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

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

read a file

In [3]:
```
sai=pd.read_csv("/home/placement/Downloads/fiat500.csv")
```

describe data

In [4]:
```
sai.describe()
```

Out[4]:

	ID	engine_power	age_in_days	km	previous_owners	lat	lon	price
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.541361	11.563428	8576.003901
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.133518	2.328190	1939.958641
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.855839	7.245400	2500.000000
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.802990	9.505090	7122.500000
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.394096	11.869260	9000.000000
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.467960	12.769040	10000.000000
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.795612	18.365520	11100.000000

In [5]:
```
sai.head()
```

Out[5]:

	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 [6]:
```
sai.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
```

maping str into int

In [7]:
```
sai['model']=sai['model'].map({'lounge':1,'pop':2,'sport':3})
sai
```

Out[7]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	1	51	882	25000	1	44.907242	8.611560	8900
1	2	2	51	1186	32500	1	45.666359	12.241890	8800
2	3	3	74	4658	142228	1	45.503300	11.417840	4200
3	4	1	51	2739	160000	1	40.633171	17.634609	6000
4	5	2	73	3074	106880	1	41.903221	12.495650	5700
...	...	...	...	...	...	...	...	...	...
1533	1534	3	51	3712	115280	1	45.069679	7.704920	5200
1534	1535	1	74	3835	112000	1	45.845692	8.666870	4600
1535	1536	2	51	2223	60457	1	45.481541	9.413480	7500
1536	1537	1	51	2557	80750	1	45.000702	7.682270	5990
1537	1538	2	51	1766	54276	1	40.323410	17.568270	7900

1538 rows × 9 columns

eliminating columns

In [8]:
```
sai.drop(['lat','lon','ID'],axis=1)
```

Out[8]:

	model	engine_power	age_in_days	km	previous_owners	price
0	1	51	882	25000	1	8900
1	2	51	1186	32500	1	8800
2	3	74	4658	142228	1	4200
3	1	51	2739	160000	1	6000
4	2	73	3074	106880	1	5700
...	...	...	...	...	...	...
1533	3	51	3712	115280	1	5200
1534	1	74	3835	112000	1	4600
1535	2	51	2223	60457	1	7500
1536	1	51	2557	80750	1	5990
1537	2	51	1766	54276	1	7900

1538 rows × 6 columns

creating dummies

In [9]:
```
pd.get_dummies(sai)
```

Out[9]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	1	51	882	25000	1	44.907242	8.611560	8900
1	2	2	51	1186	32500	1	45.666359	12.241890	8800
2	3	3	74	4658	142228	1	45.503300	11.417840	4200
3	4	1	51	2739	160000	1	40.633171	17.634609	6000
4	5	2	73	3074	106880	1	41.903221	12.495650	5700
...	...	...	...	...	...	...	...	...	...
1533	1534	3	51	3712	115280	1	45.069679	7.704920	5200
1534	1535	1	74	3835	112000	1	45.845692	8.666870	4600
1535	1536	2	51	2223	60457	1	45.481541	9.413480	7500
1536	1537	1	51	2557	80750	1	45.000702	7.682270	5990
1537	1538	2	51	1766	54276	1	40.323410	17.568270	7900

1538 rows × 9 columns

In [10]:
```
y=sai['price']
x=sai.drop('price',axis=1)
x
```

Out[10]:

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

1538 rows × 8 columns

split Train&test

In [11]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=42)
```

In [12]:
```
x_test.head(10)
```

Out[12]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
481	482	2	51	3197	120000	2	40.174702	18.167629
76	77	2	62	2101	103000	1	45.797859	8.644440
1502	1503	1	51	670	32473	1	41.107880	14.208810
669	670	1	51	913	29000	1	45.778591	8.946250
1409	1410	1	51	762	18800	1	45.538689	9.928310
1414	1415	1	51	762	39751	1	41.903221	12.495650
1089	1090	1	51	882	33160	1	45.778999	12.997090
1507	1508	1	51	701	17324	1	45.556549	9.534470
970	971	1	51	701	29000	1	36.855839	14.760470
1198	1199	1	51	1155	38000	1	41.239281	13.933020

In [13]:
```
x_train.head()
```

Out[13]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
527	528	1	51	425	13111	1	45.022388	7.58602
129	130	1	51	1127	21400	1	44.332531	7.54592
602	603	2	51	2039	57039	1	40.748241	14.52835
331	332	1	51	1155	40700	1	42.143860	12.54016
323	324	1	51	425	16783	1	41.903221	12.49565

In [14]:
```
x_train.shape
```

Out[14]: (1030, 8)

In [15]:
```
y_train.head()
```

Out[15]:
```
527    9990
129    9500
602    7590
331    8750
323    9100
Name: price, dtype: int64
```

In [16]:
```
y_test.head()
```

Out[16]:
```
481     7900
76      7900
1502    9400
669     8500
1409    9700
Name: price, dtype: int64
```

ridge algorithm

In [17]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
alpha = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20,30]
ridge=Ridge()
parameters={'alpha':alpha}

ridge_regressor = GridSearchCV(ridge, parameters)

ridge_regressor.fit(x_train, y_train)
```

Out[17]:
```
GridSearchCV(estimator=Ridge(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.0001, 0.001, 0.01, 1,
                                   5, 10, 20, 30]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [18]:
```
ridge_regressor.best_params_
```

Out[18]:
```
{'alpha': 30}
```

In [19]:
```
ridge=Ridge(alpha=30)
ridge.fit(x_train,y_train)
```

Out[19]:
```
Ridge(alpha=30)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:
```
ypred=ridge.predict(x_test)
ypred
```

Out[20]:
```
array([ 5935.946929  ,  7391.09263342,  9713.38936976,  9772.39163591,
       10023.47312697,  9525.50186705,  9722.65710008, 10091.63069299,
        9636.73096762,  9224.80526688, 10447.79862925,  7786.80716239,
        7678.19575064,  6421.69337375,  9518.10281958, 10397.95637713,
        9423.49349618,  7756.23104841,  4710.38423862, 10549.78058093,
       10437.24157043, 10418.31826122,  7632.07525135, 10001.0098366 ,
        7105.59647562,  9108.45900086,  4943.04924745,  6968.66299074,
        7803.8514556 ,  9652.86865587,  7322.65097059,  5323.78883956,
        5582.95352334,  5112.19802844,  8941.8991792 ,  5689.25952661,
       10025.39399906,  8301.88195122,  6203.90223358,  8504.37941863,
        9670.09019089,  6971.5463123 ,  8907.49429479, 10170.87993833,
        8596.94188726, 10157.76054033,  9179.69547237,  8847.52279911,
        7067.90650238,  9037.86413624,  9455.08436147, 10377.61732563,
       10088.89631974,  6940.81914325,  9673.96523836,  9499.69113131,
        9434.42007127, 10524.62886148,  9814.37143871,  7358.73006024,
        9956.75151813,  7063.75881413,  9953.33251064,  7221.0307355 ,
        6469.57255928,  9716.86980556,  9827.02367203,  8681.78718364,
        8483.20353285,  6462.42679159,  7854.31330446,  6673.32795121,
        8241.20718281, 10518.84646819,  7414.525471  ,  8608.19638711,
```

mean squared error

In [21]:
```
from sklearn.metrics import mean_squared_error
Ridge_Error=mean_squared_error(ypred,y_test)
Ridge_Error
```

Out[21]: 586211.7946814292

efficiency

In [22]:
```
from sklearn.metrics import r2_score
r2_score(y_test,ypred)
```

Out[22]: 0.8403752605647871

In [24]:
```
results=pd.DataFrame(columns=['price','predicted'])
results['price']=y_test
results['predicted']=ypred
results=results.reset_index()
results['ID']=results.index
results.head(10)
```

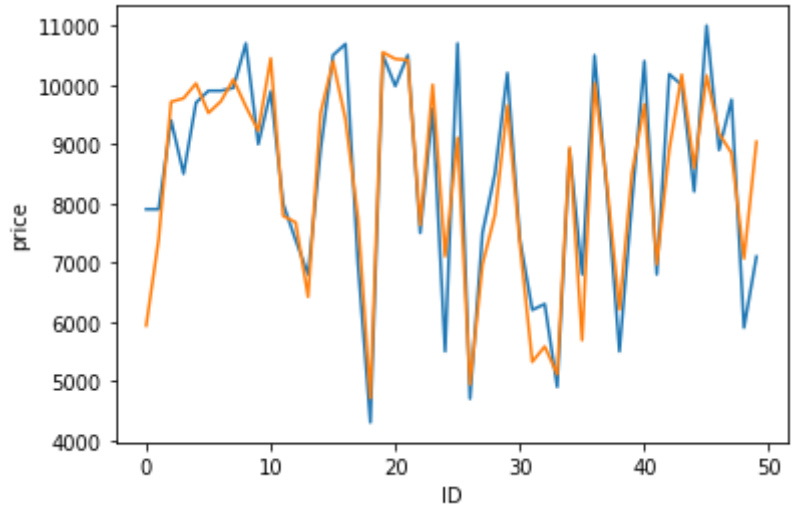
Out[24]:

	index	price	predicted	ID
0	481	7900	5935.946929	0
1	76	7900	7391.092633	1
2	1502	9400	9713.389370	2
3	669	8500	9772.391636	3
4	1409	9700	10023.473127	4
5	1414	9900	9525.501867	5
6	1089	9900	9722.657100	6
7	1507	9950	10091.630693	7
8	970	10700	9636.730968	8
9	1198	8999	9224.805267	9

ploting overall

In [25]:
```
sns.lineplot(x='ID',y='price',data=results.head(50))
sns.lineplot(x='ID',y='predicted',data=results.head(50))
plt.plot()
```

Out[25]: []



only for lounge

In [26]:
```
k=sai.loc[sai.model==1]
k
```

Out[26]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	1	51	882	25000	1	44.907242	8.611560	8900
3	4	1	51	2739	160000	1	40.633171	17.634609	6000
6	7	1	51	731	11600	1	44.907242	8.611560	10750
7	8	1	51	1521	49076	1	41.903221	12.495650	9190
11	12	1	51	366	17500	1	45.069679	7.704920	10990
...	...	...	...	...	...	...	...	...	...
1528	1529	1	51	2861	126000	1	43.841980	10.515310	5500
1529	1530	1	51	731	22551	1	38.122070	13.361120	9900
1530	1531	1	51	670	29000	1	45.764648	8.994500	10800
1534	1535	1	74	3835	112000	1	45.845692	8.666870	4600
1536	1537	1	51	2557	80750	1	45.000702	7.682270	5990

1094 rows × 9 columns

In [27]:
```
y=k['price']
x=k.drop('price',axis=1)
y
```

Out[27]:
```
0        8900
3        6000
6       10750
7        9190
11      10990
        ...
1528     5500
1529     9900
1530    10800
1534     4600
1536     5990
Name: price, Length: 1094, dtype: int64
```

In [28]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=42)
```

ridge algorithm

In [29]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
alpha = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20,30]
ridge=Ridge()
parameters={'alpha':alpha}

ridge_regressor = GridSearchCV(ridge, parameters)

ridge_regressor.fit(x_train, y_train)
```

Out[29]:
```
GridSearchCV(estimator=Ridge(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.0001, 0.001, 0.01, 1,
                                   5, 10, 20, 30]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [30]:
```
ridge_regressor.best_params_
```

Out[30]:
```
{'alpha': 30}
```

In [31]:
```
ridge=Ridge(alpha=30)
ridge.fit(x_train,y_train)
```

Out[31]:
```
Ridge(alpha=30)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```
pred=ridge.predict(x_test)
pred
```

Out[32]:
```
array([ 9912.60175361, 10141.74849333,  4775.23552146,  9870.92696571,
        9630.41788453,  8697.09201357, 10265.82288414, 10293.85186684,
        8614.34973762,  5749.67356711, 10671.67602325,  6488.02221144,
        9752.99829873, 10520.17597908,  8086.90253749,  9498.92882567,
        7801.23188858,  9783.915695  , 10522.29792692,  9641.86872663,
       10614.24629923, 10613.19901763,  9892.38749947,  6510.06240197,
       10549.52425763, 10625.76078907, 10568.39331427,  7946.89947635,
        5931.34546217,  4659.2196909 , 10428.89187791,  5655.72815127,
        9478.32068501, 10329.98145039,  7131.2852707 ,  7921.50560262,
        7874.80635726,  5954.04367445,  9722.42751047,  9680.86485103,
       10527.15377696,  9474.90517944, 10205.46024252,  6549.58459072,
        6994.35871214,  9991.85800581, 10247.34928322,  8277.34560789,
       10300.61976656, 10078.48363687, 10268.33050716,  9823.77891284,
        9669.33394656,  9513.50322923,  9152.34918875,  9631.89820083,
        6653.57742077,  9680.19991056,  9984.99476556,  5648.20897225,
       10341.67956632, 10540.84441014,  9555.12631439,  6825.22781604,
       10486.94645618, 10510.87237214,  9280.22784667,  9695.90865183,
       10300.86096344, 10620.75242063,  7255.08871011,  9512.12507442,
        9609.32308614,  7112.79851998, 10034.0749881 , 10330.98892175,
        8548.73769446,  9520.16121454,  9946.6185962 , 10135.88071505,
       10184.38248658,  6506.0325387 , 10522.28394638,  9889.0361183 ,
        9692.79785416,  6645.09656843,  7830.50421028,  9905.63015012,
        9577.17218464, 10582.05089567,  6097.15652897,  9714.66288548,
        8823.94189014, 10177.17443641, 10542.43749844,  7878.55575401,
        8982.20194888, 10550.72596946,  7089.74287761,  6771.15834746,
        5780.82200321,  6442.12029954,  9580.92651411,  6276.86875321,
        9929.59359002,  9679.28936525, 10535.03640665,  5771.91010315,
        9608.4971782 ,  7176.14803032,  9525.84417673,  9786.76124829,
       10590.77268612, 10590.43852943,  5621.28001026,  4969.18369174,
        9837.01957868,  9839.16975778,  5070.94098034, 10540.48246758,
       10039.03821544,  9743.55236996, 10307.24454309,  4765.01281868,
        5409.7256093 ,  9643.2735831 , 10542.08833354, 10133.68993901,
        8027.5823784 ,  9647.81039882,  9922.44925637,  9856.02030419,
       10079.86899098,  9527.4017113 , 10323.2834034 ,  9269.698239  ,
        8174.69678444, 10616.58083442,  8743.66370719,  7209.22489424,
        7847.26975825,  8747.91121417,  9781.53808943, 10260.4486203 ,
        7925.32703754, 10187.50685027,  4959.12317166,  8893.64244815,
        9722.39120759, 10250.28523132, 10250.36206792,  5912.56256295,
```

```
 6807.58831598,  9696.42747582,  9567.72838167,  5206.84300194,
10634.3715292 , 10556.43217805,  5999.05156088,  8131.04680241,
10633.13053344, 10603.33150892,  9375.79323009,  8253.42029703,
 9621.99222439, 10146.51674371, 10357.83931499,  9967.00754951,
 8771.07396787,  9620.54745456,  9977.38184751,  7777.47051447,
10520.11870767, 10240.92028123,  9721.01473511, 10188.15040931,
10324.27375793, 10349.61509189, 10541.09807142,  8741.26236454,
10243.01289328,  9887.14565488, 10065.29895276, 10132.38294069,
 9674.31474484,  8885.27709328, 10409.16272209,  6800.49736966,
 9117.14220826,  8864.28804571,  4840.78783722,  6300.16171102,
 6953.75162041, 10584.08252879, 10614.11269082, 10553.96978192,
 5804.94025697, 10221.87438241,  7326.66636302, 10325.42324143,
 7408.64869326, 10194.44686068, 10049.03849678, 10560.98131597,
 8561.3677542 ,  7002.24366144,  9735.12211999,  5746.03243235,
10133.21380035,  9154.14421372,  8101.18661858,  8973.15464568,
 6380.90009119, 10386.97446276,  9546.7269945 ,  9704.79454985,
 7370.37427528,  9203.56730794, 10350.60895518,  9298.59824267,
 9132.59958648, 10216.29186327,  9704.4407033 ,  7725.46131136,
10287.46667159,  9609.43361413, 10214.31349489,  9879.91785657,
 7406.28283552,  9403.64495102,  7031.26752406, 10306.11698001,
 5029.80565798,  9548.15539101,  9534.49112983,  8955.52632748,
 9337.90818294, 10026.51728349,  6718.22675615,  9679.48824761,
 8046.72553537,  8767.59579597, 10096.65316184,  9775.89475575,
10089.23188645,  9609.76334055, 10602.57044078,  9697.14354053,
 9745.26657969,  6596.4263745 ,  7553.46169797, 10246.65892842,
 9855.94030922,  6156.98155366,  5277.51949478, 10104.49039084,
 8660.57028716, 10332.35979763,  6195.48775038,  9494.48680977,
10410.11427034,  9528.85284008,  7712.5237104 ,  9668.73233268,
 9992.71217651,  7077.38641746,  8069.24557391,  9703.41609333,
10127.18251058,  8045.84754453, 10523.18229626,  9518.60318396,
10343.84782629,  5348.69279347,  7461.40351053,  9612.5431617 ,
 5438.37441051, 10162.86581681,  8982.87426257,  7854.07802564,
 9618.76245637, 10111.99943317,  6391.21095094,  9613.57830029,
10189.985113  ,  9799.75936831,  9687.10794281,  9659.78629905,
10162.29208696, 10064.49474248, 10086.16226562, 10539.35304828,
10233.25044593,  9061.65656757,  9617.05943216,  8137.16294265,
 9645.07703767,  7741.6714318 ,  5662.32693722, 10512.54814525,
10030.40533701,  7118.51975807,  6975.78482232, 10486.23349272,
10524.03417441,  9937.38057631, 10075.86556192,  9252.42552778,
10467.73081026,  7838.47608819, 10196.52378389,  7728.72341896,
 5505.94851073,  9635.83851457, 10297.36829864,  9748.29752091,
 4011.27222267,  9795.73101359, 10525.0830173 ,  7640.3285934 ,
```

```
 7336.43417344, 10200.95543901,  9152.59811595,  9834.11005597,
 5818.36746835,  9714.57400974, 10241.19807176, 10422.5660614 ,
10209.46715867,  5579.74594179,  5898.87336357,  7416.19197505,
 9719.87271397,  7075.23773519,  6931.16474141, 10401.71299323,
 6453.58999536,  8715.51600214, 10199.91621215, 10516.05238422,
 9831.90876508, 10135.61019646, 10333.0173839 , 10260.98865218,
 6011.69111458,  5220.39729696, 10384.7243347 , 10460.61757356,
 5937.8611916 ,  5903.89776229,  8830.14162146,  9727.70650583,
10714.09534551,  8716.28343859, 10654.13648518, 10545.90655668,
 6969.671378  ,  5211.67195028, 10623.12460075,  8958.70728017,
10522.2498154 ,  9723.90961557])
```

In [33]:
```
from sklearn.metrics import mean_squared_error
Ridge_Error=mean_squared_error(pred,y_test)
Ridge_Error
```

Out[33]: 529111.045536224

In [34]:
```
from sklearn.metrics import r2_score
r2_score(y_test,pred)
```

Out[34]: 0.8343797517106646

In [35]:
```
results=pd.DataFrame(columns=['price','predicted'])
results['price']=y_test
results['predicted']=pred
results=results.reset_index()
results['ID']=results.index
results.head(10)
```

Out[35]:

	index	price	predicted	ID
0	676	10250	9912.601754	0
1	215	9790	10141.748493	1
2	146	5500	4775.235521	2
3	1319	9900	9870.926966	3
4	1041	8900	9630.417885	4
5	1425	9500	8697.092014	5
6	409	10450	10265.822884	6
7	617	9790	10293.851867	7
8	1526	9300	8614.349738	8
9	1010	4600	5749.673567	9

In [40]:
```
results['actual price']=results.apply(lambda column:column.price-column.predicted,axis=1)
results
```

Out[40]:

	index	price	predicted	ID	actual price
0	676	10250	9912.601754	0	337.398246
1	215	9790	10141.748493	1	-351.748493
2	146	5500	4775.235521	2	724.764479
3	1319	9900	9870.926966	3	29.073034
4	1041	8900	9630.417885	4	-730.417885
...	...	...	...	...	...
357	757	6000	5211.671950	357	788.328050
358	167	10950	10623.124601	358	326.875399
359	156	8000	8958.707280	359	-958.707280
360	1145	10700	10522.249815	360	177.750185
361	1393	9400	9723.909616	361	-323.909616

362 rows × 5 columns

In [36]:
```
list(results)
```

Out[36]:
```
['index', 'price', 'predicted', 'ID']
```

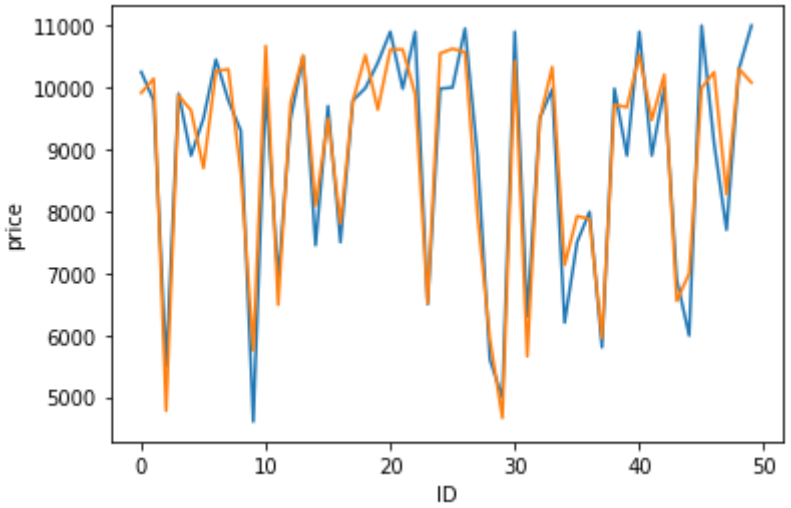
In [37]:
```
list(k)
```

Out[37]:
```
['ID',
 'model',
 'engine_power',
 'age_in_days',
 'km',
 'previous_owners',
 'lat',
 'lon',
 'price']
```

plot graph

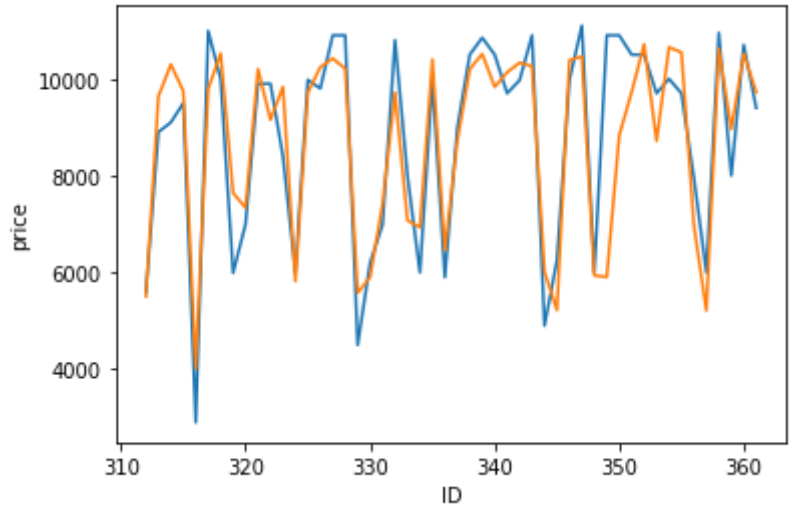
In [38]:
```
sns.lineplot(x='ID',y='price',data=results.head(50))
sns.lineplot(x='ID',y='predicted',data=results.head(50))
plt.plot()
```

Out[38]:
```
[]
```



In [39]:
```
sns.lineplot(x='ID',y='price',data=results.tail(50))
sns.lineplot(x='ID',y='predicted',data=results.tail(50))
plt.plot()
```

Out[39]: []



In [ ]: