# 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 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 |

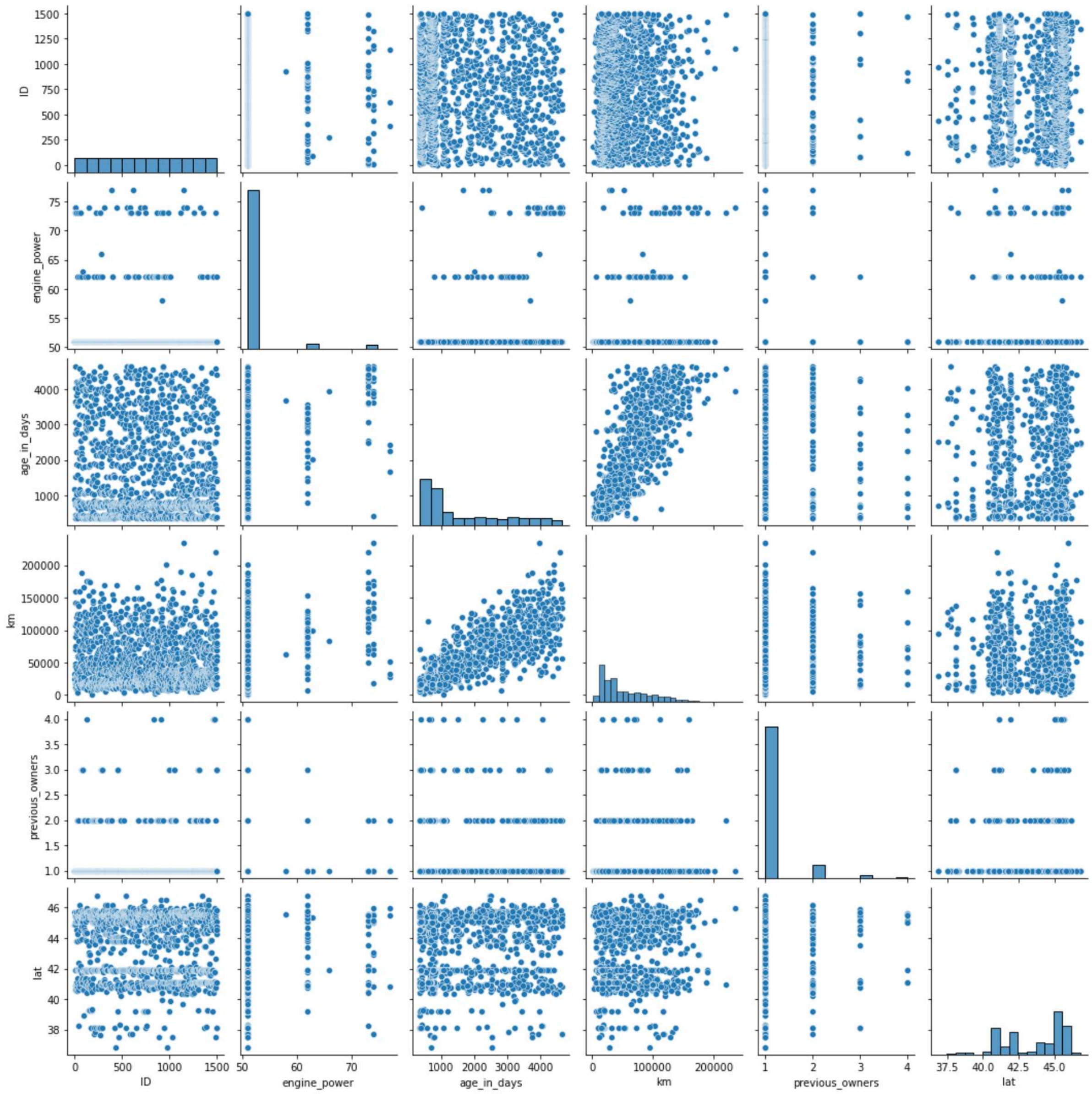
In [7]:
```
#To Display column heading
df.columns
```

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)
```

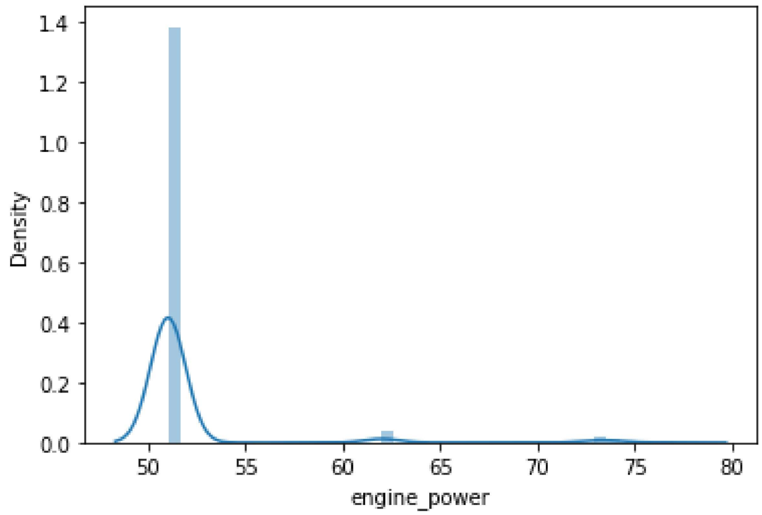
Out[8]: <seaborn.axisgrid.PairGrid at 0x1bb6cdc2970>



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'>
```

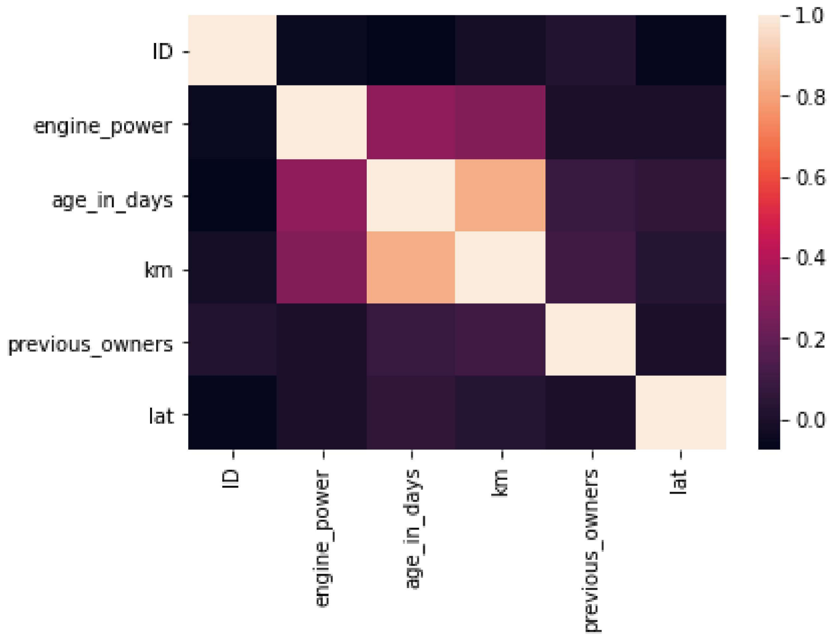



In [10]:
```
df1=df[['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price']]
```

## 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)
```

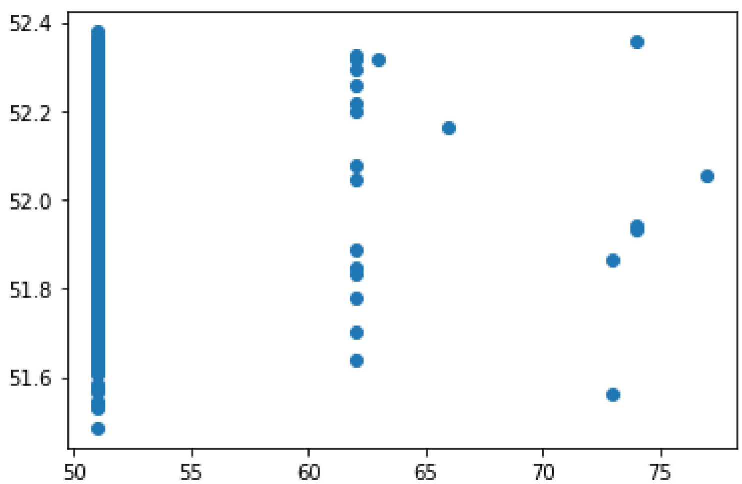
Out[14]: LinearRegression()

In [15]:
```
lr.intercept_
```
Out[15]: 51.61023539100676

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

| | Co-efficient |
|---|---|
| **ID** | -0.000506 |
| **previous_owners** | 0.027045 |
| **lat** | 0.015912 |

In [17]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```
Out[17]: <matplotlib.collections.PathCollection at 0x1bb6f4bd8b0>




# Accuracy

In [18]:
```
lr.score(x_test,y_test)
```
Out[18]: -0.005090892788124801

In [19]:
```
lr.score(x_train,y_train)
```
Out[19]: 0.0029515105123596452

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]:
```
-0.0050811016026659495
```

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]:
```
-0.0038963771735300856
```

## ElasticNet

In [25]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[25]:
```
ElasticNet()
```

In [26]:
```
print(en.coef_)
```

```
[-0.00050511  0.          0.        ]
```

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

```
52.331413192697504
```

In [28]:
```
print(en.predict(x_test))
```

```
[52.21675228 52.16068461 52.16118973 52.09653512 52.12027548 51.97682306
 51.58333914 52.02531402 51.65456024 51.75911887 51.94197019 52.24554379
 52.29908589 52.25211027 52.29504497 51.75255238 52.27332507 51.63789147
 52.21170114 51.86822352 52.09198909 51.94095996 51.79397174 52.2925194
 51.71668928 52.26423301 52.08037146 52.15361301 51.70507165 51.91620937
 51.82175302 52.23291593 52.03339585 52.07279475 52.23241082 51.69698983
 51.68486709 52.10007091 52.11522434 52.05814644 52.27585064 51.578288
 51.73790407 51.98237932 51.78841549 51.65506535 51.89297412 52.1066374
 51.91671448 52.29201429 52.01925265 51.59091586 51.77427229 51.76972626
 52.06218735 51.61314088 51.97783329 51.85559567 51.80558937 51.82377348
 51.8389269  52.20159886 51.95005201 52.0889584  52.00864525 51.9960174
 52.31929045 51.62374828 51.76922115 52.17937384 51.71264837 51.84498827
 51.69547448 51.706587   51.91065311 52.0136964  51.68688754 51.84296781
 51.86115192 51.70254608 52.19402215 51.89701503 52.01824242 51.97379238
 52.18442498 52.03693165 51.75457284 51.96975147 51.84397804 51.92176562
 51.72982225 52.13391356 51.62273805 51.86468772 51.81367119 52.07532032
 51.79195129 51.83387576 52.08592772 52.14654142 52.29100406 51.6995154
 52.28140689 51.6353659  52.07835101 52.23998753 52.00763503 51.58636983
 51.70456654 52.20412443 51.99551229 51.70557677 52.28393246 51.94651621
 51.89347923 52.14452096 52.26877904 52.21877274 51.57576243 52.26625347
 51.88842809 51.69496937 51.68991823 51.63081987 51.72477111 52.12128571
 51.83640133 52.13239822 51.83488599 51.75204727 52.28847849 52.13441868
 51.5888954  52.18493009 51.61516134 51.66011649 51.91822982 52.13593402
 51.90812754 51.98743046 51.61415111 52.12785219 51.91418891 52.02632425
 52.18189941 51.78740526 52.18391986 51.6783006  51.66971366 51.95257758
 51.90509685 52.31575465 52.18846589 52.02127311 51.95712361 51.57374198
 52.08996863 51.78690014 52.16321018 52.14856187 52.26221255 52.29706543
 51.88388206 52.0495595  51.75053193 51.68739266 52.01470662 52.20462954
 52.25918187 52.00510946 52.07633055 51.9389395  51.61112042 51.85711101
 52.15613858 51.94146507 52.28292223 52.30514726 51.8995406  52.33040296
 51.74699613 52.27029438 51.89499457 52.17129201 51.72123531 51.63637613
 52.07734078 51.80912517 52.27281995 52.28999383 52.06319758 51.68284663
 52.15714881 51.64445796 51.62879942 51.66365229 51.83084508 52.32232114
 51.76265467 51.57727777 52.31272397 51.97278215 51.67425969 51.93287813
 51.73285293 52.31828022 51.95409293 51.70860745 52.25514096 52.10411183
 51.70759722 51.8030638  52.11269877 51.72881202 52.16422041 52.06117713
 51.67779549 52.07936123 51.74396544 51.84801896 51.64647841 51.71214325
 52.21978297 52.00106854 52.18947612 51.90206617 51.96166964 51.70709211
 51.85761613 51.97480261 51.6530449  51.9424753  51.96419521 52.03491119
 51.71062791 51.90711731 51.9354037  51.90358151 52.11926525 52.01874754
 51.88640763 51.94298042 51.97631795 52.29858077 51.63738636 51.72224553
 51.7924564  52.21119603 51.8101354  51.96874124 52.08744306 52.07228964
 51.87681046 51.7783132  51.8601417  52.06622827 52.12532662 51.76063421
 52.20260909 51.6424375  51.96267987 51.8424627  52.1637153  52.05915667
 52.188971   51.70911257 52.11977037 52.06471292 52.20109374 52.17987895
 51.86418261 51.64193239 52.21574205 52.15058233 51.6459733  51.96065941
 52.03440608 52.15411813 52.08491749 52.18998123 52.04299302 52.18695055
 51.91267357 51.75709841 51.58283403 52.06976407 52.28090178 51.82680416
 52.31070351 51.61718179 52.24099776 51.90459174 52.24503867 52.27938644
 51.59748234 51.67931083 51.61869713 51.57424709 51.92580653 51.93136279
 51.97177192 52.06521804 51.71012279 51.8960048  52.31171374 51.84094736
 51.95914407 51.7601291  52.05006461 51.92378608 51.82225813 52.22584434
 52.00359411 52.29555009 52.25968698 52.18139429 52.25463584 51.64041704
 52.12987265 52.23594662 51.75103704 51.77881832 51.98591512 52.12330617
 51.95106224 52.16219995 52.20867046 52.29807566 51.95964918 52.17836361
 51.84347293 51.88236672 52.09805046 52.26978927 51.8853974  51.72527622
 52.10360671 51.66819832 51.71769951 52.26271767 51.95813384 52.21927785
```

```
 51.97429749 51.79144617 52.32939274 52.20766023 52.09754534 52.1495721
 52.04804416 51.61920225 52.07481521 51.85205987 52.21624717 51.81569165
 51.67274435 52.29049895 52.12179082 52.0924942  52.29656032 52.01773731
 51.68234152 51.8818616  51.80003311 52.3101984  52.32434159 51.82781439
 51.99702763 52.00207877 51.58081357 51.81468142 52.28241712 51.87479001
 51.60505905 51.66718809 51.74901658 52.3208058  52.08188681 51.89650991
 51.61061531 52.0854226  52.32989785 51.99046115 51.57980335 52.31878534
 51.81872234 51.7318427  52.17381758 52.11572945 51.60354371 52.04703393
 52.16573575 52.01622197 52.03743676 51.72729668 52.19149657 51.93338325
 52.16977667 52.20665    52.003089   52.17886872 51.90964288 51.8353911
 51.87377978 51.87428489 51.90004571 51.86620307 51.71719439 52.12583174
 52.30060123 52.0460237  51.91519914 51.72679156 52.06673338 52.24453356
 51.84852407 51.72780179 52.2601921  52.15866416 52.04198279 52.19654772
 52.08643283 51.8459985  52.09501977 51.65860115 52.03087028 52.19048635
 51.97025658 51.77629275 52.23140059 51.83842179 52.2101858  51.7853848
 52.08441238 51.91721959 52.15512836 52.15462324 52.11471922 51.98995603
 51.81619677 51.80508425 52.09451466 51.62172782 52.13138799 52.31727
 52.07127941 51.85963658 51.69143357 52.18644543 52.15563347 52.02278845]
```

In [29]:
```
print(en.score(x_test,y_test))
```

```
-0.004198027079855393
```

# Evaluation Metrics

In [30]:
```
from sklearn import metrics
```

In [31]:
```
print("Mean Absolute Error",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error 1.5839422630310935
```

In [32]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 11.316082599840229
```

In [33]:
```
print("Root Mean Absolute Error:",np.sqrt(metrics.mean_squared_error(y_test,pr
```

```
Root Mean Absolute Error: 3.3639385547064067
```

In [ ]: