SECONDHAND CARS RESELLING PRICE ANALYSIS

SARANGA KALALPITIYA
COADDS212P-010

TABLE OF CONTENTS

INTRODUCTION

Data analysis is the process of cleaning, analyzing, and interpreting data to inform decision-making and discover competitive advantages for your business. This is done using traditional statistical methods, AI, machine learning and more in order to gain understanding about the data selected.

Most commonly institutions/companies use data analysis to gain a better understanding on their internal products, process, markets, customers etc. Nowadays, data analysis extremely effective and act as a useful coach to understand the where the company was and what should be done to develop the for success and also to identify on failures in past.

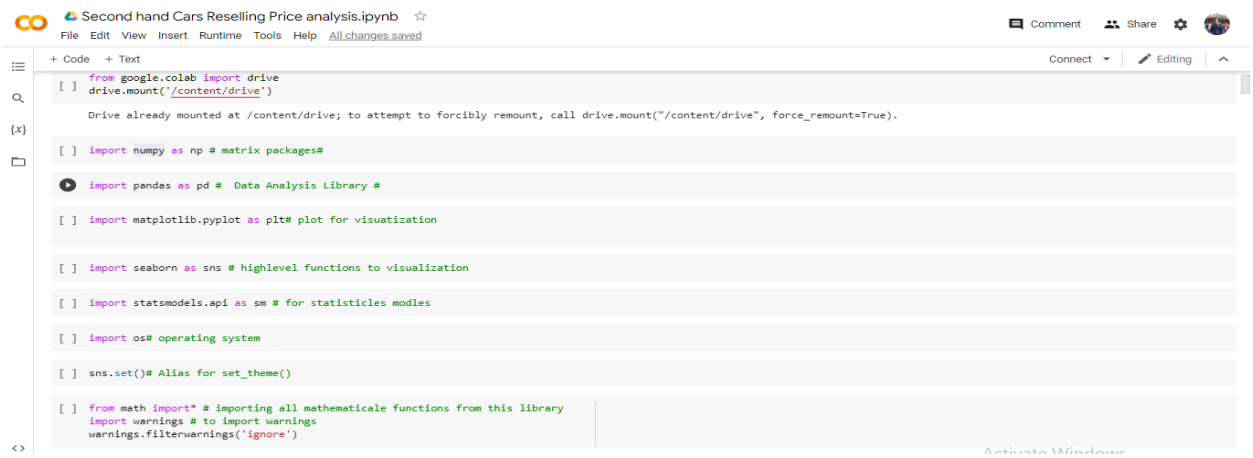
In this report, my effort is to elaborate the process of analyzing a sample data bank using data analyzing tools and techniques. For this I have selected data sample “Second hand Cars Reselling Price” which is abstracted from “kaggle”.

METHODOLOGY

To analyze the sample data analyzing tool and technique like python and regression analysis has been used while python libraries such as numpy, pandas, matplotlib, seaborn, statsmodels have been used in support of creating applications and models in a variety of fields. Google Colab version has been used to write and execute python codes.

IMPORTING LIBRARIES

First step is to installing the Libraries to the google Colab notebook as mentioned below.



IMPLEMENTATION

Loading the Raw data

Data can be imported to the notebook using following python code.

Loading the Raw Data

```
In [7]: data = pd.read_csv('/content/drive/MyDrive/Machine Learning/Regression/Second Hand Car Analysis/Second hand cars reselling price.
```

```
In [8]: data.head()
```

Out[8]:

	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

```
In [9]: data.tail()
```

Out[9]:

	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)

```
In [10]: data.shape
Out[10]: (4345, 9)
```

After inspecting data set it is understood that the data sample is quite a big with about 4500 rows. As we need to analyze the price of the second hand car, the regression analysis is used as the evaluating technique while price has been the independent variable. The depending variables are,

1. Brand:

   The BMW car will be expensive than the Toyota.

2. Mileage:

   The greater the mileage the expensive the car.

3. EngineV:

   The greater the engine volume the expensive the car. As sports cars are expensive than the family car.

4. Year :

   The older the car the cheap its price.

Preprocessing

Preprocessing

```
In [11]: data.describe(include = 'all')
```

Out[11]:

	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

From above figure descriptive statistics, it is identified that there are some missing values in the data set. To check them following code is used.

```
In [12]: data.isnull().sum()
```

```
Out[12]: Brand            0
         Price          172
         Body             0
         Mileage          0
         EngineV        150
         Engine Type      0
         Registration     0
         Year             0
         Model            0
         dtype: int64
```

Above figure shows that there are some missing values in the columns "Prics-172" and "EngineV-150". To remove unwanted data following code will be used,

Determining the variable of Interest

Determining the variables of interest

```
In [13]: data1 = data.drop(['Model'] , axis = 1)
```

In order drop the data from the table it is required two arguments, those are the row/column we want to drop and its axis. Axis = 0 means row and axis = 1 means column.

Here, we are considering dropping the column 'Model' as it is not significant for our analysis and can create huge variability in this analysis due to large number of unique values.

```
In [14]: data1.head()
```

Out[14]:

	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

```
In [15]: data.describe(include = 'all')
```

Out[15]:

	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

Here as we have two columns with missing values and as the number of missing values is less than 5% compared to total number of rows, we must drop the rows with the missing values in these columns.

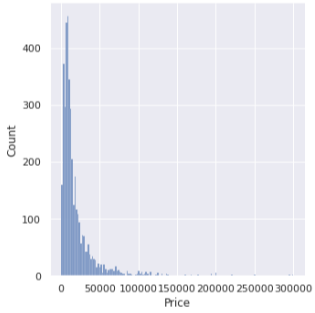
```
In [16]: data_no_mv = data1.dropna(axis = 0)

In [17]: data_no_mv.isnull().sum()

Out[17]: Brand           0
         Price           0
         Body            0
         Mileage         0
         EngineV         0
         Engine Type     0
         Registration    0
         Year            0
         dtype: int64
```

As per above figure now there is no any missing values.

Dealing with outliers

Dealing with outliers

```
In [18]: sns.displot(data_no_mv['Price'])
         plt.show()
```



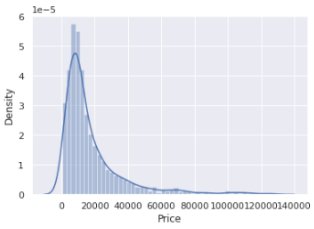
From above figure, we can say that the prices are distributed with a positive skewness. This would create problem in our regression. SO, we need to remove the outliers. To remove outliers, it is used quantile method keeping the 99 percentile values of the prices from our data.

```
In [19]: q = data_no_mv['Price'].quantile(0.99)
         data_1 = data_no_mv[data_no_mv['Price'] < q]
         data_1.describe()
```

Out[19]:

	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

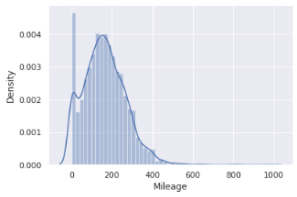
```
In [20]: sns.distplot(data_1['Price'])
         plt.show()
```



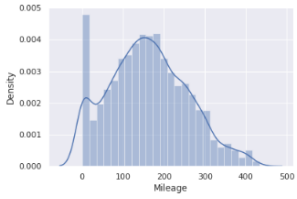
After removing outliers, we can still observe the skewness in our PDF for Prices, but this is something which can not be ignored now. So, need to deal with other variables too to progress the analysis.

Milage

```
In [21]: sns.distplot(data_no_mv['Mileage'])
         plt.show()
```

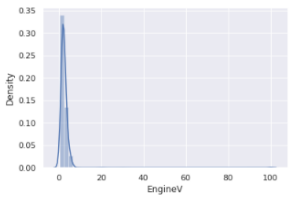



```
In [22]: q = data_no_mv['Mileage'].quantile(0.99)
         data_2 = data_1[data_1['Mileage'] < q]
         sns.distplot(data_2['Mileage'])
         plt.show()
```

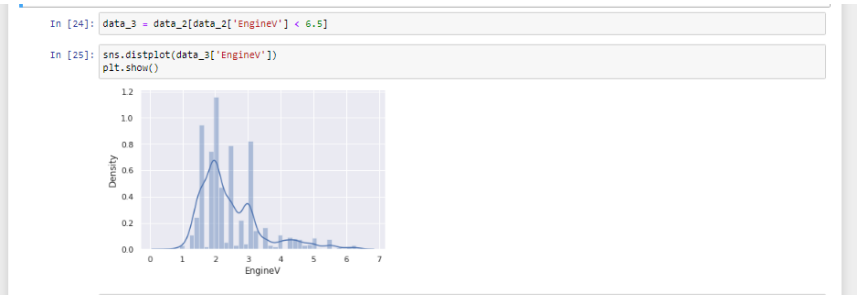


EngineV

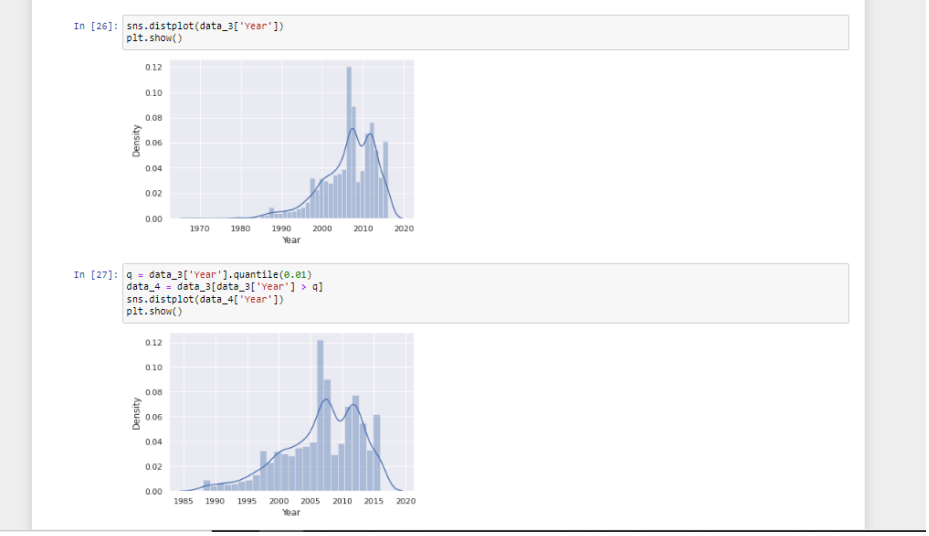
```
In [23]: sns.distplot(data_no_mv['EngineV'])
         plt.show()
```



As we can observed, there are huge outliers in the 'EngineV' column. By observing the data manually, it is understood that a column has some values equal to 99.99. This is due to fill up the null cells with these values. But, in general the value of the engine volume of a car cannot be more than 6.5 or less than 0.6. So, we must deal with this. Following figure show how to deal with that.



Year



```
In [28]: data_cleaned = data_4.reset_index(drop = True)
         data_cleaned
```

Out[28]:

	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 × 8 columns

```
In [29]: data_cleaned.describe(include = 'all')
```

Out[29]:

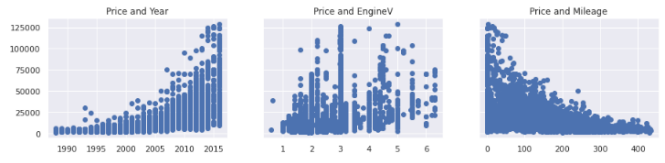
	Brand	Price	Body	Mileage	EngineV	Engine Type	Registration	Year
count	3866	3866.000000	3866	3866.000000	3866.000000	3866	3866	3866.000000
unique	7	NaN	6	NaN	NaN	4	2	NaN
top	Volkswagen	NaN	sedan	NaN	NaN	Diesel	yes	NaN
freq	848	NaN	1466	NaN	NaN	1807	3504	NaN
mean	NaN	18197.480629	NaN	160.471547	2.450246	NaN	NaN	2006.713140
std	NaN	19087.396649	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

Checking the OLS assumptions

Scatter polot will be used to plotting all independent variables against out dependent variable i.e "Price".

Checking the OLS assumptions

```
In [30]: #plotting all independent variables against out 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()
```



As we can identify in above scatter plots there is a linear relationship between the dependent and the independent variables of our data. To ensure the data best fit for the linear regression model, we should transform our data to get a linear relationship between the dependent and the independent variables.

Log Transformation

```
In [31]: log_price =np.log(data_cleaned['Price'])
         data_cleaned['log_price'] = log_price
         data_cleaned
```

Out[31]:

	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

Now the patterns are more linear.

```
In [32]: #plotting all independent variables against out dependent variable "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()
```

Removing Multicollinearity

To identify multicollinearity following method is followed,

Removing Multicollinearity

```
In [34]: 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
```

```
In [35]: vif
```

Out[35]:

	VIF	features
0	3.794319	Mileage
1	10.360888	Year
2	7.660317	EngineV

From the data frame it is understood that year has got lots of multicollinearity. So to remove that following code is used.

```
In [36]: data_no_multicollinearity = data_cleaned.drop(['Year'] , axis = 1)

In [37]: data_no_multicollinearity.head()
```

Out[37]:

	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

Creating dummies

Our data set is mix with numerical and categorical data so in order to turn them in to completely numerical, need to create dummies for all the categorical data. accordingly re-arrange the columns to make our data frame a neat one.

Creating dummies

```
In [38]: data_with_dummies = pd.get_dummies(data_no_multicollinearity , drop_first = True)

In [39]: data_with_dummies.head()
```

Out[39]:

	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

```
In [40]: data_with_dummies.columns
```

Out[40]:
```
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')
```

```
In [41]: 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']

In [42]: data_preprocessed = data_with_dummies[cols]
         data_preprocessed.head()
```

Out[42]:

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

Scaling the data

Scaling the data

```
In [45]: import sklearn as sk
         from sklearn.preprocessing import StandardScaler

In [46]: scalar = StandardScaler()
         scalar.fit(inputs)
```

Out[46]:
```
StandardScaler()
```

```
In [47]: inputs_scaled = scalar.transform(inputs)
```

Test Train Split

```
In [48]: 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)
```

```
In [49]: x_train
```

```
Out[49]: 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]])
```

```
In [50]: y_train
```

```
Out[50]: 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

Creating the Regression

```
In [51]: from sklearn.linear_model import LinearRegression
         reg = LinearRegression()
         reg.fit(x_train , y_train)
```

```
Out[51]: LinearRegression()
```

```
In [52]: yhat = reg.predict(x_train)
```

```
In [53]: 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()
```


```
In [54]: sns.distplot(y_train - yhat)
         plt.title("Residuals PDF" , size = 20)
         plt.show()
```


```
In [55]: reg.score(x_train , y_train)
```

```
Out[55]: 0.7451494225273487
```

Conclusion

After analyzing above data through linear regression model, it is understood that the model used is fitted with the data set and predicts fairly good correct values. However, overall, it's not an outstanding model and this is always a trail and test method. Hence, the more our model is tested, the more perfect it becomes.

References

1. https://www.kaggle.com/datasets
2. https://risk-engineering.org/static/PDF/slides-linear-regression.pdf
3. https://www.digitalocean.com/community/tutorials/how-to-import-modules-in-python-3