Problem Statement

Linear Regression

Import Libraries ¶

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
a=pd.read_csv("Ren.csv")
a
```

Out[2]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.241
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.634
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.495
...	...	...	...	...	...	...	...	
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Null
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

1549 rows × 11 columns

To display top 10 rows

In [3]:

```
c=a.head(15)
c
```

Out[3]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611559
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24188
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.41
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63460
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49565
5	6.0	pop	74.0	3623.0	70225.0	1.0	45.000702	7.68227
6	7.0	lounge	51.0	731.0	11600.0	1.0	44.907242	8.611559
7	8.0	lounge	51.0	1521.0	49076.0	1.0	41.903221	12.49565
8	9.0	sport	73.0	4049.0	76000.0	1.0	45.548000	11.54946
9	10.0	sport	51.0	3653.0	89000.0	1.0	45.438301	10.99170
10	11.0	pop	51.0	790.0	43286.0	1.0	40.871429	14.43896
11	12.0	lounge	51.0	366.0	17500.0	1.0	45.069679	7.704919
12	13.0	lounge	51.0	456.0	18450.0	1.0	45.426571	11.78812
13	14.0	pop	51.0	3835.0	120000.0	1.0	40.531590	17.43615
14	15.0	lounge	51.0	1035.0	40500.0	1.0	40.911362	14.21119

To find Missing values

In [4]:

```
c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               15 non-null     float64
 1   model            15 non-null     object 
 2   engine_power     15 non-null     float64
 3   age_in_days      15 non-null     float64
 4   km               15 non-null     float64
 5   previous_owners  15 non-null     float64
 6   lat              15 non-null     float64
 7   lon              15 non-null     object 
 8   price            15 non-null     object 
 9   Unnamed: 9       0 non-null      float64
 10  Unnamed: 10      0 non-null      object 
dtypes: float64(7), object(4)
memory usage: 1.4+ KB
```

To display summary of statistics

In [5]:

```
a.describe()
```

Out[5]:

	ID	engine_power	age_in_days	km	previous_owners	la
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.00000
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.54136
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.13351
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.85583
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.80299
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.39409
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.46796
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.79561

To display column heading

In [6]:

```
a.columns
```

Out[6]:

```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owner
s',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

Pairplot

In [7]:

```
s=a.dropna(axis=1)
s
```

Out[7]:

	lon	price
0	8.611559868	8900
1	12.24188995	8800
2	11.41784	4200
3	17.63460922	6000
4	12.49565029	5700
...	...	...
1544	length	5
1545	concat	lonprice
1546	Null values	NO
1547	find	1
1548	search	1

1549 rows × 2 columns

In [8]:

```
s.columns
```

Out[8]:

```
Index(['lon', 'price'], dtype='object')
```

In [9]:

```
sns.pairplot(a)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x1a71544f8b0>
```

Distribution Plot

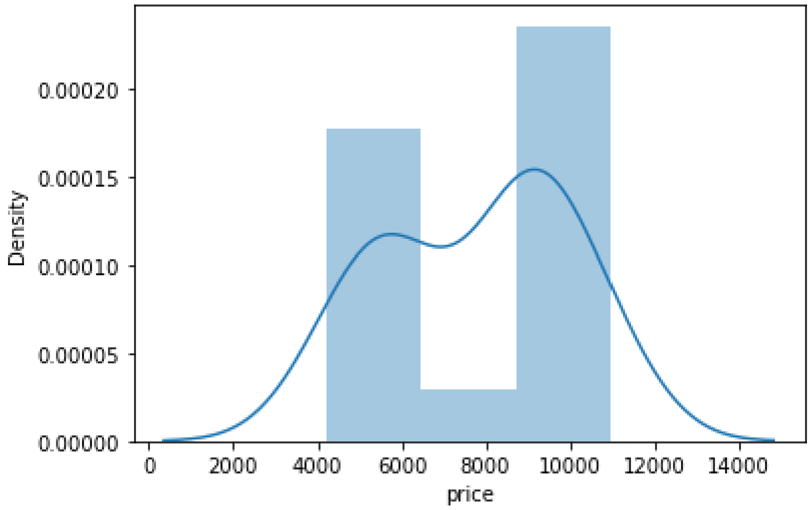
In [10]:

```
sns.distplot(c['price'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[10]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



Correlation

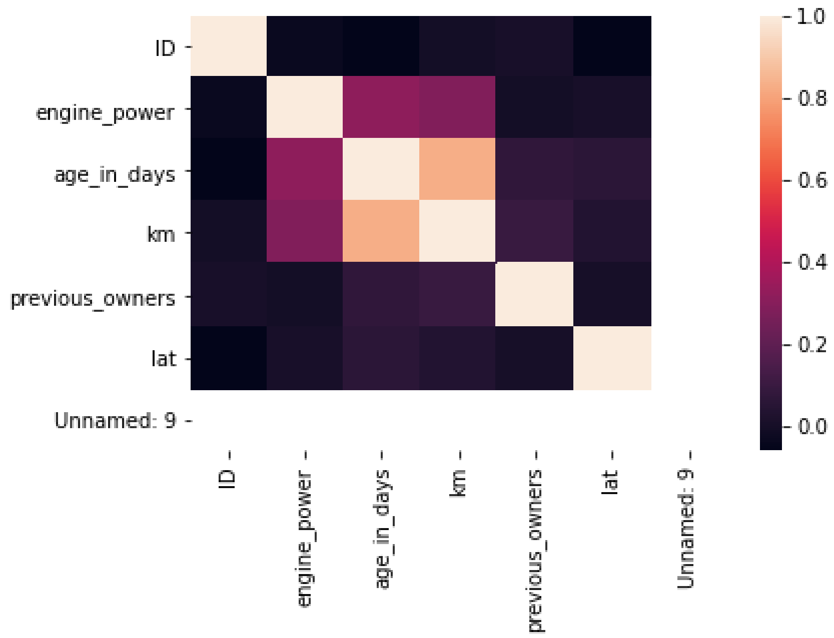
Train the model - Model Building

In [11]:

```
b=a[['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10']]
sns.heatmap(b.corr())
```

Out[11]:

```
<AxesSubplot:>
```



In [12]:

```
g=c[['price']]
h=c['price']
```

To split dataset into training end test

In [13]:

```
from sklearn.model_selection import train_test_split
g_train,g_test,h_train,h_test=train_test_split(g,h,test_size=0.6)
```

To run the model

In [14]:

```
from sklearn.linear_model import LinearRegression
```

In [15]:

```
lr=LinearRegression()
lr.fit(g_train,h_train)
```

Out[15]:

```
LinearRegression()
```

In [16]:

```
print(lr.intercept_)
```

9.094947017729282e-13

Coeffecient

In [17]:

```
coeff=pd.DataFrame(lr.coef_,g.columns,columns=['Co-effecient'])
coeff
```

Out[17]:

	Co-effecient
price	1.0

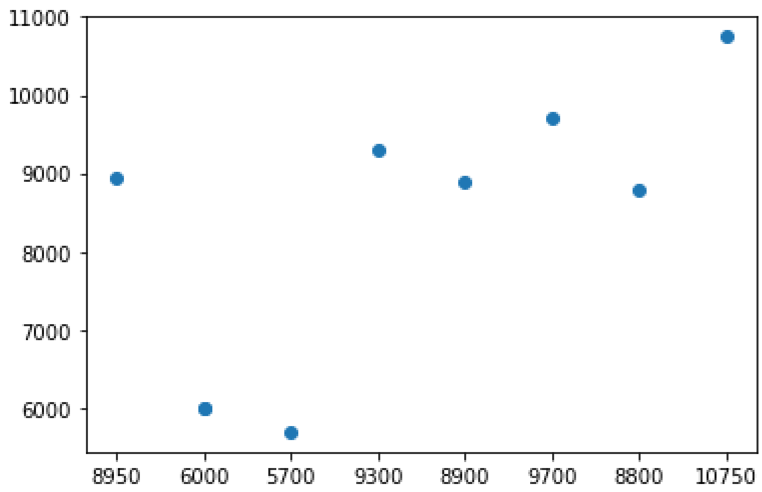
Best Fit line

In [18]:

```
prediction=lr.predict(g_test)
plt.scatter(h_test,prediction)
```

Out[18]:

<matplotlib.collections.PathCollection at 0x1a719257df0>



To find score

In [19]:

```
print(lr.score(g_test,h_test))
```

1.0

Import Lasso and ridge

In [20]:

```
from sklearn.linear_model import Ridge,Lasso
```

Ridge

In [21]:

```
ri=Ridge(alpha=5)
ri.fit(g_train,h_train)
```

Out[21]:

```
Ridge(alpha=5)
```

In [22]:

```
ri.score(g_test,h_test)
```

Out[22]:

```
0.9999999999999729
```

In [23]:

```
ri.score(g_train,h_train)
```

Out[23]:

```
0.9999999999999808
```

Lasso

In [24]:

```
l=Lasso(alpha=6)
l.fit(g_train,h_train)
```

Out[24]:

```
Lasso(alpha=6)
```

In [25]:

```
l.score(g_test,h_test)
```

Out[25]:

```
0.9999999999985933
```

In [27]:

```
ri.score(g_train,h_train)
```

Out[27]:

```
0.9999999999999808
```

ElasticNet

In [28]:

```
from sklearn.linear_model import ElasticNet
e=ElasticNet()
e.fit(g_train,h_train)
```

Out[28]:

```
ElasticNet()
```

Coeffecient,intercept

In [29]:

```
print(e.coef_)
```

```
[0.99999983]
```

In [30]:

```
print(e.intercept_)
```

```
0.0011826273603219306
```

Prediction

In [31]:

```
d=e.predict(g_test)
d
```

Out[31]:

```
array([ 8949.99969464,  6000.0001851 ,  5700.00023497,  6000.0001851 ,
        9299.99963646,  8899.99970296,  9699.99956995,  8799.99971958,
       10749.99939539])
```

In [32]:

```
print(e.score(g_test,h_test))
```

```
0.9999999999999609
```

Evaluation

In [33]:

```
from sklearn import metrics
print("Mean Absolute error:",metrics.mean_absolute_error(h_test,d))
```

```
Mean Absolute error: 0.00032068745693120744
```

In [34]:

```
print("Mean Squared error:",metrics.mean_squared_error(h_test,d))
```

```
Mean Squared error: 1.1850072779279364e-07
```

In [35]:

```
print("Mean Squared error:",np.sqrt(metrics.mean_squared_error(h_test,d)))
```

```
Mean Squared error: 0.0003442393466656501
```

In [ ]: