NITHINRAJU

used-car-prices-linearregression

May 9, 2023

0.1 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. We 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.

Table of Contents

0.2 Data Set

1. S.No. : Serial Number
2. Name : Name of the car which includes Brand name and Model name
3. Location : The location in which the car is being sold or is available for purchase Cities
4. Year : Manufacturing year of the car
5. Kilometers_driven : The total kilometers driven in the car by the previous owner(s) in KM.
6. Fuel_Type : The type of fuel used by the car. (Petrol, Diesel, Electric, CNG, LPG)
7. Transmission : The type of transmission used by the car. (Automatic / Manual)
8. Owner : Type of ownership
9. Mileage : The standard mileage offered by the car company in kmpl or km/kg
10. Engine : The displacement volume of the engine in CC.
11. Power : The maximum power of the engine in bhp.
12. Seats : The number of seats in the car.
13. New_Price : The price of a new car of the same model in INR Lakhs.(1 Lakh = 100, 000)
14. Price : The price of the used car in INR Lakhs (1 Lakh = 100, 000)

0.3 Problem

- Does various predicating factors effect the price of the used car .?
- What all independent variables effect the pricing of used cars?
- Does name of a car have any effect on pricing of car.?
- How does type of Transmission effect pricing?
- Does Location in which the car being sold has any effect on the price?
- Does kilometers_Driven,Year of manufacturing have negative correlation with price of the car?
- Does Mileage ,Engine and Power have any effect on the pricing of the car?
- How does number of seat ,Fuel type effect the pricing.?

1 Libraries

[1]:
```
### IMPORT: ..........................................................................
import scipy.stats as stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore') # To supress warnings
 # set the background for the graphs
from scipy.stats import skew
plt.style.use('ggplot')
import missingno as msno # to get visualization on missing values
from sklearn.model_selection import train_test_split # Sklearn package's␣
 ↪randomized data splitting function
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import math
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.max_rows', 300)
pd.set_option('display.max_colwidth',400) pd.set_option('display.float_format',
lambda x: '%.5f' % x) # To supress␣
 ↪numerical display in scientific notations
import statsmodels.api as sm
print("Load Libraries- Done")
```

Load Libraries- Done

2 Read and Understand data

[2]:
```
#Reading the csv file   used car data.csv
data_path='../input/cars4u/used_cars_data.csv'
df=pd.read_csv(data_path,index_col=0)
cars=df.copy()
print(f'There are {cars.shape[0]} rows and {cars.shape[1]} columns') # fstring
```

There are 7253 rows and 13 columns

[3]:
```
# inspect data, print top 5
cars.head(5)
```

[3]:
```
                                                Name    Location  Year  Kilometers_Driven  \
S.No.
```

```
0                 Maruti Wagon R LXI CNG      Mumbai  2010             72000
1       Hyundai Creta 1.6 CRDi SX Option        Pune  2015             41000
2                           Honda Jazz V     Chennai  2011             46000
3                      Maruti Ertiga VDI     Chennai  2012             87000
4        Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013             40670

     Fuel_Type Transmission Owner_Type     Mileage   Engine      Power  \
S.No.
0          CNG       Manual      First  26.6 km/kg   998 CC  58.16 bhp
1       Diesel       Manual      First  19.67 kmpl  1582 CC  126.2 bhp
2       Petrol       Manual      First   18.2 kmpl  1199 CC   88.7 bhp
3       Diesel       Manual      First  20.77 kmpl  1248 CC  88.76 bhp
4       Diesel    Automatic     Second   15.2 kmpl  1968 CC  140.8 bhp

        Seats  New_Price    Price
S.No.
0     5.00000        NaN  1.75000
1     5.00000        NaN 12.50000
2     5.00000  8.61 Lakh  4.50000
3     7.00000        NaN  6.00000
4     5.00000        NaN 17.74000
```

[4]:
```python
# bottom 5 rows:
cars.tail(5)
```

[4]:
```
                                                Name   Location  Year  \
S.No.
7248                 Volkswagen Vento Diesel Trendline  Hyderabad  2011
7249                            Volkswagen Polo GT TSI     Mumbai  2015
7250                              Nissan Micra Diesel XV    Kolkata  2012
7251                            Volkswagen Polo GT TSI       Pune  2013
7252   Mercedes-Benz E-Class 2009-2013 E 220 CDI Avantgarde    Kochi  2014

      Kilometers_Driven Fuel_Type Transmission Owner_Type     Mileage  \
S.No.
7248              89411    Diesel       Manual      First  20.54 kmpl
7249              59000    Petrol    Automatic      First  17.21 kmpl
7250              28000    Diesel       Manual      First  23.08 kmpl
7251              52262    Petrol    Automatic      Third   17.2 kmpl
7252              72443    Diesel    Automatic      First   10.0 kmpl

       Engine      Power   Seats New_Price  Price
S.No.
7248  1598 CC  103.6 bhp 5.00000       NaN    NaN
7249  1197 CC  103.6 bhp 5.00000       NaN    NaN
7250  1461 CC   63.1 bhp 5.00000       NaN    NaN
7251  1197 CC  103.6 bhp 5.00000       NaN    NaN
```

```
7252    2148 CC      170 bhp 5.00000         NaN     NaN
```

[5]:
```
#get the size of dataframe
print ("Rows       : " , cars.shape[0])  #get number of rows/observations
print ("Columns  : " , cars.shape[1]) #get number of columns
print ("#"*40,"\n","Features : \n\n", cars.columns.tolist()) #get name of␣
↪columns/features
print ("#"*40,"\nMissing values :\n\n", cars.isnull().sum().
↪sort_values(ascending=False))
print( "#"*40,"\nPercent of missing :\n\n", round(cars.isna().sum() / cars.
↪isna().count() * 100, 2)) # looking at columns with most Missing Values
print ("#"*40,"\nUnique values :  \n\n", cars.nunique())  #  count of unique␣
↪values
```

```
Rows       :   7253
Columns  :   13
########################################
 Features :

 ['Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type', 'Transmission',
'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats', 'New_Price', 'Price']
########################################
Missing values :

 New_Price            6247
Price                1234
Seats                  53
Engine                 46
Power                  46
Mileage                 2
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
dtype: int64
########################################
Percent of missing :

 Name                 0.00000
Location             0.00000
Year                 0.00000
Kilometers_Driven    0.00000
Fuel_Type            0.00000
Transmission         0.00000
```

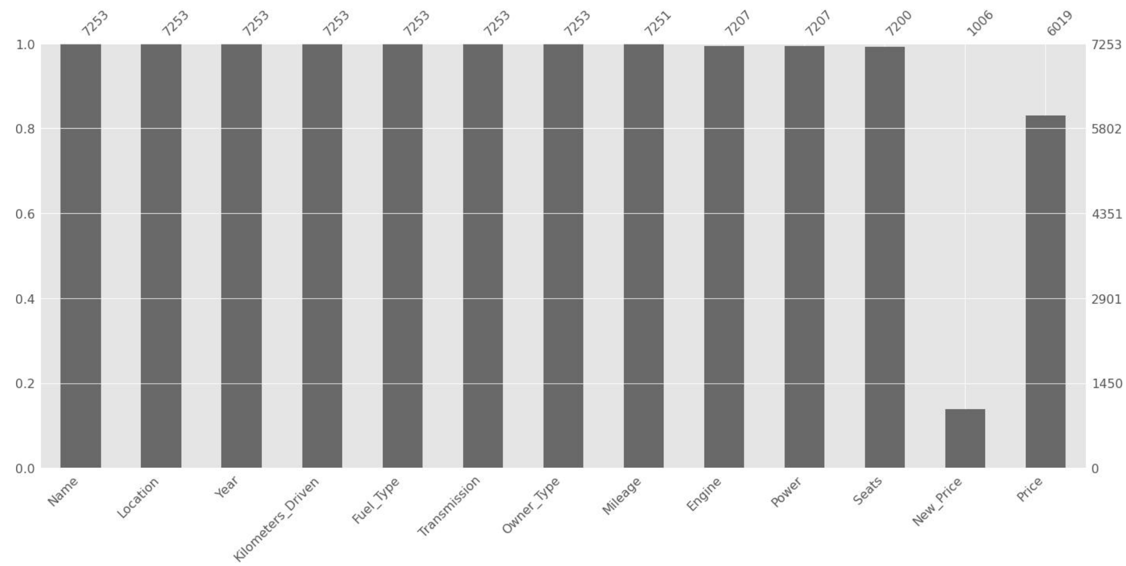
```
Owner_Type          0.00000
Mileage             0.03000
Engine              0.63000
Power               0.63000
Seats               0.73000
New_Price          86.13000
Price              17.01000
dtype: float64
#######################################
Unique values :

 Name                2041
Location               11
Year                   23
Kilometers_Driven    3660
Fuel_Type               5
Transmission            2
Owner_Type              4
Mileage               450
Engine                150
Power                 386
Seats                   9
New_Price             625
Price                1373
dtype: int64
```

[6]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7253 entries, 0 to 7252
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  ------
 0   Name               7253 non-null   object
 1   Location           7253 non-null   object
 2   Year               7253 non-null   int64
 3   Kilometers_Driven  7253 non-null   int64
 4   Fuel_Type          7253 non-null   object
 5   Transmission       7253 non-null   object
 6   Owner_Type         7253 non-null   object
 7   Mileage            7251 non-null   object
 8   Engine             7207 non-null   object
 9   Power              7207 non-null   object
 10  Seats              7200 non-null   float64
 11  New_Price          1006 non-null   object
 12  Price              6019 non-null   float64
dtypes: float64(2), int64(2), object(9)
memory usage: 793.3+ KB
```

[7]:
```
#Visualize missing values
msno.bar(cars)
```

[7]: <AxesSubplot:>



Observations

This preview shows that some columns potentially have a lot of missingness so we'll want to make sure to look into that later.

- **New_Price** has only 1006 values. 86 % values are missing
- **Price**, which is a Target variable 17 % missing values.This needs to be analysed further.
- **Seats** has only 53 values missing and number of seats can be one of key factor in deciding price.
- **Power** and **Engine** has 46 missing values.
- **Mileage** only has two values missing.
- **Mileage,Power,Engine,New_Price** we know are quantitative variables but are of object dtype here and needs to to converted to numeric.

[8]:
```
# Making a list of all categorical variables
cat_col = [
    "Fuel_Type",
    "Location",
    "Transmission",
    "Seats",
    "Year",
    "Owner_Type",
```

```
]
# Printing number of count of each unique value in each column
for column in cat_col:
    print(cars[column].value_counts())
    print("#" * 40)
```

```
Diesel      3852
Petrol      3325
CNG           62
LPG           12
Electric       2
Name: Fuel_Type, dtype: int64
########################################
Mumbai        949
Hyderabad     876
Kochi         772
Coimbatore    772
Pune          765
Delhi         660
Kolkata       654
Chennai       591
Jaipur        499
Bangalore     440
Ahmedabad     275
Name: Location, dtype: int64
########################################
Manual       5204
Automatic    2049
Name: Transmission, dtype: int64
########################################
5.00000     6047
7.00000      796
8.00000      170
4.00000      119
6.00000       38
2.00000       18
10.00000       8
9.00000        3
0.00000        1
Name: Seats, dtype: int64
########################################
2015    929
2014    925
2016    886
2013    791
2017    709
```

```
2012    690
2011    579
2010    407
2018    361
2009    252
2008    207
2007    148
2019    119
2006     89
2005     68
2004     35
2003     20
2002     18
2001      8
2000      5
1998      4
1999      2
1996      1
Name: Year, dtype: int64
########################################
First                5952
Second               1152
Third                 137
Fourth & Above         12
Name: Owner_Type, dtype: int64
########################################
```

Observations

- Maximum car being sold have fuel type as Diesel.
- Mumbai has highest numbers of car availabe for purchase.
- 5204 cars with Manual transmission are available for purchase.
- Most of the cars are 5 seaters and First owned.
- Years of car ranges form 1996- 2015

3 Data Preprocessing

3.0.1 Processing Engine,Power ,Mileage columns

Datatype for Engine ,Power and Mileage are object because of unit assigned ,so striping units.

[9]:
```
#np.random.seed(9)
cars[['Engine','Power','Mileage']].sample(10)
```

[9]:
```
           Engine      Power     Mileage
S.No.
6283      1248 CC     74 bhp   25.2 kmpl
1448      1591 CC  121.3 bhp  17.01 kmpl
1314      1995 CC    218 bhp  16.73 kmpl
```

```
4353   2494 CC   100.6 bhp  12.99 kmpl
2705   1086 CC    62.1 bhp  17.92 kmpl
1403   1199 CC    73.9 bhp  22.07 kmpl
2547   2993 CC  308.43 bhp  15.87 kmpl
5481   1991 CC   147.9 bhp   18.1 kmpl
6747   1461 CC   63.12 bhp  23.08 kmpl
4312   1586 CC  104.68 bhp   15.6 kmpl
```

[10]:
```
typeoffuel=["CNG","LPG"]
cars.loc[cars.Fuel_Type.isin(typeoffuel)].head(10)
```

[10]:
```
                                   Name   Location  Year  \
S.No.
0                 Maruti Wagon R LXI CNG     Mumbai  2010
5         Hyundai EON LPG Era Plus Option  Hyderabad  2012
127               Maruti Wagon R LXI CNG       Pune  2013
328      Maruti Zen Estilo LXI Green (CNG)       Pune  2008
440   Maruti Eeco 5 STR With AC Plus HTR CNG      Kochi  2017
839           Maruti Alto Green LXi (CNG)      Delhi  2012
893          Hyundai Accent Executive CNG  Hyderabad  2010
936            Maruti Wagon R LXI LPG BSIV  Hyderabad  2012
987           Maruti Wagon R LXI DUO BSIII     Mumbai  2008
1135     Maruti Zen Estilo LXI Green (CNG)  Ahmedabad  2011

      Kilometers_Driven Fuel_Type Transmission Owner_Type     Mileage  \
S.No.
0                 72000       CNG       Manual      First  26.6 km/kg
5                 75000       LPG       Manual      First  21.1 km/kg
127               89900       CNG       Manual      First  26.6 km/kg
328               42496       CNG       Manual      First  26.3 km/kg
440               31841       CNG       Manual      First  15.1 km/kg
839               65537       CNG       Manual      First 26.83 km/kg
893               95637       CNG       Manual     Second  13.2 km/kg
936               72000       LPG       Manual      First  26.2 km/kg
987               64226       LPG       Manual      First  17.3 km/kg
1135              76000       CNG       Manual      First  26.3 km/kg

       Engine      Power   Seats  New_Price    Price
S.No.
0      998 CC  58.16 bhp 5.00000        NaN  1.75000
5      814 CC   55.2 bhp 5.00000        NaN  2.35000
127    998 CC  58.16 bhp 5.00000        NaN  3.25000
328    998 CC   67.1 bhp 5.00000        NaN  1.40000
440   1196 CC     73 bhp 5.00000        NaN  4.70000
839    796 CC   38.4 bhp 5.00000        NaN  2.10000
893   1495 CC   93.7 bhp 5.00000        NaN  1.90000
936    998 CC   58.2 bhp 5.00000        NaN  2.85000
```

```
987     1061 CC    57.5  bhp  5.00000         NaN  1.45000
1135     998 CC    67.1  bhp  5.00000         NaN  2.00000
```

Power has some values as "nullbhp" .Mileage also has some observations as 0. For fuel type and CNG and LPG mileage is measured in km/kg where as for other type it is measured in kmpl. Since those units are in km for both of them no need of conversion . Dropping units from mileages,Engine and Power.

3.0.2 Mileage

[11]:
```python
cars[cars.Mileage.isnull()==True]
```

[11]:
```
                               Name Location  Year  Kilometers_Driven Fuel_Type  \
S.No.
4446        Mahindra E Verito D4  Chennai  2016              50000  Electric
4904   Toyota Prius 2009-2016 Z4   Mumbai  2011              44000  Electric

      Transmission Owner_Type Mileage   Engine   Power   Seats   New_Price  \
S.No.
4446     Automatic      First     NaN    72 CC  41 bhp 5.00000  13.58 Lakh
4904     Automatic      First     NaN  1798 CC  73 bhp 5.00000         NaN

          Price
S.No.
4446   13.00000
4904   12.75000
```

[12]:
```python
cars["Mileage"] = cars["Mileage"].str.rstrip(" kmpl")
cars["Mileage"] = cars["Mileage"].str.rstrip(" km/g")
```

3.0.3 Engine

[13]:
```python
#remove units
cars["Engine"] = cars["Engine"].str.rstrip(" CC")
```

3.0.4 Power

[14]:
```python
#remove bhp and replace null with nan
cars["Power"] = cars["Power"].str.rstrip(" bhp")
cars["Power"]= cars["Power"].replace(regex="null", value = np.nan)
```

[15]:
```python
#verify the data
num=['Engine','Power','Mileage']
cars[num].sample(20)
```

[15]:

S.No.	Engine	Power	Mileage
2146	1598	103.6	20.54
3505	1172	67	15.7
3016	2179	138.03	13.93
515	1461	108.45	19.01
2014	999	75	18.78
5825	1498	103.2	21.21
2683	998	67.04	23.1
6214	796	47.3	24.7
5912	2179	140	15.1
2491	796	35	14.0
4398	1396	69	19.09
3769	1461	108.45	20.37
3262	1248	74	23.4
1258	2143	201.1	13.0
3240	1197	85.8	18.6
6742	1968	143	20.38
5884	1248	74	23.4
1554	1197	85.80	21.1
4336	1248	88.5	28.09
4298	2179	140	16.0

I had seen some values in Power and Mileage as 0.0 so verifying data for Engine, Power, Mileage. Will check once again after converting datatype

[16]:
```
cars.query("Power == '0.0'")['Power'].count()
```

[16]: 0

[17]:
```
cars.query("Mileage == '0.0'")['Mileage'].count()
```

[17]: 81

Converting this observations to Nan so we will remember to handle them when handling missing values.

[18]:
```
cars.loc[cars["Mileage"]=='0.0','Mileage']=np.nan
```

[19]:
```
cars.loc[cars["Engine"]=='0.0','Engine'].count()
```

[19]: 0

[20]:
```
cars[num].nunique()
```

[20]:
```
Engine      150
Power       385
Mileage     437
```

dtype: int64

[21]:
```
cars[num].isnull().sum()
```

[21]:
```
Engine       46
Power       175
Mileage      83
dtype: int64
```

There are 46 missing values in Engine, 175 in Power,83 in Mileage.

3.0.5 Processing Seats

[22]:
```
cars.query("Seats == 0.0")['Seats']
```

[22]:
```
S.No.
3999    0.00000
Name: Seats, dtype: float64
```

[23]:
```
#seats cannot be 0 so changing it to nan and will be handled in missing value
cars.loc[3999,'Seats'] =np.nan
```

3.0.6 Processing 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.

[24]:
```
# Create a new column after splitting the New_Price values.
import re

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"]  format
            # that we see in the sample output
            print(
                "The data needs furthur processing.mismatch ",
```

```
                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.mismatch  1.28 Cr
The data needs furthur processing.mismatch  1.04 Cr
The data needs furthur processing.mismatch  1 Cr
The data needs furthur processing.mismatch  1.04 Cr
The data needs furthur processing.mismatch  1.39 Cr
The data needs furthur processing.mismatch  1.02 Cr
The data needs furthur processing.mismatch  1.4 Cr
The data needs furthur processing.mismatch  1.06 Cr
The data needs furthur processing.mismatch  1.27 Cr
The data needs furthur processing.mismatch  1.13 Cr
The data needs furthur processing.mismatch  1.36 Cr
The data needs furthur processing.mismatch  1.66 Cr
The data needs furthur processing.mismatch  1.6 Cr
The data needs furthur processing.mismatch  1.28 Cr
The data needs furthur processing.mismatch  2.3 Cr
The data needs furthur processing.mismatch  1.71 Cr
The data needs furthur processing.mismatch  1.39 Cr
The data needs furthur processing.mismatch  1.58 Cr
The data needs furthur processing.mismatch  3.75 Cr
The data needs furthur processing.mismatch  1.06 Cr
```

[25]:

```
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
cars["new_price_num"] = new_price_num

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

[25]:

S.No.	Name	Location	Year	Kilometers_Driven	\
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	
2	Honda Jazz V	Chennai	2011	46000	
3	Maruti Ertiga VDI	Chennai	2012	87000	
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	

S.No.	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	\
0	CNG	Manual	First	26.6	998	58.16	5.00000	
1	Diesel	Manual	First	19.67	1582	126.2	5.00000	
2	Petrol	Manual	First	18.2	1199	88.7	5.00000	
3	Diesel	Manual	First	20.77	1248	88.76	7.00000	
4	Diesel	Automatic	Second	15.2	1968	140.8	5.00000	

S.No.	New_Price	Price	new_price_num
0	NaN	1.75000	NaN
1	NaN	12.50000	NaN
2	8.61 Lakh	4.50000	8.61000
3	NaN	6.00000	NaN
4	NaN	17.74000	NaN

4 Feature Enginering

4.1 converting datatype

[26]:
```
#converting object data type to category data type
cars["Fuel_Type"] = cars["Fuel_Type"].astype("category")
cars["Transmission"]  =  cars["Transmission"].astype("category")
cars["Owner_Type"] = cars["Owner_Type"].astype("category")
#converting datatype
cars["Mileage"]  =  cars["Mileage"].astype(float)
cars["Power"] = cars["Power"].astype(float)
cars["Engine"]=cars["Engine"].astype(float)
```

[27]:
```
cars.describe().T
```

[27]:

	count	mean	std	min	25%	\
Year	7253.00000	2013.36537	3.25442	1996.00000	2011.00000	
Kilometers_Driven	7253.00000	58699.06315	84427.72058	171.00000	34000.00000	
Mileage	7170.00000	18.34653	4.15791	6.40000	15.30000	
Engine	7207.00000	1616.57347	595.28514	72.00000	1198.00000	
Power	7078.00000	112.76521	53.49355	34.20000	75.00000	
Seats	7199.00000	5.28046	0.80933	2.00000	5.00000	

```
Price              6019.00000     9.47947    11.18792     0.44000     3.50000
new_price_num      1006.00000    22.77969    27.75934     3.91000     7.88500

                          50%         75%             max
Year               2014.00000  2016.00000      2019.00000
Kilometers_Driven 53416.00000 73000.00000   6500000.00000
Mileage              18.20000    21.10000        33.54000
Engine             1493.00000  1968.00000      5998.00000
Power                94.00000   138.10000       616.00000
Seats                 5.00000     5.00000        10.00000
Price                 5.64000     9.95000       160.00000
new_price_num        11.57000    26.04250       375.00000
```

4.1.1 Processing Years to Derive Age of car

Since year has 2014, 1996 etc. But this will not help to understand how old cars is and its effect on price. so creating two new columns current year and Age . Current year would be 2021 and Age column would be Ageofcar= currentyear-year. And then drop currentyear columns

[28]:
```python
cars['Current_year']=2021
cars['Ageofcar']=cars['Current_year']-cars['Year']
cars.drop('Current_year',axis=1,inplace=True)
cars.head()
```

[28]:
```
                                 Name    Location  Year  Kilometers_Driven  \
S.No.
0                Maruti Wagon R LXI CNG      Mumbai  2010              72000
1      Hyundai Creta 1.6 CRDi SX Option        Pune  2015              41000
2                          Honda Jazz V     Chennai  2011              46000
3                     Maruti Ertiga VDI     Chennai  2012              87000
4       Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013              40670

      Fuel_Type Transmission Owner_Type  Mileage      Engine     Power    Seats  \
S.No.
0           CNG       Manual      First 26.60000   998.00000  58.16000 5.00000
1        Diesel       Manual      First 19.67000  1582.00000 126.20000 5.00000
2        Petrol       Manual      First 18.20000  1199.00000  88.70000 5.00000
3        Diesel       Manual      First 20.77000  1248.00000  88.76000 7.00000
4        Diesel    Automatic     Second 15.20000  1968.00000 140.80000 5.00000

        New_Price     Price  new_price_num  Ageofcar
S.No.
0             NaN   1.75000            NaN        11
1             NaN  12.50000            NaN         6
2       8.61 Lakh   4.50000        8.61000        10
3             NaN   6.00000            NaN         9
4             NaN  17.74000            NaN         8
```

4.1.2 Processing Name column

Brands do play an important role in Car selection and Prices. So extracting brand names from the Name.

[29]:
```
#dropping rows with name as null
cars["Name"] = cars.dropna(subset=["Name"])
```

[30]:
```
#As mentioned in dataset car name has Brand and model so extracting it ,This␣
↪can help to fill missing values of price column as brand
cars["Brand"] = cars["Name"].str.split(" ").str[0] #Separating Brand name from␣
↪the Name
cars["Model"] = cars["Name"].str.split(" ").str[1] + cars["Name"].str.split("␣
↪").str[2]
```

[31]:
```
cars.Brand.unique()
```

[31]:
```
array(['Maruti', 'Hyundai', 'Honda', 'Audi', 'Nissan', 'Toyota',
       'Volkswagen', 'Tata', 'Land', 'Mitsubishi', 'Renault',
       'Mercedes-Benz', 'BMW', 'Mahindra', 'Ford', 'Porsche', 'Datsun',
       'Jaguar', 'Volvo', 'Chevrolet', 'Skoda', 'Mini', 'Fiat', 'Jeep',
       'Smart', 'Ambassador', 'Isuzu', 'ISUZU', 'Force', 'Bentley',
       'Lamborghini', 'Hindustan', 'OpelCorsa'], dtype=object)
```

[32]:
```
col=["ISUZU","Isuzu","Mini","Land"]
#correcting brand names
cars[cars.Brand.isin(col)].sample(5)
```

[32]:
```
                                              Name   Location  Year  \
S.No.
6367          Land Rover Freelander 2 S Business Edition      Kochi  2015
7157                    Land Rover Range Rover 2.2L Pure  Hyderabad  2015
2604                           Mini Cooper Convertible S     Mumbai  2016
4755  Land Rover Range Rover Evoque 2.0 TD4 HSE Dynamic  Hyderabad  2016
4687                        Land Rover Freelander 2 TD4 SE     Jaipur  2012

       Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage  \
S.No.
6367               61062    Diesel    Automatic      First 12.39000
7157               49000    Diesel    Automatic     Second 12.70000
2604               15000    Petrol    Automatic      First 16.82000
4755               52000    Diesel    Automatic      First 15.68000
4687              119203    Diesel    Automatic      First       NaN

          Engine     Power   Seats  New_Price   Price  new_price_num  \
S.No.
6367  2179.00000 147.51000 5.00000        NaN     NaN            NaN
7157  2179.00000 187.70000 5.00000        NaN     NaN            NaN
```

```
2604  1998.00000 189.08000 4.00000   44.28 Lakh 35.00000        44.28000
4755  1999.00000 177.00000 5.00000   74.49 Lakh 42.00000        74.49000
4687  2179.00000 115.00000 5.00000          NaN 16.50000             NaN

       Ageofcar Brand                Model
S.No.
6367          6  Land      RoverFreelander
7157          6  Land           RoverRange
2604          5  Mini  CooperConvertible
4755          5  Land           RoverRange
4687          9  Land      RoverFreelander
```

Brand names like ISUZU and Isuzu are same and needs to be corrected. Land, Mini seems to be incorrect. So correcting brand names.

[33]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7253 entries, 0 to 7252
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -------
 0   Name               7253 non-null   object
 1   Location           7253 non-null   object
 2   Year               7253 non-null   int64
 3   Kilometers_Driven  7253 non-null   int64
 4   Fuel_Type          7253 non-null   category
 5   Transmission       7253 non-null   category
 6   Owner_Type         7253 non-null   category
 7   Mileage            7170 non-null   float64
 8   Engine             7207 non-null   float64
 9   Power              7078 non-null   float64
 10  Seats              7199 non-null   float64
 11  New_Price          1006 non-null   object
 12  Price              6019 non-null   float64
 13  new_price_num      1006 non-null   float64
 14  Ageofcar           7253 non-null   int64
 15  Brand              7253 non-null   object
 16  Model              7252 non-null   object
dtypes: category(3), float64(6), int64(3), object(5)
memory usage: 1.1+ MB
```

[34]:
```
#changing brandnames
cars.loc[cars.Brand == 'ISUZU','Brand']='Isuzu'
cars.loc[cars.Brand=='Mini','Brand']='Mini Cooper'
cars.loc[cars.Brand=='Land','Brand']='Land Rover'
#cars['Brand']=cars["Brand"].astype("category")
```

[35]:
```
cars.Brand.nunique()
```

[35]: 32

[36]:
```
cars.groupby(cars.Brand).size().sort_values(ascending =False)
```

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

There are 32 unique Brands in the dataset.Maruti brand is most available for purchase/Sold followed by Hyundai.

[37]:
```
cars.Model.isnull().sum()
```

[37]: 1

[38]:
```
#drop row with no model
cars.dropna(subset=['Model'],axis=0,inplace=True)
```

[39]:
```
cars.Model.nunique()
```

[39]:
```
726
```

[40]:
```
cars.groupby("Model")["Model"].size().nlargest(30)
```

[40]:
```
Model
SwiftDzire        189
Grandi10          179
WagonR            178
Innova2.5         145
Verna1.6          127
City1.5           122
Cityi             115
Creta1.6          110
NewC-Class        110
3Series           109
SwiftVDI           96
5Series            86
i201.2             78
SantroXing         76
XUV500W8           75
i10Sportz          75
AmazeS             69
i10Magna           69
Alto800            63
CorollaAltis       63
FigoDiesel         61
Ecosport1.5        59
A42.0              56
AltoK10            56
VitaraBrezza       55
i20Asta            54
InnovaCrysta       53
i20Sportz          53
Duster110PS        51
Fortuner4x2        50
Name: Model, dtype: int64
```

There are 726 unique models and Swift Dzire is most popular Model.

5 EDA

[41]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7252 entries, 0 to 7252
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               7252 non-null   object
 1   Location           7252 non-null   object
 2   Year               7252 non-null   int64
 3   Kilometers_Driven  7252 non-null   int64
 4   Fuel_Type          7252 non-null   category
 5   Transmission       7252 non-null   category
 6   Owner_Type         7252 non-null   category
 7   Mileage            7169 non-null   float64
 8   Engine             7206 non-null   float64
 9   Power              7077 non-null   float64
 10  Seats              7198 non-null   float64
 11  New_Price          1006 non-null   object
 12  Price              6019 non-null   float64
 13  new_price_num      1006 non-null   float64
 14  Ageofcar           7252 non-null   int64
 15  Brand              7252 non-null   object
 16  Model              7252 non-null   object
dtypes: category(3), float64(6), int64(3), object(5)
memory usage: 871.6+ KB
```

[42]:
```
cars.describe()
```

[42]:

	Year	Kilometers_Driven	Mileage	Engine	Power \
count	7252.00000	7252.00000	7169.00000	7206.00000	7077.00000
mean	2013.36652	58700.26269	18.34711	1616.60505	112.76871
std	3.25316	84433.48037	4.15791	595.32041	53.49652
min	1996.00000	171.00000	6.40000	72.00000	34.20000
25%	2011.00000	34000.00000	15.30000	1198.00000	75.00000
50%	2014.00000	53429.00000	18.20000	1493.00000	94.00000
75%	2016.00000	73000.00000	21.10000	1968.00000	138.10000
max	2019.00000	6500000.00000	33.54000	5998.00000	616.00000

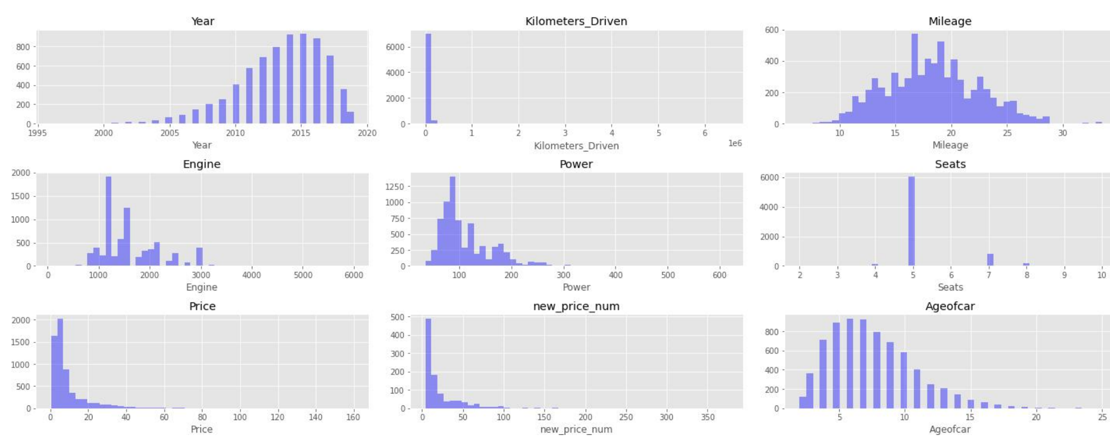
	Seats	Price	new_price_num	Ageofcar
count	7198.00000	6019.00000	1006.00000	7252.00000
mean	5.28049	9.47947	22.77969	7.63348
std	0.80938	11.18792	27.75934	3.25316
min	2.00000	0.44000	3.91000	2.00000
25%	5.00000	3.50000	7.88500	5.00000

50%	5.00000	5.64000	11.57000	7.00000
75%	5.00000	9.95000	26.04250	10.00000
max	10.00000	160.00000	375.00000	25.00000

Observations

- Years is left skewed. Years ranges from 1996- 2019 . Age of cars 2 year old to 25 years old
- Kilometer driven , median is ~53k Km and mean is ~58K. Max values seems to be 6500000. This is very high , and seems to be outlier. Need to analyze further.
- Mileage is almost Normally distrubuited
- Engine is right skewed and has outliers on higher and lower end
- Power and Price are also right skewed.
- Price 160 Lakh is too much for a used car. Seems to be an outlier.

[43]:
```python
plt.style.use('ggplot')
#select all quantitative columns for checking the spread
numeric_columns = cars.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20,25))

for i, variable in enumerate(numeric_columns):
                     plt.subplot(10,3,i+1)

                     sns.distplot(cars[variable],kde=False,color='blue')
                     plt.tight_layout()
                     plt.title(variable)
```

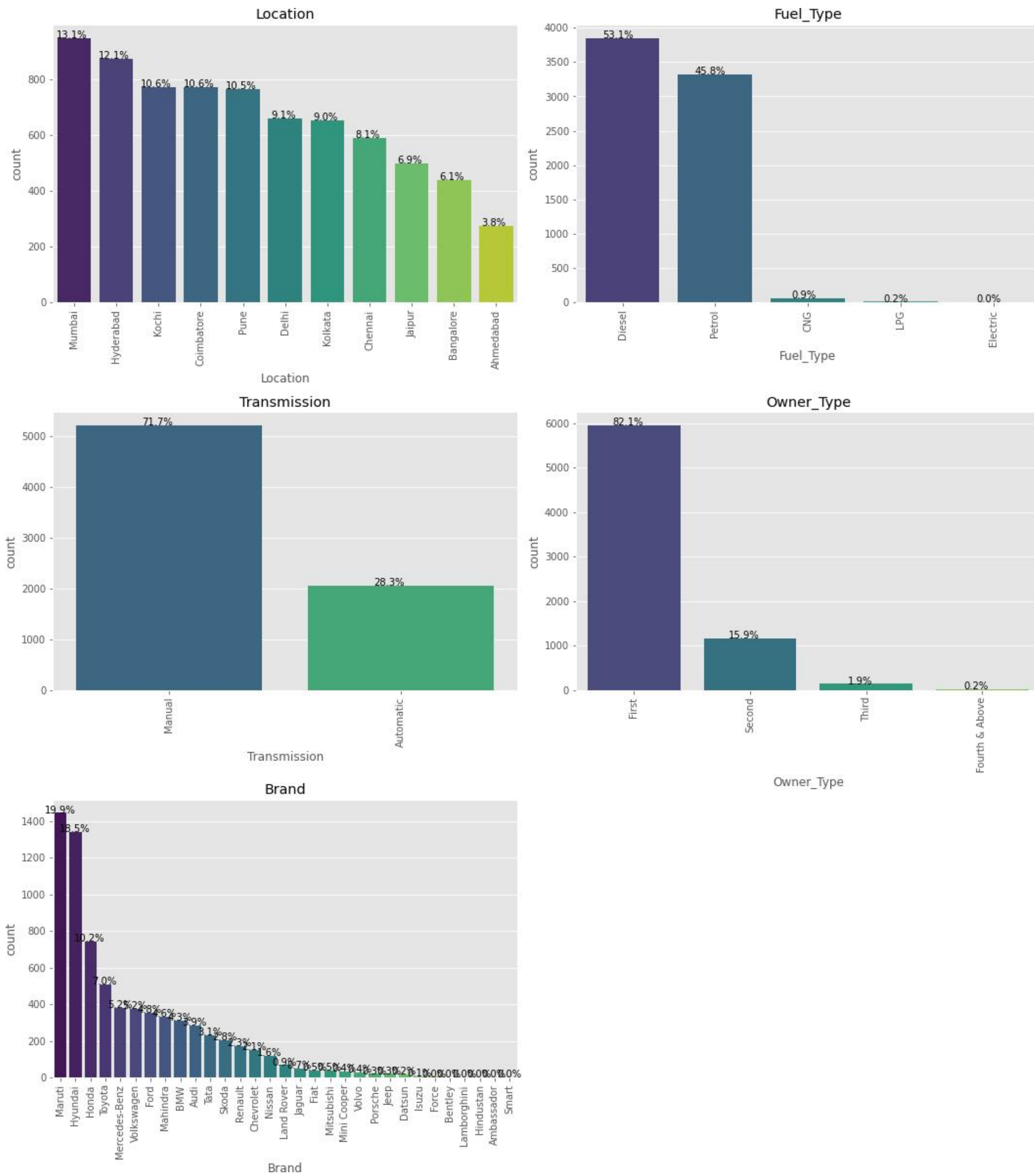


Observations

- Year is left skewed and has outilers on lower side., This column can be dropped
- Kilometer_driven is right skewed.

- Mileage is almost Normally distrubuted. Has few outliers on upper and lower side. need to check further.
- Engine ,power and price are right skewed and has outliers on upper side.
- Age of car is right skewed.

[44]:
```
cat_columns=['Location','Fuel_Type','Transmission', 'Owner_Type', 'Brand'] 
↪#cars.select_dtypes(exclude=np.number).columns.tolist()

plt.figure(figsize=(15,21))

for i, variable in enumerate(cat_columns):
                     plt.subplot(4,2,i+1)
                     order = cars[variable].value_counts(ascending=False).index
                     ax=sns.countplot(x=cars[variable], data=cars , order=order 
 ↪,palette='viridis')
                     for p in ax.patches:
                           percentage = '{:.1f}%'.format(100 * p.get_height()/
 ↪len(cars[variable]))
                           x = p.get_x() + p.get_width() / 2 - 0.05
                           y = p.get_y() + p.get_height()
                           plt.annotate(percentage, (x, y),ha='center')
                     plt.xticks(rotation=90)
                     plt.tight_layout()
                     plt.title(variable)
```



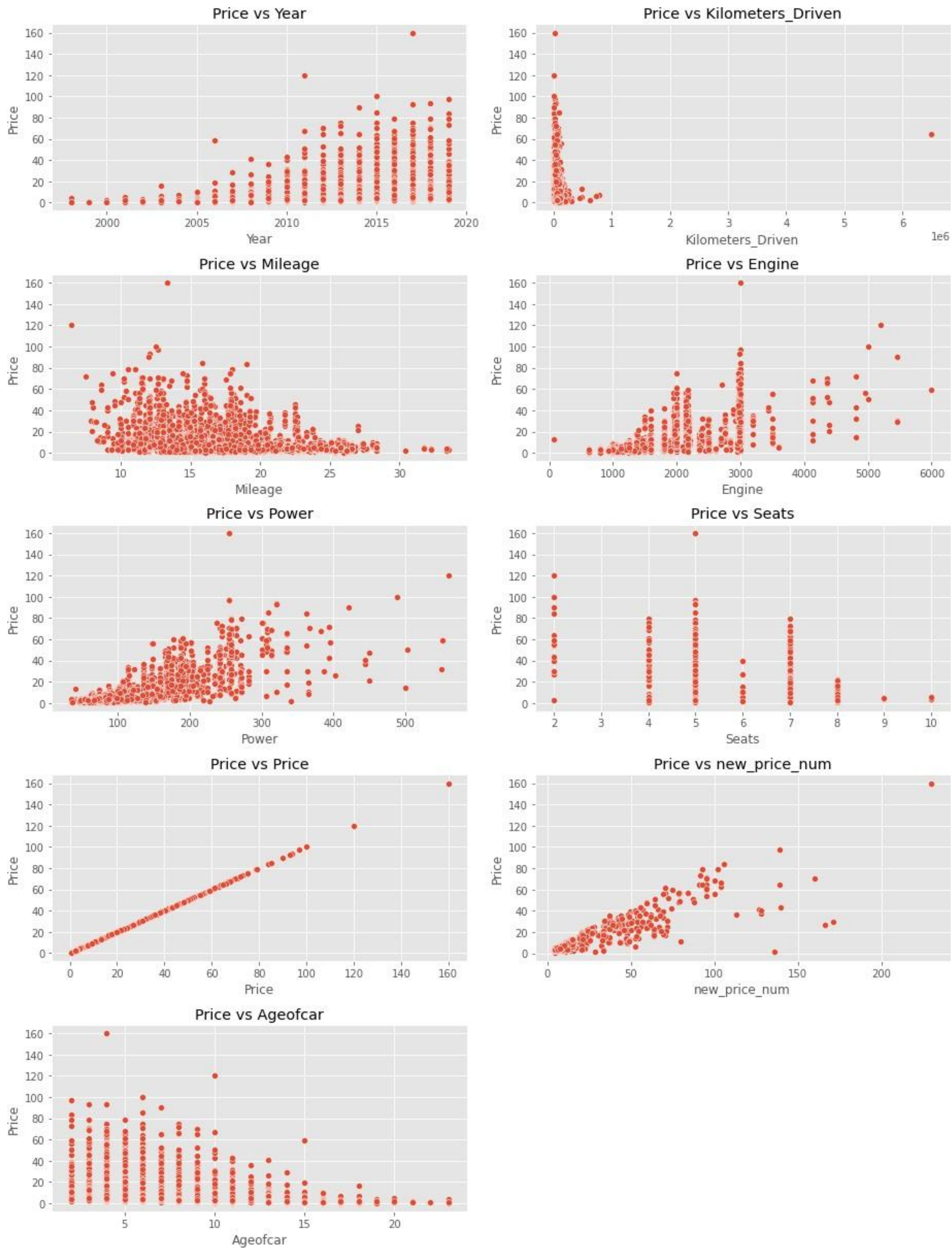
Observations

Car Profile

- ~71 % cars available for sell have manual Transmission.
- ~82 % cars are First owned cars.
- ~39% of car available for sale are from Maruti & Hyundai brands.
- ~53% of car being sold/avialable for purchase have fuel type as Diesel .
- Mumbai has highest numbers of car availabe for purchase whereas Ahmedabad has least
- Most of the cars are 5 seaters.

- Car being sold/available for purchase are in 2 - 23 years old
- ~ 71% car are lower price range car.

[45]:
```
numeric_columns= numeric_columns = cars.select_dtypes(include=np.number).
 ↪columns.tolist()
plt.figure(figsize=(13,17))

for i, variable in enumerate(numeric_columns):
                     plt.subplot(5,2,i+1)
                     sns.scatterplot(x=cars[variable],y=cars['Price']).
 ↪set(title='Price vs '+ variable)
                     #plt.xticks(rotation=90)
                     plt.tight_layout()
```


Price vs Year
Price
Year
Price vs Kilometers_Driven
Price
Kilometers_Driven
1e6
Price vs Mileage
Price
Mileage
Price vs Engine
Price
Engine
Price vs Power
Price
Power
Price vs Seats
Price
Seats
Price vs Price
Price
Price
Price vs new_price_num
Price
new_price_num
Price vs Ageofcar
Price
Ageofcar

6 Handling missing values

[46]:
```
cars.isnull().sum()
```

[46]:
```
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                83
Engine                 46
Power                 175
Seats                  54
New_Price            6246
Price                1233
new_price_num        6246
Ageofcar                0
Brand                   0
Model                   0
dtype: int64
```

6.0.1 Calculating missing values in each row

[47]:
```
# counting the number of missing values per row
num_missing = cars.isnull().sum(axis=1)
num_missing.value_counts()
```

[47]:
```
2    5025
3    1112
0     819
1     187
4      57
5      31
6      20
7       1
dtype: int64
```

[48]:
```
#Investigating how many missing values per row are there for each variable
for n in num_missing.value_counts().sort_index().index:
    if n > 0:
        print("*" *30,f'\nFor the rows with exactly {n} missing values, NAs are␣
↪found in:')
        n_miss_per_col = cars[num_missing == n].isnull().sum()
        print(n_miss_per_col[n_miss_per_col > 0])
        print('\n\n')
```

```
******************************
For the rows with exactly 1 missing values, NAs are found in:
Mileage      5
Price      182
dtype: int64



******************************
For the rows with exactly 2 missing values, NAs are found in:
New_Price        5025
new_price_num    5025
dtype: int64



******************************
For the rows with exactly 3 missing values, NAs are found in:
Mileage            25
Power              74
Seats               1
New_Price        1112
Price            1012
new_price_num    1112
dtype: int64



******************************
For the rows with exactly 4 missing values, NAs are found in:
Mileage          35
Power            50
Seats             6
New_Price        57
Price            23
new_price_num    57
dtype: int64



******************************
For the rows with exactly 5 missing values, NAs are found in:
Mileage           6
Engine           25
Power            30
Seats            26
New_Price        31
Price             6
```

```
new_price_num     31
dtype: int64


******************************
For the rows with exactly 6 missing values, NAs are found in:
Mileage          11
Engine           20
Power            20
Seats            20
New_Price        20
Price             9
new_price_num    20
dtype: int64


******************************
For the rows with exactly 7 missing values, NAs are found in:
Mileage          1
Engine           1
Power            1
Seats            1
New_Price        1
Price            1
new_price_num    1
dtype: int64
```

This confirms that certain columns tend to be missing together or all nonmissing together. So will try to fill the missing values , as much as possible.

[49]:
```
cars[num_missing==7]
```

[49] :
```
                      Name Location  Year  Kilometers_Driven Fuel_Type  \
S.No.
6633  Mahindra TUV 300 P4  Kolkata  2016              27000    Diesel

      Transmission Owner_Type  Mileage  Engine  Power  Seats New_Price  Price  \
S.No.
6633        Manual      First      NaN     NaN    NaN    NaN       NaN    NaN

       new_price_num  Ageofcar     Brand   Model
S.No.
6633             NaN         5  Mahindra  TUV300
```

[50]:
```
col=['Engine','Power','Mileage']
cars[col].isnull().sum()
```

[50]:
```
Engine      46
Power      175
Mileage     83
dtype: int64
```

We can start filling missing values by grouping name and year and fill in missing values. with median.

[51]:
```
cars.groupby(['Name','Year'])['Engine'].median().head(30)
```

[51]:
```
Name                                       Year
Ambassador Classic Nova Diesel             2003    1489.00000
Audi A3 35 TDI Attraction                  2014    1968.00000
Audi A3 35 TDI Premium                     2016    1968.00000
Audi A3 35 TDI Premium Plus                2015    1968.00000
                                           2016    1968.00000
Audi A3 35 TDI Technology                  2017    1968.00000
Audi A4 1.8 TFSI                           2010    1781.00000
                                           2011    1781.00000
Audi A4 1.8 TFSI Technology Edition        2012    1798.00000
Audi A4 2.0 TDI                            2009    1968.00000
                                           2011    1968.00000
                                           2012    1968.00000
                                           2013    1968.00000
                                           2014    1968.00000
Audi A4 2.0 TDI 177 Bhp Premium Plus       2013    1968.00000
                                           2014    1968.00000
                                           2015    1968.00000
Audi A4 2.0 TDI 177 Bhp Technology Edition 2013    1968.00000
                                           2014    1968.00000
Audi A4 2.0 TDI Celebration Edition        2011    1968.00000
Audi A4 2.0 TDI Multitronic                2009    1968.00000
                                           2010    1968.00000
                                           2011    1968.00000
                                           2012    1968.00000
                                           2013    1968.00000
                                           2014    1968.00000
                                           2015    1968.00000
Audi A4 2.0 TDI Premium Sport Limited Edition 2014 1968.00000
                                           2015    1968.00000
Audi A4 2.0 TFSI                           2011    1984.00000
Name: Engine, dtype: float64
```

[52]:
```
cars['Engine']=cars.groupby(['Name','Year'])['Engine'].apply(lambda x:x.
    ↪fillna(x.median()))
cars['Power']=cars.groupby(['Name','Year'])['Power'].apply(lambda x:x.fillna(x.
    ↪median()))
cars['Mileage']=cars.groupby(['Name','Year'])['Mileage'].apply(lambda x:x.
    ↪fillna(x.median()))
```

[53]:
```
col=['Engine','Power','Mileage']
cars[col].isnull().sum()
```

[53]:
```
Engine      45
Power      162
Mileage     82
dtype: int64
```

[54]:
```
cars.groupby(['Brand','Model'])['Engine'].median().head(10)
```

[54]:
```
Brand       Model
Ambassador  ClassicNova    1489.00000
Audi        A335           1968.00000
            A41.8          1781.00000
            A42.0          1968.00000
            A43.0          2967.00000
            A43.2          3197.00000
            A430           1395.00000
            A435           1968.00000
            A4New          1968.00000
            A62.0          1968.00000
Name: Engine, dtype: float64
```

As we can see most of the model have same engine size and instead of just applying median , grouping with model and year that should give me more granularity, and near to accurate Engine values.

[55]:
```
#chosing Median to fill the the missing value as there are many outliers,
#grouping by model and year to get  more granularity and more accurate Engine
    ↪and then fillig with median
cars['Engine']=cars.groupby(['Brand','Model'])['Engine'].apply(lambda x:x.
    ↪fillna(x.median()))
```

[56]:
```
#chosing Median to fill  the the missing value as there are many outliers,
#grouping by model to get more granularity and more accurate Engine
cars['Power']=cars.groupby(['Brand','Model'])['Power'].apply(lambda x:x.
    ↪fillna(x.median()))
```

[57]:
```
#chosing Median to fill  the the missing value as there are many outliers,
#grouping by model to get more granularity and more accurate Engine
```

```
cars['Mileage']=cars.groupby(['Brand','Model'])['Mileage'].apply(lambda x:x.
  ↪fillna(x.median()))
```

[58]:
```
col=['Engine','Power','Mileage']
cars[col].isnull().sum()
```

[58]:
```
Engine     18
Power      63
Mileage    32
dtype: int64
```

There are still missing values , analyzing further .Grouping by only Model for Engine and then filling missing values with median. For Power and Mileage Engine values for a Brand can be used to get more accurate value.

[59]:
```
cars.groupby(['Model','Year'])['Engine'].agg({'median','mean','max'}).
  ↪sort_values(by='Model',ascending='True').head(10)
```

[59]:

Model	Year	mean	median	max
1000AC	1998	970.00000	970.00000	970.00000
1Series	2013	1995.00000	1995.00000	1995.00000
	2015	1995.00000	1995.00000	1995.00000
370ZAT	2012	3696.00000	3696.00000	3696.00000
3Series	2018	1995.00000	1995.00000	1995.00000
	2017	1995.00000	1995.00000	1995.00000
	2016	1995.00000	1995.00000	1995.00000
	2015	1995.00000	1995.00000	1995.00000
	2014	2078.16667	1995.00000	2993.00000
	2013	2066.42857	1995.00000	2993.00000

[60]:
```
cars.groupby(['Brand','Engine'])['Power'].agg({'mean','median','max'}).head(10)
```

[60]:

Brand	Engine	mean	median	max
Ambassador	1489.00000	35.50000	35.50000	35.50000
Audi	1395.00000	147.51000	147.51000	147.51000
	1781.00000	163.20000	163.20000	163.20000
	1798.00000	181.03333	187.74000	187.74000
	1968.00000	167.12318	174.33000	187.74000
	1984.00000	196.02200	207.90000	226.60000
	2698.00000	179.50000	179.50000	179.50000
	2773.00000	201.00000	201.00000	201.00000
	2894.00000	444.00000	444.00000	444.00000
	2967.00000	241.74000	241.40000	246.70000

[61]:
```
cars['Seats'].isnull().sum()
```

[61]: 54

Grouping with Name should give me more granularity, and near to accurate Seat values.

[62]:
```
cars['Seats']=cars.groupby(['Name'])['Seats'].apply(lambda x:x.fillna(x.
↪median()))
```

[63]:
```
cars['Seats'].isnull().sum()
```

[63]: 47

Grouping with Model should give me more granularity, and near to accurate Seat values.

[64]:
```
cars['Seats']=cars.groupby(['Model'])['Seats'].apply(lambda x:x.fillna(x.
↪median()))
```

[65]:
```
cars['Seats'].isnull().sum()
```

[65]: 23

Lets check which car types have missing values.

[66]:
```
cars[cars['Seats'].isnull()==True].head(10)
```

[66]:
```
                             Name   Location  Year  Kilometers_Driven  \
S.No.
208            Maruti Swift 1.3 VXi    Kolkata  2010              42001
733            Maruti Swift 1.3 VXi    Chennai  2006              97800
1327           Maruti Swift 1.3 ZXI  Hyderabad  2015              50295
2074           Maruti Swift 1.3 LXI       Pune  2011              24255
2325       Maruti Swift 1.3 VXI ABS       Pune  2015              67000
2335           Maruti Swift 1.3 VXi     Mumbai  2007              55000
2369             Maruti Estilo LXI    Chennai  2008              56000
2668           Maruti Swift 1.3 VXi    Kolkata  2014              32986
3404           Maruti Swift 1.3 VXi     Jaipur  2006             125000
3810   Honda CR-V AT With Sun Roof    Kolkata  2013              27000

      Fuel_Type Transmission      Owner_Type  Mileage     Engine  Power  \
S.No.
208      Petrol       Manual           First 16.10000        NaN    NaN
733      Petrol       Manual           Third 16.10000        NaN    NaN
1327     Petrol       Manual           First 16.10000        NaN    NaN
2074     Petrol       Manual           First 16.10000        NaN    NaN
2325     Petrol       Manual           First 16.10000        NaN    NaN
2335     Petrol       Manual          Second 16.10000        NaN    NaN
2369     Petrol       Manual          Second 19.50000 1061.00000    NaN
2668     Petrol       Manual           First 16.10000        NaN    NaN
3404     Petrol       Manual  Fourth & Above 16.10000        NaN    NaN
```

```
3810      Petrol       Automatic              First  14.00000          NaN      NaN

         Seats  New_Price      Price  new_price_num  Ageofcar   Brand       Model
S.No.
208        NaN        NaN  2.11000            NaN        11  Maruti    Swift1.3
733        NaN        NaN  1.75000            NaN        15  Maruti    Swift1.3
1327       NaN        NaN  5.80000            NaN         6  Maruti    Swift1.3
2074       NaN        NaN  3.15000            NaN        10  Maruti    Swift1.3
2325       NaN        NaN  4.70000            NaN         6  Maruti    Swift1.3
2335       NaN        NaN  1.75000            NaN        14  Maruti    Swift1.3
2369       NaN        NaN  1.50000            NaN        13  Maruti   EstiloLXI
2668       NaN        NaN  4.24000            NaN         7  Maruti    Swift1.3
3404       NaN        NaN  2.35000            NaN        15  Maruti    Swift1.3
3810       NaN        NaN 11.99000            NaN         8   Honda      CR-VAT
```

[67]:
```
#most of cars are 5 seater so fillrest of 23 by 5
cars['Seats']=cars['Seats'].fillna(5)
```

[68]:
```
cars['Seats'].isnull().sum()
```

[68]:
```
0
```

Need to analyse along with price if seats plays any role in price

[69]:
```
cars["Location"]  = cars["Location"].astype("category")
cars['Brand'] =cars['Brand'].astype("category")
```

[70]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7252 entries, 0 to 7252
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -------
 0   Name               7252 non-null   object
 1   Location           7252 non-null   category
 2   Year               7252 non-null   int64
 3   Kilometers_Driven  7252 non-null   int64
 4   Fuel_Type          7252 non-null   category
 5   Transmission       7252 non-null   category
 6   Owner_Type         7252 non-null   category
 7   Mileage            7220 non-null   float64
 8   Engine             7234 non-null   float64
 9   Power              7189 non-null   float64
 10  Seats              7252 non-null   float64
 11  New_Price          1006 non-null   object
 12  Price              6019 non-null   float64
 13  new_price_num      1006 non-null   float64
```

```
 14  Ageofcar           7252 non-null   int64
 15  Brand              7252 non-null   category
 16  Model              7252 non-null   object
dtypes: category(5), float64(6), int64(3), object(3)
memory usage: 1.0+ MB
```

7 Processing New Price

[71]:
```
#For better granularity grouping has there would be same car model present so␣
↪filling with a median value brings it more near to real value
cars['new_price_num']=cars.groupby(['Name','Year'])['new_price_num'].
↪apply(lambda x:x.fillna(x.median()))
```

[72]:
```
cars.new_price_num.isnull().sum()
```

[72]: 6138

[73]:
```
cars['new_price_num']=cars.groupby(['Name'])['new_price_num'].apply(lambda x:x.
↪fillna(x.median()))
```

[74]:
```
cars.new_price_num.isnull().sum()
```

[74]: 6019

[75]:
```
cars['new_price_num']=cars.groupby(['Brand','Model'])['new_price_num'].
↪apply(lambda x:x.fillna(x.median()))
```

[76]:
```
cars.new_price_num.isnull().sum()
```

[76]: 4578

[77]:
```
cars['new_price_num']=cars.groupby(['Brand'])['new_price_num'].apply(lambda x:x.
↪fillna(x.median()))
```

[78]:
```
cars.drop(['New_Price'],axis=1,inplace=True)
```

[79]:
```
cars.new_price_num.isnull().sum()
```

[79]: 158

[80]:
```
cars.groupby(['Brand'])['new_price_num'].median().sort_values(ascending=False)
```

[80]:
```
Brand
 Bentley         375.00000
 Land Rover      139.00000
 Porsche         136.00000
```

```
BMW              55.07000
Jaguar           53.72000
Audi             53.14000
Mercedes-Benz    49.49000
Volvo            45.67000
Mini Cooper      42.30000
Isuzu            33.68000
Mitsubishi       33.21000
Jeep             22.95000
Toyota           21.08500
Nissan           15.06000
Skoda            14.92250
Ford             11.47500
Renault          11.27000
Volkswagen       10.94000
Mahindra         10.90000
Honda             8.92000
Fiat              8.62500
Hyundai           8.23000
Tata              7.70000
Maruti            7.00000
Datsun            4.98000
Ambassador            NaN
Chevrolet             NaN
Force                 NaN
Hindustan             NaN
Lamborghini           NaN
Smart                 NaN
Name: new_price_num, dtype: float64
```

[81]:
```
cars.isnull().sum()
```

[81]:
```
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                32
Engine                 18
Power                  63
Seats                   0
Price                1233
new_price_num         158
Ageofcar                0
Brand                   0
```

```
Model                        0
dtype: int64
```

[82]:
```
cols1 = ["Power","Mileage","Engine"]

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

[83]:
```
#dropping remaining rows
#cannot further fill this rows so dropping them

cars.dropna(inplace=True,axis=0)
```

[84]:
```
cars.isnull().sum()
```

[84]:
```
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Owner_Type           0
Mileage              0
Engine               0
Power                0
Seats                0
Price                0
new_price_num        0
Ageofcar             0
Brand                0
Model                0
dtype: int64
```

[85]:
```
cars.head()
```

[85]:
```
                                    Name    Location  Year  Kilometers_Driven  \
S.No.
0                 Maruti Wagon R LXI CNG      Mumbai  2010              72000
1      Hyundai Creta 1.6 CRDi SX Option        Pune  2015              41000
2                           Honda Jazz V     Chennai  2011              46000
3                      Maruti Ertiga VDI     Chennai  2012              87000
4        Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013              40670

      Fuel_Type Transmission Owner_Type  Mileage     Engine     Power   Seats  \
S.No.
0           CNG       Manual      First 26.60000  998.00000  58.16000 5.00000
1        Diesel       Manual      First 19.67000 1582.00000 126.20000 5.00000
```

```
2         Petrol       Manual      First  18.20000  1199.00000   88.70000  5.00000
3         Diesel       Manual      First  20.77000  1248.00000   88.76000  7.00000
4         Diesel    Automatic     Second  15.20000  1968.00000  140.80000  5.00000

         Price  new_price_num  Ageofcar    Brand      Model
S.No.
0      1.75000        5.29000        11   Maruti     WagonR
1     12.50000       16.06000         6  Hyundai   Creta1.6
2      4.50000        8.61000        10    Honda      JazzV
3      6.00000       11.27000         9   Maruti  ErtigaVDI
4     17.74000       53.14000         8     Audi      A4New
```

[86]:
```
cars.isnull().sum()
```

[86]:
```
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Owner_Type           0
Mileage              0
Engine               0
Power                0
Seats                0
Price                0
new_price_num        0
Ageofcar             0
Brand                0
Model                0
dtype: int64
```

[87]:
```
df.shape
```

[87]:
```
(7253, 13)
```

Finally done with all missing values handling

[88]:
```
cars.groupby(['Brand'])['Price'].agg({'median','mean','max'})
```

[88]:
```
                mean    median       max
Brand
Ambassador       NaN       NaN       NaN
Audi        25.53771  23.50000  72.94000
BMW         25.24315  21.00000  93.67000
Bentley     59.00000  59.00000  59.00000
Chevrolet        NaN       NaN       NaN
```

```
Datsun          3.04923  3.10000   3.95000
Fiat            3.26929  2.60000   7.71000
Force               NaN      NaN       NaN
Ford            6.88940  5.34500  56.80000
Hindustan           NaN      NaN       NaN
Honda           5.41174  4.95000  17.50000
Hyundai         5.34343  4.60000  23.00000
Isuzu          14.69667 16.09000  20.00000
Jaguar         37.63225 31.90000 100.00000
Jeep           18.71867 18.50000  23.91000
Lamborghini         NaN      NaN       NaN
Land Rover     39.25950 35.00000 160.00000
Mahindra        8.04592  7.57000  17.63000
Maruti          4.51727  4.15000  11.50000
Mercedes-Benz  26.80987 24.00000  90.00000
Mini Cooper    26.89692 24.28500  39.75000
Mitsubishi     11.05889  9.95000  28.00000
Nissan          4.73835  4.30000   8.92000
Porsche        48.34833 47.02000  75.00000
Renault         5.79903  5.49000  14.01000
Skoda           7.55908  6.00000  27.30000
Smart               NaN      NaN       NaN
Tata            3.56285  2.90000  17.85000
Toyota         11.58002 10.75000  35.82000
Volkswagen      5.30727  4.89000  24.90000
Volvo          18.80286 18.25000  32.50000
```

[89]:

```
#using business knowledge to create class
Low=['Maruti',
     'Hyundai',
     'Ambassdor',
     'Hindustan',
     'Force',
     'Chevrolet',
     'Fiat',
     'Tata',
     'Smart',
     'Renault',
     'Datsun',
     'Mahindra',
     'Skoda',
     'Ford',
     'Toyota',
     'Isuzu',
     'Mitsubishi','Honda']

High=['Audi',
```

```
        'Mini Cooper',
        'Bentley',
        'Mercedes-Benz',
        'Lamborghini',
        'Volkswagen',
        'Porsche',
        'Land Rover',
        'Nissan',
        'Volvo',
        'Jeep',
        'Jaguar',
        'BMW']# more than 30lakh
```

[90]:
```
def classrange(x):
    if x in Low:
        return "Low"
    elif x in High:
        return "High"
    else:
        return x
```

[91]:
```
cars['Brand_Class'] = cars['Brand'].apply(lambda x: classrange(x))
```

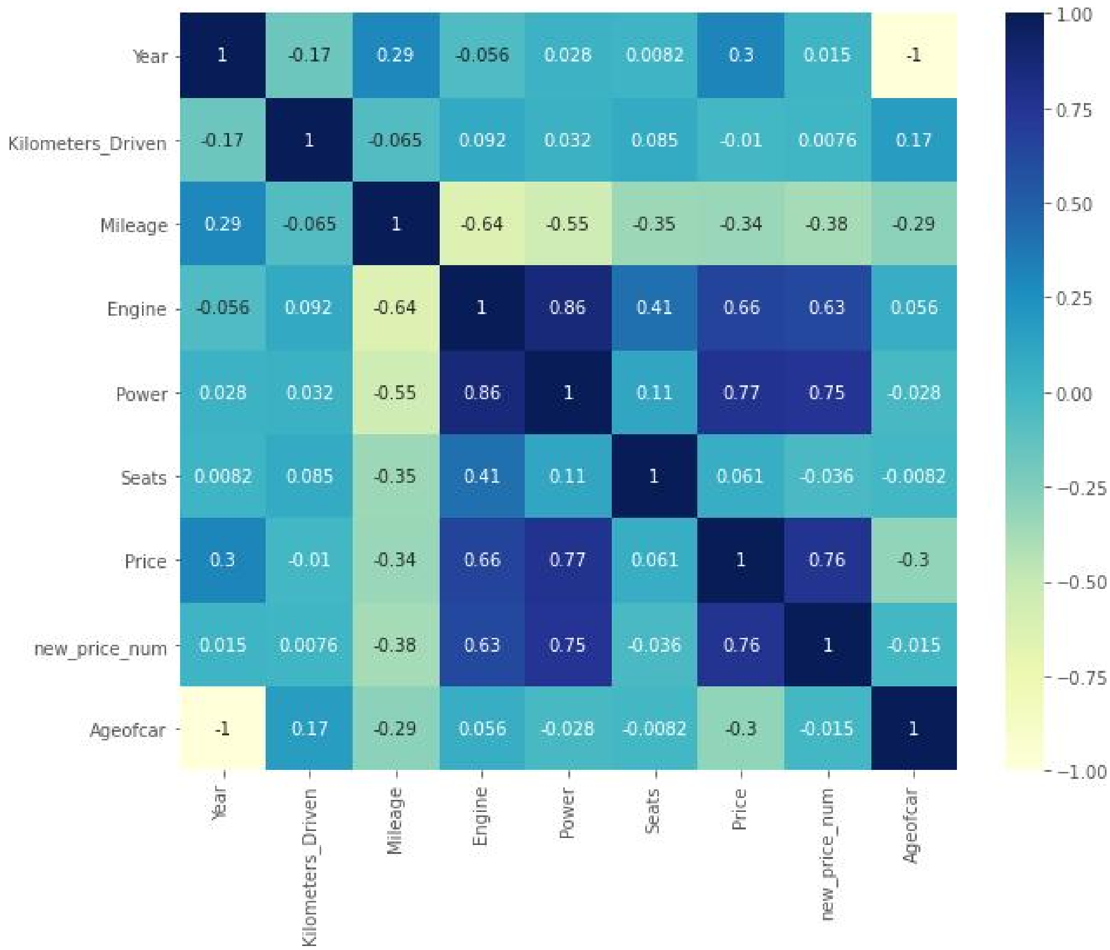
[92]:
```
cars['Brand_Class'].unique()
```

[92]:
```
array(['Low', 'High'], dtype=object)
```

[93]:
```

```

7.0.1 Bivariate & Multivariate Analysis

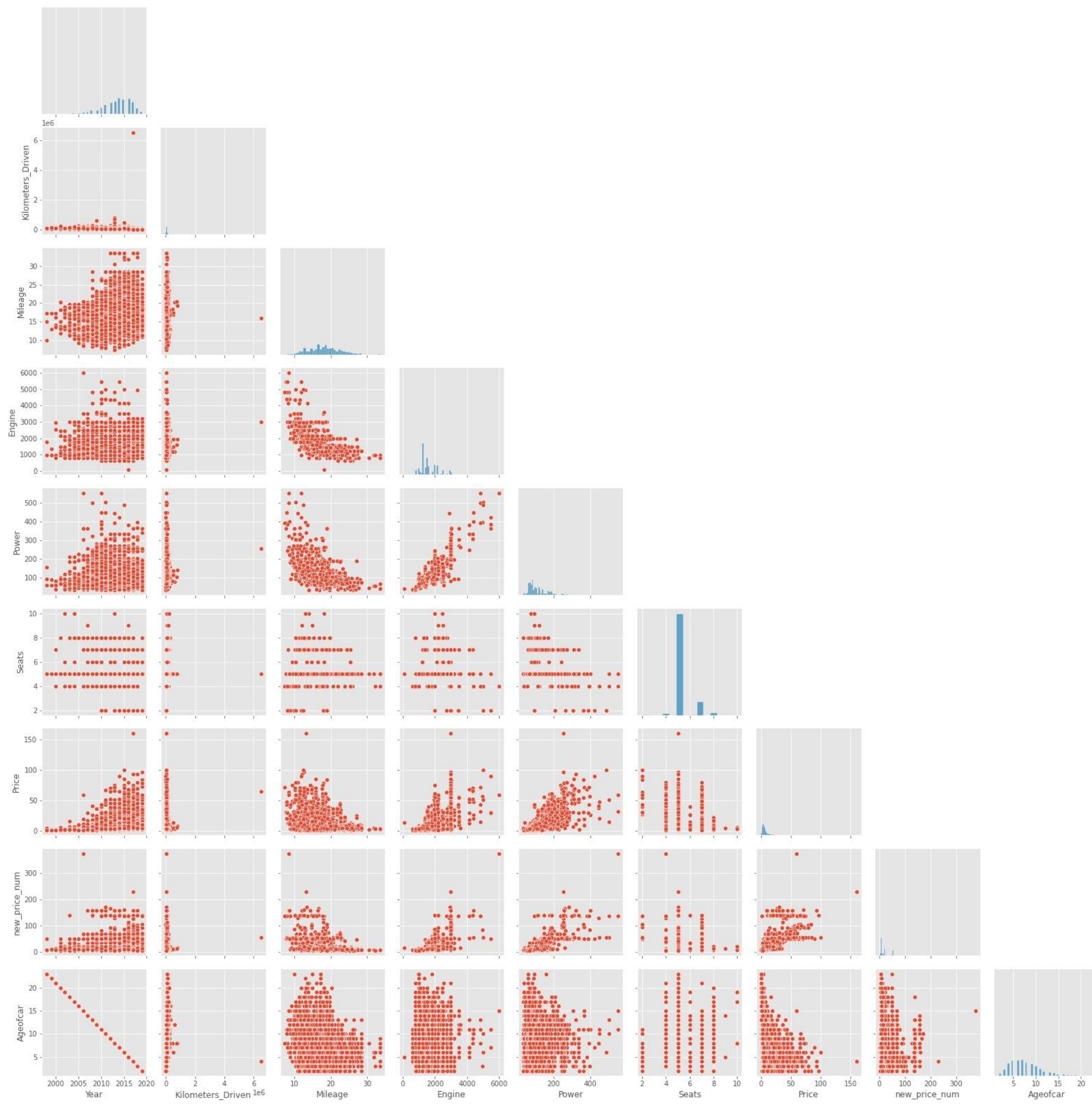
[94]:
```
plt.figure(figsize=(10,8))
sns.heatmap(cars.corr(),annot=True ,cmap="YlGnBu" )
plt.show()
```



Observations

- Engine has strong positive correlation to Power [0.86].
- Price has positive correlation to Engine[0.66] as well Power [0.77].
- Mileage is negative correlated to Engine,Power,Price.,Ageofcar
- Price has negative correlation to age of car.
- Kilometer driven doesnt impact Price

[95]:
```
sns.pairplot(data=cars , corner=True)
plt.show()
```



Observations

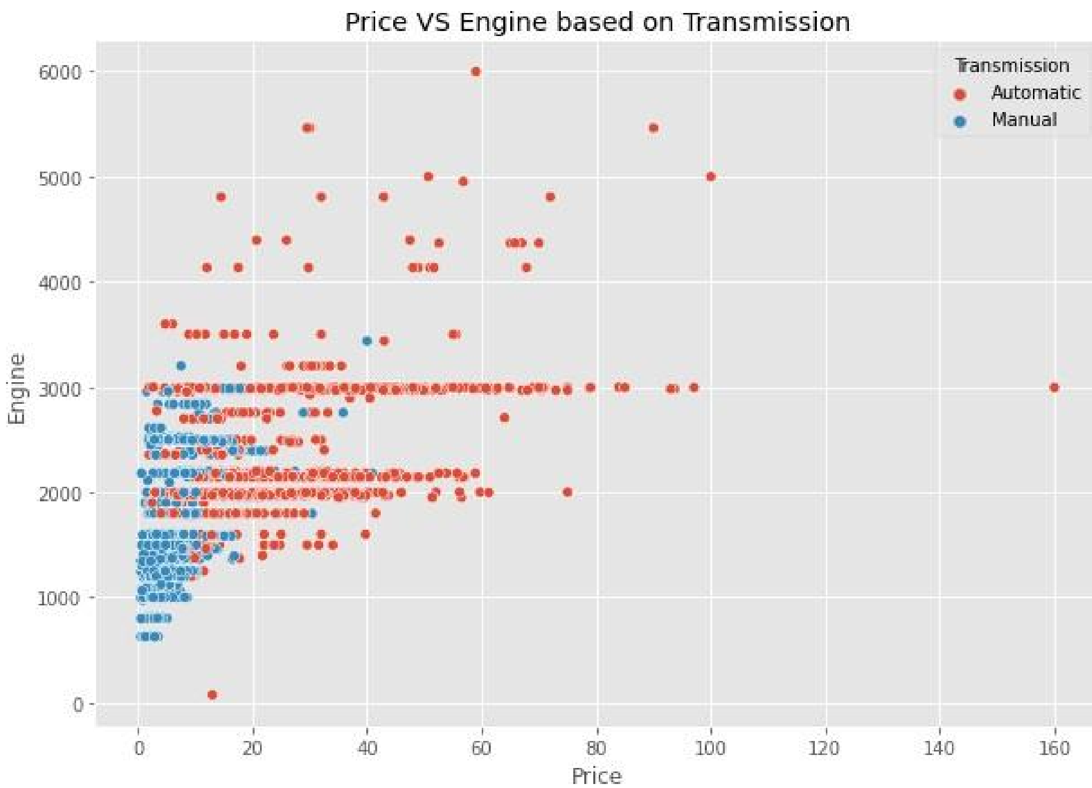
- Same observation about correlation as seen in heatmap.
- Kilometer driven doesnot have impact on Price .
- As power increase mileage decrease.
- Car with recent make sell at higher prices.
- Engine and Power increase , price of the car seems to increase.

7.0.2 Variables that are correlated with Price variable

Price Vs Engine Vs Transmission

[96]:
```python
# understand relation ship of Engine vs Price and Transmimssion
plt.figure(figsize=(10,7))

plt.title("Price VS Engine based on Transmission")
sns.scatterplot(y='Engine', x='Price', hue='Transmission', data=cars)
```

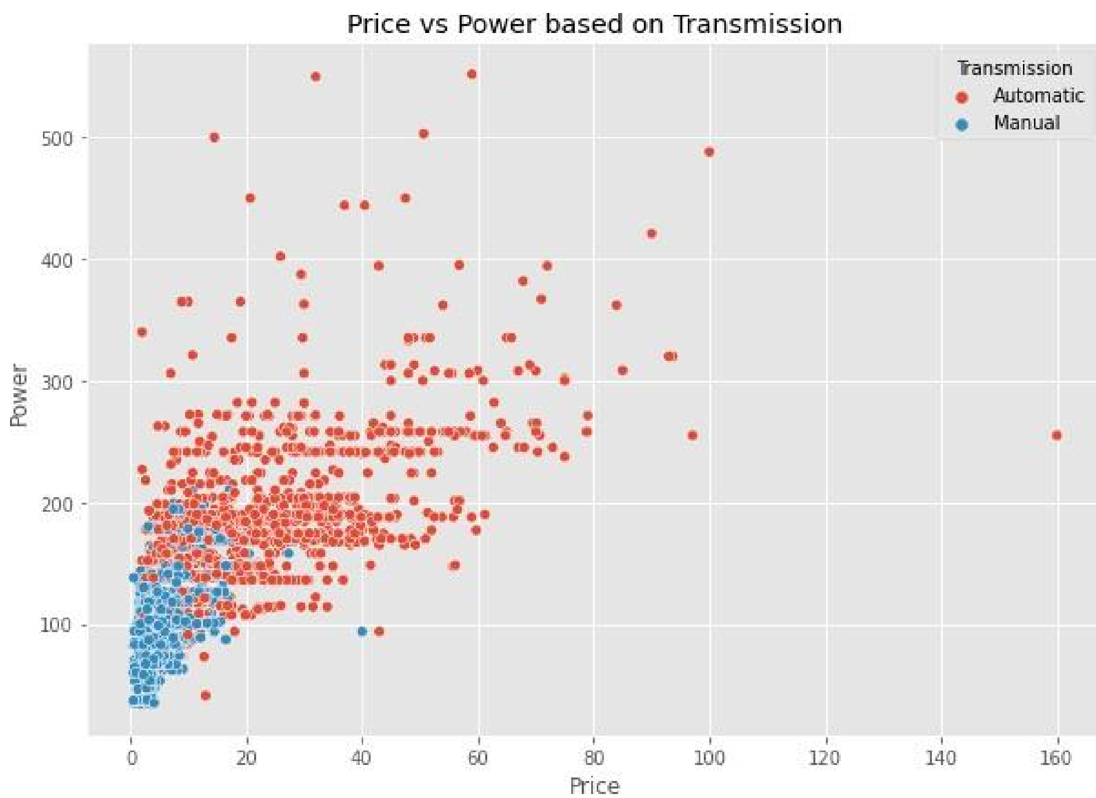
[96] : <AxesSubplot:title={'center':'Price VS Engine based on Transmission'}, xlabel='Price', ylabel='Engine'>



Price Vs Power vs Transmission

[97]:
```python
#understand relationship betweem Price and Power
plt.figure(figsize=(10,7))
plt.title("Price vs Power based on Transmission")
sns.scatterplot(y='Power', x='Price', hue='Transmission', data=cars)
```

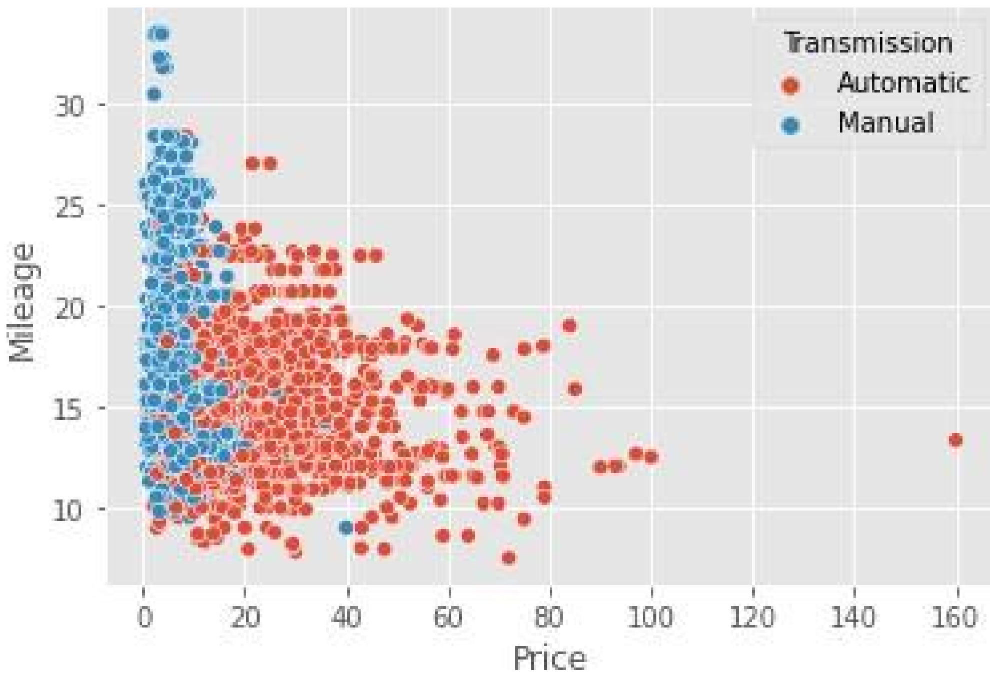
[97] : <AxesSubplot:title={'center':'Price vs Power based on Transmission'}, xlabel='Price', ylabel='Power'>



Price Vs Mileage Vs Transmission

[98]:
```
# Understand the relationships  between mileage and Price
sns.scatterplot(y='Mileage', x='Price', hue='Transmission', data=cars)
```

[98] : <AxesSubplot:xlabel='Price', ylabel='Mileage'>



Price Vs Year Vs Transmission

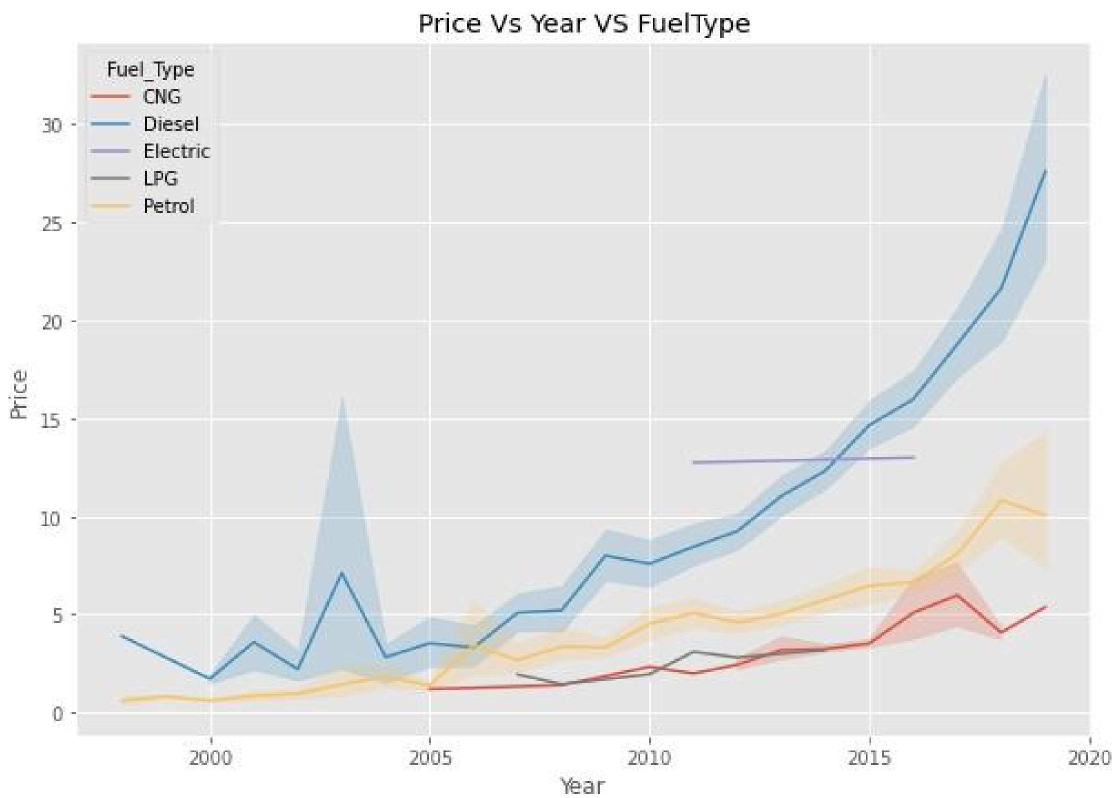
[99]:
```
# Impact of years on price
plt.figure(figsize=(10,7))
plt.title("Price based on manufacturing Year of model")
sns.lineplot(x='Year', y='Price',hue='Transmission',
             data=cars)
```

[99] :
```
<AxesSubplot:title={'center':'Price based on manufacturing Year of model'},
xlabel='Year', ylabel='Price'>
```

Price Vs Year VS Fuel Type

[100]:
```
# Impact of years on price
plt.figure(figsize=(10,7))
plt.title("Price Vs Year VS FuelType")
sns.lineplot(x="Year", y="Price",hue="Fuel_Type",
             data=cars)
```

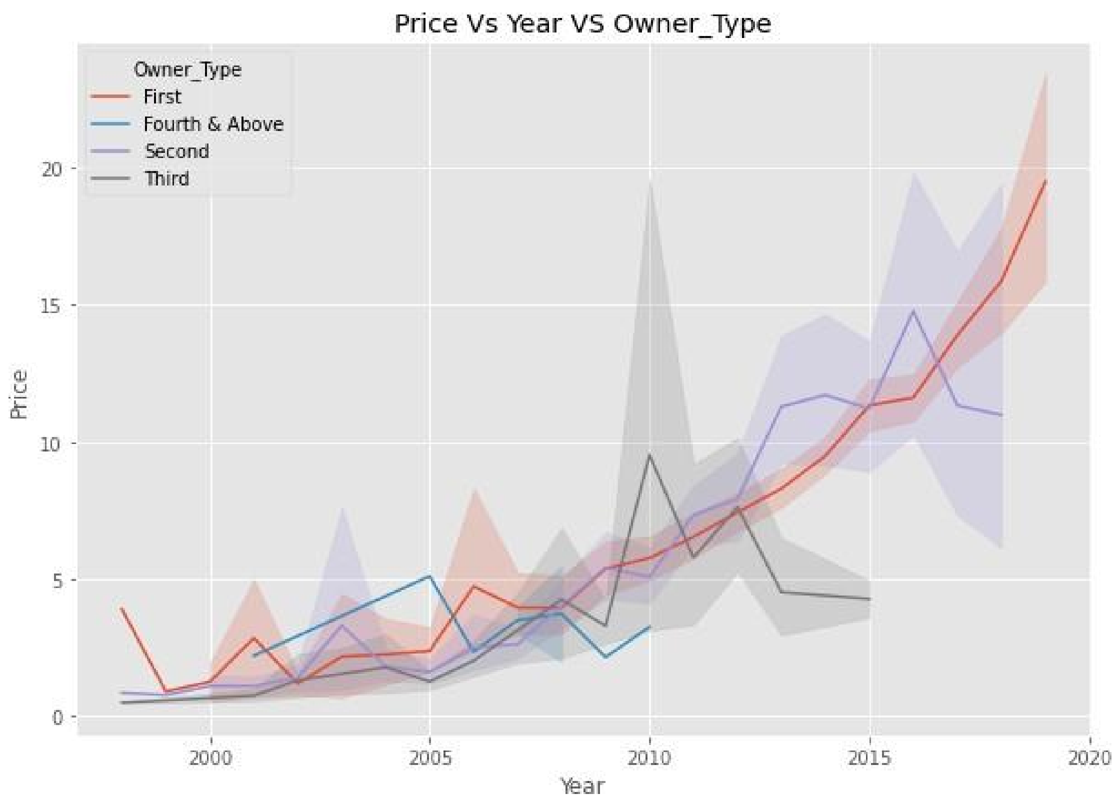
[100] : <AxesSubplot:title={'center':'Price Vs Year VS FuelType'}, xlabel='Year', ylabel='Price'>



Year Vs Price Vs Owner_Type

[101]:
```
plt.figure(figsize=(10,7))
plt.title("Price Vs Year VS Owner_Type")
sns.lineplot(x='Year', y='Price',hue='Owner_Type',
             data=cars)
```

[101] : <AxesSubplot:title={'center':'Price Vs Year VS Owner_Type'}, xlabel='Year', ylabel='Price'>



Need to check the reason for spike in price for third owner and model in 2010.

[102]:
```
cars[(cars["Owner_Type"]=="Third") & (cars["Year"].isin([2010]))].
↪sort_values(by='Price',ascending =False)
```

[102]:
```
                                 Name    Location  Year  \
S.No.
2978       Porsche Panamera 2010 2013 4S  Coimbatore  2010
5404        BMW 5 Series 2003-2012 520d        Pune  2010
3293          Toyota Innova 2.5 GX 7 STR        Pune  2010
4962              Mahindra Scorpio VLX   Bangalore  2010
3479         Volkswagen Passat 1.8 TSI MT      Mumbai  2010
1629             Maruti Swift Lxi BSIII        Pune  2010
5351  Volkswagen Polo Petrol Highline 1.2L        Pune  2010
698                 Ford Figo Diesel ZXI      Jaipur  2010

      Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage  Engine  \
S.No.
2978              42400    Petrol    Automatic      Third  8.00000    4806
5404             170000    Diesel    Automatic      Third 18.48000    1995
3293             140000    Diesel       Manual      Third 12.80000    2494
4962             144400    Diesel       Manual      Third 12.05000    2179
```

```
3479                60000  Petrol       Manual       Third  14.30000   1798
1629                54898  Petrol       Manual       Third  16.10000   1298
5351                79000  Petrol       Manual       Third  16.47000   1198
698                100002  Diesel       Manual       Third  20.00000   1399

          Power   Seats     Price  new_price_num  Ageofcar       Brand  \
S.No.
2978  394.30000 4.00000  42.91000      136.00000        11     Porsche
5404  177.00000 5.00000  12.00000       67.87000        11         BMW
3293  102.00000 7.00000   6.25000       21.08500        11      Toyota
4962  120.00000 8.00000   5.25000       10.90000        11    Mahindra
3479  160.00000 5.00000   3.50000       10.94000        11  Volkswagen
1629   88.20000 5.00000   2.50000        7.00000        11      Maruti
5351   73.90000 5.00000   2.44000       10.94000        11  Volkswagen
698    68.00000 5.00000   1.28000       11.47500        11        Ford

              Model Brand_Class
S.No.
2978   Panamera2010        High
5404        5Series        High
3293      Innova2.5         Low
4962     ScorpioVLX         Low
3479      Passat1.8        High
1629        SwiftLxi         Low
5351      PoloPetrol        High
698      FigoDiesel         Low
```

The observation is for The Porsche Panamera is expensive and luxury car so the data is valid.

[103]:
```
cars.describe()
```

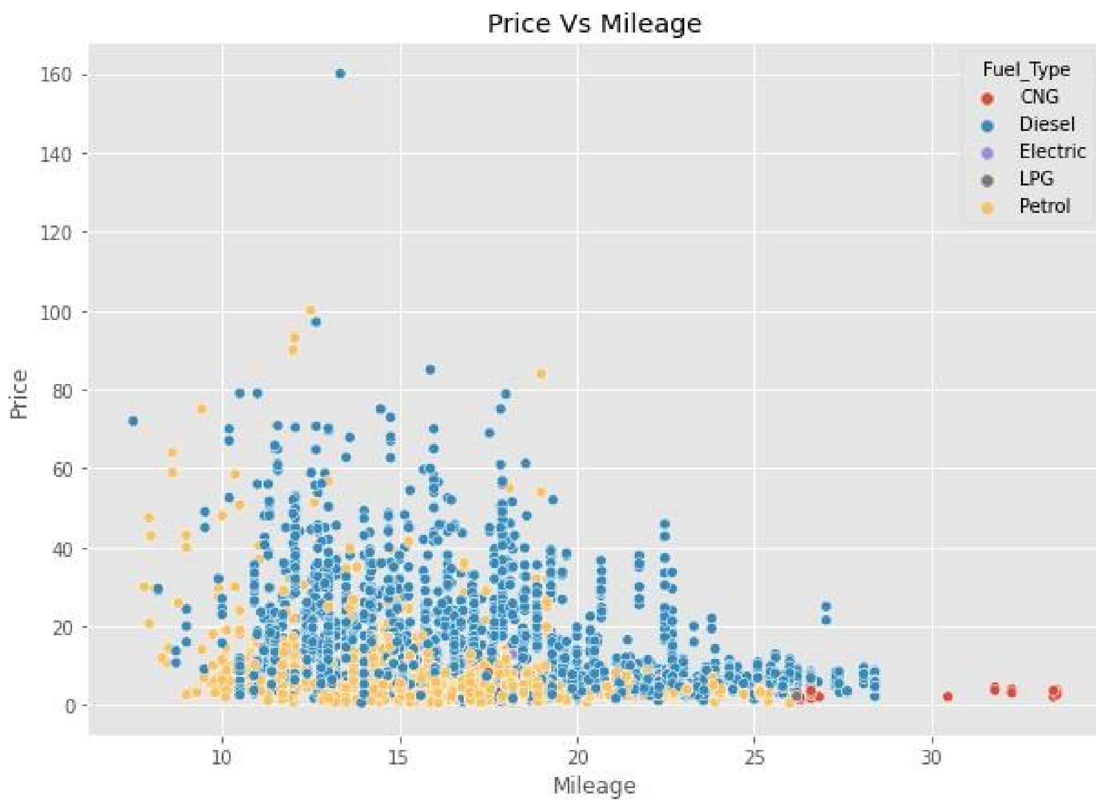
[103]:
```
            Year  Kilometers_Driven     Mileage     Engine      Power  \
count 5892.00000         5892.00000  5892.00000 5892.00000 5892.00000
mean  2013.39766        58655.30109    18.32189 1625.08452  113.00404
std      3.26869        92128.10804     4.16939  600.79522   53.52478
min   1998.00000          171.00000     7.50000   72.00000   34.20000
25%   2012.00000        33736.75000    15.30000 1198.00000   75.00000
50%   2014.00000        53000.00000    18.20000 1493.00000   93.70000
75%   2016.00000        72683.25000    21.10000 1984.00000  138.10000
max   2019.00000      6500000.00000    33.54000 5998.00000  552.00000

           Seats       Price  new_price_num    Ageofcar
count 5892.00000  5892.00000     5892.00000  5892.00000
mean     5.27834     9.59542       19.09573     7.60234
std      0.79759    11.17328       22.74291     3.26869
min      2.00000     0.44000        3.91000     2.00000
25%      5.00000     3.50000        8.07000     5.00000
```

50%	5.00000	5.75000	9.66500	7.00000
75%	5.00000	10.12000	16.45000	9.00000
max	10.00000	160.00000	375.00000	23.00000

Price Vs Mileage vs Fuel_type

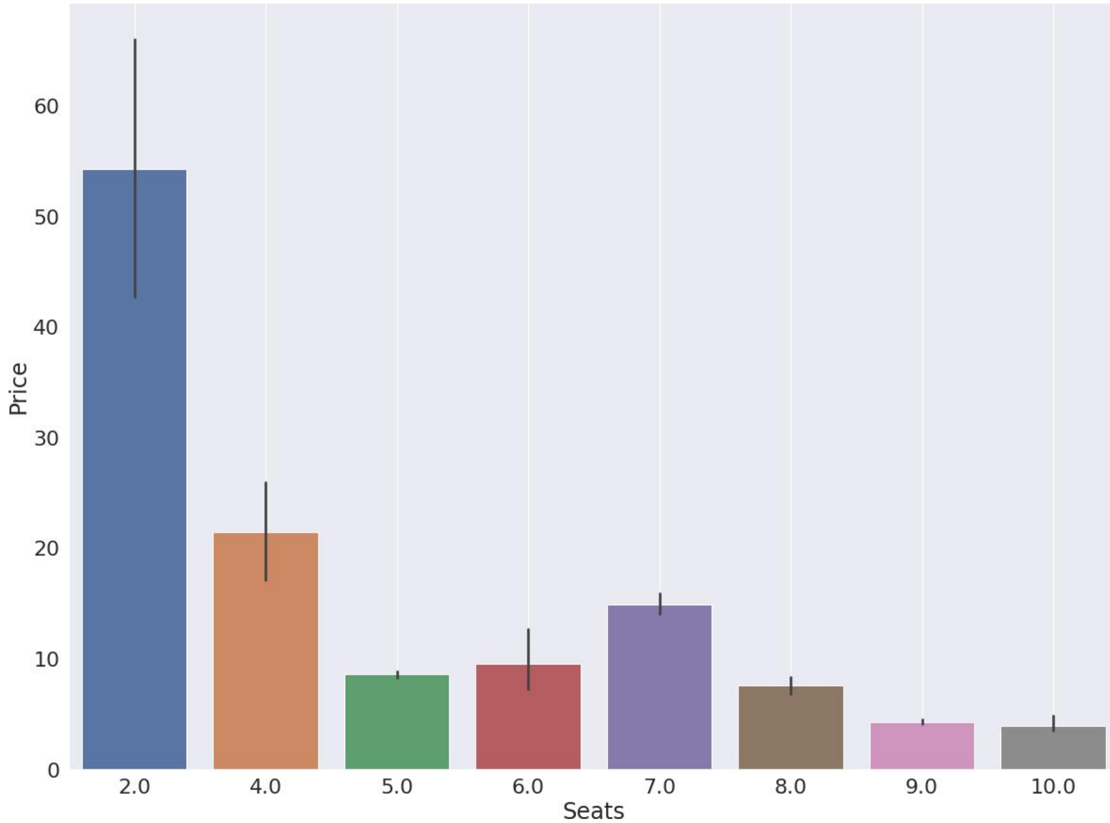
[104]:
```
# Understand relationships  between price and mileage
plt.figure(figsize=(10,7))
plt.title("Price  Vs  Mileage")
sns.scatterplot(y='Price', x='Mileage', hue='Fuel_Type', data=cars)
```

[104]: <AxesSubplot:title={'center':'Price Vs Mileage'}, xlabel='Mileage', ylabel='Price'>



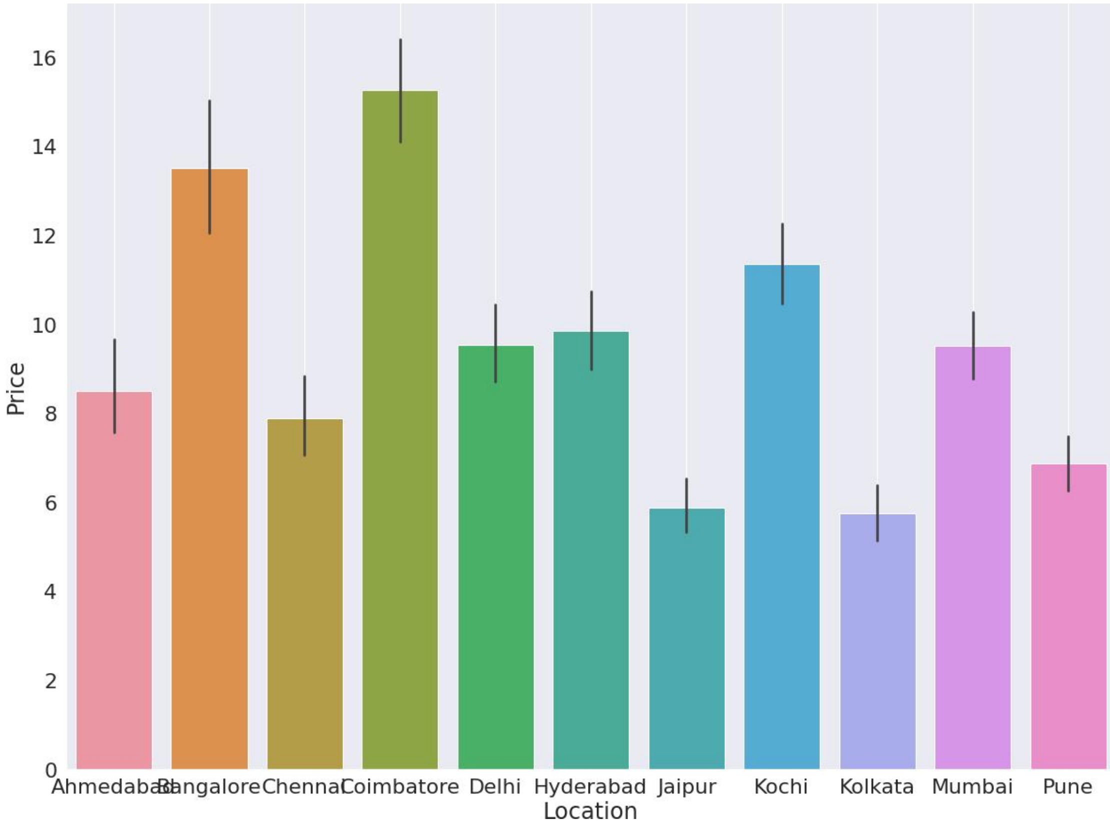
Price Vs Seat

[105]:
```
#Price and seats
plt.figure(figsize=(20,15))
sns.set(font_scale=2)
sns.barplot(x='Seats', y='Price', data=cars)
plt.grid()
```



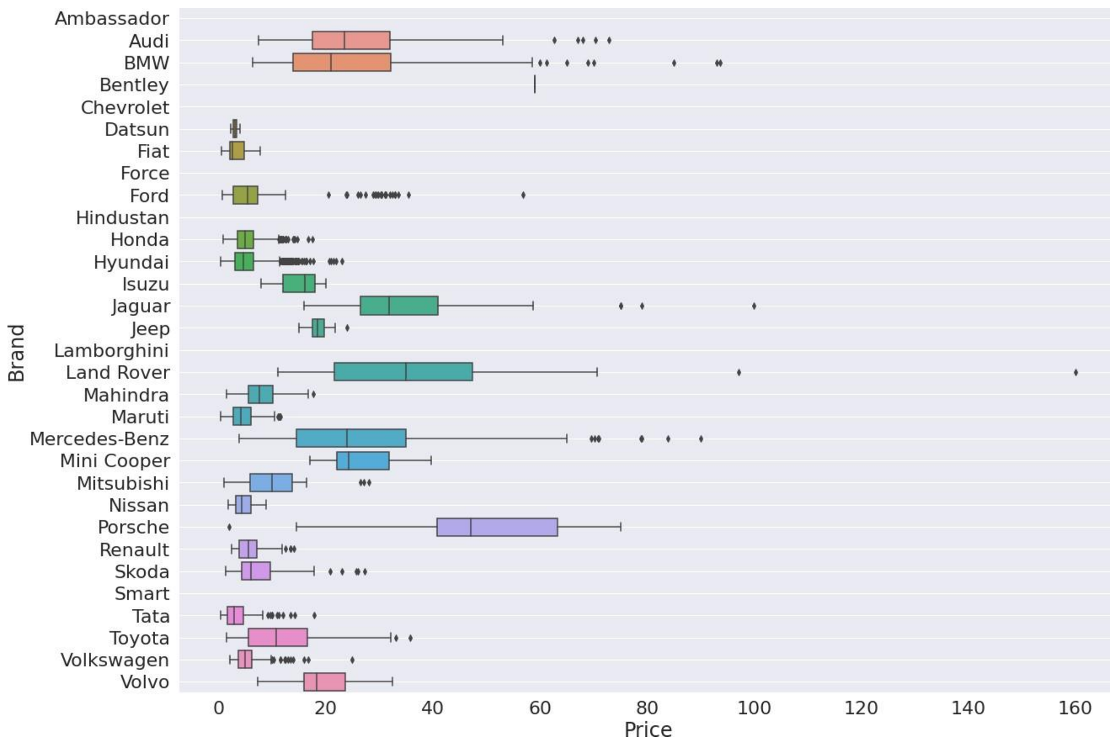
Price Vs Location

[106]:
```
#Price and LOcation
plt.figure(figsize=(20,15))
sns.set(font_scale=2)
sns.barplot(x='Location', y='Price', data=cars)
plt.grid()
```



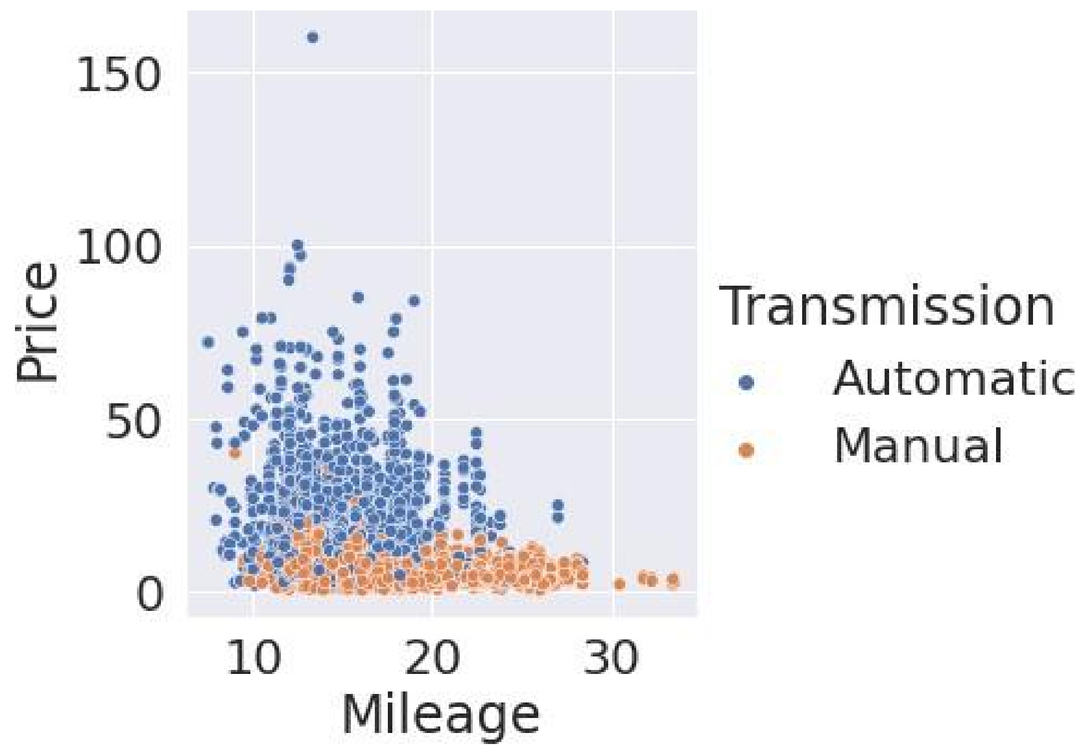
Price Vs Brand

[107]:
```
#Price and band
plt.figure(figsize=(20,15))
sns.set(font_scale=2)
sns.boxplot(x='Price', y='Brand', data=cars)
plt.grid()
```



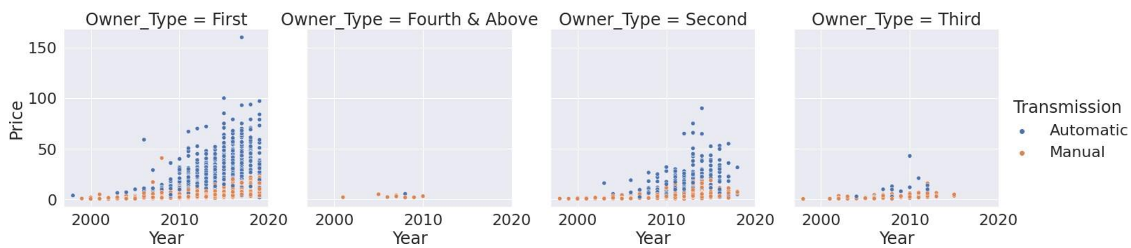
[108]:
```
sns.relplot(data=cars,
    y="Price",x="Mileage",hue="Transmission",aspect=1,height=5)
```

[108] : <seaborn.axisgrid.FacetGrid at 0x7f9389fe6590>



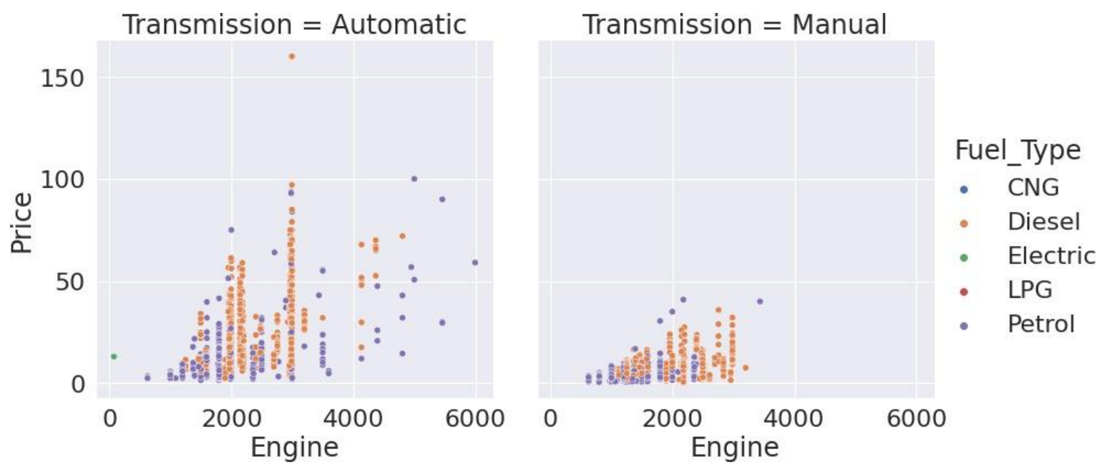
[109]:
```
sns.relplot(data=cars,␣
↪y='Price',x='Year',col='Owner_Type',hue='Transmission',aspect=1,height=5)
```

[109] : <seaborn.axisgrid.FacetGrid at 0x7f9389f3e5d0>



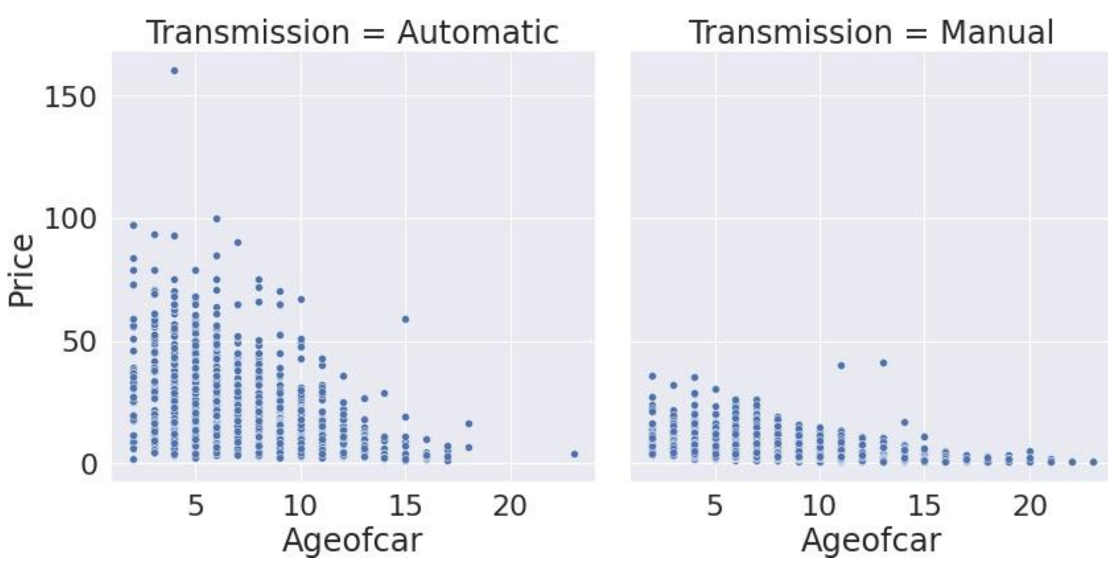
[110]:
```
sns.relplot(data=cars,␣
↪y='Price',x='Engine',col='Transmission',aspect=1,height=6,hue="Fuel_Type")
```

[110] : <seaborn.axisgrid.FacetGrid at 0x7f938a0041d0>



[111]:
```
sns.relplot(data=cars,
    y="Price",x="Ageofcar",col="Transmission",aspect=1,height=6)
```

[111]: <seaborn.axisgrid.FacetGrid at 0x7f9389f93110>



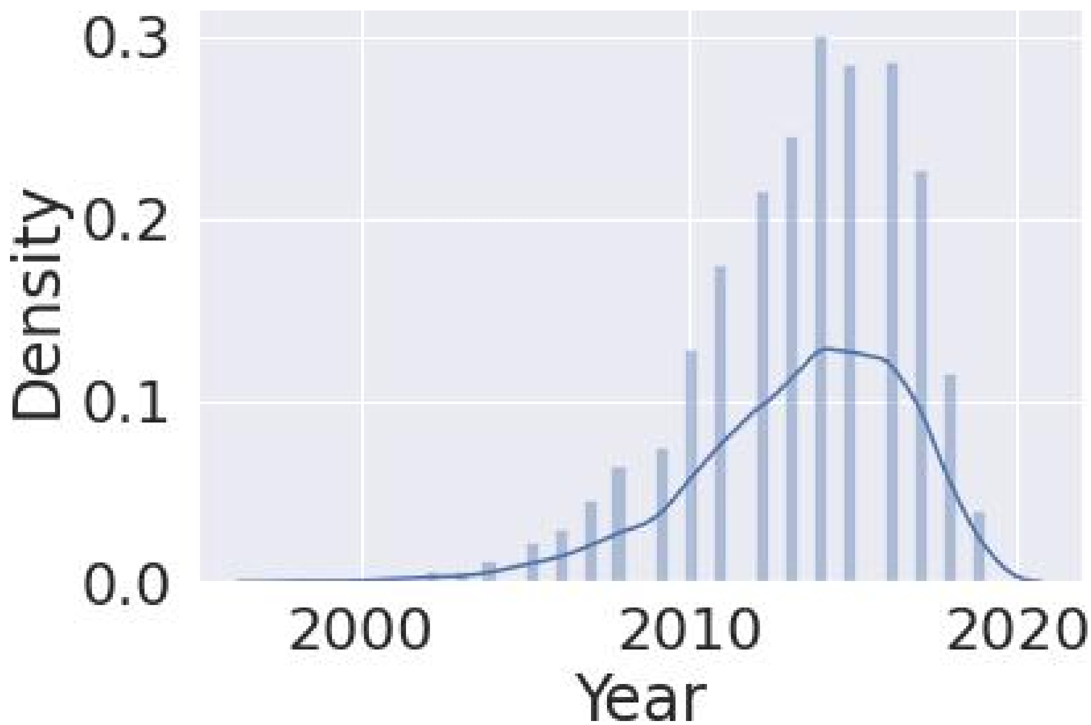
8 Insights based on EDA

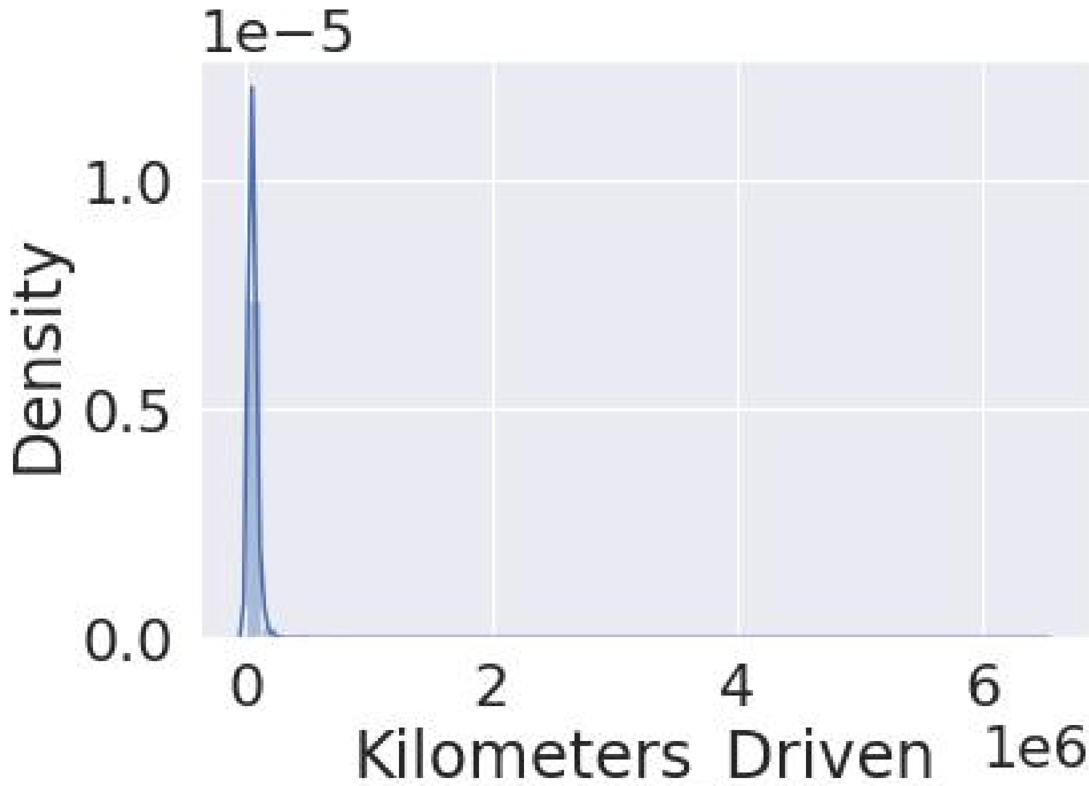
Observations

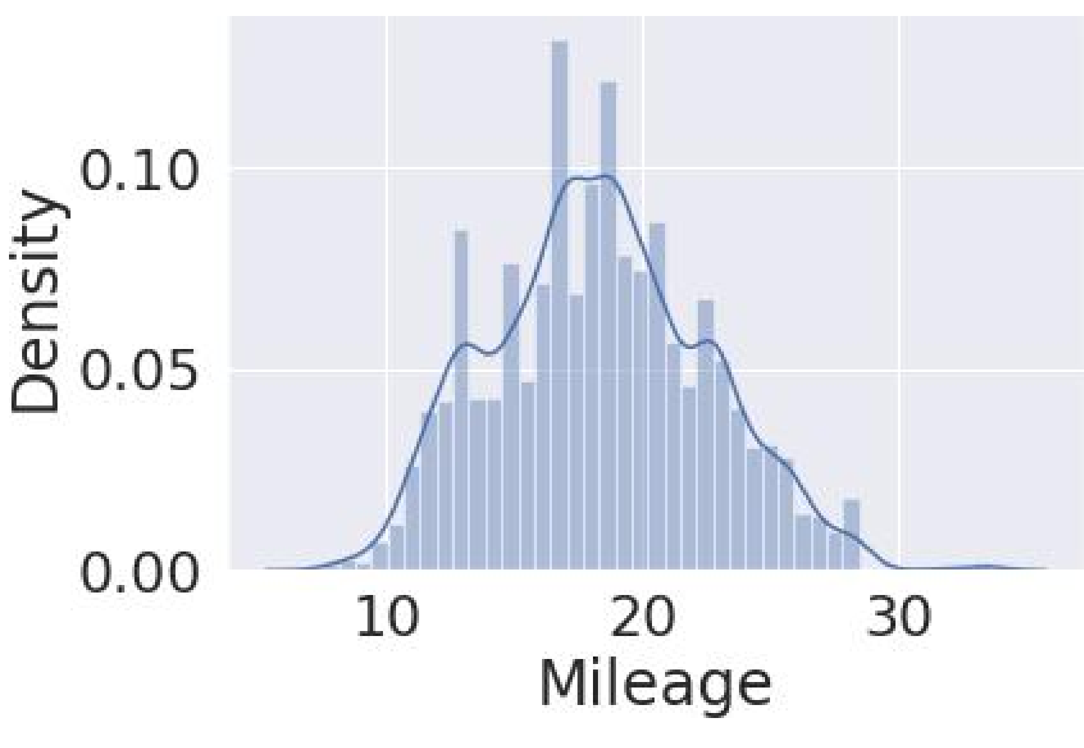
- Expensive cars are in Coimbatore and Banglore.
- 2 Seater cars are more expensive.
- Deisel Fuel type car are more expensive compared to other fuel type.
- As expected, Older model are sold cheaper compared to latest model

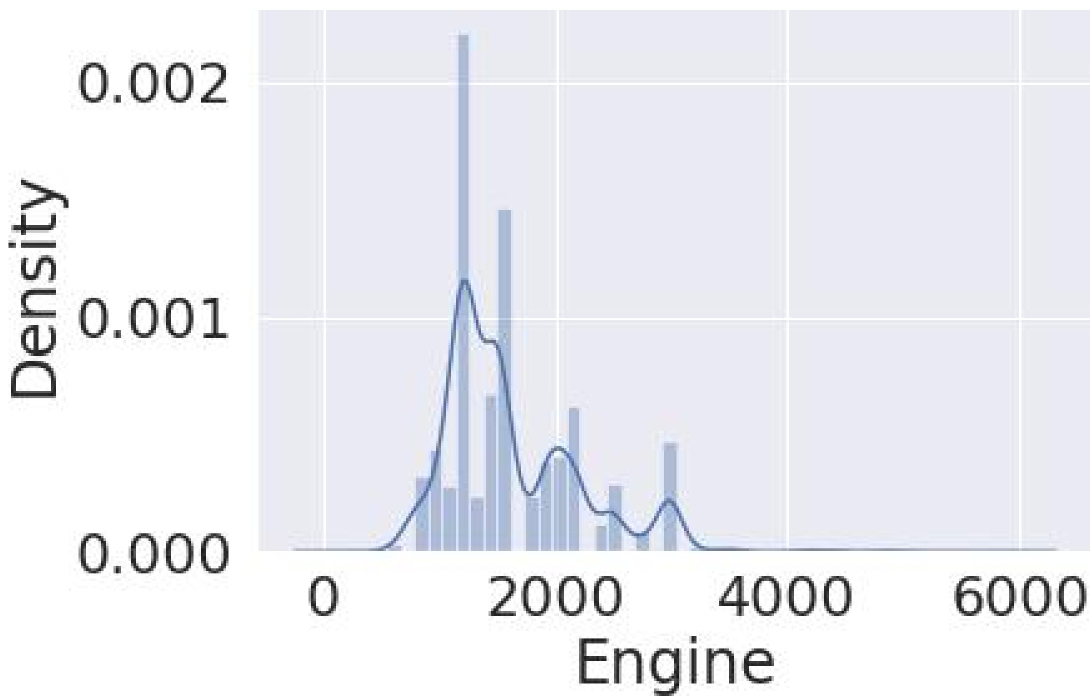
- Automatic transmission vehicle have a higher price than manual transmission vehicles.
- Vehicles with more engine capacity have higher prices.
- Price decreases as number of owner increases.
- Automatic transmission require high engine and power.
- Prices for Cars with fuel type as Deisel has increased with recent models
- Engine,Power, how old the car his, Mileage,Fuel type,location,Transmission effect the price.

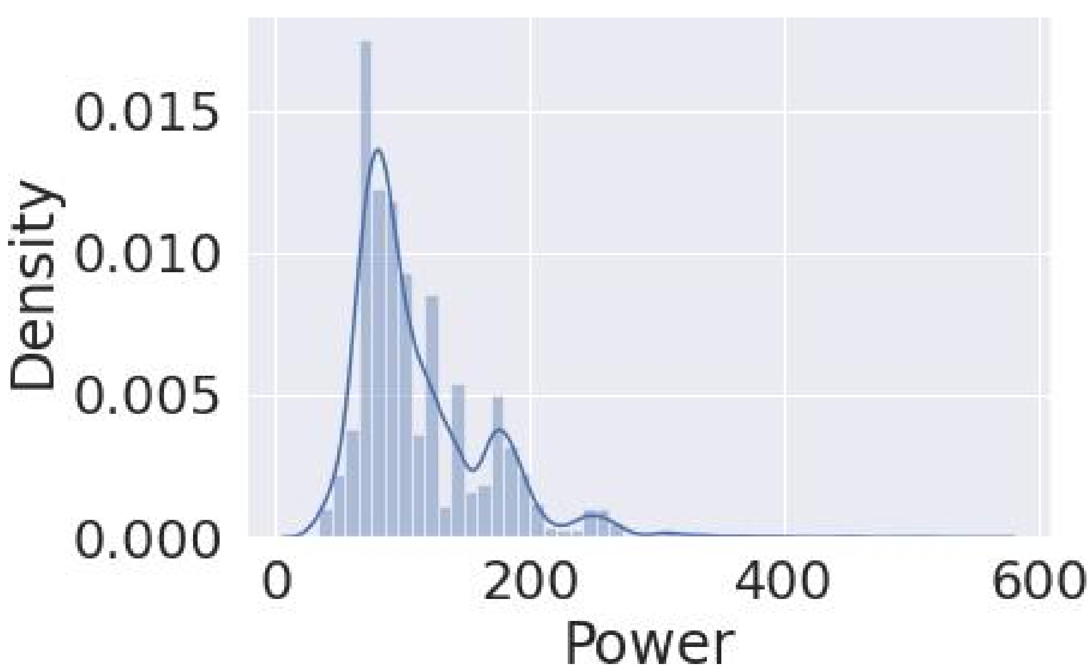
[112]:
```
# check distrubution if skewed. If distrubution is skewed , it is advice to use log transform
cols_to_log = cars.select_dtypes(include=np.number).columns.tolist()
for colname in cols_to_log:
    sns.distplot(cars[colname], kde=True)
    plt.show()
```

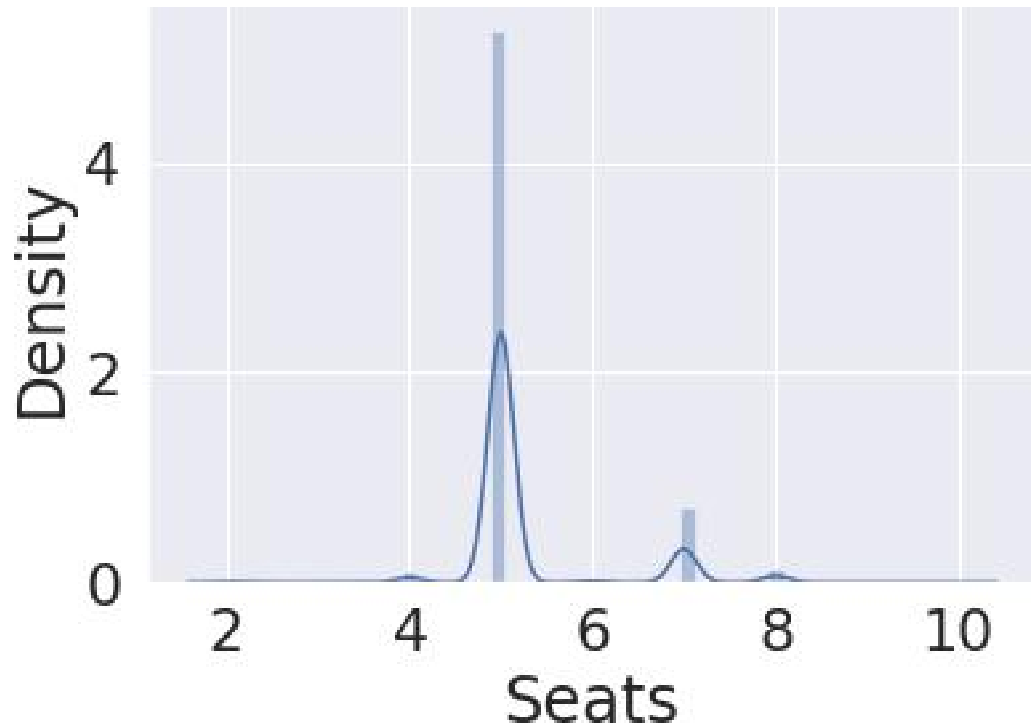


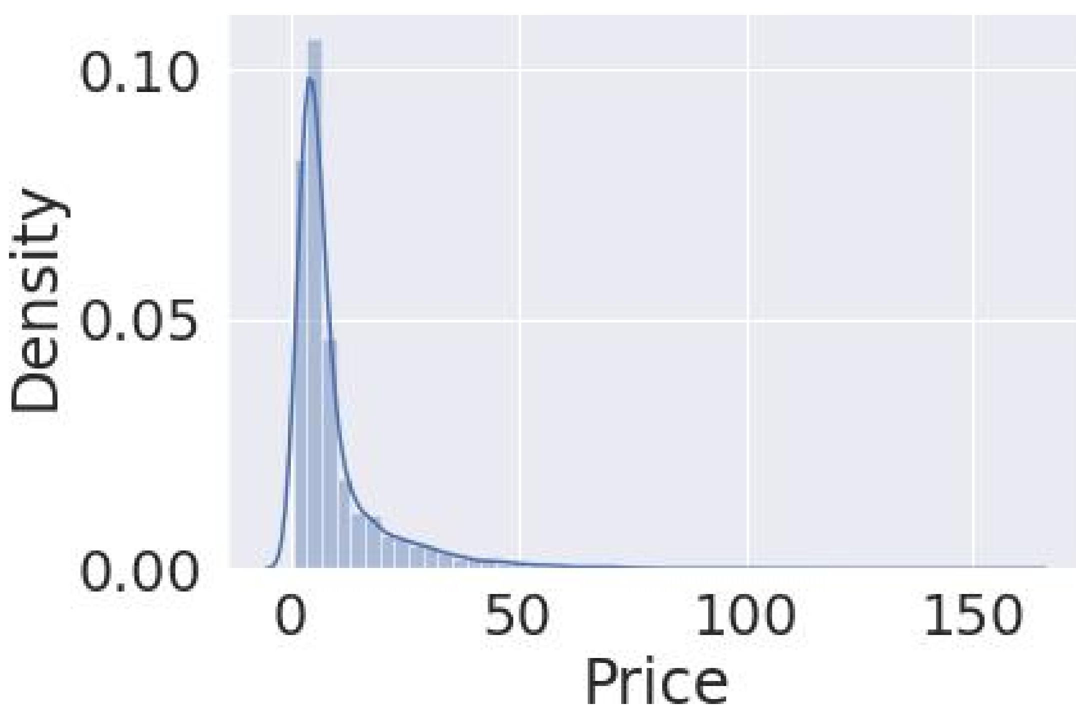

1e−5
1.0
0.5
0.0
Density
0
2
4
6
Kilometers_Driven
1e6

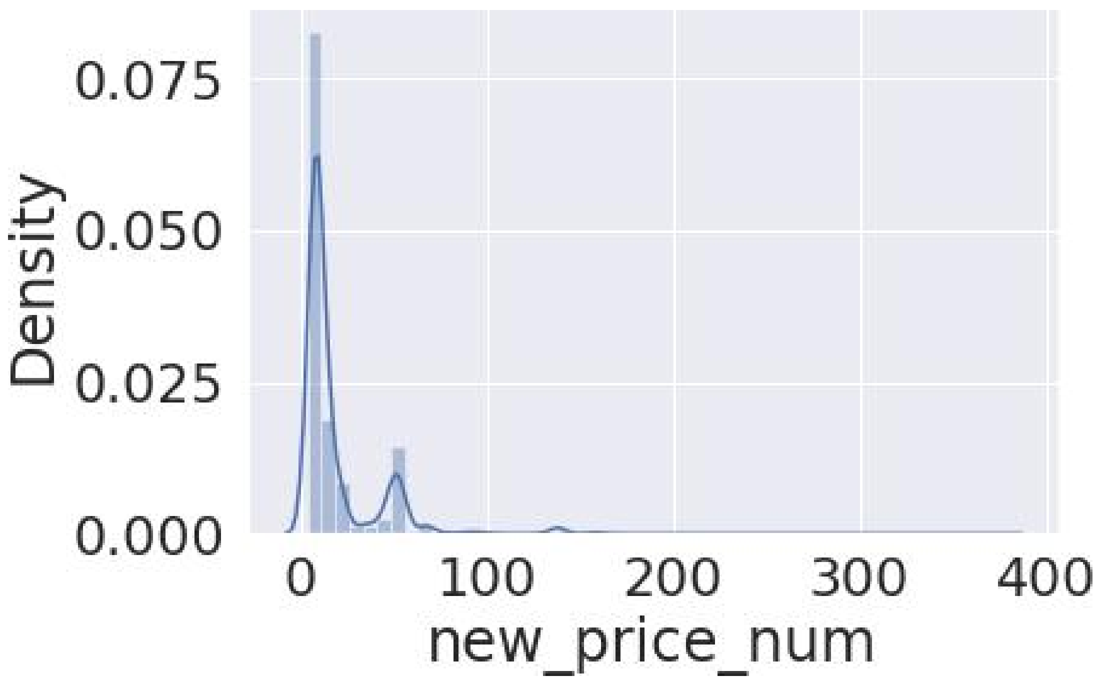

0.10
0.05
0.00
Density
10
20
30
Mileage

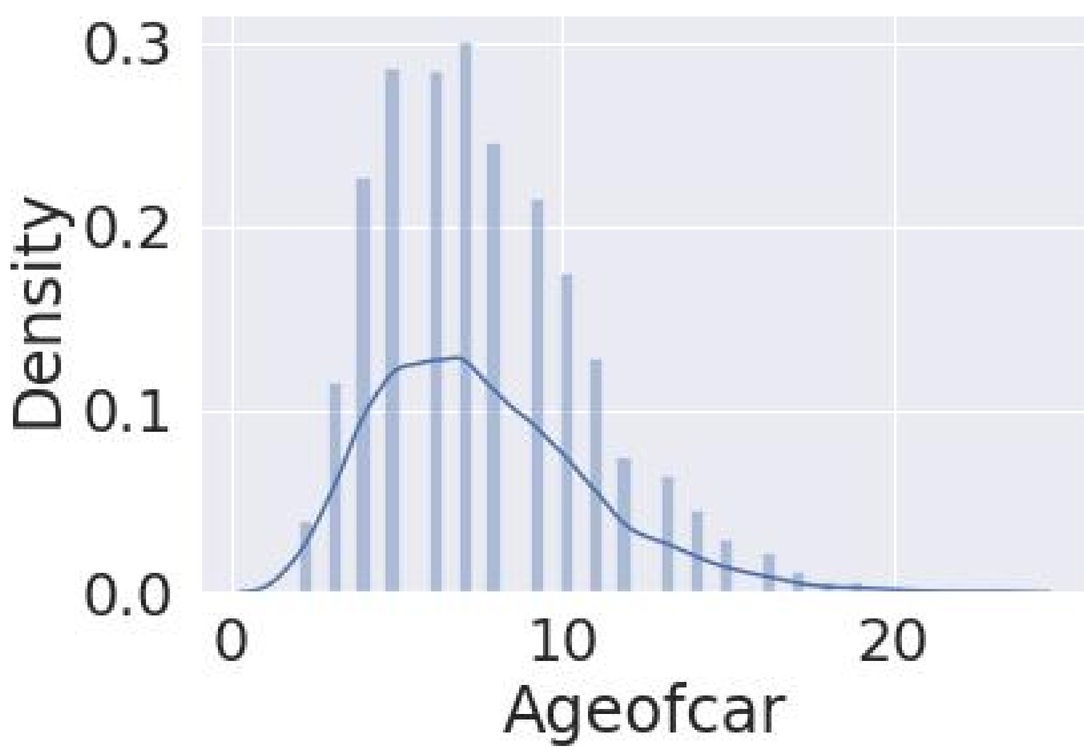

0.002
0.001
0.000
Density
0
2000
4000
6000
Engine


0.015
0.010
0.005
0.000
Density
0
200
400
600
Power


Density
0
2
4
2
4
6
8
10
Seats


Density
0.00
0.05
0.10
0
50
100
150
Price





Distrubtions are right skewed , using Log transform can help in normalization

[113]:
```
def Perform_log_transform(df,col_log):
    """#Perform Log Transformation of dataframe , and list of columns """
    for colname in col_log:
        df[colname + '_log'] = np.log(df[colname])
    #df.drop(col_log, axis=1, inplace=True)
    df.info()
```

[114]:
```
#This needs to be done before the data is split
Perform_log_transform(cars,['Kilometers_Driven','Price'])
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5892 entries, 0 to 6017
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Name                   5892 non-null   object
 1   Location               5892 non-null   category
 2   Year                   5892 non-null   int64
 3   Kilometers_Driven      5892 non-null   int64
 4   Fuel_Type              5892 non-null   category
 5   Transmission           5892 non-null   category
 6   Owner_Type             5892 non-null   category
 7   Mileage                5892 non-null   float64
 8   Engine                 5892 non-null   int64
 9   Power                  5892 non-null   float64
 10  Seats                  5892 non-null   float64
 11  Price                  5892 non-null   float64
 12  new_price_num          5892 non-null   float64
 13  Ageofcar               5892 non-null   int64
 14  Brand                  5892 non-null   category
 15  Model                  5892 non-null   object
 16  Brand_Class            5892 non-null   category
 17  Kilometers_Driven_log  5892 non-null   float64
 18  Price_log              5892 non-null   float64
dtypes: category(6), float64(7), int64(4), object(2)
memory usage: 810.3+ KB
```

[115]:
```
cars.drop(['Name','Model','Year','Brand','new_price_num'],axis=1,inplace=True)
```

[116]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5892 entries, 0 to 6017
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Location               5892 non-null   category
 1   Kilometers_Driven      5892 non-null   int64
```

```
 2   Fuel_Type              5892 non-null   category
 3   Transmission           5892 non-null   category
 4   Owner_Type             5892 non-null   category
 5   Mileage                5892 non-null   float64
 6   Engine                 5892 non-null   int64
 7   Power                  5892 non-null   float64
 8   Seats                  5892 non-null   float64
 9   Price                  5892 non-null   float64
 10  Ageofcar               5892 non-null   int64
 11  Brand_Class            5892 non-null   category
 12  Kilometers_Driven_log  5892 non-null   float64
 13  Price_log              5892 non-null   float64
dtypes: category(5), float64(6), int64(3)
memory usage: 619.1 KB
```

9 Model Building

[117]:
```
X = cars.drop(["Price", "Price_log"], axis=1)
y = cars[["Price_log", "Price"]]
```

9.0.1 Creating dummy variables

[118]:
```
def encode_cat_vars(x):
    x = pd.get_dummies(
        x,
        columns=x.select_dtypes(include=["object", "category"]).columns.
↪tolist(),
        drop_first=True,
    )
    return x
```

[119]:
```
#Dummy variable creation is done before spliting the data , so all the␣
↪different categories are covered
#create dummy  variable
X = encode_cat_vars(X)
X.head()
```

[119]:
```
       Kilometers_Driven  Mileage  Engine     Power   Seats  Ageofcar  \
S.No.
0                  72000 26.60000     998  58.16000 5.00000        11
1                  41000 19.67000    1582 126.20000 5.00000         6
2                  46000 18.20000    1199  88.70000 5.00000        10
3                  87000 20.77000    1248  88.76000 7.00000         9
4                  40670 15.20000    1968 140.80000 5.00000         8

       Kilometers_Driven_log  Location_Bangalore  Location_Chennai  \
```

```
S.No.
0                    11.18442                   0                  0
1                    10.62133                   0                  0
2                    10.73640                   0                  1
3                    11.37366                   0                  1
4                    10.61325                   0                  0

       Location_Coimbatore  ...  Location_Pune  Fuel_Type_Diesel  \
S.No.                       ...
0                        0  ...              0                 0
1                        0  ...              1                 1
2                        0  ...              0                 0
3                        0  ...              0                 1
4                        1  ...              0                 1

       Fuel_Type_Electric  Fuel_Type_LPG  Fuel_Type_Petrol  \
S.No.
0                       0              0                 0
1                       0              0                 0
2                       0              0                 1
3                       0              0                 0
4                       0              0                 0

       Transmission_Manual  Owner_Type_Fourth & Above  Owner_Type_Second  \
S.No.
0                        1                          0                  0
1                        1                          0                  0
2                        1                          0                  0
3                        1                          0                  0
4                        0                          0                  1

       Owner_Type_Third  Brand_Class_Low
S.No.
0                     0                1
1                     0                1
2                     0                1
3                     0                1
4                     0                0

[5 rows x 26 columns]
```

[120]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)
X_train.reset_index()
print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("y_train:",y_train.shape)
```

```
print("y_test:",y_test.shape)
```

```
X_train: (4124, 26)
X_test: (1768, 26)
y_train: (4124, 2)
y_test: (1768, 2)
```

[121]:
```
# Statsmodel api does not add a constant by default. We need to add it␣
↪explicitly.
X_train = sm.add_constant(X_train)
# Add constant to test data
X_test = sm.add_constant(X_test)


def build_ols_model(train):
    # Create the model
    olsmodel = sm.OLS(y_train["Price_log"], train)
    return olsmodel.fit()
```

[122]:
```
#fit statmodel
olsmodel1 = build_ols_model(X_train)
print(olsmodel1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Price_log   R-squared:                       0.894
Model:                            OLS   Adj. R-squared:                  0.893
Method:                 Least Squares   F-statistic:                     1322.
Date:                Fri, 11 Jun 2021   Prob (F-statistic):               0.00
Time:                        05:13:24   Log-Likelihood:                -658.35
No. Observations:                4124   AIC:                             1371.
Df Residuals:                    4097   BIC:                             1541.
Df Model:                          26
Covariance Type:            nonrobust
================================================================================
=============
                                  coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
--------------
const                           2.7170      0.145     18.800      0.000
2.434       3.000
Kilometers_Driven           -3.707e-07   1.94e-07     -1.915      0.056
-7.5e-07     8.8e-09
Mileage                        -0.0127      0.002     -6.383      0.000
-0.017      -0.009
Engine                          0.0002   2.24e-05      9.230      0.000
```

					0.000	0.000
Power	0.0065	0.000	29.525	0.000	0.006	0.007
Seats	0.0273	0.008	3.387	0.001	0.012	0.043
Ageofcar	−0.1130	0.002	−57.908	0.000	−0.117	−0.109
Kilometers_Driven_log	−0.0610	0.011	−5.427	0.000	−0.083	−0.039
Location_Bangalore	0.1580	0.030	5.332	0.000	0.100	0.216
Location_Chennai	0.0151	0.028	0.542	0.588	−0.040	0.070
Location_Coimbatore	0.0969	0.027	3.592	0.000	0.044	0.150
Location_Delhi	−0.0479	0.027	−1.773	0.076	−0.101	0.005
Location_Hyderabad	0.1131	0.026	4.320	0.000	0.062	0.164
Location_Jaipur	−0.0654	0.029	−2.295	0.022	−0.121	−0.010
Location_Kochi	−0.0282	0.027	−1.052	0.293	−0.081	0.024
Location_Kolkata	−0.2385	0.028	−8.651	0.000	−0.292	−0.184
Location_Mumbai	−0.0580	0.026	−2.219	0.027	−0.109	−0.007
Location_Pune	−0.0436	0.027	−1.615	0.106	−0.096	0.009
Fuel_Type_Diesel	0.1786	0.046	3.854	0.000	0.088	0.269
Fuel_Type_Electric	1.1521	0.208	5.551	0.000	0.745	1.559
Fuel_Type_LPG	−0.0062	0.118	−0.053	0.958	−0.237	0.224
Fuel_Type_Petrol	−0.0947	0.047	−2.003	0.045	−0.187	−0.002
Transmission_Manual	−0.2272	0.014	−16.257	0.000	−0.255	−0.200
Owner_Type_Fourth & Above	0.1693	0.143	1.183	0.237	−0.111	0.450
Owner_Type_Second	−0.0886	0.013	−6.789	0.000	−0.114	−0.063
Owner_Type_Third	−0.1308	0.038	−3.451	0.001	−0.205	−0.056
Brand_Class_Low	−0.2439	0.015	−16.451	0.000	−0.273	−0.215

==

```
Omnibus:                     1255.330   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            21661.083
Skew:                          -0.994   Prob(JB):                         0.00
Kurtosis:                      14.050   Cond. No.                     3.31e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.31e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- 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 89%
- The model is not an underfitting or overfitting model.
- To be able to make statistical inferences from our model, we will have to test that the linear regression assumptions are followed.
- Before we move on to assumption testing, we'll do a quick performance check on the test data.

[123]:
```python
import math

# RMSE
def rmse(predictions, targets):
    return np.sqrt(((targets - predictions) ** 2).mean())


# MAPE
def mape(predictions, targets):
    return np.mean(np.abs((targets - predictions)) / targets) * 100


# MAE
def mae(predictions, targets):
    return np.mean(np.abs((targets - predictions)))


# Model Performance on test and train data
def model_pref(olsmodel, x_train, x_test):

    # Insample Prediction
    y_pred_train_pricelog = olsmodel.predict(x_train)
    y_pred_train_Price = y_pred_train_pricelog.apply(math.exp)
    y_train_Price = y_train["Price"]

    # Prediction on test data
```

```
    y_pred_test_pricelog = olsmodel.predict(x_test)
    y_pred_test_Price = y_pred_test_pricelog.apply(math.exp)
    y_test_Price = y_test["Price"]

    print(
        pd.DataFrame(
            {
                "Data": ["Train", "Test"],
                "RMSE": [
                    rmse(y_pred_train_Price, y_train_Price),
                    rmse(y_pred_test_Price, y_test_Price),
                ],
                "MAE": [
                    mae(y_pred_train_Price, y_train_Price),
                    mae(y_pred_test_Price, y_test_Price),
                ],
                "MAPE": [
                    mape(y_pred_train_Price, y_train_Price),
                    mape(y_pred_test_Price, y_test_Price),
                ],
            }
        )
    )

# Checking model performance
model_pref(olsmodel1, X_train, X_test)  # High Overfitting.
```

```
    Data     RMSE      MAE      MAPE
0  Train  6.55093  2.28228  23.33998
1   Test  7.44709  2.51399  23.60918
```

- Root Mean Squared Error of train and test data is not different, indicating that our model is not overfitting the train data.
- Mean Absolute Error indicates that our current model is able to predict used cars prices within mean error of 2.5 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 ~23% on the test data.

10 Test Assumptions

10.1 Checking the Linear Regression Assumptions

1. **No Multicollinearity**
2. **Mean of residuals should be 0**

3. **No Heteroscedasticity**
4. **Linearity of variables**
5. **Normality of error terms**

10.1.1 Checking Assumption 1: No Multicollinearity

We will use VIF, to check if there is multicollinearity in the data.

Features having a VIF score >5 will be dropped/treated till all the features have a VIF score <5

[124]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor


def checking_vif(train):
    vif = pd.DataFrame()
    vif["feature"] = train.columns

    # calculating VIF for each feature
    vif["VIF"] = [
        variance_inflation_factor(train.values, i) for i in range(len(train.
 ↪columns))
    ]
    return vif
```

[125]:
```
# Check VIF
print(checking_vif(X_train))
```

```
                 feature         VIF
0                  const  1062.04981
1      Kilometers_Driven     2.93017
2                Mileage     3.46285
3                 Engine     8.92501
4                  Power     6.92472
5                  Seats     2.09092
6               Ageofcar     2.07487
7  Kilometers_Driven_log     3.34652
8     Location_Bangalore     2.41019
9       Location_Chennai     3.01294
10   Location_Coimbatore     3.44626
11        Location_Delhi     3.20233
12    Location_Hyderabad     3.80468
13       Location_Jaipur     2.70344
14        Location_Kochi     3.51621
15      Location_Kolkata     3.04719
16       Location_Mumbai     3.98210
17         Location_Pune     3.38664
18      Fuel_Type_Diesel    27.15677
19    Fuel_Type_Electric     1.06173
20         Fuel_Type_LPG     1.19256
```

```
21             Fuel_Type_Petrol   28.13755
22          Transmission_Manual    2.03453
23  Owner_Type_Fourth & Above     1.00991
24            Owner_Type_Second    1.16822
25             Owner_Type_Third    1.09932
26              Brand_Class_Low    2.00537
```

Let us now remove multicollinearity from the model. Engine,power,Fuel_type have high mutlicollinearity.but fuel_type is an important feature in model prediction. So will remove engine.

[126]:
```python
X_train1=X_train.drop(['Engine'],axis=1)
X_test1=X_test.drop(['Engine'],axis=1)
olsmodel2= build_ols_model(X_train1)

print(olsmodel2.summary())

# Checking model performance
model_pref(olsmodel2, X_train1, X_test1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Price_log   R-squared:                       0.891
Model:                            OLS   Adj. R-squared:                  0.891
Method:                 Least Squares   F-statistic:                     1344.
Date:                Fri, 11 Jun 2021   Prob (F-statistic):               0.00
Time:                        05:13:30   Log-Likelihood:                -700.79
No. Observations:                4124   AIC:                             1454.
Df Residuals:                    4098   BIC:                             1618.
Df Model:                          25
Covariance Type:            nonrobust
================================================================================
=============
                                 coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-----------------
const                          2.9248      0.144     20.281      0.000
2.642       3.208
Kilometers_Driven          -2.441e-07   1.95e-07     -1.252      0.211
-6.27e-07    1.38e-07
Mileage                       -0.0205      0.002    -11.339      0.000
-0.024      -0.017
Power                          0.0081      0.000     54.814      0.000
0.008       0.008
Seats                          0.0530      0.008      6.929      0.000
0.038       0.068
Ageofcar                      -0.1123      0.002    -56.990      0.000
-0.116      -0.108
```

Kilometers_Driven_log	−0.0655	0.011	−5.779	0.000	−0.088	−0.043
Location_Bangalore	0.1572	0.030	5.249	0.000	0.098	0.216
Location_Chennai	0.0220	0.028	0.780	0.436	−0.033	0.077
Location_Coimbatore	0.0990	0.027	3.635	0.000	0.046	0.152
Location_Delhi	−0.0451	0.027	−1.653	0.098	−0.099	0.008
Location_Hyderabad	0.1166	0.026	4.409	0.000	0.065	0.168
Location_Jaipur	−0.0637	0.029	−2.212	0.027	−0.120	−0.007
Location_Kochi	−0.0268	0.027	−0.989	0.323	−0.080	0.026
Location_Kolkata	−0.2366	0.028	−8.495	0.000	−0.291	−0.182
Location_Mumbai	−0.0569	0.026	−2.155	0.031	−0.109	−0.005
Location_Pune	−0.0413	0.027	−1.513	0.130	−0.095	0.012
Fuel_Type_Diesel	0.1879	0.047	4.013	0.000	0.096	0.280
Fuel_Type_Electric	1.0838	0.210	5.172	0.000	0.673	1.495
Fuel_Type_LPG	−0.0573	0.119	−0.483	0.629	−0.290	0.175
Fuel_Type_Petrol	−0.1431	0.047	−3.015	0.003	−0.236	−0.050
Transmission_Manual	−0.2226	0.014	−15.771	0.000	−0.250	−0.195
Owner_Type_Fourth & Above	0.1633	0.145	1.129	0.259	−0.120	0.447
Owner_Type_Second	−0.0914	0.013	−6.927	0.000	−0.117	−0.066
Owner_Type_Third	−0.1374	0.038	−3.588	0.000	−0.212	−0.062
Brand_Class_Low	−0.2296	0.015	−15.410	0.000	−0.259	−0.200

==

Omnibus:	1160.543	Durbin-Watson:	1.999
Prob(Omnibus):	0.000	Jarque-Bera (JB):	23002.735
Skew:	−0.843	Prob(JB):	0.00
Kurtosis:	14.446	Cond. No.	3.31e+06

==

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.31e+06. This might indicate that there are strong multicollinearity or other numerical problems.

	Data	RMSE	MAE	MAPE
0	Train	6.11552	2.26577	23.45353
1	Test	6.47005	2.38709	23.45102

[127]:
```
print(checking_vif(X_train1))
```

	feature	VIF
0	const	1036.27774
1	Kilometers_Driven	2.91546
2	Mileage	2.82471
3	Power	3.00378
4	Seats	1.84216
5	Ageofcar	2.07123
6	Kilometers_Driven_log	3.34005
7	Location_Bangalore	2.41017
8	Location_Chennai	3.01080
9	Location_Coimbatore	3.44601
10	Location_Delhi	3.20194
11	Location_Hyderabad	3.80389
12	Location_Jaipur	2.70333
13	Location_Kochi	3.51610
14	Location_Kolkata	3.04702
15	Location_Mumbai	3.98202
16	Location_Pune	3.38634
17	Fuel_Type_Diesel	27.14402
18	Fuel_Type_Electric	1.06038
19	Fuel_Type_LPG	1.18991
20	Fuel_Type_Petrol	27.79160
21	Transmission_Manual	2.03185
22	Owner_Type_Fourth & Above	1.00989
23	Owner_Type_Second	1.16763
24	Owner_Type_Third	1.09893
25	Brand_Class_Low	1.98326

We have removed multicollinearity from the data now.Fuel_Type variables are showing high vif because most cars are either diesel and petrol. These two features are correlated with each other.

We will not drop this variable from the model because this will not affect the interpretation of other features in the model.

10.1.2 Checking Assumption 2: Mean of residuals should be 0

[128]:
```
residuals = olsmodel2.resid
np.mean(residuals)
```

[128]:
```
1.1382743007670337e-12
```

Mean of redisuals is very close to 0. The second assumption is also satisfied.

10.1.3 Checking Assumption 3: No Heteroscedasticity

- Homoscedacity - If the residuals are symmetrically distributed across the regression line , then the data is said to homoscedastic.
- Heteroscedasticity- - If the residuals are not symmetrically distributed across the regression line, then the data is said to be heteroscedastic. In this case the residuals can form a funnel shape or any other non symmetrical shape.

We'll use `Goldfeldquandt Test` to test the following hypothesis

Null hypothesis : Residuals are homoscedastic Alternate hypothesis : Residuals have hetroscedasticity

alpha = 0.05

[129]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(residuals, X_train1)
lzip(name, test)
```

[129]:
```
[('F statistic', 0.8809177630408274), ('p-value', 0.9978790999679913)]
```

Since p-value > 0.05 we cannot reject the Null Hypothesis that the residuals are homoscedastic.

Assumptions 3 is also satisfied by our olsmodel2

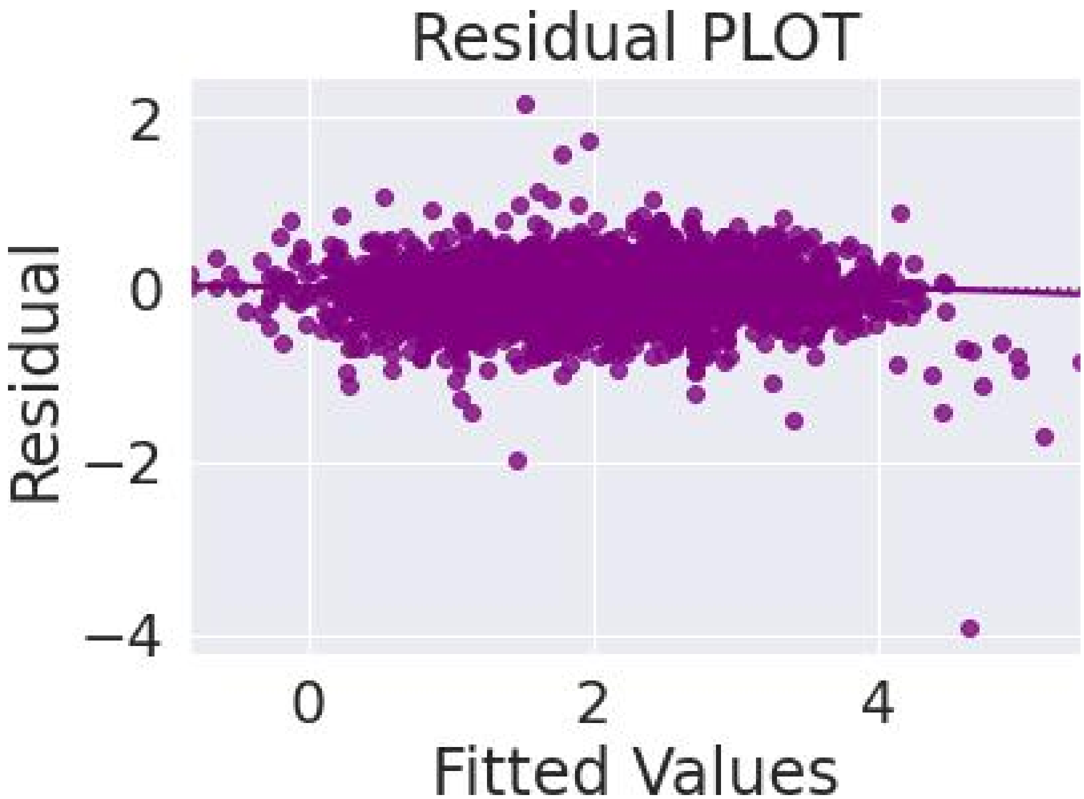
10.1.4 Checking Assumption 4: Linearity of variables

Predictor variables must have a linear relation with the dependent variable.

To test the assumption, we'll plot residuals and fitted values on a plot and ensure that residuals do not form a strong pattern. They should be randomly and uniformly scattered on the x axis.

[130]:
```
# predicted values
fitted = olsmodel2.fittedvalues

#     sns.set_style("whitegrid")
sns.residplot(fitted, residuals, color="purple", lowess=True)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
```

```
plt.title("Residual PLOT")
plt.show()
```



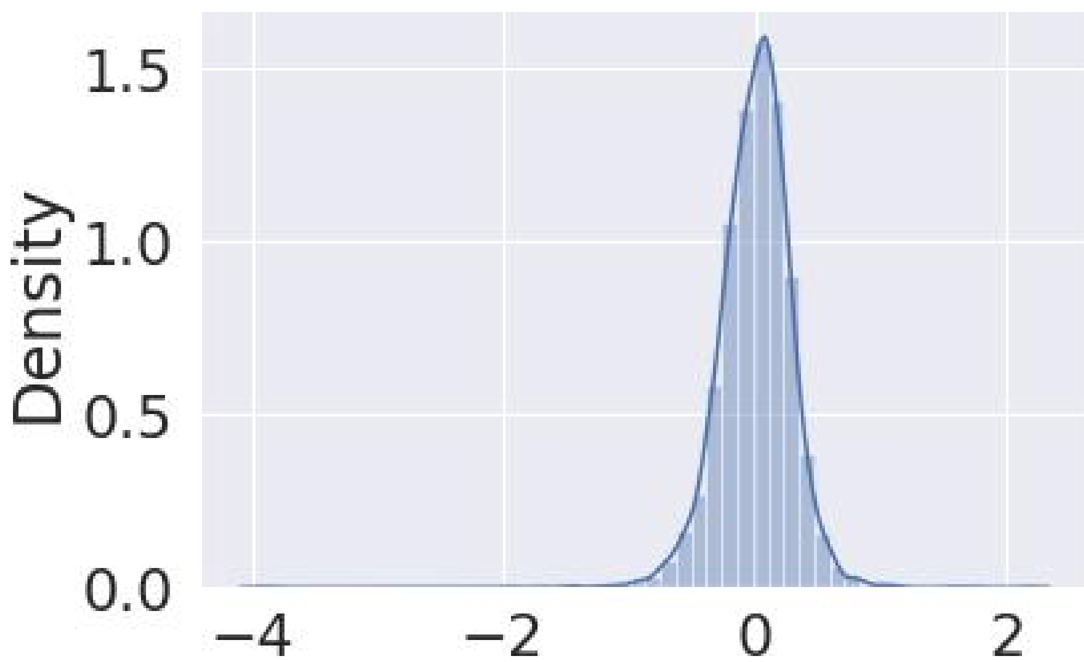
Assumptions 4 is satisfied by our olsmodel2. There is no pattern in the residual vs fitted values plot.

10.1.5 Checking Assumption 5: Normality of error terms

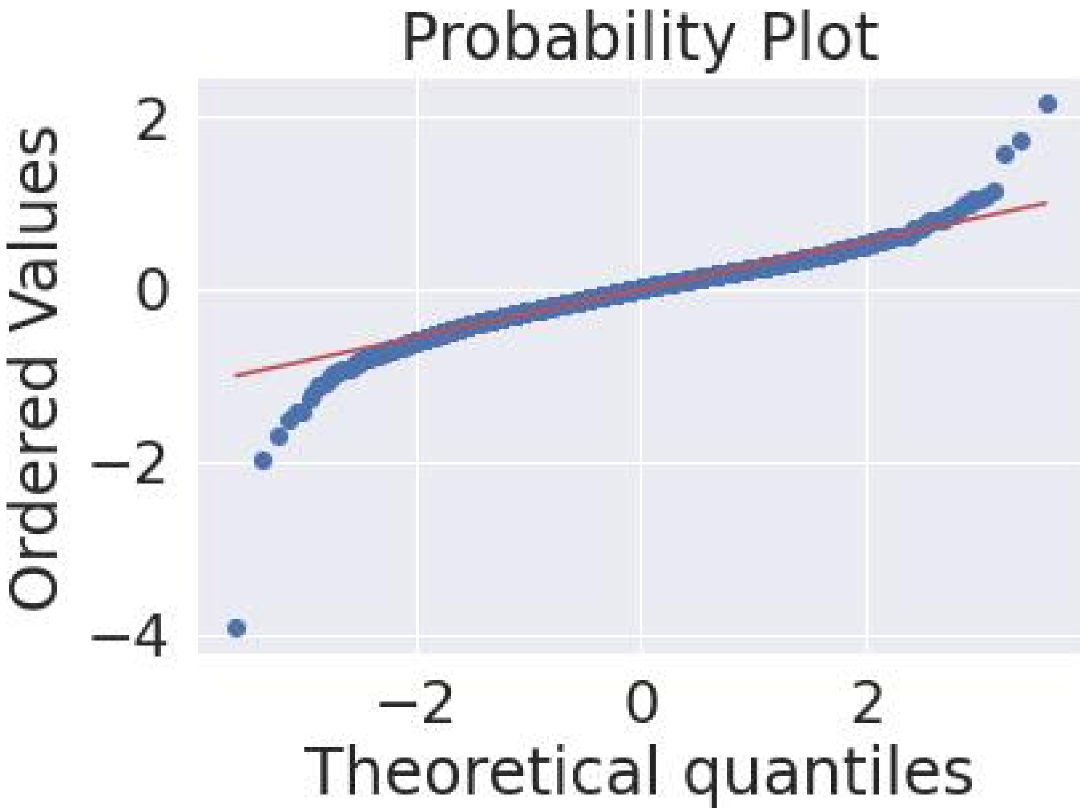
The residuals should be normally distributed.

[131]:
```
sns.distplot(residuals)
```

[131]: <AxesSubplot:ylabel='Density'>



[132]:
```
# Plot q-q plot of residuals
import pylab
import scipy.stats as stats

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



The residuals have a close to normal distribution. Assumption 5 is also satisfied. We should further investigate these values in the tails where we have made huge residual errors.

Now that we have seen that olsmodel2 follows all the linear regression assumptions. Let us use this model to draw inferences.

[133]:
```
print(olsmodel2.summary())
# Checking model performance
model_pref(olsmodel2, X_train1, X_test1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Price_log   R-squared:                       0.891
Model:                            OLS   Adj. R-squared:                  0.891
Method:                 Least Squares   F-statistic:                     1344.
Date:                Fri, 11 Jun 2021   Prob (F-statistic):               0.00
Time:                        05:13:42   Log-Likelihood:                -700.79
No. Observations:                4124   AIC:                             1454.
Df Residuals:                    4098   BIC:                             1618.
Df Model:                          25
Covariance Type:            nonrobust
==============================================================================
```

=============

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.9248	0.144	20.281	0.000	2.642	3.208
Kilometers_Driven	-2.441e-07	1.95e-07	-1.252	0.211	-6.27e-07	1.38e-07
Mileage	-0.0205	0.002	-11.339	0.000	-0.024	-0.017
Power	0.0081	0.000	54.814	0.000	0.008	0.008
Seats	0.0530	0.008	6.929	0.000	0.038	0.068
Ageofcar	-0.1123	0.002	-56.990	0.000	-0.116	-0.108
Kilometers_Driven_log	-0.0655	0.011	-5.779	0.000	-0.088	-0.043
Location_Bangalore	0.1572	0.030	5.249	0.000	0.098	0.216
Location_Chennai	0.0220	0.028	0.780	0.436	-0.033	0.077
Location_Coimbatore	0.0990	0.027	3.635	0.000	0.046	0.152
Location_Delhi	-0.0451	0.027	-1.653	0.098	-0.099	0.008
Location_Hyderabad	0.1166	0.026	4.409	0.000	0.065	0.168
Location_Jaipur	-0.0637	0.029	-2.212	0.027	-0.120	-0.007
Location_Kochi	-0.0268	0.027	-0.989	0.323	-0.080	0.026
Location_Kolkata	-0.2366	0.028	-8.495	0.000	-0.291	-0.182
Location_Mumbai	-0.0569	0.026	-2.155	0.031	-0.109	-0.005
Location_Pune	-0.0413	0.027	-1.513	0.130	-0.095	0.012
Fuel_Type_Diesel	0.1879	0.047	4.013	0.000	0.096	0.280
Fuel_Type_Electric	1.0838	0.210	5.172	0.000	0.673	1.495
Fuel_Type_LPG	-0.0573	0.119	-0.483	0.629	-0.290	0.175
Fuel_Type_Petrol	-0.1431	0.047	-3.015	0.003	-0.236	-0.050
Transmission_Manual	-0.2226	0.014	-15.771	0.000		

```
-0.250      -0.195
Owner_Type_Fourth & Above      0.1633      0.145      1.129      0.259
-0.120      0.447
Owner_Type_Second             -0.0914      0.013     -6.927      0.000
-0.117      -0.066
Owner_Type_Third              -0.1374      0.038     -3.588      0.000
-0.212      -0.062
Brand_Class_Low               -0.2296      0.015    -15.410      0.000
-0.259      -0.200
==============================================================================
Omnibus:                     1160.543   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            23002.735
Skew:                          -0.843   Prob(JB):                         0.00
Kurtosis:                      14.446   Cond. No.                     3.31e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.31e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
    Data     RMSE      MAE      MAPE
0  Train  6.11552  2.26577  23.45353
1   Test  6.47005  2.38709  23.45102
```

10.2 Observations from the model

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

1. With our linear regression model we have been able to capture ~89 variation in our data.
2. The model indicates that the most significant predictors of price of used cars are -
   - Age of the car
   - Number of seats in the car
   - Power of the engine
   - Mileage
   - Kilometers Driven
   - Location
   - Fuel_Type
   - OwnerType
   - Transmission - Automatic/Manual
3. Newer cars sell for higher prices. 1 unit increase in age of the car leads to [exp(0.1123) = 1.12 Lakh] decrease in the price of the vehicle, when everything else is constant.
4. As the number of seats increases, the price of the car increases - exp(0.05) = 1.05 Lakhs
5. Mileage is inversely correlated with Price. Generally, high mileage cars are the lower budget cars.

6. 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, everything else being 0.
7. The categorical variables are a little hard to interpret. But it can be seen that all the car_category variables in the dataset have a negative relationship with the Price and the magnitude of this negative relationship decrease as the brand category moves to lower brands.

11 Recommendations

- Our final Linear Regression model has a MAPE of 23% on the test data, which means that we are able to predict within 23% of the price value. This is a very good model but can be further improved
- 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 = -0.2) are very risky and we need to be careful about investments in this area.
- Based on Analysis, we can to divide our cars into 3 segment Low, Medium and High budget.
- Brands like Maruti, Hyundai ,Honda are low budget and very popular brands in used car market.
- Brands like BMW, Bentley, Jaguar, Land Rover, Mercedes Benz,Porche,Mini Cooper are high budget cars and are mostly bought by car enthusiast who are ready to buy a two user owned car at higher price as well.
- Brands like Toyota,Volvo can be Medium budget cars.
- Mumbai and Hyderbad seems to be more popular in Used car market, need to verify this with more data from other demographic regions. 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.
- Need to acquire more Automatic cars to earn more profits, as this car sell at higher prices.
- With Increasing petrol rates diesel car are in more demand in recent years, acquiring and selling them can high profits
- Along with this we can include scheme like take a test drive for half day to pursue customer to buy.
- We can provide Car maintenance packages where customers pays a small upfront fees and can bring the car for servicing anytime in a year to attract more customers.

Important points - There are more soft parameters which also should be considered when buying a car, the wear and tear the car has been through and how much the company will have to work on car to make it ready for sale.

- If the car as already been in some kind of accident that would also effect the price.
- Other good to have feature like AC,Moon roof,Airbags can also have impact on the price.
- Car model that are too old will depreciate a lot can impact the demand .