Background & Context

There is a huge demand for used cars in the Indian Market today. As sales of new cars have slowed down in the recent past, the pre-owned car market has continued to grow over the past years and is larger than the new car market now. Cars4U is a budding tech start-up that aims to find footholes in this market.

In 2018-19, while new car sales were recorded at 3.6 million units, around 4 million second-hand cars were bought and sold. There is a slowdown in new car sales and that could mean that the demand is shifting towards the pre-owned market. In fact, some car sellers replace their old cars with pre-owned cars instead of buying new ones. Unlike new cars, where price and supply are fairly deterministic and managed by OEMs (Original Equipment Manufacturer / except for dealership level discounts which come into play only in the last stage of the customer journey), used cars are very different beasts with huge uncertainty in both pricing and supply. Keeping this in mind, the pricing scheme of these used cars becomes important in order to grow in the market.

As a senior data scientist at Cars4U, you have to come up with a pricing model that can effectively predict the price of used cars and can help the business in devising profitable strategies using differential pricing. For example, if the business knows the market price, it will never sell anything below it.

Objective

- Explore and visualize the dataset.
- Build a linear regression model to predict the prices of used cars.
- Generate a set of insights and recommendations that will help the business.

Data Dictionary -

S.No. : Serial Number

Name : Name of the car which includes Brand name and Model name

Location : The location in which the car is being sold or is available for purchase Cities

Year : Manufacturing year of the car

Kilometers_driven : The total kilometers driven in the car by the previous owner(s) in KM.

Fuel_Type : The type of fuel used by the car. (Petrol, Diesel, Electric, CNG, LPG)

Transmission : The type of transmission used by the car. (Automatic / Manual)

Owner : Type of ownership

Mileage : The standard mileage offered by the car company in kmpl or km/kg

Engine : The displacement volume of the engine in CC.

Power : The maximum power of the engine in bhp.

Seats : The number of seats in the car.

New_Price : The price of a new car of the same model in INR Lakhs.(1 Lakh = 100, 000)

Price : The price of the used car in INR Lakhs (1 Lakh = 100, 000)

Loading libraries

In [1]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit from the number of displayed rows.
pd.set_option("display.max_columns", None)
# Changes the limit of number of displayed columns tov200
pd.set_option("display.max_rows", 200)

# To build linear model for prediction
from sklearn.linear_model import LinearRegression
```

```
# To check model performance
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

Loading and exploring the data

Loading the data into python to explore and understand it.

In [2]:
```
df = pd.read_csv("used_cars_data.csv")
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")  # f-string

np.random.seed(1)  # To get the same random results every time
df.sample(n=10)
```

```
There are 7253 rows and 14 columns.
```

Out[2]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Pric
2397	2397	Ford EcoSport 1.5 Petrol Trend	Kolkata	2016	21460	Petrol	Manual	First	17.0 kmpl	1497 CC	121.36 bhp	5.0	9.47 Lak
3777	3777	Maruti Wagon R VXI 1.2	Kochi	2015	49818	Petrol	Manual	First	21.5 kmpl	1197 CC	81.80 bhp	5.0	5.44 Lak
4425	4425	Ford Endeavour 4x2 XLT	Hyderabad	2007	130000	Diesel	Manual	First	13.1 kmpl	2499 CC	141 bhp	7.0	Nal
3661	3661	Mercedes-Benz E-Class E250 CDI Avantgrade	Coimbatore	2016	39753	Diesel	Automatic	First	13.0 kmpl	2143 CC	201.1 bhp	5.0	Nal
4514	4514	Hyundai Xcent 1.2 Kappa AT SX Option	Kochi	2016	45560	Petrol	Automatic	First	16.9 kmpl	1197 CC	82 bhp	5.0	Nal
599	599	Toyota Innova Crysta 2.8 ZX AT	Coimbatore	2019	40674	Diesel	Automatic	First	11.36 kmpl	2755 CC	171.5 bhp	7.0	28.05 Lak
186	186	Mercedes-Benz E-Class E250 CDI Avantgrade	Bangalore	2014	37382	Diesel	Automatic	First	13.0 kmpl	2143 CC	201.1 bhp	5.0	Nal
305	305	Audi A6 2011-2015 2.0 TDI Premium Plus	Kochi	2014	61726	Diesel	Automatic	First	17.68 kmpl	1968 CC	174.33 bhp	5.0	Nal
4582	4582	Hyundai i20 1.2 Magna	Kolkata	2011	36000	Petrol	Manual	First	18.5 kmpl	1197 CC	80 bhp	5.0	Nal
5434	5434	Honda WR-V Edge Edition i-VTEC S	Kochi	2019	13913	Petrol	Manual	First	17.5 kmpl	1199 CC	88.7 bhp	5.0	9.36 Lak

`S.No.` is just an index for the data entry. In all likelihood, this column will not be a significant factor in determining the price of the car. Having said that, there are instances where the index of the data entry contains the information about time factor (an entry with a smaller index corresponds to data entered years ago). Therefore, we will not drop this variable just yet. Let us see if there is any relationship with the price when we do bivariate analysis.

`Car names` contain a lot of model information. Let us check how many individual names we have. If they are too many, we can process this column to extract important information.

`Mileage`, `Engine` and `Power` will also need some processing before we are able to explore them. We'll have to extract numerical information from these columns.

`New Price` column also needs some processing. This one also contains strings and a lot of missing values.

In [3]:
```
df.info()
```

```
<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   object
 9   Engine             7207 non-null   object
 10  Power              7207 non-null   object
 11  Seats              7200 non-null   float64
 12  New_Price          1006 non-null   object
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 793.4+ KB
```

As expected, `Mileage`, `Engine`, `Power` and `New_Price` are objects when they should ideally be numerical. To be able to get summary statistics for these columns, We will have to process them first.

Processing Columns

Let us process 'Mileage', 'Engine', 'Power' and 'New_Price' and extract numerical values from them.

1. Mileage

We have car mileage in two units, kmpl and km/kg.

After a quick research on the internet it is clear that these 2 units are used for cars of 2 different fuel types.

kmpl - kilometers per litre - is used for petrol and diesel cars. km/kg - kilometers per kg - is used for CNG and LPG based engines.

We have the variable `Fuel_type` in our data. Let us check if this observations holds true in our data also.

In [4]:
```
# Create 2 new columns after splitting the mileage values.
km_per_unit_fuel = []
mileage_unit = []

for observation in df["Mileage"]:
    if isinstance(observation, str):
        if (
            observation.split(" ")[0]
            .replace(".", "")
            .isdigit()  # first element should be numeric
            and " " in observation  # space between numeric and unit
            and (
                observation.split(" ")[1]
                == "kmpl"  # units are limited to "kmpl" and "km/kg"
                or observation.split(" ")[1] == "km/kg"
            )
        ):
            km_per_unit_fuel.append(float(observation.split(" ")[0]))
            mileage_unit.append(observation.split(" ")[1])
        else:
            # To detect if there are any observations in the column that do not follow
            # the expected format [number + ' ' + 'kmpl' or 'km/kg']
            print(
                "The data needs further processing. All values are not similar ",
                observation,
            )
    else:
        # If there are any missing values in the mileage column,
        # we add corresponding missing values to the 2 new columns
        km_per_unit_fuel.append(np.nan)
        mileage_unit.append(np.nan)
```

In [5]:
```
# No print output from the function above. The values are all in the expected format or NaNs
# Add the new columns to the data

df["km_per_unit_fuel"] = km_per_unit_fuel
df["mileage_unit"] = mileage_unit
```

```
# Checking the new dataframe
df.head(5)  # looks good!
```

Out[5]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	P
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17

In [6]:

```
# Let us check if the units correspond to the fuel types as expected.
df.groupby(by=["Fuel_Type", "mileage_unit"]).size()
```

Out[6]:

```
Fuel_Type  mileage_unit
CNG        km/kg              62
Diesel     kmpl             3852
LPG        km/kg              12
Petrol     kmpl             3325
dtype: int64
```

As expected, km/kg is for CNG/LPG cars and kmpl is for Petrol and Diesel cars.

2. Engine

The data dictionary suggests that `Engine` indicates the displacement volume of the engine in CC. We will make sure that all the observations follow the same format - [numeric + " " + "CC"] and create a new numeric column from this column.

This time, lets use a regex to make all the neccesary checks.

In [7]:

```
# re module provides support for regular expressions
import re

# Create a new column after splitting the engine values.
engine_num = []

# Regex for numeric + " " + "CC"  format
regex_engine = "^\d+(\.\d+)? CC$"

for observation in df["Engine"]:
    if isinstance(observation, str):
        if re.match(regex_engine, observation):
            engine_num.append(float(observation.split(" ")[0]))
        else:
            # To detect if there are any observations in the column that do not follow [numeric + " " + "CC"]  fo
            print(
                "The data needs furthur processing. All values are not similar ",
                observation,
            )
    else:
        # If there are any missing values in the engine column, we add missing values to the new column
        engine_num.append(np.nan)
```

In [8]:

```
# No print output from the function above. The values are all in the same format - [numeric + " " + "CC"] OR NaNs
# Add the new column to the data

df["engine_num"] = engine_num

# Checking the new dataframe
df.head(5)
```

Out[8]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	P
0	0	Maruti Wagon R	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	

		LXI CNG												
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	1
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	1

3. Power

The data dictionary suggests that `Power` indicates the maximum power of the engine in bhp. We will make sure that all the observations follow the same format - [numeric + " " + "bhp"] and create a new numeric column from this column, like we did for `Engine`

In [9]:

```
# Create a new column after splitting the power values.
power_num = []

# Regex for numeric + " " + "bhp"  format
regex_power = "^\d+(\.\d+)? bhp$"

for observation in df["Power"]:
    if isinstance(observation, str):
        if re.match(regex_power, observation):
            power_num.append(float(observation.split(" ")[0]))
        else:
            # To detect if there are any observations in the column that do not follow [numeric + " " + "bhp"]  f
            # that we see in the sample output
            print(
                "The data needs furthur processing. All values are not similar ",
                observation,
            )
    else:
        # If there are any missing values in the power column, we add missing values to the new column
        power_num.append(np.nan)
```

```
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
```

The data needs furthur processing. All values are not similar null bhp

```
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
The data needs furthur processing. All values are not similar  null bhp
```

We can see that some Null values in power column exist as 'null bhp' string. Let us replace these with NaNs

In [10]:
```
power_num = []

for observation in df["Power"]:
    if isinstance(observation, str):
        if re.match(regex_power, observation):
            power_num.append(float(observation.split(" ")[0]))
        else:
            power_num.append(np.nan)
    else:
        # If there are any missing values in the power column, we add missing values to the new column
        power_num.append(np.nan)

# Add the new column to the data
df["power_num"] = power_num

# Checking the new dataframe
df.head(10)  # looks good now
```

Out[10]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN
5	5	Hyundai EON LPG Era Plus Option	Hyderabad	2012	75000	LPG	Manual	First	21.1 km/kg	814 CC	55.2 bhp	5.0	NaN
6	6	Nissan Micra Diesel XV	Jaipur	2013	86999	Diesel	Manual	First	23.08 kmpl	1461 CC	63.1 bhp	5.0	NaN
7	7	Toyota Innova Crysta 2.8 GX AT 8S	Mumbai	2016	36000	Diesel	Automatic	First	11.36 kmpl	2755 CC	171.5 bhp	8.0	21 Lakh
8	8	Volkswagen Vento Diesel Comfortline	Pune	2013	64430	Diesel	Manual	First	20.54 kmpl	1598 CC	103.6 bhp	5.0	NaN
9	9	Tata Indica Vista Quadrajet LS	Chennai	2012	65932	Diesel	Manual	Second	22.3 kmpl	1248 CC	74 bhp	5.0	NaN

4. New_Price

We know that `New_Price` is the price of a new car of the same model in INR Lakhs.(1 Lakh = 100, 000)

This column clearly has a lot of missing values. We will impute the missing values later. For now we will only extract the numeric values from this column.

In [11]:
```
# Create a new column after splitting the New_Price values.
new_price_num = []
```

```
# Regex for numeric + " " + "Lakh"  format
regex_power = "^\d+(\.\d+)? Lakh$"

for observation in df["New_Price"]:
    if isinstance(observation, str):
        if re.match(regex_power, observation):
            new_price_num.append(float(observation.split(" ")[0]))
        else:
            # To detect if there are any observations in the column that do not follow [numeric + " " + "Lakh"]
            # that we see in the sample output
            print(
                "The data needs furthur processing. All values are not similar ",
                observation,
            )
    else:
        # If there are any missing values in the New_Price column, we add missing values to the new column
        new_price_num.append(np.nan)
```

```
The data needs furthur processing. All values are not similar  1.28 Cr
The data needs furthur processing. All values are not similar  1.04 Cr
The data needs furthur processing. All values are not similar  1 Cr
The data needs furthur processing. All values are not similar  1.04 Cr
The data needs furthur processing. All values are not similar  1.39 Cr
The data needs furthur processing. All values are not similar  1.02 Cr
The data needs furthur processing. All values are not similar  1.4 Cr
The data needs furthur processing. All values are not similar  1.06 Cr
The data needs furthur processing. All values are not similar  1.27 Cr
The data needs furthur processing. All values are not similar  1.13 Cr
The data needs furthur processing. All values are not similar  1.36 Cr
The data needs furthur processing. All values are not similar  1.66 Cr
The data needs furthur processing. All values are not similar  1.6 Cr
The data needs furthur processing. All values are not similar  1.28 Cr
The data needs furthur processing. All values are not similar  2.3 Cr
The data needs furthur processing. All values are not similar  1.71 Cr
The data needs furthur processing. All values are not similar  1.39 Cr
The data needs furthur processing. All values are not similar  1.58 Cr
The data needs furthur processing. All values are not similar  3.75 Cr
The data needs furthur processing. All values are not similar  1.06 Cr
```

Not all values are in Lakhs. There are a few observations that are in Crores as well

Let us convert these to lakhs. 1 Cr = 100 Lakhs

In [12]:

```
new_price_num = []

for observation in df["New_Price"]:
    if isinstance(observation, str):
        if re.match(regex_power, observation):
            new_price_num.append(float(observation.split(" ")[0]))
        else:
            # Converting values in Crore to lakhs
            new_price_num.append(float(observation.split(" ")[0]) * 100)
    else:
        # If there are any missing values in the New_Price column, we add missing values to the new column
        new_price_num.append(np.nan)

# Add the new column to the data
df["new_price_num"] = new_price_num

# Checking the new dataframe
df.head(5)  # Looks ok
```

Out[12]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	P
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17

Feature Engineering

The `Name` column in the current format might not be very useful in our analysis. Since the name contains both the brand name and the model name of the vehicle, the column would have to have many unique values to be useful in prediction.

In [13]:
```
df["Name"].nunique()
```

Out[13]:
```
2041
```

With 2041 unique names, car names are not going to be great predictors of the price in our current data.

But we can process this column to extract important information and see if that reduces the number of levels for this information.

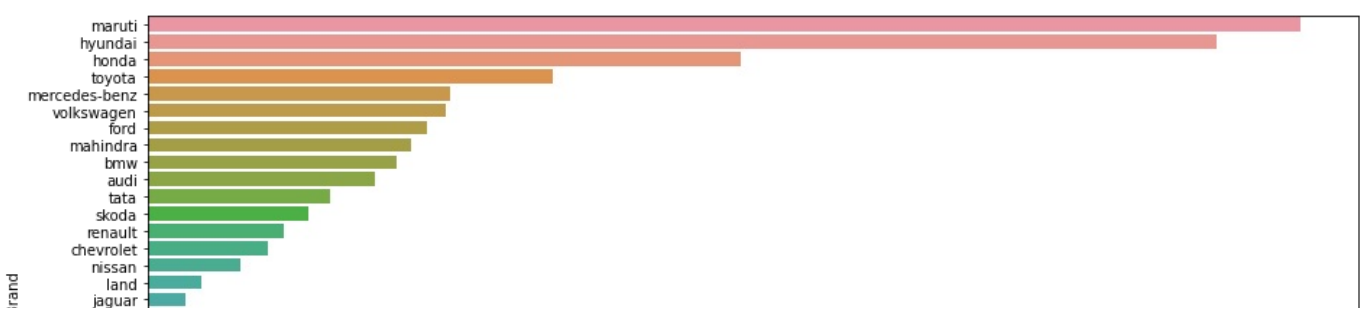
1. Car Brand Name

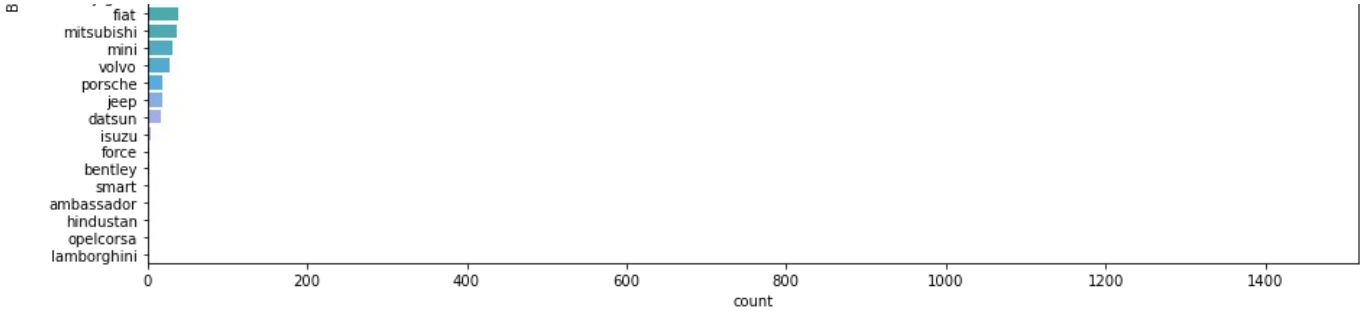
In [14]:
```
# Extract Brand Names
df["Brand"] = df["Name"].apply(lambda x: x.split(" ")[0].lower())

# Check the data
df["Brand"].value_counts()
```

Out[14]:
```
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               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
hindustan           1
opelcorsa           1
lamborghini         1
Name: Brand, dtype: int64
```

In [15]:
```
plt.figure(figsize=(15, 7))
sns.countplot(y="Brand", data=df, order=df["Brand"].value_counts().index)
```

Out[15]:
```
<AxesSubplot:xlabel='count', ylabel='Brand'>
```





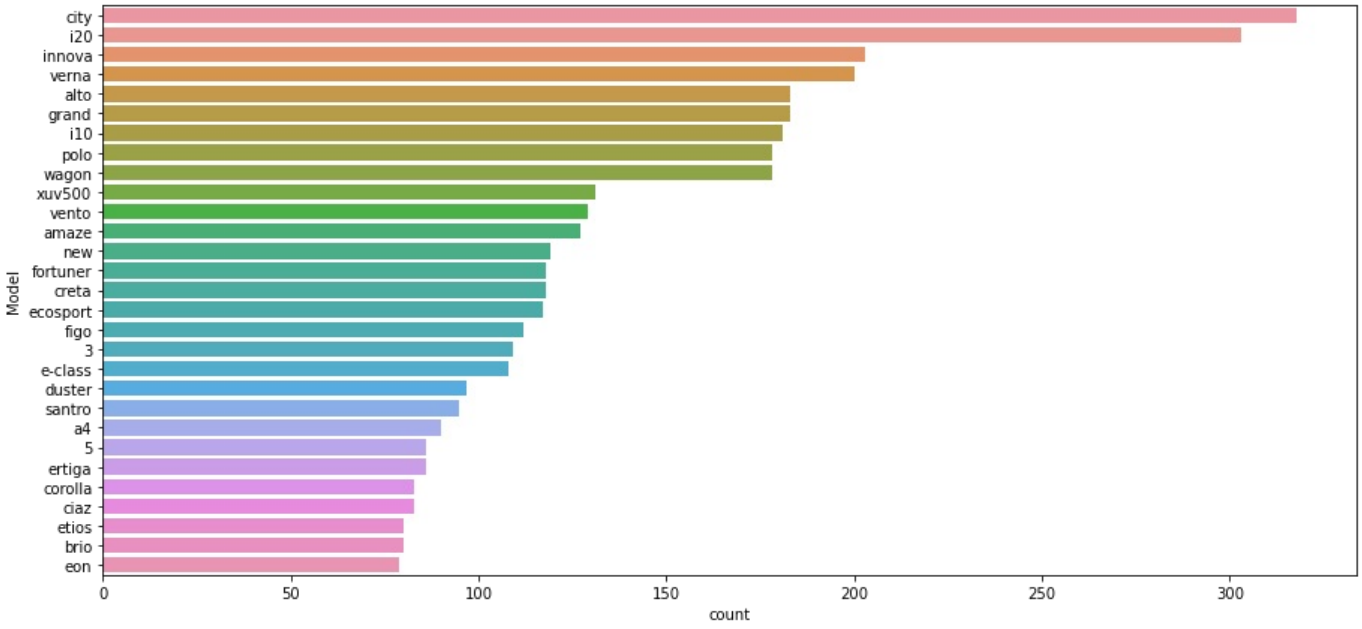
2. Car Model Name

In [16]:
```
# Extract Model Names
df["Model"] = df["Name"].apply(lambda x: x.split(" ")[1].lower())

# Check the data
df["Model"].value_counts()
```

Out[16]:
```
swift      418
city       318
i20        303
innova     203
verna      200
          ...
flying       1
beetle       1
prius        1
montero      1
boxster      1
Name: Model, Length: 218, dtype: int64
```

In [17]:
```
plt.figure(figsize=(15, 7))
sns.countplot(y="Model", data=df, order=df["Model"].value_counts().index[1:30])
```

Out[17]:
```
<AxesSubplot:xlabel='count', ylabel='Model'>
```



It is clear from the above charts that out dataset contains used cars from luxury as well as budget friendly brands.

We can create a new variable using this information. We will bin all our cars in 3 categories -

1. Budget Friendly
2. Mid Range
3. Luxury Cars

3. Car_category

In [18]:
```
df.groupby(["Brand"])["Price"].mean().sort_values(ascending=False)
```

Out[18]:
```
Brand
lamborghini      120.000000
bentley           59.000000
porsche           48.348333
land              39.259500
jaguar            37.632250
mini              26.896923
mercedes-benz     26.809874
audi              25.537712
bmw               25.243146
volvo             18.802857
jeep              18.718667
isuzu             14.696667
toyota            11.580024
mitsubishi        11.058889
force              9.333333
mahindra           8.045919
skoda              7.559075
ford               6.889400
renault            5.799034
honda              5.411743
hyundai            5.343433
volkswagen         5.307270
nissan             4.738352
maruti             4.517267
tata               3.562849
fiat               3.269286
datsun             3.049231
chevrolet          3.044463
smart              3.000000
ambassador         1.350000
hindustan               NaN
opelcorsa               NaN
Name: Price, dtype: float64
```

The output is very close to our expectation (domain knowledge), in terms of brand ordering. Mean price of a used Lamborghini is 120 Lakhs and that of cars from other luxury brands follow in a descending order.

Towards the bottom end we have the more budget friendly brands.

We can see that there is some missingness in our data. Let us come back to creating this variable once we have removed missingness from the data.

Exploratory Data Analysis

In [19]:
```
# Basic summary stats - Numeric variables
df.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	75%	max
S.No.	7253.0	3626.000000	2093.905084	0.00	1813.000	3626.00	5439.0000	7252.00
Year	7253.0	2013.365366	3.254421	1996.00	2011.000	2014.00	2016.0000	2019.00
Kilometers_Driven	7253.0	58699.063146	84427.720583	171.00	34000.000	53416.00	73000.0000	6500000.00
Seats	7200.0	5.279722	0.811660	0.00	5.000	5.00	5.0000	10.00
Price	6019.0	9.479468	11.187917	0.44	3.500	5.64	9.9500	160.00
km_per_unit_fuel	7251.0	18.141580	4.562197	0.00	15.170	18.16	21.1000	33.54
engine_num	7207.0	1616.573470	595.285137	72.00	1198.000	1493.00	1968.0000	5998.00
power_num	7078.0	112.765214	53.493553	34.20	75.000	94.00	138.1000	616.00
new_price_num	1006.0	22.779692	27.759344	3.91	7.885	11.57	26.0425	375.00

Observations

1. S.No. clearly has no interpretation here but as discussed earlier let us drop it only after having looked at the initial linear model.
2. Kilometers_Driven values have an incredibly high range. We should check a few of the extreme values to get a sense of the data.

3. Minimum and maximum number of seats in the car also warrent a quick check. On an average a car seems to have 5 seats, which is about right.
4. We have used cars being sold at less than a lakh rupees and as high as 160 lakh, as we saw for Lamborghini earlier. We might have to drop some of these outliers to build a robust model.
5. Min Mileage being 0 is also concerning, we'll have to check what is going on.
6. Engine and Power mean and median values are not very different. Only someone with more domain knowledge would be able to comment furthur on these attributes.
7. New price range seems right. We have both budget friendly Maruti cars and Lamborghinis in our stock. Mean being twice that of the median suggests that there are only a few very high range brands, which again makes sense.

In [20]:
```
# Check Kilometers_Driven extreme values
df.sort_values(by=["Kilometers_Driven"], ascending=False).head(10)
```

Out[20]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Pric
2328	2328	BMW X5 xDrive 30d M Sport	Chennai	2017	6500000	Diesel	Automatic	First	15.97 kmpl	2993 CC	258 bhp	5.0	Nal
340	340	Skoda Octavia Ambition Plus 2.0 TDI AT	Kolkata	2013	775000	Diesel	Automatic	First	19.3 kmpl	1968 CC	141 bhp	5.0	Nal
1860	1860	Volkswagen Vento Diesel Highline	Chennai	2013	720000	Diesel	Manual	First	20.54 kmpl	1598 CC	103.6 bhp	5.0	Nal
358	358	Hyundai i10 Magna 1.2	Chennai	2009	620000	Petrol	Manual	First	20.36 kmpl	1197 CC	78.9 bhp	5.0	Nal
2823	2823	Volkswagen Jetta 2013-2015 2.0L TDI Highline AT	Chennai	2015	480000	Diesel	Automatic	First	16.96 kmpl	1968 CC	138.03 bhp	5.0	Nal
3092	3092	Honda City i VTEC SV	Kolkata	2015	480000	Petrol	Manual	First	17.4 kmpl	1497 CC	117.3 bhp	5.0	Nal
4491	4491	Hyundai i20 Magna Optional 1.2	Bangalore	2013	445000	Petrol	Manual	First	18.5 kmpl	1197 CC	82.9 bhp	5.0	Nal
6921	6921	Maruti Swift Dzire Tour LDI	Jaipur	2012	350000	Diesel	Manual	First	23.4 kmpl	1248 CC	74 bhp	5.0	Nal
3649	3649	Tata Indigo LS	Jaipur	2008	300000	Diesel	Manual	First	17.0 kmpl	1405 CC	70 bhp	5.0	Nal
1528	1528	Toyota Innova 2.5 G (Diesel) 8 Seater BS IV	Hyderabad	2005	299322	Diesel	Manual	First	12.8 kmpl	2494 CC	102 bhp	8.0	Nal

It looks like the first row here is a data entry error. A car manufactured as recently as 2017 having been driven 6500000 kms is almost impossible.

The other observations that follow are also on a higher end. There is a good chance that these are outliers. We'll look at this furthur while doing the univariate analysis.

In [21]:
```
# Check Kilometers_Driven Extreme values
df.sort_values(by=["Kilometers_Driven"], ascending=True).head(10)
```

Out[21]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Pri
1361	1361	Maruti Alto 800 2016-2019 VXI	Mumbai	2019	171	Petrol	Manual	First	24.7 kmpl	796 CC	47.3 bhp	5.0	Na
5606	5606	Maruti Wagon R ZXI AMT 1.2	Mumbai	2019	600	Petrol	Automatic	First	21.5 kmpl	1197 CC	81.80 bhp	5.0	6.8 La
1198	1198	Volkswagen Polo 1.0 MPI Trendline	Hyderabad	2019	1000	Petrol	Manual	First	18.78 kmpl	999 CC	75 bhp	5.0	6.74 La
5941	5941	Renault KWID RXL	Pune	2018	1000	Petrol	Manual	First	25.17 kmpl	799 CC	53.3 bhp	5.0	4.45 La

6201	6201	Maruti Alto LXI	Kolkata	2019	1000	Petrol	Manual	First	24.7 kmpl	796 CC	47.3 bhp	5.0	3.99 La
1161	1161	Tata Tigor 1.2 Revotron XTA	Ahmedabad	2018	1000	Petrol	Automatic	First	20.3 kmpl	1199 CC	84 bhp	5.0	Na
1598	1598	Tata Xenon XT EX 4X2	Jaipur	2017	1000	Diesel	Manual	First	13.49 kmpl	2179 CC	138.03 bhp	5.0	Na
173	173	Hyundai Grand i10 1.2 Kappa Asta	Kolkata	2019	1000	Petrol	Manual	First	18.9 kmpl	1197 CC	81.86 bhp	5.0	7.39 La
1242	1242	Jaguar XE 2.0L Diesel Prestige	Delhi	2018	1000	Diesel	Automatic	First	13.6 kmpl	1999 CC	177 bhp	5.0	52.77 La
5339	5339	Hyundai i20 Active SX Dual Tone Petrol	Pune	2019	1000	Petrol	Manual	First	17.19 kmpl	1197 CC	81.86 bhp	5.0	10.25 La

After looking at the columns - Year, New Price and Price these entries seem feasible.

1000 might be default value in this case. Quite a few cars having driven exactly 1000 km is suspicious.

In [22]:
```
# Check seats extreme values
df.sort_values(by=["Seats"], ascending=True).head(100)
```

Out[22]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_P
3999	3999	Audi A4 3.2 FSI Tiptronic Quattro	Hyderabad	2012	125000	Petrol	Automatic	First	10.5 kmpl	3197 CC	null bhp	0.0	
693	693	Mercedes-Benz SLK-Class SLK 350	Coimbatore	2016	22732	Petrol	Automatic	First	18.1 kmpl	3498 CC	306 bhp	2.0	
798	798	Mercedes-Benz SLK-Class SLK 350	Bangalore	2015	10000	Petrol	Automatic	First	18.1 kmpl	3498 CC	306 bhp	2.0	
5781	5781	Lamborghini Gallardo Coupe	Delhi	2011	6500	Petrol	Automatic	Third	6.4 kmpl	5204 CC	560 bhp	2.0	
4722	4722	Mercedes-Benz SL-Class SL 500	Kolkata	2010	35000	Petrol	Automatic	First	8.1 kmpl	5461 CC	387.3 bhp	2.0	
134	134	Mercedes-Benz SLC 43 AMG	Kolkata	2017	13372	Petrol	Automatic	First	19.0 kmpl	2996 CC	362.07 bhp	2.0	95.04 L
915	915	Smart Fortwo CDI AT	Pune	2008	103000	Diesel	Automatic	Second	0.0 kmpl	799 CC	null bhp	2.0	
926	926	Porsche Cayman 2009-2012 S	Hyderabad	2010	10000	Petrol	Manual	First	9.0 kmpl	3436 CC	null bhp	2.0	
5919	5919	Jaguar F Type 5.0 V8 S	Hyderabad	2015	8000	Petrol	Automatic	First	12.5 kmpl	5000 CC	488.1 bhp	2.0	
4691	4691	Mercedes-Benz SLK-Class 55 AMG	Bangalore	2014	3000	Petrol	Automatic	Second	12.0 kmpl	5461 CC	421 bhp	2.0	
557	557	Audi TT 2.0 TFSI	Delhi	2013	12100	Petrol	Automatic	First	9.9 kmpl	1984 CC	207.8 bhp	2.0	
1288	1288	Audi TT 2.0 TFSI	Kochi	2014	14262	Petrol	Automatic	First	9.9 kmpl	1984 CC	207.8 bhp	2.0	
5294	5294	BMW Z4 2009-2013 35i	Delhi	2011	25000	Petrol	Automatic	First	10.37 kmpl	2979 CC	306 bhp	2.0	
2095	2095	Mercedes-Benz SLC 43 AMG	Coimbatore	2019	2526	Petrol	Automatic	First	19.0 kmpl	2996 CC	362.07 bhp	2.0	1.0
6960	6960	Mercedes-Benz SLC 43 AMG	Coimbatore	2018	18338	Petrol	Automatic	First	19.0 kmpl	2996 CC	362.07 bhp	2.0	1.0
2305	2305	Porsche Cayman 2009-2012 S tiptronic	Mumbai	2011	8000	Petrol	Automatic	First	9.0 kmpl	3436 CC	null bhp	2.0	

4893	4893	BMW Z4 2009-2013 Roadster 2.5i	Kochi	2018	9952	Petrol	Automatic	First	10.37 kmpl	2979 CC	306 bhp	2.0	
6842	6842	Nissan 370Z AT	Kolkata	2012	14850	Petrol	Automatic	First	10.0 kmpl	3696 CC	328.5 bhp	2.0	
1078	1078	Porsche Boxster S tiptronic	Kolkata	2015	10512	Petrol	Automatic	First	8.6 kmpl	2706 CC	265 bhp	2.0	
770	770	Chevrolet Cruze LTZ AT	Ahmedabad	2011	60000	Diesel	Automatic	First	18.1 kmpl	1991 CC	147.9 bhp	4.0	
148	148	Audi RS5 Coupe	Mumbai	2013	23000	Petrol	Automatic	First	11.05 kmpl	2894 CC	444 bhp	4.0	1.2
6826	6826	Tata Nano XTA	Coimbatore	2017	31176	Petrol	Automatic	First	21.9 kmpl	624 CC	37.48 bhp	4.0	
375	375	Maruti 800 AC	Chennai	2007	72000	Petrol	Manual	Third	16.1 kmpl	796 CC	37 bhp	4.0	
5286	5286	BMW 3 Series 320d Corporate Edition	Chennai	2008	89000	Diesel	Automatic	Second	16.07 kmpl	1995 CC	181 bhp	4.0	
3580	3580	Mercedes-Benz CLS-Class 2006-2010 350 CDI	Hyderabad	2010	29000	Diesel	Automatic	First	9.9 kmpl	3498 CC	271.72 bhp	4.0	
4223	4223	BMW X6 xDrive30d	Coimbatore	2014	39626	Diesel	Automatic	First	11.2 kmpl	2993 CC	241 bhp	4.0	
4500	4500	Tata Nano XTA	Bangalore	2016	29000	Petrol	Automatic	First	21.9 kmpl	624 CC	37.48 bhp	4.0	
761	761	Tata Nano Lx BSIV	Chennai	2011	35000	Petrol	Manual	First	25.4 kmpl	624 CC	37.48 bhp	4.0	
6388	6388	BMW X6 xDrive30d	Coimbatore	2014	58598	Diesel	Automatic	First	11.2 kmpl	2993 CC	241 bhp	4.0	
3228	3228	Maruti 800 Std	Pune	2003	52000	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0	
6846	6846	BMW 6 Series Gran Coupe	Kochi	2014	33128	Diesel	Automatic	First	17.54 kmpl	2993 CC	313 bhp	4.0	
2659	2659	Porsche Panamera Diesel	Kolkata	2015	25100	Diesel	Automatic	First	17.85 kmpl	2967 CC	300 bhp	4.0	
2422	2422	Jaguar XJ 3.0L Portfolio	Delhi	2016	12000	Diesel	Automatic	First	10.5 kmpl	2993 CC	271.23 bhp	4.0	
3553	3553	Maruti Alto 800 2016-2019 CNG LXI	Pune	2015	18000	CNG	Manual	First	33.44 km/kg	796 CC	40.3 bhp	4.0	
2645	2645	Tata Nano Twist XT	Coimbatore	2015	19963	Petrol	Manual	Second	25.4 kmpl	624 CC	37.5 bhp	4.0	
1225	1225	Maruti 800 DX BSII	Jaipur	2009	32000	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0	
5397	5397	Mini Cooper S Carbon Edition	Mumbai	2013	52000	Petrol	Automatic	First	13.6 kmpl	1598 CC	181 bhp	4.0	
2089	2089	BMW 6 Series 640d Coupe	Mumbai	2013	30000	Diesel	Automatic	First	9.52 kmpl	2993 CC	313 bhp	4.0	
3523	3523	Jaguar XJ 3.0L Premium Luxury	Hyderabad	2012	50000	Diesel	Automatic	First	12.9 kmpl	2993 CC	271.23 bhp	4.0	
1628	1628	Maruti 800 Std BSIII	Jaipur	2004	12000	Petrol	Manual	Second	16.1 kmpl	796 CC	37 bhp	4.0	
112	112	Tata Nano Twist XT	Bangalore	2014	25500	Petrol	Manual	First	25.4 kmpl	624 CC	37.5 bhp	4.0	
6120	6120	Tata Nano Lx	Hyderabad	2012	25183	Petrol	Manual	First	26.0 kmpl	624 CC	35 bhp	4.0	
3119	3119	Maruti Alto K10 LXI CNG Optional	Kochi	2018	44202	CNG	Manual	First	32.26 km/kg	998 CC	58.2 bhp	4.0	4.66 l
1837	1837	Chevrolet Cruze LTZ AT	Pune	2009	73204	Diesel	Automatic	Third	18.1 kmpl	1991 CC	147.9 bhp	4.0	
124	124	Tata Nano XTA	Coimbatore	2017	32684	Petrol	Automatic	First	21.9 kmpl	624 CC	37.48 bhp	4.0	
3017	3017	Mini Cooper S Carbon Edition	Kolkata	2018	2971	Petrol	Automatic	First	13.6 kmpl	1598 CC	181 bhp	4.0	

391	391	Mini Cooper Convertible 1.6	Bangalore	2015	20000	Petrol	Automatic	First	18.86 kmpl	1598 CC	122 bhp	4.0		
3266	3266	Mercedes-Benz E-Class E400 Cabriolet	Kolkata	2015	7501	Petrol	Automatic	First	10.0 kmpl	2996 CC	333 bhp	4.0	87.57 L	
5657	5657	Maruti 800 Std	Jaipur	2002	75000	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0		
5868	5868	BMW 3 Series 330d Convertible	Kochi	2014	51240	Diesel	Automatic	First	8.2 kmpl	2993 CC	245 bhp	4.0		
7102	7102	Maruti 800 AC	Kolkata	2009	9999	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0		
3277	3277	Mini Cooper 3 DOOR D	Kochi	2016	32298	Diesel	Automatic	First	20.7 kmpl	1496 CC	113.98 bhp	4.0	35 L	
5468	5468	Tata Nano CX	Kochi	2014	59508	Petrol	Manual	First	25.4 kmpl	624 CC	37.48 bhp	4.0		
5088	5088	Jaguar XJ 5.0 L V8 Supercharged	Coimbatore	2011	43686	Petrol	Automatic	First	10.5 kmpl	5000 CC	503 bhp	4.0		
5474	5474	Tata Nano Twist XT	Coimbatore	2015	30676	Petrol	Manual	First	25.4 kmpl	624 CC	37.5 bhp	4.0		
6538	6538	Maruti Alto K10 LXI CNG	Kochi	2015	47490	CNG	Manual	First	32.26 km/kg	998 CC	58.2 bhp	4.0		
326	326	BMW 6 Series 640d Gran Coupe	Mumbai	2011	30000	Diesel	Automatic	First	9.52 kmpl	2993 CC	313 bhp	4.0		
5050	5050	Tata Nano Twist XT	Ahmedabad	2014	18050	Petrol	Manual	Second	25.4 kmpl	624 CC	37.5 bhp	4.0		
1433	1433	Maruti 800 AC	Jaipur	2007	59000	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0		
6019	6019	Maruti Alto K10 LXI CNG	Delhi	2014	40929	CNG	Manual	First	32.26 km/kg	998 CC	58.2 bhp	4.0		
5935	5935	Maruti Alto K10 LXI CNG	Pune	2015	59525	CNG	Manual	Second	32.26 km/kg	998 CC	58.2 bhp	4.0		
3665	3665	Tata Nano XTA	Coimbatore	2017	18351	Petrol	Automatic	First	21.9 kmpl	624 CC	37.48 bhp	4.0		
3157	3157	Tata Nano Twist XT	Hyderabad	2014	16810	Petrol	Manual	First	25.4 kmpl	624 CC	37.5 bhp	4.0		
3756	3756	Maruti 800 AC Uniq	Jaipur	2009	22100	Petrol	Manual	Second	16.1 kmpl	796 CC	37 bhp	4.0		
6008	6008	Porsche Panamera Diesel	Hyderabad	2013	40000	Diesel	Automatic	Second	17.85 kmpl	2967 CC	300 bhp	4.0		
6467	6467	Tata Nano XT	Bangalore	2016	14000	Petrol	Manual	Fourth & Above	23.9 kmpl	624 CC	37.48 bhp	4.0		
6906	6906	Fiat Abarth 595 Competizione	Mumbai	2017	19476	Petrol	Automatic	First	19.0 kmpl	1368 CC	160 bhp	4.0		
2293	2293	Tata Nano Lx	Coimbatore	2013	37287	Petrol	Manual	First	26.0 kmpl	624 CC	35 bhp	4.0		
5603	5603	Porsche Panamera 2010 2013 Diesel	Delhi	2013	36400	Diesel	Automatic	First	7.5 kmpl	4806 CC	394.3 bhp	4.0		
6482	6482	Mini Cooper 3 DOOR D	Coimbatore	2017	26002	Diesel	Automatic	First	20.7 kmpl	1496 CC	113.98 bhp	4.0	36.5 L	
1458	1458	Tata Nano LX	Pune	2013	4700	Petrol	Manual	First	25.4 kmpl	624 CC	37.48 bhp	4.0		
4627	4627	BMW 6 Series 650i Coupe	Kochi	2010	65329	Petrol	Automatic	First	7.94 kmpl	4395 CC	450 bhp	4.0		
2322	2322	Tata Nano XT	Pune	2011	50000	Petrol	Manual	Second	23.9 kmpl	624 CC	37.48 bhp	4.0		
4965	4965	Tata Nano STD SE	Jaipur	2012	80000	Petrol	Manual	Second	25.4 kmpl	624 CC	37.5 bhp	4.0		
7057	7057	BMW 6 Series 650i Coupe	Delhi	2009	64000	Petrol	Automatic	First	7.94 kmpl	4395 CC	450 bhp	4.0		
5521	5521	Bentley Continental Flying Spur	Hyderabad	2006	48000	Petrol	Automatic	First	8.6 kmpl	5998 CC	552 bhp	4.0		
5518	5518	Mini Cooper 3 DOOR S	Kochi	2016	21110	Petrol	Automatic	First	17.44 kmpl	1998 CC	189.08 bhp	4.0	39.27 L	

6569	6569	BMW X6 xDrive30d	Hyderabad	2011	76000	Diesel	Automatic	First	11.2 kmpl	2993 CC	241 bhp	4.0	
5481	5481	Chevrolet Cruze LTZ AT	Hyderabad	2011	78203	Diesel	Automatic	First	18.1 kmpl	1991 CC	147.9 bhp	4.0	
740	740	Tata Nano XTA	Coimbatore	2016	24941	Petrol	Automatic	First	21.9 kmpl	624 CC	37.48 bhp	4.0	
4061	4061	Audi RS5 Coupe	Mumbai	2013	23312	Petrol	Automatic	First	11.05 kmpl	2894 CC	444 bhp	4.0	1.2
1545	1545	Mini Cooper 3 DOOR D	Mumbai	2016	20000	Diesel	Automatic	First	20.7 kmpl	1496 CC	113.98 bhp	4.0	35 L
4185	4185	Porsche Panamera Diesel	Kochi	2015	54996	Diesel	Automatic	First	17.85 kmpl	2967 CC	300 bhp	4.0	
3194	3194	Mini Cooper 3 DOOR D	Kochi	2015	27641	Diesel	Automatic	Second	20.7 kmpl	1496 CC	113.98 bhp	4.0	34.89 L
3632	3632	BMW X6 xDrive30d	Delhi	2011	59000	Diesel	Automatic	First	11.2 kmpl	2993 CC	241 bhp	4.0	
228	228	Mini Cooper Convertible S	Kochi	2017	26327	Petrol	Automatic	First	16.82 kmpl	1998 CC	189.08 bhp	4.0	44.28 L
4561	4561	Tata Nano Cx BSIV	Coimbatore	2014	42083	Petrol	Manual	First	25.4 kmpl	624 CC	37.48 bhp	4.0	
2276	2276	BMW X6 xDrive30d	Jaipur	2013	45306	Diesel	Automatic	Second	11.2 kmpl	2993 CC	241 bhp	4.0	
718	718	Mini Cooper S	Pune	2012	37000	Petrol	Automatic	Second	13.6 kmpl	1598 CC	181 bhp	4.0	
2706	2706	Porsche Panamera Diesel 250hp	Kochi	2014	60033	Diesel	Automatic	First	18.18 kmpl	2967 CC	250 bhp	4.0	
5926	5926	Maruti 800 DX BSII	Pune	2000	78000	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0	
715	715	BMW 3 Series 320d Corporate Edition	Kolkata	2013	38998	Diesel	Automatic	First	16.07 kmpl	1995 CC	181 bhp	4.0	
2364	2364	Mini Cooper Convertible S	Chennai	2012	17000	Petrol	Automatic	Second	16.82 kmpl	1998 CC	189.08 bhp	4.0	44.28 L
2711	2711	Mini Cooper Convertible S	Kochi	2017	20469	Petrol	Automatic	First	16.82 kmpl	1998 CC	189.08 bhp	4.0	44.28 L
1429	1429	Volkswagen Polo GTI	Chennai	2014	35059	Petrol	Automatic	First	17.21 kmpl	1798 CC	189 bhp	4.0	
2370	2370	Mini Cooper 3 DOOR S	Mumbai	2013	21000	Petrol	Automatic	First	17.44 kmpl	1998 CC	189.08 bhp	4.0	39.57 L
6788	6788	Maruti Alto K10 LXI CNG Optional	Delhi	2017	33000	CNG	Manual	First	32.26 km/kg	998 CC	58.2 bhp	4.0	4.66 L
1602	1602	Maruti 800 AC BSIII	Hyderabad	2011	83969	Petrol	Manual	First	16.1 kmpl	796 CC	37 bhp	4.0	
5387	5387	Tata Nano Twist XT	Coimbatore	2015	25811	Petrol	Manual	Second	25.4 kmpl	624 CC	37.5 bhp	4.0	
6774	6774	Tata Nano Cx BSIV	Ahmedabad	2013	35000	Petrol	Manual	First	25.4 kmpl	624 CC	37.48 bhp	4.0	

Audi A4 having 0 seats is clearly a data entry error. This column warrents some outlier treatment or we can treat seats == 0 as a missing value. Overall, there doesn't seem not much to be concerned about here.

In [23]:

```
# Let us check if we have a similar car in our dataset.
df[df["Name"].str.startswith("Audi A4")]
# Looks like an Audi A4 typically has 5 seats.
```

Out[23]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Pric
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
50	50	Audi A4 2.0 TDI 177 Bhp Premium Plus	Kochi	2015	13648	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
65	65	Audi A4 2.0 TDI Multitronic	Jaipur	2012	65664	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0	Na

103	103	Audi A4 3.0 TDI Quattro Premium	Kolkata	2010	30000	Diesel	Automatic	First	14.94 kmpl	2967 CC	241.4 bhp	5.0	Na
150	150	Audi A4 2.0 TDI 177 Bhp Premium Plus	Coimbatore	2015	48214	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
717	717	Audi A4 2.0 TDI 177 Bhp Premium Plus	Bangalore	2013	45979	Diesel	Automatic	Second	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
853	853	Audi A4 2.0 TDI	Delhi	2011	46000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
854	854	Audi A4 2.0 TDI	Kolkata	2012	47346	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
874	874	Audi A4 2.0 TDI Multitronic	Ahmedabad	2011	59000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0	Na
964	964	Audi A4 2.0 TDI	Kolkata	2009	34000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
1044	1044	Audi A4 35 TDI Technology	Hyderabad	2015	45000	Diesel	Automatic	First	18.25 kmpl	1968 CC	187.74 bhp	5.0	55.61 Lak
1268	1268	Audi A4 2.0 TDI 177 Bhp Premium Plus	Kochi	2014	77395	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
1316	1316	Audi A4 2.0 TDI	Delhi	2013	36000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
1637	1637	Audi A4 35 TDI Premium	Coimbatore	2016	68193	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
1807	1807	Audi A4 2.0 TDI 177 Bhp Premium Plus	Bangalore	2014	21000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
1868	1868	Audi A4 New 2.0 TDI Multitronic	Delhi	2012	60000	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
1900	1900	Audi A4 3.0 TDI Quattro Premium	Coimbatore	2012	46913	Diesel	Automatic	First	14.94 kmpl	2967 CC	241.4 bhp	5.0	Na
1918	1918	Audi A4 2.0 TDI	Kochi	2013	45330	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
1929	1929	Audi A4 2.0 TDI 177 Bhp Premium Plus	Chennai	2014	70000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
2005	2005	Audi A4 2.0 TDI	Ahmedabad	2013	82002	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
2024	2024	Audi A4 New 2.0 TDI Multitronic	Delhi	2012	44555	Diesel	Automatic	First	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
2044	2044	Audi A4 2.0 TDI	Delhi	2013	67000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
2060	2060	Audi A4 2.0 TDI 177 Bhp Technology Edition	Bangalore	2014	24000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
2078	2078	Audi A4 2.0 TDI Multitronic	Jaipur	2011	105000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	140 bhp	5.0	Na
2084	2084	Audi A4 2.0 TDI	Mumbai	2013	45200	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
2114	2114	Audi A4 35 TDI Premium	Hyderabad	2015	50000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
2189	2189	Audi A4 2.0 TFSI	Mumbai	2011	38000	Petrol	Automatic	First	10.8 kmpl	1984 CC	132 bhp	5.0	Na
2240	2240	Audi A4 2.0 TDI	Hyderabad	2014	60000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na

2284	2284	Audi A4 2.0 TDI 177 Bhp Premium Plus	Pune	2013	87000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
2298	2298	Audi A4 2.0 TDI	Hyderabad	2011	85000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0		Na
2484	2484	Audi A4 35 TDI Premium Sport	Chennai	2016	11000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
2506	2506	Audi A4 35 TDI Premium	Coimbatore	2016	39644	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
2553	2553	Audi A4 2.0 TDI Premium Sport Limited Edition	Coimbatore	2015	21979	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
2586	2586	Audi A4 2.0 TDI 177 Bhp Premium Plus	Coimbatore	2014	57773	Diesel	Automatic	Second	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
2626	2626	Audi A4 1.8 TFSI	Kolkata	2010	38001	Petrol	Automatic	First	12.3 kmpl	1781 CC	163.2 bhp	5.0		Na
2898	2898	Audi A4 2.0 TDI Multitronic	Hyderabad	2014	53000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0		Na
3151	3151	Audi A4 2.0 TDI 177 Bhp Premium Plus	Coimbatore	2015	32205	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
3158	3158	Audi A4 30 TFSI Premium Plus	Kochi	2017	61221	Petrol	Automatic	First	17.84 kmpl	1395 CC	147.51 bhp	5.0	47.16 Lak	
3162	3162	Audi A4 New 2.0 TDI Multitronic	Delhi	2012	69000	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0		Na
3261	3261	Audi A4 2.0 TDI 177 Bhp Premium Plus	Kolkata	2013	60001	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
3361	3361	Audi A4 1.8 TFSI	Mumbai	2011	53000	Petrol	Automatic	First	12.3 kmpl	1781 CC	163.2 bhp	5.0		Na
3434	3434	Audi A4 2.0 TDI 177 Bhp Premium Plus	Bangalore	2014	40000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
3492	3492	Audi A4 2.0 TDI Multitronic	Ahmedabad	2012	74002	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0		Na
3535	3535	Audi A4 35 TDI Premium	Coimbatore	2016	39237	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
3604	3604	Audi A4 New 2.0 TDI Multitronic	Jaipur	2012	60000	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0		Na
3702	3702	Audi A4 2.0 TDI Multitronic	Delhi	2010	59000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0		Na
3717	3717	Audi A4 2.0 TDI 177 Bhp Premium Plus	Kolkata	2014	40000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
3730	3730	Audi A4 35 TDI Premium	Bangalore	2015	58970	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0		Na
3899	3899	Audi A4 2.0 TDI	Bangalore	2013	76000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0		Na
3999	3999	Audi A4 3.2 FSI Tiptronic Quattro	Hyderabad	2012	125000	Petrol	Automatic	First	10.5 kmpl	3197 CC	null bhp	0.0		Na

4095	4095	Audi A4 3.0 TDI Quattro	Coimbatore	2015	35463	Diesel	Automatic	First	14.94 kmpl	2967 CC	241.4 bhp	5.0	Na
4173	4173	Audi A4 2.0 TDI Multitronic	Delhi	2009	88000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0	Na
4236	4236	Audi A4 35 TDI Premium Plus	Kochi	2015	13506	Diesel	Automatic	First	18.25 kmpl	1968 CC	187.74 bhp	5.0	50.4 Lak
4379	4379	Audi A4 2.0 TDI Multitronic	Delhi	2011	58000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0	Na
4404	4404	Audi A4 35 TDI Premium	Hyderabad	2015	60000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
4517	4517	Audi A4 New 2.0 TDI Multitronic	Delhi	2012	60000	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
4591	4591	Audi A4 35 TDI Technology Edition	Kochi	2019	38163	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
4622	4622	Audi A4 2.0 TDI Multitronic	Ahmedabad	2015	13000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0	Na
4676	4676	Audi A4 35 TDI Technology Edition	Hyderabad	2015	30000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
4846	4846	Audi A4 2.0 TDI	Hyderabad	2013	52000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
4858	4858	Audi A4 35 TDI Technology	Hyderabad	2017	14000	Diesel	Automatic	First	18.25 kmpl	1968 CC	187.74 bhp	5.0	55.61 Lak
4868	4868	Audi A4 2.0 TDI 177 Bhp Premium Plus	Bangalore	2013	88000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
4922	4922	Audi A4 New 2.0 TDI Multitronic	Kolkata	2008	17000	Diesel	Automatic	First	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
4942	4942	Audi A4 35 TDI Premium Plus	Coimbatore	2014	28469	Diesel	Automatic	First	18.25 kmpl	1968 CC	187.74 bhp	5.0	53.14 Lak
5027	5027	Audi A4 2.0 TDI 177 Bhp Technology Edition	Bangalore	2014	29000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
5042	5042	Audi A4 2.0 TDI Premium Sport Limited Edition	Coimbatore	2015	44811	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
5100	5100	Audi A4 1.8 TFSI Technology Edition	Hyderabad	2012	80000	Petrol	Automatic	First	15.64 kmpl	1798 CC	167.62 bhp	5.0	Na
5179	5179	Audi A4 2.0 TDI	Hyderabad	2011	54000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
5189	5189	Audi A4 2.0 TDI Multitronic	Bangalore	2012	88000	Diesel	Automatic	First	16.55 kmpl	1968 CC	140 bhp	5.0	Na
5264	5264	Audi A4 2.0 TDI	Kochi	2014	55730	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
5362	5362	Audi A4 2.0 TDI 177 Bhp Premium Plus	Bangalore	2014	29063	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
5429	5429	Audi A4 2.0 TDI	Kolkata	2009	46000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
5488	5488	Audi A4 2.0 TDI	Kolkata	2009	46000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
5561	5561	Audi A4 3.0 TDI	Bangalore	2014	30000	Diesel	Automatic	First	14.94	2967	241.4	5.0	Na

		Quattro								kmpl	CC	bhp		
5699	5699	Audi A4 2.0 TDI	Bangalore	2014	37600	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
5776	5776	Audi A4 2.0 TDI 177 Bhp Premium Plus	Chennai	2014	40000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
5877	5877	Audi A4 2.0 TDI Celebration Edition	Delhi	2011	57000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
5989	5989	Audi A4 35 TDI Premium Plus	Coimbatore	2013	58629	Diesel	Automatic	Second	18.25 kmpl	1968 CC	187.74 bhp	5.0	53.14 Lak
6060	6060	Audi A4 2.0 TDI 177 Bhp Technology Edition	Pune	2013	87001	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
6061	6061	Audi A4 2.0 TDI Multitronic	Hyderabad	2013	55000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	140 bhp	5.0	Na
6287	6287	Audi A4 New 2.0 TDI Multitronic	Delhi	2012	67000	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	Na
6377	6377	Audi A4 35 TDI Technology	Hyderabad	2015	59000	Diesel	Automatic	First	18.25 kmpl	1968 CC	187.74 bhp	5.0	55.61 Lak
6399	6399	Audi A4 2.0 TDI	Bangalore	2013	46000	Diesel	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
6461	6461	Audi A4 2.0 TDI	Delhi	2012	70000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
6888	6888	Audi A4 35 TDI Premium Plus	Coimbatore	2016	60249	Diesel	Automatic	First	18.25 kmpl	1968 CC	187.74 bhp	5.0	53.14 Lak
6975	6975	Audi A4 3.0 TDI Quattro Premium	Bangalore	2011	45000	Diesel	Automatic	First	14.94 kmpl	2967 CC	241.4 bhp	5.0	Na
7023	7023	Audi A4 2.0 TDI Premium Sport Limited Edition	Bangalore	2014	36000	Diesel	Automatic	First	17.11 kmpl	1968 CC	174.33 bhp	5.0	Na
7168	7168	Audi A4 2.0 TDI	Bangalore	2014	65475	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
7233	7233	Audi A4 2.0 TDI	Bangalore	2014	21143	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na
7238	7238	Audi A4 2.0 TDI	Hyderabad	2011	64000	Diesel	Automatic	First	16.55 kmpl	1968 CC	147.51 bhp	5.0	Na

In [24]:
```
# Let us replace #seats in row index 3999 form 0 to 5
df.loc[3999, "Seats"] = 5.0
```

In [25]:
```
# Check seats extreme values
df.sort_values(by=["Seats"], ascending=False).head(5)
```

Out[25]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price
2575	2575	Chevrolet Tavera LS B3 10 Seats BSIII	Hyderabad	2015	120000	Diesel	Manual	First	14.8 kmpl	2499 CC	80 bhp	10.0	NaN
1907	1907	Toyota Qualis FS B3	Bangalore	2002	63000	Diesel	Manual	Third	13.1 kmpl	2446 CC	75 bhp	10.0	NaN
6288	6288	Chevrolet Tavera LS B3 10 Seats BSIII	Hyderabad	2005	150000	Diesel	Manual	Second	14.8 kmpl	2499 CC	80 bhp	10.0	NaN

6242	6242	Tata Sumo EX 10/7 Str BSII	Chennai	2015	196000	Diesel	Manual	Second	12.2 kmpl	1948 CC	68 bhp	10.0	NaN
814	814	Toyota Qualis FS B2	Pune	2004	77757	Diesel	Manual	Second	13.1 kmpl	2446 CC	75 bhp	10.0	NaN

Of course, a Toyota Qualis has 10 seats and so does a Tata Sumo. We don't see any data entry error here.

In [26]:
```
# Check Mileage - km_per_unit_fuel extreme values
df.sort_values(by=["km_per_unit_fuel"], ascending=True).head(10)
```

Out[26]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Pric
2597	2597	Hyundai Santro Xing XP	Pune	2007	70000	Petrol	Manual	First	0.0 kmpl	1086 CC	null bhp	5.0	Na
2343	2343	Hyundai Santro AT	Hyderabad	2006	74483	Petrol	Automatic	First	0.0 kmpl	999 CC	null bhp	5.0	Na
5270	5270	Honda City 1.5 GXI	Bangalore	2002	53000	Petrol	Manual	Second	0.0 kmpl	NaN	NaN	NaN	Na
424	424	Volkswagen Jetta 2007-2011 1.9 L TDI	Hyderabad	2010	42021	Diesel	Manual	First	0.0 kmpl	1968 CC	null bhp	5.0	Na
6857	6857	Land Rover Freelander 2 TD4 SE	Mumbai	2011	87000	Diesel	Automatic	First	0.0 kmpl	2179 CC	115 bhp	5.0	Na
443	443	Hyundai Santro GLS I - Euro I	Coimbatore	2012	50243	Petrol	Manual	First	0.0 kmpl	1086 CC	null bhp	5.0	Na
5119	5119	Hyundai Santro Xing XP	Kolkata	2008	45500	Petrol	Manual	Second	0.0 kmpl	1086 CC	null bhp	5.0	Na
5022	5022	Land Rover Freelander 2 TD4 SE	Hyderabad	2013	46000	Diesel	Automatic	Second	0.0 kmpl	2179 CC	115 bhp	5.0	Na
5016	5016	Land Rover Freelander 2 TD4 HSE	Delhi	2013	72000	Diesel	Automatic	First	0.0 kmpl	2179 CC	115 bhp	5.0	Na
2542	2542	Hyundai Santro GLS II - Euro II	Bangalore	2011	65000	Petrol	Manual	Second	0.0 kmpl	NaN	NaN	NaN	Na

We will have to treat Mileage = 0 as missing values

In [27]:
```
# Check Mileage - km_per_unit_fuel extreme values
df.sort_values(by=["km_per_unit_fuel"], ascending=False).head(10)
```

Out[27]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Pri
1332	1332	Maruti Wagon R CNG LXI	Pune	2013	79494	CNG	Manual	First	33.54 km/kg	998 CC	67.04 bhp	5.0	5.54 Lakh	2.5
2059	2059	Maruti Wagon R CNG LXI	Mumbai	2013	54000	CNG	Manual	First	33.54 km/kg	998 CC	67.04 bhp	5.0	5.58 Lakh	3.(
2371	2371	Maruti Wagon R CNG LXI	Pune	2014	29202	CNG	Manual	First	33.54 km/kg	998 CC	67.04 bhp	5.0	5.54 Lakh	3.5
3129	3129	Maruti Wagon R CNG LXI	Delhi	2014	74663	CNG	Manual	First	33.54 km/kg	998 CC	67.04 bhp	5.0	5.35 Lakh	3.2
4141	4141	Maruti Wagon R CNG LXI	Mumbai	2014	47200	CNG	Manual	First	33.54 km/kg	998 CC	67.04 bhp	5.0	5.58 Lakh	2.8

3869	3869	Maruti Alto 800 2016-2019 CNG LXI	Delhi	2012	65537	CNG	Manual	Second	33.44 km/kg	796 CC	40.3 bhp	4.0	NaN	2.
1269	1269	Maruti Alto 800 2016-2019 CNG LXI	Mumbai	2018	10600	CNG	Manual	First	33.44 km/kg	796 CC	40.3 bhp	4.0	NaN	3.
3553	3553	Maruti Alto 800 2016-2019 CNG LXI	Pune	2015	18000	CNG	Manual	First	33.44 km/kg	796 CC	40.3 bhp	4.0	NaN	3.
4769	4769	Maruti Alto 800 2016-2019 CNG LXI	Kochi	2017	24310	CNG	Manual	First	33.44 km/kg	796 CC	40.3 bhp	4.0	NaN	3.
6019	6019	Maruti Alto K10 LXI CNG	Delhi	2014	40929	CNG	Manual	First	32.26 km/kg	998 CC	58.2 bhp	4.0	NaN	Na

Maruti Wagon R and Maruti Alto CNG versions are budget friendly cars with high mileage so these data points are fine.

In [28]:

```
# looking at value counts for non-numeric features

num_to_display = 10  # defining this up here so it's easy to change later
for colname in df.dtypes[df.dtypes == "object"].index:
    val_counts = df[colname].value_counts(dropna=False)  # Will also show the NA counts
    print(val_counts[:num_to_display])
    if len(val_counts) > num_to_display:
        print(f"Only displaying first {num_to_display} of {len(val_counts)} values.")
    print("\n\n")  # just for more space in between
```

```
Mahindra XUV500 W8 2WD        55
Maruti Swift VDI              49
Maruti Swift Dzire VDI        42
Honda City 1.5 S MT           39
Maruti Swift VDI BSIV         37
Maruti Ritz VDi               35
Toyota Fortuner 3.0 Diesel    35
Honda Brio S MT               32
Honda City 1.5 V MT           32
Honda Amaze S i-Dtech         32
Name: Name, dtype: int64
Only displaying first 10 of 2041 values.



Mumbai        949
Hyderabad     876
Kochi         772
Coimbatore    772
Pune          765
Delhi         660
Kolkata       654
Chennai       591
Jaipur        499
Bangalore     440
Name: Location, dtype: int64
Only displaying first 10 of 11 values.



Diesel      3852
Petrol      3325
CNG           62
LPG           12
Electric       2
Name: Fuel_Type, dtype: int64
```

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

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

```
17.0 kmpl     207
18.9 kmpl     201
18.6 kmpl     144
21.1 kmpl     106
20.36 kmpl    105
17.8 kmpl      98
18.0 kmpl      89
12.8 kmpl      87
18.5 kmpl      86
16.0 kmpl      85
Name: Mileage, dtype: int64
Only displaying first 10 of 451 values.
```

```
1197 CC    732
1248 CC    610
1498 CC    370
998 CC     309
1198 CC    281
2179 CC    278
1497 CC    273
1968 CC    266
1995 CC    212
1461 CC    188
Name: Engine, dtype: int64
Only displaying first 10 of 151 values.
```

```
74 bhp       280
98.6 bhp     166
73.9 bhp     152
140 bhp      142
null bhp     129
78.9 bhp     128
67.1 bhp     126
67.04 bhp    125
82 bhp       124
88.5 bhp     120
Name: Power, dtype: int64
Only displaying first 10 of 387 values.
```

```
NaN           6247
63.71 Lakh       6
95.13 Lakh       6
33.36 Lakh       6
4.78 Lakh        6
9.12 Lakh        5
5.04 Lakh        5
11.26 Lakh       5
44.28 Lakh       5
4.98 Lakh        5
Name: New_Price, dtype: int64
Only displaying first 10 of 626 values.
```

```
kmpl     7177
km/kg      74
NaN         2
Name: mileage_unit, dtype: int64
```

```
maruti           1444
hyundai          1340
honda             743
toyota            507
mercedes-benz     380
volkswagen        374
ford              351
mahindra          331
bmw               312
audi              285
Name: Brand, dtype: int64
Only displaying first 10 of 32 values.
```

```
swift     418
city      318
i20       303
innova    203
verna     200
alto      183
grand     183
i10       181
polo      178
wagon     178
Name: Model, dtype: int64
Only displaying first 10 of 218 values.
```

Since we haven't dropped the original columns that we processed, we have a few redunadant output here.

We had checked cars of different `Fuel_Type` earlier, but we did not encounter the 2 electric cars. Let us check why.

In [29]:
```
df.loc[df["Fuel_Type"] == "Electric"]
```

Out[29]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	P
4446	4446	Mahindra E Verito D4	Chennai	2016	50000	Electric	Automatic	First	NaN	72 CC	41 bhp	5.0	13.58 Lakh	1
4904	4904	Toyota Prius 2009-2016 Z4	Mumbai	2011	44000	Electric	Automatic	First	NaN	1798 CC	73 bhp	5.0	NaN	1

Mileage values for these cars are NaN, that is why we did not encounter these earlier with groupby.

Electric cars are very new in the market and very rare in our dataset. We can consider dropping these two observations if they turn out to be outliers later. There is a good chance that we will not be able to create a good price prediction model for electric cars, with the currently available data.

New Price for 6247 entries is missing. We need to explore if we can impute these or we should drop this column altogether.

Missing Values

Before we start looking at the individual distributions and interactions, let's quickly check the missingness in the data.

In [30]:
```
df.isnull().sum()
```

Out[30]:
```
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               46
Seats               53
```

```
New_Price            6247
Price                1234
km_per_unit_fuel        2
mileage_unit            2
engine_num             46
power_num             175
new_price_num        6247
Brand                   0
Model                   0
dtype: int64
```

- 2 Electric car variants don't have entries for Mileage.
- Engine displacement information of 46 observations is missing and maximum power of 175 entries is missing.
- Information about number of seats is not avaliable for 53 entries.
- New Price as we saw earlier has a huge missing count. We'll have to see if there is a pattern here.
- Price is also missing for 1234 entries. Since price is our response variable that we want to predict, we will have to drop these rows when we actually build a model. These rows will not be able to help us in modelling or model evaluation. But while we are analysing the distributions and doing missing value imputations, we will keep using information from these rows.

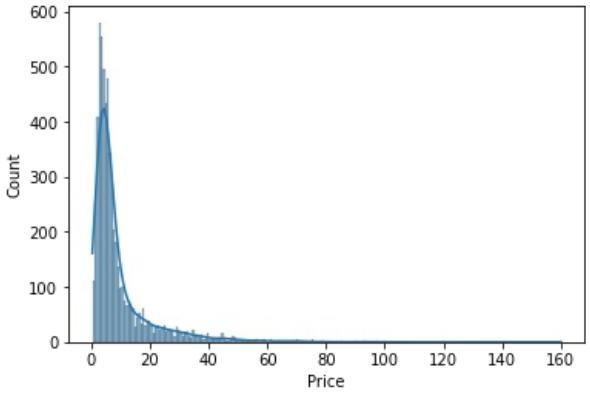
In [31]:
```
# Drop the redundant columns.
df.drop(
    columns=["Mileage", "mileage_unit", "Engine", "Power", "New_Price"], inplace=True
)
```

Distributions

Price

In [32]:
```
sns.histplot(df["Price"], kde=True)
```

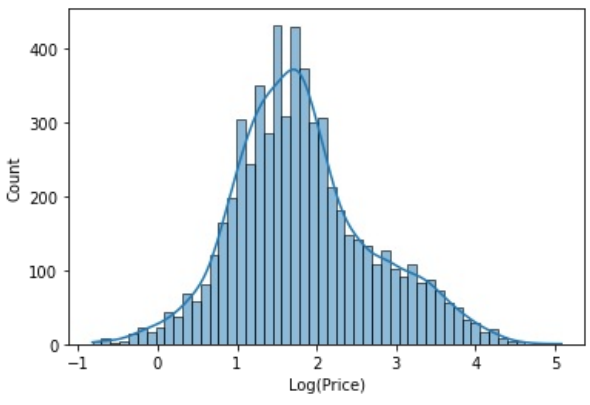
Out[32]:
```
<AxesSubplot:xlabel='Price', ylabel='Count'>
```



This is a highly skewed distribution. Let us use log transformation on this column to see of that helps normalise the distribution.

In [33]:
```
sns.histplot(np.log(df["Price"]), kde=True)
plt.xlabel("Log(Price)")
```

Out[33]:
```
Text(0.5, 0, 'Log(Price)')
```

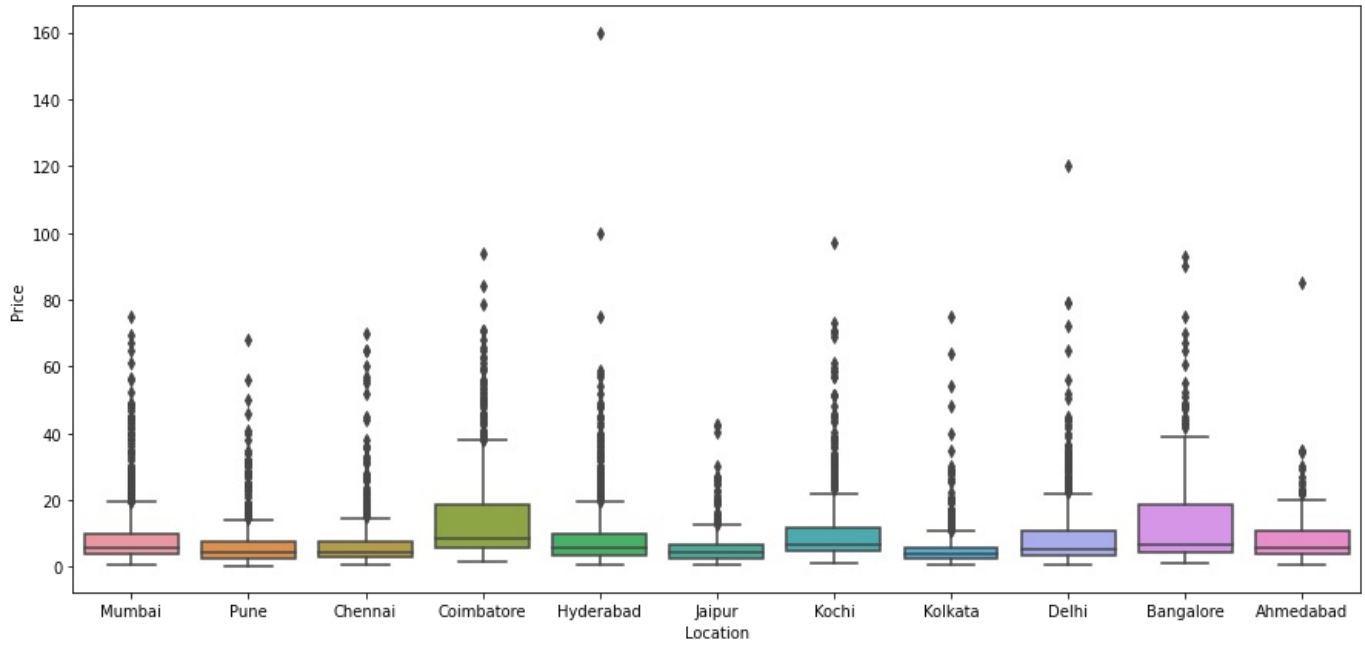


In [34]:
```
# Log Transformation has definitely helped in reducing the skew
# Creating a new column with the transformed variable.
df["price_log"] = np.log(df["Price"])
```

Price vs Location

In [35]:
```
plt.figure(figsize=(15, 7))
sns.boxplot(x="Location", y="Price", data=df)
```

Out[35]: <AxesSubplot:xlabel='Location', ylabel='Price'>

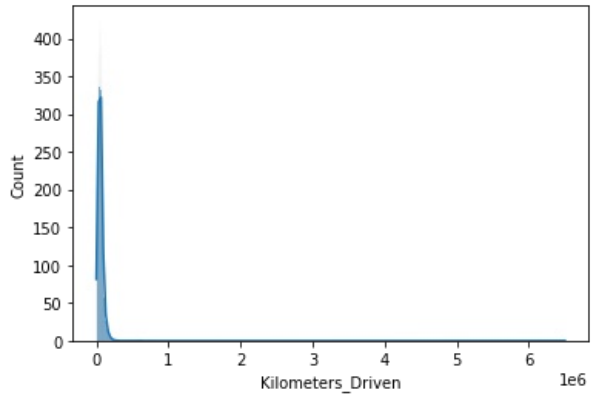


Price of used cars has a large IQR in Coimbatore and Bangalore

Kilometers_Driven

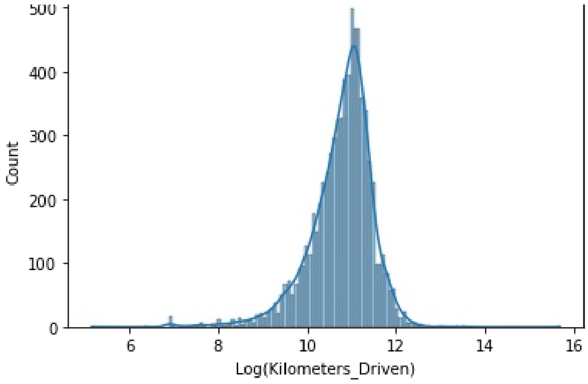
In [36]:
```
sns.histplot(df["Kilometers_Driven"], kde=True)
```

Out[36]: <AxesSubplot:xlabel='Kilometers_Driven', ylabel='Count'>



In [37]:
```
# Log transformation
sns.histplot(np.log(df["Kilometers_Driven"]), kde=True)
plt.xlabel('Log(Kilometers_Driven)')
```

Out[37]: Text(0.5, 0, 'Log(Kilometers_Driven)')



Transformation has reduced the extreme skewness.

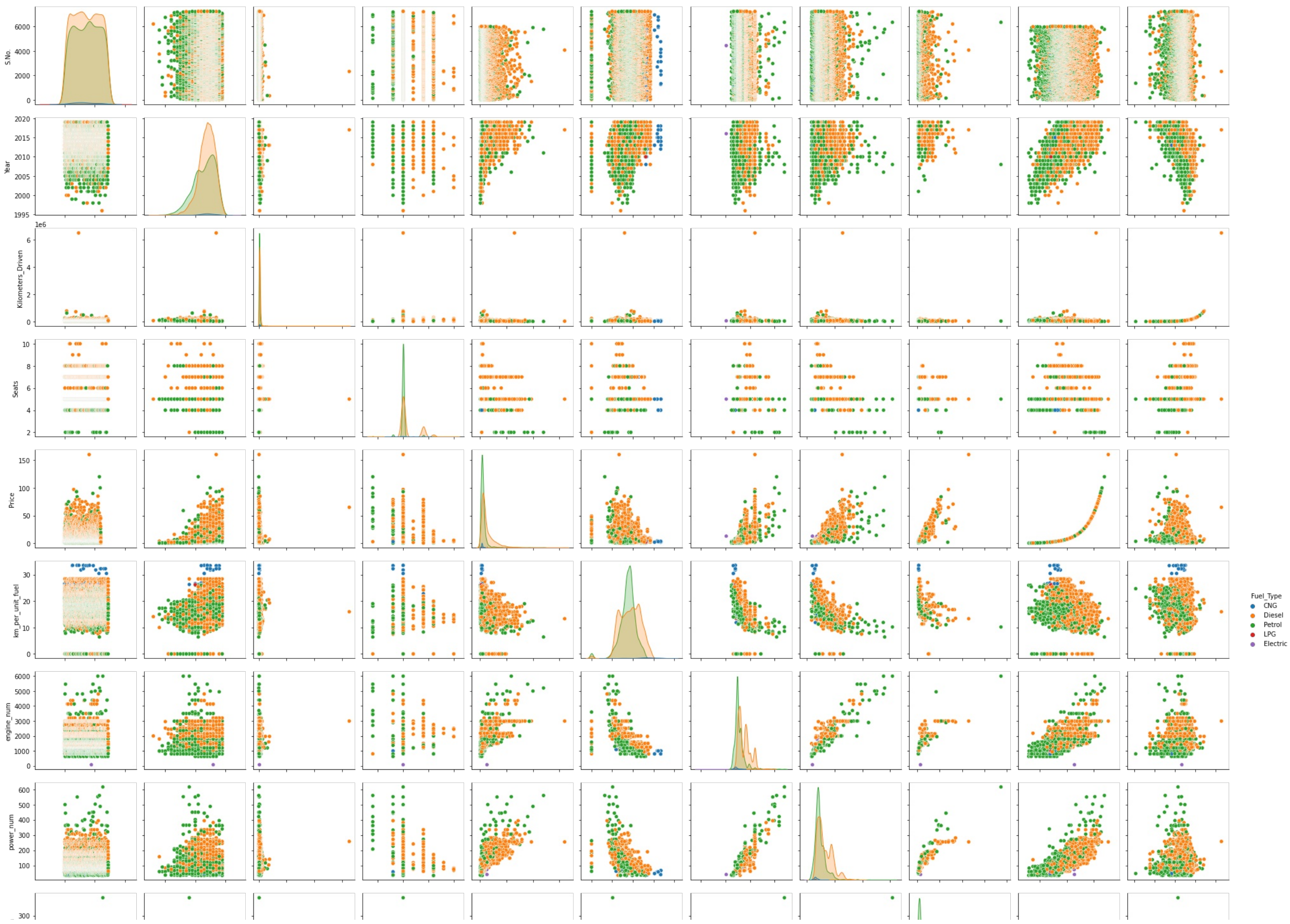
In [38]:
```
df["kilometers_driven_log"] = np.log(df["Kilometers_Driven"])
```

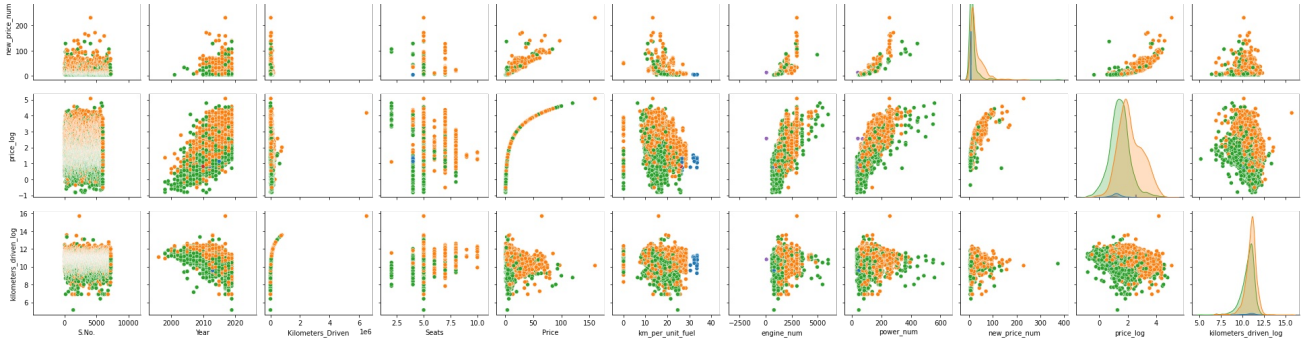
Bivariate Distributions

In [39]:
```
sns.pairplot(data=df, hue="Fuel_Type")
```

```
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
```

Out[39]:
```
<seaborn.axisgrid.PairGrid at 0x7fdee2c37f40>
```





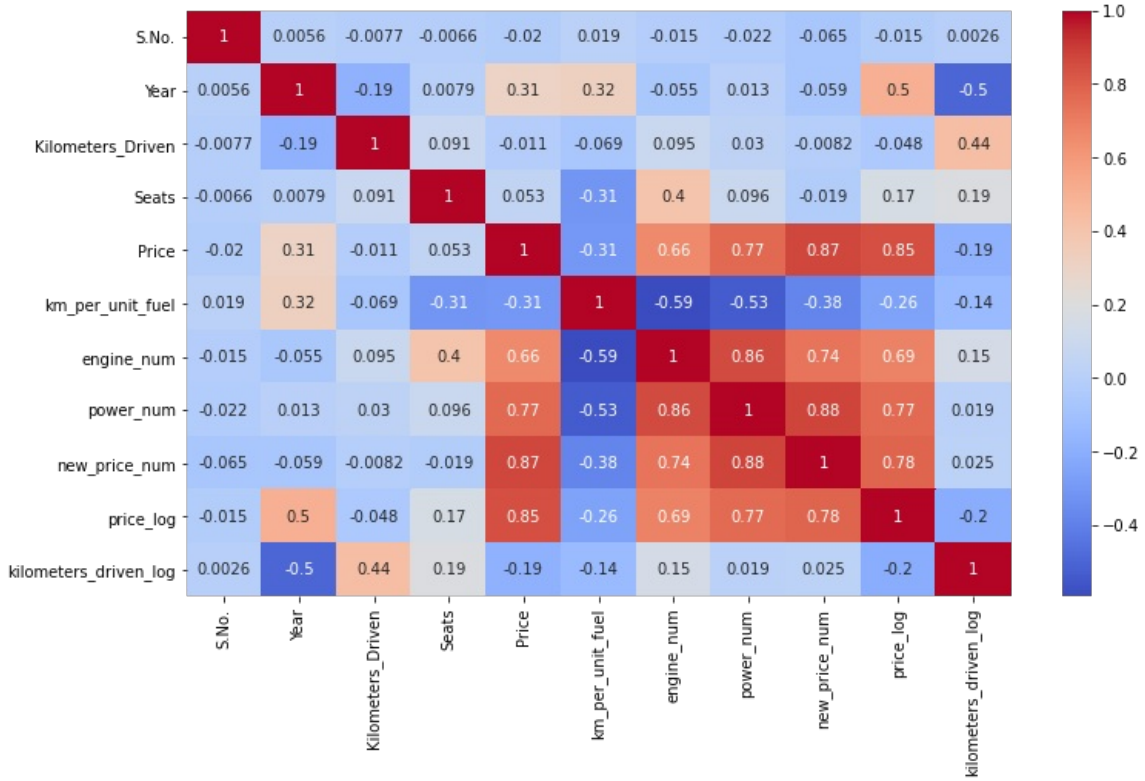
Zooming into these plots gives us a lot of information.

- Contrary to intuition Kilometers Driven does not seem to have a relationship with price.
- Price has a positive relationship with Year. Newer the car, higher the price.
- S.No. does not capture any information that we were hoping for. The temporal element of variation is captured in the year column.
- 2 seater cars are all luxury variants. Cars with 8-10 seats are exclusively mid to high range.
- Mileage does not seem to show much relationship with the price of used cars.
- Engine displacement and Power of the car have a positive relationship with the price.
- New Price and Used Car Price are also positively correlated, which is expected.
- Kilometers Driven has a peculiar relationship with the Year variable. Generally, newer the car lesser the distance it has travelled, but this is not always true.
- CNG cars are conspicuous outliers when it comes to Mileage. The mileage of these cars is very high.
- Mileage and power of newer cars is increasing owing to advancement in technology.
- Mileage has a negative correlation with engine displacement and power. More powerful the engine, more fuel it consumes in general.

Correlation between numeric Variables

In [40]:
```
plt.figure(figsize=(12, 7))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
```

Out[40]: <AxesSubplot:>



- Power and engine are important predictors of price
- We will have to work on imputing New Price missing values because this is a very important feature in predicting used car price accurately

Missing Value Treatment

In [41]:
```
# Missingness check once again
df.isnull().sum()
```

Out[41]:
```
S.No.                    0
Name                     0
Location                 0
Year                     0
Kilometers_Driven        0
Fuel_Type                0
Transmission             0
Owner_Type               0
Seats                   53
Price                 1234
km_per_unit_fuel         2
engine_num              46
power_num              175
new_price_num         6247
Brand                    0
Model                    0
price_log             1234
kilometers_driven_log    0
dtype: int64
```

Seats

In [42]:
```
# Look at a few rows where #seats is missing
df[df["Seats"].isnull()]
```

Out[42]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Seats	Price	km_per_unit_fuel	engine_nu
194	194	Honda City 1.5 GXI	Ahmedabad	2007	60006	Petrol	Manual	First	NaN	2.95	0.00	Na
208	208	Maruti Swift 1.3 VXi	Kolkata	2010	42001	Petrol	Manual	First	NaN	2.11	16.10	Na
229	229	Ford Figo Diesel	Bangalore	2015	70436	Diesel	Manual	First	NaN	3.60	0.00	1498
733	733	Maruti Swift 1.3 VXi	Chennai	2006	97800	Petrol	Manual	Third	NaN	1.75	16.10	Na
749	749	Land Rover Range Rover 3.0 D	Mumbai	2008	55001	Diesel	Automatic	Second	NaN	26.50	0.00	Na
1294	1294	Honda City 1.3 DX	Delhi	2009	55005	Petrol	Manual	First	NaN	3.20	12.80	Na
1327	1327	Maruti Swift 1.3 ZXI	Hyderabad	2015	50295	Petrol	Manual	First	NaN	5.80	16.10	Na
1385	1385	Honda City 1.5 GXI	Pune	2004	115000	Petrol	Manual	Second	NaN	1.50	0.00	Na
1460	1460	Land Rover Range Rover Sport 2005 2012 Sport	Coimbatore	2008	69078	Petrol	Manual	First	NaN	40.88	0.00	Na
1917	1917	Honda City 1.5 EXI	Jaipur	2005	88000	Petrol	Manual	Second	NaN	1.70	13.00	1493
2074	2074	Maruti Swift 1.3 LXI	Pune	2011	24255	Petrol	Manual	First	NaN	3.15	16.10	Na
2096	2096	Hyundai Santro LP zipPlus	Coimbatore	2004	52146	Petrol	Manual	First	NaN	1.93	0.00	Na
2264	2264	Toyota Etios Liva	Pune	2012	24500	Petrol	Manual	Second	NaN	2.95	18.30	Na

		V										
2325	2325	Maruti Swift 1.3 VXI ABS	Pune	2015	67000	Petrol	Manual	First	NaN	4.70	16.10	Na
2335	2335	Maruti Swift 1.3 VXi	Mumbai	2007	55000	Petrol	Manual	Second	NaN	1.75	16.10	Na
2369	2369	Maruti Estilo LXI	Chennai	2008	56000	Petrol	Manual	Second	NaN	1.50	19.50	1061
2530	2530	BMW 5 Series 520d Sedan	Kochi	2014	64158	Diesel	Automatic	First	NaN	17.89	18.48	Na
2542	2542	Hyundai Santro GLS II - Euro II	Bangalore	2011	65000	Petrol	Manual	Second	NaN	3.15	0.00	Na
2623	2623	BMW 5 Series 520d Sedan	Pune	2012	95000	Diesel	Automatic	Second	NaN	18.00	18.48	Na
2668	2668	Maruti Swift 1.3 VXi	Kolkata	2014	32986	Petrol	Manual	First	NaN	4.24	16.10	Na
2737	2737	Maruti Wagon R Vx	Jaipur	2001	200000	Petrol	Manual	First	NaN	0.70	12.00	Na
2780	2780	Hyundai Santro GLS II - Euro II	Pune	2009	100000	Petrol	Manual	First	NaN	1.60	0.00	Na
2842	2842	Hyundai Santro GLS II - Euro II	Bangalore	2012	43000	Petrol	Manual	First	NaN	3.25	0.00	Na
3272	3272	BMW 5 Series 520d Sedan	Mumbai	2008	81000	Diesel	Automatic	Second	NaN	10.50	18.48	Na
3404	3404	Maruti Swift 1.3 VXi	Jaipur	2006	125000	Petrol	Manual	Fourth & Above	NaN	2.35	16.10	Na
3520	3520	BMW 5 Series 520d Sedan	Delhi	2012	90000	Diesel	Automatic	First	NaN	14.50	18.48	Na
3522	3522	Hyundai Santro GLS II - Euro II	Kochi	2012	66400	Petrol	Manual	First	NaN	2.66	0.00	Na
3800	3800	Ford Endeavour Hurricane LE	Mumbai	2012	129000	Diesel	Automatic	First	NaN	7.00	12.80	2953
3810	3810	Honda CR-V AT With Sun Roof	Kolkata	2013	27000	Petrol	Automatic	First	NaN	11.99	14.00	Na
3882	3882	Maruti Estilo LXI	Kolkata	2010	40000	Petrol	Manual	Second	NaN	2.50	19.50	1061
4011	4011	Fiat Punto 1.3 Emotion	Pune	2011	45271	Diesel	Manual	First	NaN	2.60	20.30	Na
4152	4152	Land Rover Range Rover 3.0 D	Mumbai	2003	75000	Diesel	Automatic	Second	NaN	16.11	0.00	Na
4229	4229	Hyundai Santro Xing XG	Bangalore	2005	79000	Petrol	Manual	Second	NaN	1.65	17.00	Na
4577	4577	BMW 5 Series 520d Sedan	Delhi	2012	72000	Diesel	Automatic	Third	NaN	13.85	18.48	Na
4604	4604	Honda Jazz Select Edition	Pune	2011	98000	Petrol	Manual	First	NaN	3.15	16.70	Na
4697	4697	Fiat Punto 1.2 Dynamic	Kochi	2017	17941	Petrol	Manual	First	NaN	3.93	15.70	Na

4712	4712	Hyundai Santro Xing XG	Pune	2003	80000	Petrol	Manual	Second	NaN	0.90	17.00	Na
4952	4952	Fiat Punto 1.4 Emotion	Kolkata	2010	47000	Petrol	Manual	First	NaN	1.49	14.60	Na
5015	5015	Maruti Swift 1.3 VXi	Delhi	2006	63000	Petrol	Manual	First	NaN	1.60	16.10	Na
5185	5185	Maruti Swift 1.3 LXI	Delhi	2012	52000	Petrol	Manual	First	NaN	3.65	16.10	Na
5270	5270	Honda City 1.5 GXI	Bangalore	2002	53000	Petrol	Manual	Second	NaN	1.85	0.00	Na
5893	5893	Maruti Estilo LXI	Chennai	2008	51000	Petrol	Manual	Second	NaN	1.75	19.50	1061
6042	6042	Skoda Laura 1.8 TSI Ambition	Bangalore	2009	72000	Petrol	Manual	Second	NaN	NaN	17.50	Na
6541	6541	Toyota Etios Liva Diesel TRD Sportivo	Bangalore	2012	56600	Diesel	Manual	First	NaN	NaN	23.59	Na
6544	6544	Hyundai i20 new Sportz AT 1.4	Bangalore	2012	58000	Petrol	Automatic	Second	NaN	NaN	15.00	Na
6633	6633	Mahindra TUV 300 P4	Kolkata	2016	27000	Diesel	Manual	First	NaN	NaN	0.00	Na
6643	6643	BMW 5 Series 520d Sedan	Bangalore	2009	150000	Diesel	Automatic	Second	NaN	NaN	18.48	Na
6651	6651	Maruti Swift 1.3 VXi	Kolkata	2015	36009	Petrol	Manual	First	NaN	NaN	16.10	Na
6677	6677	Fiat Punto 1.4 Emotion	Jaipur	2010	65000	Petrol	Manual	Third	NaN	NaN	14.60	Na
6685	6685	Maruti Swift 1.3 VXi	Pune	2010	115000	Petrol	Manual	Second	NaN	NaN	16.10	Na
6880	6880	BMW 5 Series 520d Sedan	Chennai	2009	95000	Diesel	Automatic	Second	NaN	NaN	18.48	Na
6902	6902	Toyota Etios Liva V	Kochi	2012	59311	Petrol	Manual	First	NaN	NaN	18.30	Na
6957	6957	Honda Jazz 2020 Petrol	Kochi	2019	11574	Petrol	Manual	First	NaN	NaN	0.00	1199

In [43]:

```
# We'll impute these missing values one by one, by taking median number of seats for the particular car,
# using the Brand and Model name
df.groupby(["Brand", "Model"], as_index=False)["Seats"].median()
```

Out[43]:

	Brand	Model	Seats
0	ambassador	classic	5.0
1	audi	a3	5.0
2	audi	a4	5.0
3	audi	a6	5.0
4	audi	a7	5.0
...	...	...	...
217	volvo	s60	5.0
218	volvo	s80	5.0
219	volvo	v40	5.0
220	volvo	xc60	5.0

221	volvo	xc90	7.0

222 rows × 3 columns

In [44]:
```
# Impute missing Seats
df["Seats"] = df.groupby(["Brand", "Model"])["Seats"].transform(
    lambda x: x.fillna(x.median())
)
```

In [45]:
```
# Check 'Seats'
df[df["Seats"].isnull()]
```

Out[45]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Seats	Price	km_per_unit_fuel	engine_num	po
2369	2369	Maruti Estilo LXI	Chennai	2008	56000	Petrol	Manual	Second	NaN	1.50	19.5	1061.0	
3882	3882	Maruti Estilo LXI	Kolkata	2010	40000	Petrol	Manual	Second	NaN	2.50	19.5	1061.0	
5893	5893	Maruti Estilo LXI	Chennai	2008	51000	Petrol	Manual	Second	NaN	1.75	19.5	1061.0	

In [46]:
```
# Maruti Estilo can accomodate 5
df["Seats"] = df["Seats"].fillna(5.0)
```

In [47]:
```
# We will use similar methods to fill missing values for engine, power and new price
df["engine_num"] = df.groupby(["Brand", "Model"])["engine_num"].transform(
    lambda x: x.fillna(x.median())
)
df["power_num"] = df.groupby(["Brand", "Model"])["power_num"].transform(
    lambda x: x.fillna(x.median())
)
df["new_price_num"] = df.groupby(["Brand", "Model"])["new_price_num"].transform(
    lambda x: x.fillna(x.median())
)
```

In [48]:
```
df.isnull().sum()
```

Out[48]:
```
S.No.                  0
Name                   0
Location               0
Year                   0
Kilometers_Driven      0
Fuel_Type              0
Transmission           0
Owner_Type             0
Seats                  0
Price               1234
km_per_unit_fuel       2
engine_num             0
power_num             12
new_price_num       1512
Brand                  0
Model                  0
price_log           1234
kilometers_driven_log  0
dtype: int64
```

In [49]:
```
# There are still some missing values in power, mileage and new_price_num.
# There are a few car brands and models in our dataset that do not contain the new price information at all.
# Now we'll have to estimate the new price using median of the data.

cols1 = ["power_num","km_per_unit_fuel","new_price_num"]

for ii in cols1:
    df[ii] = df[ii].fillna(df[ii].median())
```

In [50]:
```
df.isnull().sum()
```

Out[50]:
```
S.No.                      0
Name                       0
Location                   0
Year                       0
Kilometers_Driven          0
Fuel_Type                  0
Transmission               0
Owner_Type                 0
Seats                      0
Price                   1234
km_per_unit_fuel           0
engine_num                 0
power_num                  0
new_price_num              0
Brand                      0
Model                      0
price_log               1234
kilometers_driven_log      0
dtype: int64
```

In [51]:
```
# Drop the redundant columns.
df.drop(columns=["Kilometers_Driven", "Name", "S.No."], inplace=True)

# Drop the rows where 'Price' == NaN and proceed to modelling
df = df[df["Price"].notna()]
```

Linear Model Building

1. What we want to predict is the "Price". We will use the normalised version 'price_log' for modelling.
2. Before we proceed to modelling, we'll have to encode categorical features. We will drop categorical features like - Name
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 a Linear Regression model using the train data.
5. Evaluate the model performance on test data.

Define dependent variable

In [52]:
```
ind_vars = df.drop(["Price", "price_log"], axis=1)
dep_var = df[["price_log"]]   #we will be using the logarithm of prices as our target variable
```

Creating dummy variables

In [53]:
```
def encode_cat_vars(x):
    x = pd.get_dummies(
        x,
        columns=x.select_dtypes(include=["object", "category"]).columns.tolist(),
        drop_first=True,
    )
    return x


ind_vars_num = encode_cat_vars(ind_vars)
ind_vars_num.head()
```

Out[53]:

	Year	Seats	km_per_unit_fuel	engine_num	power_num	new_price_num	kilometers_driven_log	Location_Bangalore	Location_Chennai	Loca
0	2010	5.0	26.60	998.0	58.16	5.51	11.184421	0	0	
1	2015	5.0	19.67	1582.0	126.20	16.06	10.621327	0	0	
2	2011	5.0	18.20	1199.0	88.70	8.61	10.736397	0	1	
3	2012	7.0	20.77	1248.0	88.76	11.27	11.373663	0	1	
4	2013	5.0	15.20	1968.0	140.80	53.14	10.613246	0	0	

Split the data into train and test

In [54]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
```

```
    ind_vars_num, dep_var, test_size=0.3, random_state=1
)
```

In [55]:
```
print("Number of rows in train data =", x_train.shape[0])
print("Number of rows in test data =", x_test.shape[0])
```

```
Number of rows in train data = 4213
Number of rows in test data = 1806
```

Fitting a linear model

In [56]:
```
lin_reg_model = LinearRegression()
lin_reg_model.fit(x_train,y_train)
```

Out[56]:
```
LinearRegression()
```

In [57]:
```
# let us check the coefficients and intercept of the model

coef_df = pd.DataFrame(np.append(lin_reg_model.coef_.flatten(), lin_reg_model.intercept_), \
                       index=x_train.columns.tolist()+['Intercept'], columns=['Coefficients'])
coef_df
```

Out[57]:

	Coefficients
Year	0.106357
Seats	0.003755
km_per_unit_fuel	-0.000140
engine_num	-0.000124
power_num	0.003030
...	...
Model_yeti	0.298165
Model_z4	0.809548
Model_zen	-0.337210
Model_zest	-0.532924
Intercept	-211.343520

265 rows × 1 columns

In [58]:
```
coef_df.T
```

Out[58]:

	Year	Seats	km_per_unit_fuel	engine_num	power_num	new_price_num	kilometers_driven_log	Location_Bangalore	Locati
Coefficients	0.106357	0.003755	-0.00014	-0.000124	0.00303	0.007317	-0.075502	0.174548	

Let us check the model performance on training data.

In [59]:
```
# MAPE
def mape(targets, predictions):
    return np.mean(np.abs((targets - predictions)) / targets) * 100

# Adjusted R^2
def adj_r2(ind_vars, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = ind_vars.shape[0]
    k = ind_vars.shape[1]
    return 1-((1-r2)*(n-1)/(n-k-1))

# Model performance check
def model_perf(model, inp, out):

    y_pred = np.exp(model.predict(inp))
    y_act = np.exp(out.values)
```

```
    return pd.DataFrame({
                "RMSE": np.sqrt(mean_squared_error(y_act, y_pred)),
                "MAE": mean_absolute_error(y_act, y_pred),
                "MAPE": mape(y_act, y_pred),
                "R^2": r2_score(y_act, y_pred),
                "Adjusted R^2": adj_r2(inp, y_act, y_pred)
            }, index=[0])
```

In [60]:
```
# Checking model performance on train set
print('Training Performance\n')
print(model_perf(lin_reg_model, x_train, y_train))
```

```
Training Performance

       RMSE       MAE       MAPE       R^2  Adjusted R^2
0  2.726211  1.196561  12.983262  0.941875      0.937989
```

- Both the R-squared and Adjusted R squared of our model are very high. This is a clear indication that we have been able to create a very good model that is able to explain variance in price of used cars for upto 94%.
- The model is not an underfitting model.
- Let us do a quick performance check on the test data.

In [61]:
```
# Checking model performance on test set
print('Test Performance\n')
print(model_perf(lin_reg_model, x_test, y_test))
```

```
Test Performance

       RMSE       MAE       MAPE       R^2  Adjusted R^2
0  3.397028  1.360523  13.039766  0.902826      0.886179
```

- RMSE of train and test data is starkly different, indicating that our model is overfitting the train data.
- Mean Absolute Error indicates that our current model is able to predict used cars prices within mean error of 1.4 lakhs on test data.
- The units of both RMSE and MAE are same - Lakhs in this case. But RMSE is greater than MAE because it peanalises the outliers more.
- Mean Absolute Percentage Error is ~13% on the test data.

In [62]:
```
df.columns
```

Out[62]:
```
Index(['Location', 'Year', 'Fuel_Type', 'Transmission', 'Owner_Type', 'Seats',
       'Price', 'km_per_unit_fuel', 'engine_num', 'power_num', 'new_price_num',
       'Brand', 'Model', 'price_log', 'kilometers_driven_log'],
      dtype='object')
```

Our current model is extremely complex. Let us first bin the Brand and Model columns as it will help us make the dataset more managable.

In [63]:
```
df.groupby(["Brand", "Model"])["new_price_num"].mean().sort_values(ascending=False)
```

Out[63]:
```
Brand          Model
mercedes-benz  s-class    171.000000
bmw            7          157.230769
porsche        cayenne    136.000000
audi           rs5        128.000000
land           rover      119.887000
                             ...
maruti         eeco         4.900000
renault        kwid         4.794750
hyundai        santro       4.550000
maruti         alto         4.366364
datsun         redi-go      4.153333
Name: new_price_num, Length: 215, dtype: float64
```

We will create a new variable Car Category by binning the new_price_num

In [64]:
```
# Create a new variable - Car Category
```

```
df1 = df.copy()
df1["car_category"] = pd.cut(
    x=df["new_price_num"],
    bins=[0, 15, 30, 50, 200],
    labels=["Budget_Friendly", "Mid-Range", "Luxury_Cars", "Ultra_luxury"],
)
df1["car_category"].value_counts()
```

Out[64]:
```
Budget_Friendly    4292
Mid-Range           653
Ultra_luxury        567
Luxury_Cars         506
Name: car_category, dtype: int64
```

In [65]:
```
# Drop the Brand and Model and new_price_num columns.
df1.drop(columns=["Brand", "Model", "new_price_num"], axis=1, inplace=True)

# We will have to create the x and y datasets again
ind_vars = df1.drop(["Price", "price_log"], axis=1)
dep_var = df1[["price_log"]]

# Dummy encoding
ind_vars_num = encode_cat_vars(ind_vars)

# Splitting data into train and test
x_train2, x_test2, y_train, y_test = train_test_split(
    ind_vars_num, dep_var, test_size=0.3, random_state=1
)

print("Number of rows in train data =", x_train2.shape[0])
print("Number of rows in test data =", x_test2.shape[0])
```

```
Number of rows in train data = 4213
Number of rows in test data = 1806
```

In [66]:
```
# fitting linear regression model
# lin_reg_model2 = LinearRegression()
# lin_reg_model2.fit(x_train2,y_train)

# let us check the coefficients and intercept of the model

coef_df = pd.DataFrame(np.append(lin_reg_model2.coef_.flatten(), lin_reg_model2.intercept_), \
                       index=x_train2.columns.tolist()+['Intercept'], columns=['Coefficients'])
coef_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-66-b3ba9e407d01> in <module>
      5 # let us check the coefficients and intercept of the model
      6
----> 7 coef_df = pd.DataFrame(np.append(lin_reg_model2.coef_.flatten(), lin_reg_model2.intercept_), \
      8                        index=x_train2.columns.tolist()+['Intercept'], columns=['Coefficients'])
      9 coef_df

NameError: name 'lin_reg_model2' is not defined
```

In [ ]:
```
coef_df.T
```

In [ ]:
```
lin_reg_model2 = LinearRegression()
lin_reg_model2.fit(x_train2,y_train)
```

In [ ]:
```
# Checking model performance on train set
print('Training Performance\n')
print(model_perf(lin_reg_model2, x_train2, y_train))

# Checking model performance on test set
print('\n\nTest Performance\n')
print(model_perf(lin_reg_model2, x_test2, y_test))
```

- Binning the car brands and models has drastically reduced the model performance.
- Let us try using forward feature selection to see if we can improve the model performance.

Forward Feature Selection

In [ ]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs

reg = LinearRegression()

# Build step forward feature selection
sfs1 = sfs(reg,k_features = x_train2.shape[1], forward=True,   # k_features denotes "Number of features to selec
           floating=False, scoring= 'r2',
           verbose=2, cv=5)

# Perform SFFS
sfs1 = sfs1.fit(x_train2, y_train)
```

We can see that R square starts decreasing after addition of 21st feature, so we will proceed only with best 20 features

- Now we'll change k_features to 20.

In [ ]:
```
reg = LinearRegression()

# # Build step forward feature selection
sfs1 = sfs(reg, k_features = 20, forward=True,
           floating=False, scoring='r2',
           verbose=2, cv=5)

# Perform SFFS
sfs1 = sfs1.fit(x_train2, y_train)
```

In [ ]:
```
# Now Which features are important?
feat_cols = list(sfs1.k_feature_idx_)
print(feat_cols)
```

In [ ]:
```
x_train2.columns[feat_cols]
```

Now we will fit a sklearn model using these features only.

In [ ]:
```
x_train3 = x_train2[x_train2.columns[feat_cols]]
```

In [ ]:
```
x_test2.columns
```

In [ ]:
```
#Creating new x_test with the same 20 variables that we selected for x_train
x_test3 = x_test2[x_train3.columns]
```

In [ ]:
```
#Fitting linear model
lin_reg_model3 = LinearRegression()
lin_reg_model3.fit(x_train3,y_train)

# let us check the coefficients and intercept of the model

coef_df = pd.DataFrame(np.append(lin_reg_model3.coef_.flatten(), lin_reg_model3.intercept_.flatten()), \
                       index=x_train3.columns.tolist()+['Intercept'], columns=['Coefficients'])
print(coef_df)

# model performance on train set
print('\n\nTraining Performance\n')
print(model_perf(lin_reg_model3, x_train3, y_train))

# model performance on test set
print('\n\nTest Performance\n')
print(model_perf(lin_reg_model3, x_test3, y_test))
```

- Even with forward feature selection we did not achieve good performance.
- We will now try forward feature selection with the data before binning the car brands and models.

In [ ]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs

reg = LinearRegression()

n_features = list(range(5,55,5))

for item in n_features:
    # Build step forward feature selection
```

```
    sfs1 = sfs(reg, k_features = item, forward=True,   # k_features denotes "Number of features to select"
              floating=False, scoring= 'r2',
              verbose=0, n_jobs=-2, cv=5)

    # Perform SFFS
    sfs1 = sfs1.fit(x_train, y_train)
    print('R^2:', sfs1.k_score_, '\nFeatures used:', item, sfs1.k_feature_names_, '\n')
```

We see that after 35 best features, adding more features does not result in a significant increase in performance. So we will move forward with 35 best features, and make a linear regression model using only these features.

In [ ]:

```
# Build step forward feature selection
sfs1 = sfs(reg, k_features = 50, forward=True,   # k_features denotes "Number of features to select"
           floating=False, scoring= 'r2',
           verbose=0, n_jobs=-2, cv=5)

# Perform SFFS
sfs1 = sfs1.fit(x_train, y_train)
feat_cols = list(sfs1.k_feature_names_)

# take the best columns as per forward feature selection
X_train_final = x_train[feat_cols]
X_test_final = x_test[feat_cols]

print(X_train_final.shape, X_test_final.shape)
X_train_final.head()
```

In [ ]:

```
#Fitting linear model
lin_reg_model4 = LinearRegression()
lin_reg_model4.fit(X_train_final,y_train)

# let us check the coefficients and intercept of the model

coef_df = pd.DataFrame(np.append(lin_reg_model4.coef_.flatten(), lin_reg_model4.intercept_.flatten()), \
                       index=X_train_final.columns.tolist()+['Intercept'], columns=['Coefficients'])
print(coef_df)

# model performance on train set
print('\n\nTraining Performance\n')
print(model_perf(lin_reg_model4, X_train_final, y_train))

# model performance on test set
print('\n\nTest Performance\n')
print(model_perf(lin_reg_model4, X_test_final, y_test))
```

Observations and Conclusions

1. With our linear regression model we have been able to capture ~85% variation in our data.
2. The model indicates that the most significant predictors of price of used cars are -
    - The year of manufacturing
    - Price of new car
    - Power of the engine
    - Mileage
    - Kilometers Driven
    - Location
    - Fuel_Type
    - Transmission - Automatic/Manual
    - Car Brand
    - Car Model
1. Newer cars sell for higher prices. One unit increase in the year of manufacture leads to exp(0.1198) = 1.13 Lakhs increase in the price of the vehicle, when everything else is constant.

It is important to note here that the predicted values are log(price) and therefore coefficients have to converted accordingly to understand that influence in Price.

1. As the price of a new car of same model increases by one unit, the price of the used car increases by exp(0.0029) = 1.00 Lakhs
2. Mileage is inversely correlated with Price. Generally, high Mileage cars are the lower budget cars.

It is important to note here that correlation is not equal to causation. That is to say that increase in Mileage does not lead to a drop in prices. It can be understood in such a way that the cars with high mileage do not have a high power engine and therefore have low prices.

1. Kilometers Driven have a negative relationship with the price which is intuitive. A car that has been driven more will have more wear and tear and hence sell at a lower price.

2. The categorical variables are a little hard to interpret. But it can be seen that most of the Brand and Model variables in the dataset have a positive relationship with the Price and the magnitude of this positive relationship increases as the brand category moves to the luxury brands.

Business Insights and Recommendations

- Our final Linear Regression model has a MAPE of ~18% on The test data, which means that we are able to predict within 18% of the price value. This is a very good model and we can use this model in production.
- Some southern markets tend to have higher prices. It might be a good strategy to plan growth in southern cities using this information. Markets like Kolkata (coeff ~ -0.18) are very risky and we need to be careful about investments in this area.
- We will have to analyse the cost side of things before we can talk about profitability in the business. We should gather data regarding that.
- The next step post that would be to cluster different sets of data and see if we should make multiple models for different locations/car types.

Analysing predictions where we were way off the mark

In [ ]:
```
# Extracting the rows from original data frame df where indexes are same as the training data
original_df = df[df.index.isin(X_train_final.index.values)].copy()

# Extracting predicted values and residuals from the final model
fitted_values = lin_reg_model4.predict(X_train_final)
residuals = fitted_values - y_train

# Add new columns for predicted values
original_df["Predicted price_log "] = fitted_values
original_df["Predicted Price"] = np.exp(fitted_values)
original_df["residuals"] = residuals
original_df["Abs_residuals"] = np.exp(residuals)
original_df["Difference in Lakhs"] = np.abs(
    original_df["Price"] - original_df["Predicted Price"]
)

# Let us look at the top 50 predictions where our model made highest extimation errors (on train data)
original_df.sort_values(by=["Difference in Lakhs"], ascending=False).head(50)
```

- A 2017 Land Rover, whose new model sells at 230 Lakhs and the used version sold at 160 Lakhs was predicted to be sold at < 3 Lakhs. It is not apparent after looking at numerical predictors, why our model predicted such low value here. This could be because many other land rovers in our data seems to have sold at lower prices.
- Another entry in the list here is a Lamborghini Gallardo that was sold at 120 Lakhs but our model predicted the price as 3.07 Lakhs. This is a huge error by the model. However, there might be a data entry error here as the price of a new Gallardo is set at 11.685 Lakhs, which is less than the selling price of a used Gallardo.
- There are a few instances where the model predicts lesser than the actual selling price. These could be a cause for concern. The model predicting lesser than potential selling price is not good for business.

Let us quickly visualise some of these observations.

In [ ]:
```
sns.scatterplot(
    data = original_df,
    x="Difference in Lakhs",
    y = "Price",
    hue=original_df["Fuel_Type"],
)
```

Our model predicts that resale value of diesel cars is higher compared to petrol cars.