kaviyadevi 20106064

In [2]:

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

In [3]:

```
#to import dataset
data1=pd.read_csv(r"C:\Users\user\Downloads\fiat500_VehicleSelection_Dataset - fi
data1
```

Out[3]:

	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
...	...	...	...	...	...	...	...	..
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	length
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	conca
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Null values
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	find
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	search

1549 rows × 11 columns

In [4]:
```
data=data1.head(100)
data
```

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
...	...	...	...	...	...	...	...	...
95	96.0	sport	51.0	4292.0	165600.0	1.0	44.715408	11.30830002
96	97.0	pop	51.0	1066.0	28000.0	1.0	41.769051	12.66281033
97	98.0	sport	51.0	2009.0	86000.0	2.0	40.633171	17.63460922
98	99.0	lounge	51.0	456.0	18592.0	2.0	45.393600	10.48223972
99	100.0	pop	51.0	731.0	41558.0	2.0	45.571220	9.159139633

100 rows × 11 columns

DATA CLEANING AND PREPROCESSING

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               100 non-null    float64
 1   model            100 non-null    object 
 2   engine_power     100 non-null    float64
 3   age_in_days      100 non-null    float64
 4   km               100 non-null    float64
 5   previous_owners  100 non-null    float64
 6   lat              100 non-null    float64
 7   lon              100 non-null    object 
 8   price            100 non-null    object 
 9   Unnamed: 9       0 non-null      float64
 10  Unnamed: 10      0 non-null      object 
dtypes: float64(7), object(4)
memory usage: 8.7+ KB
```

In [6]:
```
data.isnull()
```

Out[6]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price	Unnam
0	False	False	False	False	False	False	False	False	False	T
1	False	False	False	False	False	False	False	False	False	T
2	False	False	False	False	False	False	False	False	False	T
3	False	False	False	False	False	False	False	False	False	T
4	False	False	False	False	False	False	False	False	False	T
...	...	...	...	...	...	...	...	...	...	
95	False	False	False	False	False	False	False	False	False	T
96	False	False	False	False	False	False	False	False	False	T
97	False	False	False	False	False	False	False	False	False	T
98	False	False	False	False	False	False	False	False	False	T
99	False	False	False	False	False	False	False	False	False	T

100 rows × 11 columns

In [7]:
```
data.describe()
```

Out[7]:

	ID	engine_power	age_in_days	km	previous_owners	lat	Unnai
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	
mean	50.500000	53.010000	1935.300000	58812.180000	1.180000	43.612648	
std	29.011492	6.014284	1414.251278	44728.034639	0.500101	2.083451	
min	1.000000	51.000000	366.000000	4000.000000	1.000000	38.218128	
25%	25.750000	51.000000	723.500000	19781.750000	1.000000	41.744165	
50%	50.500000	51.000000	1446.000000	44032.000000	1.000000	44.831066	
75%	75.250000	51.000000	3265.500000	95075.750000	1.000000	45.396568	
max	100.000000	74.000000	4658.000000	188000.000000	3.000000	46.176498	

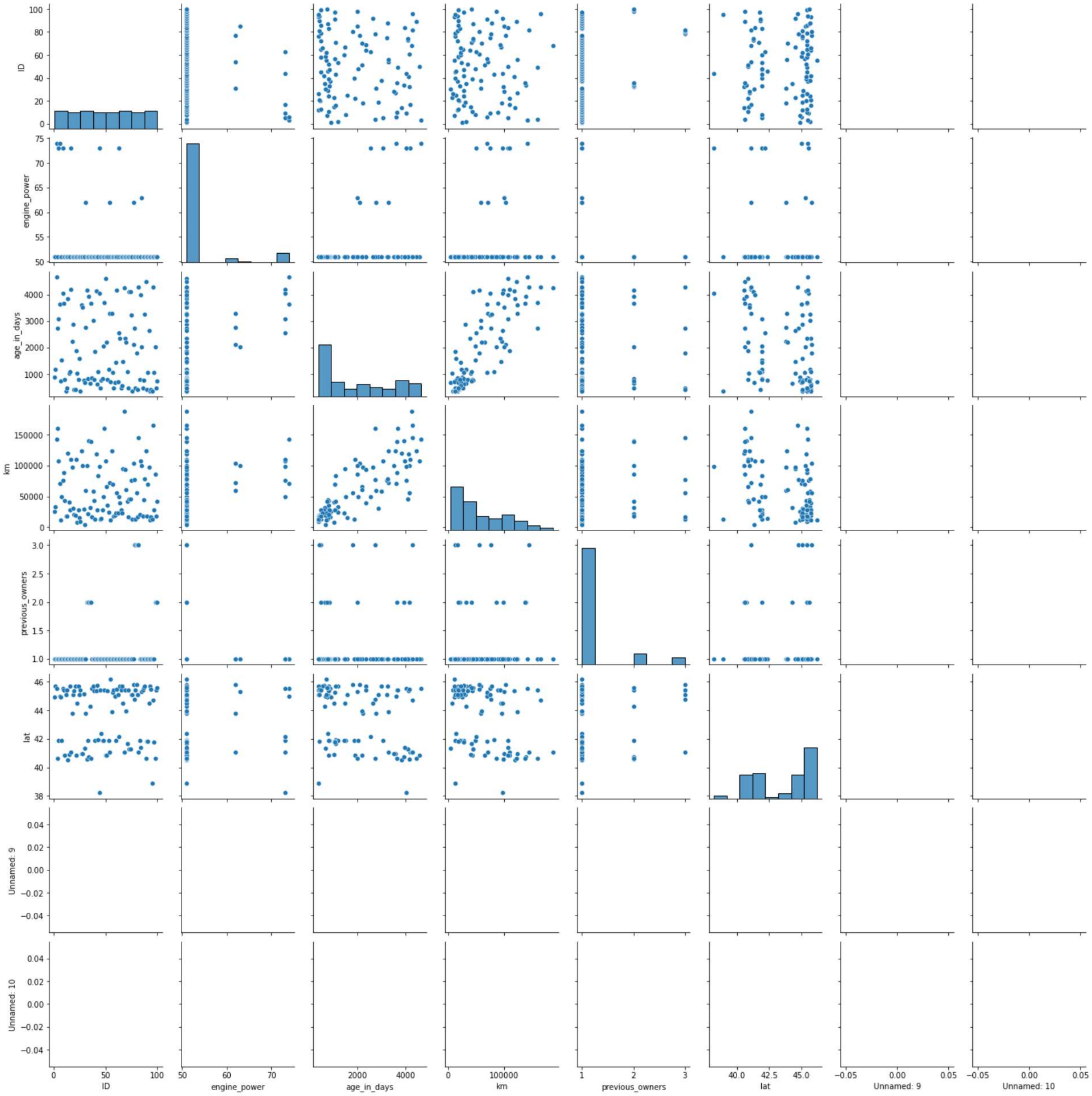
In [8]:
```
data.columns
```

Out[8]:
```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

EDA and DATA VISUALIZATION

```
In [9]: sns.pairplot(data)
```

```
Out[9]: <seaborn.axisgrid.PairGrid at 0x1bbcb339220>
```



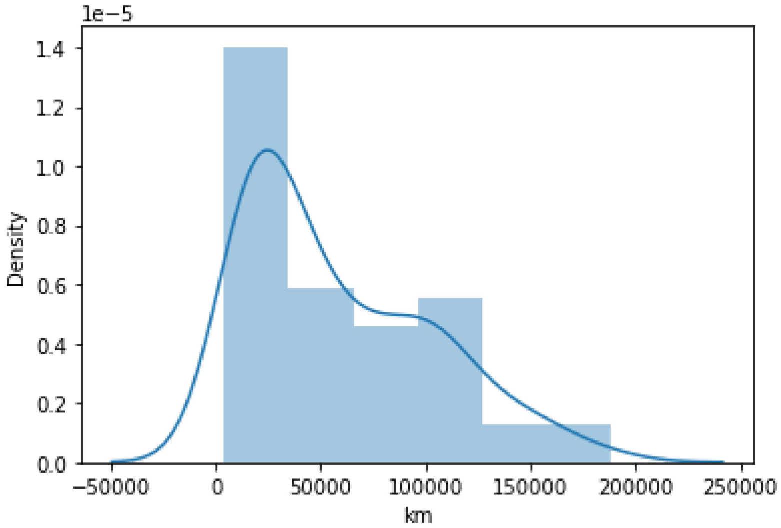
In [10]:

```
sns.distplot(data['km'])
```

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

Out[10]:

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

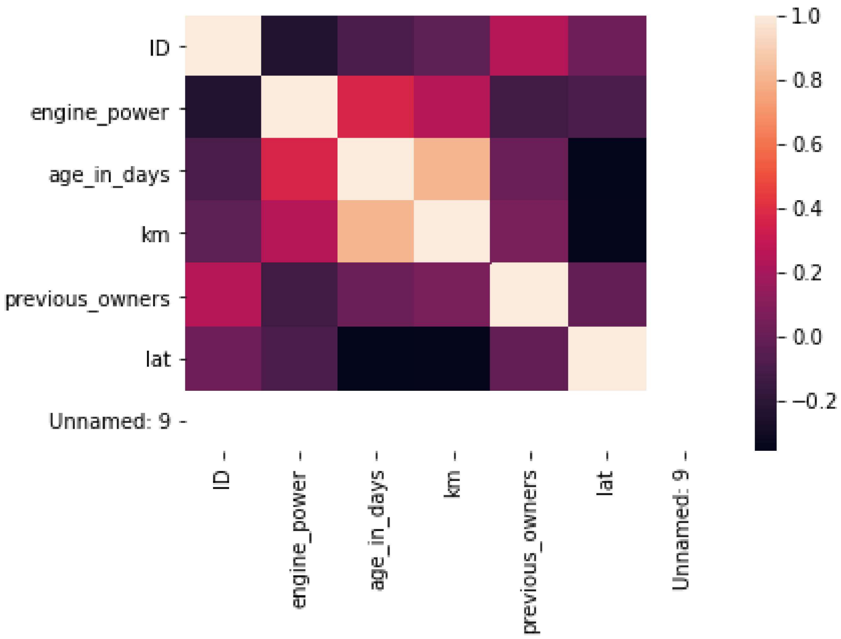


In [11]:

```
df=data[['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10']]
```

In [12]:
```
sns.heatmap(df.corr())
```

Out[12]:
```
<AxesSubplot:>
```



TRAINNING MODEL

In [13]:
```
x=df[['age_in_days','previous_owners','lat', 'lon']]
y=df[['km']]
```

In [14]:
```
#to split my dataset into trainning and test

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 [15]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

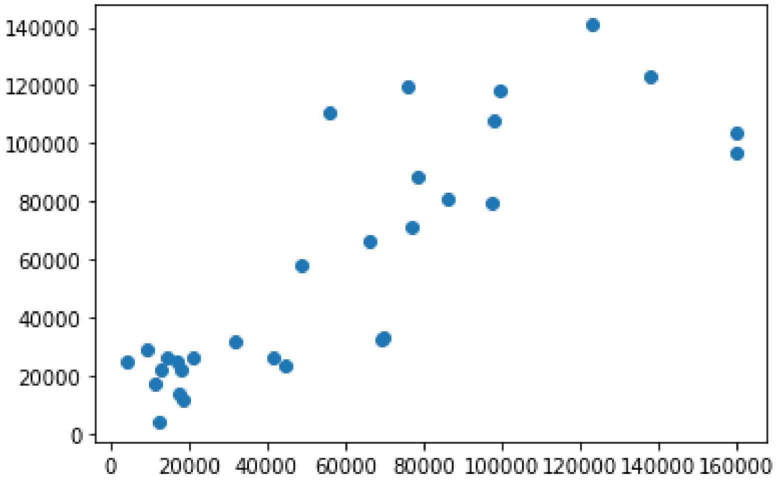
Out[15]:
```
LinearRegression()
```

In [16]:
```
#to find intercept
print(lr.intercept_)
```

```
[-236692.26303092]
```

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

Out[17]:
```
<matplotlib.collections.PathCollection at 0x1bbd4695490>
```



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

```
0.6978879547658228
```

RIDGE AND LASSO REGRESSION

In [19]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [20]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[20]:
```
Ridge(alpha=10)
```

In [21]:
```
rr.score(x_test,y_test)
```

Out[21]:
```
0.6955790410552533
```

In [22]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[22]:
```
Lasso(alpha=10)
```

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

Out[23]: 0.6978476743103819
```

```
In [24]: from sklearn.linear_model import ElasticNet
         en=ElasticNet()
         en.fit(x_train,y_train)

Out[24]: ElasticNet()
```

```
In [25]: print(en.coef_)

         [  25.36884461 1067.76521289 2407.16296854 3530.06014973]
```

```
In [26]: print(en.predict(x_test))

         [ 20173.79228902  20971.71692727  77908.69504411  26801.53035467
           24491.52062884  28150.45957597  32916.83686823  70836.99532493
          111645.47203985  75226.46305016  25276.53567127 117790.39904226
          136692.89980166  24772.85165882 116932.8435895   28998.97909626
           25838.94939963 102170.96805893  88347.79559302  35275.57278706
          116261.02819108  15063.24996831 110754.53183837  59836.14649424
           20971.71692727  15570.81745733  67359.51199745  28012.08433562
            8104.68347529  92677.95440279]
```

```
In [27]: print(en.score(x_test,y_test))

         0.6901501275328039
```

```
In [28]: from sklearn import metrics
```

```
In [29]: print("Mean Absolute error",metrics.mean_absolute_error(y_test,prediction))

         Mean Absolute error 18077.18238098023
```

```
In [30]: print("Mean Squared error",metrics.mean_squared_error(y_test,prediction))

         Mean Squared error 614997694.0900805
```

```
In [31]: print("Root Mean Absolute error",np.sqrt(metrics.mean_squared_error(y_test,predic

         Root Mean Absolute error 24799.147043599714
```