In [166…
```
# importing necessary libraries
import matplotlib.pyplot as plt # matplotlib for plotting and figures plot
import pandas as pd  # pandas for data frame
import seaborn as sns # seaborn for heatmap, countplot, boxplot
import numpy as np # numpy for array, matrix calculations
import warnings
warnings.filterwarnings('ignore') # to ignore all warning
```

In [167…
```
#__version__ to check versions of libraries
import matplotlib
np.__version__, pd.__version__, sns.__version__, matplotlib.__version__
```

Out[167…
```
('2.1.3', '2.2.3', '0.13.2', '3.10.0')
```

In [168…
```
# reading csv file through pandas
df = pd.read_csv('Cars.csv')
df
```

Out[168…

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	n
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	

8128 rows × 13 columns

In [169…

```
# printing 5 data from top
df.head()
```

Out[169…

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	milea
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	2 km
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21 km
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	1 km
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	2 km
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	1 km

In [170…

```
# printing shape (row, col)
df.shape
```

Out[170…

```
(8128, 13)
```

In [171…

```
# dataframe describing
df.describe()
```

Out[171…

	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

In [172…

```
# checking data types and null count information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [173…

```
# checking all available columns in dataframe
df.columns
```

Out[173…

```
Index(['name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_type',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torque',
       'seats'],
      dtype='object')
```

In [174…

```
# renaming all columns to work with naming conventions easily
df.rename(columns = {
    'name': 'name',
    'year': 'year',
    'selling_price': 'sell_price',
    'km_driven': 'km',
    'fuel': 'fuel',
    'seller_type': 'sell_type',
    'transmission': 'transmission',
    'owner': 'owner',
    'mileage': 'mileage',
    'engine': 'engine',
    'max_power': 'max_power',
    'torque': 'torque',
    'seats': 'seats'
}, inplace=True)

df.columns
```

Out[174…

```
Index(['name', 'year', 'sell_price', 'km', 'fuel', 'sell_type', 'transmission',
       'owner', 'mileage', 'engine', 'max_power', 'torque', 'seats'],
      dtype='object')
```

In [175…

```
# printing only owner column from dataframe
df['owner']
```

Out[175…
```
0                 First Owner
1                Second Owner
2                 Third Owner
3                 First Owner
4                 First Owner
                 ...
8123              First Owner
8124     Fourth & Above Owner
8125              First Owner
8126              First Owner
8127              First Owner
Name: owner, Length: 8128, dtype: object
```

In [176…
```
#printing unique names of owner column without repeatation
df['owner'].unique()
```

Out[176…
```
array(['First Owner', 'Second Owner', 'Third Owner',
       'Fourth & Above Owner', 'Test Drive Car'], dtype=object)
```

In [177…
```
# replacing string with numeric value to predict
df['owner'] = df['owner'].replace({
    'First Owner': 1,
    'Second Owner': 2,
    'Third Owner': 3,
    'Fourth & Above Owner': 4,
    'Test Drive Car': 5
})
```

In [178…
```
# checking whole dataframe
df
```

Out[178...

	name	year	sell_price	km	fuel	sell_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	1	23.4 kmpl
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	2	21.14 kmpl
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	3	17.7 kmpl
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	1	23.0 kmpl
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	1	16.1 kmpl
...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	1	18.5 kmpl
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	4	16.8 kmpl
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	1	19.3 kmpl
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57 kmpl
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57 kmpl

8128 rows × 13 columns

In [179...

```
df['fuel'].unique()
```

Out[179...

```
array(['Diesel', 'Petrol', 'LPG', 'CNG'], dtype=object)
```

In [180…

```
# deleting rows with name LPG
df = df[df['fuel'] != 'LPG']
df['fuel']
```

Out[180…

```
0       Diesel
1       Diesel
2       Petrol
3       Diesel
4       Petrol
         ...
8123    Petrol
8124    Diesel
8125    Diesel
8126    Diesel
8127    Diesel
Name: fuel, Length: 8090, dtype: object
```

In [181…

```
df['fuel'].unique()
```

Out[181…

```
array(['Diesel', 'Petrol', 'CNG'], dtype=object)
```

In [182…

```
df = df[df['fuel'] != 'CNG']
df['fuel'].unique()
```

Out[182…

```
array(['Diesel', 'Petrol'], dtype=object)
```

In [183…

```
df
```

Out[183…

	name	year	sell_price	km	fuel	sell_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	1	23.4 kmpl
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	2	21.14 kmpl
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	3	17.7 kmpl
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	1	23.0 kmpl
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	1	16.1 kmpl
...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	1	18.5 kmpl
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	4	16.8 kmpl
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	1	19.3 kmpl
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57 kmpl
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57 kmpl

8033 rows × 13 columns

In [184…

```
df['mileage']
```

Out[184…

```
0        23.4 kmpl
1       21.14 kmpl
2        17.7 kmpl
3        23.0 kmpl
4        16.1 kmpl
          ...
8123     18.5 kmpl
8124     16.8 kmpl
8125     19.3 kmpl
8126    23.57 kmpl
8127    23.57 kmpl
Name: mileage, Length: 8033, dtype: object
```

In [185…

```
# spliting string and taking first index of the splited string
df['mileage'] = df['mileage'].str.split(' ').str[0]
df['mileage']
```

Out[185…

```
0        23.4
1       21.14
2        17.7
3        23.0
4        16.1
        ...
8123     18.5
8124     16.8
8125     19.3
8126    23.57
8127    23.57
Name: mileage, Length: 8033, dtype: object
```

In [186…

```
# checking data type of mileage column
df['mileage'].dtype
```

Out[186…

```
dtype('O')
```

In [187…

```
# changing mileage column from object to float
df['mileage'] = df['mileage'].astype(float)
df['mileage'].dtype
```

Out[187…

```
dtype('float64')
```

In [188…

```
df['engine']
```

Out[188…

```
0       1248 CC
1       1498 CC
2       1497 CC
3       1396 CC
4       1298 CC
         ...
8123    1197 CC
8124    1493 CC
8125    1248 CC
8126    1396 CC
8127    1396 CC
Name: engine, Length: 8033, dtype: object
```

In [189…
```
df['engine'] = df['engine'].str.split(' ').str[0]
df['engine']
```

Out[189…
```
0       1248
1       1498
2       1497
3       1396
4       1298
        ...
8123    1197
8124    1493
8125    1248
8126    1396
8127    1396
Name: engine, Length: 8033, dtype: object
```

In [190…
```
df['engine'] = df['engine'].astype(float)   # changing data type of engine from obj
```

In [191…
```
df
```

Out[191…

	name	year	sell_price	km	fuel	sell_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	1	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	2	21.14
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	3	17.70
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	1	23.00
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	1	16.10
...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	1	18.50
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	4	16.80
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	1	19.30
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57

8033 rows × 13 columns

In [192…

```
df['max_power'] =  df['max_power'].str.split(' ').str[0]
df['max_power']
```

Out[192…

```
0          74
1      103.52
2          78
3          90
4        88.2
        ...
8123    82.85
8124      110
8125     73.9
8126       70
8127       70
Name: max_power, Length: 8033, dtype: object
```

In [193…

```
df['max_power'] = df['max_power'].astype(float)
```

In [194…

```
df['max_power'].dtype
```

Out[194…

```
dtype('float64')
```

In [195…

```
df
```

Out[195…

	name	year	sell_price	km	fuel	sell_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	1	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	2	21.14
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	3	17.70
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	1	23.00
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	1	16.10
...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	1	18.50
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	4	16.80
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	1	19.30
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	1	23.57

8033 rows × 13 columns

In [196…

```
df['name'] = df['name'].str.split(' ').str[0]   #splitting and taking first index 0
df['name']
```

Out[196…

```
0        Maruti
1         Skoda
2         Honda
3       Hyundai
4        Maruti
         ...
8123    Hyundai
8124    Hyundai
8125     Maruti
8126       Tata
8127       Tata
Name: name, Length: 8033, dtype: object
```

In [197…

```
df.rename(columns={'name':'brand'}, inplace=True)   # changing naming convention fr
df
```

Out[197…

	brand	year	sell_price	km	fuel	sell_type	transmission	owner	mileage	e
0	Maruti	2014	450000	145500	Diesel	Individual	Manual	1	23.40	
1	Skoda	2014	370000	120000	Diesel	Individual	Manual	2	21.14	
2	Honda	2006	158000	140000	Petrol	Individual	Manual	3	17.70	
3	Hyundai	2010	225000	127000	Diesel	Individual	Manual	1	23.00	
4	Maruti	2007	130000	120000	Petrol	Individual	Manual	1	16.10	
...	...	...	...	...	...	...	...	...	...	
8123	Hyundai	2013	320000	110000	Petrol	Individual	Manual	1	18.50	
8124	Hyundai	2007	135000	119000	Diesel	Individual	Manual	4	16.80	
8125	Maruti	2009	382000	120000	Diesel	Individual	Manual	1	19.30	
8126	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	
8127	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	

8033 rows × 13 columns

In [198…

```
df.drop(columns=['torque'], inplace=True) # dropping column torque from dataframe
df
```

Out[198…

	brand	year	sell_price	km	fuel	sell_type	transmission	owner	mileage	e
0	Maruti	2014	450000	145500	Diesel	Individual	Manual	1	23.40	
1	Skoda	2014	370000	120000	Diesel	Individual	Manual	2	21.14	
2	Honda	2006	158000	140000	Petrol	Individual	Manual	3	17.70	
3	Hyundai	2010	225000	127000	Diesel	Individual	Manual	1	23.00	
4	Maruti	2007	130000	120000	Petrol	Individual	Manual	1	16.10	
...	...	...	...	...	...	...	...	...	...	
8123	Hyundai	2013	320000	110000	Petrol	Individual	Manual	1	18.50	
8124	Hyundai	2007	135000	119000	Diesel	Individual	Manual	4	16.80	
8125	Maruti	2009	382000	120000	Diesel	Individual	Manual	1	19.30	
8126	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	
8127	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	

8033 rows × 12 columns

In [199…

```
df['owner'].unique()
```

Out[199…

```
array([1, 2, 3, 4, 5])
```

In [200…

```
df['owner'].dtype
```

Out[200…

```
dtype('int64')
```

In [201…

```
df = df[df['owner'] != 5]    # deleting row of owner column with integer 5
df['owner']
```

Out[201…

```
0       1
1       2
2       3
3       1
4       1
       ..
8123    1
8124    4
8125    1
8126    1
8127    1
Name: owner, Length: 8028, dtype: int64
```

In [202…

```
df['owner'].unique()
```

Out[202…

```
array([1, 2, 3, 4])
```

In [203…

```
df['sell_price']
```

Out[203…

```
0       450000
1       370000
2       158000
3       225000
4       130000
         ...
8123    320000
8124    135000
8125    382000
8126    290000
8127    290000
Name: sell_price, Length: 8028, dtype: int64
```

In [204…

```
# x - features (x1, x2, x3)
# y - label (price)
#scaler transform
y = np.log(df['sell_price'])
y
```

Out[204…

```
0       13.017003
1       12.821258
2       11.970350
3       12.323856
4       11.775290
          ...
8123    12.676076
8124    11.813030
8125    12.853176
8126    12.577636
8127    12.577636
Name: sell_price, Length: 8028, dtype: float64
```

In [205…

```
df
```

Out[205…

	brand	year	sell_price	km	fuel	sell_type	transmission	owner	mileage	e
0	Maruti	2014	450000	145500	Diesel	Individual	Manual	1	23.40	
1	Skoda	2014	370000	120000	Diesel	Individual	Manual	2	21.14	
2	Honda	2006	158000	140000	Petrol	Individual	Manual	3	17.70	
3	Hyundai	2010	225000	127000	Diesel	Individual	Manual	1	23.00	
4	Maruti	2007	130000	120000	Petrol	Individual	Manual	1	16.10	
...	...	...	...	...	...	...	...	...	...	
8123	Hyundai	2013	320000	110000	Petrol	Individual	Manual	1	18.50	
8124	Hyundai	2007	135000	119000	Diesel	Individual	Manual	4	16.80	
8125	Maruti	2009	382000	120000	Diesel	Individual	Manual	1	19.30	
8126	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	
8127	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	

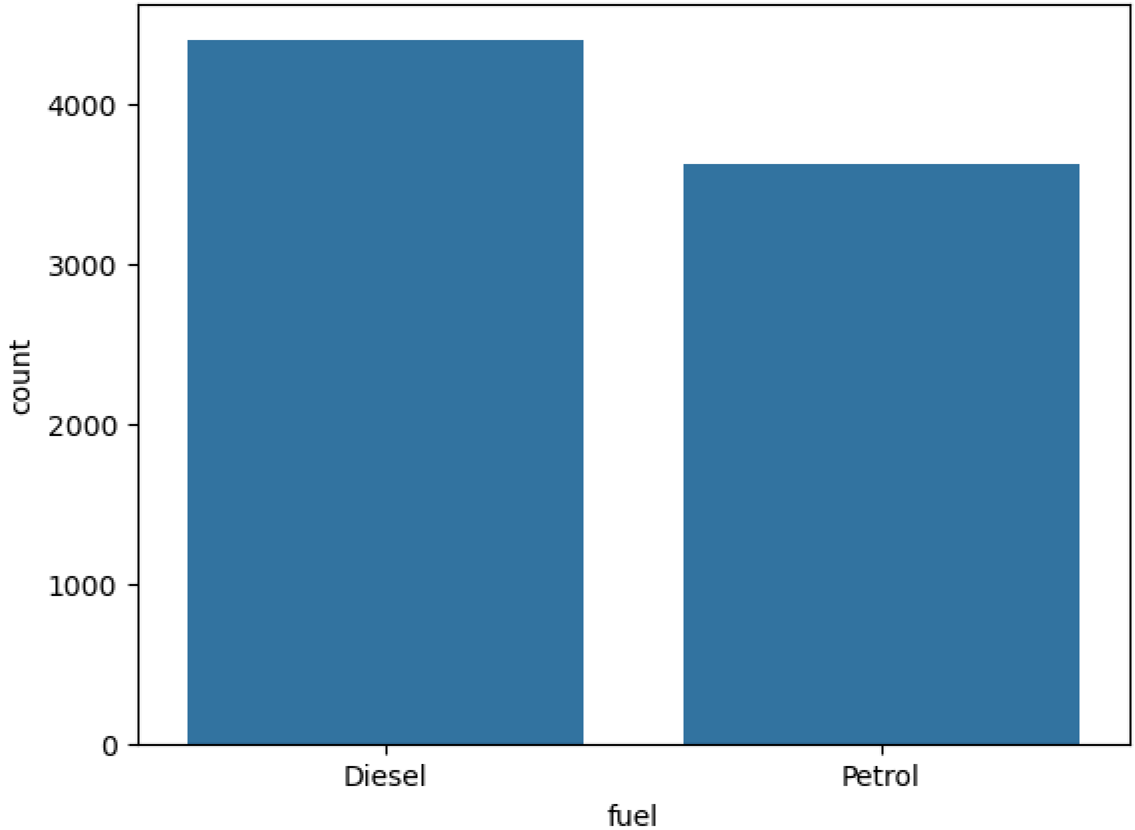
8028 rows × 12 columns

In [206…

```
# checking count of the dataframe fuel column using seaborn countplot
sns.countplot(data = df, x = 'fuel')
```

Out[206…

```
<Axes: xlabel='fuel', ylabel='count'>
```

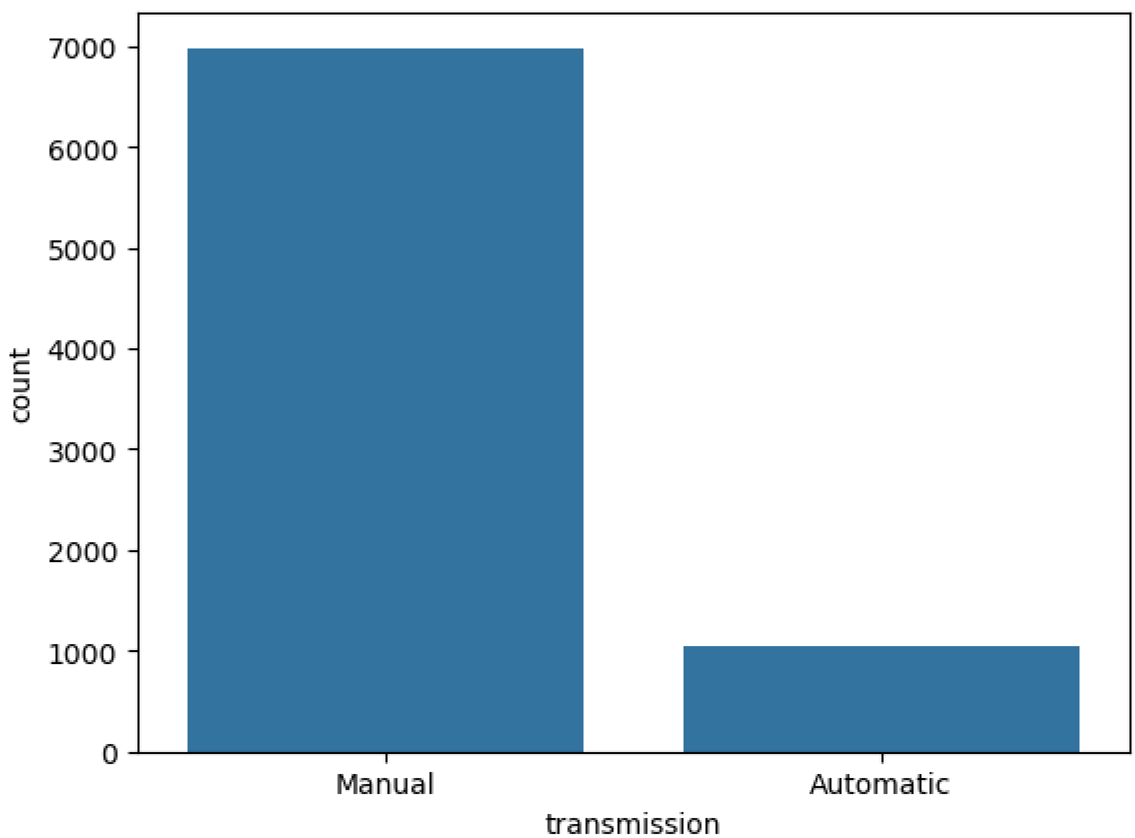


In [207…

```
sns.countplot(data = df, x = 'transmission')
```

Out[207…

```
<Axes: xlabel='transmission', ylabel='count'>
```

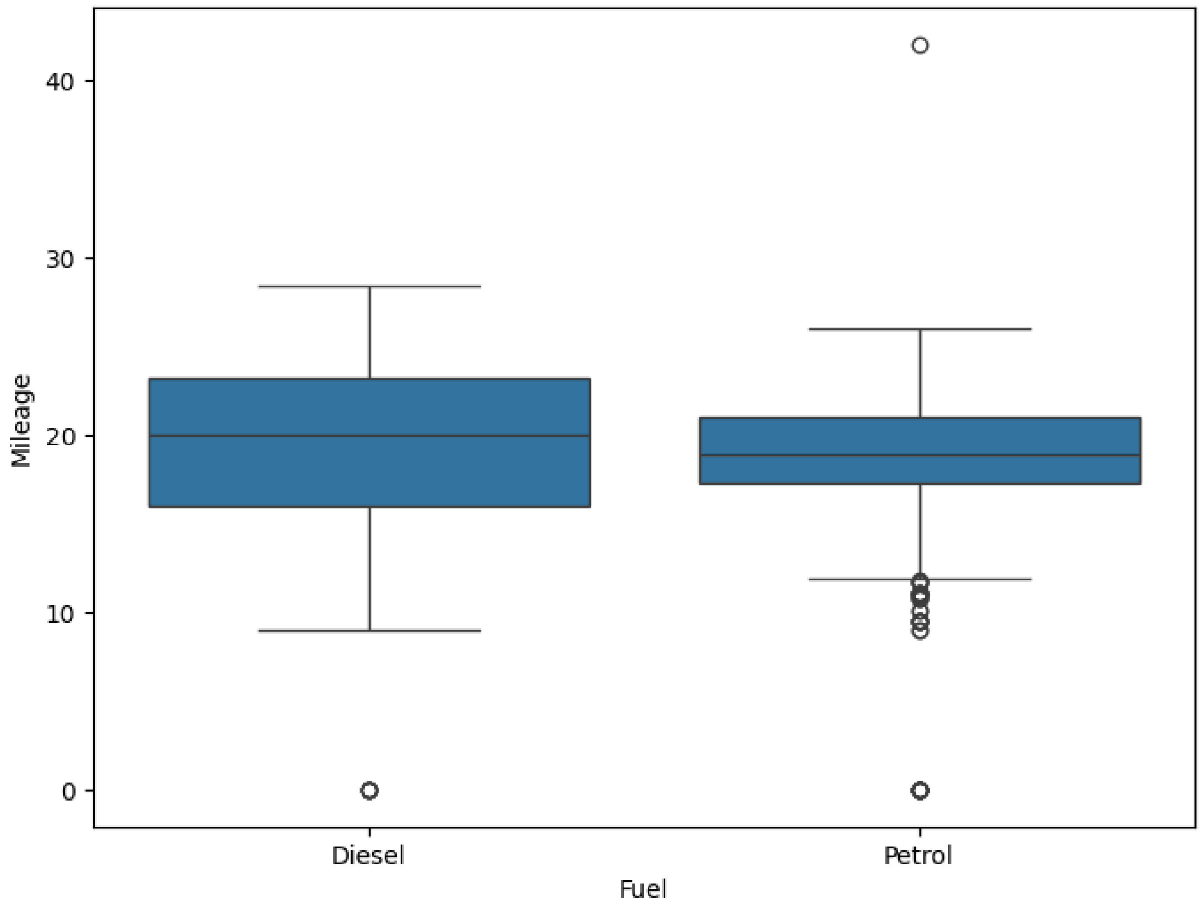


In [208…

```
# providing figure size to display
plt.figure(figsize=(8,6))
sns.boxplot(x = df['fuel'], y= df['mileage'])   # displaying fuel and mileage colum
plt.ylabel('Mileage')
plt.xlabel('Fuel')
```

Out[208…

```
Text(0.5, 0, 'Fuel')
```

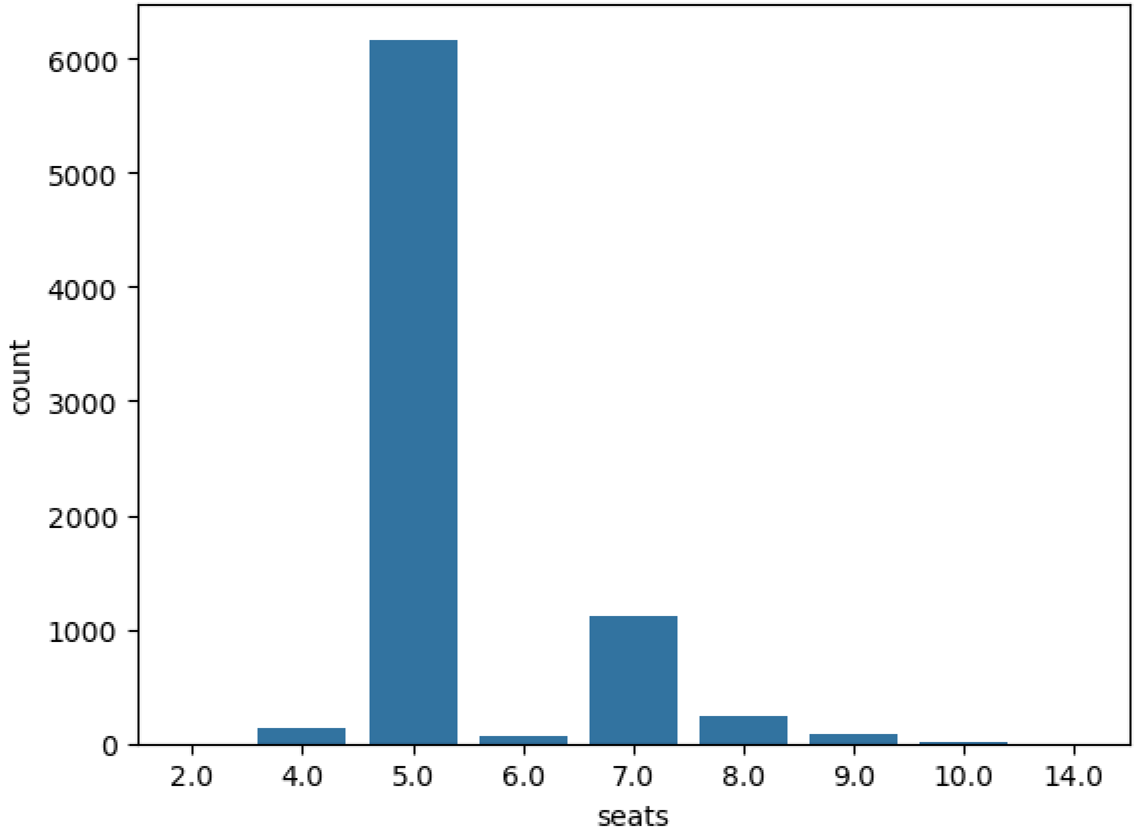


In [209…

```
sns.countplot(data = df, x = 'seats')
```

Out[209…

```
<Axes: xlabel='seats', ylabel='count'>
```

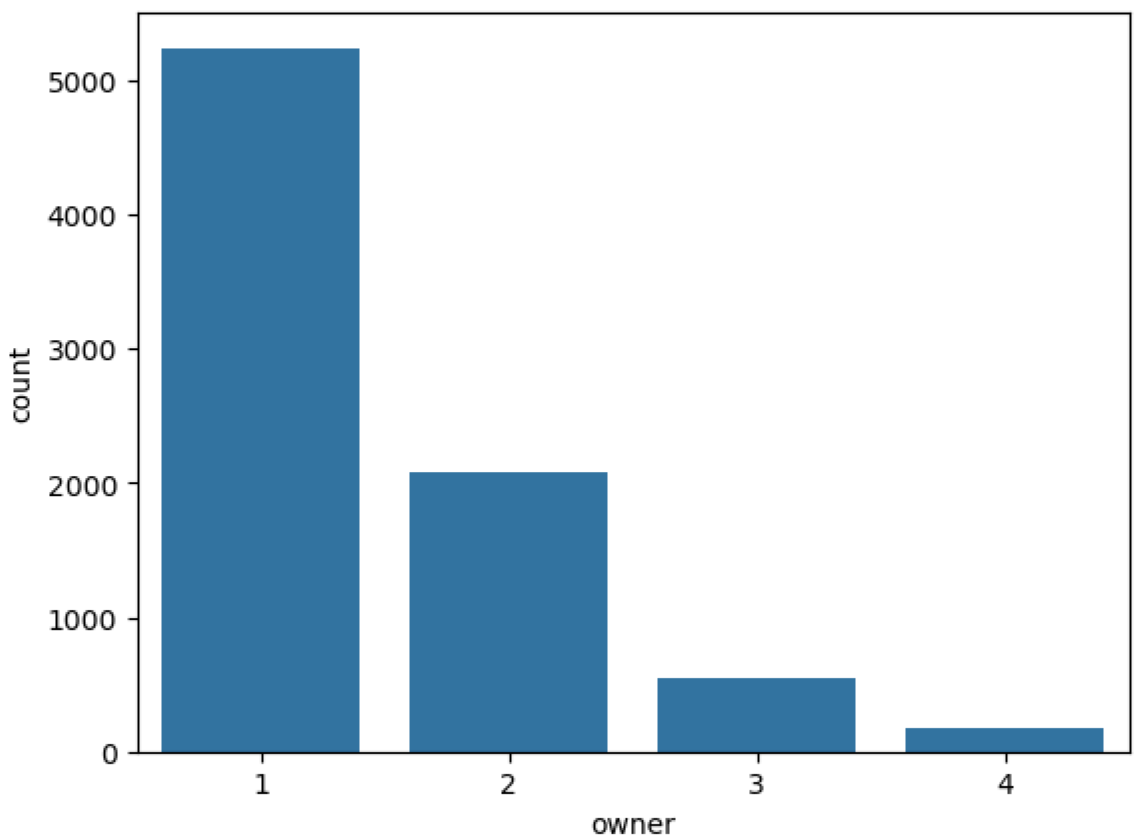


In [210…

```
sns.countplot(data = df, x = 'owner')
```

Out[210…

```
<Axes: xlabel='owner', ylabel='count'>
```



In [211…

```
df
```

Out[211…

	brand	year	sell_price	km	fuel	sell_type	transmission	owner	mileage	e
0	Maruti	2014	450000	145500	Diesel	Individual	Manual	1	23.40	
1	Skoda	2014	370000	120000	Diesel	Individual	Manual	2	21.14	
2	Honda	2006	158000	140000	Petrol	Individual	Manual	3	17.70	
3	Hyundai	2010	225000	127000	Diesel	Individual	Manual	1	23.00	
4	Maruti	2007	130000	120000	Petrol	Individual	Manual	1	16.10	
...	...	...	...	...	...	...	...	...	...	
8123	Hyundai	2013	320000	110000	Petrol	Individual	Manual	1	18.50	
8124	Hyundai	2007	135000	119000	Diesel	Individual	Manual	4	16.80	
8125	Maruti	2009	382000	120000	Diesel	Individual	Manual	1	19.30	
8126	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	
8127	Tata	2013	290000	25000	Diesel	Individual	Manual	1	23.57	

8028 rows × 12 columns

In [212…

```
print(df['fuel'].unique())
print(df['transmission'].unique())
print(df['brand'].unique())
```

```
['Diesel' 'Petrol']
['Manual' 'Automatic']
['Maruti' 'Skoda' 'Honda' 'Hyundai' 'Toyota' 'Ford' 'Renault' 'Mahindra'
 'Tata' 'Chevrolet' 'Fiat' 'Datsun' 'Jeep' 'Mercedes-Benz' 'Mitsubishi'
 'Audi' 'Volkswagen' 'BMW' 'Nissan' 'Lexus' 'Jaguar' 'Land' 'MG' 'Volvo'
 'Daewoo' 'Kia' 'Force' 'Ambassador' 'Ashok' 'Isuzu' 'Opel' 'Peugeot']
```

In [213…

```
# using scikit learn preprocessing LabelEncoder to change string data to numberic a
from sklearn.preprocessing import LabelEncoder
import pickle # import pickle to save trained model and use easily when necessary

le = LabelEncoder()

df['fuel'] = le.fit_transform(df['fuel'])
print("Fuel mapping:", dict(zip(le.classes_, le.transform(le.classes_))))   # chang

df['transmission'] = le.fit_transform(df['transmission'])
print("Transmission mapping:", dict(zip(le.classes_, le.transform(le.classes_))))

df['sell_type'] = le.fit_transform(df['sell_type'])
print("Sell Type mapping:", dict(zip(le.classes_, le.transform(le.classes_))))

df['brand'] = le.fit_transform(df['brand'])
print("Brand mapping:", dict(zip(le.classes_, le.transform(le.classes_))))
```

```
# le_brand = LabelEncoder()
# le_fuel = LabelEncoder()
# le_sell = LabelEncoder()

# 0 = Diesel, 1 = Petrol (because fit-transform transforms value as per ascending o
# df['fuel'] = le.fit_transform(df['fuel'])
# df['transmission'] = le.fit_transform(df['transmission'])
# df['sell_type'] = le.fit_transform(df['sell_type'])
# df['brand'] = le.fit_transform(df['brand'])
# # changing variables to numeric type to make sure column values work in predict c
# print(df['fuel'].unique())
# print(df['transmission'].unique())
# print(df['sell_type'].unique())
# print(df['brand'].unique())


# #load Model
# pickle.dump(le_brand, open("model/le_brand.pkl", "wb"))
# pickle.dump(le_fuel, open("model/le_fuel.pkl", "wb"))
# pickle.dump(le_sell, open("model/le_sell.pkl", "wb"))
```

```
Fuel mapping: {'Diesel': np.int64(0), 'Petrol': np.int64(1)}
Transmission mapping: {'Automatic': np.int64(0), 'Manual': np.int64(1)}
Sell Type mapping: {'Dealer': np.int64(0), 'Individual': np.int64(1), 'Trustmark Dea
ler': np.int64(2)}
Brand mapping: {'Ambassador': np.int64(0), 'Ashok': np.int64(1), 'Audi': np.int64
(2), 'BMW': np.int64(3), 'Chevrolet': np.int64(4), 'Daewoo': np.int64(5), 'Datsun':
np.int64(6), 'Fiat': np.int64(7), 'Force': np.int64(8), 'Ford': np.int64(9), 'Hond
a': np.int64(10), 'Hyundai': np.int64(11), 'Isuzu': np.int64(12), 'Jaguar': np.int64
(13), 'Jeep': np.int64(14), 'Kia': np.int64(15), 'Land': np.int64(16), 'Lexus': np.i
nt64(17), 'MG': np.int64(18), 'Mahindra': np.int64(19), 'Maruti': np.int64(20), 'Mer
cedes-Benz': np.int64(21), 'Mitsubishi': np.int64(22), 'Nissan': np.int64(23), 'Ope
l': np.int64(24), 'Peugeot': np.int64(25), 'Renault': np.int64(26), 'Skoda': np.int6
4(27), 'Tata': np.int64(28), 'Toyota': np.int64(29), 'Volkswagen': np.int64(30), 'Vo
lvo': np.int64(31)}
```

In [214…

```
print(df['fuel'].unique())
print(df['transmission'].unique())
print(df['sell_type'].unique())
print(df['brand'].unique())
```

```
[0 1]
[1 0]
[1 0 2]
[20 27 10 11 29  9 26 19 28  4  7  6 14 21 22  2 30  3 23 17 13 16 18 31
  5 15  8  0  1 12 24 25]
```

In [215…

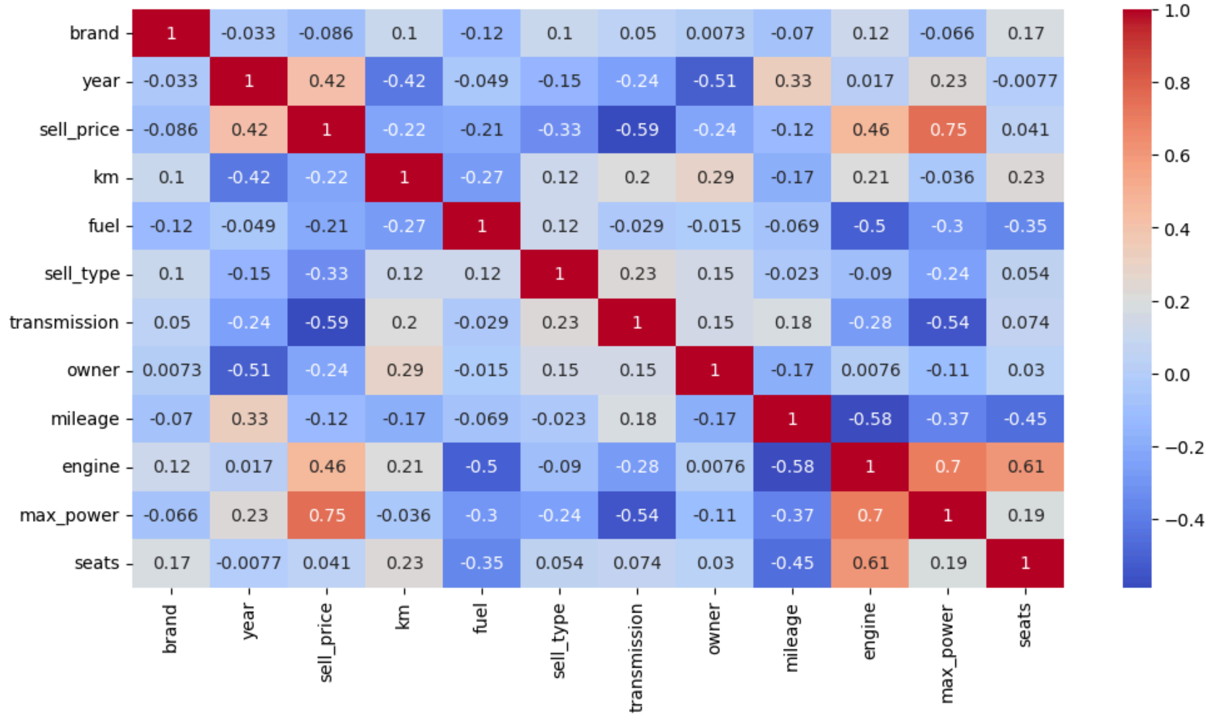
```
# df = df.drop(columns=['brand'])
```

In [216…

```
plt.figure(figsize=(12,6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm") # used heatmap to see correlati
```

Out[216…

```
<Axes: >
```



In [217…

```
df = df.drop(columns=['year','transmission', 'owner'])  # dropped columns after cor
df
```

Out[217…

	brand	sell_price	km	fuel	sell_type	mileage	engine	max_power	seats
0	20	450000	145500	0	1	23.40	1248.0	74.00	5.0
1	27	370000	120000	0	1	21.14	1498.0	103.52	5.0
2	10	158000	140000	1	1	17.70	1497.0	78.00	5.0
3	11	225000	127000	0	1	23.00	1396.0	90.00	5.0
4	20	130000	120000	1	1	16.10	1298.0	88.20	5.0
...	...	...	...	...	...	...	...	...	...
8123	11	320000	110000	1	1	18.50	1197.0	82.85	5.0
8124	11	135000	119000	0	1	16.80	1493.0	110.00	5.0
8125	20	382000	120000	0	1	19.30	1248.0	73.90	5.0
8126	28	290000	25000	0	1	23.57	1396.0	70.00	5.0
8127	28	290000	25000	0	1	23.57	1396.0	70.00	5.0

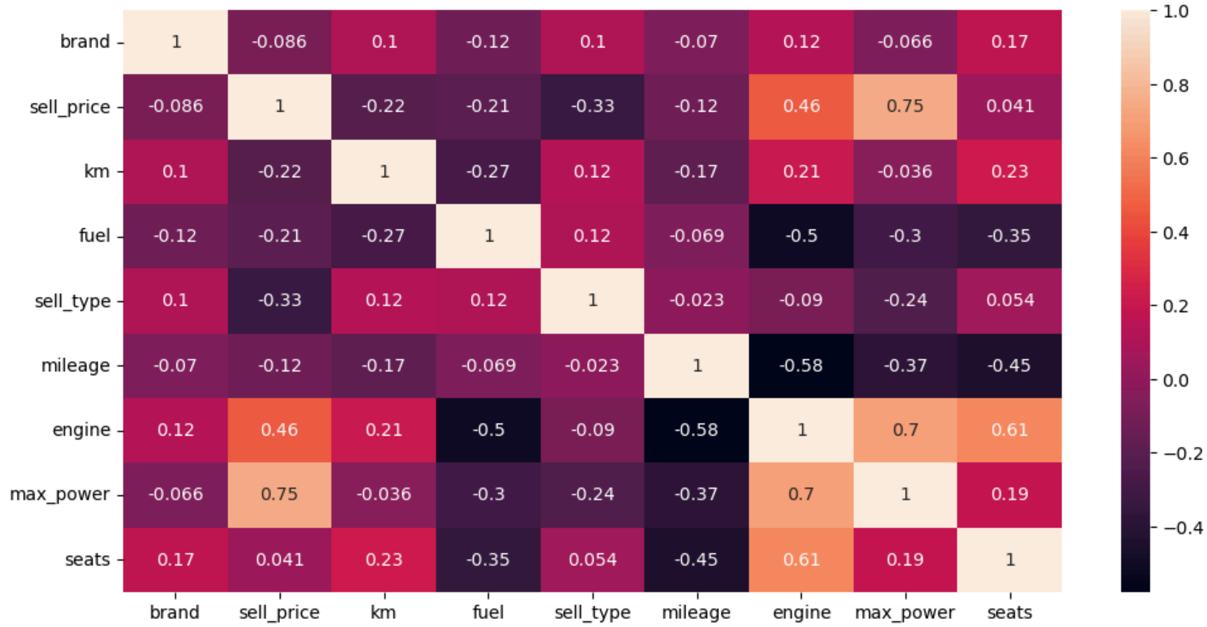
8028 rows × 9 columns

In [218…

```
plt.figure(figsize=(12,6))
sns.heatmap(df.corr(), annot=True)  # annot = annotations in figure
```

Out[218…

```
<Axes: >
```



In [219…

```
# Feature Engineering

X = df[['brand', 'km', 'fuel', 'sell_type', 'mileage', 'engine', 'seats','max_power
y = df['sell_price']     # y/Target
```

In [220…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =  train_test_split(X,y, test_size=0.3, random_sta
```

In [221…

```
# Preprocessing
# Checking if value is null
X_train[['brand', 'km', 'fuel', 'sell_type', 'mileage', 'engine', 'seats','max_powe
```

Out[221…

```
brand          0
km             0
fuel           0
sell_type      0
mileage      154
engine       154
seats        154
max_power    149
dtype: int64
```

In [222…

```
y_train.isna().sum()
```

Out[222…

```
np.int64(0)
```

In [223…

```
X_test.isna().sum()
```

Out[223…

```
brand         0
km            0
fuel          0
sell_type     0
mileage      60
engine       60
seats        60
max_power    59
dtype: int64
```

In [224…

```
# since many values null
# adding median outliers
X_train['mileage'].fillna(X_train['mileage'].median(), inplace=True)
X_train['engine'].fillna(X_train['engine'].median(), inplace=True)
X_train['seats'].fillna(X_train['seats'].median(), inplace=True)
X_train['max_power'].fillna(X_train['max_power'].median(), inplace=True)

X_test['mileage'].fillna(X_test['mileage'].median(), inplace=True)
X_test['engine'].fillna(X_test['engine'].median(), inplace=True)
X_test['seats'].fillna(X_test['seats'].median(), inplace=True)
X_test['max_power'].fillna(X_test['max_power'].median(), inplace=True)
```

In [225…

```
X_train[['brand', 'km', 'fuel', 'sell_type', 'mileage', 'engine', 'seats','max_powe
```

Out[225…

```
brand        0
km           0
fuel         0
sell_type    0
mileage      0
engine       0
seats        0
max_power    0
dtype: int64
```

In [226…

```
def outlier_count(col, data = X_train):

    # calculate your 25% quatile and 75% quatile
    q75, q25 = np.percentile(data[col], [75, 25])

    # calculate your inter quatile
    iqr = q75 - q25

    # min_val and max_val
    min_val = q25 - (iqr*1.5)
    max_val = q75 + (iqr*1.5)

    # count number of outliers, which are the data that are less than min_val or mo
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < min_val))[0])

    # calculate the percentage of the outliers
    outlier_percent = round(outlier_count/len(data[col])*100, 2)

    if(outlier_count > 0):
        print("\n"+15*'-' + col + 15*'-'+"\n")
```

```
        print('Number of outliers: {}'.format(outlier_count))
        print('Percent of data that is outlier: {}%'.format(outlier_percent))
```

In [227…

```
# calling outlier function to count outliers
for col in X_train.columns:
    outlier_count(col)
```

```
---------------km---------------

Number of outliers: 102
Percent of data that is outlier: 1.82%

---------------sell_type---------------

Number of outliers: 944
Percent of data that is outlier: 16.8%

---------------mileage---------------

Number of outliers: 14
Percent of data that is outlier: 0.25%

---------------engine---------------

Number of outliers: 836
Percent of data that is outlier: 14.88%

---------------seats---------------

Number of outliers: 1158
Percent of data that is outlier: 20.61%

---------------max_power---------------

Number of outliers: 409
Percent of data that is outlier: 7.28%
```

In [228…

```
# scikit learn preprocessing StandardScaler algorithm used to scale and transform
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

In [229…

```
print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of X_train:  (5619, 8)
Shape of X_test:  (2409, 8)
Shape of y_train:  (5619,)
Shape of y_test:  (2409,)
```

In [230…

```
from sklearn.linear_model import LinearRegression   # using linear regression algor
from sklearn.metrics import mean_squared_error, r2_score
```

```
 # using mean square error and r2 score to check error where MSE can be any number

lr = LinearRegression()
lr.fit(X_train, y_train)
yPred = lr.predict(X_test)

print("MSE: ", mean_squared_error(y_test, yPred))
print("r2: ", r2_score(y_test, yPred))
```

```
MSE:  238980166156.9503
r2:  0.6423908700337254
```

In [231…

```
# Cross Validation + Grid Search - Checking which model is suitable
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor


algorithms = [LinearRegression(), SVR(), KNeighborsRegressor(), DecisionTreeRegress
              RandomForestRegressor(n_estimators = 100, random_state = 0)]

# Models
algorithm_names = ["Linear Regression", "SVR", "KNeighbors Regressor", "Decision-Tr
```

In [232…

```
# lowest the mean number greater accuracy algorithm
from sklearn.model_selection import KFold, cross_val_score

#lists for keeping mse
train_mse = []
test_mse = []

#defining splits
kfold = KFold(n_splits=5, shuffle=True)

for i, model in enumerate(algorithms):
    scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='neg_mean_s
    print(f"{algorithm_names[i]} - Score: {scores}; Mean: {scores.mean()}")
```

```
Linear Regression - Score: [-2.33017504e+11 -2.50085713e+11 -2.52836380e+11 -2.15329
406e+11
 -2.62574948e+11]; Mean: -242768790160.712
SVR - Score: [-6.91691775e+11 -5.26685153e+11 -8.01143671e+11 -7.69024203e+11
 -5.62516834e+11]; Mean: -670212327151.9222
KNeighbors Regressor - Score: [-4.41374789e+10 -7.12703968e+10 -6.04619583e+10 -6.69
145207e+10
 -8.32820446e+10]; Mean: -65213279874.65128
Decision-Tree Regressor - Score: [-4.53980095e+10 -9.44538533e+10 -6.27062677e+10 -
1.10243359e+11
 -6.88182752e+10]; Mean: -76323952857.03886
Random-Forest Regressor - Score: [-2.94011169e+10 -2.48059907e+10 -4.41301871e+10 -
4.18007174e+10
 -5.46396389e+10]; Mean: -38955530214.16865
```

In [233…

```
# random forest best because less
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'bootstrap': [True], 'max_depth': [5, 10, None],
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15]}

rf = RandomForestRegressor(random_state = 1)

grid = GridSearchCV(estimator = rf,
                    param_grid = param_grid,
                    cv = kfold,
                    n_jobs = -1,
                    return_train_score=True,
                    refit=True,
                    scoring='neg_mean_squared_error')

# Fit your grid_search
grid.fit(X_train, y_train)
```

Out[233…

```
GridSearCV

best_estimator_:
RandomForestRegressor

RandomForestRegressor
```

In [234…

```
from sklearn.metrics import mean_squared_error, r2_score

best_rf = grid.best_estimator_

# Predict
yPred = best_rf.predict(X_test)

# Evaluation
print("Best Parameters: ", grid.best_params_)
print("MSE: ", mean_squared_error(y_test, yPred))
print("R Square: ", r2_score(y_test, yPred))
```

```
Best Parameters:  {'bootstrap': True, 'max_depth': None, 'n_estimators': 15}
MSE:  53778051425.11272
R Square:  0.9195267017732935
```

In [235…

```
# checking best grid paramters
grid.best_params_
```

Out[235…

```
{'bootstrap': True, 'max_depth': None, 'n_estimators': 15}
```

In [236…

```
# best Mean Square Error (MSE) grid score
best_mse = grid.best_score_
best_mse
```

Out[236…

```
np.float64(-37140144416.330505)
```

In [237…

```
# Testing feature importance
rf = grid.best_estimator_
```

```
rf.feature_importances_
```

Out[237…

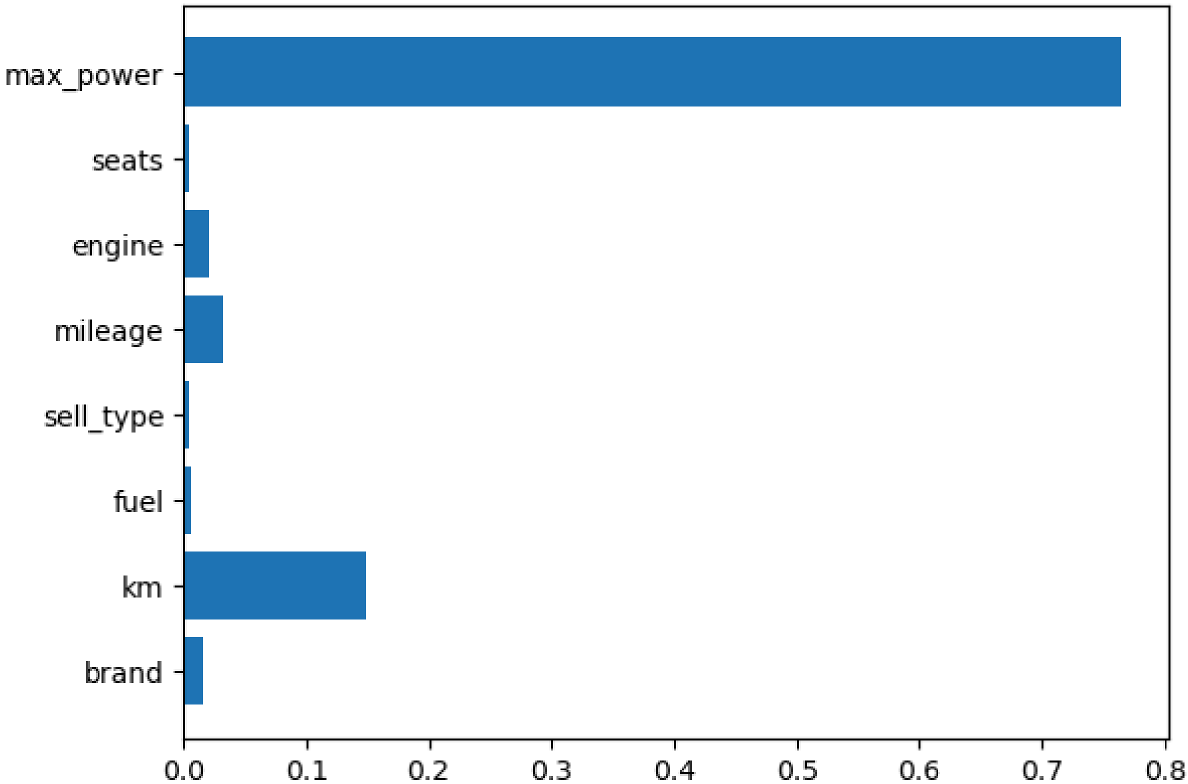
```
array([0.01563404, 0.14954239, 0.00666197, 0.00500465, 0.03287881,
       0.02039684, 0.00458287, 0.76529844])
```

In [238…

```
# using plot checking feature importance .i.e. which feature more suitable
plt.barh(X.columns, rf.feature_importances_)
```

Out[238…

```
<BarContainer object of 8 artists>
```

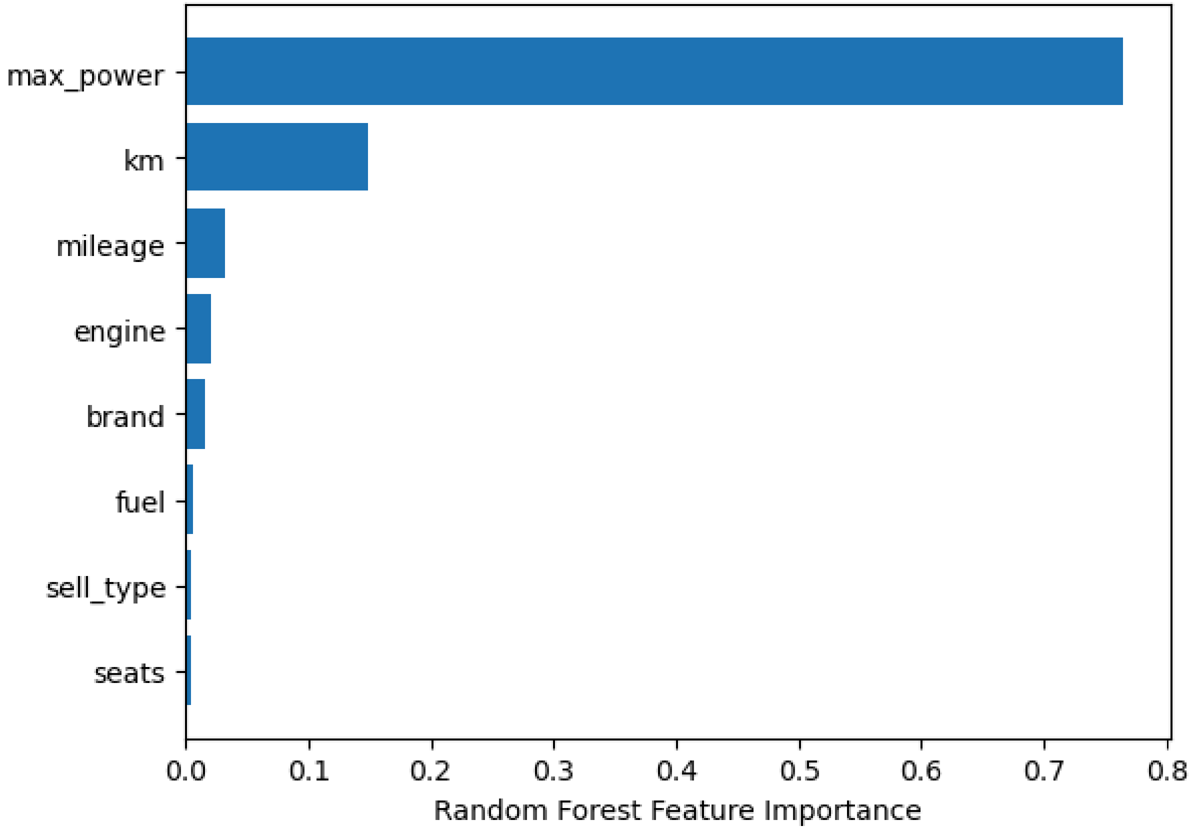


In [239…

```
# sorting feature importance
sorted_idx = rf.feature_importances_.argsort()
plt.barh(X.columns[sorted_idx], rf.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[239…

```
Text(0.5, 0, 'Random Forest Feature Importance')
```

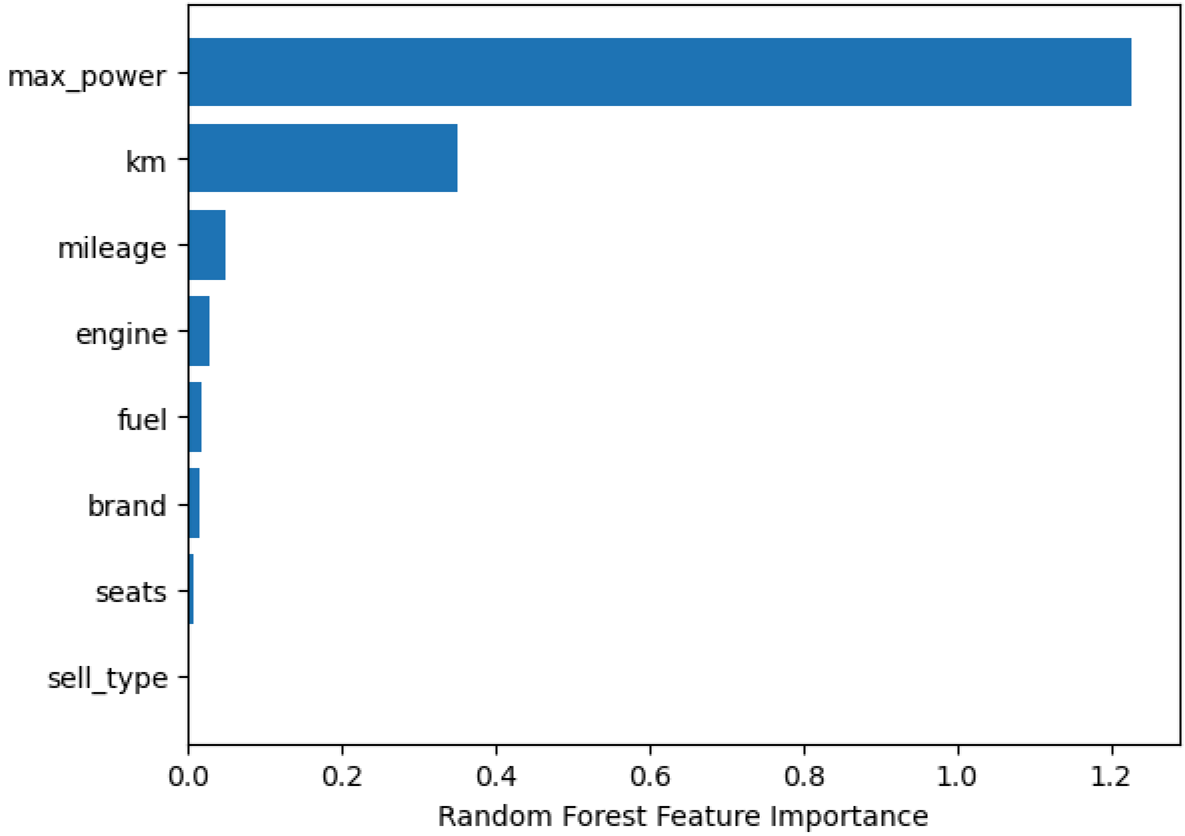


In [240…

```python
# checking permutation importance from feature importance
from sklearn.inspection import permutation_importance

perm_importance = permutation_importance(rf, X_test, y_test)

#let's plot
sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(X.columns[sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[240…

```
Text(0.5, 0, 'Random Forest Feature Importance')
```



In [241…

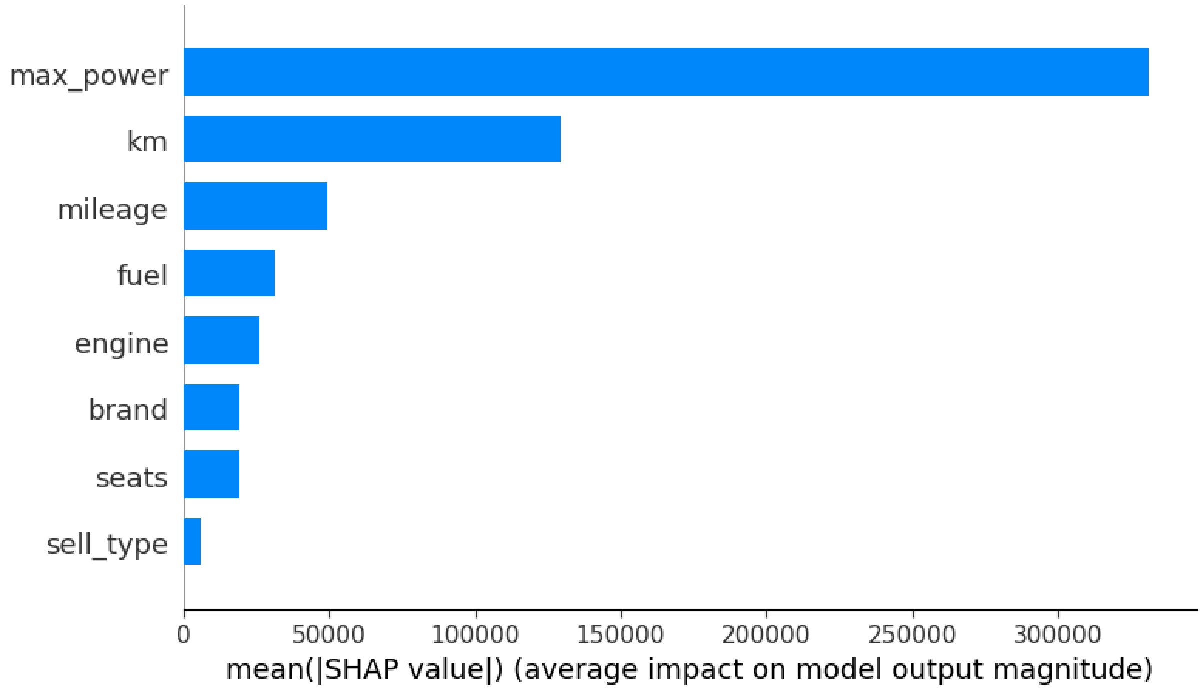
```
# importing shap to check shap values
import shap

explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap_values
```

Out[241…

```
array([[ -27357.04393395,   46420.44323557,  -33609.9681708 , ...,
         -41123.50174294,  -11544.65827251, -161514.06837105],
       [  14370.49201115, -184185.62286117,   34105.51599632, ...,
          65608.76469156,  -15135.84834552,  173200.17589474],
       [    808.67895293,  -65198.48649481,  -32966.35532951, ...,
         -11626.42454984,  -19176.07864822, -264403.84685037],
       ...,
       [ -18364.56826239,   84938.42368942,  -16405.3652584 , ...,
         -20151.61289766,  -25933.91927101, -413764.88105811],
       [  16961.03753411,  130454.28669583,   55214.38808812, ...,
         -14602.91986752,  -10654.56273411,  -62541.31587635],
       [  -8558.8137263 ,  142077.31236595,  -18560.16171573, ...,
         -21801.30642687,   -9449.49387753, -509548.69791831]])
```

In [242…

```
shap.summary_plot(shap_values, X_test, plot_type="bar", feature_names = X.columns)
```



In [243…

```
# Inference
# Save Model
import pickle

filename = 'model/carPricePrediction.model'
pickle.dump(grid, open(filename, 'wb'))
```

In [244…

```
loaded_model = pickle.load(open(filename, 'rb'))
```

In [245…

```
df[['brand', 'km', 'fuel', 'sell_type', 'mileage', 'engine', 'seats','max_power']].
```

Out[245…

```
brand             27.00
km            120000.00
fuel               0.00
sell_type          1.00
mileage           21.14
engine          1498.00
seats              5.00
max_power        103.52
Name: 1, dtype: float64
```

In [246…

```
# providing data set sample to predict result
# sample = np.array([[27,120000,0,1,21.14,1498,5,103]])
sample = np.array([[1,145500,0,1,23.4,1248,5,74]])
```

In [247…

```
# finally car price is predicted
predicted_Car_Price = loaded_model.predict(sample)
predicted_Car_Price
```

Out[247…

```
array([1868666.66666667])
```

In [ ]:

Report on Car Price Prediction

In my Car Price Prediction Model I took 8 features naming ['brand', 'km', 'fuel', 'sell_type', 'mileage', 'engine', 'seats','max_power'] whereas dropped Year, Transmission, Owner, and Torque. The reason I dropped these features were because it had very poor correlation matrix and doesn't relate with Car Price so dropping those would increase Model accuracy so as per that I dropped those columns. Since, max power, km driven and mileage are main factor that affects price of car those are features I prefered and also as per feature importance it showed important thats the reason I prefered using those features.

Firstly, I used Linear Regression and tested the model with accuracy and got really bad accuracy R square being around 0.64 then I used Cross validation and Grid Search to get which Model best works as per my data. Then Random Forest had the highest accuracy which is around 0.91 so I prefered using Random Forest Regressor Model to predict my data and also I checked data to make sure its working mechanism. The result I got was pretty good so I used Random Forest Regressor Algorithm. Linear Regressor is mainly used in Linear data like feature and price but car pricing is non-linear data that is why Random Forest Regressor have better result.

Finally, explaining whole step in conclusion in brief, I imported data from 'Cars.csv' file cleaned ta data set filled the null value or delete rows if less null data set, dropped unwanted rows for example as per question LPG gas, CPG gas had different mileage system so its required to delete those data with LPG and CPG so I did to clean my data set and so on test drive cars were expensive thats why I deleted those row as well. Then remove every unit changed to float and LabelEncoded datas because Prediction work only on numeric values. Then split using train_test_split, gave 30% test data and remaining 70% for training. Then used Linear Regression and test model using (MSE) Mean Square Error and R Square where accuracy was not what we expected was poor accuracy around 0.64 so used Cross Validation and Grid Search to test which model works better. As per the result RandomForestRegressor worked better with around 0.91 accuracy R square which is close to 1 so I preffered using RandomForest and at last exported/saved model using pickle. As I checked feature importance where max_power was on top which affect most prices of car and sell_type had lowest feature importance among features I selected. Then I worked on deploying model using plotly(python data visualization library). Here I created form to take user data and as per that data car price is predicted.

github link for Car Price Prediction: https://github.com/shakyarahul435/AIT-ML-CarPricePrediction-st125982