In [35]:
```
import warnings
import pandas as r
dt=r.read_csv("/home/placemnet/YUVA/fiat500.csv")
```

In [36]:
```
dt.describe()
```

Out[36]:

	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 [37]:
```
d1=dt.drop(['lat','lon','ID'],axis=1)
d1.describe()
```

Out[37]:

	engine_power	age_in_days	km	previous_owners	price
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	51.904421	1650.980494	53396.011704	1.123537	8576.003901
std	3.988023	1289.522278	40046.830723	0.416423	1939.958641
min	51.000000	366.000000	1232.000000	1.000000	2500.000000
25%	51.000000	670.000000	20006.250000	1.000000	7122.500000
50%	51.000000	1035.000000	39031.000000	1.000000	9000.000000
75%	51.000000	2616.000000	79667.750000	1.000000	10000.000000
max	77.000000	4658.000000	235000.000000	4.000000	11100.000000

In [38]:
```
d1=r.get_dummies(d1)
d1.describe()
```

Out[38]:

	engine_power	age_in_days	km	previous_owners	price	model_lounge	model_pop	model_sport
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	51.904421	1650.980494	53396.011704	1.123537	8576.003901	0.711313	0.232770	0.055917
std	3.988023	1289.522278	40046.830723	0.416423	1939.958641	0.453299	0.422734	0.229836
min	51.000000	366.000000	1232.000000	1.000000	2500.000000	0.000000	0.000000	0.000000
25%	51.000000	670.000000	20006.250000	1.000000	7122.500000	0.000000	0.000000	0.000000
50%	51.000000	1035.000000	39031.000000	1.000000	9000.000000	1.000000	0.000000	0.000000
75%	51.000000	2616.000000	79667.750000	1.000000	10000.000000	1.000000	0.000000	0.000000
max	77.000000	4658.000000	235000.000000	4.000000	11100.000000	1.000000	1.000000	1.000000

In [39]:
```
d1.shape
```

Out[39]: (1538, 8)

In [40]:
```
a=d1['price']
b=d1.drop('price',axis=1)
b
```

Out[40]:

	engine_power	age_in_days	km	previous_owners	model_lounge	model_pop	model_sport
0	51	882	25000	1	1	0	0
1	51	1186	32500	1	0	1	0
2	74	4658	142228	1	0	0	1
3	51	2739	160000	1	1	0	0
4	73	3074	106880	1	0	1	0
...	...	...	...	...	...	...	...
1533	51	3712	115280	1	0	0	1
1534	74	3835	112000	1	1	0	0
1535	51	2223	60457	1	0	1	0
1536	51	2557	80750	1	1	0	0
1537	51	1766	54276	1	0	1	0

1538 rows × 7 columns

In [41]:
```
a
```

Out[41]:
```
0       8900
1       8800
2       4200
3       6000
4       5700
        ...
1533    5200
1534    4600
1535    7500
1536    5990
1537    7900
Name: price, Length: 1538, dtype: int64
```

In [42]:
```
#pip install -U scikit-learn
```

In [43]:
```
from sklearn.model_selection import train_test_split
a_train,a_test,b_train,b_test=train_test_split(a,b,test_size=0.33,random_state=42)
```

In [44]:
```
b_test.head()
```

Out[44]:

	engine_power	age_in_days	km	previous_owners	model_lounge	model_pop	model_sport
481	51	3197	120000	2	0	1	0
76	62	2101	103000	1	0	1	0
1502	51	670	32473	1	1	0	0
669	51	913	29000	1	1	0	0
1409	51	762	18800	1	1	0	0

Linear RegreSsioN

In [45]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()   #creating object of linear regression
reg.fit(b_train,a_train)   #training and fitting LR object using training data
```

Out[45]:
```
▾ LinearRegression

LinearRegression()
```

In [46]:
```
apred=reg.predict(b_test)
apred
```

Out[46]:
```
array([ 5867.6503378 ,  7133.70142341,  9866.35776216,  9723.28874535,
       10039.59101162,  9654.07582608,  9673.14563045, 10118.70728123,
        9903.85952664,  9351.55828437, 10434.34963575,  7732.26255693,
        7698.67240131,  6565.95240435,  9662.90103518, 10373.20344286,
        9599.94844451,  7699.34400418,  4941.33017994, 10455.2719478 ,
       10370.51555682, 10391.60424404,  7529.06622456,  9952.37340054,
        7006.13845729,  9000.1780961 ,  4798.36770637,  6953.10376491,
        7810.39767825,  9623.80497535,  7333.52158317,  5229.18705519,
        5398.21541073,  5157.65652129,  8948.63632836,  5666.62365159,
        9822.1231461 ,  8258.46551788,  6279.2040404 ,  8457.38443276,
        9773.86444066,  6767.04074749,  9182.99904787, 10210.05195479,
        8694.90545226, 10328.43369248,  9069.05761443,  8866.7826029 ,
        7058.39787506,  9073.33877162,  9412.68162121, 10293.69451263,
       10072.49011135,  6748.5794244 ,  9785.95841801,  9354.09969973,
        9507.9444386 , 10443.01608254,  9795.31884316,  7197.84932877,
       10108.31707235,  7009.6597206 ,  9853.90699412,  7146.87414965,
        6417.69133992,  9996.97382441,  9781.18795953,  8515.83255277,
        8456.30006203,  6499.76668237,  7768.57829985,  6832.86406122,
        8347.96113362, 10439.02404036,  7356.43463051,  8562.56562053,
        9820.78555199, 10035.83571539,  7370.77198022,  9411.45894006,
       10352.85155564,  8045.21588007, 10446.80664758,  3736.20118868,
       10348.63930496, 10435.96627494,  6167.80169017, 10390.11317804,
        6527.69471073,  9116.4755691 , 10484.52829   ,  9335.69889855,
        6709.57413543,  3390.72353093, 10106.33753331,  9792.46732008,
        6239.49568346,  4996.26346266,  9044.38667681,  9868.09959448,
        5484.13199252,  5698.5954821 , 10086.86206874,  8115.81693479,
       10392.37800936,  6835.6573351 ,  6657.61744836,  5738.50576764,
        8896.80120764,  9952.37340054, 10390.28377419,  9419.10788866,
        9082.56591129, 10122.82465116, 10410.00504522, 10151.77663915,
        9714.85367238,  9291.92963633, 10346.99073888,  5384.22311343,
        9772.85146492,  6069.77107828,  9023.26394782, 10220.56195956,
        9238.89392583,  9931.47195375,  8321.42715662,  8377.80491069,
        7528.53327408, 10552.64805598, 10465.02437243, 10110.68940664,
       10238.17869436,  6841.77264488,  9625.64505547, 10412.59988875,
        9653.06224923,  7948.63618724,  9704.82523573,  7971.05970955,
       10399.51752022,  9176.43567301,  5803.03205787,  6698.19524313,
        8257.83550573, 10452.95284574,  9948.66454584,  9789.65062843,
```

```
10582.50828537,  7568.91955482,  6804.97705225,  8065.01292384,
10310.29143419,  8836.34894739,  8390.05091229,  9582.13932508,
 9745.34784981, 10045.45021387, 10294.09872915,  7145.15315349,
 9727.85493167,  6281.78952194,  7901.36245623,  9387.9203723 ,
 5039.55649797,  9351.49777725,  9980.70844784, 10094.79341516,
 6359.24321991,  9856.10227211,  9099.07023804,  5234.05388382,
 5534.45288323,  4495.02309231, 10199.78432943, 10024.87037067,
 5465.58034188,  8520.72057674,  7034.71038647, 10054.65061446,
10191.12067767,  6008.34860428,  9748.18097947,  9669.4333196 ,
 9145.3756075 ,  9175.66562699, 10087.86753845,  9825.02990067,
 7340.29803785,  5083.8487301 ,  9441.50914802, 10243.05490667,
 5556.42300245, 10676.01945733,  6126.99295838,  9845.16661356,
 9850.77978959,  7840.83596305,  6552.05146566,  9938.82104889,
 8327.79232274,  9119.62204137,  6111.83787367, 10410.00504522,
 6360.97695249,  8601.59209793,  8377.80258216,  9803.81343895,
 8285.09831762, 10091.75635129, 10003.86694939, 10028.60283146,
10354.61956534,  8552.21002673,  6726.65446676,  9381.22662706,
 6520.9999373 , 10352.85155564,  9063.7534579 , 10456.89121831,
 9127.72470241,  9952.37340054,  8376.6975881 ,  9220.36267675,
10036.24981328,  8418.65456209,  4717.7579531 , 10076.86950203,
10017.8490121 , 10590.33289679, 10161.75393066,  4927.49556508,
 7276.18410037,  9678.26477249,  9764.65653403,  5643.53722047,
10062.84554534,  5163.04602382,  8307.60791348,  7441.80993846,
 7868.82460983,  9725.36143983,  8669.20982667, 10447.15719448,
 7124.58453563,  9718.32989102,  8059.66615638,  7430.65975056,
10425.57075395, 10364.18738085,  5433.2724385 ,  9102.40298437,
 9629.06913727, 10532.3506032 , 10129.42684118,  9149.48843328,
 6158.13422239,  9721.03634157, 10419.02236947,  8838.50241314,
 8182.78836676, 10012.21373766,  9468.92324529,  9904.31954667,
10475.66003551, 10475.0702782 ,  9609.27020577,  8115.22501265,
10439.02404036, 10363.81936482,  8720.0683498 ,  8274.3579289 ,
 6889.7195761 , 10191.45963957,  4819.0674709 ,  8814.11814085,
 5737.62378403, 10051.06593609,  8840.87520652, 10054.31165256,
 9686.269121  , 10463.56977746, 10133.15815395,  9762.80613855,
 9793.03056946,  6796.69068198,  9599.3262671 ,  8488.31539047,
 6705.66818403, 10307.58651641, 10045.18332239, 10120.36242166,
 5836.93199112,  8772.49782933,  9680.77538859,  5719.87463854,
 8398.59735084,  9680.77538859,  4334.81943405, 10015.00600846,
 9850.72458719,  7864.73798641, 10072.71245374, 10552.64805598,
10253.47474908,  6861.80736606,  6484.22649656, 10374.62123623,
 8426.37409382,  5447.47569851,  9914.20077691,  4687.39013431,
```

```
 7885.32100747,  5431.00822998,  9911.86294348, 10390.16991322,
 9680.84745901,  8844.57815539,  7764.08471024,  4257.54640953,
 9882.76503303, 10341.35258769,  5736.4484335 , 10179.87154436,
 9501.423448  ,  7997.3181334 ,  5532.33458288,  9894.57834738,
10437.97459358,  6381.35845844,  9591.23555726,  9574.27908517,
10322.30715736,  9501.22785499,  9789.955758  ,  9593.26549752,
 6775.82788536,  7915.34831306, 10389.98590521, 10351.58343315,
 7381.32686464,  9966.53983093, 10430.87188433, 10554.43156462,
10285.85574963, 10035.88086558,  9526.63034431,  7742.78157141,
 9297.64938364, 10051.42272678, 10004.81256571,  9985.84167026,
 9374.6573594 ,  9561.57499854,  9754.94184269,  9819.85893758,
 8780.31447831,  6255.99008069,  6281.53627686,  8190.88781577,
 8588.91394592,  6566.97963218,  6850.70237466,  5511.29438169,
 8119.97866315,  9847.74830838,  7775.93862032,  9875.05509733,
10121.29366536,  5791.92464084,  9835.42728501, 10043.91426822,
 8027.28015259,  4527.22080416, 10609.02444098,  3808.29240951,
 9952.37340054, 10511.20945172,  5746.34019592,  5486.40214756,
10395.91036208,  6788.47519216,  8953.20120295, 10442.24187982,
 9455.6934072 ,  9976.26574762,  8528.35753837,  7960.77147517,
10400.05054235,  5359.97362399,  9899.4913613 , 10203.35814213,
10303.33499967,  9507.16596227,  9151.43928526,  9805.06469343,
 5661.99787503,  4904.40690461,  4742.8827765 ,  9663.32864144,
 6102.95247322,  9870.62050425, 10066.06916341,  5001.24291171,
 8029.35471733,  9773.79143856,  5962.75261232, 10401.02638592,
 5511.44251977,  9627.19072277, 10106.26833963, 10199.67798189,
 9458.07047019,  4890.1778697 ,  5833.90060934,  7022.25799652,
10011.26407146, 10402.02002918,  9945.08219601,  7770.52280413,
 8840.08397206,  9916.27565791, 10287.45603992,  9964.3213269 ,
 8403.51255128,  9345.81907605,  8521.46225147,  9743.68712672,
 9791.34520178,  9779.16293972,  6753.27416058,  7354.16762745,
 8760.24542762,  9923.66596418,  9812.92276721, 10466.90125415,
 8163.46726237,  6659.46839415,  9987.65677522,  8866.7826029 ,
 9952.37340054, 10187.72427693, 10231.39378767, 10091.11325493,
 9365.98570732, 10009.10088406,  9141.00566394, 10099.11667176,
 7803.77049829,  6009.84398185,  8800.33824151, 10237.60733785,
 5609.98366311, 10097.61555355,  9684.99946572,  7644.67379732,
 9276.37891542,  7371.5492091 , 10287.98873148, 10067.26428381,
10552.64805598,  9966.72383894, 10068.46126756,  6232.53552963,
10584.55044373,  9965.98687522, 10529.44404458,  9602.67646085,
 9665.77720284,  6186.06948587,  8073.87436253, 10345.58323918,
 6344.74803956,  7361.62678204, 10058.57116223,  6792.219309  ,
```

```
       7897.72464823,  5261.45936067,  4540.24137423,  8709.36468047,
       6882.0117409 ,  7406.73353952,  6795.61189392,  7047.27998963,
       9945.33400083,  8856.93910595,  9378.02074127, 10389.561154  ,
      10092.46332921, 10381.52000388,  9723.92466625,  5996.3331428 ,
       9786.14866981,  7708.49649098,  5583.48163469,  4932.92788329,
       9856.66053994,  9236.22981005, 10092.64052142,  6256.43516278,
       8592.63841379, 10341.5365957 ,  5177.96595576, 10032.66513491,
       6281.53627686,  9986.327508  ,  8381.51701951, 10371.142553131)
```

In [47]:
```
from sklearn.metrics import r2_score
r2_score(a_test,apred)     #to check the efficiency
```

Out[47]:
```
0.8415526986865394
```

In [48]:
```
from sklearn.metrics import mean_squared_error   #calculating MSE
mean_squared_error(apred,a_test)
```

Out[48]:
```
581887.727391353
```

In [49]:
```
print(mean_squared_error(apred,a_test)**(1/2))
```

```
762.8156575420782
```

In [50]:
```
results=r.DataFrame(columns=['Price','Predicted'])
results['Price']=a_test
results['Predicted']=apred
results.head()
results=results.reset_index()
results['Id']=results.index
```

In [51]:
```
results['Difference']=results.apply(lambda row:row.Price-row.Predicted,axis=1)
results
```

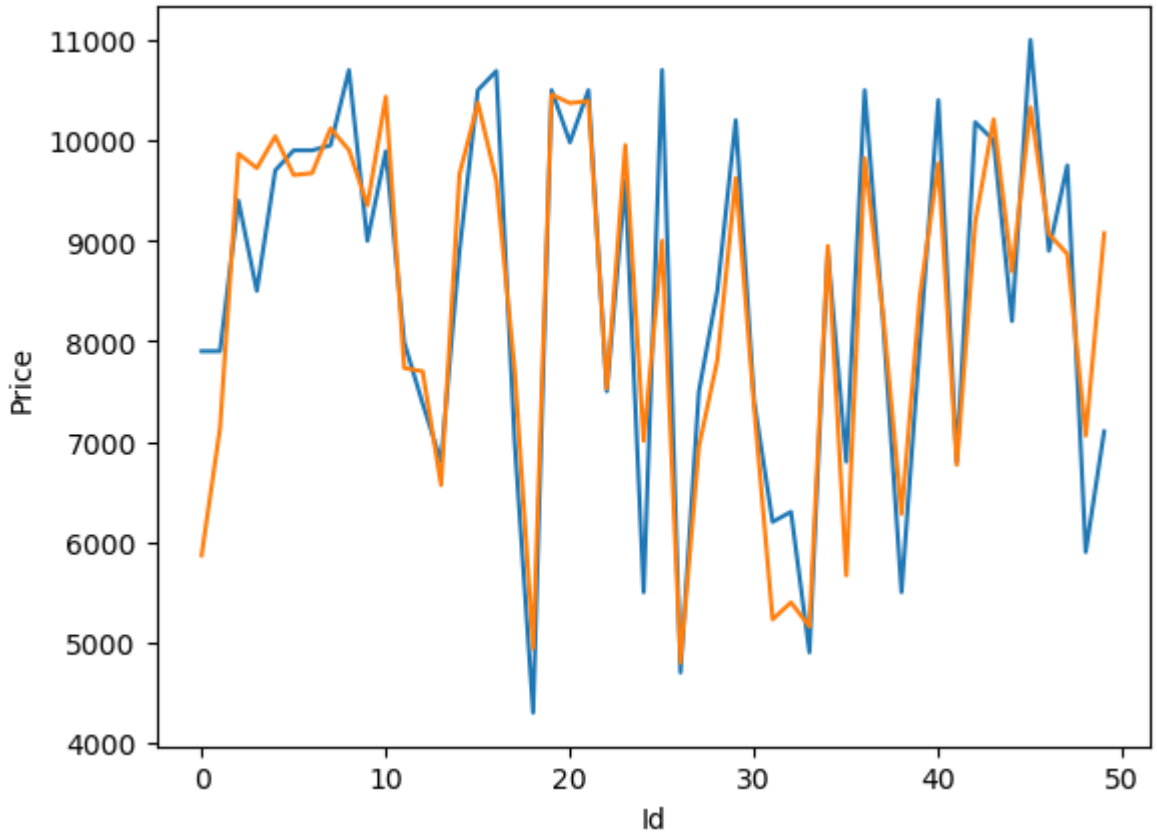
Out[51]:

	index	Price	Predicted	Id	Difference
0	481	7900	5867.650338	0	2032.349662
1	76	7900	7133.701423	1	766.298577
2	1502	9400	9866.357762	2	-466.357762
3	669	8500	9723.288745	3	-1223.288745
4	1409	9700	10039.591012	4	-339.591012
...	...	...	...	...	...
503	291	10900	10032.665135	503	867.334865
504	596	5699	6281.536277	504	-582.536277
505	1489	9500	9986.327508	505	-486.327508
506	1436	6990	8381.517020	506	-1391.517020
507	575	10900	10371.142553	507	528.857447

508 rows × 5 columns

In [52]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.lineplot(x='Id',y='Price',data=results.head(50))
sns.lineplot(x='Id',y='Predicted',data=results.head(50))
plt.plot()
```

Out[52]: []



Ridge Regression

In [53]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge                #Ridge regression
import warnings
warnings.filterwarnings("ignore")
```

In [54]:
```
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(b_train, a_train)
```

Out[54]:

▾ **GridSearchCV**

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

▾ **estimator: Ridge**

Ridge()

▾ Ridge

Ridge()

In [55]:
```
ridge_regressor.best_params_
```

Out[55]:
```
{'alpha': 30}
```

In [56]:
```
ridge=Ridge(alpha=30)
ridge.fit(b_train,a_train)
a_pred_ridge=ridge.predict(b_test)
```

In [57]:
```
from sklearn.metrics import mean_squared_error
Ridge_error=mean_squared_error(a_pred_ridge,a_test)
Ridge_error
```

Out[57]:
```
579521.7970897449
```

In [58]:
```
from sklearn.metrics import r2_score    #to check the efficiency
r2_score(a_test,a_pred_ridge)
```

Out[58]:
```
0.8421969385523054
```

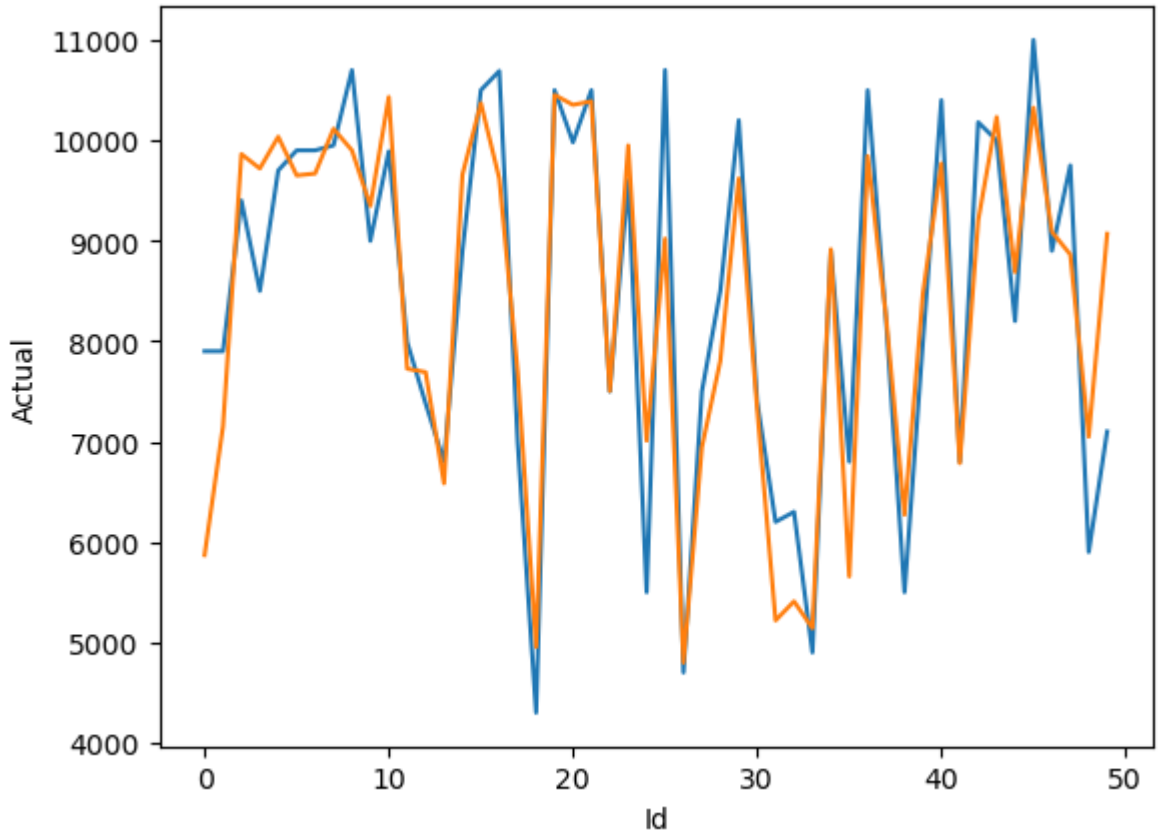
In [59]:
```
results=r.DataFrame(columns=['Actual','Predicted'])   #to compare the actual and pedicted price
results['Actual']=a_test
results['Predicted']=a_pred_ridge
results=results.reset_index()
results['Id']=results.index
results.head()
```

Out[59]:

	index	Actual	Predicted	Id
0	481	7900	5869.741155	0
1	76	7900	7149.563327	1
2	1502	9400	9862.785355	2
3	669	8500	9719.283532	3
4	1409	9700	10035.895686	4

In [60]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.lineplot(x='Id',y='Actual',data=results.head(50))
sns.lineplot(x='Id',y='Predicted',data=results.head(50))
plt.plot()
```

Out[60]: []



Elastic Regression

In [61]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
import warnings
warnings.filterwarnings("ignore")
elastic = ElasticNet()
parameters = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20]}
elastic_regressor = GridSearchCV(elastic, parameters)
elastic_regressor.fit(b_train, a_train)
```

Out[61]:

▾ **GridSearchCV**

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

▾ **estimator: ElasticNet**

```
ElasticNet()
```

▾ ElasticNet

```
ElasticNet()
```

In [62]:
```
elastic_regressor.best_params_
```

Out[62]:
```
{'alpha': 0.01}
```

In [63]:
```
elastic=ElasticNet(alpha=.01)
elastic.fit(b_train,a_train)
a_pred_elastic=elastic.predict(b_test)
```

In [64]:
```
from sklearn.metrics import r2_score    #to check the efficiency
r2_score(a_test,a_pred_elastic)
```

Out[64]:
```
0.841688021120299
```

In [65]:
```
from sklearn.metrics import mean_squared_error
Elastic_error=mean_squared_error(a_pred_elastic,a_test)
Elastic_error
```

Out[65]:
```
581390.7642825295
```

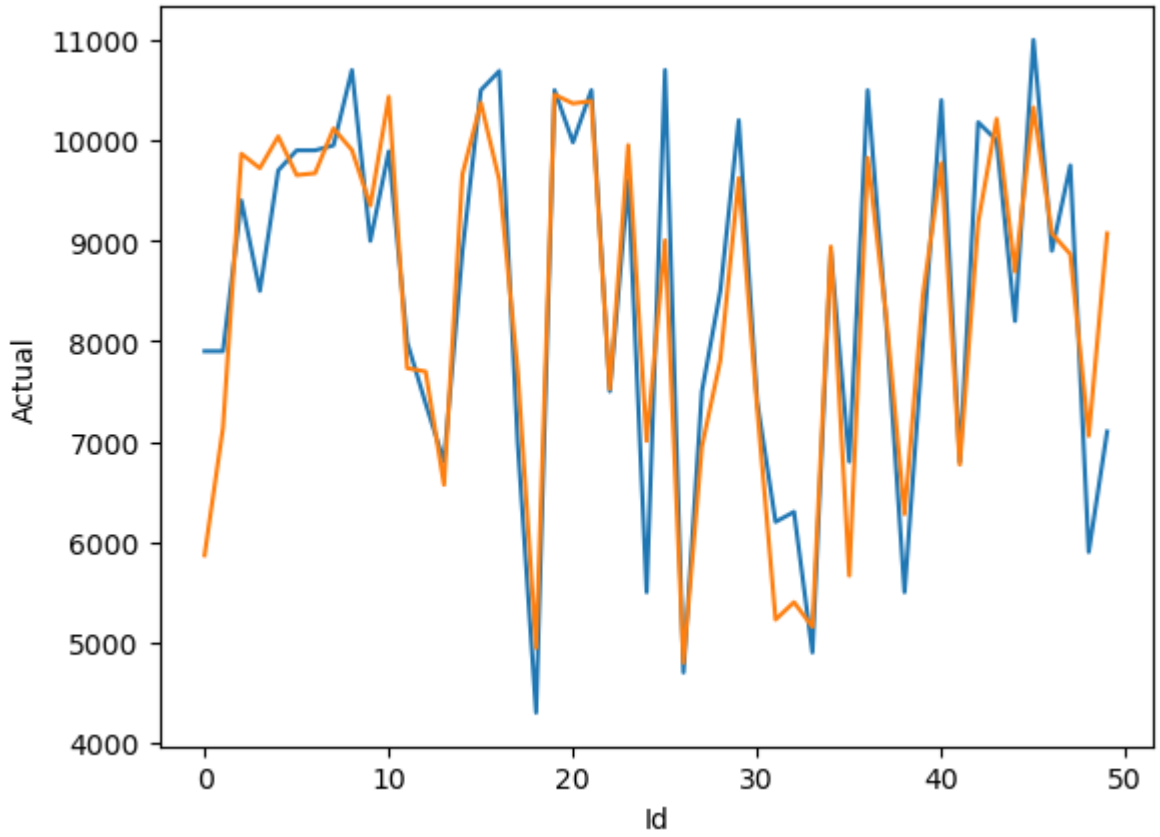
In [66]:
```
results=r.DataFrame(columns=['Actual','Predicted'])   #to compare the actual and pedicted price
results['Actual']=a_test
results['Predicted']=a_pred_elastic
results=results.reset_index()
results['Id']=results.index
results.head()
```

Out[66]:

	index	Actual	Predicted	Id
0	481	7900	5867.742075	0
1	76	7900	7136.527402	1
2	1502	9400	9865.726723	2
3	669	8500	9722.573593	3
4	1409	9700	10038.936496	4

In [67]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.lineplot(x='Id',y='Actual',data=results.head(50))
sns.lineplot(x='Id',y='Predicted',data=results.head(50))
plt.plot()
```

Out[67]: []



In [ ]: