Problem Statement:

A real estate agent want to help to predict the house price for regions in USA.He gave us the dataset to work on to use Linear Regression modelCreate a Model that helps him to estimate of what the house would sell for

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("fiat.csv",low_memory=False)[0:1500]
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 × 11 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 11 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 
 9   Unnamed: 9       0 non-null      float64
 10  Unnamed: 10      0 non-null      object 
dtypes: float64(7), object(4)
memory usage: 129.0+ KB
```

In [4]:
```
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]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 11 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 
 9   Unnamed: 9       0 non-null      float64
 10  Unnamed: 10      0 non-null      object 
dtypes: float64(7), object(4)
memory usage: 129.0+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	ID	engine_power	age_in_days	km	previous_owners	lat	U
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]:
```
df.dropna(axis='columns')
```

Out[7]:

	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 [8]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[8]:
```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price'],
      dtype='object')
```

In [9]:
```
df.columns
```

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

EDA and VISUALIZATION

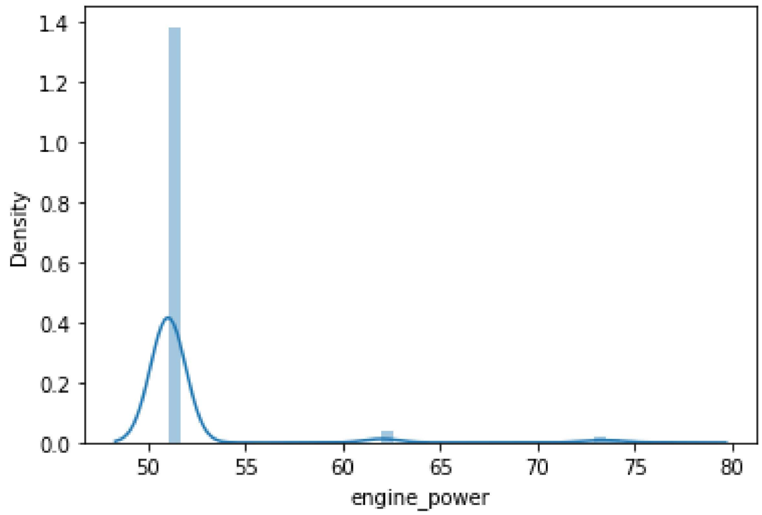
In [10]:
```
sns.pairplot(df)
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x1822c816ca0>

In [11]:
```
sns.distplot(df['engine_power'])
```

```
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 function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[11]:
```
<AxesSubplot:xlabel='engine_power', ylabel='Density'>
```

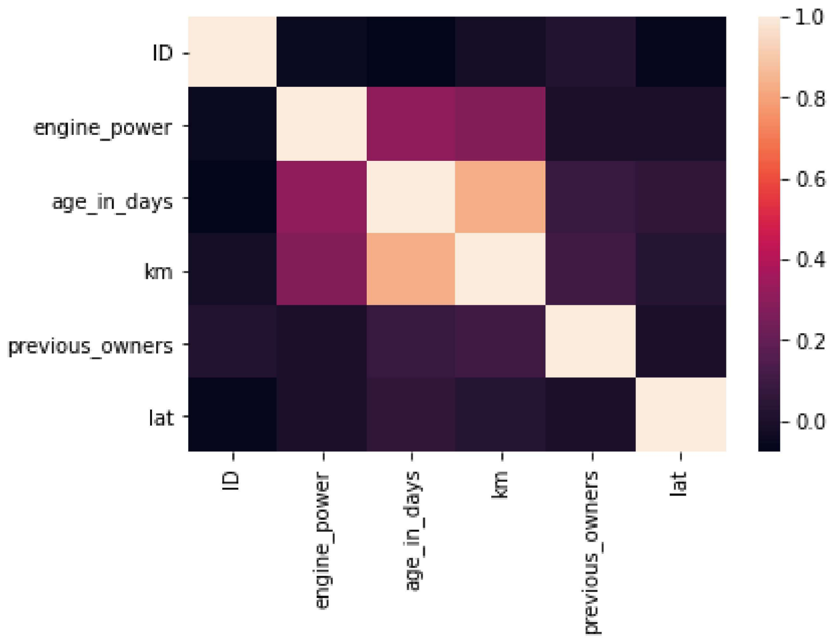


In [12]:
```
df1=df[['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
        'lat', 'lon', 'price']]
```

Plot Using Heat Map

In [13]:
```
sns.heatmap(df1.corr())
```

Out[13]: <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 [14]:
```
x=df1[['ID', 'previous_owners','lat']]
y=df1['engine_power']
```

To Split my dataset into training and test data

In [15]:
```
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 [16]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Out[16]: LinearRegression()

In [17]:
```
lr.intercept_
```

Out[17]: 52.603888253753354

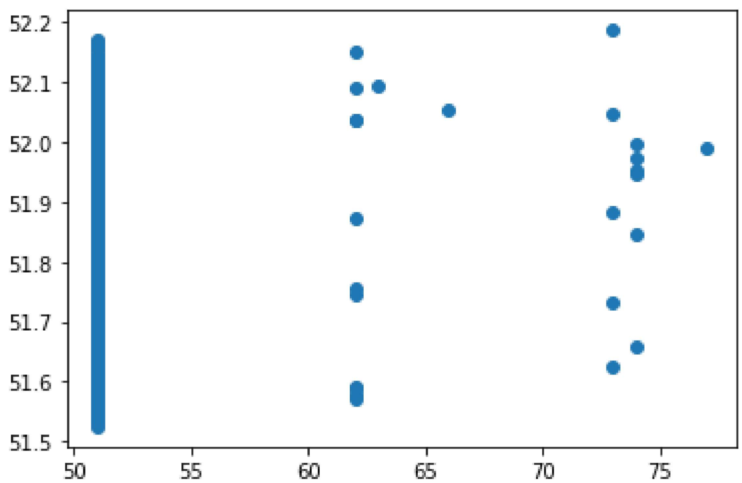
In [18]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[18]:

	Co-efficient
ID	-0.000402
previous_owners	0.020723
lat	-0.010976

In [19]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[19]: <matplotlib.collections.PathCollection at 0x18230e83880>



In [20]:
```
lr.score(x_test,y_test)
```

Out[20]: 0.0026850845075601093

In [21]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [22]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
rr.score(x_train,y_train)
```

Out[22]: 0.0020595119628535885

In [23]:
```
rr.score(x_test,y_test)
```

Out[23]: 0.0026884180061897966

In [24]:
```
la = Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[24]:
```
Lasso(alpha=10)
```

In [25]:
```
la.score(x_test,y_test)
```

Out[25]:
```
0.0026369917869903947
```

In [26]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[26]:
```
ElasticNet()
```

In [27]:
```
print(en.coef_)
```

```
[-0.00039555  0.          -0.        ]
```

In [28]:
```
print(en.intercept_)
```

```
52.14543649034472
```

In [29]:
```
print(en.predict(x_test))
```

```
[51.81871148 51.78588076 52.04931763 52.05722865 52.12407674 51.61421169
 52.10390365 51.65930449 51.6640511  51.69648627 51.5964119  52.02281572
 51.7815297  51.92155471 52.11933013 51.71547271 51.67077546 51.92036805
 51.80091169 51.99354496 52.00343373 51.8974261  51.89307504 51.83690682
 51.73366805 52.12091234 51.74474347 51.86222208 52.03033119 52.11379242
 52.08333501 51.83453352 51.60155406 51.63952694 52.10746361 51.9409367
 51.74276572 51.91127038 51.73881021 51.80288945 51.61065173 52.13436107
 51.90059051 52.01450916 51.72694368 52.12170344 51.71428606 51.5730744
 51.92669687 51.76293881 52.11141912 51.72338373 52.07028183 52.0117403
 51.8274136  51.75661    51.80724051 51.67789538 52.01292695 51.82345809
 52.14504094 51.68224644 51.7807386  51.56714114 51.93737674 52.08610386
 52.0785884  52.03942886 51.60115851 51.95240767 51.65376678 51.9860295
 51.74553457 51.92709242 51.79893394 52.14227208 52.09559708 51.77005873
 52.02637568 51.78944072 51.88872398 51.99908267 52.1035081  51.69609072
 51.65218457 52.0793795  51.59126974 51.56990999 52.06751297 51.94133225
 51.73841466 51.99314941 51.67987313 52.05445979 51.65178902 51.94449666
 52.04615323 51.73287695 51.62251826 51.92986127 51.95834094 51.88714178
 51.68264199 51.94766106 51.70123288 51.97811848 51.55606571 51.85272886
 51.69134411 51.95280322 51.89782165 51.84086233 51.98049178 51.75344559
 52.0133225  51.83769792 51.58652313 51.99038055 51.72219707 52.009367
 51.61381614 51.84244453 51.95478098 51.65653563 51.56318563 51.72654813
 51.62924262 52.08570831 51.6399225  52.05485534 51.79418733 52.0342867
 51.87369305 52.00778479 51.95952759 51.74039241 51.69767292 51.56635004
 52.1042992  51.86934199 51.62172716 52.0576242  52.00699369 51.58256762
 51.62330936 51.99868712 51.81159157 51.71507716 52.12486785 51.55369241
 51.97970068 51.6648422  51.88318627 52.1284278  51.6189583  51.79260513
 51.82148034 51.84007123 51.95873649 51.85787102 52.11497907 52.08214835
 51.56041677 51.61579389 52.05960195 51.81910703 51.91285259 52.02518903
 52.01213585 51.97574517 52.14108543 52.11893458 51.87883521 51.62014495
 52.14148098 51.96822971 51.96387865 51.77876085 51.6889708  52.04140662
 51.616585   51.55329686 51.99394051 51.78825407 52.06118416 51.69292631
 51.87646191 51.57505216 51.67433542 51.92630132 51.85945322 51.60274072
 51.76452102 51.94805661 51.60471847 51.71784601 51.89900831 51.60946508
 51.79062737 51.76649877 51.9425189  52.03310005 51.57821656 51.81792038
 51.97099856 52.056042   52.08649941 51.58810533 51.81673373 51.72892144
 51.63042928 51.62607822 52.08729051 51.83295131 51.97930513 51.83216021
 52.02795789 52.00659814 51.65139347 51.80605386 51.9168081  51.7574011
 52.02083797 51.76135661 51.90731487 51.57623881 51.78271635 51.61777165
 51.6430869  51.85668436 51.69688182 51.96071424 51.58968754 51.9191814
 51.62845152 51.93539899 51.82227144 51.86855089 51.76689432 52.12724115
 52.11181467 51.8515422  51.9674386  52.09203712 51.68303754 52.1019259
 51.85233331 51.70202398 51.59245639 51.86617759 51.81475597 52.11260577
 51.70637504 51.89465725 52.03230895 51.71270385 51.65455788 51.97218522
 51.55962567 51.83097356 51.66602885 51.74593013 52.11616573 51.58177652
 52.01609136 51.67947758 52.06830407 52.0358689  51.58612758 52.06395301
 51.79853839 52.0825439  51.75186339 51.65851339 52.10706806 52.10232145
 51.79774729 52.05999751 51.75305004 51.5979941  52.01371806 51.63161593
 51.64743796 51.79379178 51.70162843 51.64190025 51.86182653 51.83255576
 51.80209835 51.74316127 51.89979941 52.03389115 51.59285194 52.11537463
 51.709935   52.02677123 51.9417278  51.87764856 52.12921891 51.99275386
 51.73010809 51.78627631 51.96704305 52.10785916 51.66326    51.81989814
 51.68145534 51.64269135 51.67156656 51.88397737 51.66840216 52.07067738
 51.74157907 52.04298882 52.13633882 51.58217207 51.8950528  51.74118351
 52.03982441 51.63359368 52.03824221 51.56437228 51.60867398 51.93144348
 51.86657314 51.88595513 51.7783653  51.67196211 51.56674559 51.82108479
 51.90573267 52.07423734 51.77243203 51.79576953 51.72061487 51.67037991
 51.66761106 51.74869898 51.99591827 51.84204898 51.57782101 51.55448351
```

```
 51.79616508 51.82543585 51.91799475 51.8258314  52.0592064  51.93183903
 51.93975004 51.74988563 52.03903331 51.70044178 51.72852589 52.11972568
 51.70874834 52.01530026 52.10073924 52.11339687 51.64862461 51.77717864
 51.93421233 51.66207334 51.68501529 52.06276636 51.83532462 52.00145598
 51.95794538 51.67393987 52.02954009 51.64981127 51.87290195 51.70677059
 51.65890894 51.89386615 51.68422419 52.05089984 52.00224708 52.08847717
 51.75977441 51.77480534 51.79656063 52.14266763 51.9666475  51.98444729
 52.11656128 51.56397673 51.90810598 52.05010873 51.5714922  51.74197462
 51.97297632 51.84996    52.12803225 51.95557208 51.61856275 51.62686932
 51.82780915 52.12012124 52.00897144 51.64466911 51.76412546 52.04813098
 51.97337187 51.94647441 51.66444665 51.67473097 52.01925577 51.88516403
 51.63438478 52.13356996 51.84719114 51.5948297  51.55290131 51.58058987
 51.62647377 51.76175216 51.75265449 51.67829093 51.71389051 51.85905767
 52.00738924 52.00976255 51.70756169 51.99987378 52.04536212 51.90415047
 51.72457038 51.6672155  51.91957695 51.79339623 52.03666    52.0350778
 51.77440979 52.09876149 51.65099792 51.80961381 51.90494157 52.13910768
 51.64941572 52.00422483 51.74949008 52.06869962 51.74434792 51.64348245
 51.76887207 51.97416297 52.07344623 51.86143098 52.08373056 51.94370555]
```

In [30]:
```python
print(en.score(x_test,y_test))
```

```
0.0028688722245815423
```

In [31]:
```python
# Evaluation Metrics
from sklearn import metrics
```

In [32]:
```python
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.6860751719200184
```

In [33]:
```python
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 16.786489456805334
```

In [34]:
```python
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,pred
```

```
Root Mean Squared Error: 4.097131857385766
```