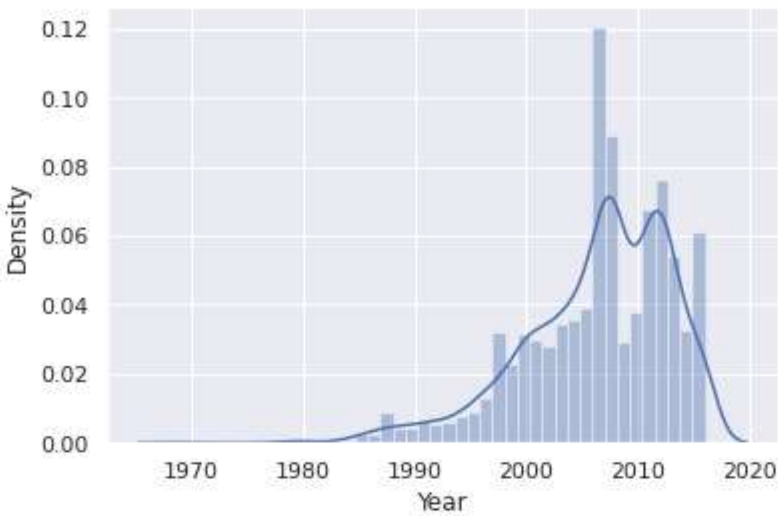
```
data_3 = data_2[data_2['EngineV'] < 6.5]
```

```
sns.distplot(data_3['EngineV'])
plt.show()
```



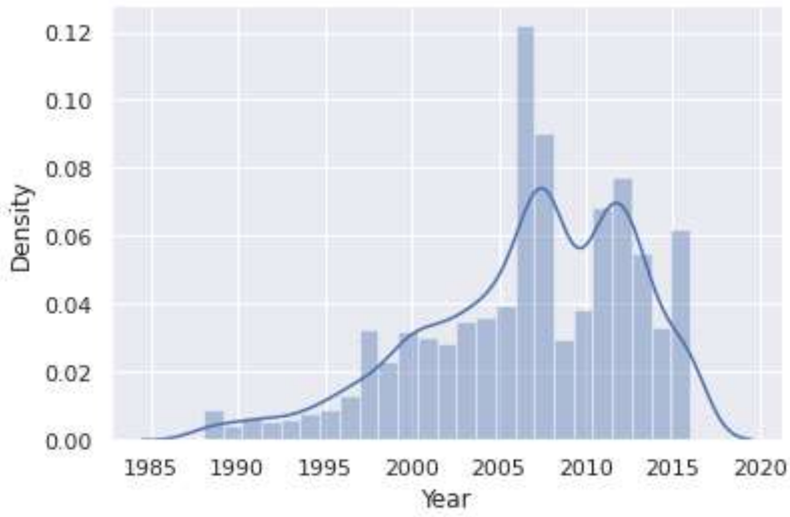
2.4.4 Independent Variable 3 - Year

```
sns.distplot(data_3['Year'])
plt.show()
```



Data is negatively skewed. 1 quantile is considered.

```
q = data_3['Year'].quantile(0.01)
data_4 = data_3[data_3['Year'] > q]
sns.distplot(data_4['Year'])
plt.show()
```



2.5 Getting the cleaned data

```
data_cleaned = data_4.reset_index(drop=True)
data_cleaned.describe()
```

	Price	Mileage	EngineV	Year
count	3867.000000	3867.000000	3867.000000	3867.000000
mean	18194.455679	160.542539	2.450440	2006.709853
std	19085.855165	95.633291	0.949366	6.103870
min	800.000000	0.000000	0.600000	1988.000000
25%	7200.000000	91.000000	1.800000	2003.000000
50%	11700.000000	157.000000	2.200000	2008.000000
75%	21700.000000	225.000000	3.000000	2012.000000
max	129222.000000	435.000000	6.300000	2016.000000

2.6 Checking the OLS Assumptions

Plotting independent variables against dependent variable “Price”.

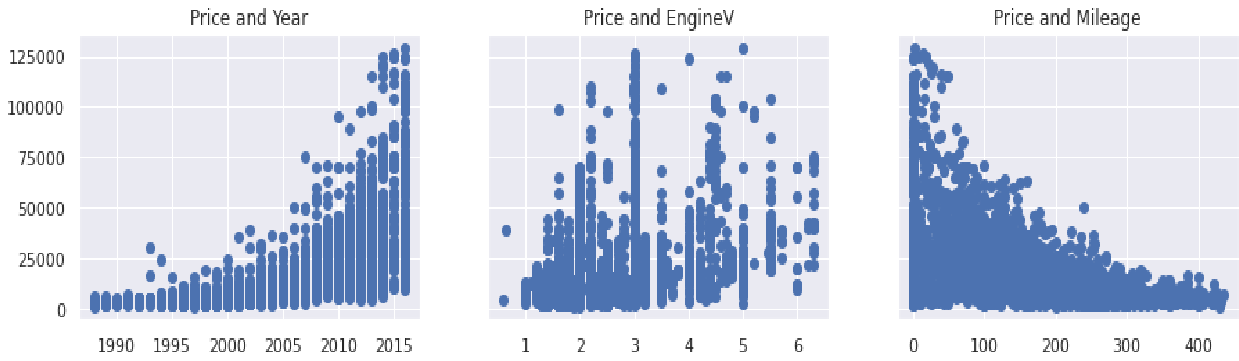
```
f,(ax1,ax2,ax3) = plt.subplots( 1, 3, sharey= True , figsize=(15,3))

ax1.scatter(data_cleaned['Year'], data_cleaned['Price'])
ax1.set_title('Price and Year')

ax2.scatter(data_cleaned['EngineV'], data_cleaned['Price'])
ax2.set_title('Price and EngineV')
```

```
ax3.scatter(data_cleaned['Mileage'], data_cleaned['Price'])
ax3.set_title('Price and Mileage')

plt.show()
```



Although a pattern is visible between dependent and independent variables, they are not linear.

2.7 Log Transformation

To make data best fit for the linear regression model data is transformed to get the linear relationship between the dependent and independent variables.

```
log_price =np.log(data_cleaned['Price'])
data_cleaned['log_price'] = log_price
data_cleaned
```

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year	log_price
0	BMW	4200.0	sedan	277	2.0	Petrol	yes	1991	8.342840
1	Mercedes-Benz	7900.0	van	427	2.9	Diesel	yes	1999	8.974618
2	Mercedes-Benz	13300.0	sedan	358	5.0	Gas	yes	2003	9.495519
3	Audi	23000.0	crossover	240	4.2	Petrol	yes	2007	10.043249
4	Toyota	18300.0	crossover	120	2.0	Petrol	yes	2011	9.814656
...	...	...	...	...	...	...	...	...	...
3861	Volkswagen	11500.0	van	163	2.5	Diesel	yes	2008	9.350102
3862	Toyota	17900.0	sedan	35	1.6	Petrol	yes	2014	9.792556
3863	Mercedes-Benz	125000.0	sedan	9	3.0	Diesel	yes	2014	11.736069
3864	BMW	6500.0	sedan	1	3.5	Petrol	yes	1999	8.779557
3865	Volkswagen	13500.0	van	124	2.0	Diesel	yes	2013	9.510445

3866 rows × 9 columns

Plotting all the independent variables against “log_price”

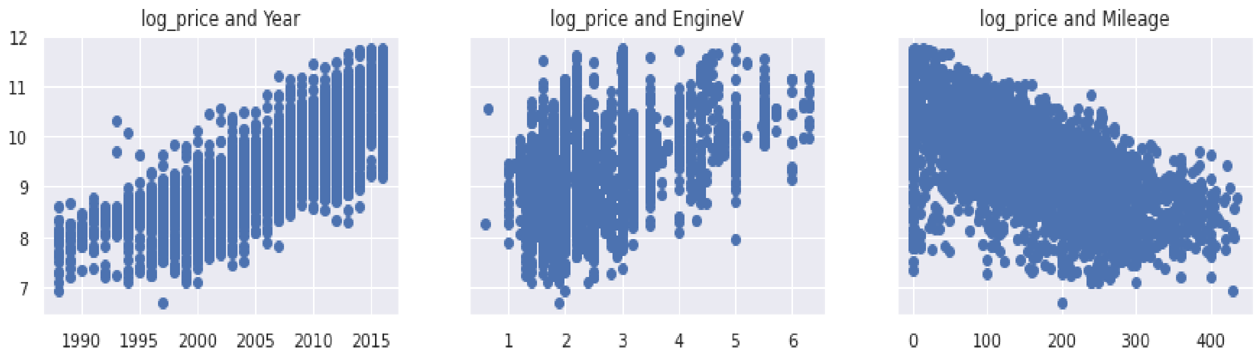
```
f,(ax1,ax2,ax3) = plt.subplots( 1, 3, sharey= True , figsize=
(15,3))
```

```
ax1.scatter(data_cleaned['Year'], data_cleaned['log_price'])
ax1.set_title('log_price and Year')

ax2.scatter(data_cleaned['EngineV'], data_cleaned['log_price'])
ax2.set_title('log_price and EngineV')

ax3.scatter(data_cleaned['Mileage'], data_cleaned['log_price'])
ax3.set_title('log_price and Mileage')

plt.show()
```



Now the pattern is more linear.

Dropping price column from the cleaned data

```
data_cleaned = data_cleaned.drop(['Price'] , axis = 1)
```

2.8 Removing Multicollinearity

Multicollinearity is a statistical concept where several independent variables in the model are correlated.

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

```
variables = data_cleaned[['Mileage','Year','EngineV']]
vif = pd.DataFrame()
vif["VIF"]= [variance_inflation_factor(variables.values ,i) for i
in range(variables.shape[1])]
vif["features"] = variables.columns
```

	VIF	features
0	3.791584	Mileage
1	10.354854	Year
2	7.662068	EngineV

Variable “Year” has high multicollinearity. This needs to be dropped.

```
data_no_multicollinearity = data_cleaned.drop(['Year'] , axis = 1)
```

2.9 Creating Dummies

All the data above are categorical. To analyse further, it needs to be converted to numerical.

```
data_with_dummies = pd.get_dummies(data_no_multicollinearity, drop_first = True)
```

```
data_with_dummies.head()
```

	Mileage	EngineV	log_price	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault	Brand_Toyota	Brand_Volkswagen	Body_hatch	Body_other	Body_sedan
0	277	2.0	8.342840	1	0	0	0	0	0	0	0	1
1	427	2.9	8.974618	0	1	0	0	0	0	0	0	0
2	358	5.0	9.495519	0	1	0	0	0	0	0	0	1
3	240	4.2	10.043249	0	0	0	0	0	0	0	0	0
4	120	2.0	9.814656	0	0	0	0	1	0	0	0	0

Rearranging the columns for better understanding of data

```
data_with_dummies.columns
```

```
cols = [ 'log_price', 'Mileage', 'EngineV', 'Brand_BMW', 'Brand_Mercedes-Benz',
       'Brand_Mitsubishi', 'Brand_Renault', 'Brand_Toyota', 'Brand_Volkswagen',
       'Body_hatch', 'Body_other', 'Body_sedan', 'Body_vagon', 'Body_van',
       'Engine Type_Gas', 'Engine Type_Other', 'Engine Type_Petrol',
       'Registration_yes']
```

```
data_preprocessed = data_with_dummies[cols]
data_preprocessed.head()
```

	log_price	Mileage	EngineV	Brand_BMW	Brand_Mercedes-Benz	Brand_Mitsubishi	Brand_Renault	Brand_Toyota	Brand_Volkswagen	Body_hatch	Body_other	Body_sedan
0	8.342840	277	2.0	1	0	0	0	0	0	0	0	1
1	8.974618	427	2.9	0	1	0	0	0	0	0	0	0
2	9.495519	358	5.0	0	1	0	0	0	0	0	0	1
3	10.043249	240	4.2	0	0	0	0	0	0	0	0	0
4	9.814656	120	2.0	0	0	0	0	1	0	0	0	0

2.10 Downloading preprocessed data

```
data_preprocessed.to_csv('data_preprocessed.csv')
data_preprocessed = pd.read_csv('data_preprocessed.csv')
```

2.11 Linear Regression Model

```
targets = data_preprocessed['log_price'] #dependent Variable
inputs   =   data_preprocessed.drop(['log_price'],   axis=   1)   #
Independent variables
```

2.12 Scaling the data

Transforming the data so that it fits within a specific scale.

```
import sklearn as sk
from sklearn.preprocessing import StandardScaler
```

```
scalar = StandardScaler()
scalar.fit(inputs)
```

```
input_scaled = scalar.transform(inputs)
```

2.13 Test - Train split

Usually the data set is splitted as 80% for training and 20% for testing. It is recommended to have 2 data sets for each.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(input_scaled,
targets,test_size = 0.25, random_state=365)
```

```
x_train
```

```
array([[ 1.52377477, -0.44490826, -0.89591293, ..., -0.16209221,
        -0.75037043,  0.32137366],
       [ 1.50137949,  0.25577519, -0.26383071, ..., -0.16209221,
        -0.75037043,  0.32137366],
       [ 0.6987327 ,  1.92904912, -0.57987182, ..., -0.16209221,
        -0.75037043, -3.11164272],
       ...,
       [-1.01405823,  0.64271979,  3.21262147, ..., -0.16209221,
         1.33267512,  0.32137366],
       [ 0.7229196 ,  1.24928159,  0.05221039, ...,  6.16932785,
        -0.75037043, -3.11164272],
       [ 1.55244073, -0.58086177, -0.47452478, ..., -0.16209221,
        -0.75037043,  0.32137366]])
```

```
y_train
```

```
3634     9.433484
3609     9.464983
2713     8.318742
1229     9.449357
1735     8.779404
           ...
428     11.074421
859     10.434116
801      9.928180
2740     7.824046
3666    10.488493
Name: log_price, Length: 2900, dtype: float64
```

2.14 Creating the Regression

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train, y_train)
```

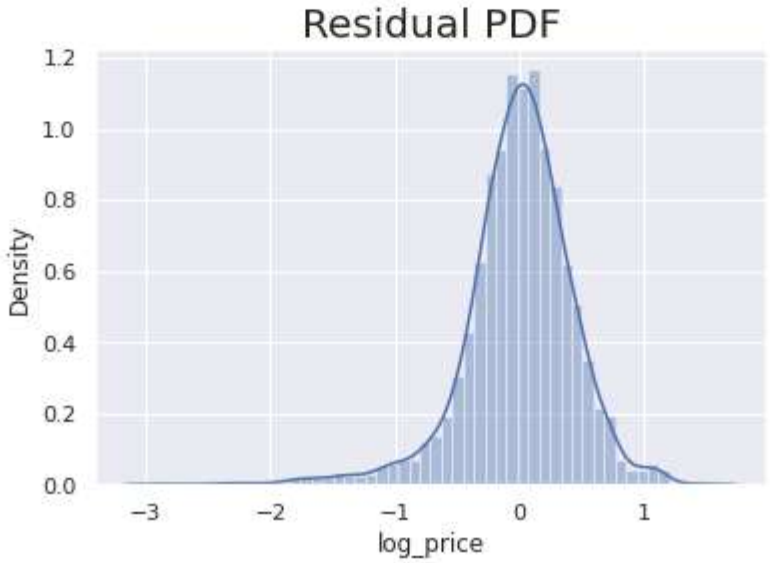
```
LinearRegression()
```

```
yhat = reg.predict(x_train)
```

```
plt.scatter(y_train,yhat)
plt.xlabel('Targets(y_train)',fontsize=20)
plt.ylabel('Predictions(yhat)', fontsize=20)
plt.xlim(6,13)
plt.ylim(6,13)
plt.show()
```

Model is optimised

```
sns.distplot(y_train - yhat)
plt.title("Residual PDF", size = 20)
plt.show()
```



2.15 Finding the weight and bias

```
reg.intercept_
```

```
9.416679846154171
```

```
reg.coef_
```

```
array([-0.00174668,   -0.44989337,      0.21269329,      0.00081477,
0.00407662,
       -0.13817763,   -0.18857294,   -0.06228124,   -0.09250456,   -
0.15037925,
       -0.09887957,   -0.19579238,   -0.12473283,   -0.16030906,   -
0.12065016,
       -0.03757449, -0.15276342,  0.3203097 ])
```

```
reg_summary   =   pd.DataFrame(inputs.columns.values    ,   columns   =
['Features'])
reg_summary['Weights'] = reg.coef_
reg_summary
```

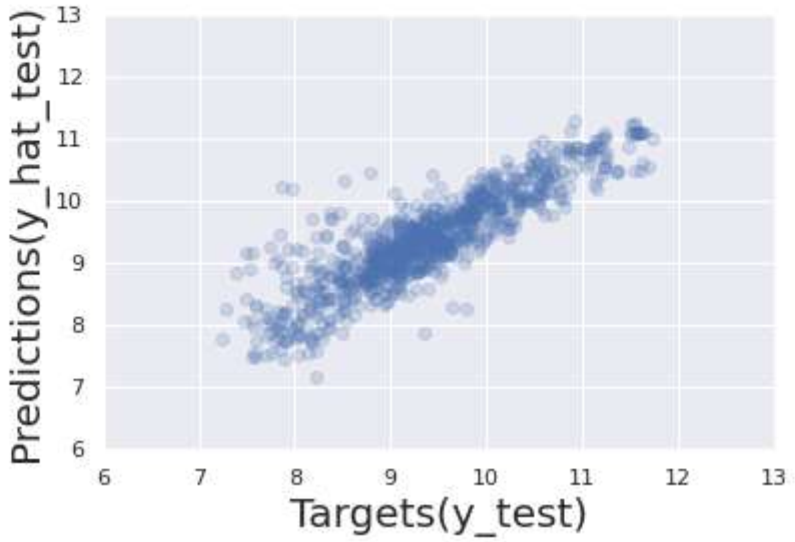
	Features	Weights
0	Unnamed: 0	-0.001747
1	Mileage	-0.449893
2	EngineV	0.212693
3	Brand_BMW	0.000815
4	Brand_Mercedes-Benz	0.004077
5	Brand_Mitsubishi	-0.138178
6	Brand_Renault	-0.188573
7	Brand_Toyota	-0.062281
8	Brand_Volkswagen	-0.092505
9	Body_hatch	-0.150379
10	Body_other	-0.098880
11	Body_sedan	-0.195792
12	Body_vagon	-0.124733
13	Body_van	-0.160309
14	Engine Type_Gas	-0.120650
15	Engine Type_Other	-0.037574
16	Engine Type_Petrol	-0.152763
17	Registration_yes	0.320310

Positive values indicates that it is directly proportional to the target variable and negative value indicates it is inversely propotional to the target variable.

“Audi” brand was considered as the benchmark varible and positive value of the brand means it is more expensive than “Audi” brand; negative value indicates cheaper than Audi.

```
y_hat_test = reg.predict(x_test)
```

```
plt.scatter(y_test , y_hat_test , alpha = 0.2)
plt.xlabel('Targets(y_test)',fontsize=20)
plt.ylabel('Predictions(y_hat_test)',fontsize=20)
plt.xlim(6,13)
plt.ylim(6,13)
plt.show()
```




```
df_pf = pd.DataFrame(np.exp(y_hat_test) , columns = ['Predictions'])
df_pf.head()
```

	Predictions
0	10691.158455
1	8199.295244
2	6748.051995
3	7566.974426
4	11356.875404

```
y_test = y_test.reset_index(drop=True)
df_pf['Target'] = np.exp(y_test)
df_pf
```

	Predictions	Target
0	10691.158455	2300.0
1	8199.295244	13200.0
2	6748.051995	8100.0
3	7566.974426	6400.0
4	11356.875404	9150.0
...	...	...
962	3716.355928	2500.0
963	10799.962127	16500.0
964	30587.755733	40500.0
965	11283.414276	8200.0
966	9460.882129	3799.0

967 rows × 2 columns

```
df_pf['Residual'] = df_pf['Target'] - df_pf['Predictions']
df_pf['Difference%']                                                    =
np.absolute(df_pf['Residual']/df_pf['Target']*100)
df_pf
```

	Predictions	Target	Residual	Difference%
0	10691.158455	2300.0	-8391.158455	364.832976
1	8199.295244	13200.0	5000.704756	37.884127
2	6748.051995	8100.0	1351.948005	16.690716
3	7566.974426	6400.0	-1166.974426	18.233975
4	11356.875404	9150.0	-2206.875404	24.118857
...	...	...	...	...
962	3716.355928	2500.0	-1216.355928	48.654237
963	10799.962127	16500.0	5700.037873	34.545684
964	30587.755733	40500.0	9912.244267	24.474677
965	11283.414276	8200.0	-3083.414276	37.602613
966	9460.882129	3799.0	-5661.882129	149.036118

967 rows × 4 columns

```
df_pf.describe()
```

	Predictions	Target	Residual	Difference%
count	967.000000	967.000000	967.000000	967.000000
mean	15925.759374	18051.489235	2125.729861	36.434191
std	13262.949089	19925.420780	10344.970349	63.821652
min	1306.918112	1400.000000	-28629.745414	0.019869
25%	7421.393407	6900.000000	-2011.243983	10.499259
50%	11666.953948	11200.000000	127.603461	23.661753
75%	19524.915598	20488.770000	3146.612792	38.563680
max	79615.045883	124000.000000	80413.626919	972.068508

It is observed that although most of the outcomes are inline still there are a large number of outliers and the model did not perform well in this regard.

```
pd.options.display.max_rows = 999
df_pf.sort_values(by=['Difference%'])
```

	Predictions	Target	Residual	Difference%
280	12496.516625	12499.00	2.483375	0.019869
954	3097.719431	3100.00	2.280569	0.073567
698	30440.703675	30500.00	59.296325	0.194414
242	7516.834736	7500.00	-16.834736	0.224463
528	18753.469531	18800.00	46.530469	0.247502
574	12963.452305	13000.00	36.547695	0.281136
391	51866.080110	52055.25	189.169890	0.363402
317	11547.584987	11500.00	-47.584987	0.413782
114	27435.625390	27300.00	-135.625390	0.496796
836	16886.199255	16800.00	-86.199255	0.513091
379	17893.553040	17800.00	-93.553040	0.525579
127	23072.396539	23200.00	127.603461	0.550015
172	10864.683858	10800.00	-64.683858	0.598925
601	34350.301141	34600.00	249.698859	0.721673
264	11415.563024	11500.00	84.436976	0.734235
907	3832.913503	3800.00	-32.913503	0.866145
872	9789.000507	9700.00	-89.000507	0.917531
782	4542.440254	4500.00	-42.440254	0.943117
84	37868.628030	37500.00	-368.628030	0.983008
742	17172.688587	16999.00	-173.688587	1.021758
931	3082.036557	3050.00	-32.036557	1.050379
949	21567.043224	21800.00	232.956776	1.068609
952	17311.577974	17500.00	188.422026	1.076697

3. Concussion

From the above analysis it can be concluded that although the model fits the data set and predicts values fairly well it can be enhanced up to an outstanding level by more testing.