In [1]:
```python
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
#Seaborn is a Python data visualization library
```

In [2]:
```python
df=pd.read_csv("carIndia.csv")
```

In [3]:
```python
df
```

Out[3]:

	model_year	maker	model_name	city	distance_covered (km)	fuel_type	pre_owner
0	2012	Maruti	Alto K10 VXI	Mumbai	29067	Petrol	2nd Owner
1	2011	Hyundai	i20 SPORTZ 1.2 O	Mumbai	36791	Petrol	2nd Owner
2	2010	Maruti	A Star VXI	Mumbai	35171	Petrol	1st Owner
3	2011	Hyundai	Santro Xing GLS	Mumbai	19908	Petrol	1st Owner
4	2012	Hyundai	Santro Xing GLS	Mumbai	43847	Petrol	3rd Owner
...	...	...	...	...	...	...	...
3360	2014	Honda	City S MT DIESEL	Kolkata	61643	Diesel	3rd Owner
3361	2006	Maruti	Wagon R LXI	Kolkata	26500	Petrol	1st Owner
3362	2016	Maruti	S Cross ZETA 1.3	Kolkata	57828	Diesel	1st Owner
3363	2012	BMW	3 Series 320D	Kolkata	23782	Diesel	2nd Owner
3364	2016	Hyundai	Creta 1.4 BASE	Kolkata	33130	Diesel	1st Owner

3365 rows × 8 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3365 entries, 0 to 3364
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   model_year             3365 non-null   int64 
 1   maker                  3365 non-null   object
 2   model_name             3365 non-null   object
 3   city                   3365 non-null   object
 4   distance_covered (km)  3365 non-null   int64 
 5   fuel_type              3365 non-null   object
 6   pre_owner              3365 non-null   object
 7   price (₹)              3365 non-null   int64 
dtypes: int64(3), object(5)
memory usage: 210.4+ KB
```

In [5]:
```
df['model_name'].unique()
```

```
       'Laura AMBIENTE 1.8 TSI', 'Swift LXI 1.3', 'Duster 85 PS RXL',
       'Polo COMFORTLINE 1.2L DIESEL', 'Tiago XZ 1.05 REVOTORQ',
       'Amaze 1.2 SAT I VTEC', 'Beat LS DIESEL', 'Etios Liva D 4D G
D',
       'Creta 1.6 S', 'Rexton RX7', 'i20 SPORTZ 1.4 CRDI',
       'City 1.5 E MT PETROL', 'Etios GD', 'City V CVT',
       'Innova Crysta Touring Sport Diesel MT',
       'Elite i20 MAGNA 1.4 CRDI', 'Santro Xing GL',
       'Creta 1.6 SX (O) CRDI', 'Superb ELEGANCE 1.8 TSI AT',
       'Corolla H2 1.8 E', 'Ecosport 1.5 AMBIENTE TDCI',
       'VENUE SX(O) CRDi', 'Creta 1.6 SX AT PETROL',
       'Duster RXZ AMT 110 PS', 'Vento HIGHLINE PETROL AT',
       'S Cross ALPHA 1.3', 'City S MT DIESEL',
       'Grand i10 MAGNA 1.1 CRDI', 'Jetta TRENDLINE 1.4 TSI MT',
       'Duster 85 PS RXL OPT', 'S Cross ALPHA SHVS',
       'Punto EVO MULTIJET 1.3 90 HP', 'i10 ERA', 'Dzire VDI AMT',
       'Innova 2.5 GX 7 STR BS IV', 'Etios Liva GD exclusive',
       'Benz E Class E 220 CDI ELEGANCE', 'Swift Dzire VXI AT',
       'Cruze LTZ', 'Camry HYBRID', 'Vitara Brezza ZDI PLUS',
       'Etios Liva D 4D GD SP', 'Jetta HIGHLINE TDI AT',
       'Verna FLUIDIC 1.6 EX CRDI', 'Ciaz DELTA 1.3 DDIS SHVS',
```

In [6]:
```
df.describe()
```

Out[6]:

	model_year	distance_covered (km)	price (₹)
count	3365.000000	3365.000000	3.365000e+03
mean	2013.876374	60937.813967	4.336549e+05
std	3.035588	41342.775191	2.595909e+05
min	2001.000000	60.000000	2.700000e+04
25%	2012.000000	30598.000000	2.769990e+05
50%	2014.000000	53488.000000	3.647990e+05
75%	2016.000000	82414.000000	4.975990e+05
max	2021.000000	428123.000000	3.600000e+06

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
model_year               0
maker                    0
model_name               0
city                     0
distance_covered (km)    0
fuel_type                0
pre_owner                0
price (₹)                0
dtype: int64
```

In [8]:
```
df.shape
```

Out[8]:
```
(3365, 8)
```

In [9]:
```
df.groupby(['model_name']).count()
```

Out[9]:

model_name	model_year	maker	city	distance_covered (km)	fuel_type	pre_owner	price (₹)
3 Series 320 D PERFORMANCE EDITION	1	1	1	1	1	1	1
3 Series 320D	2	2	2	2	2	2	2
3 Series 320D LUXURYLINE	2	2	2	2	2	2	2
3 Series 320D SPORTLINE	1	1	1	1	1	1	1
5 Series 520D 2.0	3	3	3	3	3	3	3
...	...	...	...	...	...	...	...
i20 Magna O 1.4 CRDI	1	1	1	1	1	1	1
i20 SPORTZ 1.2 O	21	21	21	21	21	21	21
i20 SPORTZ 1.2 VTVT	15	15	15	15	15	15	15
i20 SPORTZ 1.4 CRDI	8	8	8	8	8	8	8
i20 SPORTZ O 1.4 CRDI	5	5	5	5	5	5	5

677 rows × 7 columns

In [10]:
```
#df1=df.drop(['maker','city'],axis=1)
```

In [11]:
```
#df1
```

Getting the maximum number of cars

SPLITTING

In [12]:

```
def split_model(model):
    parts = model.split(' ')
    Model = ' '.join(parts[:1])
    suffix = parts[-1]
    return Model, suffix
df[['Model', 'Suffix']] = df['model_name'].apply(lambda x: pd.Series(split
#split_model is a custom function defined to split a string (representing
#model.split(' ') splits the input string model into a list of substrings
#Model = ' '.join(parts[:1]): This line attempts to join the first part of
#suffix = parts[-1] retrieves the last part of the split string as the 'Su
#Finally, the function returns a tuple containing 'Model' and 'Suffix'.
#lambda is a keyword used to create anonymous functions, which are functio
```

In [13]:

```
df[['Model', 'Suffix']]
```

Out[13]:

	Model	Suffix
0	Alto	VXI
1	i20	O
2	A	VXI
3	Santro	GLS
4	Santro	GLS
...	...	...
3360	City	DIESEL
3361	Wagon	LXI
3362	S	1.3
3363	3	320D
3364	Creta	BASE

3365 rows × 2 columns

In [ ]:

In [14]:

```
df[['Model', 'Suffix']].isnull().sum()
```

Out[14]:

```
Model     0
Suffix    0
dtype: int64
```

In [15]:

```python
# Grouping by 'Model' and 'Suffix' and getting the count of occurrences
result = df[['Model', 'Suffix']].groupby(['Model']).size().reset_index(nar

# Sorting the result by 'Count' in ascending order
sorted_result = result.sort_values('Count', ascending=False)

print(sorted_result)
```

```
     Model  Count
98   Swift    465
9     Alto    393
110  Wagon    234
121    i10    209
50   Grand    133
..     ...    ...
86     S60      1
90    Sail      1
33     E2O      1
93   Scala      1
61   Laura      1

[123 rows x 2 columns]
```

extracting the maximum and predicting

Swift cars(max)

In [16]:

```python
df_Swift=df.loc[(df.Model)=='Swift']
df_Swift.isnull().sum()
```

Out[16]:

```
model_year               0
maker                    0
model_name               0
city                     0
distance_covered (km)    0
fuel_type                0
pre_owner                0
price (₹)                0
Model                    0
Suffix                   0
dtype: int64
```

In [17]:
```
df_Swift=df_Swift.drop(['maker','city','Suffix','model_name'],axis=1)
df_Swift
```

Out[17]:

	model_year	distance_covered (km)	fuel_type	pre_owner	price (₹)	Model
10	2009	42533	Petrol	2nd Owner	274899	Swift
12	2015	23070	Petrol	2nd Owner	478799	Swift
28	2012	50581	Petrol	1st Owner	406299	Swift
36	2012	47260	Petrol	1st Owner	364799	Swift
37	2012	50525	Diesel	1st Owner	376799	Swift
...	...	...	...	...	...	...
3299	2015	61537	Diesel	1st Owner	450000	Swift
3322	2017	18140	Petrol	1st Owner	457699	Swift
3345	2013	78261	Petrol	1st Owner	327699	Swift
3348	2007	90265	Diesel	3rd Owner	130000	Swift
3354	2017	43860	Petrol	1st Owner	429999	Swift

465 rows × 6 columns

In [18]:
```
df_Alto=df.loc[(df.Model)=='Alto']
df_Alto.isnull().sum()
df_Alto=df_Alto.drop(['maker','city','Suffix','model_name'],axis=1)
df_Alto
```

Out[18]:

	model_year	distance_covered (km)	fuel_type	pre_owner	price (₹)	Model
0	2012	29067	Petrol	2nd Owner	165199	Alto
6	2010	50742	Petrol	2nd Owner	170399	Alto
7	2015	12657	Petrol	1st Owner	282299	Alto
17	2010	34995	Petrol	1st Owner	165999	Alto
34	2015	8322	Petrol	1st Owner	286399	Alto
...	...	...	...	...	...	...
3319	2006	58542	Petrol + CNG	2nd Owner	85000	Alto
3326	2017	11705	Petrol	1st Owner	283199	Alto
3332	2019	1306	Petrol	1st Owner	390999	Alto
3333	2018	3697	Petrol	1st Owner	316999	Alto
3336	2019	2000	Petrol	1st Owner	381299	Alto

393 rows × 6 columns

In [19]:
```
df_Wagon=df.loc[(df.Model)=='Wagon']
df_Wagon.isnull().sum()
df_Wagon=df_Wagon.drop(['maker','city','Suffix','model_name'],axis=1)
df_Wagon
```

Out[19]:

	model_year	distance_covered (km)	fuel_type	pre_owner	price (₹)	Model
8	2013	13688	Petrol	1st Owner	326199	Wagon
14	2011	18514	Petrol	1st Owner	269399	Wagon
16	2012	20712	Petrol	2nd Owner	258399	Wagon
19	2012	39652	Petrol	3rd Owner	288299	Wagon
21	2014	6858	Petrol	1st Owner	358399	Wagon
...	...	...	...	...	...	...
3314	2014	95432	Petrol + CNG	1st Owner	270000	Wagon
3327	2013	22008	Petrol	1st Owner	332599	Wagon
3331	2014	25852	Petrol	1st Owner	345499	Wagon
3357	2001	72000	Petrol	2nd Owner	38000	Wagon
3361	2006	26500	Petrol	1st Owner	100000	Wagon

234 rows × 6 columns

In [20]:
```
combined_df = pd.concat([df_Swift,df_Alto,df_Wagon ], axis=0)
```

In [21]:
```
combined_df
```

Out[21]:

	model_year	distance_covered (km)	fuel_type	pre_owner	price (₹)	Model
10	2009	42533	Petrol	2nd Owner	274899	Swift
12	2015	23070	Petrol	2nd Owner	478799	Swift
28	2012	50581	Petrol	1st Owner	406299	Swift
36	2012	47260	Petrol	1st Owner	364799	Swift
37	2012	50525	Diesel	1st Owner	376799	Swift
...	...	...	...	...	...	...
3314	2014	95432	Petrol + CNG	1st Owner	270000	Wagon
3327	2013	22008	Petrol	1st Owner	332599	Wagon
3331	2014	25852	Petrol	1st Owner	345499	Wagon
3357	2001	72000	Petrol	2nd Owner	38000	Wagon
3361	2006	26500	Petrol	1st Owner	100000	Wagon

1092 rows × 6 columns

In [ ]:

```

```

In [22]:

```
combined_df=pd.get_dummies(combined_df,dtype=int)
```

In [23]:

```
combined_df
```

Out[23]:

	model_year	distance_covered (km)	price (₹)	fuel_type_Diesel	fuel_type_Petrol	fuel_type_F +
10	2009	42533	274899	0	1	
12	2015	23070	478799	0	1	
28	2012	50581	406299	0	1	
36	2012	47260	364799	0	1	
37	2012	50525	376799	1	0	
...	...	...	...	...	...	
3314	2014	95432	270000	0	0	
3327	2013	22008	332599	0	1	
3331	2014	25852	345499	0	1	
3357	2001	72000	38000	0	1	
3361	2006	26500	100000	0	1	

1092 rows × 14 columns

In [24]:

```
cor_mat=combined_df.corr()
```

In [25]:

```
cor_mat
```

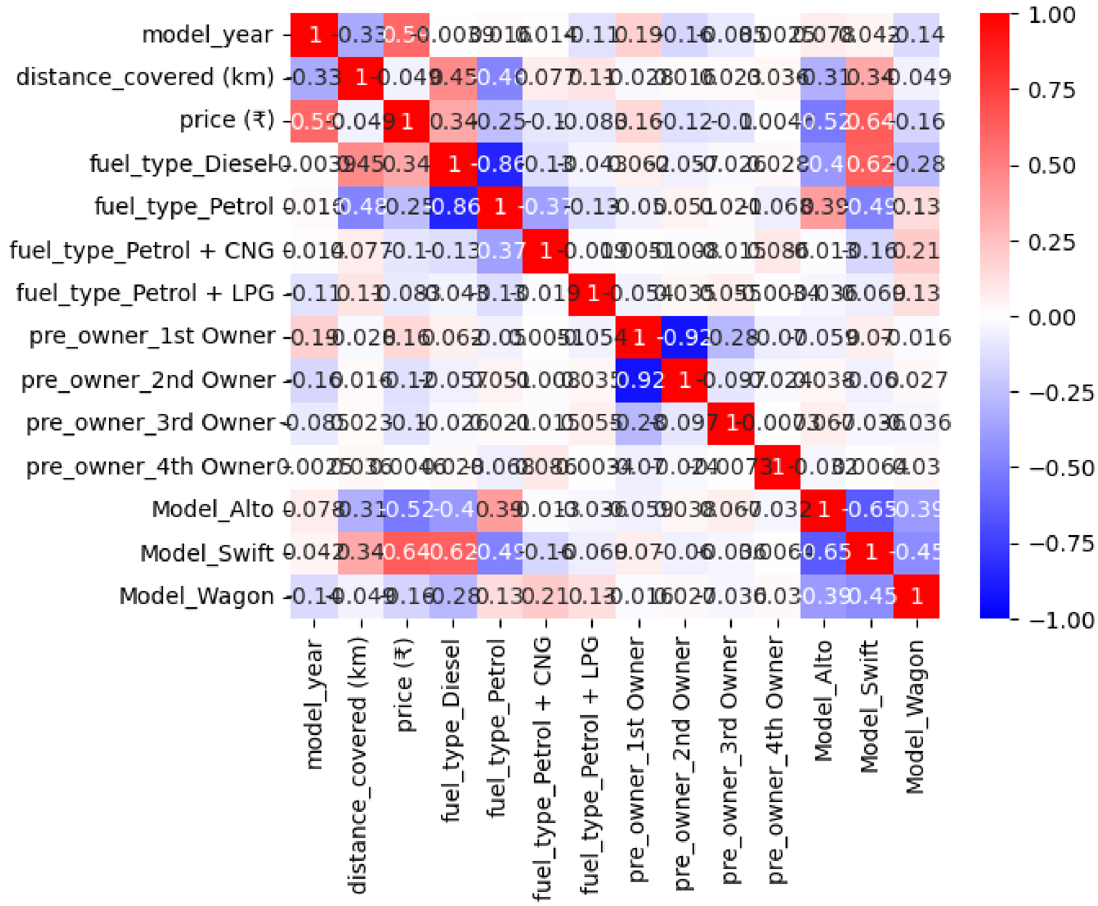
Out[25]:

	model_year	distance_covered (km)	price (₹)	fuel_type_Diesel	fuel_type_Petr
model_year	1.000000	-0.332116	0.589902	-0.003899	0.0164
distance_covered (km)	-0.332116	1.000000	-0.049384	0.454001	-0.4774
price (₹)	0.589902	-0.049384	1.000000	0.343490	-0.2534
fuel_type_Diesel	-0.003899	0.454001	0.343490	1.000000	-0.8558
fuel_type_Petrol	0.016400	-0.477410	-0.253481	-0.855815	1.0000
fuel_type_Petrol + CNG	0.014366	0.076816	-0.100712	-0.125882	-0.3744
fuel_type_Petrol + LPG	-0.112159	0.108164	-0.082885	-0.043086	-0.1281
pre_owner_1st Owner	0.185190	-0.027847	0.155933	0.062116	-0.0504
pre_owner_2nd Owner	-0.159739	0.016274	-0.122632	-0.057498	0.0509
pre_owner_3rd Owner	-0.085008	0.023381	-0.102872	-0.025504	0.0213
pre_owner_4th Owner	0.002483	0.036118	0.004611	0.028439	-0.0683
Model_Alto	0.077723	-0.305099	-0.517942	-0.402212	0.3851
Model_Swift	0.042038	0.336850	0.639359	0.622880	-0.4850
Model_Wagon	-0.141573	-0.049043	-0.164617	-0.280131	0.1339

In [26]:

```
#import seaborn as sns
sns.heatmap(cor_mat,vmax=1,vmin=-1,annot=True,linewidth=0,cmap='bwr')
#cmap is used for vizualization
```

Out[26]: <Axes: >



In [27]:

```
y=combined_df['price (₹)']
print(y)
```

```
10      274899
12      478799
28      406299
36      364799
37      376799
         ...
3314    270000
3327    332599
3331    345499
3357     38000
3361    100000
Name: price (₹), Length: 1092, dtype: int64
```

In [28]:

```
combined_df.shape
```

Out[28]: (1092, 14)

In [29]:

```
X=combined_df.drop('price (₹)',axis=1)
print(X)
```

```
      model_year  distance_covered (km)  fuel_type_Diesel  fuel_type_Pet
rol  \
10          2009                  42533                 0
1
12          2015                  23070                 0
1
28          2012                  50581                 0
1
36          2012                  47260                 0
1
37          2012                  50525                 1
0
...          ...                    ...               ...
...
3314        2014                  95432                 0
0
3327        2013                  22008                 0
1
3331        2014                  25852                 0
1
3357        2001                  72000                 0
1
3361        2006                  26500                 0
1

      fuel_type_Petrol + CNG  fuel_type_Petrol + LPG  pre_owner_1st Owne
r  \
10                         0                       0
0
12                         0                       0
0
28                         0                       0
1
36                         0                       0
1
37                         0                       0
1
...                      ...                     ...
...
3314                       1                       0
1
3327                       0                       0
1
3331                       0                       0
1
3357                       0                       0
0
3361                       0                       0
1

      pre_owner_2nd Owner  pre_owner_3rd Owner  pre_owner_4th Owner  \
10                      1                    0                    0
12                      1                    0                    0
28                      0                    0                    0
36                      0                    0                    0
37                      0                    0                    0
...                   ...                  ...                  ...
```

```
3314            0            0            0
3327            0            0            0
3331            0            0            0
3357            1            0            0
3361            0            0            0

      Model_Alto  Model_Swift  Model_Wagon
10             0            1            0
12             0            1            0
28             0            1            0
36             0            1            0
37             0            1            0
...          ...          ...          ...
3314           0            0            1
3327           0            0            1
3331           0            0            1
3357           0            0            1
3361           0            0            1

[1092 rows x 13 columns]
```

In [30]:

```
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_s
```

In [31]:

```
X_test.head(5)
```

Out[31]:

	model_year	distance_covered (km)	fuel_type_Diesel	fuel_type_Petrol	fuel_type_Petrol + CNG	fu
217	2011	82414	0	1	0	
736	2012	102746	1	0	0	
612	2009	74844	0	1	0	
3172	2019	61704	0	1	0	
2039	2015	53317	0	1	0	

In [32]:

```
X_train.shape
```

Out[32]: (731, 13)

In [33]:
```
y_train
```

Out[33]:
```
2510    247299
1908    391799
515     379699
508     399999
2195    352099
         ...
2070    393899
6       170399
693     189099
2518    490199
16      258399
Name: price (₹), Length: 731, dtype: int64
```

In [34]:
```
y_train.shape
```

Out[34]:
```
(731,)
```

In [35]:
```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(X_train,y_train)
LinearRegression()
```

Out[35]:
```
LinearRegression()
```

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 [36]:
```
ypred=reg.predict(X_test)
```

In [37]:

```
ypred
```

Out[37]:
```
array([195539.36996269, 371699.9915188 , 324974.83253337, 372023.6435042
8,
       288041.70348188, 267115.14946891, 526562.2076845 , 379910.7954368
4,
       250510.56655651, 198464.03027955, 265901.20588164, 241589.9378731
6,
       379730.50353812, 357774.47395833, 195335.10010635, 336269.4230663
1,
       472187.63673207, 196009.44182672, 199329.38984743, 208995.2907243
1,
       348176.17478424, 460394.52768942, 369432.14481308, 371844.0569791
3,
       398324.41459649, 277905.43543871, 310763.46081751, 500630.3430110
1,
       424435.56735088, 300887.34073452, 507442.97542729, 208183.1211062
4,
       218802.41753968, 413497.86382763, 411618.7129089 , 247149.7571234
7,
       299683.53460184, 341757.98877269, 398681.06002911, 287084.0577299
1,
       357405.31299671, 376696.96359129, 268588.27595057, 295892.6764566
2,
       245629.64636811, 411854.5530712 , 237688.99259116, 186357.5660186
9,
       445577.66637183, 373541.6367037 , 208895.04812662, 422545.1592638
7,
       331105.54251634, 203016.07317305, 293560.04711973, 230782.3258485
,
       376518.22812511, 207505.99738307, 271268.05708779, 150969.0283474
8,
       449291.58893806, 338091.49870601, 283910.4957249 , 297590.2093524
9,
       351753.9369653 , 376657.56760898, 178278.7259516 , 180049.7835105
4,
       401189.67283436, 419464.45781165, 489711.68262935, 312558.3524506
5,
       300088.38690739, 284227.2566974 , 231208.43031457, 201357.9867786
8,
       461503.33356599, 289624.09648616, 397252.02539693, 312172.9439679
8,
       359943.66987058, 444532.67637746, 446571.95996616, 249027.2112383
5,
       229846.04634757, 452927.93165959, 159571.37484411, 247009.0165174
2,
       256247.63326975, 169689.79059616, 455444.90736096, 406228.9240977
2,
       402489.92912628, 425914.53480088, 485813.42417437, 246997.6884208
4,
       428618.77865295, 264264.13742673, 446304.53816669, 487288.4223976
2,
       350726.15755045, 441968.52572034, 205931.76284677, 144227.3158642
7,
       352000.27348242, 192916.09392791, 298762.15239481, 379006.4580846
5,
       251177.50335263, 445969.59194408, 443778.12055088, 264959.1380422
6,
       397485.07308071, 246531.32185064, 373321.37575774, 443831.5953272
```

```
3,
       336000.62331974, 310494.80123378, 250868.29680867, 429342.7893866
2,
       433507.91409022, 164150.36167596, 434660.64782907, 394835.5196007
4,
       403798.93537169, 157649.26281208, 415673.70852392, 414246.3873519
3,
       307465.60732548, 378541.56377341, 200023.91984352, 141087.6942947
5,
       306693.92452379, 230922.45225444, 153496.91480777, 285409.1339258
5,
       336094.29065692, 321869.48007991, 245966.97066512, 313237.6521938
6,
       315473.11558623, 265633.78663628, 318815.14587458, 258877.632126
,
       349133.25853489, 460364.41810273, 249041.19930235, 405527.0523315
4,
       253520.44948988, 323073.17817434, 376921.2442832 , 271281.3632293
6,
       252884.96875146, 371288.96528591, 293358.09307985, 341218.5881279
4,
       380241.527817  , 511530.87310753, 364668.91661638, 222342.0495835
3,
       419377.97597047, 398641.2812205 , 270095.04550287, 357604.4275147
3,
       311354.637048  , 233736.41382962, 407769.54559939,  98262.6701804
1,
       395661.15787908, 316765.02629025, 391699.9017693 , 318812.3137731
2,
       169353.37552906, 259544.28768098, 508590.54588293, 270248.1284350
3,
       411066.63735847, 427877.56217891, 368476.08956157, 552274.0387195
1,
       309469.73165366, 465821.89081709, 346620.47557388, 363847.2205344
6,
       429169.12326761, 357265.18355215, 266332.07520799, 505632.4713668
5,
       370850.94377624, 443186.8938012 , 465196.39815734, 295499.7784625
7,
       354997.91311735, 183208.67552912, 406518.8966507 , 467293.0823323
,
       407446.76577497, 205369.28183891, 349165.64423499, 335635.4618589
3,
       210408.23400614, 555675.69512435, 217404.1015026 , 227458.7409081
6,
       155809.54108662, 404096.53837916, 385208.31851414, 417274.0754752
8,
       312757.80610828, 232182.06864027, 393613.34481183, 296837.3516887
7,
       439625.49775854, 361494.36083205, 212130.48845214, 369854.2547992
9,
       479903.13765605, 337553.81531535, 394906.26974539, 384890.4999185
4,
       320188.94217157, 273821.17467812,  80875.0839765 , 381430.3088568
9,
       326803.64349989, 288852.73656257, 285873.82997659, 444568.5261701
6,
```

```
        295692.29691274, 198324.91810316, 315082.21509865, 352864.8936087
9,
        463795.04755688, 226395.29152752, 318267.73936945, 204979.7128631
,
        410086.49072031, 291598.03462312, 309125.65992098, 385752.5605467
1,
        251280.615917  , 290017.91469362, 463782.38312691, 276117.7677775
5,
        349560.29749696, 483251.6041111 , 416321.8844509 , 250930.4034679
5,
        451104.36607328, 400460.01583071, 234516.93339521, 357306.0846285
5,
        199584.53448899, 439886.67405051, 441647.34025425, 223161.8426562
3,
        295884.16108032, 381151.89105947, 217148.52833141, 346657.3208667
9,
        414458.97736271, 315515.06246999, 184585.47692198, 440667.8248519
7,
        555073.4526161 , 381315.39313077, 239745.11038357, 343183.5396173
1,
        229281.51957241, 390880.42700659, 297670.44824027, 407786.4349354
1,
        453540.74941368, 398651.39822487, 377699.31777961, 334620.3066632
,
        437172.1106319 , 210303.21795141, 181216.09883086, 504607.9965645
3,
        199232.43455404, 328852.92541163, 403314.28196623, 417339.1328944
9,
        375881.08526492, 403197.77888165, 188979.68008263, 362402.5598572
6,
        294628.95253173, 335824.58233231, 289447.37292442, 307905.8562066
,
        304212.45930815, 245918.7448544 , 423369.21745346, 209751.1133410
1,
        464508.50821053, 313268.23305184, 437642.36232284, 255194.9729920
7,
        338515.44760229, 337044.41925766, 231297.50959276, 293494.7894233
8,
        362754.76034607, 466011.06815767, 504203.3892541 , 363063.0468049
9,
        356859.94483848, 382768.01336357, 273565.90838058, 341887.0994200
6,
        202561.12525852, 226021.59803407, 184738.12254742, 406294.9987851
,
        179836.69140424, 241548.18232332, 279529.38908598, 470977.1194133
2,
        247048.39464473, 349530.74752485, 290662.6784132 , 405964.8433277
5,
        419839.63745783, 476301.22336329, 293833.95263045, 310118.9218101
7,
        414281.2229151 , 289571.93742224, 302926.42538758, 463863.0124927
3,
        249128.71268129, 363590.1357771 , 388101.81719233, 397753.4656928
9,
        368188.89521752, 205614.05536704, 489256.78284726, 419420.9946028
3,
        206308.0617221 , 246852.45197237, 460241.26493034, 347139.7576852
```

```
4,
        520785.04039565, 363882.45319352, 424699.80363967, 353747.9492970
9,
        247791.56778919, 328276.38226609, 293096.68948041, 437324.5325352
5,
        259243.61420097, 445686.65165331, 268774.9486716 , 391885.3533658
2,
        348443.30855119, 362335.85311476, 442031.26258421, 439827.9712021
7,
        227816.06826322])
```

In [38]:

```
from sklearn.metrics import r2_score
#implements several loss, score, and utility functions to measure classifi
r2_score(y_test,ypred)
#r square score is used to evauate performance of regression
```

Out[38]:

```
0.7731910238316071
```

In [39]:

```
from sklearn.metrics import mean_squared_error
#measures how close the points are to the line
mean_squared_error(ypred,y_test)
```

Out[39]:

```
2568269005.9400477
```

In [40]:

```
results=pd.DataFrame(columns=['price (₹)','predicted'])
results['price (₹)']=y_test
results['predicted']=ypred
results=results.reset_index()
#it resets indexes from 0
results['ID']=results.index
results.head(15)
```

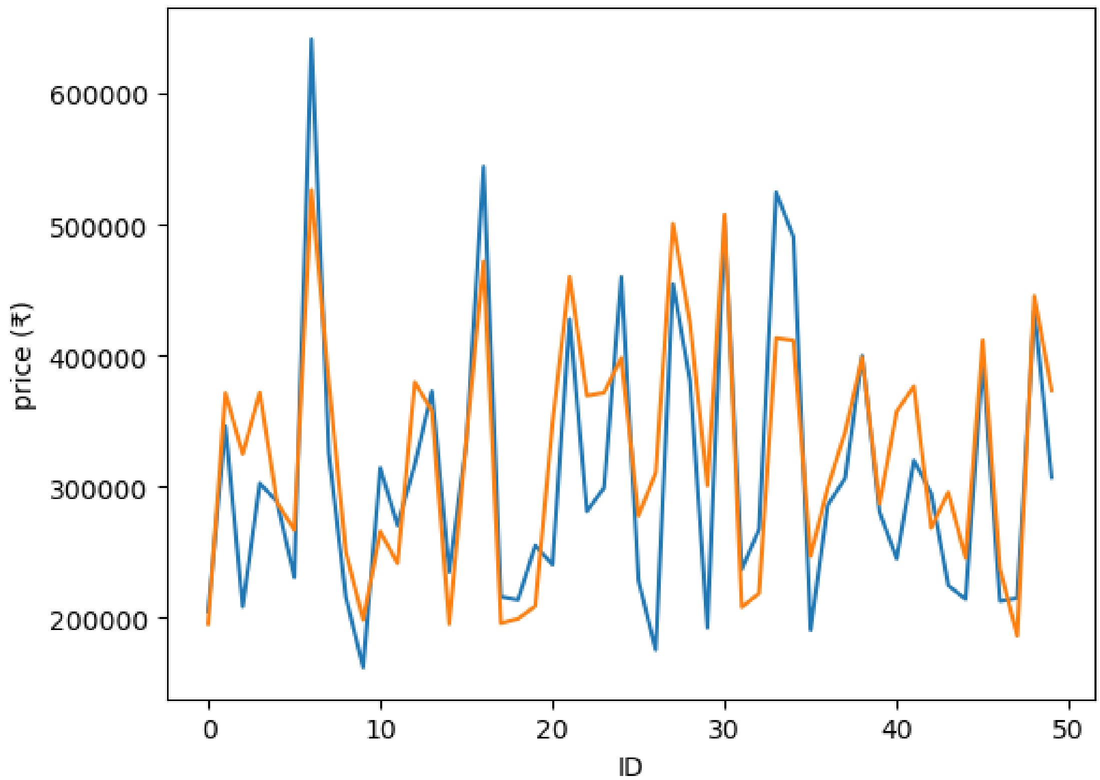
Out[40]:

	index	price (₹)	predicted	ID
0	217	205099	195539.369963	0
1	736	346399	371699.991519	1
2	612	208699	324974.832533	2
3	3172	302699	372023.643504	3
4	2039	288699	288041.703482	4
5	2787	231099	267115.149469	5
6	1728	641499	526562.207685	6
7	3096	326099	379910.795437	7
8	3312	216000	250510.566557	8
9	3008	162299	198464.030280	9
10	1135	314399	265901.205882	10
11	2655	270599	241589.937873	11
12	3065	316999	379730.503538	12
13	1255	372999	357774.473958	13
14	1106	234999	195335.100106	14

In [41]:

```
import seaborn as sns
# library which is used for making statistcal graphs
import matplotlib.pyplot as plt
#library for visualiztion
sns.lineplot(x='ID',y='price (₹)',data=results.head(50))
sns.lineplot(x='ID',y='predicted',data=results.head(50))
plt.plot()
```

Out[41]: []



Random Forest Regression

In [44]:
```
from sklearn.model_selection import GridSearchCV #GridSearchCV is for para
from sklearn.ensemble import RandomForestRegressor
reg=RandomForestRegressor()
n_estimators=[25,50,75,100,125,150,175,200] #number of decision trees in t
criterion=['mse'] #criteria for choosing nodes default = 'gini'
max_depth=[3,5,10] #maximum number of nodes in a tree default = None (it w
#parameters={'n_estimators': n_estimators,'criterion':criterion,'max_depth
parameters = {'n_estimators': n_estimators, 'criterion': ['friedman_mse',
RFC_reg = GridSearchCV(reg, parameters)
RFC_reg.fit(X_train,y_train)
```

Out[44]:
```
GridSearchCV(estimator=RandomForestRegressor(),
             param_grid={'criterion': ['friedman_mse', 'squared_error',
                                       'absolute_error', 'poisson'],
                         'max_depth': [3, 5, 10],
                         'n_estimators': [25, 50, 75, 100, 125, 150, 17
5, 200]})
```

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 [ ]:
```
#GridSearchCV is tool for a machine learning model to find the best ones a
#model (let's say it's a Random Forest) that has some settings you can adj
# Random Forest has hyperparameters like the number of trees in the forest
#Now, trying every possible combination of these settings manually to find
# from sklearn.ensemble module importing random forest regressor algorithm
#It belongs to the ensemble learning methods and is based on constructing
#criterion parameter in scikit-learn's RandomForestRegressor specifies the
#the default value for criterion is already set to 'mse', which stands for
```