Approach

1)Each column of the datasets were checked for any data inconsistency.

2)Required actions were taken for specific columns where data inconsistencies were found.

3)Different regression algorithms were used to build different models.

4) **RandomForestRegressor** gave us the best model.So the .ipynb file contains only the random forest models.

5)Tuning of hyperparameters were required for random forest regressor to optimize the RMSLE value.

6)Best features were selected using VIF,RFE,forward elimnation,backward elimination,random forest and extra trees techniques. Features extracted using extra trees technique gave us the best model.

Importing Librabries

In [10]:

```
# suppress display of warnings
import warnings
warnings.filterwarnings("ignore")

# 'Pandas' is used for data manipulation and analysis
import pandas as pd

# 'Numpy' is used for mathematical operations on large, multi-dimensional arrays and ma
import numpy as np

# 'Matplotlib' is a data visualization library for 2D and 3D plots, built on numpy
import matplotlib.pyplot as plt

# 'Seaborn' is based on matplotlib; used for plotting statistical graphics
import seaborn as sns

# import various functions to perform regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor

#importing metrics for tabulating the result
from sklearn.metrics import mean_squared_log_error

#setting the plot size using rcParams
plt.rcParams['figure.figsize'] = [15,8]
```

Importing the data

In [11]:

```
df = pd.read_csv('car_price_train.csv')
df_test = pd.read_csv('test.csv')
```

Understanding the data

In [12]:
```
df.head()
```

Out[12]:

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005 km
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3	192000 km
2	45774419	8467	-	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000 km
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966 km
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901 km

From the above display we can see that:

1)The 'Levy' column contains '-' symbol.We need to look into this column.

2)In the 'Doors' column there are month names which we need to remove.

3)In the 'MIleage' column there is 'km' written, we need to seperate this 'km' for model building purpose.

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

Out[13]:

	ID	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinder
0	44020629	-	VOLKSWAGEN	Golf	2012	Hatchback	No	Diesel	2.0 Turbo	0 km	
1	45784798	-	HYUNDAI	Sonata	2012	Sedan	Yes	Petrol	2.4	26000 km	
2	45800257	-	NISSAN	Tiida	2005	Sedan	No	Petrol	1.5	168000 km	
3	45797981	975	VOLVO	XC90	2012	Jeep	Yes	Petrol	3.2	143000 km	
4	45814303	-	OPEL	Astra	1993	Hatchback	No	Petrol	1.6	200000 km	

From the above display we can see that:

1)The 'Levy' column contains '-' symbol.We need to look into this column.

2)In the 'Doors' column there are month names which we need to remove.

3)In the 'Mileage' column there is 'km' written, we need to seperate this 'km' for model building purpose.

4)We need to delete the 'Price' column as we need to predict it.

In [7]:
```
#Understanding the shape of the data
df.shape
```

Out[7]:
```
(19237, 18)
```

We can see that there are 19237 records and 18 rows

In [8]:
```
#assigning the target variable
y=df['Price']
```

In [9]:
```
#Concatenting both the test and train datasets together so that we can perform all the
df = df.drop(['Price'],axis=1)
df_test=df_test.drop(['Price'],axis=1)
df_merge = df.append(df_test)
df_merge.reset_index(inplace=True)
df_merge= df_merge.drop(['index'],axis=1)
```

In [45]:
```
#checking the shape of the merged dataset
df_merge.shape
```

Out[45]:
```
(27482, 17)
```

In [46]:
```
#checking the dtypes and Unique values
info = pd.DataFrame()
info['DataTypes'] = df_merge.dtypes
info['Unique_values'] = df_merge.nunique()
info
```

Out[46]:

	DataTypes	Unique_values
ID	int64	26886
Levy	object	635
Manufacturer	object	68
Model	object	1981
Prod. year	int64	59
Category	object	11

	DataTypes	Unique_values
Leather interior	object	2
Fuel type	object	7
Engine volume	object	112
Mileage	object	9977
Cylinders	float64	16
Gear box type	object	4
Drive wheels	object	3
Doors	object	3
Wheel	object	2
Color	object	16
Airbags	int64	17

We need to convert 'Mileage' column into float as we know that it is of float/integer datatype.

In [47]:
```
df_merge.describe(include='object')
```

Out[47]:

	Levy	Manufacturer	Model	Category	Leather interior	Fuel type	Engine volume	Mileage	Gear box type	Drive wheels
count	27482	27482	27482	27482	27482	27482	27482	27482	27482	27482
unique	635	68	1981	11	2	7	112	9977	4	3
top	-	HYUNDAI	Sonata	Sedan	Yes	Petrol	2	0 km	Automatic	Front
freq	8273	5358	1557	12534	19926	14492	5587	1013	19296	18374

In [48]:
```
df_merge.describe(include=np.number)
```

Out[48]:

	ID	Prod. year	Cylinders	Airbags
count	2.748200e+04	27482.000000	27482.000000	27482.000000
mean	4.557857e+07	2010.912452	4.593734	6.563569
std	9.244972e+05	5.632865	1.413918	4.329256
min	2.074688e+07	1939.000000	1.000000	0.000000
25%	4.569808e+07	2009.000000	4.000000	4.000000
50%	4.577207e+07	2012.000000	4.000000	6.000000
75%	4.580197e+07	2015.000000	4.000000	12.000000
max	4.581665e+07	2020.000000	127.000000	16.000000

From the describe() function we can get the mean,count and quantiles values for numeric data and count,frequency of object type data. From the above displays it can be seen that there are no missing values.

Rectifying the data

In [49]:
```
#removing the 'km' from the mileage column and converting it to float
df_merge['Mileage'] = pd.to_numeric(df_merge.Mileage.str.split(' ').str[0], downcast='f
```

In [50]:
```
#replacing all the '0' values with the mean values of the 'Mileage' column
df_merge['Mileage'] = np.where(df_merge['Mileage'] == 0.0,df_merge['Mileage'].mean(),df
```

In [51]:
```
#checking the unique values of 'Doors' column
df_merge['Doors'].unique()
```

Out[51]:
```
array(['04-May', '02-Mar', '>5'], dtype=object)
```

In [52]:
```
#cleaning the Doors column
df_merge['Doors'] = np.where((df_merge['Doors'] == '04-May') | (df_merge['Doors'] == '0
```

In [53]:
```
#checking the unique values of 'Doors' column after cleaning
df_merge['Doors'].unique()
```

Out[53]:
```
array(['04', '02', '>5'], dtype=object)
```

In [54]:
```
#checking the unique values of 'Levy' column after cleaning
df_merge['Levy'].unique()
```

Out[54]:
```
array(['1399', '1018', '-', '862', '446', '891', '761', '751', '394',
       '1053', '1055', '1079', '810', '2386', '1850', '531', '586',
       '1249', '2455', '583', '1537', '1288', '915', '1750', '707',
       '1077', '1486', '1091', '650', '382', '1436', '1194', '503',
       '1017', '1104', '639', '629', '919', '781', '530', '640', '765',
       '777', '779', '934', '769', '645', '1185', '1324', '830', '1187',
       '1111', '760', '642', '1604', '1095', '966', '473', '1138', '1811',
       '988', '917', '1156', '687', '11714', '836', '1347', '2866',
       '1646', '259', '609', '697', '585', '475', '690', '308', '1823',
       '1361', '1273', '924', '584', '2078', '831', '1172', '893', '1872',
       '1885', '1266', '447', '2148', '1730', '730', '289', '502', '333',
       '1325', '247', '879', '1342', '1327', '1598', '1514', '1058',
       '738', '1935', '481', '1522', '1282', '456', '880', '900', '798',
       '1277', '442', '1051', '790', '1292', '1047', '528', '1211',
       '1493', '1793', '574', '930', '1998', '271', '706', '1481', '1677',
       '1661', '1286', '1408', '1090', '595', '1451', '1267', '993',
       '1714', '878', '641', '749', '1511', '603', '353', '877', '1236',
       '1141', '397', '784', '1024', '1357', '1301', '770', '922', '1438',
       '753', '607', '1363', '638', '490', '431', '565', '517', '833',
       '489', '1760', '986', '1841', '1620', '1360', '474', '1099', '978',
       '1624', '1946', '1268', '1307', '696', '649', '666', '2151', '551',
```

```
       '800', '971', '1323', '2377', '1845', '1083', '694', '463', '419',
       '345', '1515', '1505', '2056', '1203', '729', '460', '1356', '876',
       '911', '1190', '780', '448', '2410', '1848', '1148', '834', '1275',
       '1028', '1197', '724', '890', '1705', '505', '789', '2959', '518',
       '461', '1719', '2858', '3156', '2225', '2177', '1968', '1888',
       '1308', '2736', '1103', '557', '2195', '843', '1664', '723',
       '4508', '562', '501', '2018', '1076', '1202', '3301', '691',
       '1440', '1869', '1178', '418', '1820', '1413', '488', '1304',
       '363', '2108', '521', '1659', '87', '1411', '1528', '3292', '7058',
       '1578', '627', '874', '1996', '1488', '5679', '1234', '5603',
       '400', '889', '3268', '875', '949', '2265', '441', '742', '425',
       '2476', '2971', '614', '1816', '1375', '1405', '2297', '1062',
       '1113', '420', '2469', '658', '1951', '2670', '2578', '1995',
       '1032', '994', '1011', '2421', '1296', '155', '494', '426', '1086',
       '961', '2236', '1829', '764', '1834', '1054', '617', '1529',
       '2266', '637', '626', '1832', '1016', '2002', '1756', '746',
       '1285', '2690', '1118', '5332', '980', '1807', '970', '1228',
       '1195', '1132', '1768', '1384', '1080', '7063', '1817', '1452',
       '1975', '1368', '702', '1974', '1781', '1036', '944', '663', '364',
       '1539', '1345', '1680', '2209', '741', '1575', '695', '1317',
       '294', '1525', '424', '997', '1473', '1552', '2819', '2188',
       '1668', '3057', '799', '1502', '2606', '552', '1694', '1759',
       '1110', '399', '1470', '1174', '5877', '1474', '1688', '526',
       '686', '5908', '1107', '2070', '1468', '1246', '1685', '556',
       '1533', '1917', '1346', '732', '692', '579', '421', '362', '3505',
       '1855', '2711', '1586', '3739', '681', '1708', '2278', '1701',
       '722', '1482', '928', '827', '832', '527', '604', '173', '1341',
       '3329', '1553', '859', '167', '916', '828', '2082', '1176', '1108',
       '975', '3008', '1516', '2269', '1699', '2073', '1031', '1503',
       '2364', '1030', '1442', '5666', '2715', '1437', '2067', '1426',
       '2908', '1279', '866', '4283', '279', '2658', '3015', '2004',
       '1391', '4736', '748', '1466', '644', '683', '2705', '1297', '731',
       '1252', '2216', '3141', '3273', '1518', '1723', '1588', '972',
       '682', '1094', '668', '175', '967', '402', '3894', '1960', '1599',
       '2000', '2084', '1621', '714', '1109', '3989', '873', '1572',
       '1163', '1991', '1716', '1673', '2562', '2874', '965', '462',
       '605', '1948', '1736', '3518', '2054', '2467', '1681', '1272',
       '1205', '750', '2156', '2566', '115', '524', '3184', '676', '1678',
       '612', '328', '955', '1441', '1675', '3965', '2909', '623', '822',
       '867', '3025', '1993', '792', '636', '4057', '3743', '2337',
       '2570', '2418', '2472', '3910', '1662', '2123', '2628', '3208',
       '2080', '3699', '2913', '864', '2505', '870', '7536', '1924',
       '1671', '1064', '1836', '1866', '4741', '841', '1369', '5681',
       '3112', '1366', '2223', '1198', '1039', '3811', '3571', '1387',
       '1171', '1365', '1531', '1590', '11706', '2308', '4860', '1641',
       '1045', '1901', '1883', '1613', '4746', '3061', '2547', '3258',
       '7528', '2317', '1846', '3005', '2802', '1545', '2096', '2625',
       '1526', '1721', '437', '818', '1364', '1168', '2132', '3515',
       '771', '4514', '1844', '914', '923', '1830', '2553', '2008', '977',
       '2969', '4053', '1594', '1415', '1698', '5220', '1254', '1747',
       '3718', '2643', '1044', '4735', '1555', '5668', '582', '2390',
       '238', '1040', '835', '763', '2040', '1696', '1837', '1614',
       '1519', '2800', '1035', '2528', '4739', '1271', '1359', '2488',
       '2235', '2052', '1226', '4159', '2068', '2243', '2504', '1762',
       '3592', '1899', '704', '2373', '884'], dtype=object)
```

In [55]:

```
#converting the Levy column to float as it is the Tax
df_merge['Levy'] = pd.to_numeric(df_merge['Levy'].replace('-', '0'), downcast='float')
```

In [56]:
```
#Replacing the 0 in the 'Levy' column with mean of that column
df_merge['Levy'] = np.where(df_merge['Levy'] == 0.0,df_merge['Levy'].mean(),df_merge['L
```

In [57]:
```
#checking the unique values in the 'Engine volume' column
df_merge['Engine volume'].unique()
```

Out[57]:
```
array(['3.5', '3', '1.3', '2.5', '2', '1.8', '2.4', '4', '1.6', '3.3',
       '2.0 Turbo', '2.2 Turbo', '4.7', '1.5', '4.4', '3.0 Turbo',
       '1.4 Turbo', '3.6', '2.3', '1.5 Turbo', '1.6 Turbo', '2.2',
       '2.3 Turbo', '1.4', '5.5', '2.8 Turbo', '3.2', '3.8', '4.6', '1.2',
       '5', '1.7', '2.9', '0.5', '1.8 Turbo', '2.4 Turbo', '3.5 Turbo',
       '1.9', '2.7', '4.8', '5.3', '0.4', '2.8', '3.2 Turbo', '1.1',
       '2.1', '0.7', '5.4', '1.3 Turbo', '3.7', '1', '2.5 Turbo', '2.6',
       '1.9 Turbo', '4.4 Turbo', '4.7 Turbo', '0.8', '0.2 Turbo', '5.7',
       '4.8 Turbo', '4.6 Turbo', '6.7', '6.2', '1.2 Turbo', '3.4',
       '1.7 Turbo', '6.3 Turbo', '2.7 Turbo', '4.3', '4.2', '2.9 Turbo',
       '0', '4.0 Turbo', '20', '3.6 Turbo', '0.3', '3.7 Turbo', '5.9',
       '5.5 Turbo', '0.2', '2.1 Turbo', '5.6', '6', '0.7 Turbo',
       '0.6 Turbo', '6.8', '4.5', '0.6', '7.3', '0.1', '1.0 Turbo', '6.3',
       '4.5 Turbo', '0.8 Turbo', '4.2 Turbo', '3.1', '5.0 Turbo', '6.4',
       '3.9', '5.7 Turbo', '0.9', '0.4 Turbo', '5.4 Turbo', '0.3 Turbo',
       '5.2', '5.8', '1.1 Turbo', '6.6', '6.1', '10.8', '6.0 Turbo',
       '2.6 Turbo'], dtype=object)
```

In [58]:
```
#We can see that there are some values with 'Turbo' and some values without 'Turbo'
#So we remove the word 'Turbo' from all records that have it
df_merge['Engine volume'] = pd.to_numeric(df_merge['Engine volume'].str.split(' ').str[
```

In [59]:
```
#Replacing the '0' in the 'Engine volume' column with the mean value of that column
df_merge['Engine volume']=np.where(df_merge['Engine volume'] == 0.0,df_merge['Engine vo
```

In [60]:
```
#Feature engineering the production year column
import datetime as dt
currt_time = dt.datetime.now()
df_merge['Prod. year'] = currt_time.year - df_merge['Prod. year']
```

In [61]:
```
#Checking the dataset after all the retification
df_merge.head()
```

Out[61]:

	ID	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage
0	45654403	1399.000000	LEXUS	RX 450	11	Jeep	Yes	Hybrid	3.5	186005.0
1	44731507	1018.000000	CHEVROLET	Equinox	10	Jeep	No	Petrol	3.0	192000.0
2	45774419	636.172974	HONDA	FIT	15	Hatchback	No	Petrol	1.3	200000.0
3	45769185	862.000000	FORD	Escape	10	Jeep	Yes	Hybrid	2.5	168966.0

	ID	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage
4	45809263	446.000000	HONDA	FIT	7	Hatchback	Yes	Petrol	1.3	91901.0

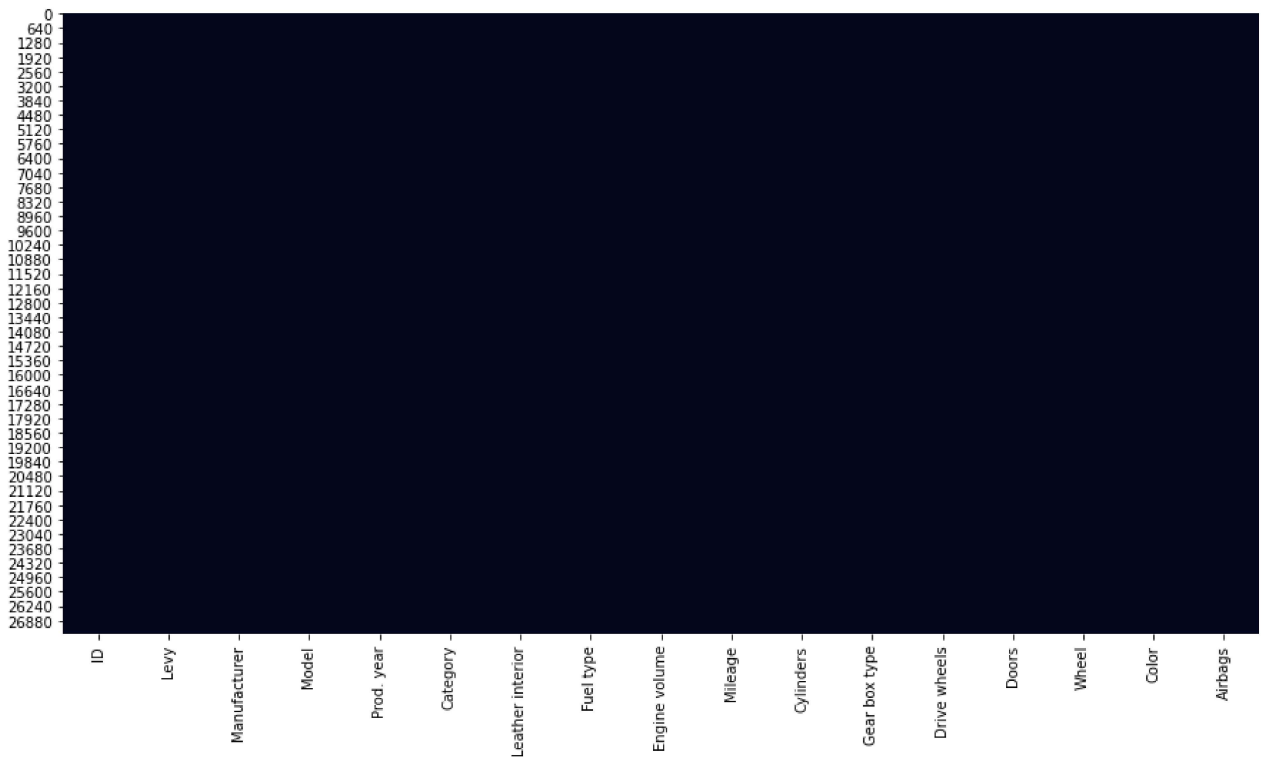
Extrapolatory Data Analysis

In [62]:

```
sns.heatmap(df_merge.isnull(),cbar=False)
plt.show
```

Out[62]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



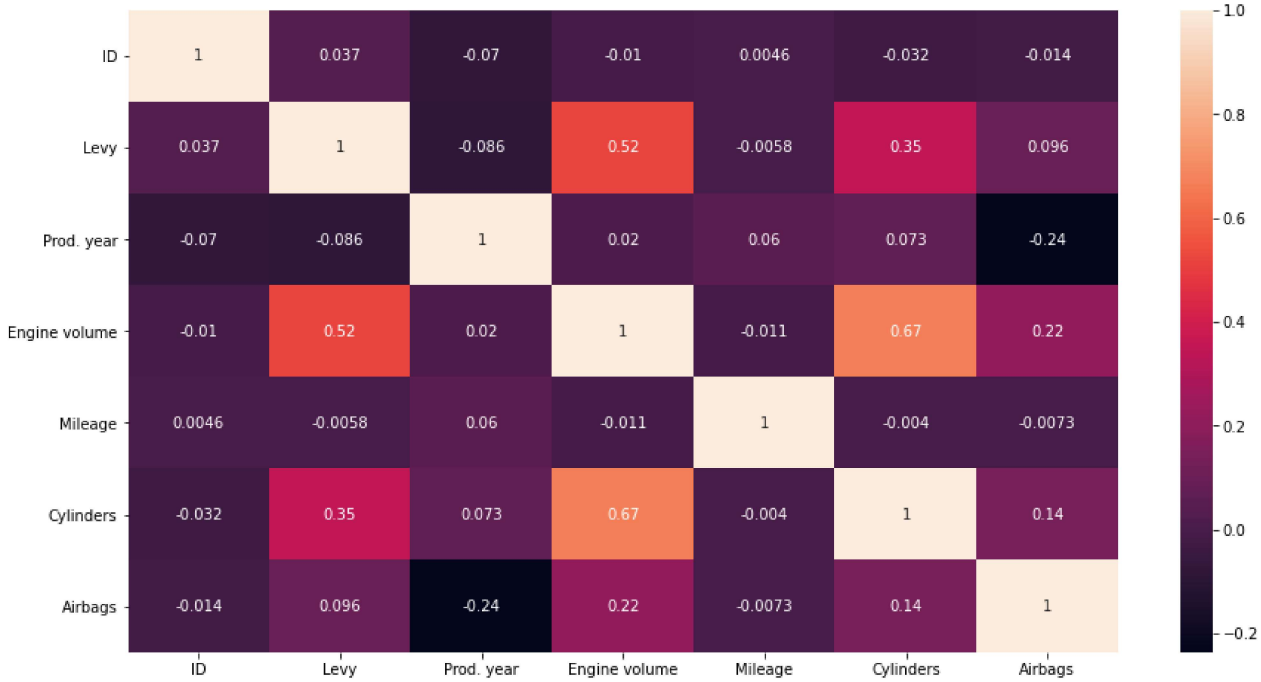
We can see that there are no missing values

In [63]:

```
sns.heatmap(df_merge.corr(), cbar=True, annot=True)
```

Out[63]:

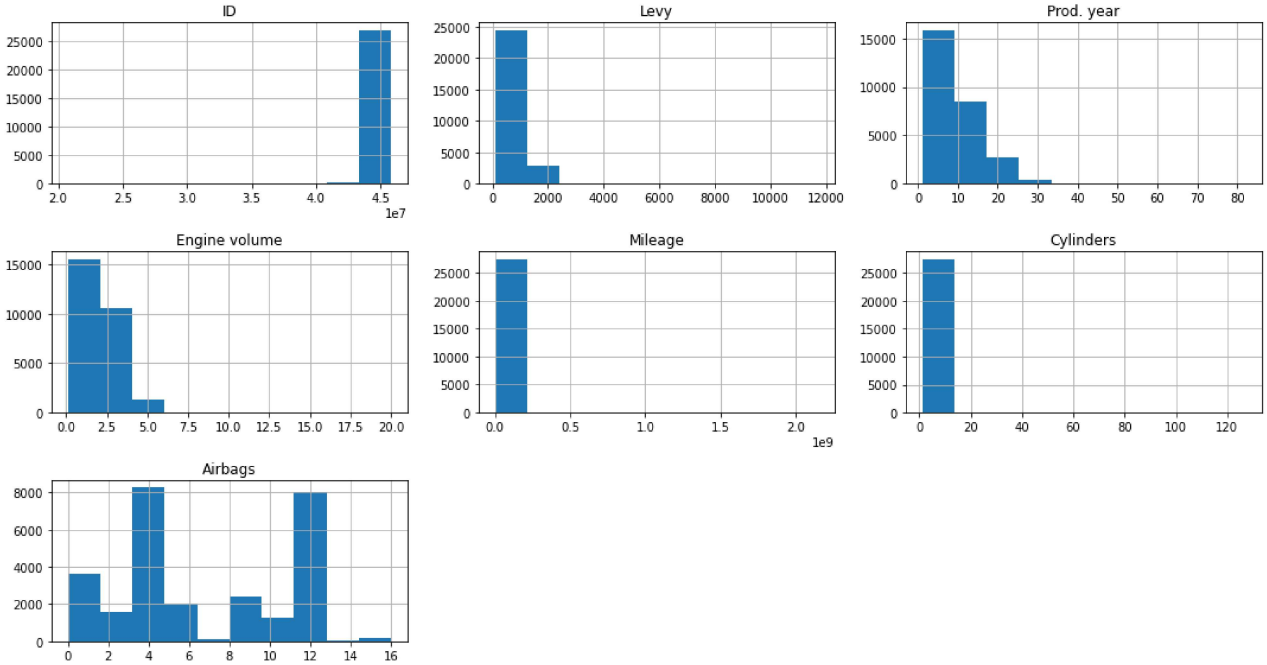
```
<AxesSubplot:>
```



We can see that 'Engine volume' is having high correlation with 'Cylinders' and 'Levy' columns.

In [64]:

```
#distribution of numeric variables
df_merge.hist()
plt.tight_layout()
plt.show()
```



We can see that 'Prod. year','Levy' and 'Engine volume' columns are right skewed.

In [65]:

```
#shapiro test to check the skewness of the target variable
from scipy.stats import shapiro
x = shapiro(y)
if x[1] <= 0:
    print('Negatively skewed')
else:
```

```
    print('Positively Skewed')
```

```
Negatively skewed
```

In [66]:

```
#As from the shapiro test we can see that 'Price' column is negatively skewed we need t
y = np.log(y)
```

Building the model

In [67]:

```
categ = df_merge.select_dtypes(include='object')
num = df_merge.select_dtypes(include = np.number)
```

In [68]:

```
#getting dummies for the categorical variables
cat_dummies = pd.get_dummies(categ,drop_first=True)
```

In [69]:

```
#creating the final dataset
df_final = pd.concat([num,cat_dummies], axis=1)
```

In [70]:

```
#checking the shape of the final dataset
df_final.shape
```

Out[70]:

```
(27482, 2094)
```

In [71]:

```
#segregating the training and test data before model building
train_data = df_final.iloc[:19237]
train_data.shape
```

Out[71]:

```
(19237, 2094)
```

In [72]:

```
test_data = df_final.iloc[19237:]
test_data.shape
```

Out[72]:

```
(8245, 2094)
```

In [75]:

```
#splitting the data into test and train
X = train_data
Y=y

X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3, random_state=10
```

In [76]:

```
#Randomized Search CV for searching the best parameters

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
```

```
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [77]:

```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_f
eatures': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 30], 'min_samples_split':
[2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

In [78]:

```
rf_model = RandomForestRegressor()
```

In [79]:

```
rf_random_model = RandomizedSearchCV(estimator = rf_model, param_distributions = random
```

In [81]:

```
rf_random_model.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_est
imators=900; total time=  25.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_est
imators=900; total time=  25.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_est
imators=900; total time=  26.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_est
imators=900; total time=  25.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_est
imators=900; total time=  25.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_es
timators=1100; total time=  51.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_es
timators=1100; total time=  50.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_es
timators=1100; total time=  49.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_es
timators=1100; total time=  51.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_es
timators=1100; total time=  48.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_e
stimators=300; total time= 2.1min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_e
stimators=300; total time= 2.1min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_e
stimators=300; total time= 2.2min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_e
```

```
stimators=300; total time= 2.1min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_e
stimators=300; total time= 2.1min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_est
imators=400; total time= 3.1min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_est
imators=400; total time= 2.2min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_est
imators=400; total time= 2.3min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_est
imators=400; total time= 2.3min
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_est
imators=400; total time= 2.2min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_es
timators=700; total time= 3.9min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_es
timators=700; total time= 3.9min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_es
timators=700; total time= 3.9min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_es
timators=700; total time= 4.9min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_es
timators=700; total time= 5.4min
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_est
imators=1000; total time= 2.6min
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_est
imators=1000; total time= 2.5min
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_est
imators=1000; total time= 2.5min
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_est
imators=1000; total time= 2.7min
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_est
imators=1000; total time= 2.3min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_es
timators=1100; total time=  11.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_es
timators=1100; total time=  12.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_es
timators=1100; total time=  11.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_es
timators=1100; total time=  12.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_es
timators=1100; total time=  12.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_es
timators=300; total time=  13.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_es
timators=300; total time=  14.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_es
timators=300; total time=  12.8s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_es
timators=300; total time=  13.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_es
timators=300; total time=  13.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_est
imators=700; total time=   8.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_est
imators=700; total time=   8.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_est
imators=700; total time=   8.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_est
```

```
imators=700; total time=   8.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_est
imators=700; total time=   8.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_es
timators=700; total time= 4.4min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_es
timators=700; total time= 4.3min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_es
timators=700; total time= 4.7min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_es
timators=700; total time= 4.2min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_es
timators=700; total time= 4.3min
```

Out[81]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [82]:

```
#getting the best parameters
rf_random_model.best_params_
```

Out[82]:

```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

Feature selection using extra tree regressor

Since we need to find features to train the model so that it neither gets underfitted or overfitted, we use feature selection technique.The best feature selection technique that worked for this problem statement is using extra tree regressor.

In [83]:

```
reg= ExtraTreesRegressor()
reg.fit(X_train,y_train)
ExtraTreesRegressor()
```

Out[83]:

```
ExtraTreesRegressor()
```

In [185…

```
#finding important features
feat_importances = pd.Series(reg.feature_importances_, index=X_train.columns)
```

In [214…

```
pd.DataFrame(feat_importances.nlargest(30)).index
```

Out[214…

```
Index(['Airbags', 'Mileage', 'Prod. year', 'ID', 'Gear box type_Tiptronic',
       'Leather interior_Yes', 'Levy', 'Fuel type_Diesel', 'Engine volume',
```

```
       'Manufacturer_HYUNDAI', 'Fuel type_Hybrid', 'Color_White',
       'Color_Black', 'Drive wheels_Front', 'Model_FIT', 'Color_Grey',
       'Color_Silver', 'Cylinders', 'Wheel_Right-hand drive', 'Category_Sedan',
       'Manufacturer_TOYOTA', 'Category_Jeep', 'Gear box type_Variator',
       'Manufacturer_SSANGYONG', 'Fuel type_Petrol', 'Drive wheels_Rear',
       'Model_Prius', 'Color_Blue', 'Category_Hatchback',
       'Manufacturer_LEXUS'],
      dtype='object')
```

In [98]:

```
#instantiating the randomforest regressor using the best parameters
mod4 = RandomForestRegressor(n_estimators= 1000, max_depth= 25,
 max_features= 'sqrt',
 min_samples_leaf=1,
 min_samples_split = 2
 )
```

In [220…

```
X1 = train_data[['Airbags', 'Mileage', 'Prod. year', 'ID', 'Gear box type_Tiptronic',
       'Leather interior_Yes', 'Levy', 'Fuel type_Diesel', 'Engine volume',
       'Manufacturer_HYUNDAI', 'Fuel type_Hybrid', 'Color_White',
       'Color_Black', 'Drive wheels_Front', 'Model_FIT', 'Color_Grey',
       'Color_Silver', 'Cylinders', 'Wheel_Right-hand drive', 'Category_Sedan',
       'Manufacturer_TOYOTA', 'Category_Jeep', 'Gear box type_Variator',
       'Manufacturer_SSANGYONG', 'Fuel type_Petrol', 'Drive wheels_Rear',
       'Model_Prius']]
y1=y

X1_train, X1_test, y1_train, y1_test = train_test_split(X1,y1, test_size=0.3, random_st
```

In [221…

```
#fitting the model
model = mod4.fit(X1_train, y1_train)
```

In [222…

```
#predicting the data
y_predict=model.predict(X1_test)
```

In [223…

```
#calculating the RMLSE score
RMLSE=np.sqrt(mean_squared_log_error(np.exp(y1_test),np.exp(y_predict)))
```

In [224…

```
#Printing the RMLSE score
RMLSE
```

Out[224…

```
0.8421683069074718
```

Finding best features using random forest regressor

In [133…

```
mod3 = RandomForestRegressor(n_estimators= 1000, max_depth= 25,
 max_features= 'sqrt',
 min_samples_leaf=1,
```

```
 min_samples_split = 2
 )
```

In [135…

```
model_random = mod3.fit(X_train, y_train)
```

In [136…

```
feat_importances = pd.Series(model_random.feature_importances_, index=X_train.columns)
```

In [138…

```
pd.DataFrame(feat_importances.nlargest(50)).index
```

Out[138…

```
Index(['Airbags', 'Mileage', 'Prod. year', 'ID', 'Gear box type_Tiptronic',
       'Leather interior_Yes', 'Levy', 'Fuel type_Diesel', 'Engine volume',
       'Manufacturer_HYUNDAI', 'Fuel type_Hybrid', 'Color_White',
       'Color_Black', 'Drive wheels_Front', 'Model_FIT', 'Color_Grey',
       'Color_Silver', 'Cylinders', 'Wheel_Right-hand drive', 'Category_Sedan',
       'Manufacturer_TOYOTA', 'Category_Jeep', 'Gear box type_Variator',
       'Manufacturer_SSANGYONG', 'Fuel type_Petrol', 'Drive wheels_Rear',
       'Model_Prius', 'Color_Blue', 'Category_Hatchback', 'Manufacturer_LEXUS',
       'Model_GX 470', 'Model_X5', 'Manufacturer_MERCEDES-BENZ',
       'Gear box type_Manual', 'Fuel type_LPG', 'Color_Red', 'Model_Camry',
       'Model_Land Cruiser Prado', 'Model_Highlander', 'Model_E 350',
       'Model_Jetta', 'Doors_04', 'Model_RX 450', 'Model_GX 460',
       'Manufacturer_BMW', 'Model_ML 350', 'Model_E 300', 'Model_Prius C',
       'Model_Sonata', 'Manufacturer_AUDI'],
      dtype='object')
```

In [225…

```
X2=train_data[['Airbags', 'Mileage', 'Prod. year', 'ID', 'Gear box type_Tiptronic',
       'Leather interior_Yes', 'Levy', 'Fuel type_Diesel', 'Engine volume',
       'Manufacturer_HYUNDAI', 'Fuel type_Hybrid', 'Color_White',
       'Color_Black', 'Drive wheels_Front', 'Model_FIT', 'Color_Grey',
       'Color_Silver', 'Cylinders', 'Wheel_Right-hand drive', 'Category_Sedan',
       'Manufacturer_TOYOTA', 'Category_Jeep', 'Gear box type_Variator',
       'Manufacturer_SSANGYONG', 'Fuel type_Petrol', 'Drive wheels_Rear',
       'Model_Prius', 'Color_Blue', 'Category_Hatchback']]
Y2=y

X2_train, X2_test, y2_train, y2_test = train_test_split(X2,Y2, test_size=0.3, random_st
```

In [226…

```
model1 = mod4.fit(X2_train, y2_train)
```

In [227…

```
y_pred=model1.predict(X2_test)
```

In [228…

```
RMLSE1=np.sqrt(mean_squared_log_error(np.exp(y2_test),np.exp(y_pred)))
```

In [229…

```
RMLSE1
```

Out[229…

```
0.842671518662218
```

Predicting the 'Price' for test dataset

In [230…

```
#selecting the columns from test_data
z=test_data[['Airbags', 'Mileage', 'Prod. year', 'ID', 'Gear box type_Tiptronic',
       'Leather interior_Yes', 'Levy', 'Fuel type_Diesel', 'Engine volume',
       'Manufacturer_HYUNDAI', 'Fuel type_Hybrid', 'Color_White',
       'Color_Black', 'Drive wheels_Front', 'Model_FIT', 'Color_Grey',
       'Color_Silver', 'Cylinders', 'Wheel_Right-hand drive', 'Category_Sedan',
       'Manufacturer_TOYOTA', 'Category_Jeep', 'Gear box type_Variator',
       'Manufacturer_SSANGYONG', 'Fuel type_Petrol', 'Drive wheels_Rear',
       'Model_Prius']]
```

In [156…

```
#predicting the 'Price' column for test data
test_data_predicted = model.predict(z)
#converting the data into exponential data
dt2 = pd.DataFrame(np.exp(test_data_predicted))
#renaming the column
dt2.rename(columns={0:'Price'},inplace=True)
```

In [157…

```
#submission
dt2.to_csv("current.csv",index=False)
```