

MACHINE LEARNING



SECOND HAND CARS RE-SELLING PRICE ANALYSIS REPORT

Jayasanka Hettiarachchi – ADDS21.2P 017

16 Jan 2023

Contents

Introduction	1
Implementation	1
LOADING THE RAW DATA	2
DATA PREPROCESSING	3
DETERMINING THE VARIABLES OF INTEREST	4
DEALING WITH OUTLIERS	5
CHECKING THE OLS ASSUMPTIONS	10
LOG TRANSFORMATION	10
REMOVING MULTICOLLINEARITY	13
CREATING DUMMIES	12
DOWNLOADING PREPROCESSED DATA	13
LINEAR REGRESSION MODEL	13
SCALING THE DATA	13
TEST TRAIN SPLIT	14
CREATING THE REGRESSION	15
Conclusion	16

Second Hand Cars Re-selling Price Analysis

Introduction

Manufacturers set the prices for new cars in the industry with additional costs added in the form of government taxes. As a result of that customers can feel confident in the value of their investment when purchasing a new car. However due to the high cost of new cars and many customers' inability to afford them, the market for used cars is on the rise globally. Predicting the prices of used cars is a crucial task as customers can be taken advantage of by unscrupulous sellers setting unrealistic prices. A system for determining the worth of used cars through various features is therefore essential. In developed countries, many people opt to purchase cars on a lease basis, which allows for resale of the car once the lease agreement is completed. Therefore, reselling has become an important aspect of the the world right now.

The resale price of a second-hand car refers to the amount of money that a used car can be sold for on the market. The value of a used car can be affected by several factors including the model and brand of the car, the body of the car, the mileage on the car as well as Engine version with engine type and the year of the car. Based on that fact, this report is going to present a project of analyzing the price of the secondhand car. Here 4500 rows data have been taken into consideration for the Analysis.

Implementation

This Project is based on the Machine learning & python language. For applying python language Google colab has been used for the analysis. Data analysis of Second Hand Cars Re-selling Price has been done as below figures published in the report. The data set of Second Hand Cars Re-selling Price was taken from Kaggle.com website which provides a platform for users to discover and utilize datasets.

Libraires used for the Data Analysis here,



```
Second Hand Cars Re-selling Price Analysis

[5] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

import numpy as np #It is used for mathematical function & woking with arrays
import pandas as pd #It is used for reading dataset & analysing data
import matplotlib.pyplot as plt #It is used for creating figures , plotting area in figure(Data Visualaization)
import seaborn as sns #It is used for high level interface of Data Visualaization
import statsmodels.api as sm #It is used for statistical models, as well as for conducting statistical tests, and statistical data exploration
import os #We are importing python language
sns.set() #For customizing theme

from math import * # for all mathematical function
import warnings # for ignoring warnings
warnings.filterwarnings('ignore')
```

Main libraries used for,

- Importing numpy library is used for mathematical function & working with arrays
- Pandas library here to read the data set as well as for analyzing purpose
- Matplotlib.pyplot has been imported to create figures as Data visualization
- Seaborn is also here for Data visualization with High level
- Here Statical models are used ,then statmodels.api library was imported
- For importing python language here ,imported OS

LOADING THE RAW DATA

```
[ ] data = pd.read_csv('/content/drive/MyDrive/ML - 02/Second hand cars reselling price.csv')

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)
```

In the Data set of Secondhand Cars Re-selling Price has 4345 Rows & 09 Columns, When Considering 09 Columns as variables,

- I. Brand
- II. Price
- III. Body
- IV. Mileage
- V. EngineV
- VI. Engine Type
- VII. Registration
- VIII. Year
- IX. Model

Here Price has become the main variable that we discuss throughout the Report, from Data, based on few variables secondhand car reselling price depends as follow,

Brand : The BMW car will be expensive than the Toyota.

Mileage : The greater the mileage the expensive the car.

EngineV : The greater the engine volumn the expensive the car. As sports cars are expensive than the family car.

Year : The older the car the cheap its price.

DATA PREPROCESSING

Why We want to preprocess data here is Data obtained from the real-world is often disorganized and can contain inconsistencies due to being generated, handled and saved by various people, systems and applications. Therefore, a collection of data may lack specific information, have mistakes introduced by human input, or have duplicate entries or synonyms for the same concept. These issues can be identified and corrected by humans during their everyday use of the data for business purposes.

▼ Preprocessing

```
[ ] data.describe(include='all') #It is used for calculating some statistical data like percentile, mean and std of the nume
```

	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 there
```

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

Found some missing values ,after checking Price & EngineV ,Both variables have missing values.

DETERMINING THE VARIABLES OF INTEREST

```
[ ] data1=data.drop(['Model'], axis = 1)
```

Identified “Model” column is not significant to the analysis done here as having many unique values there can be a lot mismatches. Then dropped the “Model” column as above figure.

```
data1.head()
```

	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

Here showing “Model” column is not getting further & There are 02 columns indicating missing values.

Missing values percentage is 5% ,Need to drop these 02 columns.

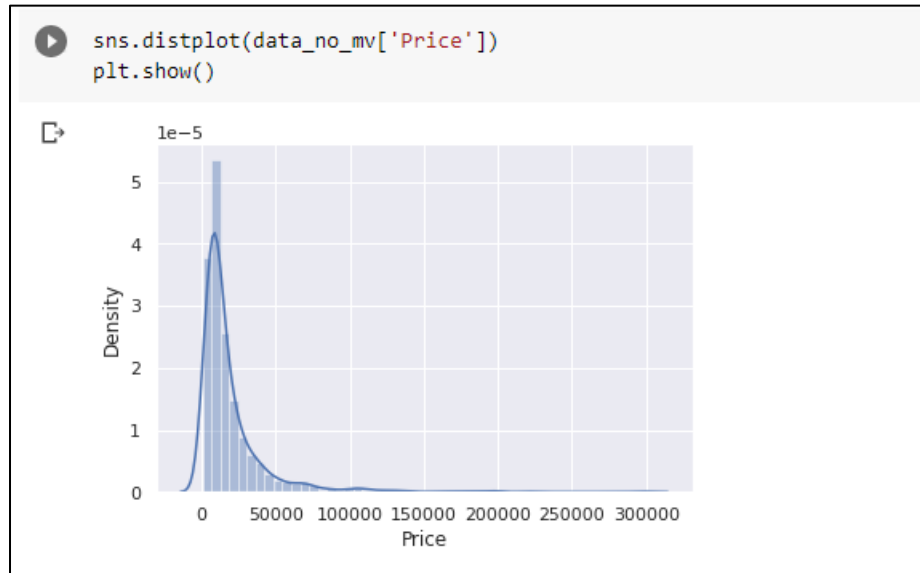
```
[ ] data_no_mv = data1.dropna(axis=0) #if axis=0 rows & if axis=1 columns
```

```
data_no_mv.isnull().sum()
```

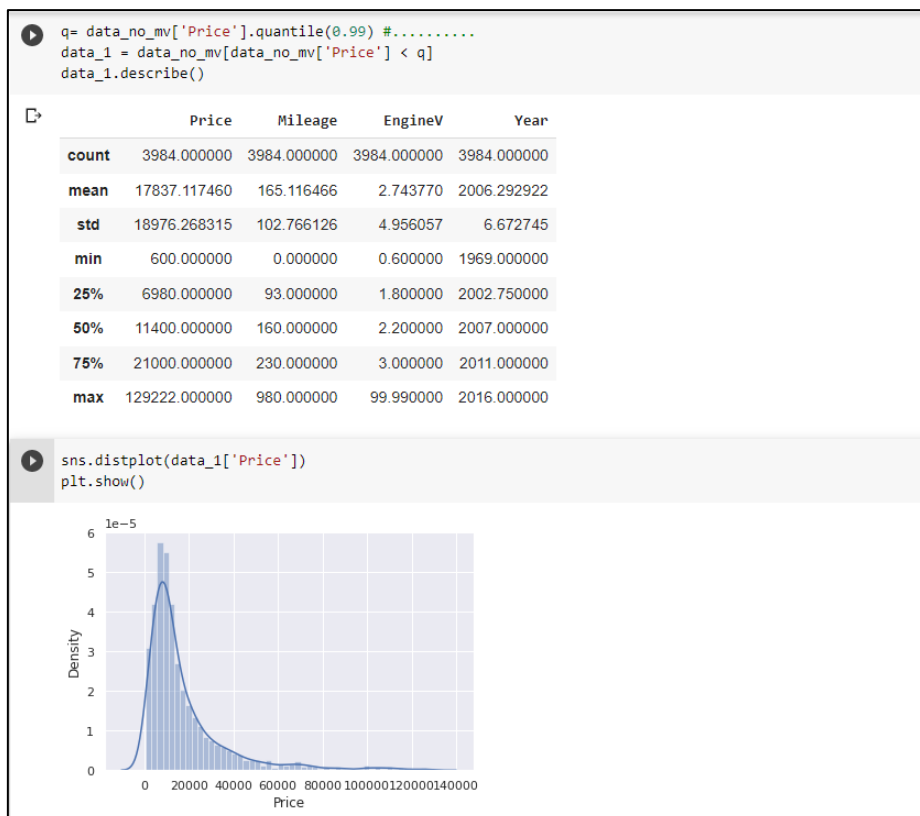
Brand	0
Price	0
Body	0
Mileage	0
EngineV	0
Engine Type	0
Registration	0
Year	0
dtype:	int64

There are no missing values hereafter with data of Second Hand Cars Re-selling Price.

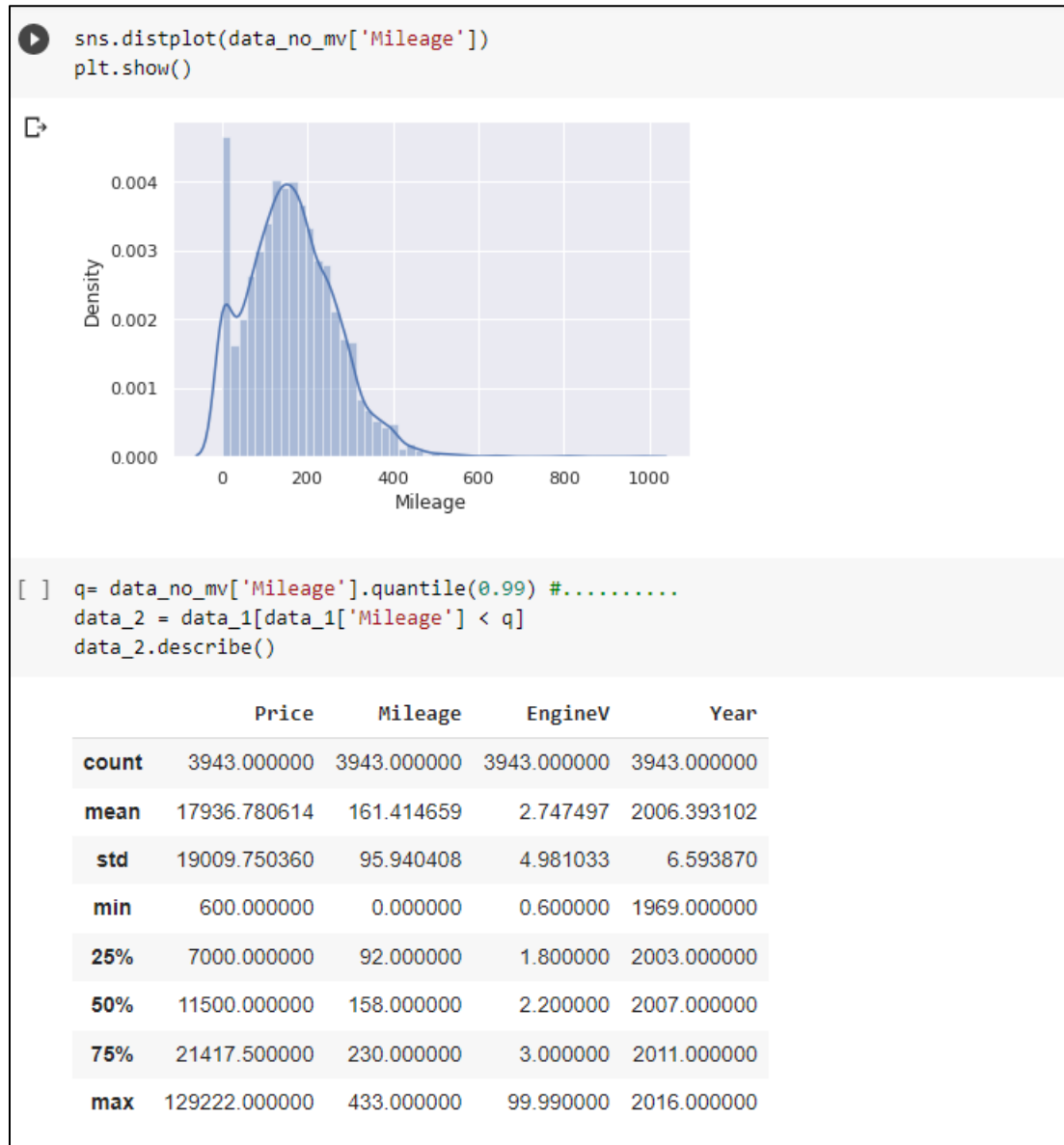
DEALING WITH OUTLIERS



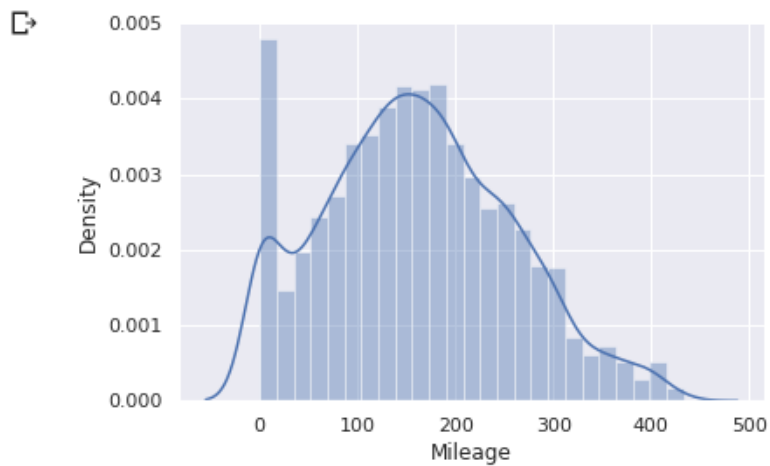
Positive skewness can be seen after prices are distributed but, in the regression, outliers need to be removed others it will lead to have an issue with for the regression.



For removing outliers , quantile method was used by keeping 99 percentile value for the price. Above figure shows it from the price, it has decreased up to certain level. After that further Positive skewness can be seen in PDF for prices. Further applied the same for “Mileage” as below.

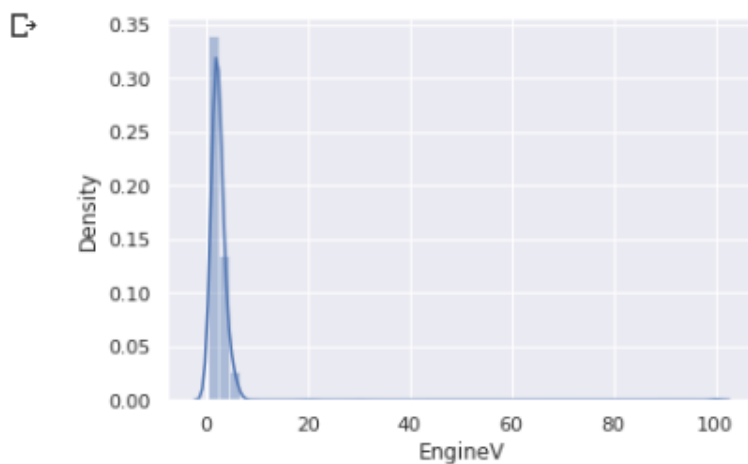



```
sns.distplot(data_2['Mileage'])  
plt.show()
```

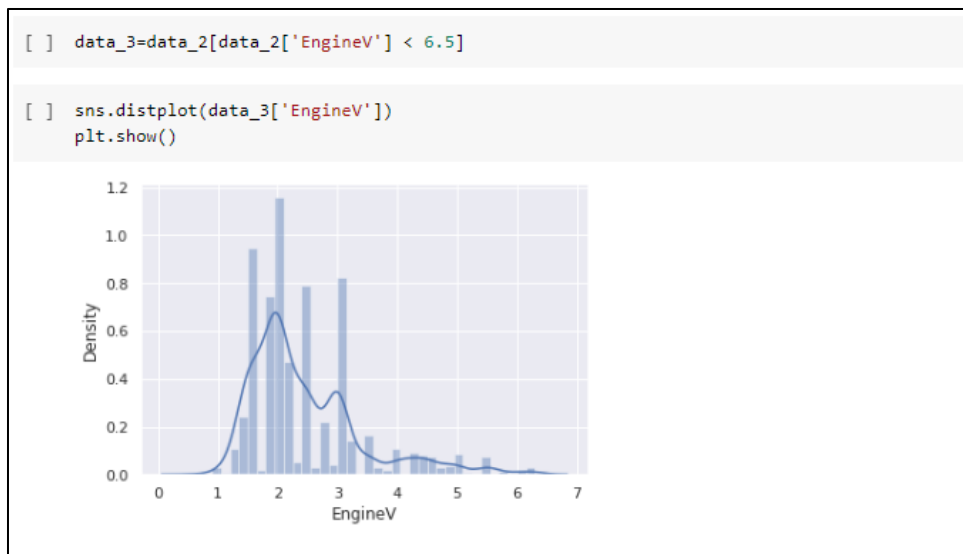


Found a lot of outliers from “EngineV” & after checking the data there were some values were equal to 99.99. Based on the general fact volume of engine can’t be more than 6.5 but we had values beyond that. It shows below figure.

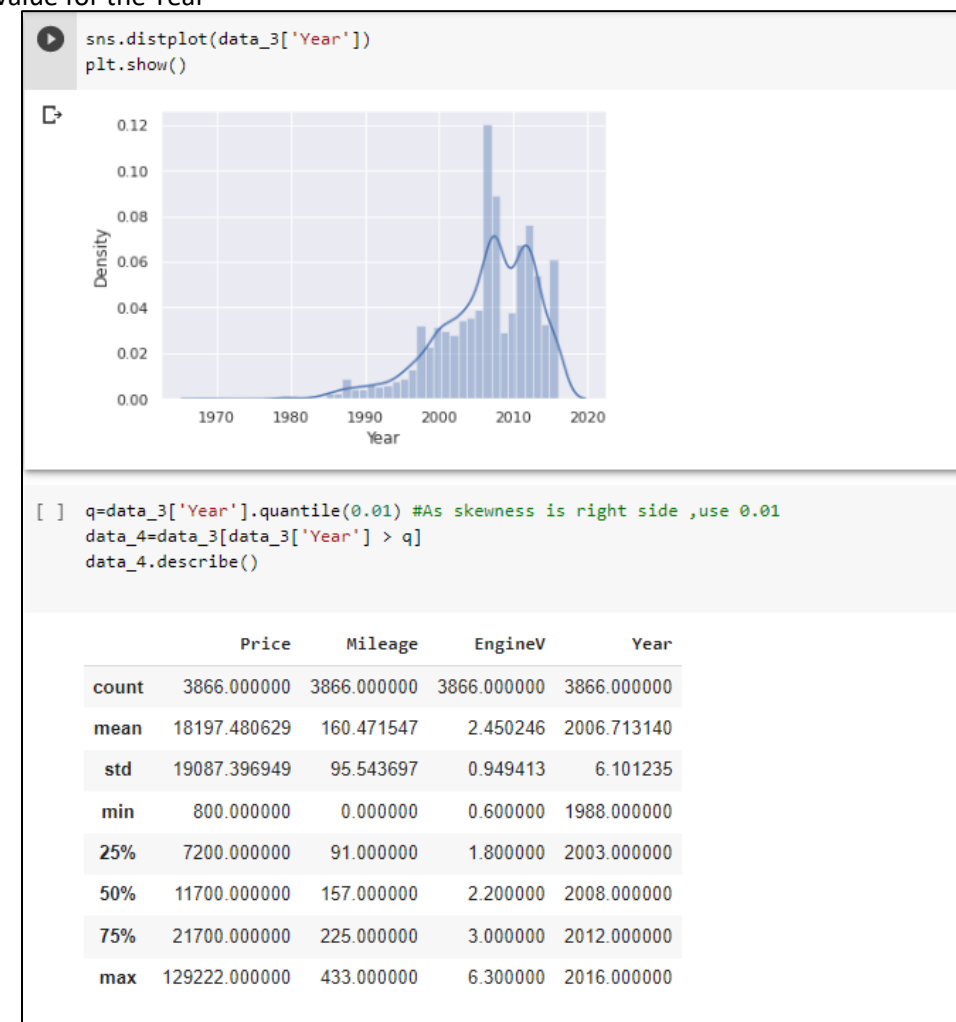
```
sns.distplot(data_no_mv['EngineV'])  
plt.show()
```



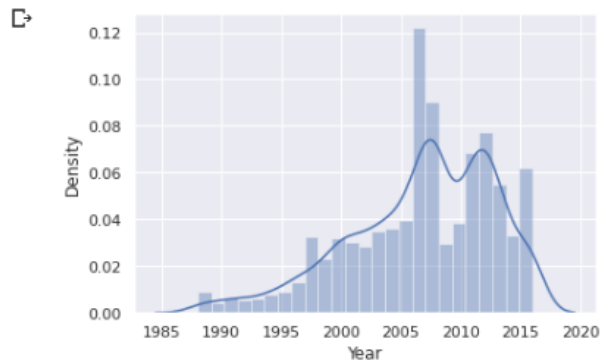
For dispelling that situation change the “EngineV” values by getting < 6.5 as below figure.



After that we moved to the “Year”. For that also used quantile method was used by keeping 99 percentile value for the Year



```
sns.distplot(data_4['Year'])
plt.show()
```



Using reset index ,Deleting current index & replacing numeric index by setting drop as 'True' as below figure.

```
[ ] data_cleaned = data_4.reset_index(drop = True)
data_cleaned
```

	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
...
3861	Volkswagen	11500.0	van	163	2.5	Diesel	yes	2008
3862	Toyota	17900.0	sedan	35	1.6	Petrol	yes	2014
3863	Mercedes-Benz	125000.0	sedan	9	3.0	Diesel	yes	2014
3864	BMW	6500.0	sedan	1	3.5	Petrol	yes	1999
3865	Volkswagen	13500.0	van	124	2.0	Diesel	yes	2013

3866 rows x 8 columns

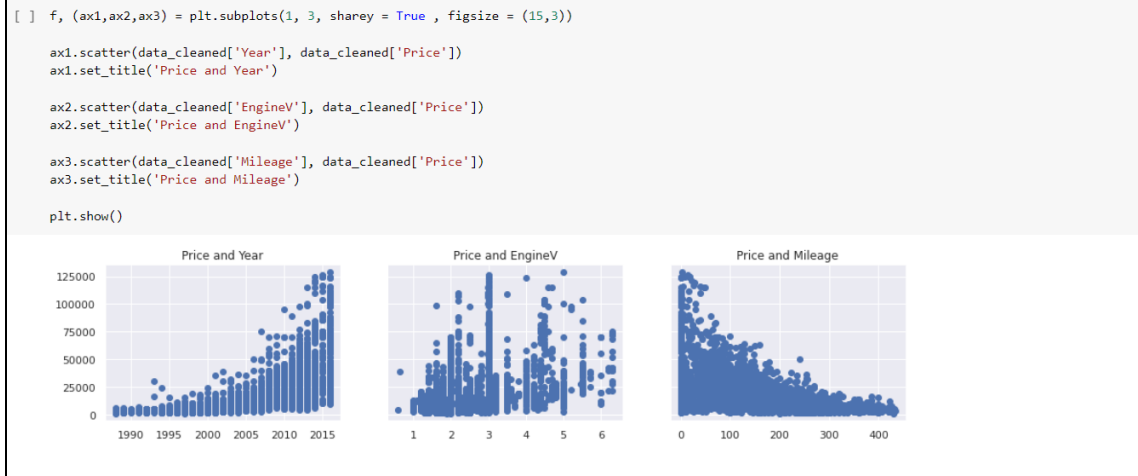
```
[ ] data_cleaned.describe(include='all')
```

	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	3866	3866.000000	3866	3866.000000	3866.000000	3866	3866	3866.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	848	NaN	1466	NaN	NaN	1807	3504	NaN
mean	NaN	18197.480629	NaN	160.471547	2.450246	NaN	NaN	2006.713140
std	NaN	19087.396949	NaN	95.543697	0.949413	NaN	NaN	6.101235
min	NaN	800.000000	NaN	0.000000	0.600000	NaN	NaN	1988.000000
25%	NaN	7200.000000	NaN	91.000000	1.800000	NaN	NaN	2003.000000
50%	NaN	11700.000000	NaN	157.000000	2.200000	NaN	NaN	2008.000000
75%	NaN	21700.000000	NaN	225.000000	3.000000	NaN	NaN	2012.000000
max	NaN	129222.000000	NaN	433.000000	6.300000	NaN	NaN	2016.000000

CHECKING THE OLS ASSUMPTIONS

For checking the relationship between independent & dependent variable, here scatter plot is used.

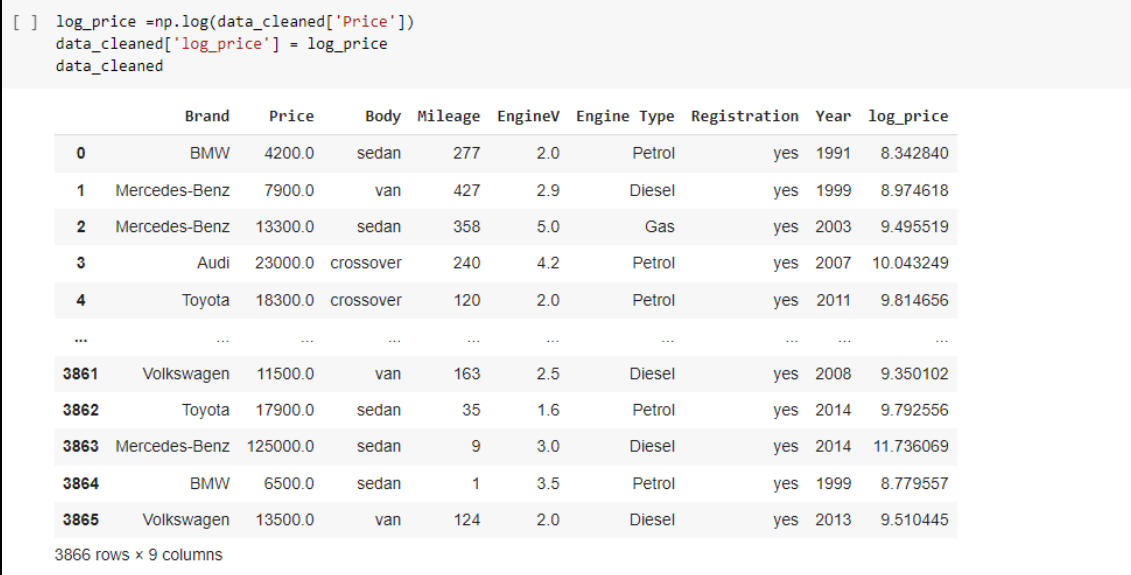
Plotting all independent variables against “Price” as it is dependent variable



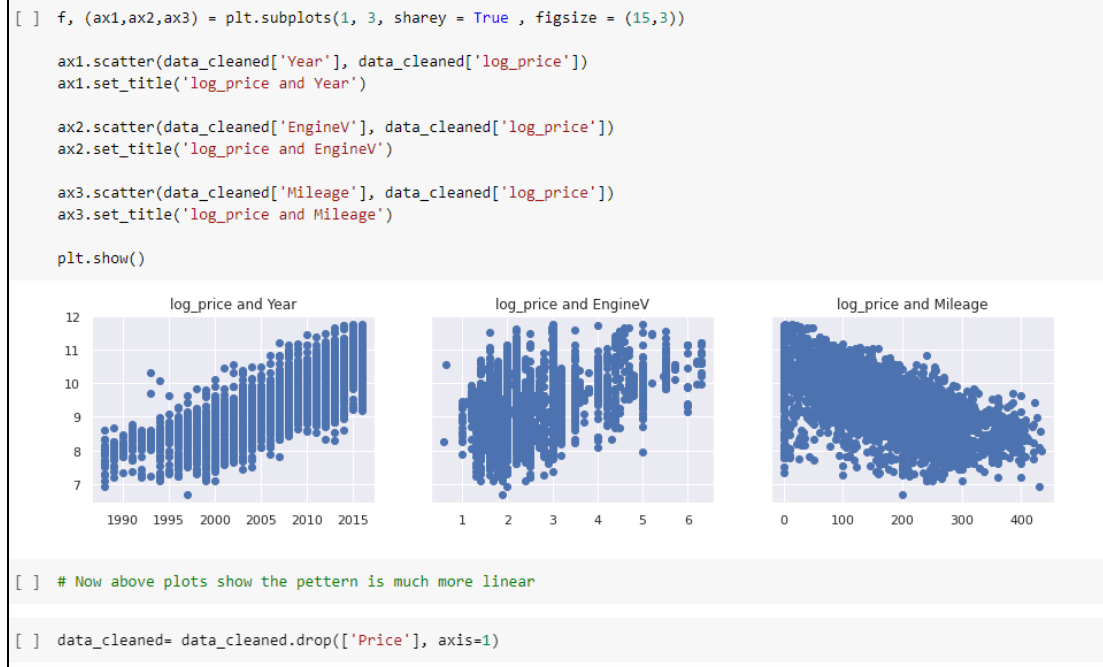
From above figure, Linear relationship can't be found for making the data we considered here to fit for the linear regression model & as a result of that data had to be transformed between dependent & independent variables. Then we had to used log transformation as next step.

LOG TRANSFORMATION

Calculating Natural logarithm of 'Price' in data set & it created new column as log_price & it shows from below figure.



After that plotted all independent variables against out dependent variable as "log_price"

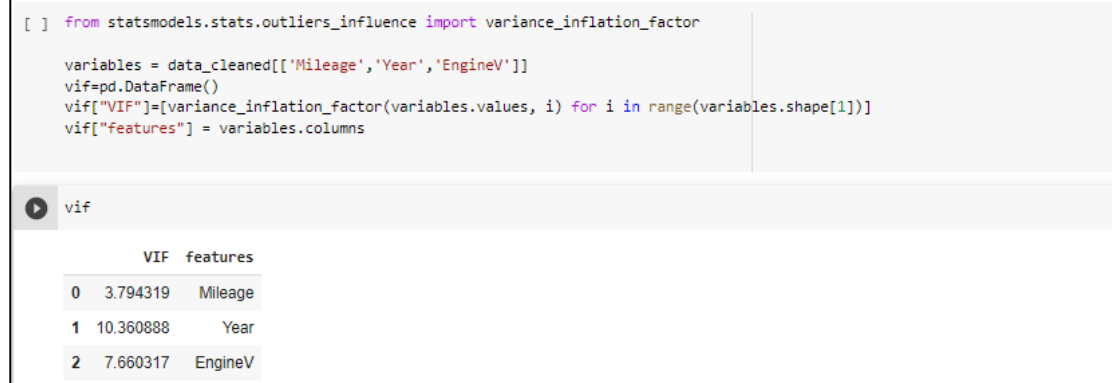


After making the 'log_price', above figure showed , the plot patterns have much more linear state.

As a result of that , no need to keep 'Price' column in data further, using drop code ,it dropped form data_cleaned.

REMOVING MULTICOLLINEARITY

Firstly we need to get an idea of Multicollinearity, it is a situation in which multiple independent variables in a multiple regression model have a strong correlation with each other. This can make it difficult to determine the unique impact of each variable on the dependent variable. For that, used "variance_inflation_factor", The Variance Inflation Factor (VIF) is a statistical measure that indicates the extent to which the variance of an independent variable is increased due to its correlation with other independent variables in a multiple regression model.



Based on the above figure, the VIF data frame shows that the variable "Year" has a high degree of multicollinearity. Therefore, it had to be removed from the model as below figure.

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

```
[ ] data_no_multicollinearity.head()
```

	Brand	Body	Mileage	EngineV	Engine Type	Registration	log_price
0	BMW	sedan	277	2.0	Petrol	yes	8.342840
1	Mercedes-Benz	van	427	2.9	Diesel	yes	8.974618
2	Mercedes-Benz	sedan	358	5.0	Gas	yes	9.495519
3	Audi	crossover	240	4.2	Petrol	yes	10.043249
4	Toyota	crossover	120	2.0	Petrol	yes	9.814656

After applying many steps, from here data is ready for doing the analysis.

CREATING DUMMIES

Creating dummies with all categorical data, using “get. dummies” encoded all into numerical quantities as below figure & it is a dummy encoded data frame.

```
[ ] 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
0	277	2.0	8.342840	1	0	0	0	0	0	0	0
1	427	2.9	8.974618	0	1	0	0	0	0	0	0
2	358	5.0	9.495519	0	1	0	0	0	0	0	0
3	240	4.2	10.043249	0	0	0	0	0	0	0	0
4	120	2.0	9.814656	0	0	0	0	1	0	0	0

Here “log_price” column need to be kept as first column as it is the main data we are using here. Then it was positioned as below figure.

```
[ ] data_with_dummies.columns
```

```
Index(['Mileage', 'EngineV', 'log_price', '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'],  
      dtype='object')
```

```
[ ] 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	Body_vagon
0	8.342840	277	2.0	1	0	0	0	0	0	0	0	1	0
1	8.974618	427	2.9	0	1	0	0	0	0	0	0	0	0
2	9.495519	358	5.0	0	1	0	0	0	0	0	0	1	0
3	10.043249	240	4.2	0	0	0	0	0	0	0	0	0	0
4	9.814656	120	2.0	0	0	0	0	1	0	0	0	0	0

DOWNLOADING PREPROCESSED DATA

From this step , finally complete the data after preprocessing & it is totally change with the data set used here to implement this project of Second Hand Cars Re-selling Price Analysis. The preprocessed data set was downloaded as a CSV file as below figure.

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

LINEAR REGRESSION MODEL

```
[ ] targets = data_preprocessed['log_price']
    inputs = data_preprocessed.drop(['log_price'], axis = 1)
```

In this stage here, applied Linear regression model as above graph

SCALING THE DATA

After applying Linear regression model, the data had to be scaled using sklearn library as below. Both centering and scaling are applied independently to each feature by calculating the appropriate statistics from the samples in the training set. The mean and standard deviation are then saved to be utilized on future data using the transform method & Standardizing a dataset is a frequent requirement for various machine learning models.

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

[ ] scalar=StandardScaler()
   scalar.fit(inputs)

   StandardScaler()

   StandardScaler()

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


TEST TRAIN SPLIT

To divide data here into training and testing set we use the `train test split()` method. We must first separate our data into features (X) and labels (Y) (y). The data frame is then divided into four sections: X train, X test, y train, and y test. The X train and y train sets are used to train and fit the model and the X test and y test sets are used to assess the model's ability to predict the proper output/labels. The train and test set sizes may be defined explicitly.

Here specially when considering 0.25 for `test_size`, The `train_test_split()` function uses the `test_size` parameter to determine the percentage or number of samples to include in the test set. If a float is provided, it should be between 0.0 and 1.0, representing the percentage of the dataset to use for testing. If an int is provided, it indicates the total number of test samples. If `test_size` is set to None, the complement of the train size will be used. If both `test_size` and `train_size` are None, `test_size` will be set to 0.25 by default.

```
[ ] from sklearn.model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split(inputs_scaled , targets , test_size = 0.25 , random_state = 365)
```

 x_train

```
array([[ -1.34406430e-03,  1.41612724e-01, -2.63613631e-01, ...,
        -1.62113726e-01, -7.50101044e-01,  3.21419511e-01],
       [ 1.50131983e+00,  2.56758184e-01, -2.63613631e-01, ...,
        -1.62113726e-01, -7.50101044e-01,  3.21419511e-01],
       [ 6.98465416e-01,  1.93160124e+00, -5.79639358e-01, ...,
        -1.62113726e-01, -7.50101044e-01, -3.11119881e+00],
       ...,
       [-1.01387251e+00,  6.44065640e-01,  3.21266937e+00, ...,
        -1.62113726e-01,  1.33315372e+00,  3.21419511e-01],
       [ 7.23554617e-01, -1.29247164e+00,  5.79121641e-01, ...,
        -1.62113726e-01, -7.50101044e-01,  3.21419511e-01],
       [ 1.55329031e+00,  8.53421022e-01, -2.63613631e-01, ...,
        -1.62113726e-01, -7.50101044e-01,  3.21419511e-01]])
```

[] y_train

```
1931    9.341369
3608    9.464983
2712    8.318742
1229    9.449357
1734   10.273325
...
428     11.074421
859     10.434116
801     9.928180
2740    10.609057
3666     8.824678
Name: log_price, Length: 2899, dtype: float64
```


CREATING THE REGRESSION

Now We have prepared our training and testing sets as above . There are many different types of models available in Scikit-Learn including LinearRegression which can be easily imported and trained. To fit the line to our data, we will utilize the .fit() method with our X_train and y_train data. After executing ,the result showed as it is Linear Regression & Using predict function, Getting prediction for 'x_train' data. as below figure. After executing the result by applying scatter plot it showed as below.

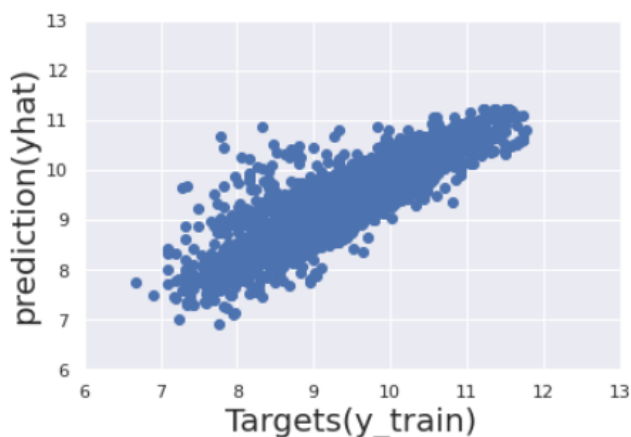
```
from sklearn.linear_model import LinearRegression  
reg=LinearRegression()  
reg.fit(x_train,y_train)
```

```
LinearRegression()
```

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

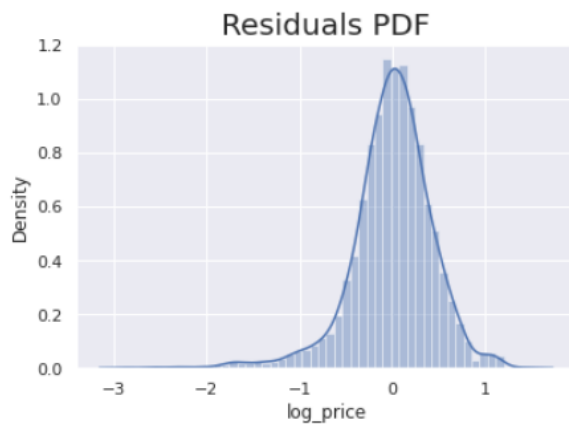
After executing the result by applying scatter plot it showed as below by confirming it was linear regression & the plot is optimized.

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



A residuals PDF is a plot that illustrates the distribution of the residuals in a model. The residuals are calculated as the difference between the actual values (y) and the predicted values (\hat{y}) of a model. This plot is used to evaluate the normality assumption of errors in a linear regression model. The distribution of residuals should resemble a normal distribution, if the model fits the data well. Here we found a well-fitting model. The residuals are approximately normally distributed with no patterns or outliers. The mean of the residuals is close to zero as below figure.

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



Conclusion

According to our analysis and the results of the linear regression model here our model well enough fits the data set and predicts the right values and it is not a particularly good model. More times we evaluate our model it becomes more accurate but we may also build a model that fits the data much better.

After starting the analysis with the data set downloaded from Kaggle.com, Data preprocessing as well as data cleaning & creating linear regression model were done gradually. In that data set ,it has 4500 rows & 09 columns. Specially examining was done for non-values after that that problem was rectified using proper method as missing values were less than 5% from all data. Next attention was for stats part. Then data distribution was taken in to consideration. After creating plots, identified outliers & gave a solution for that by applying proper method. As next main part, log normal transformation was applied as well as got the linear relation. After that multicollinearity was checked & removed for avoiding wrong prediction by the model. Then scaling the data & splitting the data were done. Why it was done is one part for testing & other part for training the model. After completing such procedures, Finally done the creating regression for the analysis.