Used Cars Price Prediction

Problem Definition

[ ] ↳ *5 cells hidden*

Data Overview

- Observations
- Sanity checks

```
# Load the data
data=pd.read_csv('/content/drive/MyDrive/ADSP MIT/Colab Notebooks/Capstone Project/used_cars.csv');
# Save data to a variable
df = data.copy()
```

```
#Check whether the data was loaded in properly
df.head(10)
```

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_pr
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	N
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	N
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.20	1199.0	88.70	5.0	8
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77	1248.0	88.76	7.0	N
4	4	Audi A4 New 2.0 TDI	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968.0	140.80	5.0	N

```
df.tail(10)
```

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_
7243	7243	Renault Duster 85PS Diesel RxL	Chennai	2015	70000	Diesel	Manual	First	19.87	1461.0	83.8	5.0	
7244	7244	Chevrolet Aveo 1.4 LS	Pune	2009	45463	Petrol	Manual	First	14.49	1399.0	92.7	5.0	
7245	7245	Honda Amaze S i-Vtech	Kochi	2015	44776	Petrol	Manual	First	18.00	1198.0	86.7	5.0	
7246	7246	Hyundai Grand i10 AT Asta	Coimbatore	2016	18242	Petrol	Automatic	First	18.90	1197.0	82.0	5.0	
		Hyundai											

Observations

- The data seems to have loaded in properly with each feature in the data dictionary present.
- It seems the Serial Number corresponds to the record index number - we will verify this later and proceed as needed
- From the data preview, it seems some of the car prices (both new and resold) are missing => To be explored in the EDA

```
# Check for duplicate records
df.duplicated().value_counts()
```

	count
False	7253

dtype: int64

Observations

- There are no duplicate records, so we can proceed

```
# Check data overview information
print(df.shape)
print('-'*50)
print(df.info())
print('-'*50)
df.isnull().sum()
```

```
(7253, 14)
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   S.No.              7253 non-null   int64
 1   Name               7253 non-null   object
 2   Location           7253 non-null   object
 3   Year               7253 non-null   int64
 4   Kilometers_Driven  7253 non-null   int64
 5   Fuel_Type          7253 non-null   object
 6   Transmission       7253 non-null   object
 7   Owner_Type         7253 non-null   object
 8   Mileage            7251 non-null   float64
 9   Engine             7207 non-null   float64
 10  Power              7078 non-null   float64
 11  Seats              7200 non-null   float64
 12  New_price          1006 non-null   float64
 13  Price              6019 non-null   float64
dtypes: float64(6), int64(3), object(5)
memory usage: 793.4+ KB
None
--------------------------------------------------
```

	0
S.No.	0
Name	0
Location	0
Year	0
Kilometers_Driven	0
Fuel_Type	0
Transmission	0
Owner_Type	0
Mileage	2
Engine	46
Power	175
Seats	53
New_price	6247
Price	1234

dtype: int64

Observations:

- 5 of the columns are object-type, the rest of numerical (3 integers, 6 float)
- We have 7253 total records
    - Mileage, Engine type, Power, No. seats, New_Price, and resale price are all missing some entries, these will have to be treated later

```
# Check with certainty whether the serial number column correponds to the index and drop the column if that's the case
if np.all(df.index==df['S.No.'].values):
  print('Dropped serial number column')
  df.drop('S.No.',axis=1,inplace=True)
```

```
else:
  print('Serial number column does not correpond with index number')
```

```
Dropped serial number column
```

Observations

- The Serial Number column didn't add any value to our data, so we removed it. In the exploratory data analysis, we will look at each column individually and get a general idea of what the data looks like overall

Exploratory Data Analysis

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions:

1. What is the summary statistics of the data? Explore summary statistics for numerical variables and the categorical variables
2. Find out number of unique observations in each category of categorical columns? Write your findings/observations/insights
3. Check the extreme values in different columns of the given data and write down the observtions? Remove the data where the values are un-realistic

Overall Summary statistics

```
# Look at summary statistics before looking at individual columns
df.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Name	7253	2041	Mahindra XUV500 W8 2WD	55	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Location	7253	11	Mumbai	949	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Year	7253.0	NaN	NaN	NaN	2013.365366	3.254421	1996.0	2011.0	2014.0	2016.0	2019.0
Kilometers_Driven	7253.0	NaN	NaN	NaN	58699.063146	84427.720583	171.0	34000.0	53416.0	73000.0	6500000.0
Fuel_Type	7253	5	Diesel	3852	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Transmission	7253	2	Manual	5204	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Owner_Type	7253	4	First	5952	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Mileage	7251.0	NaN	NaN	NaN	18.14158	4.562197	0.0	15.17	18.16	21.1	33.54
Engine	7207.0	NaN	NaN	NaN	1616.57347	595.285137	72.0	1198.0	1493.0	1968.0	5998.0
Power	7078.0	NaN	NaN	NaN	112.765214	53.493553	34.2	75.0	94.0	138.1	616.0
Seats	7200.0	NaN	NaN	NaN	5.280417	0.809277	2.0	5.0	5.0	5.0	10.0
New_price	1006.0	NaN	NaN	NaN	22.779692	27.759344	3.91	7.885	11.57	26.0425	375.0
Price	6019.0	NaN	NaN	NaN	9.479468	11.187917	0.44	3.5	5.64	9.95	160.0

Observations

We included categorical and numerical variables. We can already make some interesting observations:

Categorical Variables

- **Name:** There are 2041 unique car models across all brands in our data. It may be useful to add an extra column for car brands to see if there's any trend in price across brands
- **Location:** There are 11 unique cities/locations where cars are sold. We will need to check for possible typos. Mumbai has the most records, which on the surface makes sense as it is the second most populated city in India **reference**
- **Fuel Type:** We see 5 fuel types, as described in the data dictionary. We will make sure these correspond to what we expect. Diesel cars are the most frequent
- **Transmission:** Transmission data similarly seems to have been correctly collected as we only see two kinds of entries (presumably manual and automatic). We will check price trends between these two types

- **Owner Type:** We see 4 unique kinds of ownership. Later, we may be able to see how cars with different numbers of previous owners tend to be sold, and hopefully extract helpful insight into the used car market

Numerical Variables

- **Year:** Cars manufacture year range from 1996 to 2019, with an average of 2013. The quartiles also show that cars sold seem to be skewed towards those more recently manufactured.
- **km driven:** We see a wide range of kilometrage for the cars, from 171km to 73000km. We observe a crude outlier in 6.5 million km. This record will have to be looked at in more detail, and the column's statistics looked at again afterwards
- **Mileage:** It is impossible for cars to have a mileage of 0kmpl. These values will have to be looked at in detail. We see an average of of 18kmpl, however, this may be skewed because of the existence of 0.0kmpl mileage entries. It will be interesting to see trends between mileage and car brands or price.
- **Engine:** Engine cubic centimetres have a wide range of 72cc-5998cc. These seem to be outliers, as the quartiles are all within the 1000cc-2000cc range.
- **Power:** Similarly to engine displacement volume, we see a large upper outlier at 616bhp. It will be interesting to see whether this corresponds to the same car as the 5998cc car.
- **Seats:** It seems most cars have 5 seats. Note that this variable is discrete numerical
- **Prices:** From the summary statistics, we can see that the reselling price is on average much lower than the brand new car prices (as expected).

Individual categorical statistics overview

Name

```
#First of all, check the name column
print(df['Name'])
print('-'*50)
print(df['Name'].value_counts().value_counts())
```

```
0                                Maruti Wagon R LXI CNG
1                        Hyundai Creta 1.6 CRDi SX Option
2                                             Honda Jazz V
3                                     Maruti Ertiga VDI
4                          Audi A4 New 2.0 TDI Multitronic
                              ...
7248                     Volkswagen Vento Diesel Trendline
7249                               Volkswagen Polo GT TSI
7250                               Nissan Micra Diesel XV
7251                               Volkswagen Polo GT TSI
7252    Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...
Name: Name, Length: 7253, dtype: object
--------------------------------------------------
count
1     862
2     408
3     214
4     127
5      93
6      58
7      53
8      42
10     28
12     18
13     18
9      17
15     17
11     13
14     12
20      8
18      7
19      6
17      6
22      5
23      5
32      4
16      3
25      3
21      2
35      2
42      1
39      1
37      1
28      1
31      1
30      1
29      1
49      1
26      1
```

```
55      1
Name: count, dtype: int64
```

Observations

- We observe the same information as we did in the general summary statistics above. We can also see that a vast majority of car models only appear once in our dataset, with about half as many appearing twice. We will delve deeper into the frequency of certain models in our univariate analysis

Brand

- It may be useful for our analysis to create a new column based on the 'Name' column denoting only the brands of each model.

Creating column in data

```
# Create a brands list by extracting brand names from the 'Name' column and
brands=[]
for i in df['Name'].str.split():
  brands.append(i[0])

# Add the brands column to the dataframe
df.insert(loc=1,column='Brand',value=brands)
```

```
# Run sanity checks on the list
brands=pd.Series(brands) # Convert to pandas series to look at the data more easily
brands.unique() # Look at the unique brand names in our series
```

```
array(['Maruti', 'Hyundai', 'Honda', 'Audi', 'Nissan', 'Toyota',
       'Volkswagen', 'Tata', 'Land', 'Mitsubishi', 'Renault',
       'Mercedes-Benz', 'BMW', 'Mahindra', 'Ford', 'Porsche', 'Datsun',
       'Jaguar', 'Volvo', 'Chevrolet', 'Skoda', 'Mini', 'Fiat', 'Jeep',
       'Smart', 'Ambassador', 'Isuzu', 'ISUZU', 'Force', 'Bentley',
       'Lamborghini', 'Hindustan', 'OpelCorsa'], dtype=object)
```

```
# Check the brand column has been successfully introduced
df.head()
```

	Name	Brand	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_pr
0	Maruti Wagon R LXI CNG	Maruti	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	
1	Hyundai Creta 1.6 CRDi SX Option	Hyundai	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	

Observations

The brand column has successfully been introduced

There are a few important observations here:

- 'Land' likely comes from 'Land Rover'. We will have to correct this in the data
- There seem to be two Isuzu kinds due to the capitalisation of one. We will change this so there is only one Isuzu brand in our data
- 'OpelCorsa' will be renamed as 'Opel' since Corsa is the car model and Opel is the brand

Before making the changes, let's have a specific look at the data so we know what we're doing

```
# Land Rover
df.loc[df.Brand=='Land']
```

	Name	Brand	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_
13	Land Rover Range Rover 2.2L Pure	Land	Delhi	2014	72000	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	
14	Land Rover Freelander 2 TD4 SE	Land	Pune	2012	85000	Diesel	Automatic	Second	0.00	2179.0	115.00	5.0	
191	Land Rover Range Rover 2.2L Dynamic	Land	Coimbatore	2018	36091	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	
311	Land Rover Range	Land	Delhi	2017	44000	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	

Observations

- 'Land' does indeed correspond to 'Land Rover'

```
# Isuzu
df.loc[df.Brand=='Isuzu']
```

	Name	Brand	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_pric
2900	Isuzu MUX 4WD	Isuzu	Jaipur	2017	34429	Diesel	Automatic	First	13.80	2999.0	174.57	7.0	33.6

```
# ISUZU
df.loc[df.Brand=='ISUZU']
```

	Name	Brand	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_pric
3624	ISUZU D-MAX V-	ISUZU	Coimbatore	2018	20422	Diesel	Manual	First	12.4	2499.0	134.0	5.0	Na

Observations

- 'ISUZU' seems to be Isuzu rewritten as anticipated

```
# OpelCorsa
df.loc[df.Brand=='OpelCorsa']
```

	Name	Brand	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Ne
	OpelCorsa												

Observations

- As we said earlier, the brand should be Opel, instead of OpelCorsa

We can now apply the necessary changes to our data

```
# Changing all instances of 'Land' to 'Land Rover'
df.Brand.replace('Land','Land Rover',inplace=True)
# Changing ISUZU to Isuzu
df.Brand.replace('ISUZU','Isuzu',inplace=True)
# Changing OpelCorsa to Opel
df.Brand.replace('OpelCorsa','Opel',inplace=True)
# Creating variable for column
brands=df.Brand
```

EDA of the brands column

```
# Let's look at the new column in more detail now
print(df.Brand.unique(),'\n','-'*50,'\n',df.Brand.describe(),'\n','-'*50,'\n',df.Brand.value_counts(),sep='')
```

```
['Maruti' 'Hyundai' 'Honda' 'Audi' 'Nissan' 'Toyota' 'Volkswagen' 'Tata'
 'Land Rover' 'Mitsubishi' 'Renault' 'Mercedes-Benz' 'BMW' 'Mahindra'
 'Ford' 'Porsche' 'Datsun' 'Jaguar' 'Volvo' 'Chevrolet' 'Skoda' 'Mini'
 'Fiat' 'Jeep' 'Smart' 'Ambassador' 'Isuzu' 'Force' 'Bentley'
 'Lamborghini' 'Hindustan' 'Opel']
--------------------------------------------------
count       7253
unique        32
top       Maruti
freq        1444
Name: Brand, dtype: object
--------------------------------------------------
Brand
Maruti           1444
Hyundai          1340
Honda             743
Toyota            507
Mercedes-Benz     380
Volkswagen        374
Ford              351
Mahindra          331
BMW               312
Audi              285
Tata              228
Skoda             202
Renault           170
Chevrolet         151
Nissan            117
Land Rover         67
Jaguar             48
Fiat               38
Mitsubishi         36
Mini               31
Volvo              28
Porsche            19
Jeep               19
Datsun             17
Isuzu               5
Force               3
Bentley             2
Smart               1
Ambassador          1
Lamborghini         1
Hindustan           1
Opel                1
Name: count, dtype: int64
```

Observations

- We see 32 unique car brands, with Maruti Suzuki as the most frequent, followed closely by Hyundai. All discrepencies in the brand column have been addressed

Location

```
print(df.Location.unique(),'\n','-'*50,'\n',df.Location.value_counts(),sep='')
```

```
['Mumbai' 'Pune' 'Chennai' 'Coimbatore' 'Hyderabad' 'Jaipur' 'Kochi'
 'Kolkata' 'Delhi' 'Bangalore' 'Ahmedabad']
--------------------------------------------------
Location
Mumbai        949
Hyderabad     876
Coimbatore    772
Kochi         772
Pune          765
Delhi         660
Kolkata       654
Chennai       591
Jaipur        499
Bangalore     440
Ahmedabad     275
Name: count, dtype: int64
```

Observations

- There are 11 unique locations
- Nothing seems out of the ordinary for the location data. It is interesting to see fewer cars sold in Delhi than Mumbai, perhaps because of more competition from other companies or a smaller market for used cars in the capital area
- Ahmedabad sees significantly fewers sales than the other locations

Fuel Type

```
print(df.Fuel_Type.unique(),'\n','-'*50,'\n',df.Fuel_Type.value_counts(),sep='')
```

```
['CNG' 'Diesel' 'Petrol' 'LPG' 'Electric']
--------------------------------------------------
Fuel_Type
Diesel      3852
Petrol      3325
CNG           62
LPG           12
Electric       2
Name: count, dtype: int64
```

Observations

- We find the 5 fuel types we expect, so no need to edit the data.
- Diesel and Petrol are by far the most frequent fuel types

Transmission

```
print(df['Transmission'].value_counts())
```

```
Transmission
Manual       5204
Automatic    2049
Name: count, dtype: int64
```

Observations

- We observe the vaues we expect
- There are considerably more manual (i.e. stick shift) cars than automatic

Owner Type

```
print(df['Owner_Type'].value_counts())
```

```
Owner_Type
First             5952
Second            1152
Third              137
Fourth & Above      12
Name: count, dtype: int64
```

Observations

- No issues with the data for number of previous owners
- The vast majority of cars have had one previous owners, with those cars appearing 5 times more than all other kinds of ownership tallies combined

Numerical features statistics overview

- We will look at features of interest, which are all except year

[ ] ↳ *24 cells hidden*

Summary

[ ] ↳ *3 cells hidden*

Univariate Analysis

Questions:

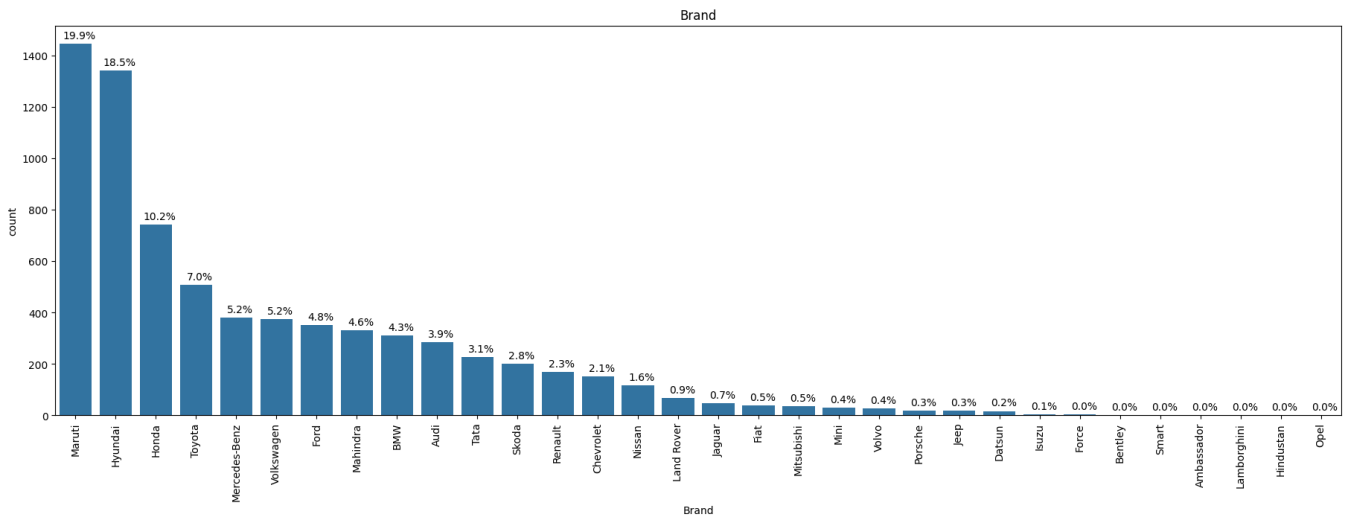
1. Do univariate analysis for numerical and categorical variables?
2. Check the distribution of the different variables? is the distributions skewed?
3. Do we need to do log_transformation, if so for what variables we need to do?
4. Perfoem the log_transformation(if needed) and write down your observations?

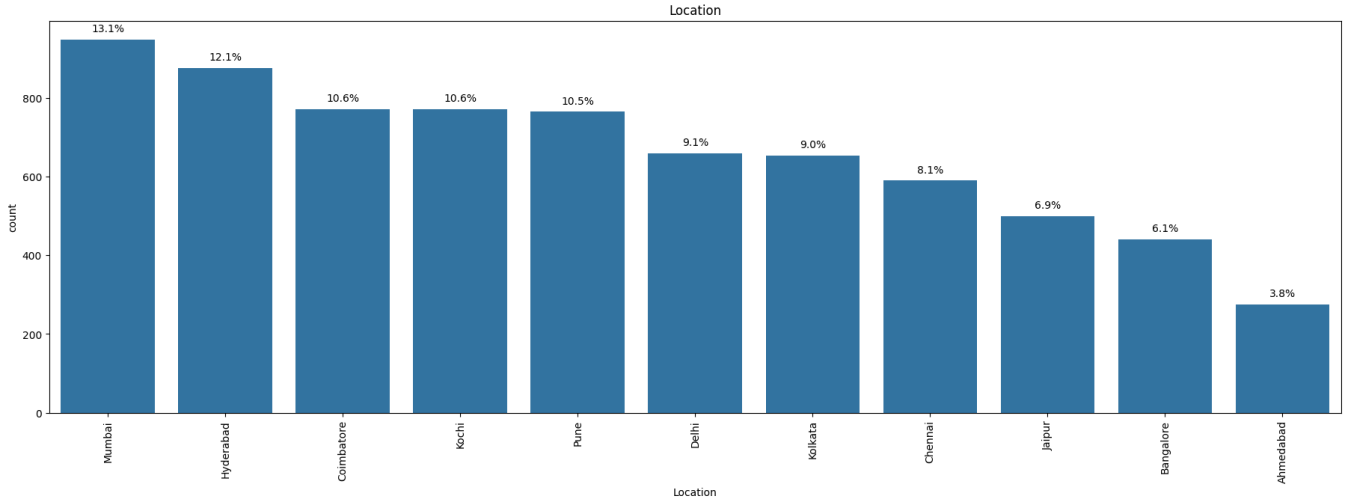
Categorical variables

Categorical variables are: Name, Brand, Location, Fuel Type, Tramsission, and Owner Type. We will include Seats here because is it discrete

```
# Create list that includes all categorical variables and the Seats variable, excluding Name, which we will look at separately
cat=df.select_dtypes(include='object').columns.to_list()
cat.append('Seats')
cat.remove('Name')


# Create for loop for each categorical variable (excluding Name, and including Seats)
for i in cat:
# Set graph dimensions & titles
  plt.figure(figsize=(18,6))
  plt.title(i)
  plt.tight_layout()
# Make labels vertical to read easier
  plt.xticks(rotation=90)
# Plot the data
  plot=sns.countplot(data=df,x=i,order=df[i].value_counts().index)
# Create variables to be used later when displaying data as pie charts
  x=np.array(df[i].value_counts()) # NumPy array of count of unique values of feature i
  y=df[i].value_counts().index.to_list() # Corresponding unique values to x. We don't use df[i].unique() to make sure the x and y iterat
  percs=[] # List we will use later to collect percentages
# Add percentages to the bins
  total=0
# Get total number of entries for the feature i
  for j in plot.patches:
    total=total+j.get_height()
# Find percentages, add to the percentages list, and annotate bar chart
  for k in plot.patches:
    perc=round(k.get_height()*100/total,1)
    percs.append(perc)
    if i=='Brand':
      plot.annotate(str(perc)+'%',(k.get_x()+0.5*k.get_width()+0.1,k.get_y()+k.get_height()+20),ha='center')
    else:
      plot.annotate(str(perc)+'%',(k.get_x()+0.5*k.get_width(),k.get_y()+k.get_height()+20),ha='center')
  plt.show()
# Create pie chart
  if i=='Brand':
    print('-'*200,'\n'*5)
    continue
  else:
    plt.title(i)
    plt.pie(x)
    plt.tight_layout()
    labels=[str(cat)+' - '+str(p)+'%' for cat,p in zip(y,percs)] # Use y and percs to write the custom legends for each pie chart
    plt.legend(labels,bbox_to_anchor=(1,1))
    plt.show()
  print('-'*200,'\n'*5)

# Name graph
plt.figure(figsize=(10,8))
plt.title('Car names')
plt.xticks(visible=False) # Too many names, so remove the labels and instead show the top 5 cars
sns.countplot(data=df,x='Name',order=df.Name.value_counts().index)
plt.show()
print('Top 5 most sold cars','\n',pd.DataFrame(df.Name.value_counts()).rename(columns = {'Name':'Count'}).head(),'\n'*2)

# Plot bar chart showing the count of unique car models that have a certain occurence in our data
# Create dataframe
namecalcs=pd.DataFrame({'Count':df.Name.value_counts().value_counts().iloc[:15].index,'Value':df.Name.value_counts().value_counts().iloc
namecalcs.loc[len(namecalcs.index)] = ['16+', df.Name.value_counts().value_counts().iloc[15:].sum()]
# Plot bar chart
plt.figure(figsize=(10,8))
plot2=sns.barplot(data=namecalcs,x='Count',y='Value',color='Blue')
plt.title('Number of unique car models per occurence rate in our data')
plt.xlabel('Occurence')
plt.ylabel('Count of unique car models per occurence')
# Annotate
percs2=[]
```

```
for k in plot2.patches:
  plot2.annotate(k.get_height(),(k.get_x()+0.5*k.get_width(),k.get_y()+k.get_height()+10),ha='center')
plt.show()
print('The number of car models that appear with statistical significance (30 or more times) in our data is:',(df.Name.value_counts()>=3
```





Observations

Many of the categorical variables seem to be skewed heavily to one or a few values. This may indicate a relative uniform market for those features

- **Brand:** About half the car brands have less than a 1% share of the pre-owned car market. Since we have 32 car brands, an equal market share between all of them would be 3.125%. We see 21 brands have a lower share of the market than this. We also see that over 50% of the market is shared between only three car brands: Maruti Suzuki, Hyundai, and Honda
- **Location:** The distribution for location is more uniform than for brand, with only about one third of the locations having less than 1/11th of the market share. As seen earlier, Ahmedabad sees significantly fewer sales than other locations. It may be interesting to compare car brands and location
- **Fuel Type:** As seen earlier, this feature is dominated by Diesel and Petrol, with the remaining fuel types only adding up to just above 1%. It will be interesting to see how these affect prices. It may also be good to look at the manufacture year for cars with these fuel types.
- **Transmission:** There is about a 5:2 ratio between manual:automatic cars
- **Owner Type:** The vast majority of cars sold had only one previous owner. We can look at how this distribution differs between the other features
- **No. seats:** The number of seats is heavily skewed towards 5 at 84% market share, with the next highest being 7 seats at 11%
- **Car names & models:** We see an exponential decrease in car name counts. Out of the top 5 car models, 3 are Maruti Suzuki, and 1 Honda. Surprisingly, the most sold car is a Mahindra, which only has a 4.6% market share. We can also see that the vast majority of car models

appear only once in our data (at 862 distinct models), and 409 appear twice. This means that over half of our 2041 disinct car models appear only once or twice in our data. This makes predictions based on specific car models/names quite difficult, and we may end up dropping the 'Name' column in favour of 'Brand' from our data because of it unless we aggregate the data from the 'Name' column some other way. This is reinforced by seeing that only 13 unique car models appear 30 or more times in our data - not nearly enough to make any useful conclusions about trends and predictions in prices based off the 'Name' column

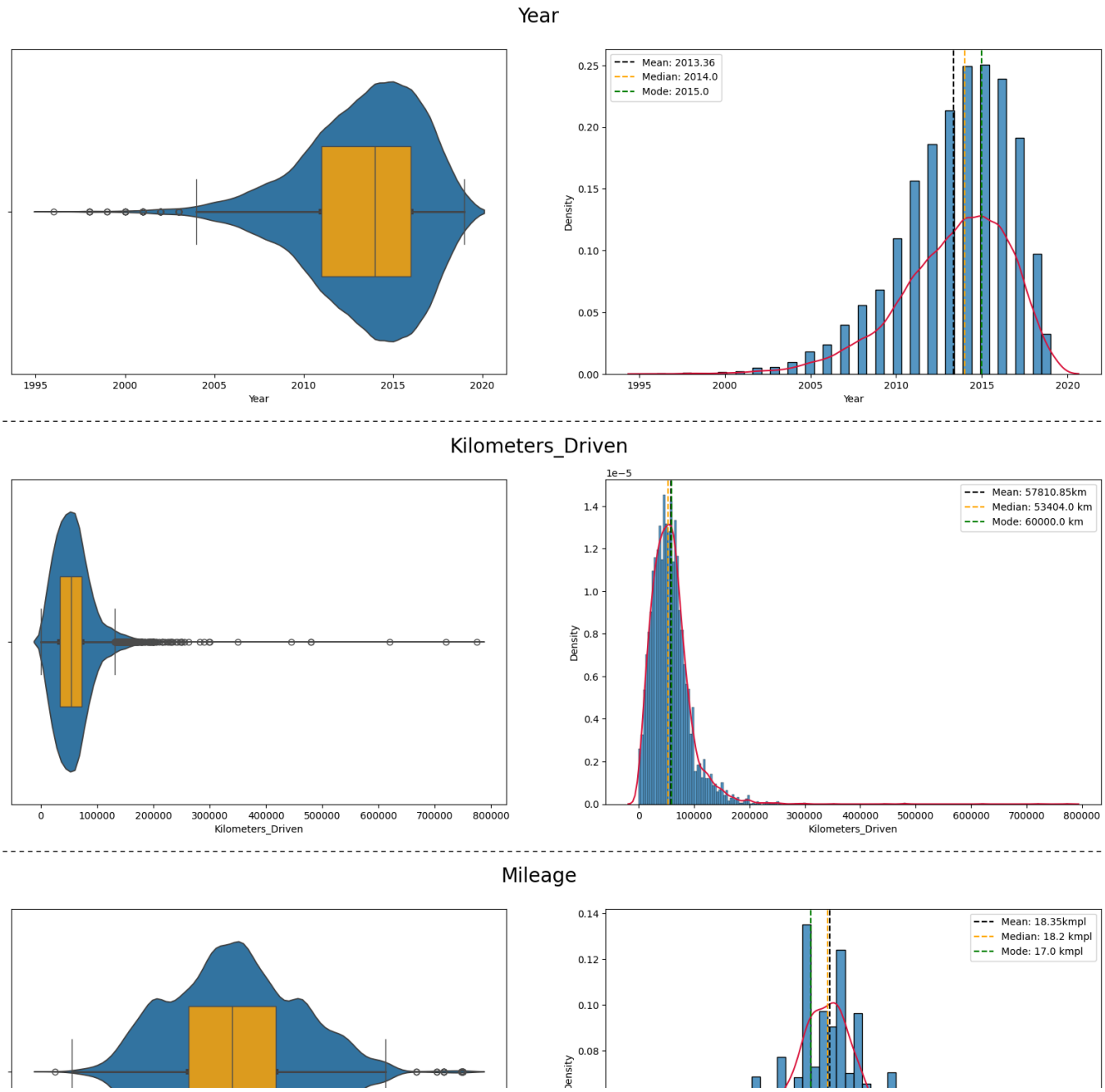
Numerical variables

```
# Create list with all numerical variables (including Seats)
num=df.select_dtypes(include=['int64','float64']).columns.to_list()
```

```
# Create list for each numerical variable's units and correspond the variables to the list as their indices
un=['','km','kmpl','cc','bhp','seats','INR 100,000','INR 100,000']
uns=pd.Series(un,index=num)
# Create function that returns the unit when given a numerical variable
def units(unit):
  for j in uns.index:
    if unit==j:
      return uns[j]
```



```
'''
We will make two kinds of plot for each numerical variable:
  1. Violin plot superimposed by a boxplot
  2. Histogram with the kernel density estimation and vertical lines for the mean, median, and mode
Note:
  - We construct a separate if statement for the prices since we aren't including modal price lines
  - We construct a separate if statement for the Year variable to make sure the legend doesn't obstruct the graph
  - To better visualise the price graph, we print one plot, instead of plotting the violin plot and histogram on the same row
  -
'''
for i in num:
  if i=='New_price' or i=='Price':
    fig,(b,h)=plt.subplots(nrows=2,ncols=1,figsize=(20,15)) # Increase figsize here to better see the price graphs
    fig.suptitle(i,fontsize=20)
    fig.tight_layout()
    sns.violinplot(data=df,x=i,ax=b)
    sns.boxplot(data=df,x=i,width=0.4,color='orange',boxprops={'zorder': 2},ax=b) # Superimpose a boxplot on top of the violinplot
    sns.histplot(data=df,x=i,kde=False,ax=h,stat='density')
    sns.kdeplot(data=df,x=i,color='crimson',ax=h)
    meanline=h.axvline(x=df[i].mean(),linestyle='dashed',color='black',label='Mean: '+str(round(df[i].mean(),2))+' '+units(i))
    medianline=h.axvline(x=df[i].median(),linestyle='dashed',color='orange',label='Median: '+str(round(df[i].median(),2))+' '+units(i))
    h.legend(loc='upper right',handles=[meanline,medianline],fontsize='xx-large')
    plt.show()
    print('-'*200)
  elif i=='Year':
    fig,(b,h)=plt.subplots(nrows=1,ncols=2,figsize=(20,6))
    fig.suptitle(i,fontsize=20)
    sns.violinplot(data=df,x=i,ax=b)
    sns.boxplot(data=df,x=i,width=0.4,color='orange',boxprops={'zorder': 2},ax=b)
    sns.histplot(data=df,x=i,kde=False,ax=h,stat='density')
    sns.kdeplot(data=df,x=i,color='crimson',ax=h)
    meanline=h.axvline(x=df[i].mean(),linestyle='dashed',color='black',label='Mean: '+str(round(df[i].mean(),2))+units(i))
    medianline=h.axvline(x=df[i].median(),linestyle='dashed',color='orange',label='Median: '+str(round(df[i].median(),2))+units(i))
    mode=float(df[i].mode())
    modeline=h.axvline(x=mode,linestyle='dashed',color='green',label='Mode: '+str(round(float(df[i].mode()),2))+units(i))
    h.legend(loc='upper left',handles=[meanline,medianline,modeline])
    plt.show()
    print('-'*200)
  else:
    fig,(b,h)=plt.subplots(nrows=1,ncols=2,figsize=(20,6))
    fig.suptitle(i,fontsize=20)
    sns.violinplot(data=df,x=i,ax=b)
    sns.boxplot(data=df,x=i,width=0.4,color='orange',boxprops={'zorder': 2},ax=b)
    sns.histplot(data=df,x=i,kde=False,ax=h,stat='density')
    sns.kdeplot(data=df,x=i,color='crimson',ax=h)
    meanline=h.axvline(x=df[i].mean(),linestyle='dashed',color='black',label='Mean: '+str(round(df[i].mean(),2))+units(i))
    medianline=h.axvline(x=df[i].median(),linestyle='dashed',color='orange',label='Median: '+str(round(df[i].median(),2))+' '+units(i))
    mode=float(df[i].mode())
    modeline=h.axvline(x=mode,linestyle='dashed',color='green',label='Mode: '+str(round(float(df[i].mode()),2))+' '+units(i))
    h.legend(loc='upper right',handles=[meanline,medianline,modeline])
    plt.show()
    print('-'*200)
```



Observations

- **Year:** Looks pretty normally distributed, and doesn't have many significant outliers
- **Kilometers Driven:** Looks incredibly skewed, there are scores of outliers, including some very significant extreme values. Despite this, the median and mean values are very close together
- **Mileage:** Seems to be nigh-normally skewed, with the mean and median both very similar
- **Engine:** The distribution is a bit wild, with a few outliers and a bit of skew
- **Power:** Like Kilometers Driven, very skewed, with significant outliers. However here, the mean and median aren't as close to each other
- **Seats:** This feature is discrete, and so the above distributions don't carry very useful information compared to the countplots we plotted earlier. In our model building, we may create a model where we convert the Seats feature into categorical data instead of numerical, if the Seats variable is of statistcal significance
- **Price and New Price:** These are both very positively skewed, with many outliers

Since price, new price, and kilometers driven are clearly very skewed. Let's have a deeper look below

```
for i in num:
  print(i,'skew = ',df[i].skew())
```

```
Year skew =  -0.8397050406997638
Kilometers_Driven skew =  3.832545848818344
Mileage skew =  0.20494087213469345
Engine skew =  1.413133430829896
Power skew =  1.9624262822569614
Seats skew =  1.9548877812669965
New_price skew =  4.128299677490268
Price skew =  3.337576076530038
```

Observations

- As expected. Kilometers_Driven, Price, and New_price are very skewed
- We can also see Power and Seats are highly skewed too
- We may have to look at the engine feature too, as it is pretty skewed too

```
# Take the log of the skewed variables
km_log=np.log(df.Kilometers_Driven)
power_log=np.log(df.Power)
seats_log=np.log(df.Seats)
newprice_log=np.log(df.New_price)
price_log=np.log(df.Price)
num_log=[km_log,power_log,seats_log,newprice_log,price_log]
```

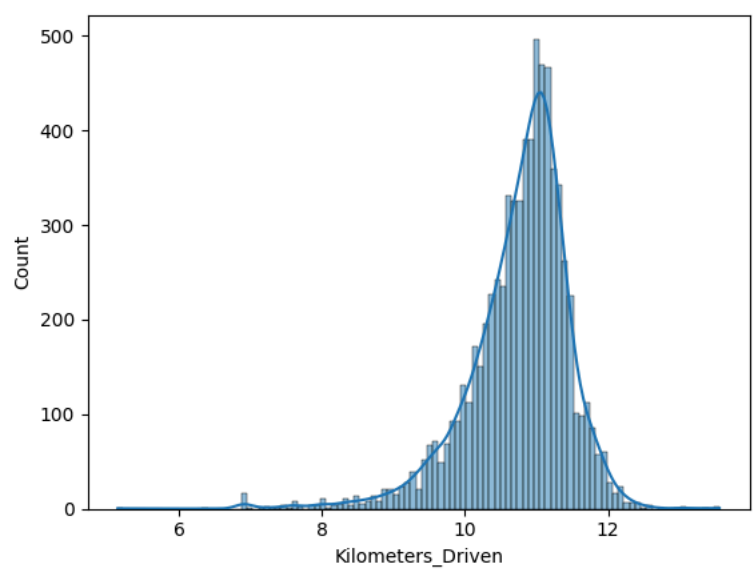
```
# Look at the skews again
for i,j in zip(num_log,['km_log','power_log','seats_log','newprice_log','price_log']):
  print(j,'skew:',i.skew())
```

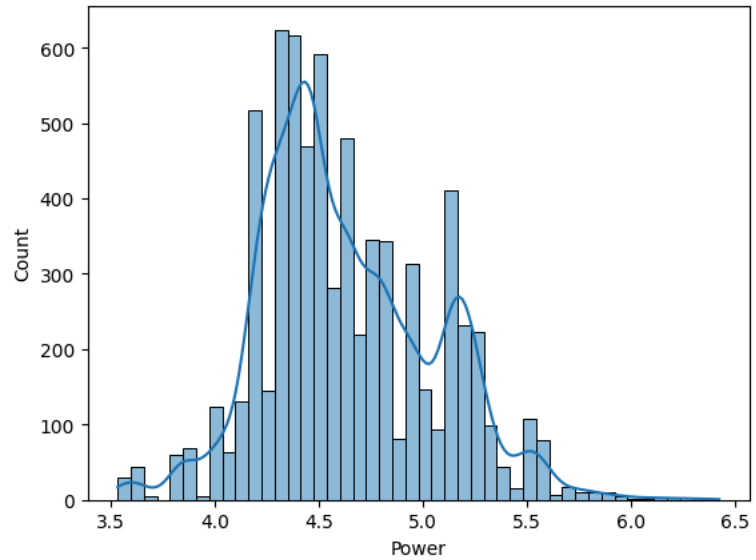
```
km_log skew: -1.3539415163105548
power_log skew: 0.4443657782789483
seats_log skew: 0.8630957198416143
newprice_log skew: 0.7238320014363577
price_log skew: 0.41621665771741323
```

Observations

- These all now look reasonable. -1.35 is still a bit extreme for km_log, but it is similar to the value we have for engine, so we will take the risk of taking this skewed data into linear regression

```
# Look at simple histograms for the log-transformed features
for i in num_log:
  sns.histplot(data=df,x=i,kde=True)
  plt.show()
  print('-'*100)
```





Observations

- The histograms are decidedly less skewed now that we have performed a log transform on the skewed features

```
# Add the log-transformed variables to the DataFrame
df['km_log']=km_log
df['power_log']=power_log
df['seats_log']=seats_log
df['newprice_log']=newprice_log
df['price_log']=price_log
```

```
# Move Price column to end of dataframe for ease in visualisation
movecol = df.pop("Price")
df.insert(len(df.columns), "Price", movecol)
# Check whether the new features have successfully been added
df.head()
```

	Name	Brand	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_pr
0	Maruti Wagon R LXI CNG	Maruti	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	N
1	Hyundai Creta 1.6 CRDi SX Option	Hyundai	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	N
2	Honda Jazz V	Honda	Chennai	2011	46000	Petrol	Manual	First	18.20	1199.0	88.70	5.0	8
3	Maruti Ertiga VDI	Maruti	Chennai	2012	87000	Diesel	Manual	First	20.77	1248.0	88.76	7.0	N
4	Audi A4 New 2.0 TDI Multitronic	Audi	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968.0	140.80	5.0	N

```
# Create new dataframe with only the transformed variables
df2=df.copy().drop(columns=['Kilometers_Driven','Power','Seats','New_price'])
```

```
# Check the new DataFrame was created correctly
df2.head()
```

	Name	Brand	Location	Year	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	km_log	power_log	seats_log	newpric
0	Maruti Wagon R LXI CNG	Maruti	Mumbai	2010	CNG	Manual	First	26.60	998.0	11.184421	4.063198	1.609438	
1	Hyundai Creta 1.6 CRDi SX Option	Hyundai	Pune	2015	Diesel	Manual	First	19.67	1582.0	10.621327	4.837868	1.609438	

Observations

- We have performed a log transformation on the variables that had high skew
- We created a new dataframe, df2, replacing the original variables with their transformed selves. We will be using df2 henceforth

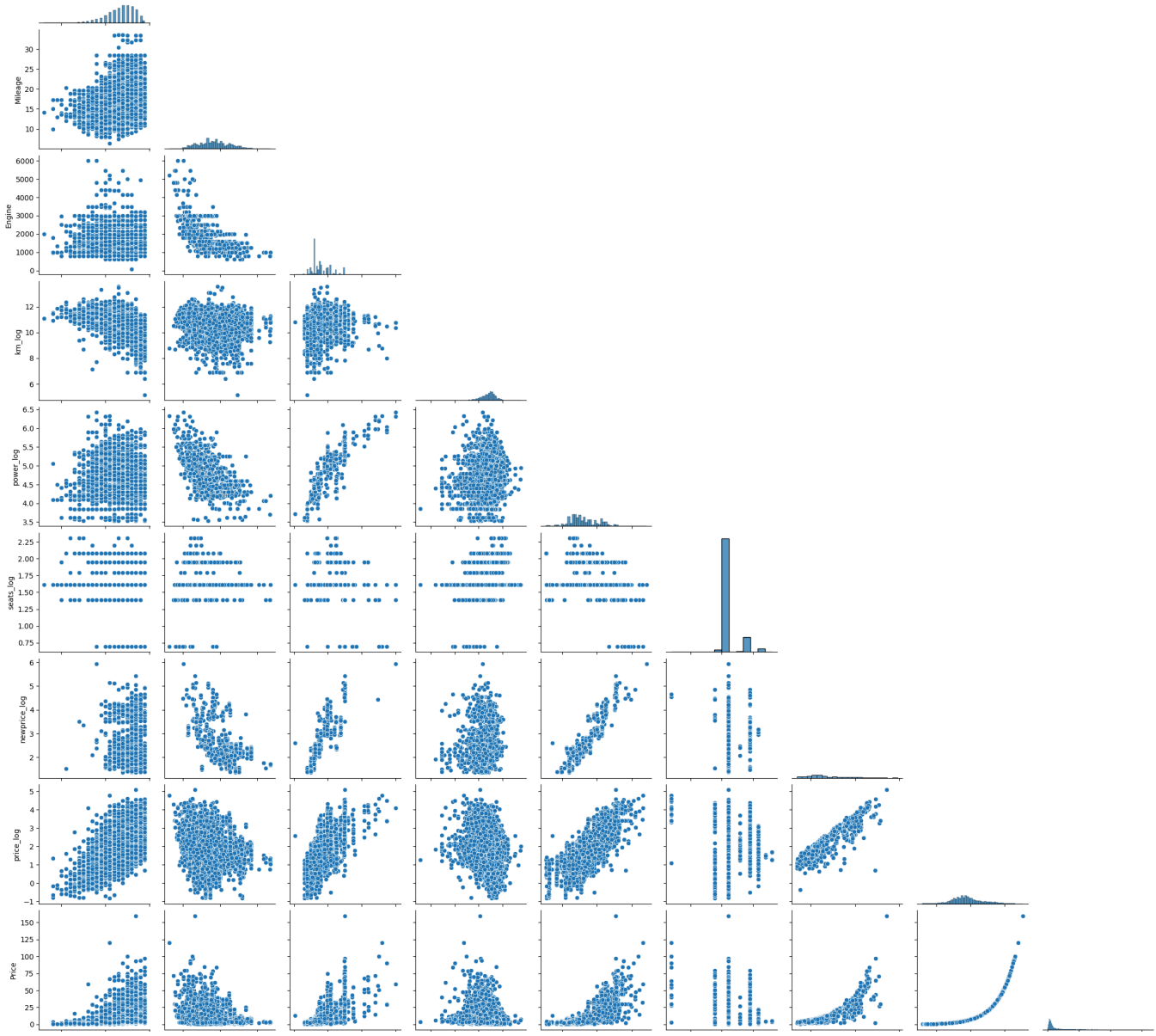
Bivariate Analysis

Questions:

1. Plot a scatter plot for the log transformed values(if log_transformation done in previous steps)?
2. What can we infer form the correlation heatmap? Is there correlation between the dependent and independent variables?
3. Plot a box plot for target variable and categorical variable 'Location' and write your observations?

```
# Plot pair-wise scatter plots for the features in df2
sns.pairplot(data=df2,dropna=True,corner=True)
```

```
<seaborn.axisgrid.PairGrid at 0x7d3c1ed18790>
```



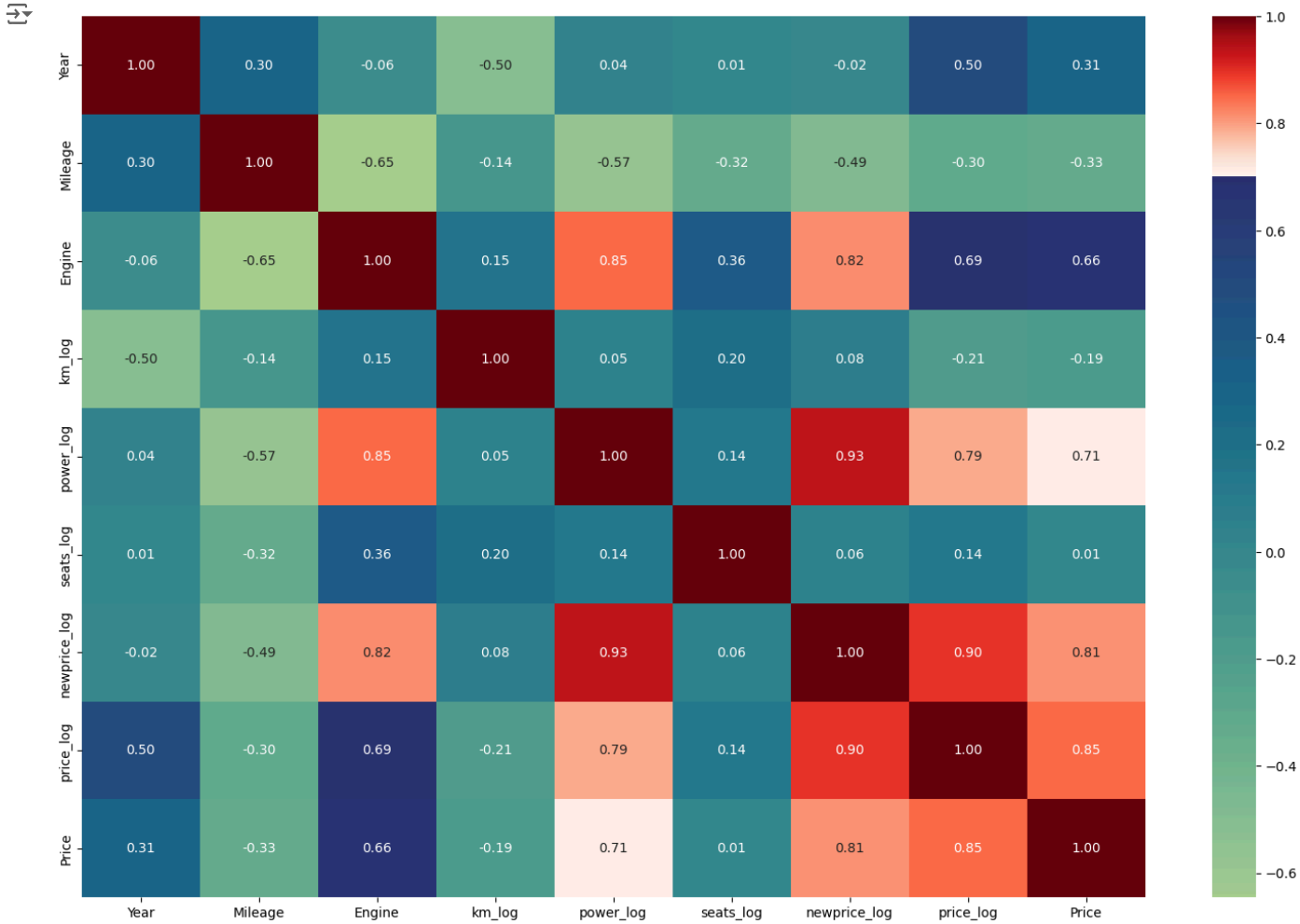
Observations for plots with price(log)

- Year seems very positive
- Mileage is negative, but not too strongly
- Engine seems very positive
- km_log has little to no correlation
- power_log big positive
- seats no correlation
- newprice big positive correlation (as expected)
- To note as well": The correlations with price are mostly non-linear, whereas the relationships with price_log seem much more linear, as we want

Let's see how this compares to the heatmap!

```
# Import colormap libraries
from matplotlib import colormaps as cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
# Concatenate a colormap with a more vibrant one to make correlations above 0.7 stand out more
cmap1= cm['crest'].resampled(1000)
cmap2= cm['Reds'].resampled(1000)

stacked_colors = np.vstack((cmap1(np.linspace(0, 1, 4500)),
                        cmap2(np.linspace(0, 1, 1000))))
custom_cmap = ListedColormap(stacked_colors)
```

```
plt.figure(figsize = (18, 12))

sns.heatmap(df2.corr(numeric_only = True), annot = True, fmt = '0.2f', cmap=custom_cmap)

plt.show()
```



Observations

The data here mirrors our pairplot:

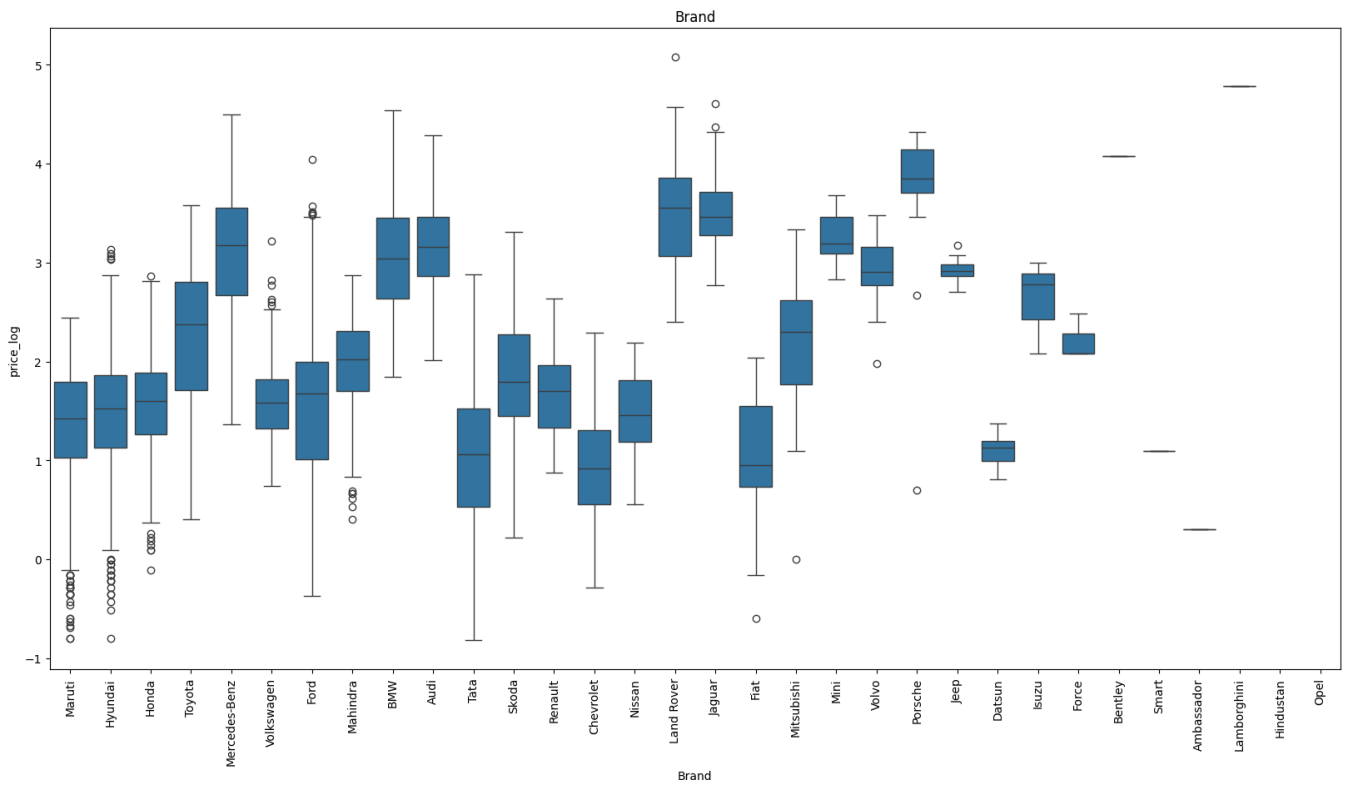
Dependent variable

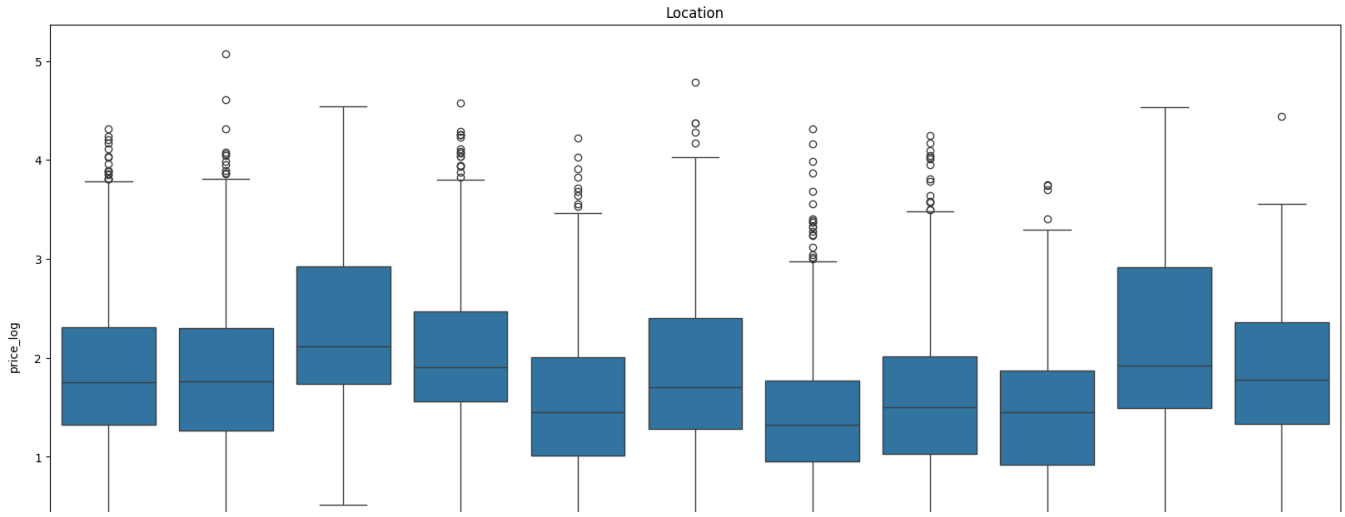
- Price(log) has high correlations with log(New_Price), power(log), and engine, with the highest being New_price(log) at 0.90! It is also moderaly correlated with Year, although not significantly so. All these correlations are positive.

Independent variables

- New_price(log) is highly correlated with power(log) and engine - more so than price(log) is. The correlation between New_price(log) and power(log) is the strongest between any two variables. These positive correlations make sense as you would expect a car with more engine displacement (hence a bigger engine) and a higher horsepower output would be more expensive
- Seat number doesn't have any significant correlations
- Something important to note is the high correlation between engine displacement and power(log), which, alongside the correlation with New_price(log), implies there is strong multicollinearity in our data. We will have to address this when building our models
- There are some interesting mild negative correlations between Km driven(log) & Year of manufacture, mileage & power(log), and mileage & engine

```
# Check boxplots between price(log) and categorical variables (except Name and adding Seats No.)
cat2=df2.select_dtypes(include='object').columns.to_list()
cat2.append('seats_log')
cat2.remove('Name')

for i in cat2:
  if i=='seats_log':
    plt.figure(figsize=(20,10))
    plt.xticks(rotation=90)
    plt.title(i)
    sns.boxplot(data=df2,x=i,y='price_log',order=df2[i].value_counts().sort_index().index)
    plt.show()
    print('-'*250)
  else:
    plt.figure(figsize=(20,10))
    plt.xticks(rotation=90)
    plt.title(i)
    sns.boxplot(data=df2,x=i,y='price_log',order=df2[i].value_counts().index)
    plt.show()
    print('-'*250)
```





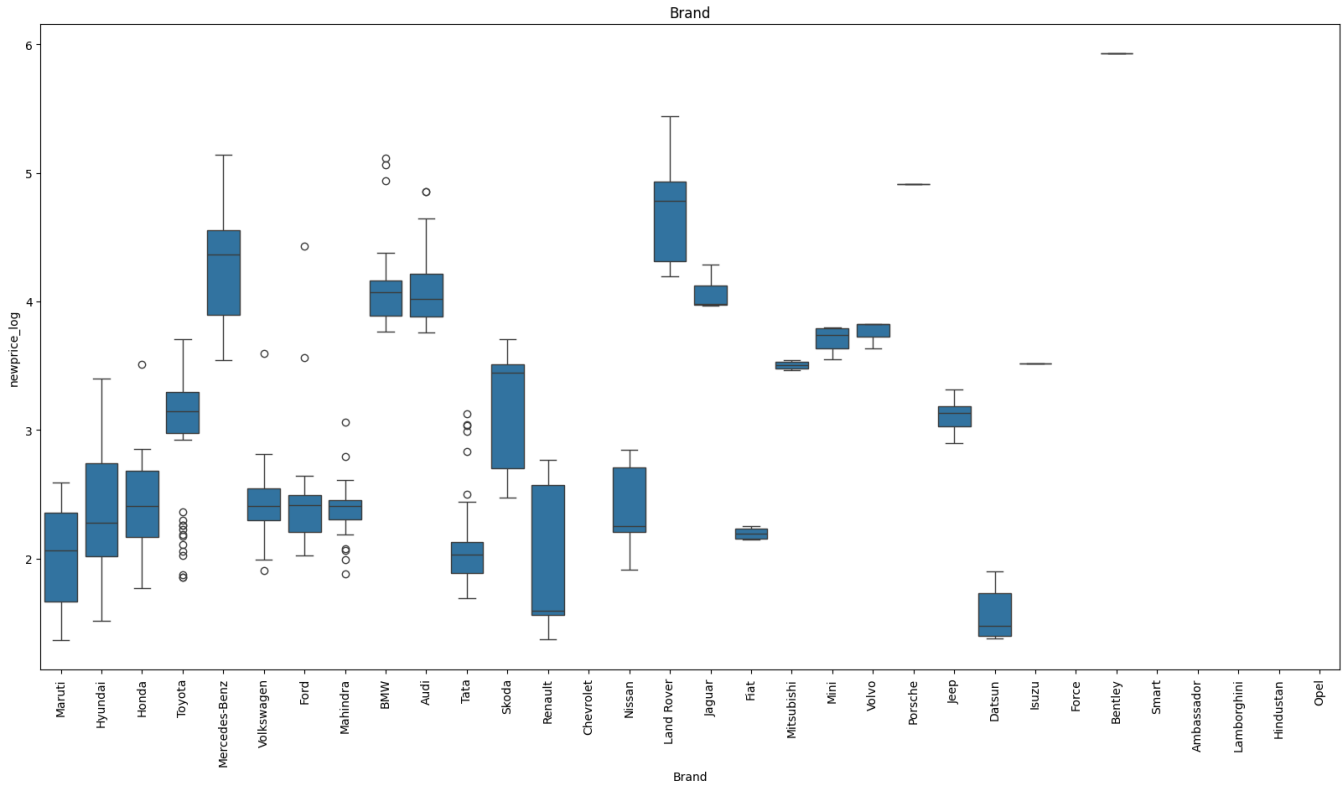
Observations

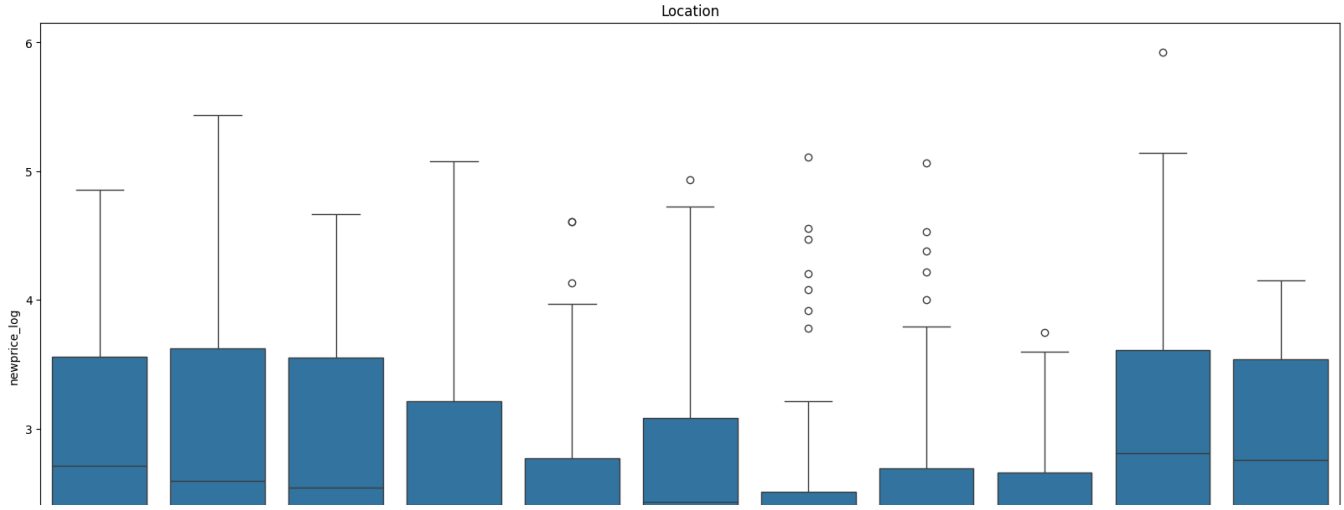
Many of the categorical variables seem to be skewed heavily to one or a few values. This may indicate a relative uniform market for those features

- **Brand:** About half the car brands have less than a 1% share of the pre-owned car market. Since we have 32 car brands, an equal market share between all of them would be 3.125%. We see 21 brands have a lower share of the market than this. We also see that over 50% of the market is shared between only three car brands: Maruti Suzuki, Hyundai, and Honda

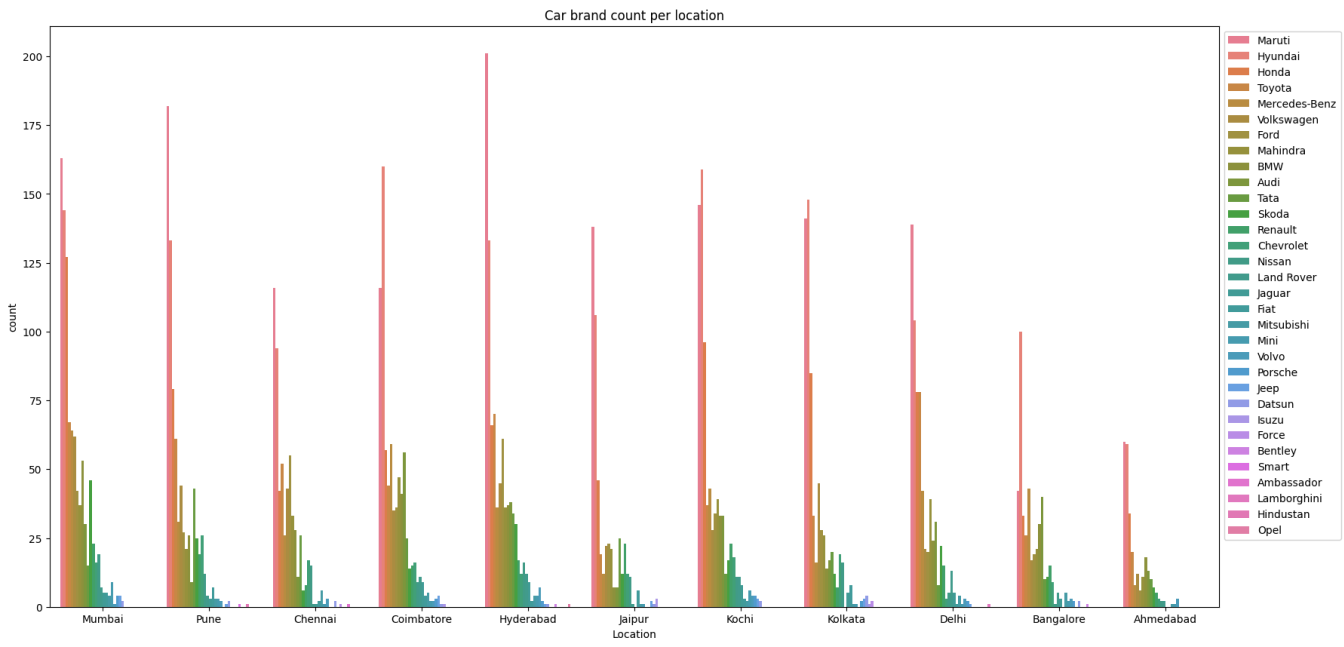
- **Location:** The distribution for location is more uniform than for brand, with only about one third of the locations having less than 1/11th of the market share. As seen earlier, Ahmedabad sees significantly fewer sales than other locations. It may be interesting to compare car brands and location
- **Fuel Type:** As seen earlier, this feature is dominated by Diesel and Petrol, with the remaining fuel types only adding up to just above 1%. It will be interesting to see how these affect prices. It may also be good to look at the manufacture year for cars with these fuel types.
- **Transmission:** There is about a 5:2 ratio between manual:automatic cars
- **Owner Type:** The vast majority of cars sold had only one previous owner. We can look at how this distribution differs between the other features
- **No. seats:** The number of seats is heavily skewed towards 5 at 84% market share, with the next highest being 7 seats at 11%
- **Car names & models:** We see an exponential decrease in car name counts. Out of the top 5 car models, 3 are Maruti Suzuki, and 1 Honda. Surprisingly, the most sold car is a Mahindra, which only has a 4.6% market share. We can also see that the vast majority of car models appear only once in our data (at 862 distinct models), and 409 appear twice. This means that over half of our 2041 disinct car models appear only once or twice in our data. This makes predictions based on specific car models/names quite difficult, and we may end up dropping the 'Name' column in favour of 'Brand' from our data because of it unless we aggregate the data from the 'Name' column some other way. This is reinforced by seeing that only 13 unique car models appear 30 or more times in our data - not nearly enough to make any useful conclusions about trends and predictions in prices based off the 'Name' column

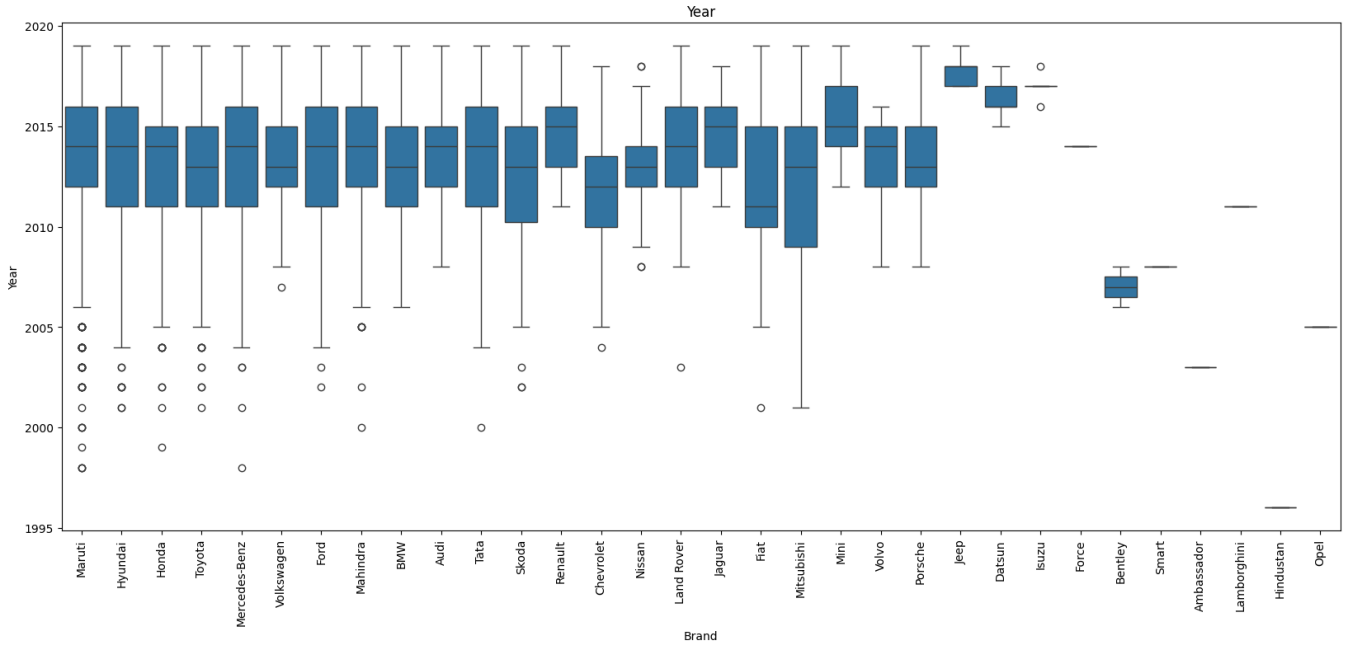
```
for i in cat2:
  if i=='seats_log':
    plt.figure(figsize=(20,10))
    plt.xticks(rotation=90)
    plt.title(i)
    sns.boxplot(data=df2,x=i,y='newprice_log',order=df2[i].value_counts().sort_index().index)
    plt.show()
    print('-'*250)
  else:
    plt.figure(figsize=(20,10))
    plt.xticks(rotation=90)
    plt.title(i)
    sns.boxplot(data=df2,x=i,y='newprice_log',order=df2[i].value_counts().index)
    plt.show()
    print('-'*250)
```






```
# Check specific relationships between Brand and Location, Brand and Year, & Brand and power_log
plt.figure(figsize=(20,10))
plt.title('Car brand count per location')
sns.countplot(data=df2,hue='Brand',x='Location',hue_order=df2.Brand.value_counts().index)
plt.legend(bbox_to_anchor=(1,1))
plt.show()
print('-'*250)

for i in ['Year','power_log','Engine','Mileage']:
  plt.figure(figsize=(20,8))
  plt.title(i)
  plt.xticks(rotation=90)
  sns.boxplot(data=df2,x='Brand',y=i,order=df2.Brand.value_counts().index)
  plt.show()
  print('-'*250)
```



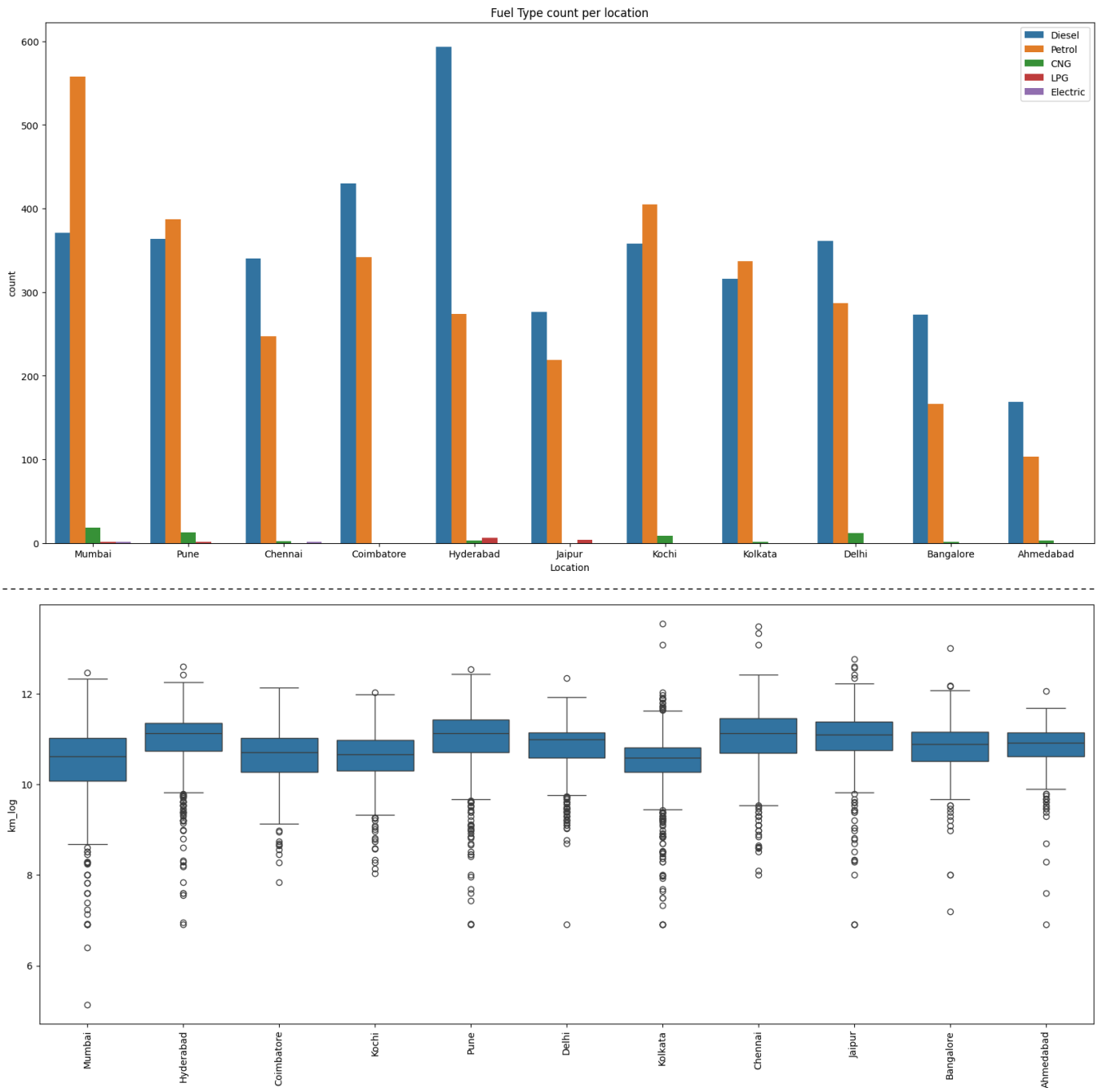


Observations

- **Brand and Location:** The distribution of brands at each location seem very similar! We see that the overall shape are near-identical, with some discrepencies in the order of the top 3. For example, most locations have Maruti Suzuki as their top brand, but Koimbatore, Kochi, Kolkata, and Bangalore have more sales of Hyundai
- **Brand and Year:** Surprisingly, there isn't much variation between the brands in terms of their distribution of year of manufacture (disregarding the without enough entries to have a realistic look at their distribution). It is interesting to note that Maruti Suzuki has more outliers towards the older years than the other brands - perhaps indicating that this brand has been consistently popular throughout the years, hence why we are seeing more resales nowadays
- **Brand and Power (log):** There is less consistency between brands here. The top 3 brands, on averagem have lower power_log scores than most of the other brands. It is reassuring to see brands such as Porsche and Jaguar have high power ratings as expected.
- **Brand and Engine displacement:** Similar to Power, the top 3 car brands have lower engine displacement values. This is an interesting quirk of the data - especially when we consider that both power and engine displacement are highly correlated with price.
- **Brand and Mileage:** There seems to be less of a conclusive trend here. We can note the a car's kmpl score is mildly negatively correlated with its price - possibly again providing a reason why Maruti has a higher than average Mileage statistic. Mileage is also moderately correlated to Engine

```
# Compare Location and Fuel Type, and Location and km_log
plt.figure(figsize=(20,10))
plt.title('Fuel Type count per location')
sns.countplot(data=df2,hue='Fuel_Type',x='Location',hue_order=df2.Fuel_Type.value_counts().index)
plt.legend(bbox_to_anchor=(1,1))
plt.show()
```

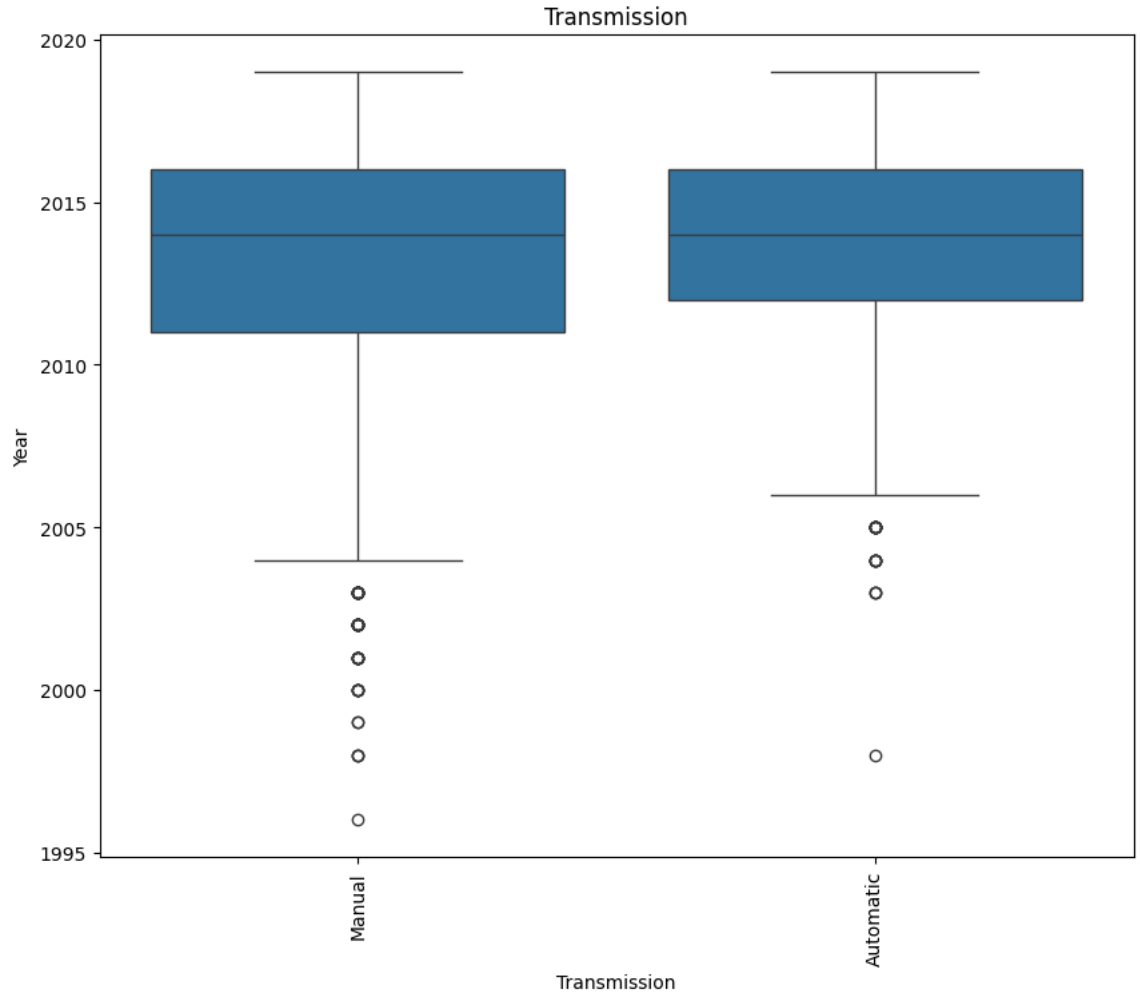
```
print('-'*250)

plt.figure(figsize=(20,8))
plt.xticks(rotation=90)
sns.boxplot(data=df2,x='Location',y='km_log',order=df2.Location.value_counts().index)
plt.show()
print('-'*250)
```

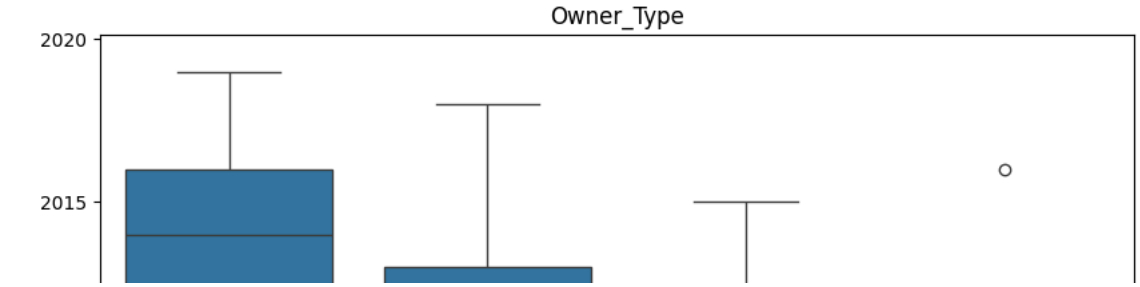


Observations

- **Location and Fuel Type:** A majority of locations have more Diesel sales than Petrol, which follows Diesel being the most frequent fuel type. Interestingly, the most frequent location, Mumbai, has a higher count of Petrol than Diesel. The data doesn't include additional information on the cities, such as the infrasctructure type or popoulation, so we cannot explain these differeces between cities with certainty
- **Location and Kilometers Driven:** Surprisingly, there isn't much difference between km driven between locations

```
# Relationship between Year and Transmission, Year and Owner Type, Year and Location, & Year and Fuel Type
for i in ['Transmission','Owner_Type','Location','Fuel_Type']:
  plt.figure(figsize=(10,8))
  plt.title(i)
  plt.xticks(rotation=90)
  sns.boxplot(data=df2,x=i,y='Year',order=df2[i].value_counts().index)
  plt.show()
  print('-'*250)
```





Observations

- **Year and Transmission:** The difference in manufacture year between automatic and manual cars is very small, surprisingly!
- **Year and Owner Type:** The manufacture year consistently decreases as the number of previous owners increases, which is what would normally be expected
- **Year and Location:** Thre isn't much variation between locations when it comes to the average year of manufacture of cars. This is with the exception of Coimbatore and Kochi, which each have a more recent average manufacture year
- **Year and Fuel_Type:** There isn't any strong relationship between Fuel Type and year of manufacture. Cars that run on LPG are on average older than other cars that run on other fuel types. This, combined with the fact there isn't much relationship between year and location, provides additional evidence that there may be an unseen factor affecting fuel types by location

Data Preprocessing

Missing value treatment

```
# Check missing values again
df2.isnull().sum().sort_values(ascending=False)
```

	0
newprice_log	6246
price_log	1234
Price	1234
power_log	175
Mileage	83
seats_log	53
Engine	46
Name	0
Brand	0
Location	0
Year	0
Fuel_Type	0
Transmission	0
Owner_Type	0
km_log	0

dtype: int64

Observations

- We will have to look at newprice_log, price, price_log, power_log, Mileage, seats_log, and Engine individually to treat their missing values.

Engine

```
# We know from previous analysis that the Engine distribution's median and mean are fairly close. We saw some high values earlier. Let's
print(df2.Engine.describe())
print()
print('There are',df2.loc[df2.Engine>3000].shape[0],'high-valued entries in the Engine column')
```

```
count    7206.000000
mean     1616.382459
std       595.105530
min        72.000000
25%      1198.000000
50%      1493.000000
75%      1968.000000
max      5998.000000
Name: Engine, dtype: float64

There are 65 high-valued entries in the Engine column
```

```
# Since there are only 65 entries with engine displacement of over 3000cc, they shouldn't affect our data too much, so we'll go with the
df2.Engine=df2.Engine.fillna(df2.Engine.median())
df2.Engine.isna().sum()
```

```
0
```

Seats (log)

```
# Check missing values
print(df2.seats_log.isna().sum(),' missing values')
missseat=df2.loc[seats_log.isnull()].Name.unique()
missseat
```

```
53  missing values
array(['Honda City 1.5 GXI', 'Maruti Swift 1.3 VXi', 'Ford Figo Diesel',
       'Land Rover Range Rover 3.0 D', 'Honda City 1.3 DX',
       'Maruti Swift 1.3 ZXI',
       'Land Rover Range Rover Sport 2005 2012 Sport',
       'Honda City 1.5 EXI', 'Maruti Swift 1.3 LXI',
       'Hyundai Santro LP zipPlus', 'Toyota Etios Liva V',
       'Maruti Swift 1.3 VXI ABS', 'Maruti Estilo LXI',
       'BMW 5 Series 520d Sedan', 'Hyundai Santro GLS II - Euro II',
       'Maruti Wagon R Vx', 'Ford Endeavour Hurricane LE',
       'Honda CR-V AT With Sun Roof', 'Fiat Punto 1.3 Emotion',
       'Hyundai Santro Xing XG', 'Honda Jazz Select Edition',
       'Fiat Punto 1.2 Dynamic', 'Fiat Punto 1.4 Emotion',
```

```
            'Skoda Laura 1.8 TSI Ambition',
            'Toyota Etios Liva Diesel TRD Sportivo',
            'Hyundai i20 new Sportz AT 1.4', 'Mahindra TUV 300 P4',
            'Honda Jazz 2020 Petrol'], dtype=object)
```

We now know the car models of missing values in the seats column. From this we can look at each name individually, and split them into two groups:

1. Those that have other instances with a non-null seats entry => Impute the seat number for the specific car model
2. Those where every entry of the model is missing an entry for seat number => Use the median value of seats for the brand

```
# Create list of all car models that only contain missing seat number values
missing_seats=[]
some_missing_seats=[]
for i in missseat:
  if all(df2.loc[df2.Name==i].seats_log.isna()): # This line checks whether every entry for seats is missing for the corresponding car r
    missing_seats.append(i)
  else: # This line checks whether every entry for seats is missing for the corresponding car model and if True, adds to the list
    some_missing_seats.append(i)
print('Missing every seats values:', missing_seats)
print()
print('Missing some seats values:',some_missing_seats)
```

```
Missing every seats values: ['Maruti Swift 1.3 VXi', 'Ford Figo Diesel', 'Land Rover Range Rover 3.0 D', 'Honda City 1.3 DX', 'Maruti

Missing some seats values: ['Honda City 1.5 GXI', 'Honda City 1.5 EXI', 'Toyota Etios Liva V', 'Honda Jazz Select Edition']
```

```
# Create function for later use
def st(hue,stat='median',cat='Brand',num='seats_log'):
  '''
  Function that returns the mean, median, mode of the given unique value of categorical variable with respect to a numeric feature.
  The default values are to make the seats_log missing value analysis easier

  hue == unique value corresponding to the chosen categorical variable
  stat == median, mode, or mean
  cat == chosen categorical variable
  num == chosen numerical variable
  '''
  msg=[]
  if df2[num].dtypes not in ['int64','float64']:
    msg.extend(['Please enter a valid numerical variable for the num argument:',df2.select_dtypes(include=['int64','float64']).columns.t
  elif df2[cat].dtypes!='O':
    msg.extend(['Please enter a valid categorical variable for the cat argument:',df2.select_dtypes(include='object').columns.to_list()]
  elif hue not in df2[cat].values:
    msg.append(['Please give a unique value for the chosen categorical variable. Here is a list of unique values in your chosen categor
  elif stat=='median':
    return df2.loc[df2[cat]==hue][num].median()
  elif stat=='mode':
    return df2.loc[df2[cat]==hue][num].mode()
  elif stat=='mean':
    return df2.loc[df2[cat]==hue][num].mean()
  else:
    msg.append("Not valid stat. Please enter 'median','mean', or 'mode' for the stat argument")
  print(str(msg))


for i in missing_seats:
  b=df2.loc[df2['Name']==i].Brand.unique() # Record the brand of the car model
  df2.loc[df2['Name']==i,'seats_log']=df2.loc[df2['Name']==i,'seats_log'].fillna(st(b[0])) # Impute missing values with median number se
df2.seats_log.isna().sum() # Check how many missing values are left in the 'seats_log' column
```

```
7
```

```
# Sanity check to make sure each car model only has one unique number of seats value
for i in some_missing_seats:
  print(i)
  print(df2.loc[df2['Name']==i,'seats_log'].value_counts(dropna=False))
  print('-'*50)
```

```
Honda City 1.5 GXI
seats_log
NaN         3
1.609438    2
Name: count, dtype: int64
--------------------------------------------------
Honda City 1.5 EXI
seats_log
1.609438    4
NaN         1
```

```
Name: count, dtype: int64
--------------------------------------------------
Toyota Etios Liva V
seats_log
NaN         2
1.609438    1
Name: count, dtype: int64
--------------------------------------------------
Honda Jazz Select Edition
seats_log
1.609438    1
NaN         1
Name: count, dtype: int64
--------------------------------------------------
```

We can confirm that the car models in the list some_missing_seats that do indeed have at least 1 entry in the seats_log column. We also see that this value is np.log(5) for every car so this makes filling the missing values easier. Finally, by can count instances of NaN above just as we expected

```
# Impute remaining missing values
df2.seats_log=df2.seats_log.fillna(np.log(5))
# Check we have filled every missing value in the 'seats_log' column
df2.seats_log.isnull().sum()
```

```
0
```

Mileage

```
# Check number missing values
df2.Mileage.isna().sum()
```

```
83
```

```
# Mileage is the closest feature we have that is normally distributed. We can go ahead and use the mean to impute missing values with th
df2.Mileage=df2.Mileage.fillna(df.Mileage.mean())
df2.Mileage.isna().sum()
```

```
0
```

Power (log)

```
# Check number missing values
df2.power_log.isna().sum()
```

```
175
```

```
# power_log has significantly reduced skew when compared to Power. The mean and median are pretty similar, so we can go ahead and use th
df2.power_log=df2.power_log.fillna(df.power_log.mean())
# Check we have filled every missing value
df2.power_log.isna().sum()
```

```
0
```

New_price (log)

```
# Check new price missing values percentage
newpricemissing=df2.newprice_log.isna().sum()*100/len(df2.newprice_log)
print('Percentage of missing values in the new_price column: ',round(newpricemissing,2),'%',sep='')
```

```
Percentage of missing values in the new_price column: 86.13%
```

```
# Since over 85% of the records have missing values in the newprice_log feature column, we remove the entire column to make our regressi
df3=df2.drop('newprice_log',axis=1)
# Check whether the newprice_log column has successfully been removed in our new dataframe
df3
```

	Name	Brand	Location	Year	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	km_log	power_log	seats_log
0	Maruti Wagon R LXI CNG	Maruti	Mumbai	2010	CNG	Manual	First	26.60	998.0	11.184421	4.063198	1.609438
1	Hyundai Creta 1.6 CRDi SX Option	Hyundai	Pune	2015	Diesel	Manual	First	19.67	1582.0	10.621327	4.837868	1.609438
2	Honda Jazz V	Honda	Chennai	2011	Petrol	Manual	First	18.20	1199.0	10.736397	4.485260	1.609438
3	Maruti Ertiga VDI	Maruti	Chennai	2012	Diesel	Manual	First	20.77	1248.0	11.373663	4.485936	1.945910
4	Audi A4 New 2.0 TDI Multitronic	Audi	Coimbatore	2013	Diesel	Automatic	Second	15.20	1968.0	10.613246	4.947340	1.609438
...	...	...	...	...	...	...	...	...	...	...	...	...
	Volkswagen											

```
# We will also keep df2 and impute the missing values in the newprice_log column to build two kinds of models, one with and one without
# Check the difference between imputing missing values with the mean vs with the median
check_mean=df2.newprice_log.fillna(df2.newprice_log.mean())
print('Impute with mean')
print(check_mean.describe())
print('-'*50)
check_median=df2.newprice_log.fillna(df2.newprice_log.median())
print('Impute with median')
print(check_median.describe())
print('-'*50)
print('Skew')
newprice_log.skew()
```

```
Impute with mean
count    7252.000000
mean        2.691519
std         0.321856
min         1.363537
25%         2.691519
50%         2.691519
75%         2.691519
max         5.926926
Name: newprice_log, dtype: float64
--------------------------------------------------
Impute with median
count    7252.000000
mean        2.482139
std         0.332646
min         1.363537
25%         2.448415
50%         2.448415
75%         2.448415
max         5.926926
Name: newprice_log, dtype: float64
--------------------------------------------------
Skew
0.7238320014363577
```

```
# We go with the median because the skew, although not ridiculously strong, is enough to cause worry if we were to impute with the mean
df2.newprice_log=df2.newprice_log.fillna(df.newprice_log.median())
# Check we have sucessfully removed all missing values
df2.newprice_log.isna().sum()
```

```
0
```

Price and Price (log)

- Since this is the dependent variable, we simply remove the records where we have missing values

```
# Sanity check of number missing values
print(df2.isna().sum())
```

```
Name              0
Brand             0
Location          0
Year              0
Fuel_Type         0
Transmission      0
```

```
Owner_Type         0
Mileage            0
Engine             0
km_log             0
power_log          0
seats_log          0
newprice_log       0
price_log       1234
Price           1234
dtype: int64
```

```
# Drop the rows with missing values
df2.dropna(axis=0,inplace=True)
# Make sure we don't have any more missing values in our DataFrame
print(df2.isnull().sum())
# Check shape of df2, they should be 7252-1234=6018 rows and 15 columns for df2 and 14 columns for df3
df2.shape
```

```
Name            0
Brand           0
Location        0
Year            0
Fuel_Type       0
Transmission    0
Owner_Type      0
Mileage         0
Engine          0
km_log          0
power_log       0
seats_log       0
newprice_log    0
price_log       0
Price           0
dtype: int64
(6018, 15)
```

```
# Final look at the summary statistics of df2
df2.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Name	6018	1876	Mahindra XUV500 W8 2WD	49	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Brand	6018	30	Maruti	1211	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Location	6018	11	Mumbai	790	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Year	6018.0	NaN	NaN	NaN	2013.357594	3.269677	1998.0	2011.0	2014.0	2016.0	2019.0
Fuel_Type	6018	5	Diesel	3204	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Transmission	6018	2	Manual	4299	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Owner_Type	6018	4	First	4928	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Mileage	6018.0	NaN	NaN	NaN	18.3427	4.151352	6.4	15.4	18.346859	21.1	33.54
Engine	6018.0	NaN	NaN	NaN	1620.281157	599.424157	72.0	1198.0	1493.0	1969.0	5998.0
km_log	6018.0	NaN	NaN	NaN	10.757961	0.713022	5.141664	10.434116	10.878047	11.198215	13.560618
power_log	6018.0	NaN	NaN	NaN	4.637298	0.411343	3.532226	4.356709	4.591071	4.927471	6.327937
seats_log	6018.0	NaN	NaN	NaN	1.653288	0.139545	0.693147	1.609438	1.609438	1.609438	2.302585
newprice_log	6018.0	NaN	NaN	NaN	2.484624	0.3323	1.363537	2.448415	2.448415	2.448415	5.438079
price_log	6018.0	NaN	NaN	NaN	1.824705	0.873606	-0.820981	1.252763	1.729884	2.297573	5.075174
Price	6018.0	NaN	NaN	NaN	9.470243	11.165926	0.44	3.5	5.64	9.95	160.0

Building Various Models

1. What we want to predict is the "Price". We will use the normalized version 'price_log' for modeling.
2. Before we proceed to the model, we'll have to encode categorical features. We will drop the categorical feature Name as it doesn't provide us with useful information.
3. We'll split the data into train and test, to be able to evaluate the model that we build on the train data.
4. Build Regression models using train data.
5. Evaluate the model performance.

Note: When building our models, we will be using df2 and df3 in parallel. Through this we will be able to compare the impact of removing the newprice_log column on our model performances

```
# Importing libraries for regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score, mean_absolute_percentage_error, mean_absolute_error, mean_squared_error
import statsmodels.api as sm
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
```

Split the Data

- Step1: Separate the indepdent variables (X) and the dependent variable (y).
- Step2: Encode the categorical variables in X using pd.dummies.
- Step3: Split the data into train and test using train_test_split.

Question:

1. Why we should drop 'Name', 'Price', 'price_log', 'Kilometers_Driven' from X before splitting?
    - We drop the Name feature because it has too many unique values. This makes it unsuitable if we want to build a good predictive algorithms
    - We drop the price_log and Price variable because they are our dependent (y) variable
    - We have already dropped Kilometers_Driven and the other independent variables that we replaced with log versions

For Regression Problems, some of the algorithms used are :

1) Linear Regression
2) Ordinary Least Squares Regression
3) Ridge / Lasso Regression
4) Decision Trees
5) Random Forest

```
# Define X independent variables data for df2
X = df2.drop(['Name','Price','price_log'],axis=1)
# Define X independent variables data without newprice_log
X2 = df2.drop(['Name','Price','price_log','newprice_log'],axis=1)
# Define y dependent variable data
y = df2[['price_log','Price']]
```

```
# Step 2: Encode cat variables using dummy variables
X = pd.get_dummies(X, drop_first = True)
X2= pd.get_dummies(X2,drop_first=True)
```

```
# Step 3: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 1)
X2_train,X2_test,y2_train,y2_test=train_test_split(X2,y,test_size=0.3,random_state=1)
```

```
# Metrics function: RMS, RMSE, MSE, R^2, MAE, MAPE
def performance(model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test,ls=False):
  perf=[]
  r2_train=r2_score(y_pred=np.exp(model.predict(X_train)), y_true=y_train['Price'])
  r2_test=r2_score(y_pred=np.exp(model.predict(X_test)), y_true=y_test['Price'])
  RMSE_train=np.sqrt(mean_squared_error(y_pred=np.exp(model.predict(X_train)), y_true=y_train['Price']))
  RMSE_test=np.sqrt(mean_squared_error(y_pred=np.exp(model.predict(X_test)), y_true=y_test['Price']))
  MAE_train=mean_absolute_error(y_pred=np.exp(model.predict(X_train)),y_true=y_train['Price'])
  MAE_test=mean_absolute_error(y_pred=np.exp(model.predict(X_test)),y_true=y_test['Price'])
  MAPE_train=mean_absolute_percentage_error(y_pred=np.exp(model.predict(X_train)),y_true=y_train['Price'])
  MAPE_test=mean_absolute_percentage_error(y_pred=np.exp(model.predict(X_test)),y_true=y_test['Price'])
  perf.extend((r2_train,r2_test,RMSE_train,RMSE_test,MAE_train,MAE_test,MAPE_train,MAPE_test))
  if ls==True:
    return dict(zip(['r^2 train','r^2 test','RMSE train','RMSE test','MAE train','MAE test','MAPE train','MAPE test'],perf))
```

Linear Regression

```
# Linear regression model with newprice_log
lreg=LinearRegression()
lreg.fit(X_train, y_train['price_log'])
```

```
linper1=performance(lreg,ls=True)
linper1
```

```
{'r^2 train': 0.8959013287155702,
 'r^2 test': 0.8918200219148048,
 'RMSE train': 3.604739299373938,
 'RMSE test': 3.6656659197644132,
 'MAE train': 1.6131691836306685,
 'MAE test': 1.679331730286168,
 'MAPE train': 0.18095614892301592,
 'MAPE test': 0.17203186451925814}
```

```
# Linear regression model without newprice_log
lreg2=LinearRegression()
lreg2.fit(X2_train,y2_train['price_log'])
linper2=performance(lreg2,X_train=X2_train,y_train=y2_train,X_test=X2_test,y_test=y2_test,ls=True)
linper2
```

```
{'r^2 train': 0.8963894393638642,
 'r^2 test': 0.8855697768616961,
 'RMSE train': 3.596278196950494,
 'RMSE test': 3.770073435318271,
 'MAE train': 1.6224931910008915,
 'MAE test': 1.6933540205345146,
 'MAPE train': 0.18093091642598016,
 'MAPE test': 0.17262773821240776}
```

OLS Regression

```
def vif(X_train_scaled):
  vif_scores = pd.Series([variance_inflation_factor(X_train_scaled.values, i) for i in range(len(X_train_scaled.columns))], index=X_trai
  print('VIF scores')
  print(vif_scores)
```

With newprice_log feature

```
# Scale variables
X_train_scaled = pd.DataFrame(MinMaxScaler().fit_transform(X_train),index=X_train.index,columns=X_train.columns)
X_test_scaled= pd.DataFrame(MinMaxScaler().fit_transform(X_test),index=X_test.index,columns=X_test.columns)
```

```
# Build OLS model
# Add intercept term in regression
X_train_scaled=sm.add_constant(X_train_scaled)
X_test_scaled=sm.add_constant(X_test_scaled)
# Create version 1 of model
olsmod1=sm.OLS(y_train['price_log'], X_train_scaled).fit()
# Model performance
print(olsmod1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.930
Model:                            OLS   Adj. R-squared:                  0.929
Method:                 Least Squares   F-statistic:                     1022.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):              0.00
Time:                        16:20:04   Log-Likelihood:                203.49
No. Observations:                4212   AIC:                            -297.0
Df Residuals:                    4157   BIC:                             52.03
Df Model:                          54
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                          -0.0930      0.246     -0.378      0.705      -0.575       0.389
Year                            2.4633      0.034     72.815      0.000       2.397       2.530
Mileage                        -0.2613      0.046     -5.621      0.000      -0.352      -0.170
Engine                          1.0686      0.085     12.563      0.000       0.902       1.235
km_log                         -0.5957      0.057    -10.510      0.000      -0.707      -0.485
power_log                       1.9057      0.062     30.715      0.000       1.784       2.027
seats_log                       0.2565      0.062      4.132      0.000       0.135       0.378
newprice_log                    0.1305      0.045      2.884      0.004       0.042       0.219
Brand_Audi                      0.0872      0.236      0.369      0.712      -0.376       0.551
Brand_BMW                       0.0476      0.237      0.201      0.841      -0.416       0.511
Brand_Bentley                   0.5266      0.335      1.571      0.116      -0.131       1.184
Brand_Chevrolet                -0.8284      0.236     -3.507      0.000      -1.292      -0.365
Brand_Datsun                   -0.8950      0.248     -3.610      0.000      -1.381      -0.409
Brand_Fiat                     -0.8455      0.241     -3.508      0.000      -1.318      -0.373
Brand_Force                    -0.4234      0.287     -1.475      0.140      -0.986       0.139
Brand_Ford                     -0.5969      0.235     -2.536      0.011      -1.058      -0.135
Brand_Honda                    -0.5319      0.236     -2.256      0.024      -0.994      -0.070
Brand_Hyundai                  -0.5524      0.235     -2.347      0.019      -1.014      -0.091
```

Brand_Isuzu	-0.8115	0.271	-2.997	0.003	-1.342	-0.281
Brand_Jaguar	0.1562	0.240	0.651	0.515	-0.314	0.627
Brand_Jeep	-0.4028	0.246	-1.637	0.102	-0.885	0.080
Brand_Lamborghini	1.0559	0.335	3.154	0.002	0.400	1.712
Brand_Land Rover	0.4066	0.238	1.705	0.088	-0.061	0.874
Brand_Mahindra	-0.6715	0.236	-2.849	0.004	-1.134	-0.209
Brand_Maruti	-0.5030	0.235	-2.138	0.033	-0.964	-0.042
Brand_Mercedes-Benz	0.1009	0.236	0.427	0.669	-0.362	0.564
Brand_Mini	0.4401	0.242	1.815	0.070	-0.035	0.916
Brand_Mitsubishi	-0.2789	0.241	-1.157	0.247	-0.751	0.194
Brand_Nissan	-0.5673	0.237	-2.397	0.017	-1.031	-0.103
Brand_Porsche	0.0912	0.245	0.372	0.710	-0.390	0.572
Brand_Renault	-0.5706	0.236	-2.416	0.016	-1.034	-0.108
Brand_Skoda	-0.4904	0.236	-2.077	0.038	-0.953	-0.027
Brand_Smart	-0.2530	0.333	-0.761	0.447	-0.905	0.399
Brand_Tata	-0.9860	0.236	-4.185	0.000	-1.448	-0.524
Brand_Toyota	-0.3424	0.235	-1.454	0.146	-0.804	0.119
Brand_Volkswagen	-0.5616	0.236	-2.385	0.017	-1.023	-0.100
Brand_Volvo	-0.1272	0.243	-0.524	0.601	-0.603	0.349
Location_Bangalore	0.1779	0.023	7.583	0.000	0.132	0.224
Location_Chennai	0.0223	0.022	1.002	0.316	-0.021	0.066
Location_Coimbatore	0.1188	0.021	5.569	0.000	0.077	0.161
Location_Delhi	-0.0599	0.022	-2.763	0.006	-0.102	-0.017
Location_Hyderabad	0.1255	0.021	6.039	0.000	0.085	0.166
Location_Jaipur	-0.0497	0.023	-2.175	0.030	-0.095	-0.005
Location_Kochi	-0.0263	0.021	-1.228	0.219	-0.068	0.016

Observations

- We see very high R^2 and adjusted R^2 values of 0.930 and 0.929 respectively. Before we have a look at the p-values, let's look at collinearity with VIF scores

```
# Check VIF scores for multicollinearity
vif(X_train_scaled)
```

```
VIF scores
const                        4727.065431
Year                            2.190916
Mileage                         3.977859
Engine                          7.208944
km_log                          1.795975
power_log                       6.590606
seats_log                       2.276907
newprice_log                    1.248835
Brand_Audi                    160.695761
Brand_BMW                     187.552116
Brand_Bentley                   2.085964
Brand_Chevrolet                87.288871
Brand_Datsun                   10.249069
Brand_Fiat                     20.400595
Brand_Force                     3.059614
Brand_Ford                    207.044419
Brand_Honda                   389.521580
Brand_Hyundai                 658.367353
Brand_Isuzu                     4.080738
Brand_Jaguar                   30.805406
Brand_Jeep                     12.329724
Brand_Lamborghini               2.080171
Brand_Land Rover               38.721937
Brand_Mahindra                183.422796
Brand_Maruti                  693.839583
Brand_Mercedes-Benz           211.305204
Brand_Mini                     18.485410
Brand_Mitsubishi               18.262270
Brand_Nissan                   68.591820
Brand_Porsche                  14.474388
Brand_Renault                 105.080496
Brand_Skoda                   129.500834
Brand_Smart                     2.053451
Brand_Tata                    134.686837
Brand_Toyota                  268.181889
Brand_Volkswagen              216.592654
Brand_Volvo                    17.453528
Location_Bangalore              2.366456
Location_Chennai                2.817520
Location_Coimbatore             3.449045
Location_Delhi                  3.026835
Location_Hyderabad              3.705900
Location_Jaipur                 2.579653
Location_Kochi                  3.430040
Location_Kolkata                3.030365
Location_Mumbai                 3.908529
Location_Pune                   3.264986
Fuel_Type_Diesel               29.543056
Fuel_Type_Electric              1.035553
Fuel_Type_LPG                   1.212623
Fuel_Type_Petrol               30.989757
Transmission_Manual             2.253404
```

```
Owner_Type_Fourth & Above       1.014139
Owner_Type_Second               1.182974
Owner_Type_Third                1.150417
dtype: float64
```

Observations

- We can ignore categorical variables with VIF values above 5 as these are expected when using dummy variables
- Engine and power_log have high VIF scores - we will remove Engine as it has the highest VIF score out of the numerical variables

```
# Collinearity check pt.2
X_train_scaled2=X_train_scaled.drop('Engine',axis=1)
vif(X_train_scaled2)
```

```
VIF scores
const                       4703.966374
Year                           2.187670
Mileage                        3.551060
km_log                         1.794817
power_log                      4.160777
seats_log                      2.251025
newprice_log                   1.246183
Brand_Audi                   160.527569
Brand_BMW                    187.388225
Brand_Bentley                  2.061577
Brand_Chevrolet               87.181236
Brand_Datsun                  10.241367
Brand_Fiat                    20.358805
Brand_Force                    3.058248
Brand_Ford                   206.896862
Brand_Honda                  389.073752
Brand_Hyundai                657.439890
Brand_Isuzu                    4.080730
Brand_Jaguar                  30.797396
Brand_Jeep                    12.306883
Brand_Lamborghini              2.067654
Brand_Land Rover              38.698116
Brand_Mahindra               183.332233
Brand_Maruti                 693.088075
Brand_Mercedes-Benz          211.188503
Brand_Mini                    18.466755
Brand_Mitsubishi              18.262119
Brand_Nissan                  68.530001
Brand_Porsche                 14.458298
Brand_Renault                104.958724
Brand_Skoda                  129.381419
Brand_Smart                    2.047501
Brand_Tata                   134.589227
Brand_Toyota                 268.153699
Brand_Volkswagen             216.324583
Brand_Volvo                   17.424677
Location_Bangalore             2.366042
Location_Chennai               2.817407
Location_Coimbatore            3.447176
Location_Delhi                 3.026293
Location_Hyderabad             3.703197
Location_Jaipur                2.578714
Location_Kochi                 3.429806
Location_Kolkata               3.030115
Location_Mumbai                3.907007
Location_Pune                  3.264978
Fuel_Type_Diesel              29.440761
Fuel_Type_Electric             1.035544
Fuel_Type_LPG                  1.209604
Fuel_Type_Petrol              30.297023
Transmission_Manual            2.253398
Owner_Type_Fourth & Above      1.014131
Owner_Type_Second              1.182971
Owner_Type_Third               1.150414
dtype: float64
```

```
# Update test set too
X_test_scaled2=X_test_scaled.drop('Engine',axis=1)
```

Observations

- All the VIF scores are acceptable now! Let's rebuild our model with this new training dataset

```
# Build second model
olsmod2=sm.OLS(y_train['price_log'], X_train_scaled2).fit()
print(olsmod2.summary())
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:              price_log   R-squared:                       0.927
Model:                            OLS   Adj. R-squared:                  0.926
Method:                 Least Squares   F-statistic:                     1000.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):              0.00
Time:                        16:20:07   Log-Likelihood:                125.01
No. Observations:                4212   AIC:                            -142.0
Df Residuals:                    4158   BIC:                             200.6
Df Model:                          53
Covariance Type:            nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                          0.1229      0.250      0.492      0.623      -0.367       0.613
Year                           2.4469      0.034     71.057      0.000       2.379       2.514
Mileage                       -0.4525      0.045    -10.116      0.000      -0.540      -0.365
km_log                        -0.6138      0.058    -10.633      0.000      -0.727      -0.501
power_log                      2.3789      0.050     47.372      0.000       2.280       2.477
seats_log                      0.3397      0.063      5.402      0.000       0.216       0.463
newprice_log                   0.1043      0.046      2.265      0.024       0.014       0.195
Brand_Audi                    -0.0089      0.241     -0.037      0.971      -0.481       0.463
Brand_BMW                     -0.0403      0.241     -0.167      0.867      -0.513       0.432
Brand_Bentley                  0.9820      0.339      2.893      0.004       0.316       1.648
Brand_Chevrolet               -0.9326      0.241     -3.878      0.000      -1.404      -0.461
Brand_Datsun                  -0.9804      0.252     -3.883      0.000      -1.475      -0.485
Brand_Fiat                    -0.9826      0.245     -4.006      0.000      -1.463      -0.502
Brand_Force                   -0.4996      0.292     -1.709      0.088      -1.073       0.074
Brand_Ford                    -0.6758      0.240     -2.820      0.005      -1.146      -0.206
Brand_Honda                   -0.6323      0.240     -2.634      0.008      -1.103      -0.162
Brand_Hyundai                 -0.6634      0.240     -2.769      0.006      -1.133      -0.194
Brand_Isuzu                   -0.8160      0.276     -2.959      0.003      -1.357      -0.275
Brand_Jaguar                   0.1076      0.244      0.440      0.660      -0.372       0.587
Brand_Jeep                    -0.5358      0.250     -2.140      0.032      -1.027      -0.045
Brand_Lamborghini              1.3821      0.340      4.065      0.000       0.716       2.049
Brand_Land Rover               0.3323      0.243      1.368      0.171      -0.144       0.808
Brand_Mahindra                -0.7374      0.240     -3.071      0.002      -1.208      -0.267
Brand_Maruti                  -0.6003      0.240     -2.506      0.012      -1.070      -0.131
Brand_Mercedes-Benz            0.0312      0.241      0.130      0.897      -0.440       0.503
Brand_Mini                     0.3433      0.247      1.390      0.164      -0.141       0.827
Brand_Mitsubishi              -0.2876      0.246     -1.171      0.241      -0.769       0.194
Brand_Nissan                  -0.6566      0.241     -2.724      0.006      -1.129      -0.184
Brand_Porsche                  0.1939      0.250      0.777      0.437      -0.296       0.683
Brand_Renault                 -0.6716      0.240     -2.793      0.005      -1.143      -0.200
Brand_Skoda                   -0.5805      0.240     -2.414      0.016      -1.052      -0.109
Brand_Smart                   -0.4779      0.338     -1.413      0.158      -1.141       0.185
Brand_Tata                    -1.0657      0.240     -4.442      0.000      -1.536      -0.595
Brand_Toyota                  -0.3728      0.240     -1.554      0.120      -0.843       0.097
Brand_Volkswagen              -0.6657      0.240     -2.776      0.006      -1.136      -0.196
Brand_Volvo                   -0.2512      0.247     -1.016      0.310      -0.736       0.233
Location_Bangalore             0.1818      0.024      7.607      0.000       0.135       0.229
Location_Chennai               0.0241      0.023      1.061      0.289      -0.020       0.069
Location_Coimbatore            0.1250      0.022      5.755      0.000       0.082       0.168
Location_Delhi                -0.0636      0.022     -2.878      0.004      -0.107      -0.020
Location_Hyderabad             0.1326      0.021      6.263      0.000       0.091       0.174
Location_Jaipur               -0.0443      0.023     -1.901      0.057      -0.090       0.001
Location_Kochi                -0.0240      0.022     -1.104      0.270      -0.067       0.019
Location_Kolkata              -0.2343      0.022    -10.512      0.000      -0.278      -0.191
```

```
# Remove features with p-values above 0.05, since we fail to reject the null hypothesis for these
pval=pd.DataFrame({'p':olsmod2.pvalues})
insig=pval.drop(pval[pval['p']<=0.05].index).index.tolist()
# Create new training data that doesn't include the insignificant features, and repeat
X_train_scaled3=X_train_scaled2.drop(insig,axis=1)
X_test_scaled3=X_test_scaled2.drop(insig,axis=1)


# New model
olsmod2_2=sm.OLS(y_train['price_log'], X_train_scaled3).fit()
print(olsmod2_2.summary())
```

```
                                 OLS Regression Results
==============================================================================================
Dep. Variable:              price_log   R-squared (uncentered):                   0.985
Model:                            OLS   Adj. R-squared (uncentered):              0.985
Method:                 Least Squares   F-statistic:                              7695.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):                        0.00
Time:                        16:20:07   Log-Likelihood:                         -117.55
No. Observations:                4212   AIC:                                      305.1
Df Residuals:                    4177   BIC:                                      527.2
Df Model:                          35
Covariance Type:            nonrobust
==============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Year                       2.4301      0.033     72.982      0.000       2.365       2.495
Mileage                   -0.3551      0.041     -8.721      0.000      -0.435      -0.275
km_log                    -0.6337      0.049    -12.836      0.000      -0.730      -0.537
power_log                  2.6404      0.046     57.689      0.000       2.551       2.730
seats_log                 -0.0201      0.054     -0.369      0.712      -0.127       0.087
newprice_log               0.1569      0.048      3.302      0.001       0.064       0.250
```

```
Brand_Bentley           0.8998      0.252      3.577      0.000       0.407       1.393
Brand_Chevrolet        -0.7424      0.030    -24.615      0.000      -0.802      -0.683
Brand_Datsun           -0.7463      0.086     -8.666      0.000      -0.915      -0.577
Brand_Fiat             -0.8104      0.059    -13.777      0.000      -0.926      -0.695
Brand_Ford             -0.4972      0.021    -23.591      0.000      -0.538      -0.456
Brand_Honda            -0.4787      0.018    -26.666      0.000      -0.514      -0.444
Brand_Hyundai          -0.4931      0.016    -30.394      0.000      -0.525      -0.461
Brand_Isuzu            -0.6771      0.145     -4.673      0.000      -0.961      -0.393
Brand_Jeep             -0.4217      0.077     -5.479      0.000      -0.573      -0.271
Brand_Lamborghini       1.1556      0.255      4.536      0.000       0.656       1.655
Brand_Mahindra         -0.4989      0.023    -22.013      0.000      -0.543      -0.455
Brand_Maruti           -0.4075      0.018    -22.646      0.000      -0.443      -0.372
Brand_Nissan           -0.4792      0.033    -14.362      0.000      -0.545      -0.414
Brand_Renault          -0.4986      0.028    -17.565      0.000      -0.554      -0.443
Brand_Skoda            -0.4622      0.024    -19.113      0.000      -0.510      -0.415
Brand_Tata             -0.8693      0.026    -33.391      0.000      -0.920      -0.818
Brand_Volkswagen       -0.4980      0.021    -23.776      0.000      -0.539      -0.457
Location_Bangalore      0.2097      0.017     12.134      0.000       0.176       0.244
Location_Coimbatore     0.1555      0.014     11.468      0.000       0.129       0.182
Location_Delhi         -0.0505      0.014     -3.526      0.000      -0.079      -0.022
Location_Hyderabad      0.1623      0.013     12.761      0.000       0.137       0.187
Location_Kolkata       -0.2108      0.015    -14.359      0.000      -0.240      -0.182
Location_Mumbai        -0.0226      0.013     -1.797      0.072      -0.047       0.002
Fuel_Type_Diesel        0.1571      0.035      4.440      0.000       0.088       0.226
Fuel_Type_Electric      0.8881      0.252      3.521      0.000       0.394       1.383
Fuel_Type_Petrol       -0.1203      0.033     -3.664      0.000      -0.185      -0.056
Transmission_Manual    -0.1627      0.012    -13.498      0.000      -0.186      -0.139
Owner_Type_Second      -0.0533      0.011     -4.771      0.000      -0.075      -0.031
Owner_Type_Third       -0.1132      0.029     -3.934      0.000      -0.170      -0.057
==============================================================================
Omnibus:                     1100.970   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            32237.412
Skew:                          -0.619   Prob(JB):                         0.00
Kurtosis:                      16.497   Cond. No.                          115.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant
```

```
# Repeat the above since seats_log and Mumbai still have p-values above 0.05
pval2=pd.DataFrame({'p':olsmod2_2.pvalues})
insig2=pval2.drop(pval2[pval2['p']<=0.05].index).index.tolist()
X_train_scaled4=X_train_scaled3.drop(insig2,axis=1)
X_test_scaled4=X_test_scaled3.drop(insig2,axis=1)
```

```
olsmod2_3=sm.OLS(y_train['price_log'], X_train_scaled4).fit()
print(olsmod2_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared (uncentered):                   0.985
Model:                            OLS   Adj. R-squared (uncentered):              0.985
Method:                 Least Squares   F-statistic:                              8158.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):                        0.00
Time:                        16:20:07   Log-Likelihood:                         -119.37
No. Observations:                4212   AIC:                                      304.7
Df Residuals:                    4179   BIC:                                      514.1
Df Model:                          33
Covariance Type:            nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Year                   2.4249      0.030     80.529      0.000       2.366       2.484
Mileage               -0.3522      0.039     -9.106      0.000      -0.428      -0.276
km_log                -0.6354      0.043    -14.866      0.000      -0.719      -0.552
power_log              2.6308      0.045     57.854      0.000       2.542       2.720
newprice_log           0.1506      0.047      3.182      0.001       0.058       0.243
Brand_Bentley          0.9077      0.252      3.609      0.000       0.415       1.401
Brand_Chevrolet       -0.7429      0.030    -24.635      0.000      -0.802      -0.684
Brand_Datsun          -0.7473      0.086     -8.688      0.000      -0.916      -0.579
Brand_Fiat            -0.8100      0.059    -13.776      0.000      -0.925      -0.695
Brand_Ford            -0.4984      0.021    -23.660      0.000      -0.540      -0.457
Brand_Honda           -0.4799      0.018    -26.750      0.000      -0.515      -0.445
Brand_Hyundai         -0.4931      0.016    -30.390      0.000      -0.525      -0.461
Brand_Isuzu           -0.6726      0.145     -4.642      0.000      -0.957      -0.389
Brand_Jeep            -0.4179      0.077     -5.439      0.000      -0.569      -0.267
Brand_Lamborghini      1.1724      0.253      4.632      0.000       0.676       1.669
Brand_Mahindra        -0.5026      0.022    -23.219      0.000      -0.545      -0.460
Brand_Maruti          -0.4091      0.018    -22.810      0.000      -0.444      -0.374
Brand_Nissan          -0.4806      0.033    -14.429      0.000      -0.546      -0.415
Brand_Renault         -0.4996      0.028    -17.610      0.000      -0.555      -0.444
Brand_Skoda           -0.4644      0.024    -19.256      0.000      -0.512      -0.417
Brand_Tata            -0.8692      0.026    -33.384      0.000      -0.920      -0.818
Brand_Volkswagen      -0.4989      0.021    -23.922      0.000      -0.540      -0.458
Location_Bangalore     0.2152      0.017     12.654      0.000       0.182       0.249
Location_Coimbatore    0.1616      0.013     12.263      0.000       0.136       0.187
Location_Delhi        -0.0451      0.014     -3.223      0.001      -0.072      -0.018
Location_Hyderabad     0.1672      0.012     13.465      0.000       0.143       0.191
Location_Kolkata      -0.2053      0.014    -14.373      0.000      -0.233      -0.177
```

```
Fuel_Type_Diesel        0.1510      0.034      4.413      0.000       0.084       0.218
Fuel_Type_Electric      0.8616      0.252      3.421      0.001       0.368       1.355
Fuel_Type_Petrol       -0.1276      0.032     -3.991      0.000      -0.190      -0.065
Transmission_Manual    -0.1639      0.012    -13.927      0.000      -0.187      -0.141
Owner_Type_Second      -0.0531      0.011     -4.755      0.000      -0.075      -0.031
Owner_Type_Third       -0.1128      0.029     -3.926      0.000      -0.169      -0.056
==============================================================================
Omnibus:                     1089.607   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            31212.894
Skew:                          -0.612   Prob(JB):                         0.00
Kurtosis:                      16.280   Cond. No.                         107.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Observations

- We have removed features with p-values lower than 0.05. Our R^2 scores between our training data and test data seem to have gotten slightly closer - however not by much
- We can now check whether our model fulfills the linear regression assumptions

```
# Rename the final model
ols_model1=olsmod2_3
```

Linear Regression Assumptions

```
# 1. Residuals mean = 0
res=ols_model1.resid

np.mean(res)
```

```
0.00043433776237101324
```

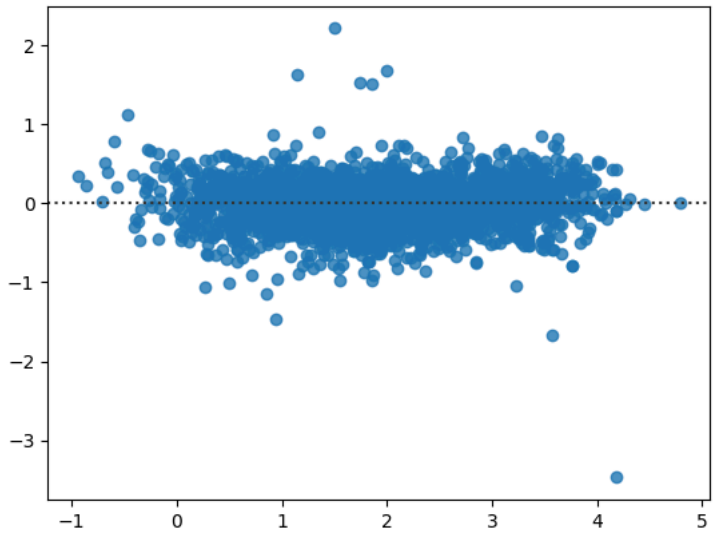
Observations

- The mean value of redisuals for our model is incredibly close to zero, just as we want

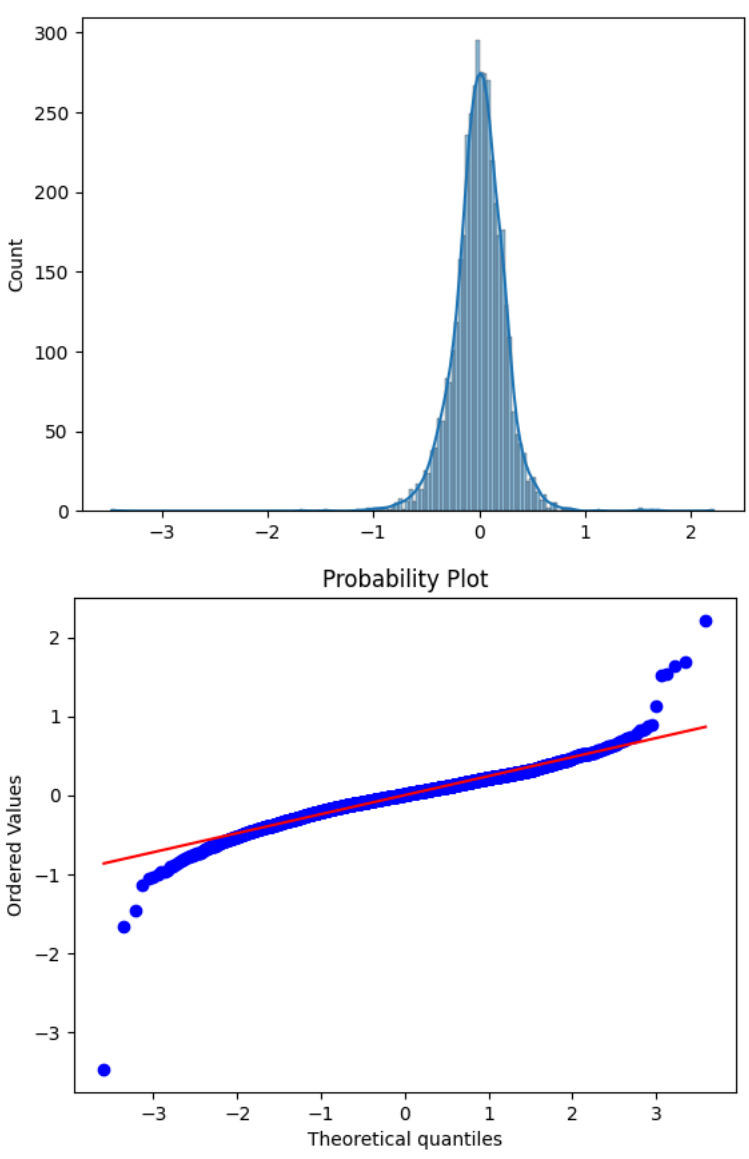
```
# 2. Goldfeld-Quandt test for heteroskedasticity
from statsmodels.stats.diagnostic import het_white
from statsmodels.compat import lzip
import statsmodels.stats.api as sms
import statsmodels.stats.api as sms

s = ["F statistic", "p-value"]
gfq = sms.het_goldfeldquandt(y_train['price_log'], X_train_scaled4)
lzip(s, gfq)
```

```
[('F statistic', 1.0776963200847345), ('p-value', 0.04422614636518398)]
```

```
# Linearity of variables, i.e. residuals should be randomly distributed
# Predicted values
x = ols_model1.fittedvalues
# Residual plot
sns.residplot(x = x, y = res)
plt.show()
```



```
# Check for normality of error terms
# Residuals histogram
sns.histplot(res, kde = True)
plt.show()

# q-q plot of residuals to check for normality (plot must follow linear path)
import pylab
import scipy.stats as stats

stats.probplot(res, dist = "norm", plot = pylab)
plt.show()
```



Observations

- Our OLS model satisfies all the linear regression assumptions except homoskedasticity

```
# Finally check the model's performance on the training and test sets
olsper1=performance(ols_model1,X_train=X_train_scaled4,X_test=X_test_scaled4,ls=True)
olsper1
```

```
{'r^2 train': 0.8677586392713029,
 'r^2 test': 0.8089042876449293,
 'RMSE train': 4.062888677419786,
 'RMSE test': 4.871973402524837,
 'MAE train': 1.7800895119007885,
 'MAE test': 2.3144109843761163,
 'MAPE train': 0.1922963877120215,
 'MAPE test': 0.24180689760238827}
```

Without newprice_log

```
# Scale variables
X2_train_scaled = pd.DataFrame(MinMaxScaler().fit_transform(X2_train),index=X_train.index,columns=X2_train.columns)
X2_test_scaled= pd.DataFrame(MinMaxScaler().fit_transform(X2_test),index=X_test.index,columns=X2_test.columns)
```

```
# Build OLS model
# Add intercept term in regression
X2_train_scaled=sm.add_constant(X2_train_scaled)
X2_test_scaled=sm.add_constant(X2_test_scaled)
# Create version 1 of model
olsmoda=sm.OLS(y2_train['price_log'], X2_train_scaled).fit()
# Model performance
print(olsmoda.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.930
Model:                            OLS   Adj. R-squared:                  0.929
Method:                 Least Squares   F-statistic:                     1039.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):               0.00
Time:                        16:20:08   Log-Likelihood:                199.28
No. Observations:                4212   AIC:                            -290.6
Df Residuals:                    4158   BIC:                             52.10
Df Model:                          53
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           -0.0558      0.246     -0.227      0.821      -0.538       0.426
Year                             2.4648      0.034     72.803      0.000       2.398       2.531
Mileage                         -0.2556      0.046     -5.498      0.000      -0.347      -0.164
Engine                           1.0573      0.085     12.432      0.000       0.891       1.224
km_log                          -0.5986      0.057    -10.553      0.000      -0.710      -0.487
power_log                        1.9298      0.062     31.364      0.000       1.809       2.050
seats_log                        0.2593      0.062      4.173      0.000       0.137       0.381
Brand_Audi                       0.0782      0.237      0.331      0.741      -0.386       0.542
Brand_BMW                        0.0441      0.237      0.186      0.852      -0.420       0.508
Brand_Bentley                    0.5122      0.335      1.527      0.127      -0.146       1.170
Brand_Chevrolet                 -0.8380      0.236     -3.545      0.000      -1.302      -0.375
Brand_Datsun                    -0.9232      0.248     -3.723      0.000      -1.409      -0.437
Brand_Fiat                      -0.8564      0.241     -3.550      0.000      -1.329      -0.384
Brand_Force                     -0.4359      0.287     -1.517      0.129      -0.999       0.127
Brand_Ford                      -0.6070      0.236     -2.577      0.010      -1.069      -0.145
Brand_Honda                     -0.5435      0.236     -2.303      0.021      -1.006      -0.081
Brand_Hyundai                   -0.5632      0.236     -2.391      0.017      -1.025      -0.101
Brand_Isuzu                     -0.8116      0.271     -2.995      0.003      -1.343      -0.280
Brand_Jaguar                     0.1417      0.240      0.590      0.555      -0.329       0.613
Brand_Jeep                      -0.3945      0.246     -1.602      0.109      -0.877       0.088
Brand_Lamborghini                1.0415      0.335      3.109      0.002       0.385       1.698
Brand_Land Rover                 0.4020      0.239      1.684      0.092      -0.066       0.870
Brand_Mahindra                  -0.6830      0.236     -2.895      0.004      -1.145      -0.220
Brand_Maruti                    -0.5152      0.235     -2.188      0.029      -0.977      -0.054
Brand_Mercedes-Benz              0.0945      0.236      0.400      0.689      -0.369       0.558
Brand_Mini                       0.4534      0.243      1.868      0.062      -0.022       0.929
Brand_Mitsubishi                -0.2782      0.241     -1.153      0.249      -0.751       0.195
Brand_Nissan                    -0.5780      0.237     -2.440      0.015      -1.042      -0.114
Brand_Porsche                    0.0818      0.245      0.333      0.739      -0.399       0.563
Brand_Renault                   -0.5849      0.236     -2.475      0.013      -1.048      -0.122
Brand_Skoda                     -0.4995      0.236     -2.113      0.035      -0.963      -0.036
Brand_Smart                     -0.2642      0.333     -0.794      0.427      -0.917       0.388
Brand_Tata                      -0.9977      0.236     -4.231      0.000      -1.460      -0.535
Brand_Toyota                    -0.3491      0.236     -1.481      0.139      -0.811       0.113
Brand_Volkswagen                -0.5722      0.236     -2.428      0.015      -1.034      -0.110
Brand_Volvo                     -0.1345      0.243     -0.553      0.580      -0.611       0.342
Location_Bangalore               0.1763      0.023      7.510      0.000       0.130       0.222
Location_Chennai                 0.0209      0.022      0.939      0.348      -0.023       0.065
```

```
Location_Coimbatore          0.1182      0.021      5.538      0.000       0.076       0.160
Location_Delhi              -0.0620      0.022     -2.856      0.004      -0.105      -0.019
Location_Hyderabad           0.1250      0.021      6.007      0.000       0.084       0.166
Location_Jaipur             -0.0503      0.023     -2.198      0.028      -0.095      -0.005
Location_Kochi              -0.0270      0.021     -1.263      0.207      -0.069       0.015
Location_Kolkata            -0.2385      0.022    -10.893      0.000      -0.281      -0.196
```

```
# Check VIF scores
vif(X2_train_scaled)
```

```
VIF scores
const                        4713.992297
Year                            2.190402
Mileage                         3.970639
Engine                          7.193633
km_log                          1.795409
power_log                       6.470574
seats_log                       2.276386
Brand_Audi                    160.667806
Brand_BMW                     187.547332
Brand_Bentley                   2.085497
Brand_Chevrolet                87.271498
Brand_Datsun                   10.233123
Brand_Fiat                     20.395604
Brand_Force                     3.058917
Brand_Ford                    206.998025
Brand_Honda                   389.407126
Brand_Hyundai                 658.199751
Brand_Isuzu                     4.080738
Brand_Jaguar                   30.791753
Brand_Jeep                     12.328060
Brand_Lamborghini               2.079712
Brand_Land Rover               38.720236
Brand_Mahindra                183.371143
Brand_Maruti                  693.616936
Brand_Mercedes-Benz           211.286512
Brand_Mini                     18.478755
Brand_Mitsubishi               18.262253
Brand_Nissan                   68.575209
Brand_Porsche                  14.471821
Brand_Renault                 105.034055
Brand_Skoda                   129.477686
Brand_Smart                     2.053172
Brand_Tata                    134.647312
Brand_Toyota                  268.155738
Brand_Volkswagen              216.540176
Brand_Volvo                    17.451621
Location_Bangalore              2.365120
Location_Chennai                2.816212
Location_Coimbatore             3.448749
Location_Delhi                  3.023625
Location_Hyderabad              3.705586
Location_Jaipur                 2.579472
Location_Kochi                  3.429512
Location_Kolkata                3.028230
Location_Mumbai                 3.908523
Location_Pune                   3.263298
Fuel_Type_Diesel               29.523304
Fuel_Type_Electric              1.035553
Fuel_Type_LPG                   1.212003
Fuel_Type_Petrol               30.978184
Transmission_Manual             2.251047
Owner_Type_Fourth & Above       1.014131
Owner_Type_Second               1.182848
Owner_Type_Third                1.150414
dtype: float64
```

Observations

- We can ignore the high VIF scores of categorical variables
- Engine and power_log have high VIF scores

```
# Drop the 'Engine' variable
X2_train_scaled2=X2_train_scaled.drop('Engine',axis=1)
X2_test_scaled2=X2_test_scaled.drop('Engine',axis=1)
```

```
# Check VIF scores again
vif(X2_train_scaled2)
```

```
VIF scores
const                        4692.421361
Year                            2.187028
Mileage                         3.537789
km_log                          1.794323
power_log                       4.085201
seats_log                       2.250787
```

```
Brand_Audi                    160.505530
Brand_BMW                     187.385668
Brand_Bentley                   2.060745
Brand_Chevrolet                87.167591
Brand_Datsun                   10.226394
Brand_Fiat                     20.355049
Brand_Force                     3.057637
Brand_Ford                    206.857698
Brand_Honda                   388.979013
Brand_Hyundai                 657.306375
Brand_Isuzu                     4.080730
Brand_Jaguar                   30.784663
Brand_Jeep                     12.304598
Brand_Lamborghini               2.066946
Brand_Land Rover               38.696948
Brand_Mahindra                183.286594
Brand_Maruti                  692.901137
Brand_Mercedes-Benz           211.173837
Brand_Mini                     18.459017
Brand_Mitsubishi               18.262097
Brand_Nissan                   68.516183
Brand_Porsche                  14.455098
Brand_Renault                 104.918870
Brand_Skoda                   129.362824
Brand_Smart                     2.047328
Brand_Tata                    134.555147
Brand_Toyota                  268.129941
Brand_Volkswagen              216.282378
Brand_Volvo                    17.423390
Location_Bangalore              2.364633
Location_Chennai                2.816061
Location_Coimbatore             3.446807
Location_Delhi                  3.023197
Location_Hyderabad              3.702791
Location_Jaipur                 2.578492
Location_Kochi                  3.429243
Location_Kolkata                3.027908
Location_Mumbai                 3.907006
Location_Pune                   3.263297
Fuel_Type_Diesel               29.416597
Fuel_Type_Electric              1.035544
Fuel_Type_LPG                   1.208850
Fuel_Type_Petrol               30.275679
Transmission_Manual             2.251047
Owner_Type_Fourth & Above       1.014122
Owner_Type_Second               1.182846
Owner_Type_Third                1.150410
dtype: float64
```

Observations

- VIF scores are now acceptable

```
# Rebuild model
olsmodb=sm.OLS(y2_train['price_log'], X2_train_scaled2).fit()
# Model performance
print(olsmodb.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.927
Model:                            OLS   Adj. R-squared:                  0.926
Method:                 Least Squares   F-statistic:                     1019.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):              0.00
Time:                        16:20:10   Log-Likelihood:                122.42
No. Observations:                4212   AIC:                            -138.8
Df Residuals:                    4159   BIC:                             197.5
Df Model:                          52
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                            0.1509      0.250      0.604      0.546      -0.339       0.640
Year                             2.4483      0.034     71.071      0.000       2.381       2.516
Mileage                         -0.4464      0.045     -9.991      0.000      -0.534      -0.359
km_log                          -0.6160      0.058    -10.667      0.000      -0.729      -0.503
power_log                        2.3943      0.050     48.093      0.000       2.297       2.492
seats_log                        0.3412      0.063      5.423      0.000       0.218       0.464
Brand_Audi                      -0.0153      0.241     -0.063      0.949      -0.487       0.457
Brand_BMW                       -0.0423      0.241     -0.175      0.861      -0.515       0.430
Brand_Bentley                    0.9665      0.340      2.846      0.004       0.301       1.632
Brand_Chevrolet                 -0.9394      0.241     -3.905      0.000      -1.411      -0.468
Brand_Datsun                    -1.0023      0.252     -3.971      0.000      -1.497      -0.507
Brand_Fiat                      -0.9901      0.245     -4.035      0.000      -1.471      -0.509
Brand_Force                     -0.5090      0.293     -1.740      0.082      -1.083       0.065
Brand_Ford                      -0.6833      0.240     -2.850      0.004      -1.153      -0.213
Brand_Honda                     -0.6408      0.240     -2.668      0.008      -1.112      -0.170
Brand_Hyundai                   -0.6711      0.240     -2.800      0.005      -1.141      -0.201
```

Brand_Isuzu	-0.8161	0.276	-2.957	0.003	-1.357	-0.275
Brand_Jaguar	0.0963	0.245	0.394	0.694	-0.383	0.576
Brand_Jeep	-0.5281	0.250	-2.108	0.035	-1.019	-0.037
Brand_Lamborghini	1.3679	0.340	4.022	0.000	0.701	2.035
Brand_Land Rover	0.3293	0.243	1.355	0.175	-0.147	0.806
Brand_Mahindra	-0.7459	0.240	-3.106	0.002	-1.217	-0.275
Brand_Maruti	-0.6092	0.240	-2.542	0.011	-1.079	-0.139
Brand_Mercedes-Benz	0.0266	0.241	0.111	0.912	-0.445	0.498
Brand_Mini	0.3548	0.247	1.436	0.151	-0.129	0.839
Brand_Mitsubishi	-0.2870	0.246	-1.168	0.243	-0.769	0.195
Brand_Nissan	-0.6644	0.241	-2.755	0.006	-1.137	-0.192
Brand_Porsche	0.1855	0.250	0.743	0.458	-0.304	0.675
Brand_Renault	-0.6823	0.241	-2.836	0.005	-1.154	-0.211
Brand_Skoda	-0.5871	0.241	-2.440	0.015	-1.059	-0.115
Brand_Smart	-0.4850	0.338	-1.433	0.152	-1.149	0.179
Brand_Tata	-1.0743	0.240	-4.476	0.000	-1.545	-0.604
Brand_Toyota	-0.3779	0.240	-1.575	0.115	-0.848	0.093
Brand_Volkswagen	-0.6733	0.240	-2.807	0.005	-1.144	-0.203
Brand_Volvo	-0.2560	0.247	-1.035	0.301	-0.741	0.229
Location_Bangalore	0.1805	0.024	7.551	0.000	0.134	0.227
Location_Chennai	0.0230	0.023	1.012	0.312	-0.022	0.067
Location_Coimbatore	0.1245	0.022	5.729	0.000	0.082	0.167
Location_Delhi	-0.0652	0.022	-2.950	0.003	-0.108	-0.022
Location_Hyderabad	0.1321	0.021	6.237	0.000	0.091	0.174
Location_Jaipur	-0.0448	0.023	-1.921	0.055	-0.090	0.001
Location_Kochi	-0.0247	0.022	-1.132	0.258	-0.067	0.018
Location_Kolkata	-0.2357	0.022	-10.572	0.000	-0.279	-0.192
Location_Mumbai	-0.0490	0.021	-2.314	0.021	-0.091	-0.007

```
# Remove insignifcant features with p-values above 0.05
pvala=pd.DataFrame({'p':olsmodb.pvalues})
insiga=pvala.drop(pvala[pvala['p']<=0.05].index).index.tolist()
X2_train_scaled3=X2_train_scaled2.drop(insiga,axis=1)
X2_test_scaled3=X2_test_scaled2.drop(insiga,axis=1)
```

```
# Build updated model
olsmodc=sm.OLS(y2_train['price_log'], X2_train_scaled3).fit()
# Model performance
print(olsmodc.summary())
```

```
                                 OLS Regression Results
================================================================================
Dep. Variable:              price_log   R-squared (uncentered):                   0.985
Model:                            OLS   Adj. R-squared (uncentered):              0.985
Method:                 Least Squares   F-statistic:                              7902.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):                        0.00
Time:                        16:20:11   Log-Likelihood:                         -123.04
No. Observations:                4212   AIC:                                      314.1
Df Residuals:                    4178   BIC:                                      529.8
Df Model:                          34
Covariance Type:            nonrobust
================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Year                     2.4370      0.033     73.246      0.000       2.372       2.502
Mileage                 -0.3352      0.040     -8.313      0.000      -0.414      -0.256
km_log                  -0.6224      0.049    -12.622      0.000      -0.719      -0.526
power_log                2.6718      0.045     59.608      0.000       2.584       2.760
seats_log               -0.0082      0.054     -0.150      0.881      -0.115       0.099
Brand_Bentley            0.8836      0.252      3.510      0.000       0.390       1.377
Brand_Chevrolet         -0.7456      0.030    -24.704      0.000      -0.805      -0.686
Brand_Datsun            -0.7728      0.086     -9.003      0.000      -0.941      -0.604
Brand_Fiat              -0.8146      0.059    -13.835      0.000      -0.930      -0.699
Brand_Ford              -0.5012      0.021    -23.798      0.000      -0.543      -0.460
Brand_Honda             -0.4857      0.018    -27.207      0.000      -0.521      -0.451
Brand_Hyundai           -0.4981      0.016    -30.789      0.000      -0.530      -0.466
Brand_Isuzu             -0.6678      0.145     -4.604      0.000      -0.952      -0.383
Brand_Jeep              -0.4023      0.077     -5.236      0.000      -0.553      -0.252
Brand_Lamborghini        1.1479      0.255      4.501      0.000       0.648       1.648
Brand_Mahindra          -0.5054      0.023    -22.355      0.000      -0.550      -0.461
Brand_Maruti            -0.4143      0.018    -23.148      0.000      -0.449      -0.379
Brand_Nissan            -0.4841      0.033    -14.504      0.000      -0.550      -0.419
Brand_Renault           -0.5077      0.028    -17.949      0.000      -0.563      -0.452
Brand_Skoda             -0.4652      0.024    -19.226      0.000      -0.513      -0.418
Brand_Tata              -0.8748      0.026    -33.628      0.000      -0.926      -0.824
Brand_Volkswagen        -0.5024      0.021    -24.005      0.000      -0.543      -0.461
Location_Bangalore       0.2092      0.017     12.091      0.000       0.175       0.243
Location_Coimbatore      0.1556      0.014     11.458      0.000       0.129       0.182
Location_Delhi          -0.0518      0.014     -3.615      0.000      -0.080      -0.024
Location_Hyderabad       0.1632      0.013     12.822      0.000       0.138       0.188
Location_Kolkata        -0.2110      0.015    -14.356      0.000      -0.240      -0.182
Location_Mumbai         -0.0206      0.013     -1.637      0.102      -0.045       0.004
Fuel_Type_Diesel         0.1667      0.035      4.725      0.000       0.098       0.236
Fuel_Type_Electric       0.8954      0.253      3.545      0.000       0.400       1.390
Fuel_Type_Petrol        -0.1082      0.033     -3.313      0.001      -0.172      -0.044
Transmission_Manual     -0.1641      0.012    -13.606      0.000      -0.188      -0.140
Owner_Type_Second       -0.0531      0.011     -4.750      0.000      -0.075      -0.031
Owner_Type_Third        -0.1118      0.029     -3.880      0.000      -0.168      -0.055
```

```
==============================================================================
Omnibus:                     1032.190   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27676.931
Skew:                          -0.561   Prob(JB):                         0.00
Kurtosis:                      15.508   Cond. No.                         114.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Remove seats_log and Mumbai features
pvalb=pd.DataFrame({'p':olsmodc.pvalues})
insigb=pvalb.drop(pvalb[pvalb['p']<=0.05].index).index.tolist()
X2_train_scaled4=X2_train_scaled3.drop(insigb,axis=1)
X2_test_scaled4=X2_test_scaled3.drop(insigb,axis=1)
```

```
# Build updated model
olsmodd=sm.OLS(y2_train['price_log'], X2_train_scaled4).fit()
# Model performance
print(olsmodd.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared (uncentered):                   0.985
Model:                            OLS   Adj. R-squared (uncentered):              0.985
Method:                 Least Squares   F-statistic:                              8394.
Date:                Mon, 25 Nov 2024   Prob (F-statistic):                        0.00
Time:                        16:20:11   Log-Likelihood:                         -124.46
No. Observations:                4212   AIC:                                      312.9
Df Residuals:                    4180   BIC:                                      516.0
Df Model:                          32
Covariance Type:            nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Year                   2.4346      0.030     81.178      0.000       2.376       2.493
Mileage               -0.3354      0.038     -8.745      0.000      -0.411      -0.260
km_log                -0.6201      0.043    -14.584      0.000      -0.703      -0.537
power_log              2.6621      0.044     59.889      0.000       2.575       2.749
Brand_Bentley          0.8904      0.252      3.537      0.000       0.397       1.384
Brand_Chevrolet       -0.7457      0.030    -24.715      0.000      -0.805      -0.687
Brand_Datsun          -0.7721      0.086     -9.003      0.000      -0.940      -0.604
Brand_Fiat            -0.8144      0.059    -13.839      0.000      -0.930      -0.699
Brand_Ford            -0.5023      0.021    -23.859      0.000      -0.544      -0.461
Brand_Honda           -0.4866      0.018    -27.274      0.000      -0.522      -0.452
Brand_Hyundai         -0.4979      0.016    -30.783      0.000      -0.530      -0.466
Brand_Isuzu           -0.6643      0.145     -4.580      0.000      -0.949      -0.380
Brand_Jeep            -0.4004      0.077     -5.219      0.000      -0.551      -0.250
Brand_Lamborghini      1.1582      0.253      4.571      0.000       0.661       1.655
Brand_Mahindra        -0.5073      0.022    -23.466      0.000      -0.550      -0.465
Brand_Maruti          -0.4153      0.018    -23.266      0.000      -0.450      -0.380
Brand_Nissan          -0.4855      0.033    -14.575      0.000      -0.551      -0.420
Brand_Renault         -0.5085      0.028    -17.989      0.000      -0.564      -0.453
Brand_Skoda           -0.4672      0.024    -19.367      0.000      -0.515      -0.420
Brand_Tata            -0.8745      0.026    -33.622      0.000      -0.926      -0.824
Brand_Volkswagen      -0.5033      0.021    -24.165      0.000      -0.544      -0.463
Location_Bangalore     0.2142      0.017     12.587      0.000       0.181       0.248
Location_Coimbatore    0.1610      0.013     12.203      0.000       0.135       0.187
Location_Delhi        -0.0468      0.014     -3.343      0.001      -0.074      -0.019
Location_Hyderabad     0.1677      0.012     13.491      0.000       0.143       0.192
Location_Kolkata      -0.2058      0.014    -14.397      0.000      -0.234      -0.178
Fuel_Type_Diesel       0.1626      0.034      4.771      0.000       0.096       0.229
Fuel_Type_Electric     0.8729      0.252      3.463      0.001       0.379       1.367
Fuel_Type_Petrol      -0.1140      0.032     -3.592      0.000      -0.176      -0.052
Transmission_Manual   -0.1647      0.012    -13.981      0.000      -0.188      -0.142
Owner_Type_Second     -0.0529      0.011     -4.730      0.000      -0.075      -0.031
Owner_Type_Third      -0.1111      0.029     -3.866      0.000      -0.168      -0.055
==============================================================================
Omnibus:                     1025.305   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27021.268
Skew:                          -0.558   Prob(JB):                         0.00
Kurtosis:                      15.358   Cond. No.                         106.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Rename final model
ols_model2=olsmodd
```

Check linear regression assumptions

```
# 1. Residuals mean = 0
resa=ols_model2.resid

np.mean(res)
```

```
0.00043433776237101324
```

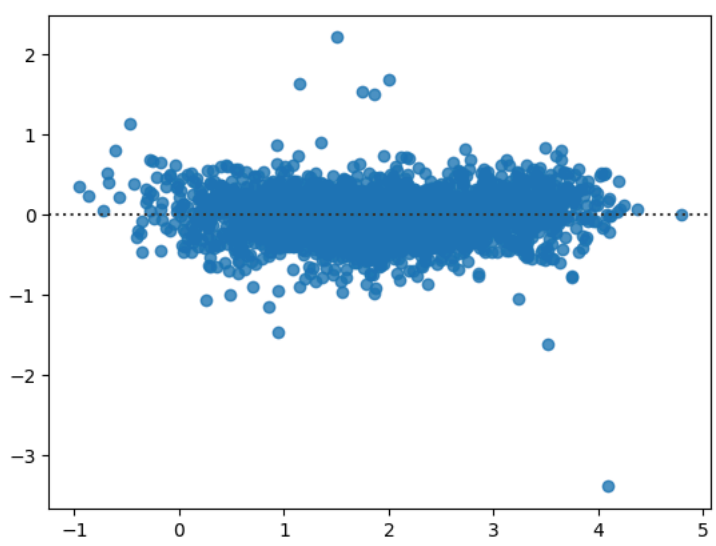
Observations

- The mean value of redisuals for our model is incredibly close to zero, just as we want

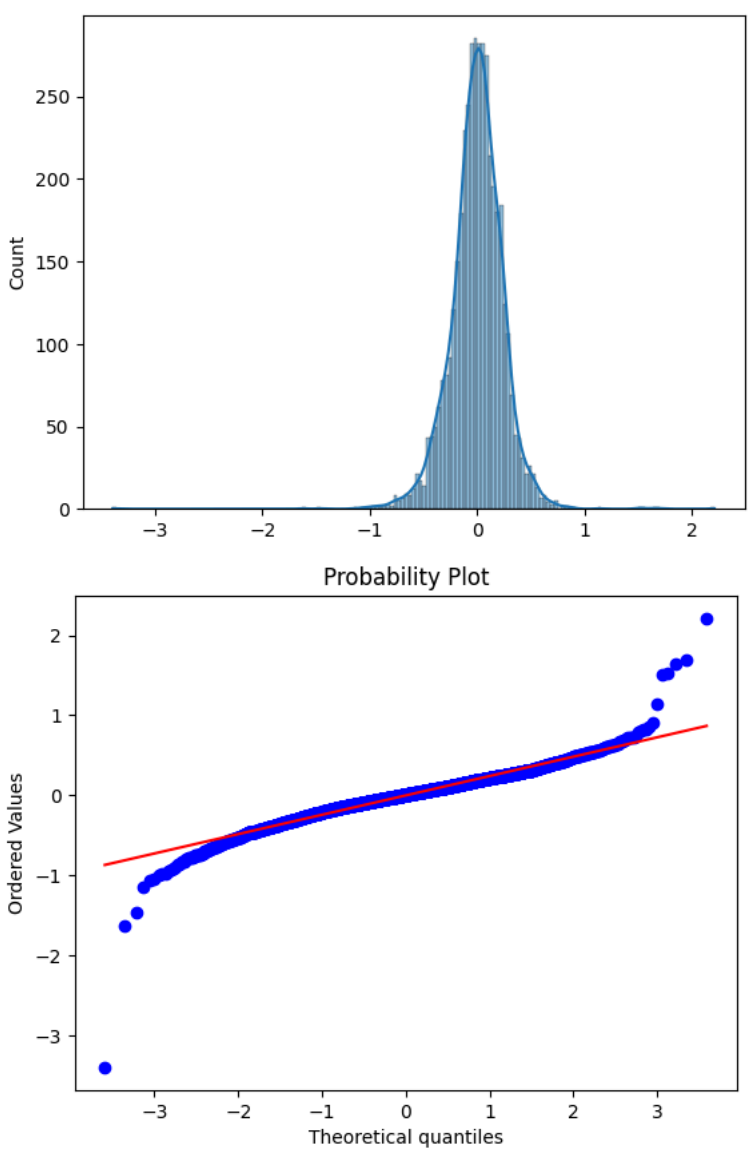
```
# 2. Goldfeld-Quandt test for heteroskedasticity
from statsmodels.stats.diagnostic import het_white
from statsmodels.compat import lzip
import statsmodels.stats.api as sms
import statsmodels.stats.api as sms

s = ["F statistic", "p-value"]
gfq = sms.het_goldfeldquandt(y2_train['price_log'], X2_train_scaled4)
lzip(s, gfq)
```

```
[('F statistic', 1.0702803215586685), ('p-value', 0.060954674879136056)]
```

```
# Linearity of variables, i.e. residuals should be randomly distributed
# Predicted values
x = ols_model2.fittedvalues
# Residual plot
sns.residplot(x = x, y = resa)
plt.show()
```




```
# Check for normality of error terms
# Residuals histogram
sns.histplot(resa, kde = True)
plt.show()

# q-q plot of residuals to check for normality (plot must follow linear path)
import pylab
import scipy.stats as stats

stats.probplot(resa, dist = "norm", plot = pylab)
plt.show()
```



Observations

- We fail to reject the Null Hypothesis for the Goldfeld-Quandt test, so we cannot say with certainty that this model satisfies the linear regression assumptions
- We won't consider this model further

Ridge / Lasso Regression

We have found that many of the independent variables have a moderately significant to very significant effect on the dependent variable. Therefore, we will use Ridge Regression instead of Lasso

```
# import Ridge model library
from sklearn.linear_model import Ridge
# Build model
ridgemod1 = Ridge()
# Fit Ridge model
ridgemod1.fit(X_train, y_train['price_log'])
# Check performance metrics
ridgeper1=performance(ridgemod1,ls=True)
ridgeper1
```

```
{'r^2 train': 0.890310819649026,
 'r^2 test': 0.8918923560258172,
 'RMSE train': 3.7002678627264007,
 'RMSE test': 3.6644401983859543,
 'MAE train': 1.6241861276709826,
 'MAE test': 1.675601342556319,
 'MAPE train': 0.1809957108620235,
 'MAPE test': 0.1725307154752447}
```

```
# Build model
ridgemod2 = Ridge()
# Fit Ridge model
ridgemod2.fit(X2_train, y2_train['price_log'])
# Check performance metrics
ridgeper2=performance(ridgemod2,X_train=X2_train,X_test=X2_test,ls=True)
ridgeper2
```

```
{'r^2 train': 0.8908774508821465,
 'r^2 test': 0.8850534414911536,
 'RMSE train': 3.6906980851367317,
 'RMSE test': 3.7785695792992633,
 'MAE train': 1.6330230767488614,
 'MAE test': 1.689801627591195,
 'MAPE train': 0.18096380770856033,
 'MAPE test': 0.17313542321814465}
```

Observations

- Ridge model has very good R^2 values. We will tune the hyperparameters later

Decision Tree

```
dectreemod1 = DecisionTreeRegressor(random_state=1)
dectreemod1.fit(X_train, y_train['price_log'])
treeper1=performance(dectreemod1,ls=True)
treeper1
```

```
{'r^2 train': 0.9999965696959587,
 'r^2 test': 0.8129416504880371,
 'RMSE train': 0.020692719736775493,
 'RMSE test': 4.82023249704715,
 'MAE train': 0.0009472934472943435,
 'MAE test': 2.0157100118069042,
 'MAPE train': 0.00014019474974059603,
 'MAPE test': 0.21265430192026272}
```

```
dectreemod2 = DecisionTreeRegressor(random_state=1)
dectreemod2.fit(X2_train, y2_train['price_log'])
treeper2=performance(dectreemod2,X_train=X2_train,X_test=X2_test,ls=True)
treeper2
```

```
{'r^2 train': 0.9999965696959587,
 'r^2 test': 0.8038698006346671,
 'RMSE train': 0.020692719736775493,
 'RMSE test': 4.935733156947941,
 'MAE train': 0.0009472934472943435,
 'MAE test': 2.029588771321101,
 'MAPE train': 0.00014019474974059603,
 'MAPE test': 0.21403633468206928}
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
ranformod1 = RandomForestRegressor()
ranformod1.fit(X_train, y_train['price_log'])
forper1=performance(ranformod1,ls=True)
forper1
```

```
{'r^2 train': 0.9761922018191125,
 'r^2 test': 0.8607022125289363,
 'RMSE train': 1.7238952951147808,
 'RMSE test': 4.159599779126469,
 'MAE train': 0.581379650672455,
 'MAE test': 1.5134844708537847,
 'MAPE train': 0.05675166522715659,
 'MAPE test': 0.14771052416373043}
```

```
# Model when removing newprice_log column
ranformod2 = RandomForestRegressor()
ranformod2.fit(X2_train, y2_train['price_log'])
forper2=performance(ranformod2,X_train=X2_train,X_test=X2_test,ls=True)
forper2
```

```
{'r^2 train': 0.9741352036783257,
 'r^2 test': 0.8700003219714543,
 'RMSE train': 1.7968250749604775,
 'RMSE test': 4.0183760399285715,
 'MAE train': 0.5848569283956446,
```

```
 'MAE test': 1.5377098146064179,
 'MAPE train': 0.056711935387157075,
 'MAPE test': 0.14990501889896418}
```

Hyperparameter Tuning

First, let's look and compare the various models we have built. For Ridge Regression, Decision Trees, and Random Forest, we will pick the best alternative according to our performance metrics and tune them individually.

```
# Hyperparameter tuning: Import Grid Search
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, f1_score
from sklearn import metrics
from sklearn import tree
```

```
# Create function to be used later to show information about tuned model
def tunper(model1,model2):
  return print('Optimised Negative RMSE:\n',model1.best_score_,'\n','-'*100,'\n','Optimised regression parameters:\n',model1.best_params
```

```
# Print DataFrame with the performance of each model so far (Note: since the OLS2 model doesn't satisfy the linear regression assumptior
perframe=pd.DataFrame([linper1,linper2,olsper1,ridgeper1,ridgeper2,treeper1,treeper2,forper1,forper2],index=['Linear 1','Linear 2','OLS
perframe
```

	r^2 train	r^2 test	RMSE train	RMSE test	MAE train	MAE test	MAPE train	MAPE test
Linear 1	0.895901	0.891820	3.604739	3.665666	1.613169	1.679332	0.180956	0.172032
Linear 2	0.896389	0.885570	3.596278	3.770073	1.622493	1.693354	0.180931	0.172628
OLS	0.867759	0.808904	4.062889	4.871973	1.780090	2.314411	0.192296	0.241807
Ridge 1	0.890311	0.891892	3.700268	3.664440	1.624186	1.675601	0.180996	0.172531
Ridge 2	0.890877	0.885053	3.690698	3.778570	1.633023	1.689802	0.180964	0.173135
Decision Tree 1	0.999997	0.812942	0.020693	4.820232	0.000947	2.015710	0.000140	0.212654
Decision Tree 2	0.999997	0.803870	0.020693	4.935733	0.000947	2.029589	0.000140	0.214036
Random Forest 1	0.976192	0.860702	1.723895	4.159600	0.581380	1.513484	0.056752	0.147711
Random Forest 2	0.974135	0.870000	1.796825	4.018376	0.584857	1.537710	0.056712	0.149905

Decision Tree

Immediately, we see that both Decision Tree models are very ovefit to the training data; they perform much better on the trainig set than the test datase across all performance metrics. We will hopefully decrease this overfitting after the hyperparameter tuning. Again, there is little difference between the model with the newprice_log feature, and the one without. However, it seems Decision Tree 1 performs slightly better

```
# Check parameters for DecisionTreeRegressor from sklearn website
["criterion='squared_error'", "splitter='best'", 'max_depth=None', 'min_samples_split=2', 'min_samples_leaf=1', 'min_weight_fraction_lea
```

```
["criterion='squared_error'",
 "splitter='best'",
 'max_depth=None',
 'min_samples_split=2',
 'min_samples_leaf=1',
 'min_weight_fraction_leaf=0.0',
 'max_features=None',
 'random_state=None',
 'max_leaf_nodes=None',
 'min_impurity_decrease=0.0',
 'ccp_alpha=0.0']
```

Interesting parameters to look at:

- max_depth => See if there's a sweet spot where we minimise information loss, and also minimise overfitting
- min_samples_leaf => Restrict the number of endpoints in our tree - offset max_depth cutting our tree too soon

```
# Look at feature importances pre-tuning to gather a better idea of what values to choose for our parameters
plt.figure(figsize=(10,16))
plt.title('Decision Tree feature importance')
sns.barplot(pd.DataFrame(data=dectreemod1.feature_importances_,index = X_train.columns,columns=['Importance']).sort_values(by='Importanc
plt.show()
pd.DataFrame(data=dectreemod1.feature_importances_,index = X_train.columns,columns=['Importance']).sort_values(by='Importance',ascending
```

	power_log	Year	Engine	km_log	Mileage	Brand_Land Rover	Brand_Tata	newprice_log	Transmission_Manual	Location_K
Importance	0.605544	0.230474	0.05603	0.016636	0.012824	0.005818	0.005439	0.005219	0.004612	0.0

1 rows × 54 columns

```
# Check the price column summary
y_train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
price_log	4212.0	1.815175	0.871181	-0.820981	1.252763	1.713798	2.274956	4.787492
Price	4212.0	9.386500	11.173850	0.440000	3.500000	5.550000	9.727500	120.000000

Observations

- We see that power_log is the most important feature by a large margin. The top three features have at least a 5% importance in our model. The categorical variables do not seem particularly important, even if we sum up each subfeature for each variable. We will try with depths of 3-10 and full
- There are 4212 records in the training dataset, so we can comfortably choose a wide variety of values for min_samples_leaf

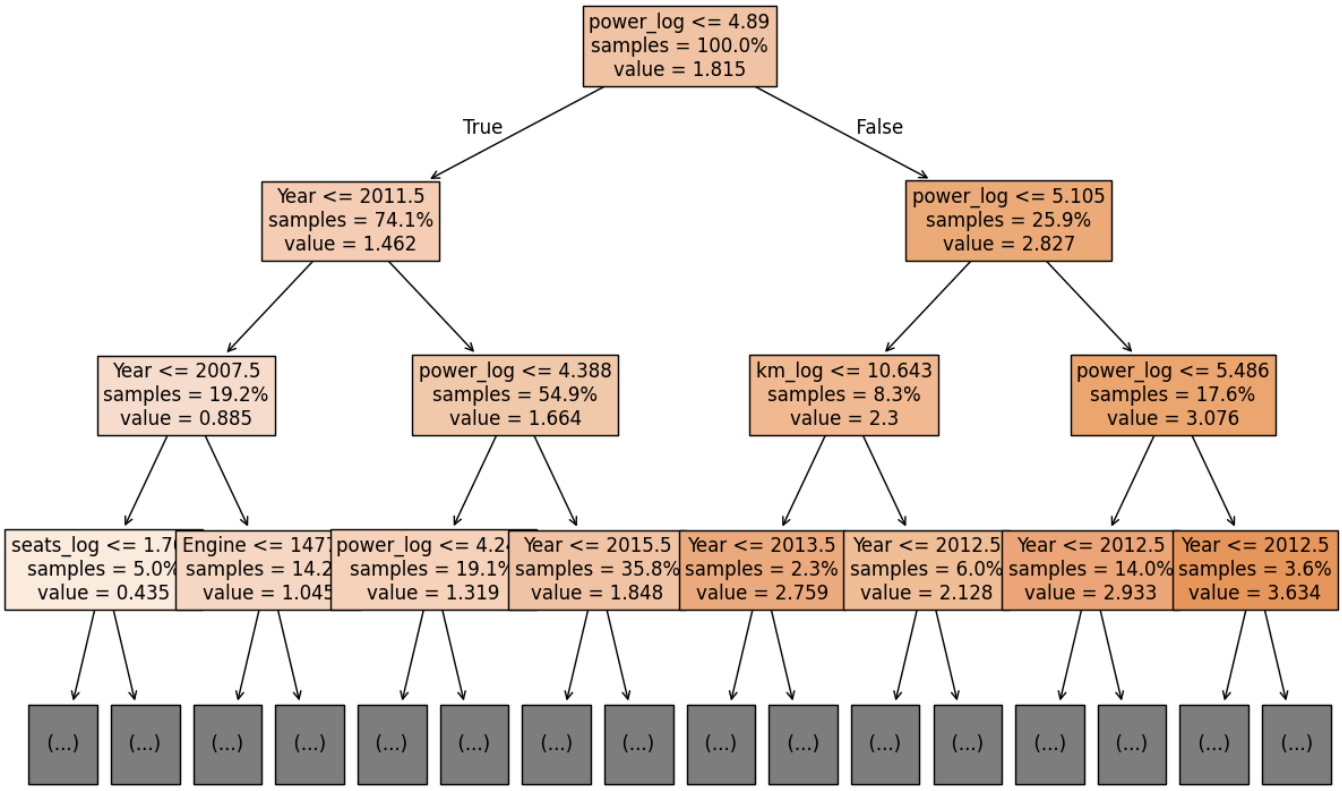
```
# Model to be tuned
dectree_tuned=dectreemod1
# Define the parameters for Grid to choose from
parameters = {'max_depth' : np.arange(3,20),
              'min_samples_leaf' : np.arange(1,201,10),
              'max_features':[0.3,0.5,0.8,1],
              'min_samples_split':np.arange(2,30,7)
              }
# Grid Search
gscv = GridSearchCV(dectree_tuned, parameters, scoring = 'neg_root_mean_squared_error', cv = 10)
gscv = gscv.fit(X_train, y_train['price_log'])
# Set the model to the best combination of parameters
dectree_tuned=gscv.best_estimator_
# Fit the best algorithm to the data
dectree_tuned.fit(X_train,y_train['price_log'])
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=14, max_features=0.8, min_samples_split=23,
                      random_state=1)
```

```
# Check info and performance
tunper(gscv,dectree_tuned)
```

```
Optimised Negative RMSE:
-0.27918361131229347
-----------------------------------------------------------------------------------------------
Optimised regression parameters:
{'max_depth': 14, 'max_features': 0.8, 'min_samples_leaf': 1, 'min_samples_split': 23}
-----------------------------------------------------------------------------------------------
Extra performance metrics:
{'r^2 train': 0.9212844725706231, 'r^2 test': 0.7712592233398539, 'RMSE train': 3.1345952639409846, 'RMSE test': 5.330294848614565,
```

```
plt.figure(figsize = (15, 10))

tree.plot_tree(dectree_tuned, max_depth = 3, feature_names = list(X_train.columns), filled = True, fontsize = 12, node_ids = False,impur

plt.show()
```



Observations Still have significant overfitting, but a shows a good RSME score

Random Forest

Like the Decision Trees, the random forest models are very overfit. We will use the Random Forest 1 model.

```
# Check parameters for RandomForestRegressor from sklearn website
['n_estimators=100', "criterion='squared_error'", 'max_depth=None', 'min_samples_split=2', 'min_samples_leaf=1', 'min_weight_fraction_le
```

```
['n_estimators=100',
 "criterion='squared_error'",
 'max_depth=None',
 'min_samples_split=2',
 'min_samples_leaf=1',
 'min_weight_fraction_leaf=0.0',
 'max_features=1.0',
 'max_leaf_nodes=None',
 'min_impurity_decrease=0.0',
 'bootstrap=True',
 'oob_score=False',
 'n_jobs=None',
 'random_state=None',
 'verbose=0',
 'warm_start=False',
 'ccp_alpha=0.0',
 'max_samples=None']
```

```
# Check feature importance pre-tuning
plt.figure(figsize=(10,16))
plt.title('Random Forest feature importance')
sns.barplot(pd.DataFrame(data=ranformod1.feature_importances_,index = X_train.columns,columns=['Importance']).sort_values(by='Importanc
plt.show()
pd.DataFrame(data=ranformod1.feature_importances_,index = X_train.columns,columns=['Importance']).sort_values(by='Importance',ascending=
```

	power_log	Year	Engine	km_log	Mileage	Brand_Land Rover	newprice_log	Brand_Tata	Location_Kolkata	Transmission_
Importance	0.608342	0.233298	0.043345	0.018694	0.01602	0.006795	0.00602	0.005297	0.005218	0.

1 rows × 54 columns

```
# Model to tune
ranfor_tuned = ranformod1

# Parameters
parameters = {"n_estimators": [100, 125, 150],
```

```
    "max_depth": np.arange(3,7,15),
    "min_samples_leaf": [1,15,30],
    "max_features": [0.7, 1.0],}

# Run the grid search
gscv2 = GridSearchCV(ranfor_tuned, parameters, scoring = 'neg_root_mean_squared_error', cv = 10)

gscv2 = gscv2.fit(X_train, y_train['price_log'])

# Set the classifier to the best combination of parameters
ranfor_tuned = gscv2.best_estimator_
# Fit best estimator
ranfor_tuned.fit(X_train,y_train['price_log'])
```

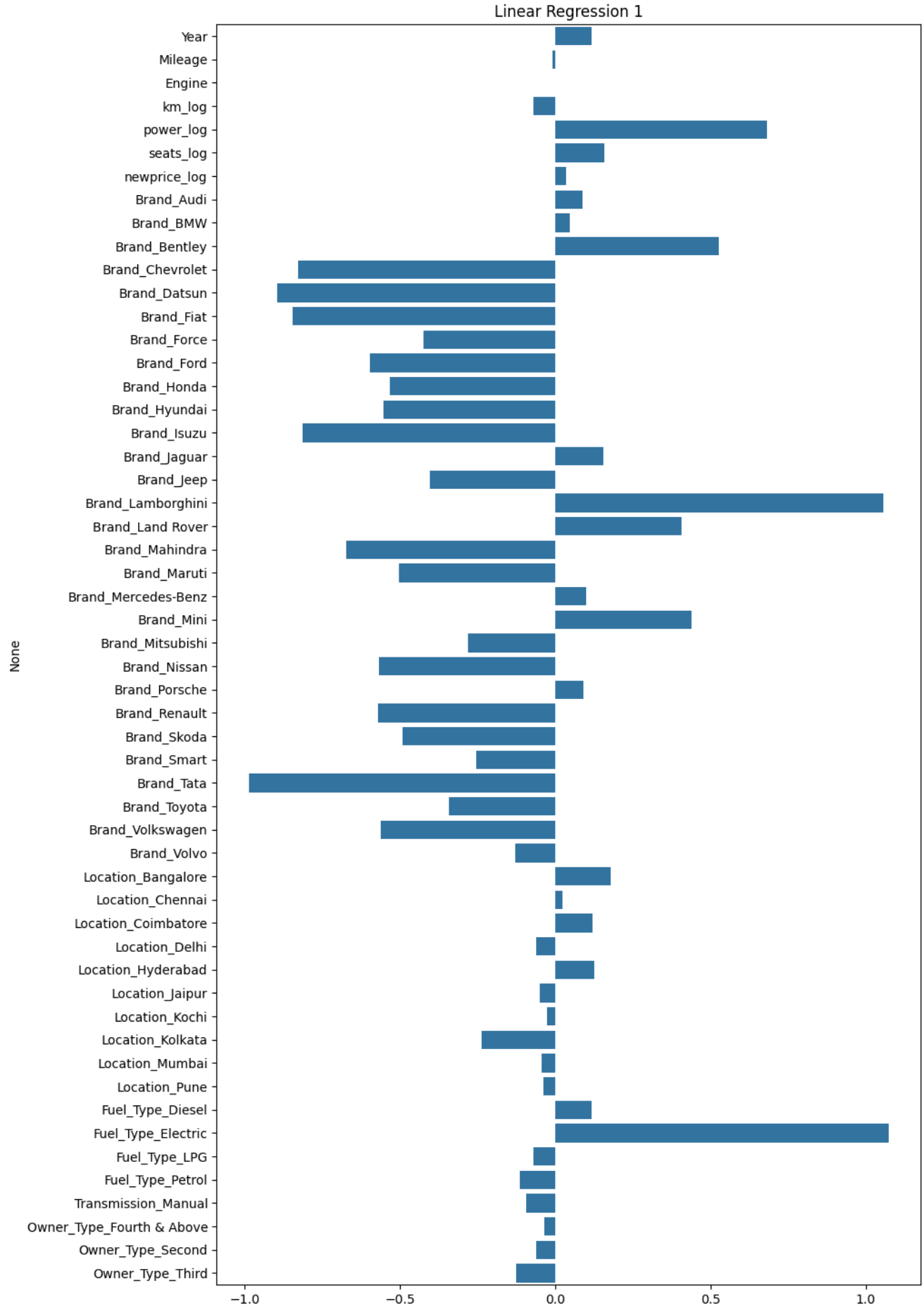
```
RandomForestRegressor(max_depth=3, max_features=0.7, min_samples_leaf=30,
                      n_estimators=150)
```

```
tunper(gscv2,ranfor_tuned)
```

```
Optimised Negative RMSE:
-0.39572208319293384
----------------------------------------------------------------------------------------------------
Optimised regression parameters:
{'max_depth': 3, 'max_features': 0.7, 'min_samples_leaf': 30, 'n_estimators': 150}
----------------------------------------------------------------------------------------------------
Extra performance metrics:
{'r^2 train': 0.6726944643423941, 'r^2 test': 0.6751486641258999, 'RMSE train': 6.3918701455270766, 'RMSE test': 6.352164930713315,
```

Feature Importance

```
# Create function to store the feature importance values for the decision tree and random forest models
def featimp(model,X_train=X_train):
  impor=pd.DataFrame(data=model.feature_importances_,index = X_train.columns,columns=['Importance']).sort_values(by='Importance',ascend
  return impor


# Feature importances
for i,j in zip([lreg,ridgemod1],['Linear Regression 1','Ridge Regression 1']):
  plt.figure(figsize=(10,18))
  sns.barplot(x=i.coef_,y=X_train.columns,orient='h')
  plt.title(j)
  plt.show()
for i,j in zip([lreg2,ridgemod2],['Linear Regression 2','Ridge Regression 2']):
  plt.figure(figsize=(10,18))
  sns.barplot(x=i.coef_,y=X2_train.columns,orient='h')
  plt.title(j)
  plt.show()
```


Linear Regression 1
Year
Mileage
Engine
km_log
power_log
seats_log
newprice_log
Brand_Audi
Brand_BMW
Brand_Bentley
Brand_Chevrolet
Brand_Datsun
Brand_Fiat
Brand_Force
Brand_Ford
Brand_Honda
Brand_Hyundai
Brand_Isuzu
Brand_Jaguar
Brand_Jeep
Brand_Lamborghini
Brand_Land Rover
Brand_Mahindra
Brand_Maruti
Brand_Mercedes-Benz
Brand_Mini
Brand_Mitsubishi
Brand_Nissan
Brand_Porsche
Brand_Renault
Brand_Skoda
Brand_Smart
Brand_Tata
Brand_Toyota
Brand_Volkswagen
Brand_Volvo
Location_Bangalore
Location_Chennai
Location_Coimbatore
Location_Delhi
Location_Hyderabad
Location_Jaipur
Location_Kochi
Location_Kolkata
Location_Mumbai
Location_Pune
Fuel_Type_Diesel
Fuel_Type_Electric
Fuel_Type_LPG
Fuel_Type_Petrol
Transmission_Manual
Owner_Type_Fourth & Above
Owner_Type_Second
Owner_Type_Third
None
−1.0
−0.5
0.0
0.5
1.0
Ridge Regression 1
Year
Mileage
Engine
km_log
power_log