In [31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [32]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\fiat500-VehicleSelection-Dataset.csv"
df.fillna(0,inplace=True)
df
```

Out[32]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
0	1	lounge	51	882	25000	1	44.907242	8.611560
1	2	pop	51	1186	32500	1	45.666359	12.241890
2	3	sport	74	4658	142228	1	45.503300	11.417840
3	4	lounge	51	2739	160000	1	40.633171	17.634609
4	5	pop	73	3074	106880	1	41.903221	12.495650
...	...	...	...	...	...	...	...	...
1533	1534	sport	51	3712	115280	1	45.069679	7.704920
1534	1535	lounge	74	3835	112000	1	45.845692	8.666870
1535	1536	pop	51	2223	60457	1	45.481541	9.413480
1536	1537	lounge	51	2557	80750	1	45.000702	7.682270
1537	1538	pop	51	1766	54276	1	40.323410	17.568270

1538 rows × 9 columns

In [33]:
```
df.head()
```

Out[33]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	lounge	51	882	25000	1	44.907242	8.611560	8900
1	2	pop	51	1186	32500	1	45.666359	12.241890	8800
2	3	sport	74	4658	142228	1	45.503300	11.417840	4200
3	4	lounge	51	2739	160000	1	40.633171	17.634609	6000
4	5	pop	73	3074	106880	1	41.903221	12.495650	5700

In [34]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1538 entries, 0 to 1537
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1538 non-null   int64  
 1   model            1538 non-null   object 
 2   engine_power     1538 non-null   int64  
 3   age_in_days      1538 non-null   int64  
 4   km               1538 non-null   int64  
 5   previous_owners  1538 non-null   int64  
 6   lat              1538 non-null   float64
 7   lon              1538 non-null   float64
 8   price            1538 non-null   int64  
dtypes: float64(2), int64(6), object(1)
memory usage: 108.3+ KB
```

In [5]:
```
import seaborn as sns
```

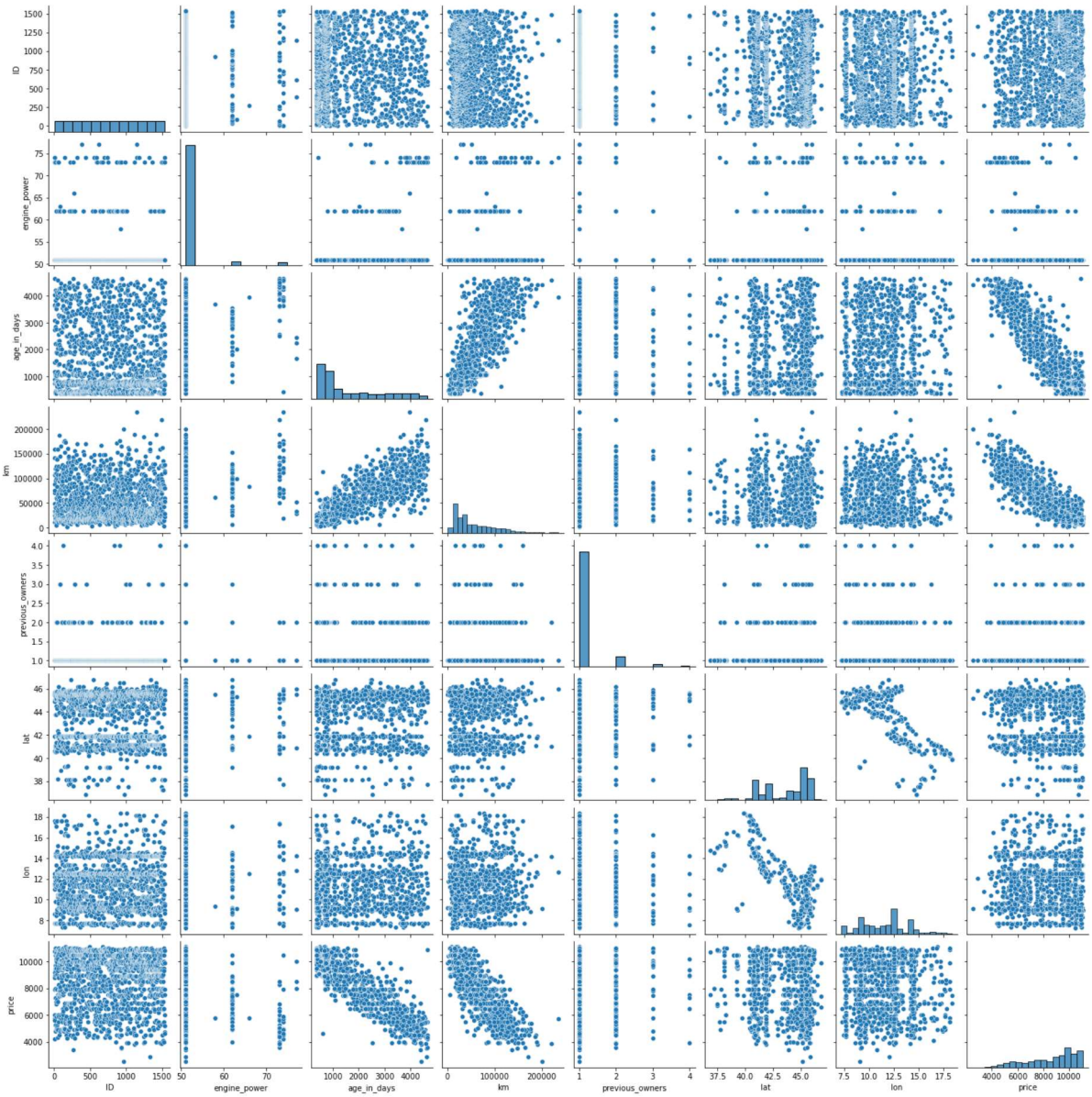
In [35]:
```
df.describe()
```

Out[35]:

	ID	engine_power	age_in_days	km	previous_owners	lat	
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.541361	
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.133518	
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.855839	
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.802990	
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.394096	
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.467960	
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.795612	

In [36]:
```
sns.pairplot(df)
```

Out[36]:
```
<seaborn.axisgrid.PairGrid at 0x235ee892c40>
```

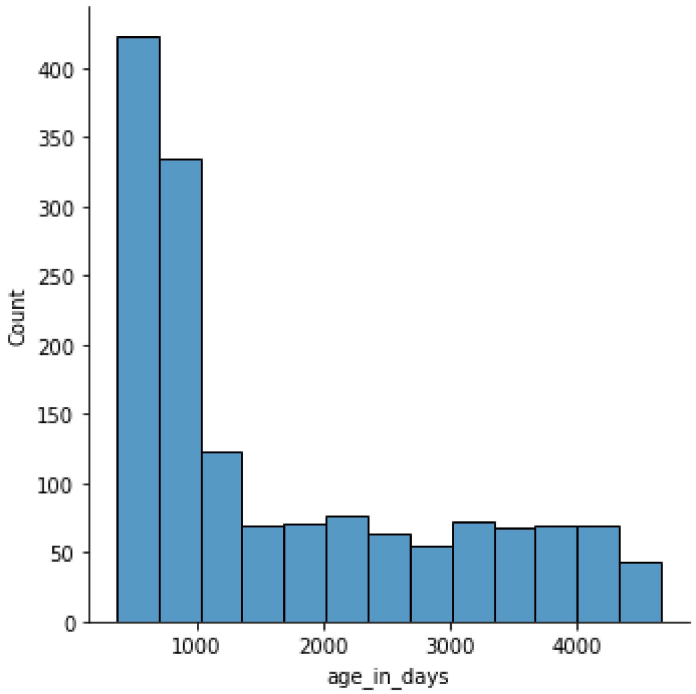


In [37]:
```
df1=df.drop(['age_in_days'],axis=1)
df1
df1=df1.drop(df1.index[1537:])
df1.isna().sum()
```

Out[37]:
```
ID                0
model             0
engine_power      0
km                0
previous_owners   0
lat               0
lon               0
price             0
dtype: int64
```

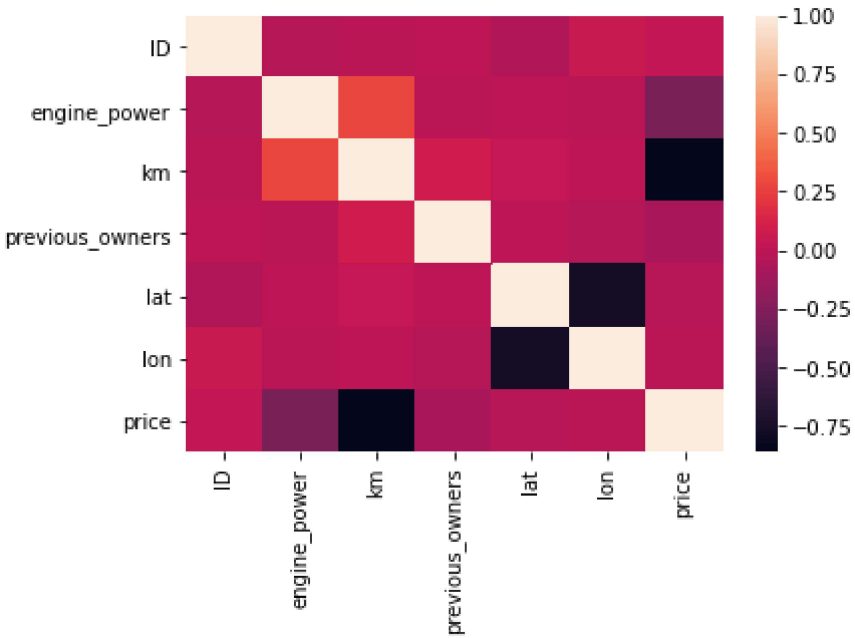
In [39]:
```
sns.displot(df['age_in_days'])
```

Out[39]:
```
<seaborn.axisgrid.FacetGrid at 0x235f2c9d880>
```



In [40]:
```
sns.heatmap(df1.corr())
```

Out[40]:
```
<AxesSubplot:>
```



In [41]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [42]:
```
df1.isna().sum()
```

Out[42]:
```
ID                0
model             0
engine_power      0
km                0
previous_owners   0
lat               0
lon               0
price             0
dtype: int64
```

In [13]:
```
y=df1['fixed acidity']
x=df1.drop(['chlorides','residual sugar'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
print(x_train)
```

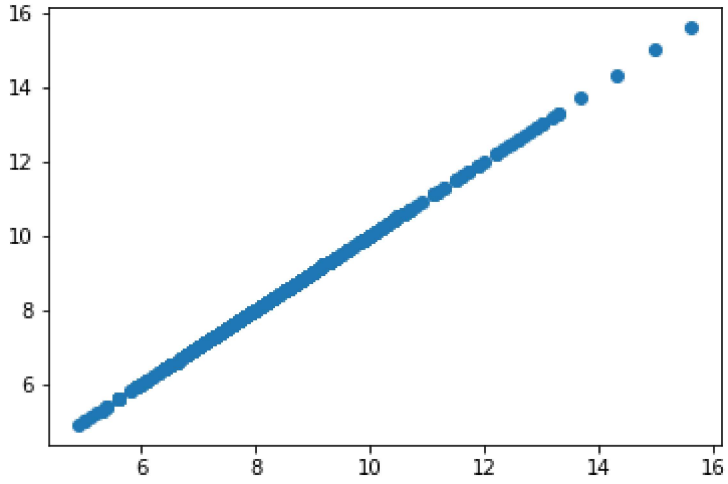
```
      fixed acidity  volatile acidity  free sulfur dioxide  \
1355            6.1             0.320                  5.0
1217            8.2             0.340                 43.0
824             7.1             0.480                  6.0
461             8.3             0.615                  6.0
1149           10.0             0.350                  6.0
...             ...               ...                  ...
1390            6.0             0.490                 15.0
952             8.2             0.310                  6.0
386             7.8             0.540                 23.0
59              7.3             0.390                  9.0
585             7.6             0.510                  8.0

      total sulfur dioxide  density    pH  sulphates  alcohol  quality
1355                  32.0  0.99464  3.36       0.44     10.1        5
1217                  74.0  0.99408  3.23       0.81     12.0        6
824                   16.0  0.99682  3.24       0.53     10.3        5
461                   19.0  0.99820  3.26       0.61      9.3        5
1149                  11.0  0.99585  3.23       0.52     12.0        6
...                    ...      ...   ...        ...      ...      ...
1390                  33.0  0.99292  3.58       0.59     12.5        6
952                   10.0  0.99536  3.31       0.68     11.2        7
386                   48.0  0.99810  3.41       0.74      9.2        6
59                    46.0  0.99620  3.41       0.54      9.4        6
585                   38.0  0.99800  3.47       0.66      9.6        6

[1075 rows x 9 columns]
```

In [43]:
```
model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

Out[43]:
```
5.329070518200751e-15
```

In [44]:
```
prediction=model.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[44]: <matplotlib.collections.PathCollection at 0x235f3c83af0>



In [45]:
```
model.score(x_test,y_test)
```

Out[45]: 1.0

In [46]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [47]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[47]: Ridge(alpha=10)

In [48]:
```
rr.score(x_test,y_test)
```

Out[48]: 0.9999860069949209

In [49]:
```
la =Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[49]: Lasso(alpha=10)

In [50]:
```
la.score(x_test,y_test)
```

Out[50]: -0.00021750194178205007

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

Out[51]: ElasticNet()

In [52]:
```
print(en.coef_)
```

```
[ 0.70820719 -0.         -0.          -0.0015529   0.         -0.
  0.         -0.          0.        ]
```

In [53]:
```
print(en.intercept_)
```

```
2.5191973790547832
```

In [54]:
```
print(en.predict(x_test))
```

```
[ 7.38719227  7.60774687  9.79354383  8.50945351  7.9407224   8.92195466
  8.4902063   8.56785107 12.15486142  9.99824151  8.87475534  8.87847359
  8.6576345   6.92965715  7.73757761  8.70516179 10.63035683  7.23933925
  9.15460446  6.8252852   6.72806526  7.43782535  8.02241338  8.44889059
  7.05421674  8.37557655  9.50870805  8.71570408  8.65297581  8.87753316
  7.78571736 11.05993984  7.58972459  8.31967232 11.87157855  8.71570408
  8.8545677   7.64812215  9.96407781  7.79908096  9.24749368  9.23257719
  8.43491454  8.05812997  9.14528709  7.42789551  8.72657435  8.72906768
  7.60120732 10.76950494  8.74087838  9.72272311  8.60606098  9.08378374
  7.8829373   9.79354383  8.28612108  6.88212986  7.5288337   7.80063385
  8.7880777  10.91580507  7.265126    8.56196746  8.4249847   7.10390939
  8.80671244 10.04732169  8.00006039  8.72346856  6.33142103  9.00209275
  8.3125203   8.2814624  12.62326681  7.83324465  8.31096741  7.43316666
  7.35924015  7.03308867 10.36881449  8.11746796  7.73757761  7.85686605
  7.74844788  8.01558934  7.87456036  7.38098069  8.30447134  7.44714272
  8.22555817  6.13448782  8.03671741  7.84722071  7.21138713  7.28841943
  5.92419102  6.80975625  8.59114449  6.76938097  9.63171475  9.88238683
 10.91114638  9.01856214  7.90561827  7.91338274  7.2977368   9.46461453
  7.09148623  9.08844242  7.44558982  7.42229639  7.72421401  8.4799485
  8.29576643  9.65190239 11.65290481  7.09459202  6.74204132  7.70092059
  8.69895021  7.59033705 11.60071883 10.71266028  9.40838232  7.43876578
  7.65993285  7.76242393  9.42019303  7.34526409  7.67235601  7.03308867
  7.76242393  7.48069395 10.02092247  7.27599627  9.48635506  9.05117294
  7.02532419  8.77876033 11.52989811  9.50404937  8.56474527  8.61476589
  7.95935714 13.10503578  7.44093114 10.59059402  7.3437112   8.42931542
  9.44565181  8.16871351  7.70713217  6.89300013  7.77639999  7.87983151
  8.61537836  7.91182985  9.99824151  7.9733332   7.47448237  7.52450299
  9.76653215 11.61780068  8.5806022   8.37247075  7.73819007  7.90095958
  7.58412547 10.4309303   9.22358779  8.78963059  9.44969803  9.85970586
  7.16541274  9.20773087  7.92425301  8.52625087  8.91729598  8.33830706
  7.65649909  8.2472987   7.9391695   9.57115184  7.8074579   6.93836206
  7.77639999  9.56865851  8.07925804 10.18896313  8.30414336 10.78192811
  7.8192686  10.09174319  7.88853642  9.25215237  7.9391695   5.97421164
  6.95233812  8.37868234  8.72191567  8.93593072  9.57797588  7.29618391
  7.93046459  9.5953857   8.35538891 10.97730842  8.86045131  8.01060268
  7.04955806  6.44915658  8.65730653  8.70638671  7.24493837  6.25810698
  7.43161377  9.08006549  7.31792444  7.43782535  7.89507597  9.47920304
  7.5732552   9.16796805  7.28686654  6.60971725  8.11996129  7.48474018
  7.17473011  7.37881533  7.41702525  6.46497002  8.10410437 10.91425217
  6.27208304 10.0442159   7.46827079  6.9451861   9.5953857   9.20773087
  7.30488882  9.12975813  6.36403183  9.85038849  8.89555544  6.92033978
  7.38098069 10.50951549  7.2014573   7.23562099  7.46111878  9.02011503
  7.07301547  7.37881533  7.39123849  7.48535264  9.71651153  8.87287448
 11.65290481 10.34613352  7.60431311  9.90506779  7.83479755  9.99513572
  7.36949796  8.37557655  9.70037011 13.50045509  7.71023796  8.60450809
  7.4667179   9.91344473  7.6533933   8.30786162  7.85809098  8.92040177
  9.85038849 11.32675333  9.19747306  7.17317721  8.05502418  6.97563155
  9.55002376  9.85194138  7.89507597  8.30009714  7.64812215  7.96806205
  6.19909696  9.065149    8.58338002  9.47920304  7.78261157  7.24027968
  7.80839833  9.93207947  8.40324417  7.78821069  8.25195739  5.98258858
  7.25270284  7.55927915 11.20219374  7.35025075  8.60450809  7.55927915
 10.63252219  6.48797896  7.53753861  9.73453381  6.48082694  8.72224364
  8.8309463   8.01370847  7.87051414  6.46807581  8.58897914  7.8844902
  8.2370409   8.10876305  9.15460446  8.76167848  7.55772625  8.67067012
  7.31947734  8.7094925   7.87921905  7.66524748  8.33209548  7.66581646
  8.08857542  8.09384656  7.46827079  7.83884377  7.74129586  7.54996178
  9.01606881  7.68788496 11.4115501   7.44403693  8.23332265  7.265126
```

```
 8.74925532  6.68458419  7.09614491  7.47914106 11.48392372  8.75330154
 7.35768726  9.52362454  7.90345291  7.55306757  7.78261157  9.56399982
 7.46422457  9.14528709  8.33520127  7.67856759  7.53071457  6.59296337
 7.81211658  9.07912505  7.75931814  6.33607972  9.23723588 11.12671433
 7.31171286  9.69942968  7.23002188 10.35700379  8.8902843   7.44187157
 6.66035033  9.02011503  7.5748081   7.2887474   7.94288775  7.68477917
 7.7658577   7.59593617  7.51951633  7.83479755  7.63475855  8.73156101
 6.76782807  7.65588662 10.32749878  6.96941996  9.98892414  9.17512007
11.14722994  7.70247348  7.81677527 10.57817086  8.63961222  8.41223357
 9.86902323  7.97022741  7.55306757  7.15454247  9.46677988  7.41608481
 7.54624352  8.46907823  6.65413875  8.33054258  8.59114449  7.74411716
 8.45260885  6.96165549  8.93748361  7.13995395  8.50324193  9.36800705
 9.20679043  7.87921905  9.64569081  6.176416   10.07216802  8.61537836
 8.8561206   8.62935441  7.4191906  11.85604959  7.04955806  9.43727487
10.49864523  8.29948468  7.25270284  7.80063385  7.8177157   9.27450536
 7.56238494  8.94835388  8.78497191  7.52572791 11.39757405  7.59965442
 6.79578019  6.39353684  6.66035033  7.89413554  8.05657708  9.84667024
 8.26903924  9.65190239  5.98880016  6.12423002  8.12927866  7.03464156
 8.2370409  11.81318099  8.36160049  7.6770147   7.3437112   9.85504717
 7.95314556 10.30731114  7.05421674  7.41608481  8.39858548  8.23393511]
```

In [55]:
```
print(en.score(x_test,y_test))
```

```
0.9157209899533858
```

EVALUATION METRICS

In [56]:
```
from sklearn import metrics
```

In [57]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.3918640170192439e-15
```

In [58]:
```
print("Mean squarred Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean squarred Error: 3.811120217431382e-30
```

In [59]:
```
print("Root Mean squared Error:",np.sqrt(metrics.mean_squared_error(y_test,pre
```

```
Root Mean squared Error: 1.952209060892655e-15
```

In [ ]:
```

```