In [1]:

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

In [2]:

```
#import dataset
df=pd.read_csv(r"E:\154\fiat500_VehicleSelection_Dataset - fiat500_VehicleSele
df
```

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	8.61155
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.2418
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.4
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.6346
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.4956
...	...	...	...	...	...	...	...	
1495	1496.0	pop	62.0	3347.0	80000.0	3.0	44.283878	11.8881
1496	1497.0	pop	51.0	1461.0	91055.0	3.0	44.508839	11.4690
1497	1498.0	lounge	51.0	397.0	15840.0	3.0	38.122070	13.3611
1498	1499.0	sport	51.0	1400.0	60000.0	1.0	45.802021	9.18778
1499	1500.0	pop	51.0	1066.0	53100.0	1.0	38.122070	13.3611

1500 rows × 9 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1500 non-null   float64
 1   model            1500 non-null   object 
 2   engine_power     1500 non-null   float64
 3   age_in_days      1500 non-null   float64
 4   km               1500 non-null   float64
 5   previous_owners  1500 non-null   float64
 6   lat              1500 non-null   float64
 7   lon              1500 non-null   object 
 8   price            1500 non-null   object 
dtypes: float64(6), object(3)
memory usage: 105.6+ KB
```

In [4]:

```
#to display top 5 rows
df.head()
```

Out[4]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611559868
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24188995
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.41784
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63460922
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49565029

Data cleaning and Pre-Processing

In [5]:
```
#To find null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1500 non-null   float64
 1   model            1500 non-null   object 
 2   engine_power     1500 non-null   float64
 3   age_in_days      1500 non-null   float64
 4   km               1500 non-null   float64
 5   previous_owners  1500 non-null   float64
 6   lat              1500 non-null   float64
 7   lon              1500 non-null   object 
 8   price            1500 non-null   object 
dtypes: float64(6), object(3)
memory usage: 105.6+ KB
```

In [6]:
```
# To display summary of statistics
df.describe()
```

Out[6]:

	ID	engine_power	age_in_days	km	previous_owners	lat
count	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000
mean	750.500000	51.875333	1641.629333	53074.900000	1.126667	43.545904
std	433.157015	3.911606	1288.091104	39955.013731	0.421197	2.112907
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.855839
25%	375.750000	51.000000	670.000000	20000.000000	1.000000	41.802990
50%	750.500000	51.000000	1035.000000	38720.000000	1.000000	44.360376
75%	1125.250000	51.000000	2616.000000	78170.250000	1.000000	45.467960
max	1500.000000	77.000000	4658.000000	235000.000000	4.000000	46.795612

In [7]:
```
#To Display column heading
df.columns
```

Out[7]:
```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price'],
      dtype='object')
```

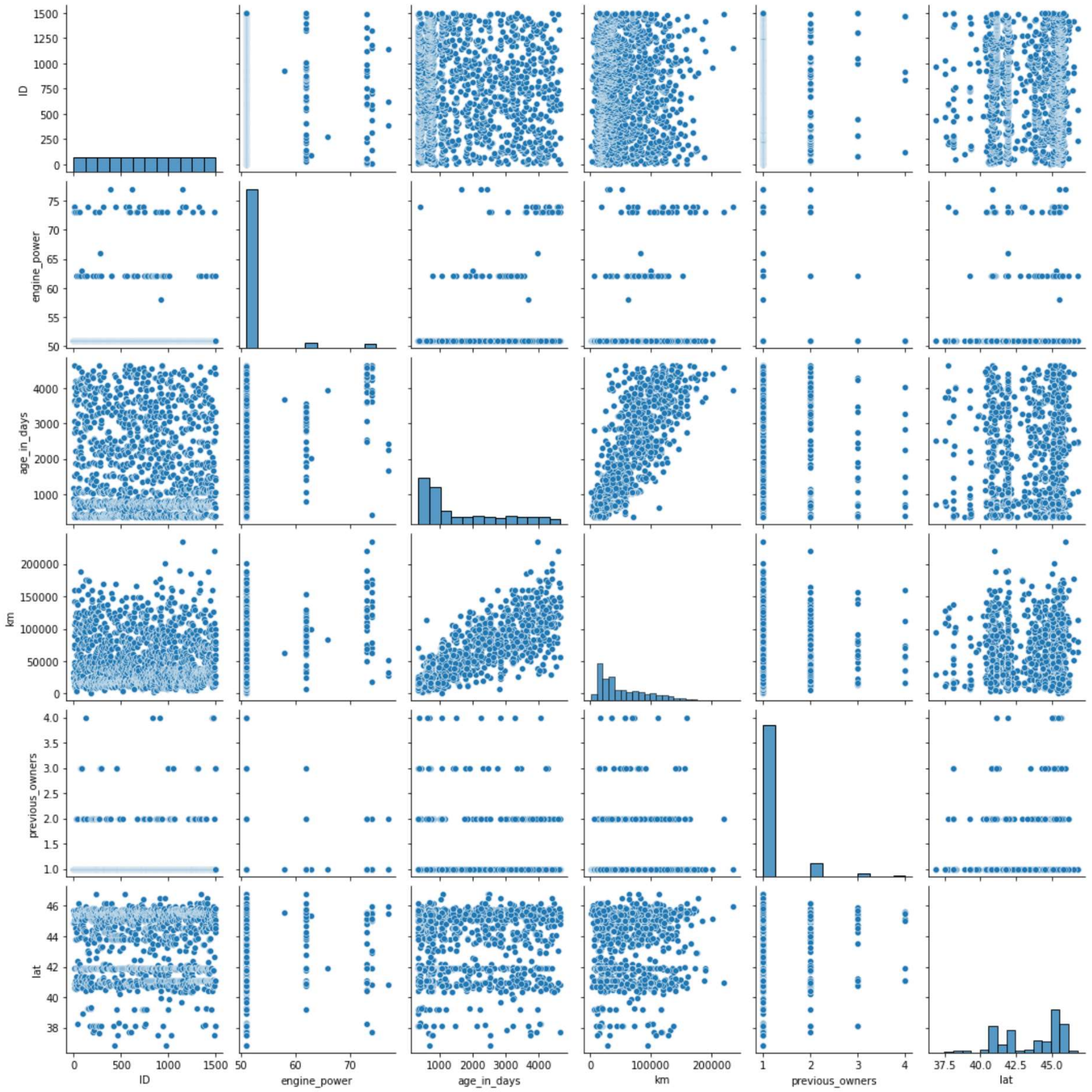
EDA and VISUALIZATION

In [8]:

```
sns.pairplot(df)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x22f718d1b20>
```



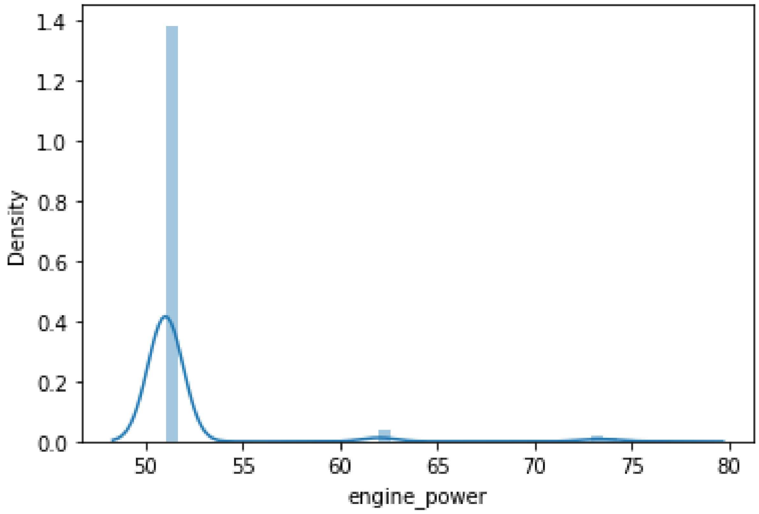
In [9]:

```
sns.distplot(df['engine_power'])
```

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

Out[9]:

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



In [10]:

```
df1=df[['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price']]
```

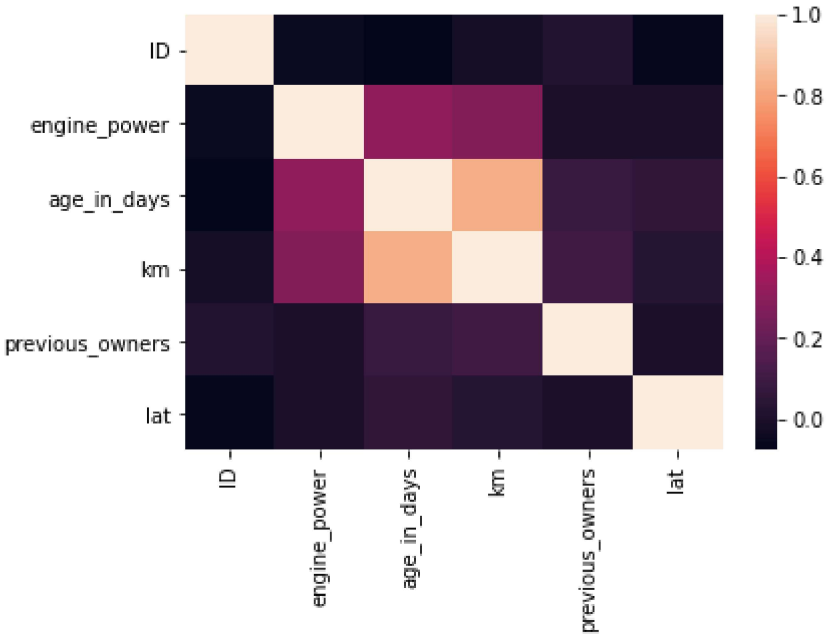
Plot Using Heat Map

In [11]:

```
sns.heatmap(df1.corr())
```

Out[11]:

```
<AxesSubplot:>
```



To Train The Model-Model Building

we are going to train Linera Regression Model;We need to split out data into two variables x and y where x is independent variable(input) and y is dependent on x(output) we could ignore address column as it required for our model

In [12]:

```
x=df1[['ID',  'previous_owners',
       'lat']]
y=df1['engine_power']
```

To Split my dataset into training and test data

In [13]:

```

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [14]:

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

Out[14]:

```
LinearRegression()
```

In [15]:
```
lr.intercept_
```

Out[15]:
```
52.907555247329896
```

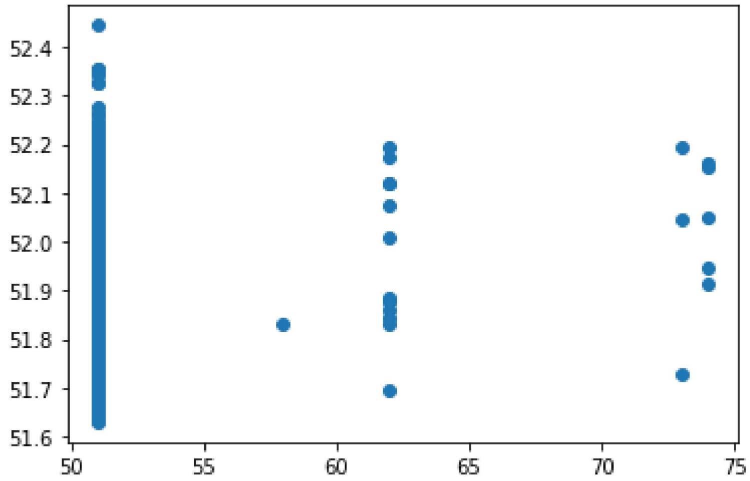
In [16]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[16]:

	Co-efficient
ID	-0.000345
previous_owners	0.117449
lat	-0.019168

In [17]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x22f73fce8e0>
```



Accuracy

In [18]:
```
lr.score(x_test,y_test)
```

Out[18]:
```
-0.0006739861794684554
```

In [19]:
```
lr.score(x_train,y_train)
```

Out[19]:
```
0.0014686639744330154
```

In [20]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [21]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[21]:
```
Ridge(alpha=10)
```

```
In [22]: rr.score(x_test,y_test)

Out[22]: -0.00060309972910O4425
```

```
In [23]: la =Lasso(alpha=10)
         la.fit(x_train,y_train)

Out[23]: Lasso(alpha=10)
```

```
In [24]: la.score(x_test,y_test)

Out[24]: 0.0005139188675424844
```