In [ ]:
```
Importing Necessary Libraries
```

In [169]:
```
import pandas as pd
import numpy as np
```

In [170]:
```
df=pd.read_csv('CAR DETAILS FROM CAR DEKHO.csv')
```

In [171]:
```
df
```

Out[171]:

| | name | year | selling_price | km_driven | fuel | seller_type | transmission | owner |
|---|---|---|---|---|---|---|---|---|
| **0** | Maruti 800 AC | 2007 | 60000 | 70000 | Petrol | Individual | Manual | First Owner |
| **1** | Maruti Wagon R LXI Minor | 2007 | 135000 | 50000 | Petrol | Individual | Manual | First Owner |
| **2** | Hyundai Verna 1.6 SX | 2012 | 600000 | 100000 | Diesel | Individual | Manual | First Owner |
| **3** | Datsun RediGO T Option | 2017 | 250000 | 46000 | Petrol | Individual | Manual | First Owner |
| **4** | Honda Amaze VX i-DTEC | 2014 | 450000 | 141000 | Diesel | Individual | Manual | Second Owner |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **4335** | Hyundai i20 Magna 1.4 CRDi (Diesel) | 2014 | 409999 | 80000 | Diesel | Individual | Manual | Second Owner |
| **4336** | Hyundai i20 Magna 1.4 CRDi | 2014 | 409999 | 80000 | Diesel | Individual | Manual | Second Owner |
| **4337** | Maruti 800 AC BSIII | 2009 | 110000 | 83000 | Petrol | Individual | Manual | Second Owner |
| **4338** | Hyundai Creta 1.6 CRDi SX Option | 2016 | 865000 | 90000 | Diesel | Individual | Manual | First Owner |
| **4339** | Renault KWID RXT | 2016 | 225000 | 40000 | Petrol | Individual | Manual | First Owner |

4340 rows × 8 columns

In [173]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   name           4340 non-null   object
 1   year           4340 non-null   int64 
 2   selling_price  4340 non-null   int64 
 3   km_driven      4340 non-null   int64 
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

In [174]:
```
df.describe()
```

Out[174]:

| | year | selling_price | km_driven |
|---|---|---|---|
| **count** | 4340.000000 | 4.340000e+03 | 4340.000000 |
| **mean** | 2013.090783 | 5.041273e+05 | 66215.777419 |
| **std** | 4.215344 | 5.785487e+05 | 46644.102194 |
| **min** | 1992.000000 | 2.000000e+04 | 1.000000 |
| **25%** | 2011.000000 | 2.087498e+05 | 35000.000000 |
| **50%** | 2014.000000 | 3.500000e+05 | 60000.000000 |
| **75%** | 2016.000000 | 6.000000e+05 | 90000.000000 |
| **max** | 2020.000000 | 8.900000e+06 | 806599.000000 |

In [175]:
```
df['km_driven'].value_counts()
```

Out[175]:
```
70000     236
80000     228
50000     222
120000    220
60000     215
         ... 
19107       1
32077       1
6480        1
118400      1
112198      1
Name: km_driven, Length: 770, dtype: int64
```

In [176]:
```
df['year'].value_counts()
```

Out[176]:
```
2017    466
2015    421
2012    415
2013    386
2014    367
2018    366
2016    357
2011    271
2010    234
2019    195
2009    193
2008    145
2007    134
2006    110
2005     85
2020     48
2004     42
2003     23
2002     21
2001     20
1998     12
2000     12
1999     10
1997      3
1996      2
1995      1
1992      1
Name: year, dtype: int64
```

In [178]:
```
print(df['name'].unique())
print(df['fuel'].unique())
print(df['seller_type'].unique())
print(df['transmission'].unique())
print(df['owner'].unique())
```

```
['Maruti 800 AC' 'Maruti Wagon R LXI Minor' 'Hyundai Verna 1.6 SX' ...
 'Mahindra Verito 1.5 D6 BSIII'
 'Toyota Innova 2.5 VX (Diesel) 8 Seater BS IV'
 'Hyundai i20 Magna 1.4 CRDi']
['Petrol' 'Diesel' 'CNG' 'LPG' 'Electric']
['Individual' 'Dealer' 'Trustmark Dealer']
['Manual' 'Automatic']
['First Owner' 'Second Owner' 'Fourth & Above Owner' 'Third Owner'
 'Test Drive Car']
```

In [179]:
```
data.isnull().sum()
```

Out[179]:
```
name             0
year             0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
price_inlakh     0
dtype: int64
```

In [128]:
```
# There is no null values to drop,so we can proceed further
```

In [185]:
```
# Converting selling price into Lakhs
df['price_inlakh']=df['selling_price']/100000
```

In [187]:
```
df.drop(['selling_price'],axis=1,inplace=True)
```

In [191]:
```
df
```

Out[191]:

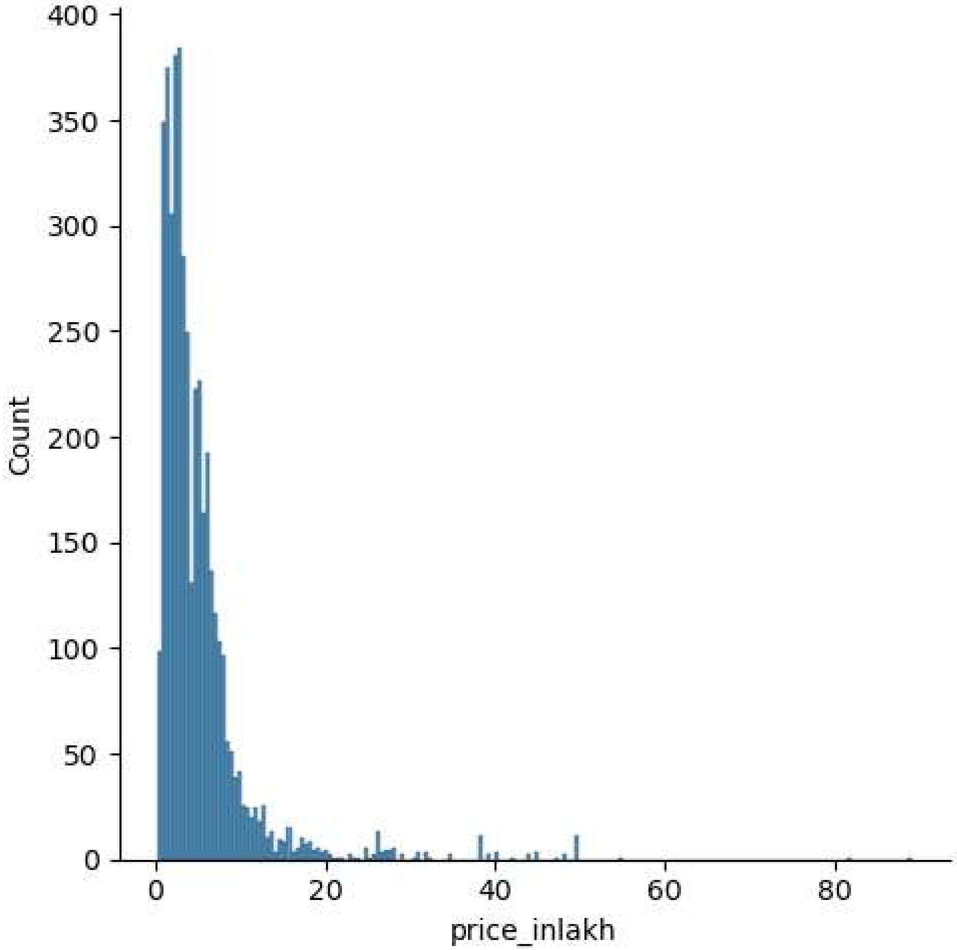
| | name | year | km_driven | fuel | seller_type | transmission | owner | price_inlakh |
|---|---|---|---|---|---|---|---|---|
| **0** | Maruti 800 AC | 2007 | 70000 | Petrol | Individual | Manual | First Owner | 0.60000 |
| **1** | Maruti Wagon R LXI Minor | 2007 | 50000 | Petrol | Individual | Manual | First Owner | 1.35000 |
| **2** | Hyundai Verna 1.6 SX | 2012 | 100000 | Diesel | Individual | Manual | First Owner | 6.00000 |
| **3** | Datsun RediGO T Option | 2017 | 46000 | Petrol | Individual | Manual | First Owner | 2.50000 |
| **4** | Honda Amaze VX i-DTEC | 2014 | 141000 | Diesel | Individual | Manual | Second Owner | 4.50000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **4335** | Hyundai i20 Magna 1.4 CRDi (Diesel) | 2014 | 80000 | Diesel | Individual | Manual | Second Owner | 4.09999 |
| **4336** | Hyundai i20 Magna 1.4 CRDi | 2014 | 80000 | Diesel | Individual | Manual | Second Owner | 4.09999 |
| **4337** | Maruti 800 AC BSIII | 2009 | 83000 | Petrol | Individual | Manual | Second Owner | 1.10000 |
| **4338** | Hyundai Creta 1.6 CRDi SX Option | 2016 | 90000 | Diesel | Individual | Manual | First Owner | 8.65000 |
| **4339** | Renault KWID RXT | 2016 | 40000 | Petrol | Individual | Manual | First Owner | 2.25000 |

4340 rows × 8 columns

In [ ]:
```
Exploratory Data Analysis(EDA)
```

In [196]:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.displot(df['price_inlakh'])
plt.show()
```

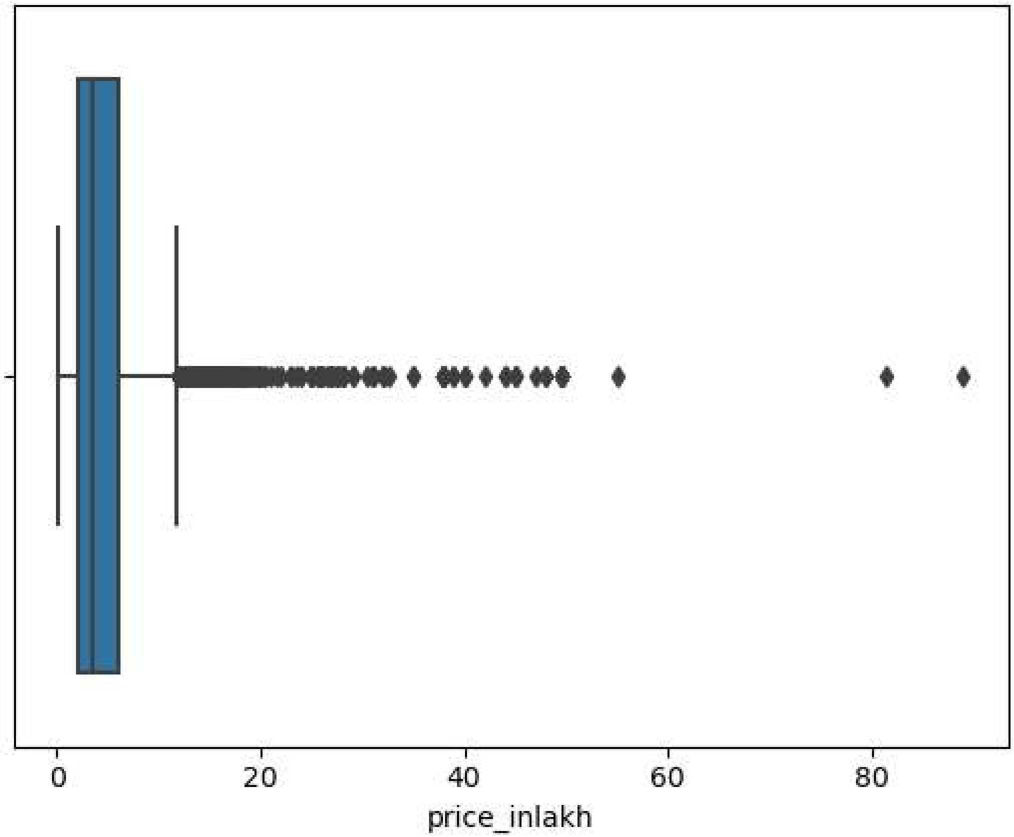



In [197]:

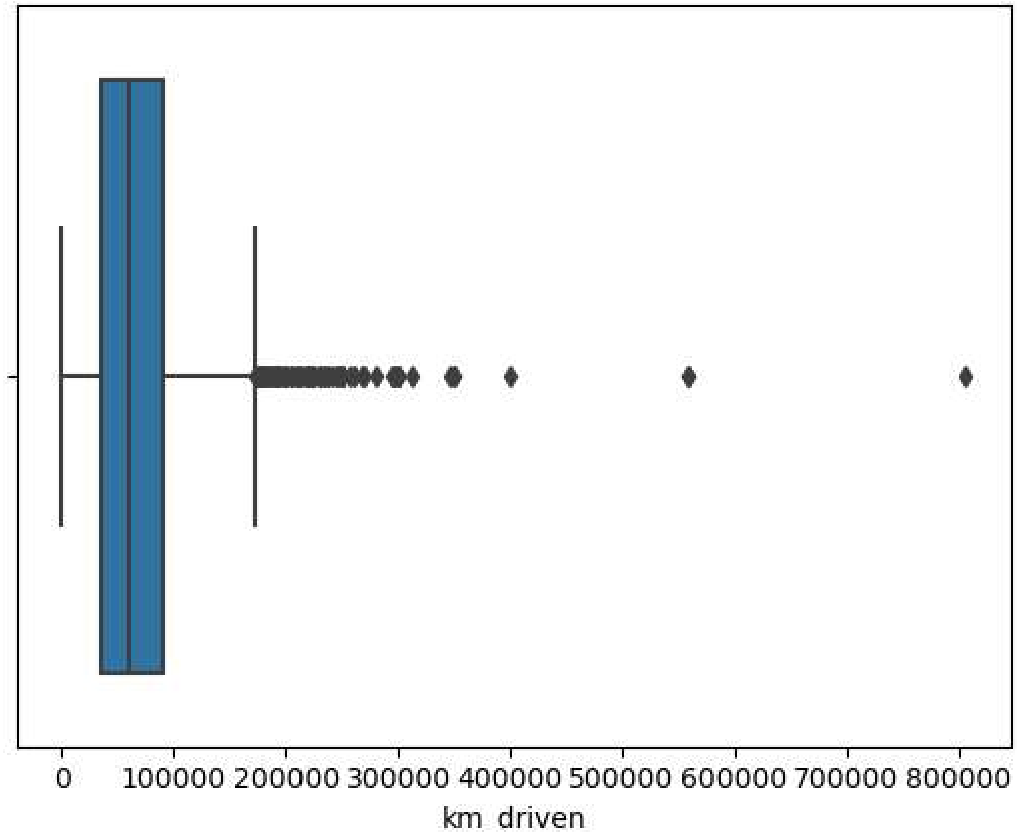
```
import warnings
warnings.filterwarnings('ignore')
```

In [198]:

```
import matplotlib.pyplot as plt
sns.boxplot(df['price_inlakh'])
plt.show()
```

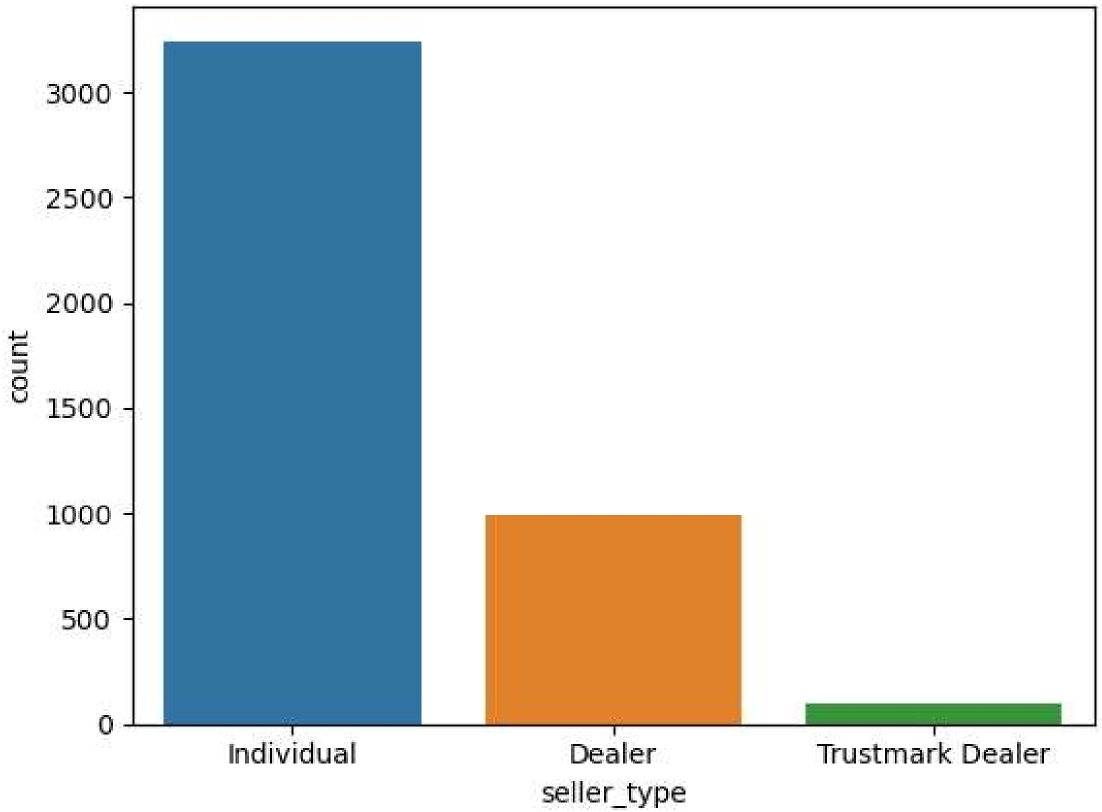


In [199]:
```
sns.boxplot(data['km_driven'])
plt.show()
```



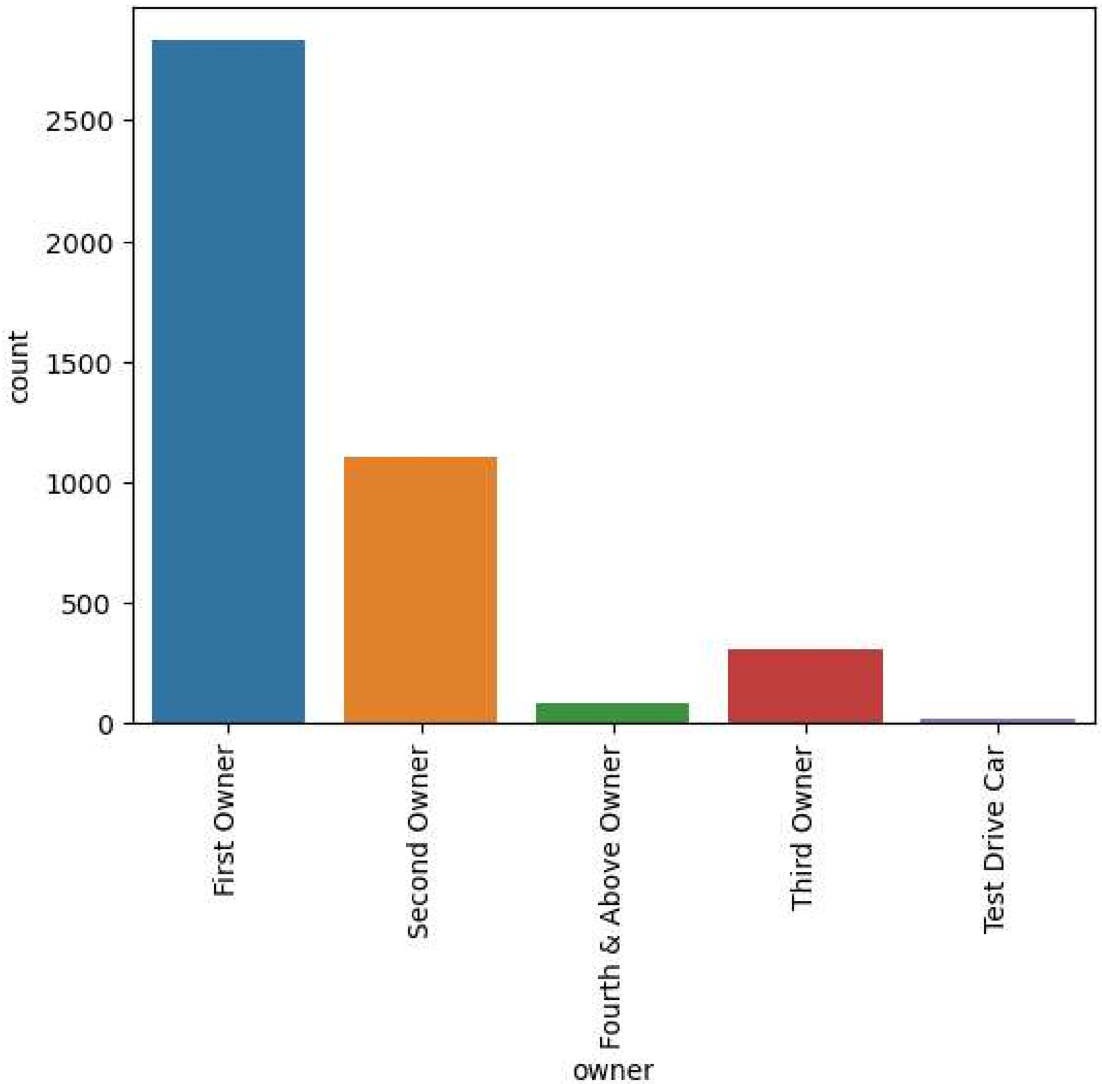
In [201]:

```
sns.countplot(df['seller_type'])
plt.show()
```



In [203]:

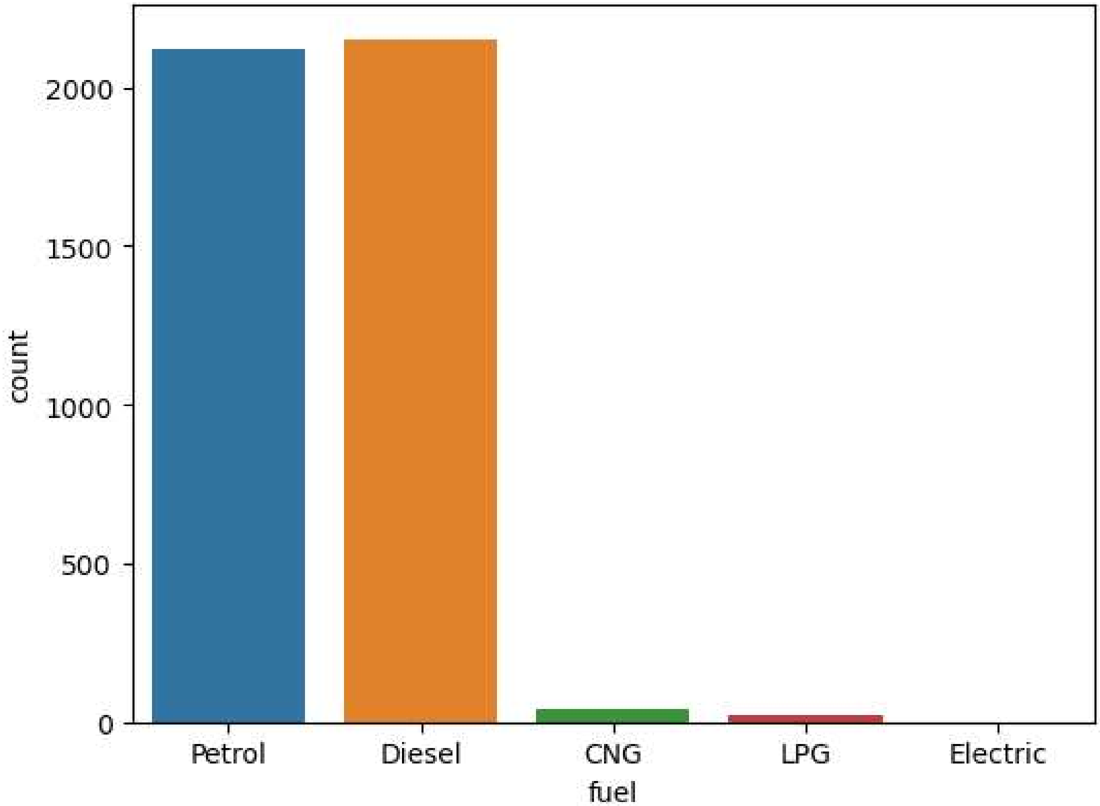
```
sns.countplot(df['owner'])
plt.xticks(rotation=90)
plt.show()
```



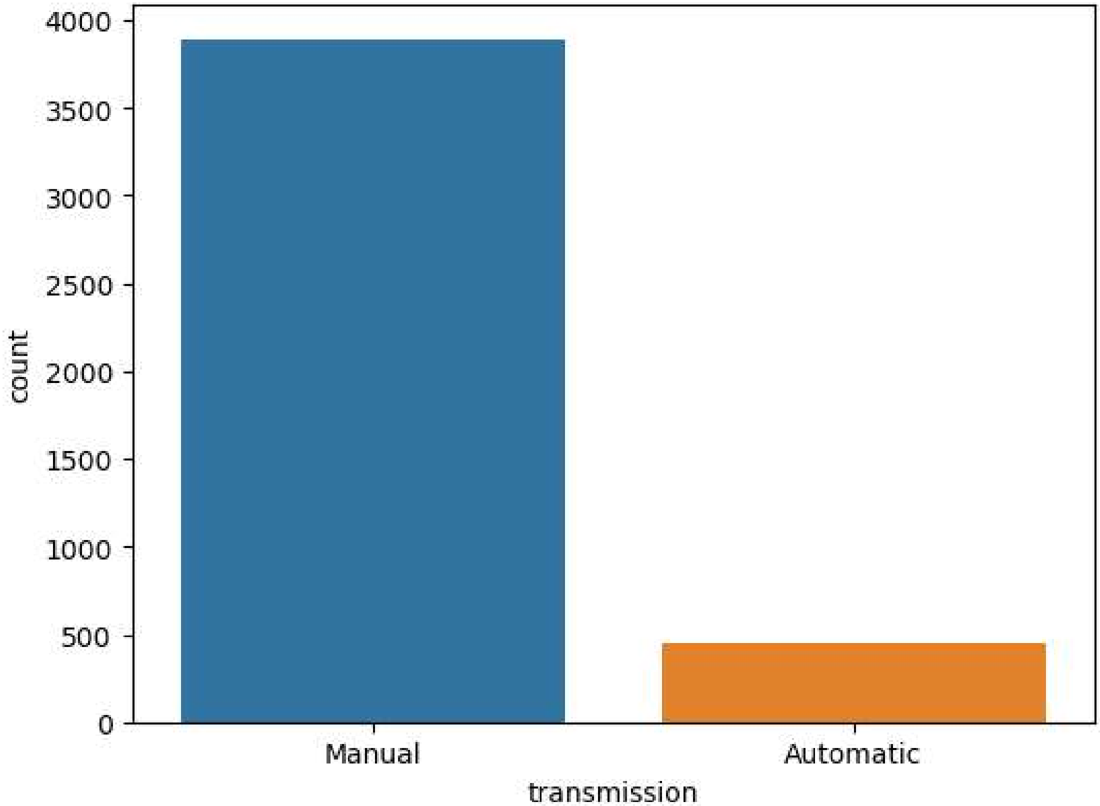

In [135]:

```
# First and Second owner cars mostly sold when compared to more owners
```

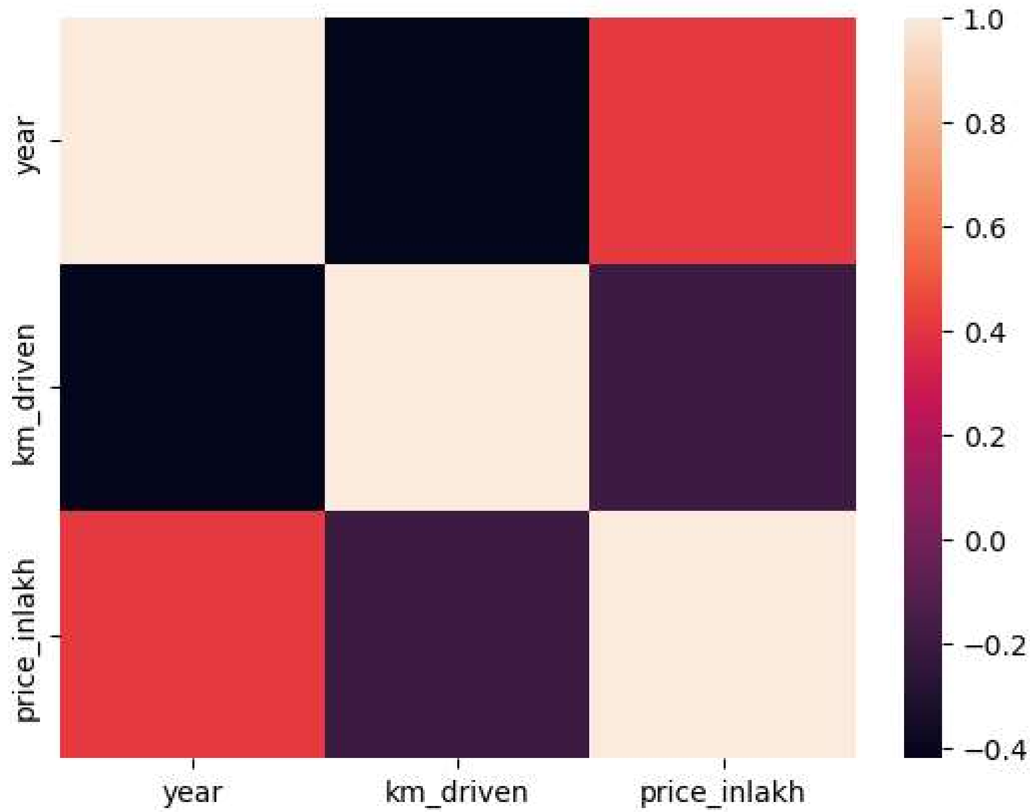
In [205]:
```
sns.countplot(df['fuel'])
plt.show()
```



In [206]:
```
sns.countplot(df['transmission'])
plt.show()
```



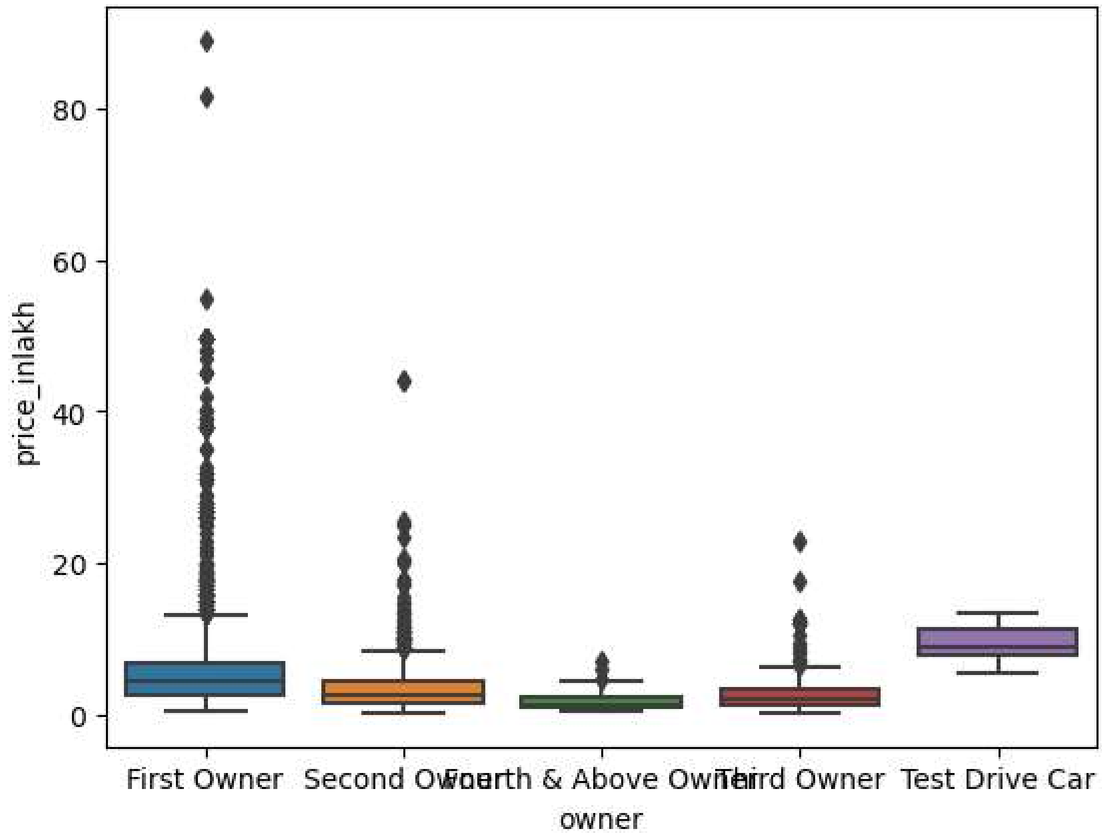
In [207]:
```
corr=df.corr()
sns.heatmap(corr)
plt.show()
```



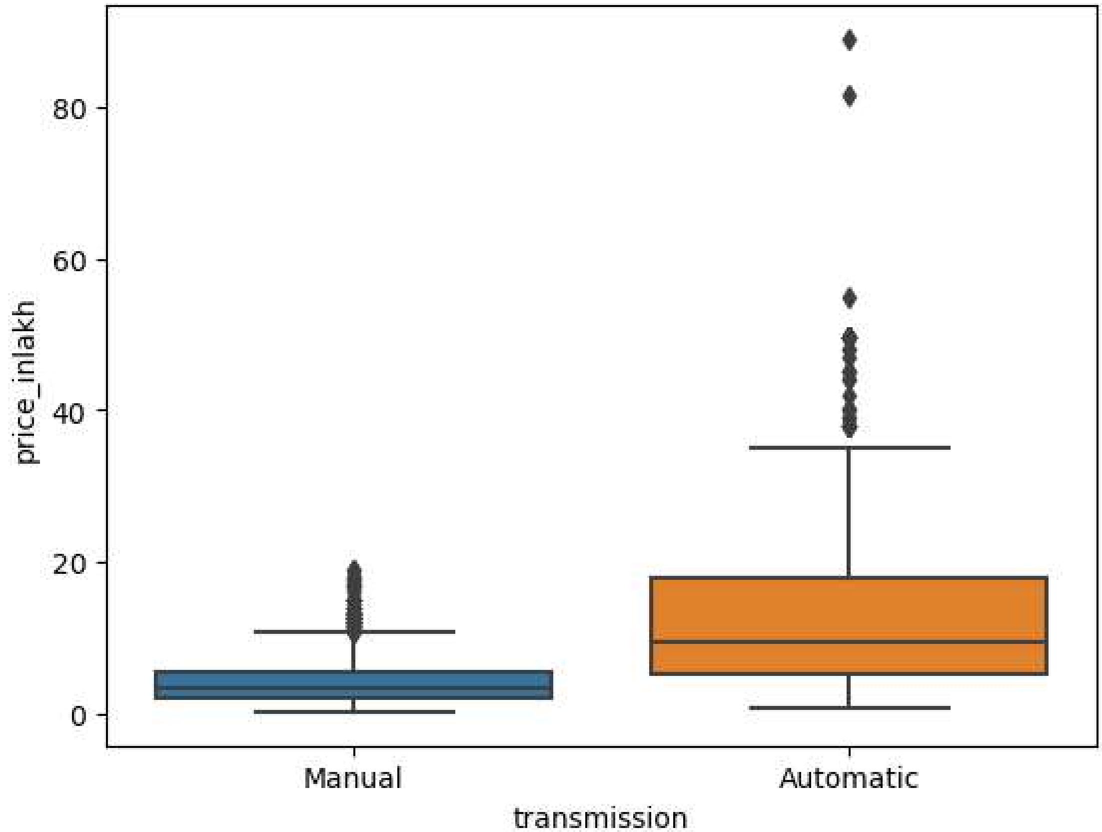

In [139]:
```
# There is no such relation between the features
```

In [208]:

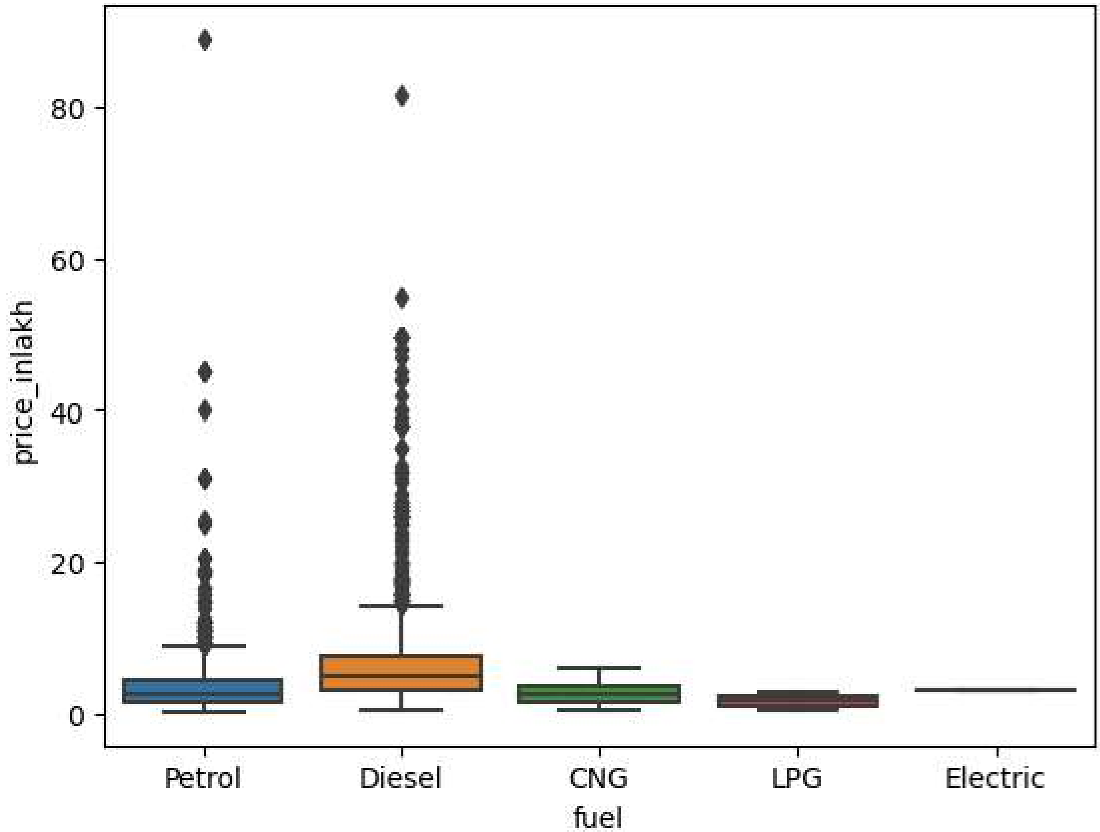
```
sns.boxplot(y='price_inlakh',x='owner',data=df)
plt.show()
```



In [210]:
```
sns.boxplot(y='price_inlakh',x='transmission',data=df)
plt.show()
```



In [146]:
```
sns.boxplot(y='price_inlakh',x='fuel',data=data)
plt.show()
```




In [ ]:
```
# After looking transmission,fuel,owner vs price_in lakh and boxplot of pr
# above forty lakh.We can remove that outliers
```

In [211]:
```
df2=df[df['price_inlakh']<40]
```

In [215]:
```
df2=df2[df2['km_driven']<300000]
```

In [216]:
```
df2.shape
```

Out[216]: (4301, 8)

In [ ]:
```
# 39 records identified as outliers and removed
```

In [ ]:
```
# Data Preprocessing
```

In [217]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [218]:
```
df2.name=le.fit_transform(df2.name)
df2.fuel=le.fit_transform(df2.fuel)
df2.transmission=le.fit_transform(df2.transmission)
df2.owner=le.fit_transform(df2.owner)
df2.seller_type=le.fit_transform(df2.seller_type)
```

In [219]:
```
df2
```

Out[219]:

| | name | year | km_driven | fuel | seller_type | transmission | owner | price_inlakh |
|---|---|---|---|---|---|---|---|---|
| **0** | 767 | 2007 | 70000 | 4 | 1 | 1 | 0 | 0.60000 |
| **1** | 1033 | 2007 | 50000 | 4 | 1 | 1 | 0 | 1.35000 |
| **2** | 500 | 2012 | 100000 | 1 | 1 | 1 | 0 | 6.00000 |
| **3** | 113 | 2017 | 46000 | 4 | 1 | 1 | 0 | 2.50000 |
| **4** | 274 | 2014 | 141000 | 1 | 1 | 1 | 2 | 4.50000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **4335** | 597 | 2014 | 80000 | 1 | 1 | 1 | 2 | 4.09999 |
| **4336** | 596 | 2014 | 80000 | 1 | 1 | 1 | 2 | 4.09999 |
| **4337** | 769 | 2009 | 83000 | 4 | 1 | 1 | 2 | 1.10000 |
| **4338** | 376 | 2016 | 90000 | 1 | 1 | 1 | 0 | 8.65000 |
| **4339** | 1142 | 2016 | 40000 | 4 | 1 | 1 | 0 | 2.25000 |

4301 rows × 8 columns

In [ ]:
```
# Model Building
```

In [220]:
```
X=df2.drop('price_inlakh',axis=1)
Y=df2['price_inlakh']
```

In [221]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=.20,random_st
```

In [222]:
```
#Linear Regressor
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[222]:

```
▾ LinearRegression
LinearRegression()
```

In [223]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sc
y_pre_lr=lr.predict(x_test)
mae = round(mean_absolute_error(y_test, y_pre_lr), 2)
mse = round(mean_squared_error(y_test, y_pre_lr), 2)
rmse = round(np.sqrt(mse), 2)
r2 = round(r2_score(y_test, y_pre_lr), 2)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("r-squared:", r2)
```

```
MAE: 2.1
MSE: 10.82
RMSE: 3.29
r-squared: 0.49
```

In [224]:

```
print(lr.score(x_train,y_train))
print(lr.score(x_test,y_test))
```

```
0.5000870729486298
0.4913890214038892
```

In [235]:

```
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators=100)
rfr.fit(x_train, y_train)
y_pre_rfr= rf_reg.predict(x_test)
print("Accuracy on Traing set: ",rfr.score(x_train,y_train))
print("Accuracy on Testing set: ",rfr.score(x_test,y_test))
```

```
Accuracy on Traing set:  0.9782170621584568
Accuracy on Testing set:  0.822319241792626
```

In [236]:

```
mae = round(mean_absolute_error(y_test, y_pre_rfr), 2)
mse = round(mean_squared_error(y_test, y_pre_rfr), 2)
rmse = round(np.sqrt(mse), 2)
r2 = round(r2_score(y_test, y_pre_rfr), 2)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("r-squared:", r2)
```

```
MAE: 0.73
MSE: 2.36
RMSE: 1.54
r-squared: 0.89
```

In [231]:
```
# Gradient Boosting Regressor
from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor()
gbr.fit(x_train, y_train)
y_pre_gbr= gbr.predict(x_test)
print("Accuracy on Traing set: ",gbr.score(x_train,y_train))
print("Accuracy on Testing set: ",gbr.score(x_test,y_test))
```

```
Accuracy on Traing set:  0.8509833393341351
Accuracy on Testing set:  0.7779316920916748
```

In [232]:
```
mae = round(mean_absolute_error(y_test, y_pre_gbr), 2)
mse = round(mean_squared_error(y_test, y_pre_gbr), 2)
rmse = round(np.sqrt(mse), 2)
r2 = round(r2_score(y_test, y_pre_gbr), 2)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("r-squared:", r2)
```

```
MAE: 1.25
MSE: 4.73
RMSE: 2.17
r-squared: 0.78
```

In [233]:
```
# Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(x_train, y_train)
y_preddtr= dtr.predict(x_test)
print("Accuracy on Traing set: ",dtr.score(x_train,y_train))
print("Accuracy on Testing set: ",dtr.score(x_test,y_test))
```

```
Accuracy on Traing set:  0.9998629923781052
Accuracy on Testing set:  0.7341216130779781
```

In [234]:
```
mae = round(mean_absolute_error(y_test, y_preddtr), 2)
mse = round(mean_squared_error(y_test, y_preddtr), 2)
rmse = round(np.sqrt(mse), 2)
r2 = round(r2_score(y_test, y_preddtr), 2)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("r-squared:", r2)
```

```
MAE: 1.12
MSE: 5.66
RMSE: 2.38
r-squared: 0.73
```

In [237]:
```
# AdaBoost Regressor
from sklearn.ensemble import AdaBoostRegressor
abr = AdaBoostRegressor()
abr.fit(x_train, y_train)
y_pre_abr= abr.predict(x_test)
print("Accuracy on Traing set: ",abr.score(x_train,y_train))
print("Accuracy on Testing set: ",abr.score(x_test,y_test))
```

```
Accuracy on Traing set:  0.6200215076739344
Accuracy on Testing set:  0.5800465008927578
```

In [238]:
```
mae = round(mean_absolute_error(y_test, y_pre_abr), 2)
mse = round(mean_squared_error(y_test, y_pre_abr), 2)
rmse = round(np.sqrt(mse), 2)
r2 = round(r2_score(y_test, y_pre_abr), 2)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("r-squared:", r2)
```

```
MAE: 2.14
MSE: 8.94
RMSE: 2.99
r-squared: 0.58
```

In [239]:
```
regression_models = [lr,rfr,gbr,dtr,abr]
score_train = list()
score_test = list()

for model in regression_models :
    model.fit(x_train,y_train)
    y_pred = model.predict(x_test)

    score_train.append(model.score(x_train,y_train))
    score_test.append(model.score(x_test,y_test))
```

In [240]:
```
model_names = ['Linear Regression','Random Forest Regressor','Gradient Boo
               'AdaBoostRegressor']

scores = pd.DataFrame([model_names,score_train,score_test])
scores
```

Out[240]:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **0** | Linear Regression | Random Forest Regressor | Gradient Boosting Regressor | Decision Tree Regressor | AdaBoostRegressor |
| **1** | 0.500087 | 0.977721 | 0.850983 | 0.999863 | 0.614269 |
| **2** | 0.491389 | 0.813603 | 0.777932 | 0.723507 | 0.55971 |

In [241]:
```
scores = scores.transpose()
scores.columns = [ 'Model','Training Set Accuracy','Testing set Accuracy']
scores
```

Out[241]:

| | Model | Training Set Accuracy | Testing set Accuracy |
|---|---|---|---|
| 0 | Linear Regression | 0.500087 | 0.491389 |
| 1 | Random Forest Regressor | 0.977721 | 0.813603 |
| 2 | Gradient Boosting Regressor | 0.850983 | 0.777932 |
| 3 | Decision Tree Regressor | 0.999863 | 0.723507 |
| 4 | AdaBoostRegressor | 0.614269 | 0.55971 |

In [168]:
```
# Conclusion : Random Forest with 81 % accuracy and
              0.89 as R-Squared value has chosen for prediction
```