# Case Study: Cars4U using Linear Regression

## Context:

**Cars4U is a budding tech start-up that aims to find footholes in this market.**

- 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.
- There is a slowdown in new car sales and that could mean that the demand is shifting towards the pre-owned market.
- Used cars are very different beasts with huge uncertainty in both pricing and supply.
- The pricing scheme of these used cars becomes important in order to grow in the market..

## Problem:

**The dataset aims to answer the following key questions:**

- What were the features that affect the price of the car?
- What are the predicting variables actually affecting the Price?
- Does power and engine also affect the selling price, or perhaps, something else?
- Does Kilometers Driven and car age affect Price?
- Does Price has positive or negative correlation with a number of seats, transmission, fuel type, etc?
- What is the impact of location on the price of used cars?
- What is the impact of brand on the price of used cars?

## Objective:

Explore the dataset and extract insights from the data.

1. Build a linear regression model to predict the prices of used cars.
2. Generate a set of insights and recommendations that will help the business.

## Data Dictionary:

The data is for 100 randomly selected users of a online news portal called E-news Express. It contains the following variables:

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) This is our target value. This means “price” is the value that we want to predict from the data-set, and the predictors should be all the other variables listed

---

# Key steps

---



## Import libraries

In [1]:
```
# this will help in making the Python code more structured automatically (goo
%load_ext nb_black
```

In [2]:
```
# silence unnecessary warnings
import warnings

warnings.filterwarnings("ignore")
```

In [3]:
```
# Import necessary libraries.
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

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

# To enable plotting graphs in Jupyter notebook
%matplotlib inline

# To build linear regression_model
from sklearn.linear_model import LinearRegression

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

In [4]:
```
# Load the data into pandas dataframe
data = pd.read_csv("used_cars_data.csv")
```

In [5]:
```
# Coping the data to another variable to avoid any changes to the original da
car = data.copy()
```

# Overview of the data

In [6]:
```
print(f"There are {car.shape[0]} rows and {car.shape[1]} columns.")  # f-stri

# Look at 10 random rows
# Setting the random seed via np.random.seed to get the same random results e
np.random.seed(1)
car.sample(n=10)
```

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

Out[6]:

| | S.No. | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owne |
|---|---|---|---|---|---|---|---|---|
| **2397** | 2397 | Ford EcoSport 1.5 Petrol Trend | Kolkata | 2016 | 21460 | Petrol | Manual | |
| **3777** | 3777 | Maruti Wagon R VXI 1.2 | Kochi | 2015 | 49818 | Petrol | Manual | |
| **4425** | 4425 | Ford Endeavour 4x2 XLT | Hyderabad | 2007 | 130000 | Diesel | Manual | |
| **3661** | 3661 | Mercedes-Benz E-Class E250 CDI Avantgrade | Coimbatore | 2016 | 39753 | Diesel | Automatic | |
| **4514** | 4514 | Hyundai Xcent 1.2 Kappa AT SX Option | Kochi | 2016 | 45560 | Petrol | Automatic | |

| | S.No. | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owne |
|---|---|---|---|---|---|---|---|---|
| **599** | 599 | Toyota Innova Crysta 2.8 ZX AT | Coimbatore | 2019 | 40674 | Diesel | Automatic | |
| **186** | 186 | Mercedes-Benz E-Class E250 CDI Avantgrade | Bangalore | 2014 | 37382 | Diesel | Automatic | |
| **305** | 305 | Audi A6 2011-2015 2.0 TDI Premium Plus | Kochi | 2014 | 61726 | Diesel | Automatic | |
| **4582** | 4582 | Hyundai i20 1.2 Magna | Kolkata | 2011 | 36000 | Petrol | Manual | |
| **5434** | 5434 | Honda WR-V Edge Edition i-VTEC S | Kochi | 2019 | 13913 | Petrol | Manual | |

In [7]:
```
# Understand the data before doing any analysis. Data has a variety of types,
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   S.No.              7253 non-null   int64  
 1   Name               7253 non-null   object 
 2   Location           7253 non-null   object 
 3   Year               7253 non-null   int64  
 4   Kilometers_Driven  7253 non-null   int64  
 5   Fuel_Type          7253 non-null   object 
 6   Transmission       7253 non-null   object 
 7   Owner_Type         7253 non-null   object 
 8   Mileage            7251 non-null   object 
 9   Engine             7207 non-null   object 
 10  Power              7207 non-null   object 
 11  Seats              7200 non-null   float64
 12  New_Price          1006 non-null   object 
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 793.4+ KB
```

## Observation:

- *S.No.*: is the same as Index (We can drop it);
- *Name*: Name of the car which includes Brand name and Model name, we can categorize just by Brand, reduzing number of dummies when we start building the model;
- *Mileage, Engine, Power, New_Price* : represented as strings but that we really will want to be numeric;
- *New_Price*: has NaN values that need to be treated.

In [8]:
```
# looking at which columns have the most missing values
car.isnull().sum().sort_values(ascending=False)
```

Out[8]:
```
New_Price            6247
Price                1234
Seats                  53
Power                  46
Engine                 46
Mileage                 2
Owner_Type              0
Transmission            0
Fuel_Type               0
Kilometers_Driven       0
Year                    0
Location                0
Name                    0
S.No.                   0
dtype: int64
```

## Observations:

- Some columns have less than 7253 observations non-null, which indicat that there are missing values in it. *(treatment of missing values is necessary)*.
- Mileage, Engine and Power should be float variables.
- New_Price has 6247 missing values
- Price (dependent) has 1234 missing values

In [9]:
```
# checking descriptive statistics
# Are there any mathematical issues that may exist, such as extreme outliers
# include all means even the ones that is not numerical like categorical
car.describe(include="all").T
```

Out[9]:

| | count | unique | top | freq | mean | std | min | 25% | 50% | 75 |
|---|---|---|---|---|---|---|---|---|---|---|
| **S.No.** | 7253 | NaN | NaN | NaN | 3626 | 2093.91 | 0 | 1813 | 3626 | 54 |
| **Name** | 7253 | 2041 | Mahindra XUV500 W8 2WD | 55 | NaN | NaN | NaN | NaN | NaN | N |
| **Location** | 7253 | 11 | Mumbai | 949 | NaN | NaN | NaN | NaN | NaN | N |
| **Year** | 7253 | NaN | NaN | NaN | 2013.37 | 3.25442 | 1996 | 2011 | 2014 | 20 |
| **Kilometers_Driven** | 7253 | NaN | NaN | NaN | 58699.1 | 84427.7 | 171 | 34000 | 53416 | 730 |
| **Fuel_Type** | 7253 | 5 | Diesel | 3852 | NaN | NaN | NaN | NaN | NaN | N |
| **Transmission** | 7253 | 2 | Manual | 5204 | NaN | NaN | NaN | NaN | NaN | N |
| **Owner_Type** | 7253 | 4 | First | 5952 | NaN | NaN | NaN | NaN | NaN | N |
| **Mileage** | 7251 | 450 | 17.0 kmpl | 207 | NaN | NaN | NaN | NaN | NaN | N |
| **Engine** | 7207 | 150 | 1197 CC | 732 | NaN | NaN | NaN | NaN | NaN | N |
| **Power** | 7207 | 386 | 74 bhp | 280 | NaN | NaN | NaN | NaN | NaN | N |
| **Seats** | 7200 | NaN | NaN | NaN | 5.27972 | 0.81166 | 0 | 5 | 5 | |
| **New_Price** | 1006 | 625 | 33.36 Lakh | 6 | NaN | NaN | NaN | NaN | NaN | N |
| **Price** | 6019 | NaN | NaN | NaN | 9.47947 | 11.1879 | 0.44 | 3.5 | 5.64 | 9. |

### Observations:

- *S.No.*: is just a index, we can drop it;
- *Name*: is Brand name and Model name, there is 2041 unique values, we'll keep just Brand to reduce unique and apply dummies latter on.
- Location has 11 unique value.
- *Year* range is 1998 to 2019
- *Milage*, *Engine* and *Power* is numerical, we need to remove string.
- *Kilometers_Driven*, *Engine* and *Power* have outliers
- *Price (dependent)* goes in the range of 0.440 to 160.0

# Cleaning Data/Missing Values

## Cleaning Data:

### 1. Droping columns

In [10]:
```
# Dropping S.No. column once it just represent the index
car.drop(["S.No."], axis=1, inplace=True)
```

### 2. Numerical Columns containing string

> - There are some columns that should be numerical.
> - The values all end with some string representing the unit.
> - First We want to detect which columns fit this pattern, and then We'll turn these into numbers.

In [11]:
```
# Function to remove str from numerical values.


def str_to_num(pos_val):
    """For each value, take the number before the ' '
    unless it is not a string value. This will only happen
    for NaNs so in that case we just return NaN.
    """
    if isinstance(pos_val, str):
        return float(pos_val.split()[0])
    else:
        return np.nan


position_cols = ["Mileage", "Engine", "Power"]

car["Power"] = car["Power"].replace(
    "null bhp", np.nan
)  # replacing some NaN values that were set as str null.

for colname in position_cols:
    car[colname] = car[colname].apply(str_to_num)
```

### Observations:

- Running the funciton `str_to_num` , We got an error showing that some null values were set as a string instead of blank, We fixed it using `replace`

In [12]:
```
# check column types and number of values
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 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   float64
 8   Engine             7207 non-null   float64
 9   Power              7078 non-null   float64
 10  Seats              7200 non-null   float64
 11  New_Price          1006 non-null   object
 12  Price              6019 non-null   float64
dtypes: float64(5), int64(2), object(6)
memory usage: 736.8+ KB
```

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

Out[13]:

| | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mil |
|---|---|---|---|---|---|---|---|---|
| **0** | Maruti Wagon R LXI CNG | Mumbai | 2010 | 72000 | CNG | Manual | First | 2 |
| **1** | Hyundai Creta 1.6 CRDi SX Option | Pune | 2015 | 41000 | Diesel | Manual | First | 1 |
| **2** | Honda Jazz V | Chennai | 2011 | 46000 | Petrol | Manual | First | 1 |
| **3** | Maruti Ertiga VDI | Chennai | 2012 | 87000 | Diesel | Manual | First | 2 |
| **4** | Audi A4 New 2.0 TDI Multitronic | Coimbatore | 2013 | 40670 | Diesel | Automatic | Second | 1 |

# Missing values

1. *New_Price*: 86% of New_Price is missing, We'll drop the column;
2. *Price*: We'll drop all the rows without price, since the price of used cars is what we’re trying to predict in our upcoming analysis.
3. *Seats, Power, Engine, Mileage*: We should analize summary statistics to decide between mean or median to replace missing values

In [14]:
```
car.isnull().sum().sort_values(ascending=False)
```

Out[14]:
```
New_Price            6247
Price                1234
Power                 175
Seats                  53
Engine                 46
Mileage                 2
Owner_Type              0
Transmission            0
Fuel_Type               0
Kilometers_Driven       0
Year                    0
Location                0
Name                    0
dtype: int64
```

## New_Price

> New_ Price has 6247 missing data that represent 86% of datapoints. We'll drop the whole column

In [15]:
```
# Dropping New_Price column once 86% of data is missing
car.drop("New_Price", axis=1, inplace=True)
```

## Price

> Price has 1234 missing data and we should delete the whole row, because the Price is what we want to predict. Whitout Price information, we cant use the others informations on our predict model.

In [16]:
```
# Dropping rows where Price is Nan
car.dropna(subset=["Price"], inplace=True)
```

## Seats, Power, Engine and Milage

> Is missing values, and after checked descriptive statistics we'll choose to replace NaN values with `median` .

In [17]:
```
# we will replace missing values in every column with its median
medianFiller = lambda x: x.fillna(x.median())
numeric_columns = car.select_dtypes(include=np.number).columns.tolist()
car[numeric_columns] = car[numeric_columns].apply(medianFiller, axis=0)
```

# Exploratory Data Analysis

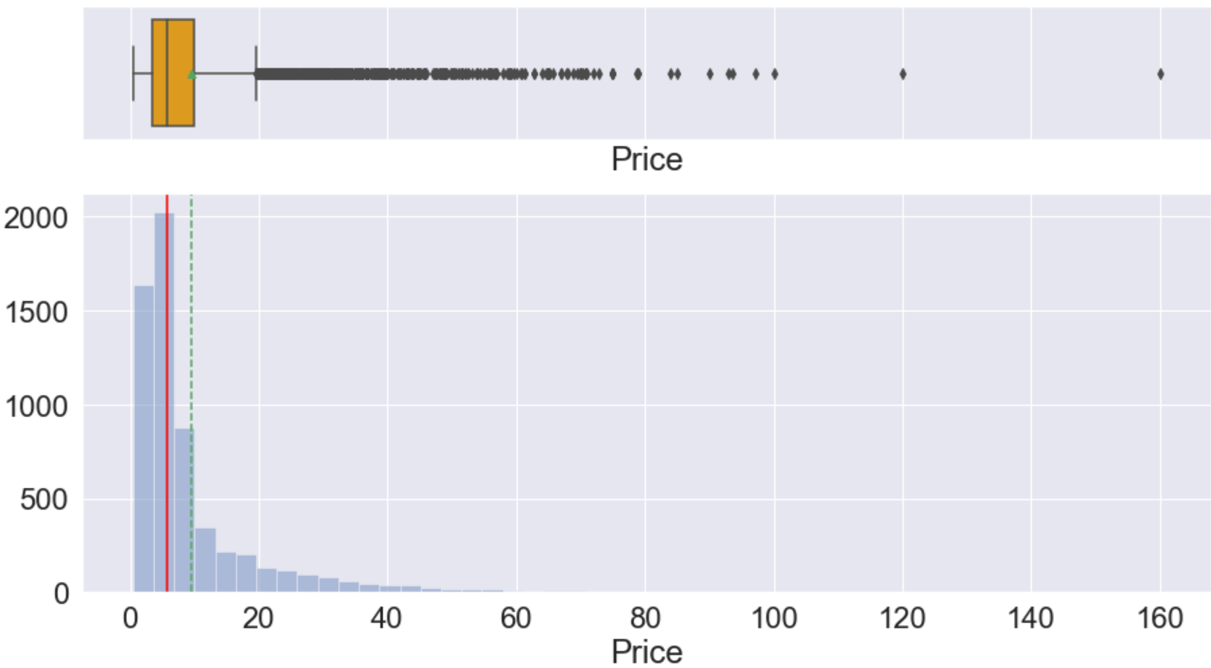
## Univariate analysis

In [18]:
```
# While doing univariate analysis of numerical variables we want to study the
# Let us write a function that will help us create boxplot and histogram for
# This function takes the numerical column as the input and returns the boxpl
# Let us see if this help us write faster and cleaner code.
```

```
def histogram_boxplot(feature, figsize=(15, 8), bins=None):
    """Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    """
    sns.set(font_scale=2)  # setting the font scale for seaborn
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid=2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        feature, ax=ax_box2, showmeans=True, color="orange"
    )  # boxplot will be created and a star will indicate the mean value of t
    sns.distplot(feature, kde=F, ax=ax_hist2, bins=bins) if bins else sns.dis
        feature, kde=False, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        feature.mean(), color="g", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        feature.median(), color="red", linestyle="-"
    )  # Add median to the histogram
```

**Exploring the dependent variable *Price***

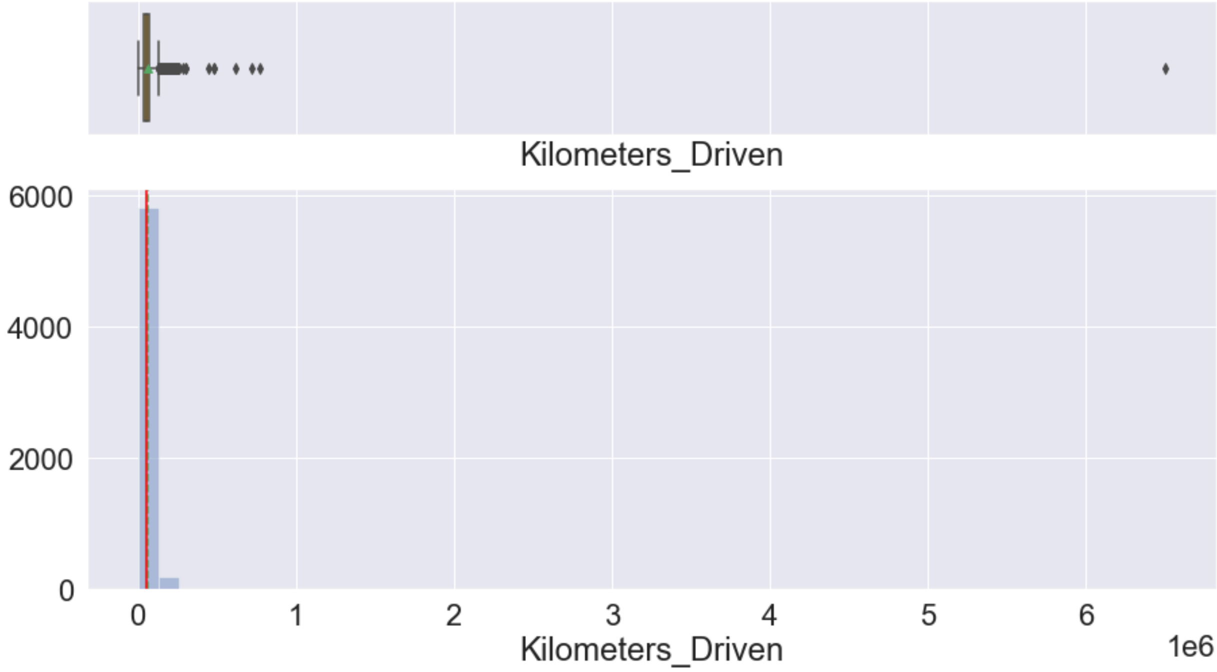
In [19]:
```
histogram_boxplot(car.Price)
```




**Observations**

- *Price* is right skewed, which means some brands have cars with price upper than 60 Lakhs
- Mean Price is around 5.640.

In [20]:
```
histogram_boxplot(car.Kilometers_Driven)
```

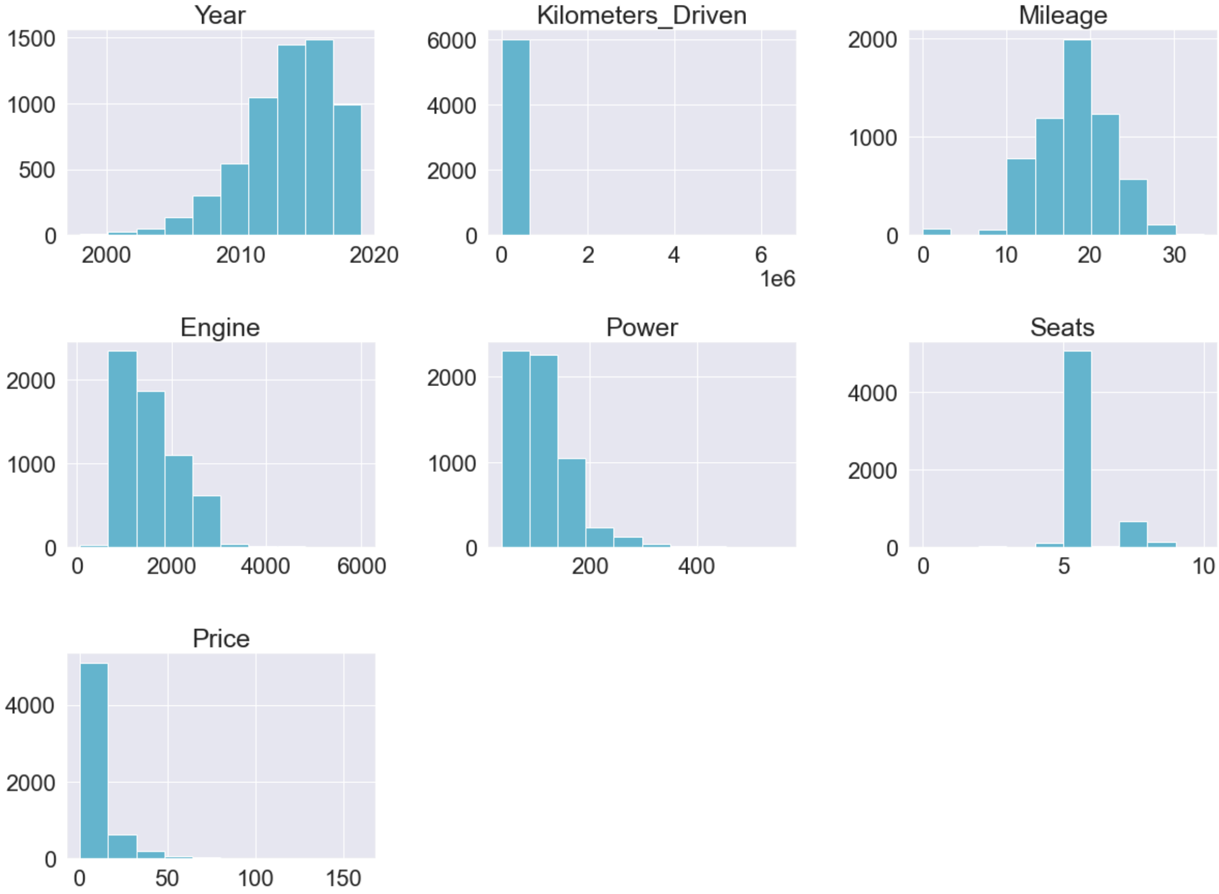



**Observations**

- *Kilometers_Driven* is right skewed.
- There is one car with a really high Kilometers (6500000), an outliers, with deep analises, seems to have a extra 0 on the number.

## Distribution of each numerical variable

In [21]:
```
# lets plot histogram of all numerical variables

all_col = car.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(17, 75))

for i in range(len(all_col)):
    plt.subplot(18, 3, i + 1)
    plt.hist(car[all_col[i]], color="c")
    # sns.histplot(car[all_col[i]], kde=True)  # you can comment the previous
    plt.tight_layout()
    plt.title(all_col[i], fontsize=25)

plt.show()
```



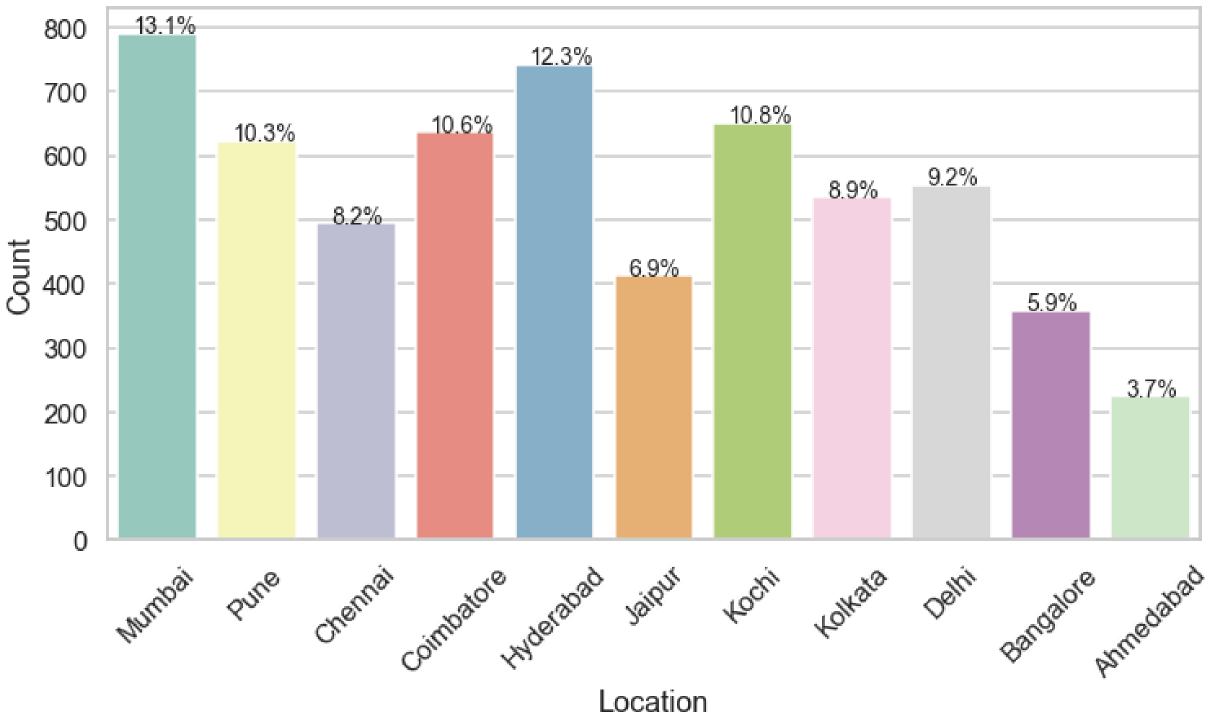

### Observation

- *Mileage* is somewhat normal distributed.
- *Kilometers_Driven*, *Engine*, *Power* and *Price* are right-skewed, and *Seats* is left-skewed.

In [22]:
```
# Function to create barplots that indicate percentage for each category.


def bar_perc(dataframe, xlabel_df, colors, ylabel_df="Count"):
    """
    This function takes the category column as the input and returns the barp
    dataframe: 1-d categorical feature array
    xlabel_df: x axis label
    ylabel_df: y axis label (default 'Count')
    colors: list of colors to use for the different variables
    """

    # Figure aesthetics
    sns.set_style("whitegrid")
    sns.set_context("talk")

    # Plot informations
    plt.figure(figsize=(12, 6))
    plot_df = sns.countplot(dataframe, palette=colors)
    plt.xlabel(xlabel_df)
    plt.ylabel(ylabel_df)
    plt.xticks(rotation=45)

    # Calculating the length of the column
    total = len(dataframe)

    # Looping to calculate percentage of each class of the category and annot
    for cat in plot_df.patches:
        percentage = "{:.1f}%".format(100 * cat.get_height() / total)
```

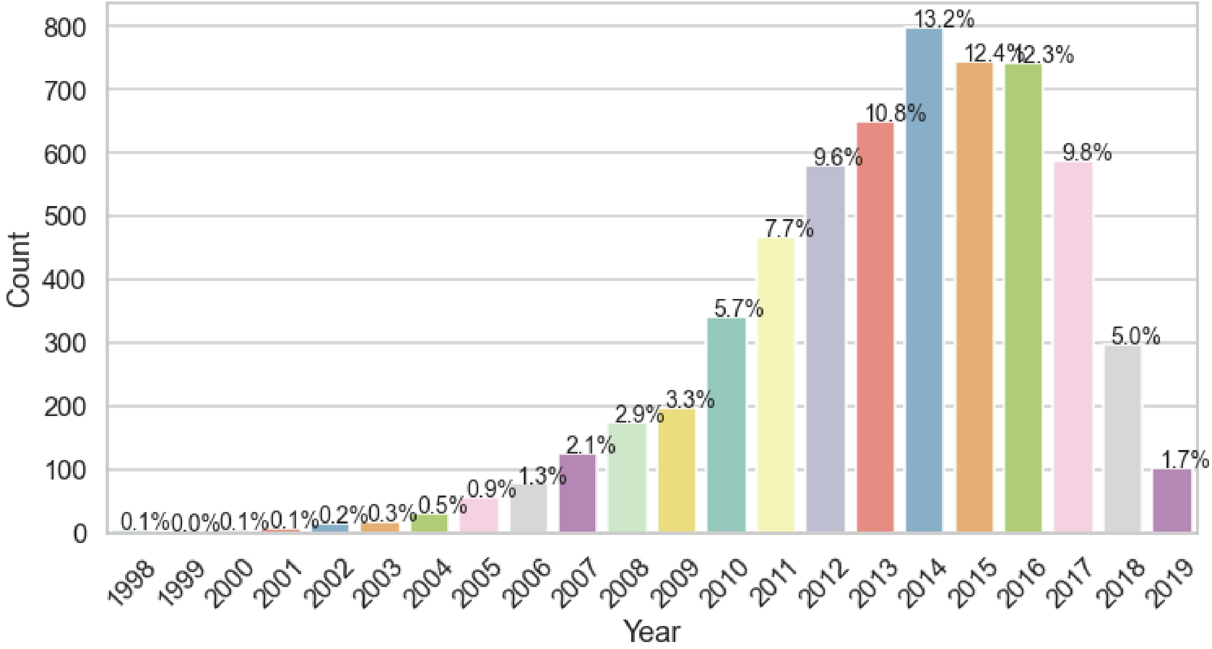
```
        # setting plot annotate location and size
        x = cat.get_x() + cat.get_width() / 2 - 0.25
        y = cat.get_y() + cat.get_height() + 1
        plot_df.annotate(percentage, (x, y), size=14)
```

In [23]:
```
# List of colors to use for the different products
colors = "Set3"

# Using the functio to plot barplot wtih percentage values
bar_perc(car["Location"], "Location", colors)
```



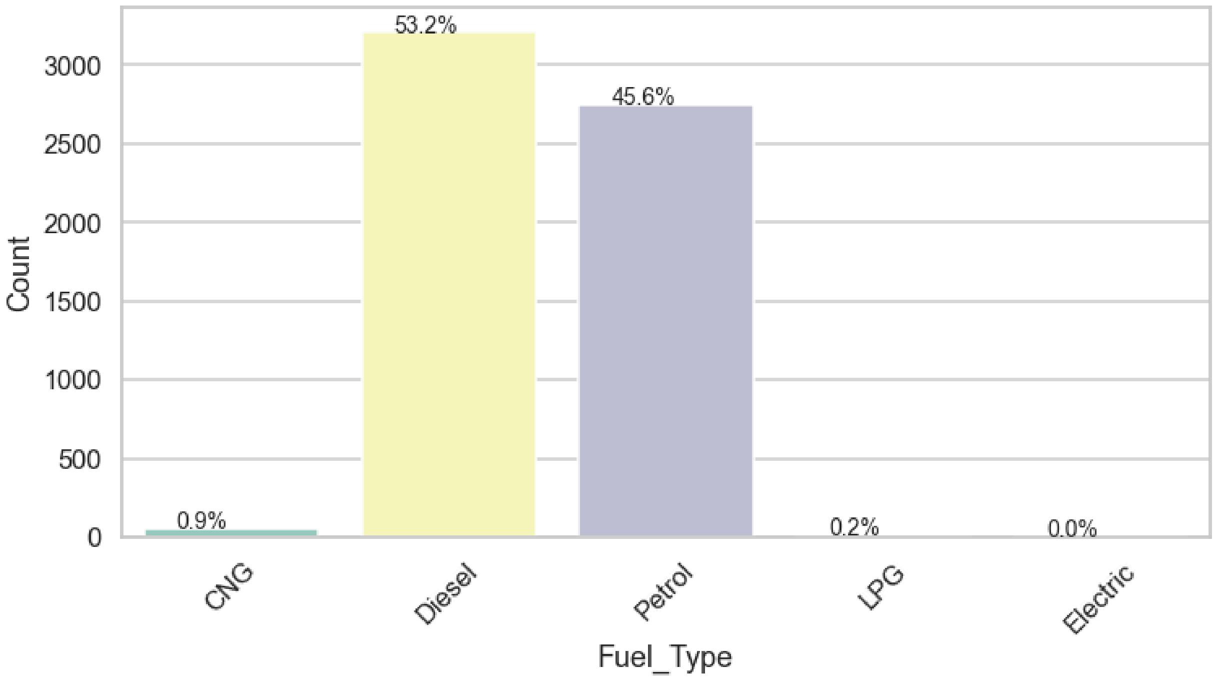

- Distribution between Location is fair, around 9.1% of datapoint by location.

In [24]:
```
# List of colors to use for the different products
colors = "Set3"

# Using the functio to plot barplot wtih percentage values
bar_perc(car["Year"], "Year", colors)
```



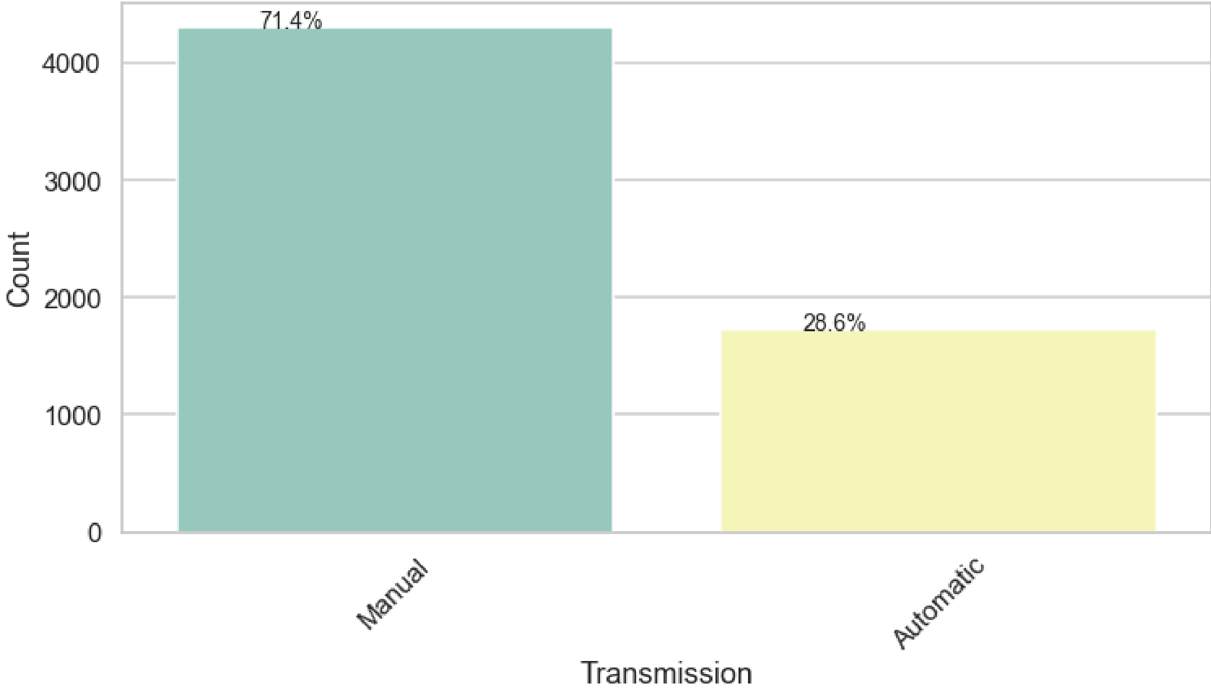

- Pre-owned car market has continued to grow over the past years

In [25]:
```
# List of colors to use for the different products
colors = "Set3"

# Using the functio to plot barplot wtih percentage values
bar_perc(car["Fuel_Type"], "Fuel_Type", colors)
```



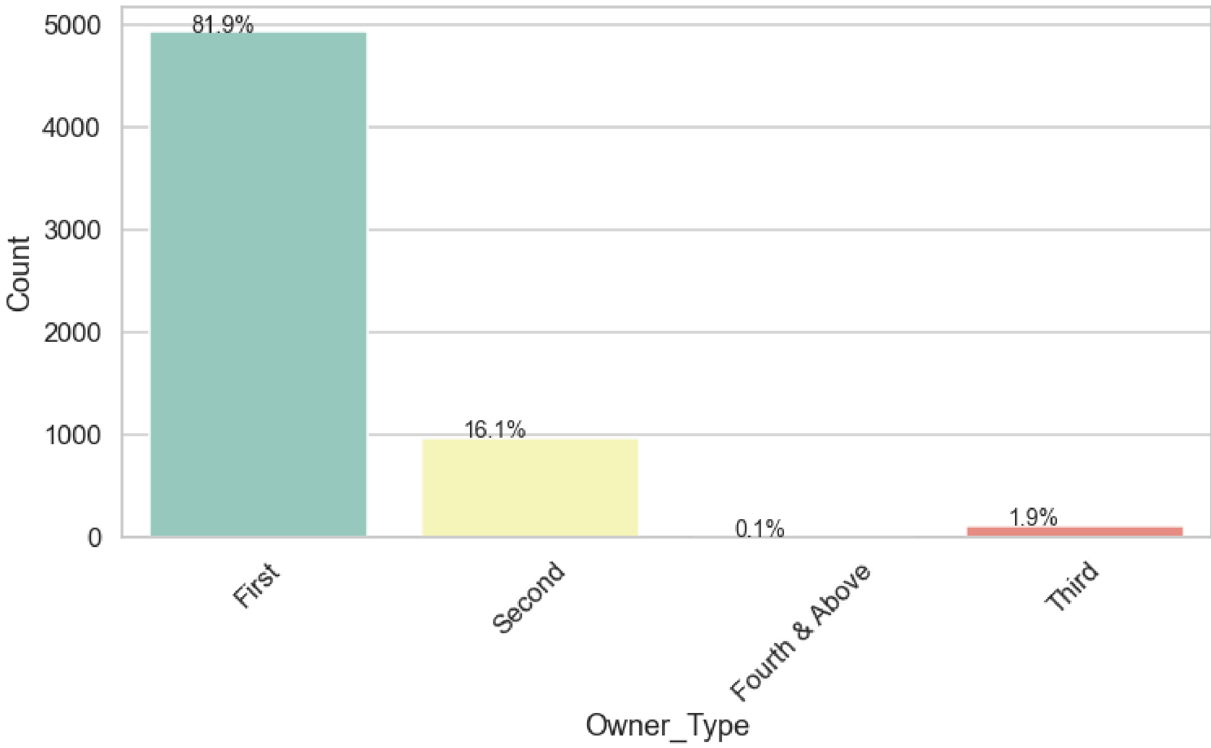

- Used car market is divided in Diesel and Petrol Fuel Type, showing high demand for this kind of cars.

In [26]:
```
# List of colors to use for the different products
colors = "Set3"

# Using the functio to plot barplot wtih percentage values
bar_perc(car["Transmission"], "Transmission", colors)
```




- There is a preference for Manual cars, witch can be explained with more analysis between correlation with price
- Manual Transmission is most popular on the market (71.4%).

In [27]:
```
# List of colors to use for the different products
colors = "Set3"

# Using the functio to plot barplot wtih percentage values
bar_perc(car["Owner_Type"], "Owner_Type", colors)
```



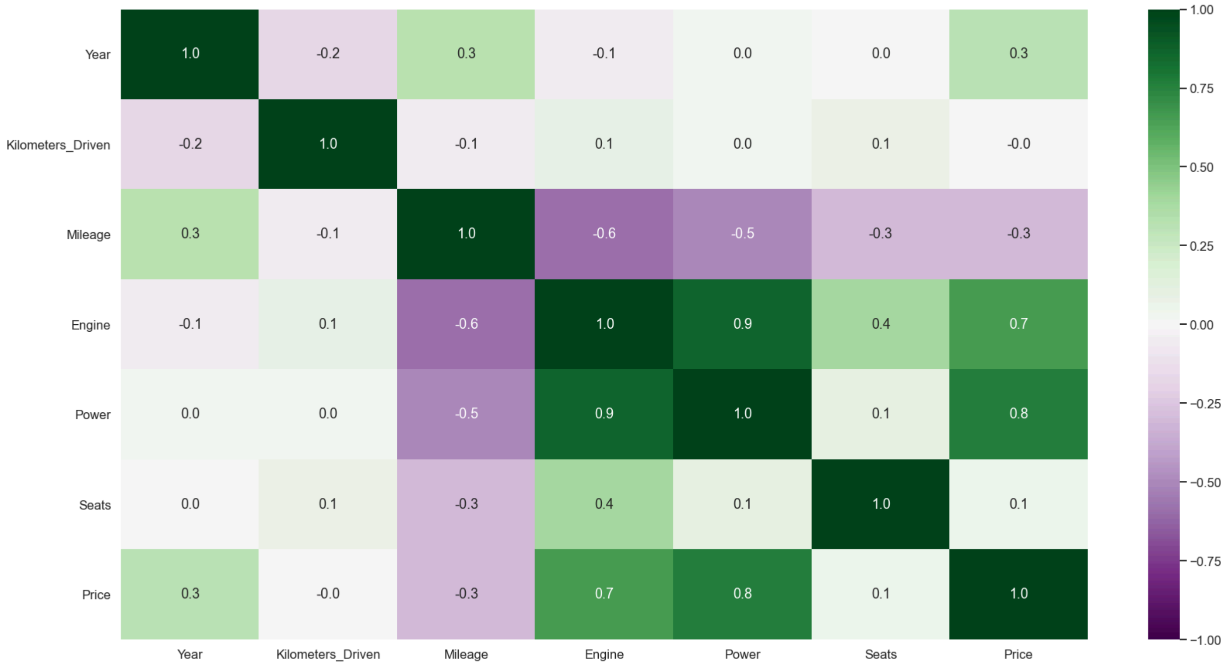

- Firt Owner represent 82% of cars availables on the market.

# Bivariate Analysis

## Looking for correlations (HeatMap)

In [28]:
```
numeric_columns = car.select_dtypes(include=np.number).columns.tolist()
corr = (car[numeric_columns].corr())#.sort_values(by=["Price"], ascending=Fal

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(28, 15))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(
    corr,
    cmap='PRGn',
    annot=True,
    fmt=".1f",
    vmin=-1,
)
```

Out[28]: <AxesSubplot:>




In [29]:
```
car[car.columns[:]].corr()["Price"][:]
```

Out[29]:
```
Year                 0.305327
Kilometers_Driven   -0.011493
Mileage             -0.306588
Engine               0.657347
Power                0.769711
Seats                0.052811
Price                1.000000
Name: Price, dtype: float64
```

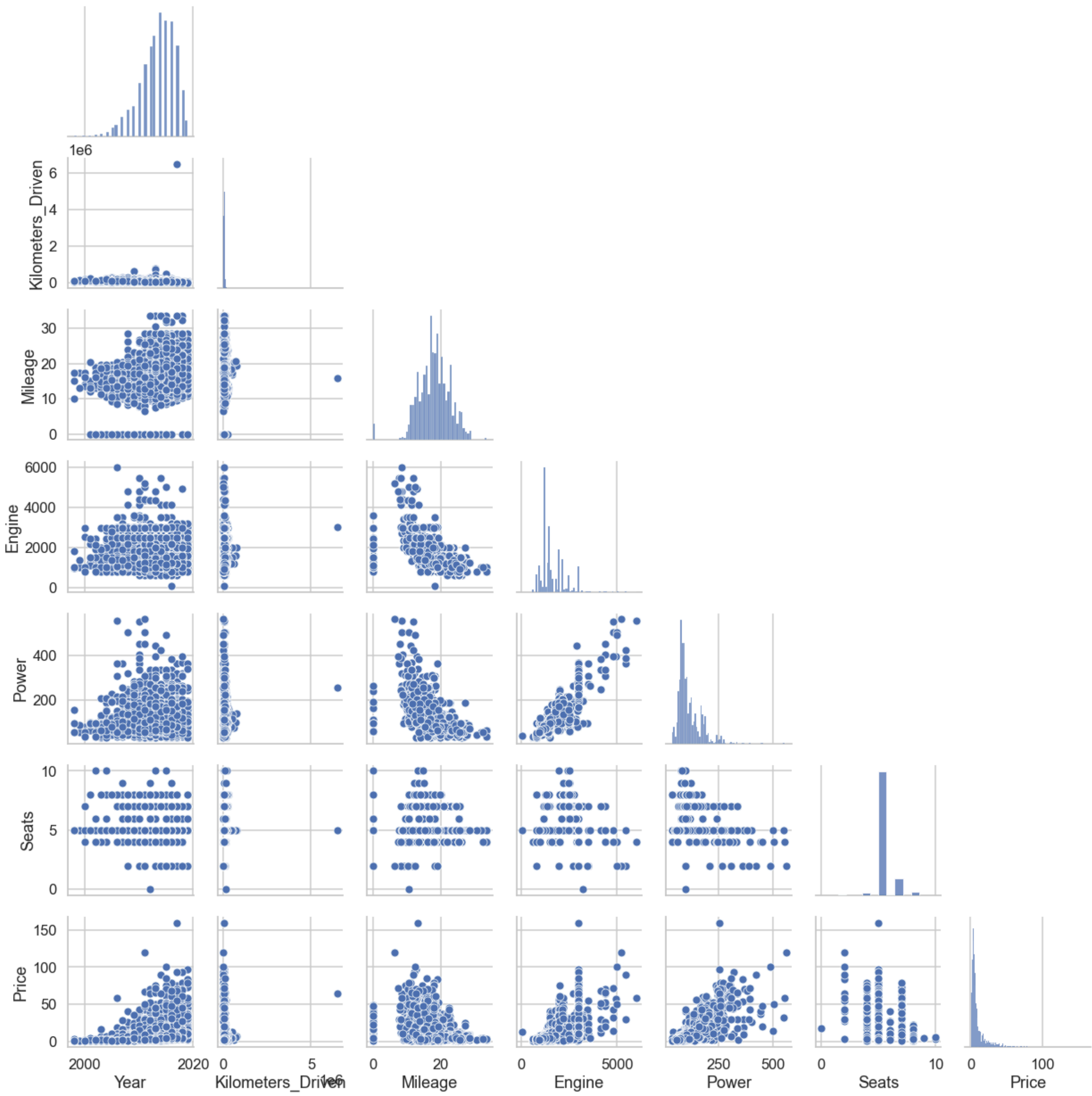
#### Observations

- *Price* is highly correlated with *Power* and *Engine*, which means that when Power or Engine moved up, the Price tend to move in the same direction.
- *Year* have a positive weaker correlated with price.

- *Price* have a negative weaker linear relationship with *Mileage* (a negative correlation: where the values of one variable tend to increase when the values of the other variable decrease.).

In [30]:
```
# Ploting Bivariate Scatter Plots
sns.pairplot(car[numeric_columns], corner=True)
```

Out[30]: <seaborn.axisgrid.PairGrid at 0x2472dee2670>

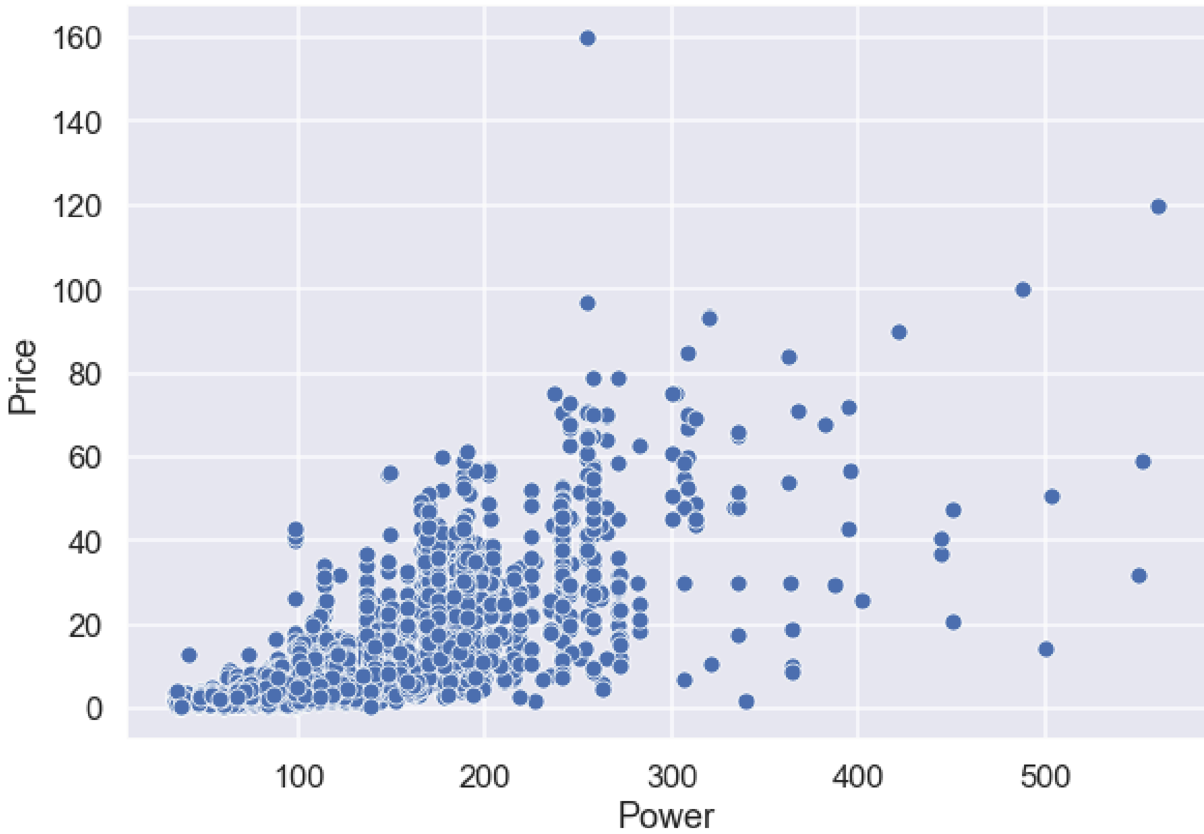



## Looking at the graphs of variables that are `correlated` with *Price*.

# Observations on Price by Power

In [31]:
```
sns.set_style("darkgrid")
plt.figure(figsize=(10, 7))
sns.scatterplot(
    y="Price",
    x="Power",
    data=car,
)
```

Out[31]: <AxesSubplot:xlabel='Power', ylabel='Price'>



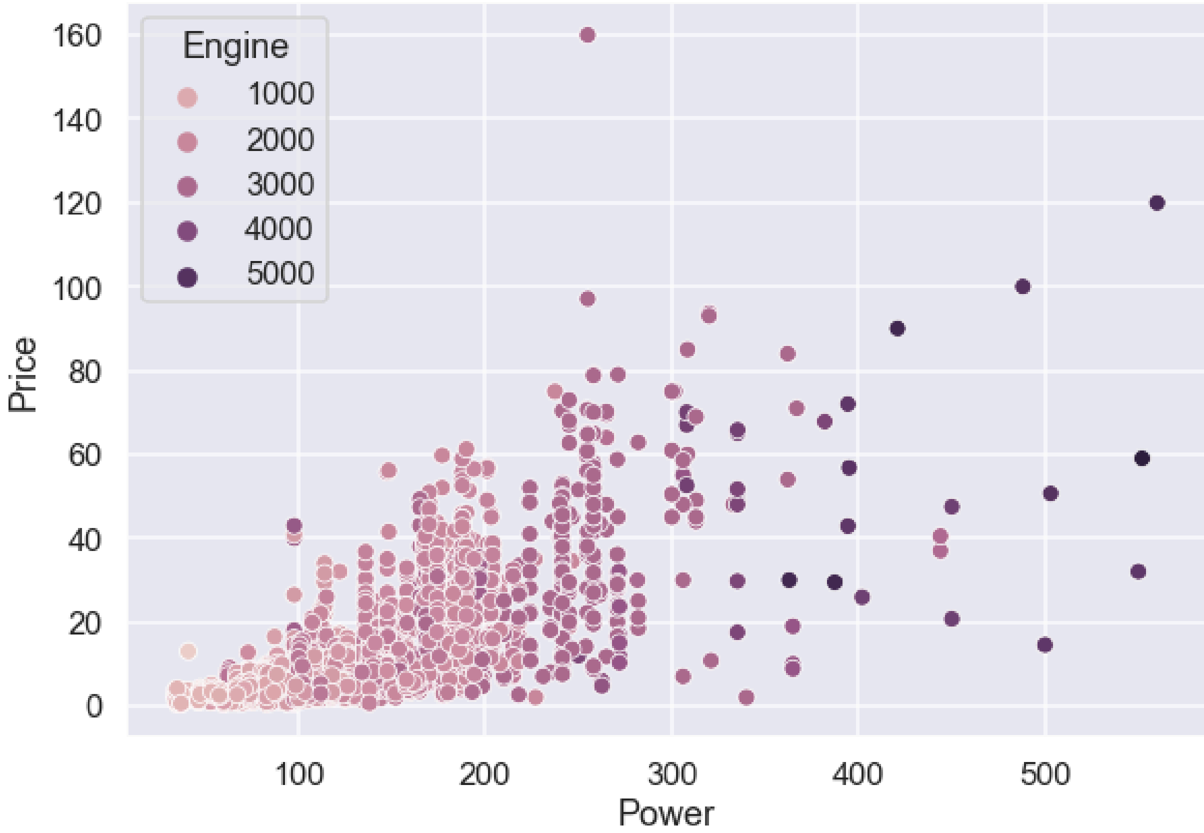

### Observation:

- Could power possibly predict the price of a car?
- There is a linear relationship between Price and Power.
- This relationship make sense, cars with more Power tend to me more expensive.
- There is some outliers that needs to be treated.

## Observations on Price by Power per Engine

In [32]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Price", x="Power", hue="Engine", data=car)
```

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



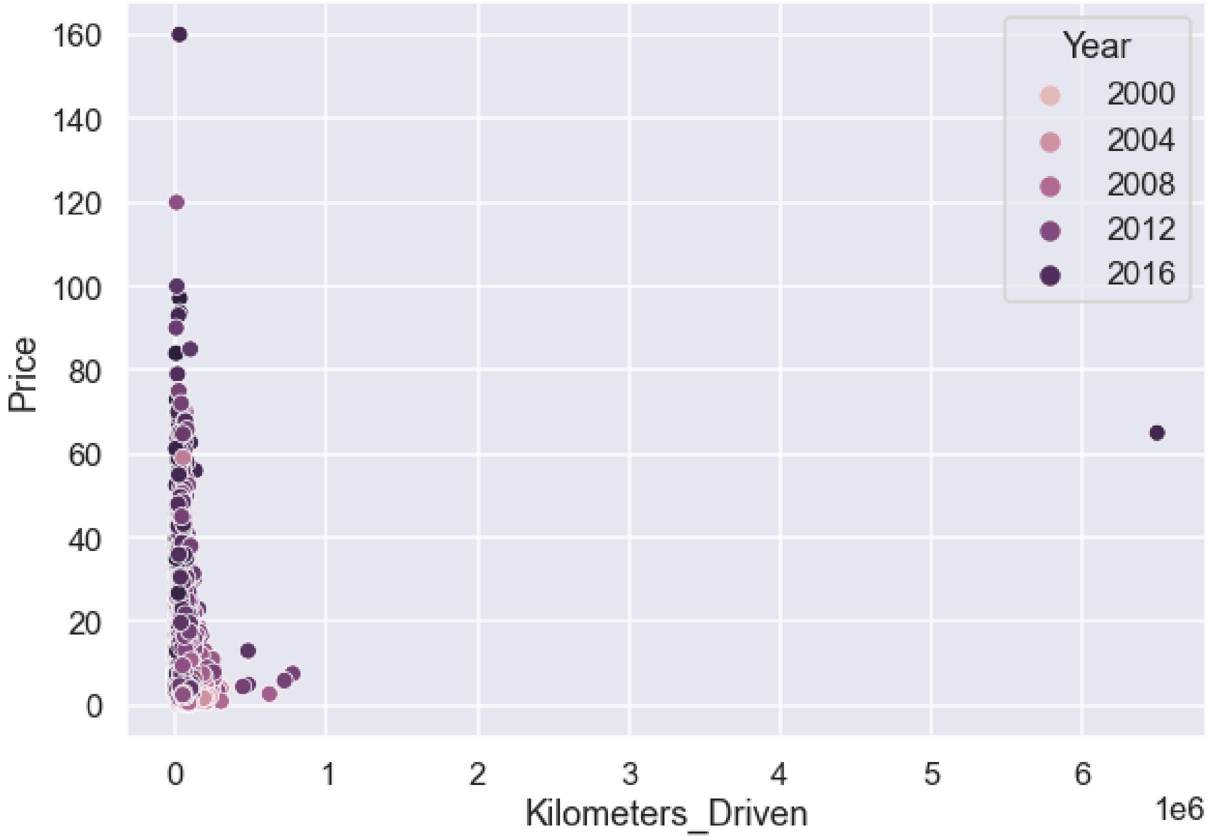

### Observation:

- There is a positive correlation between Price, Power and Engine.
- Higher the price, higer the car Power an Engine.
- Data points are concentrated on lower power, engine and power.

## Observations on Price by Kilometers_Driven per Car Year

In [33]:
```
sns.set_style("darkgrid")
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Price", x="Kilometers_Driven", hue="Year", data=car)
```

Out[33]: <AxesSubplot:xlabel='Kilometers_Driven', ylabel='Price'>




### Observation:

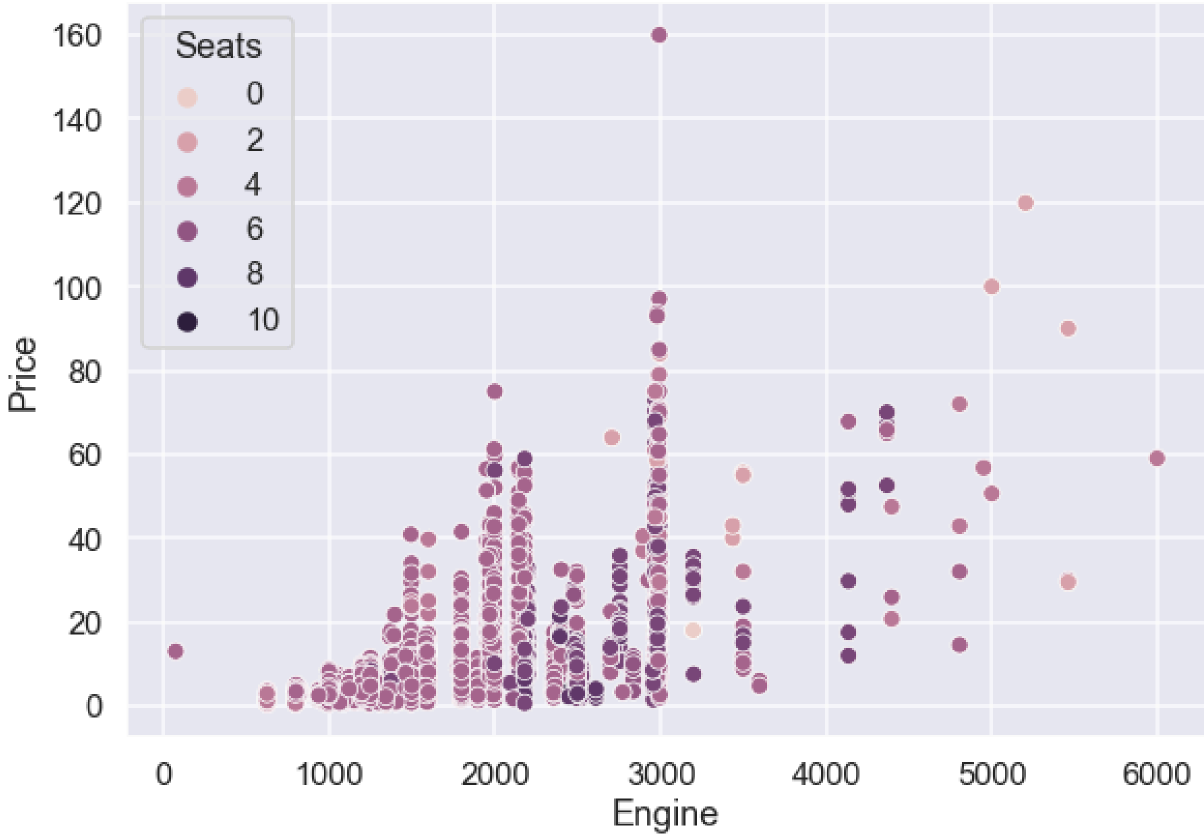
- Kilometers_Driven has one data set highly far from mean (outlier).
- Is this Kilometers_Driven outlier correct? It seems to be a new car (2016), to have so many Kilometers Driven.
- There is a relationship between Year and Price, We can see that newer is the car it tends to have high price.

### Looking at the graphs of a few variables that are `not correlated` with *Price*.

## Observations on Price by Seats

In [34]:
```
sns.set_style("darkgrid")
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Price", x="Engine", hue="Seats", data=car)
```

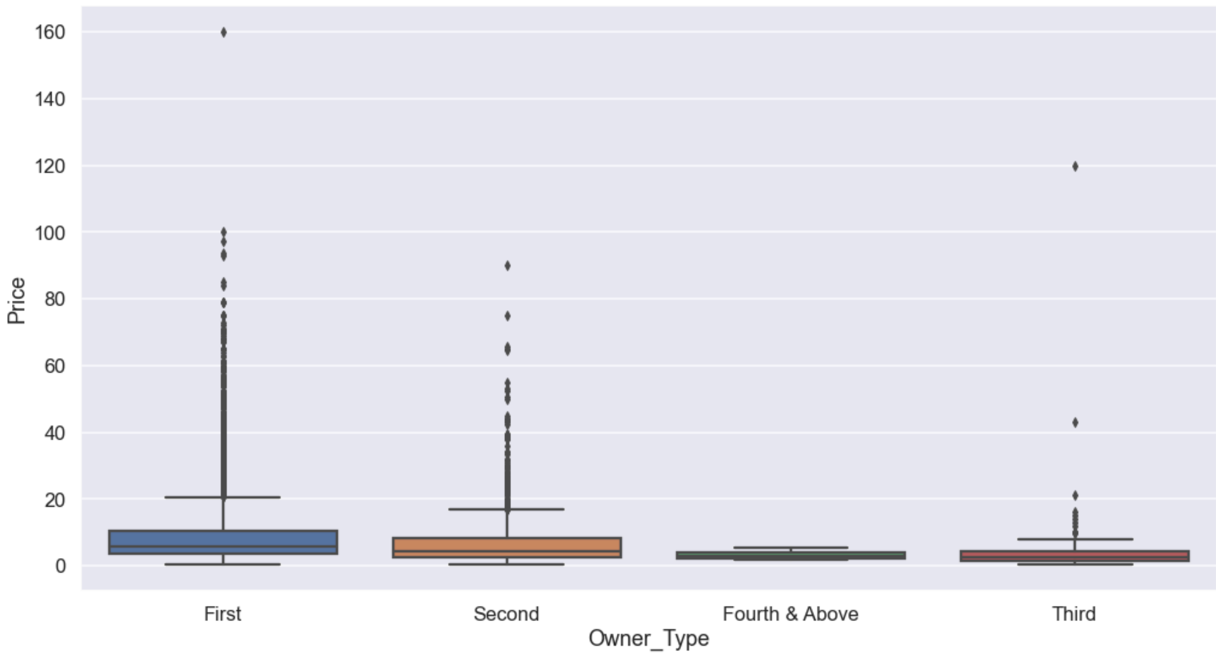
Out[34]: <AxesSubplot:xlabel='Engine', ylabel='Price'>




## Observation:

- Cars with 2 Seats tend to have bigger Engine and Higher Price.
- Cars are concentrated on lower price, lower engine and 5 seats (mostly comum cars)

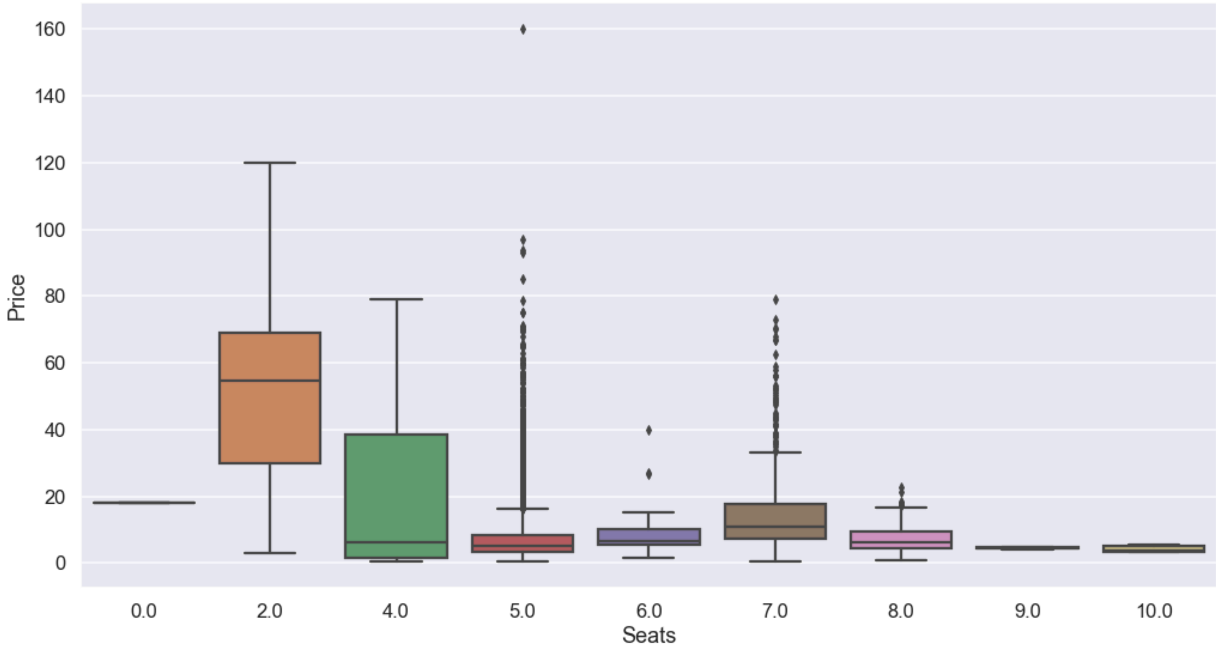
In [35]:
```
plt.figure(figsize=(17, 9))
sns.boxplot(x="Owner_Type", y="Price", data=car)
plt.show()
```




## Observation:

- Cars on First Owner Type has higher mean of price.

In [36]:
```
plt.figure(figsize=(17, 9))
sns.boxplot(x="Seats", y="Price", data=car)
plt.show()
```

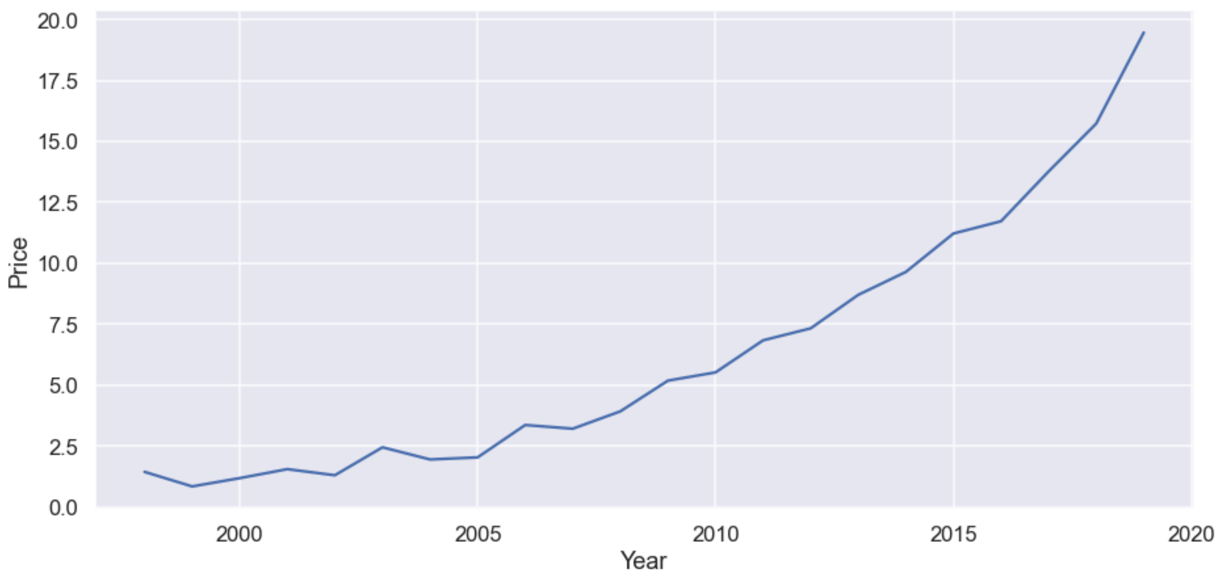



## Observation:

- Cars with 2 seats tend to have higher price (Bigger Engine)
- Is this Kilometers_Driven outlier correct? It seems to be a new car (2016), to have so many Kilometers Driven.
- There is a relationship between Year and Price, We can see that newer is the car it tends to have high price.

In [37]:
```
# price by age of car
plt.figure(figsize=(15, 7))
sns.lineplot(x="Year", y="Price", data=car, ci=None)
```

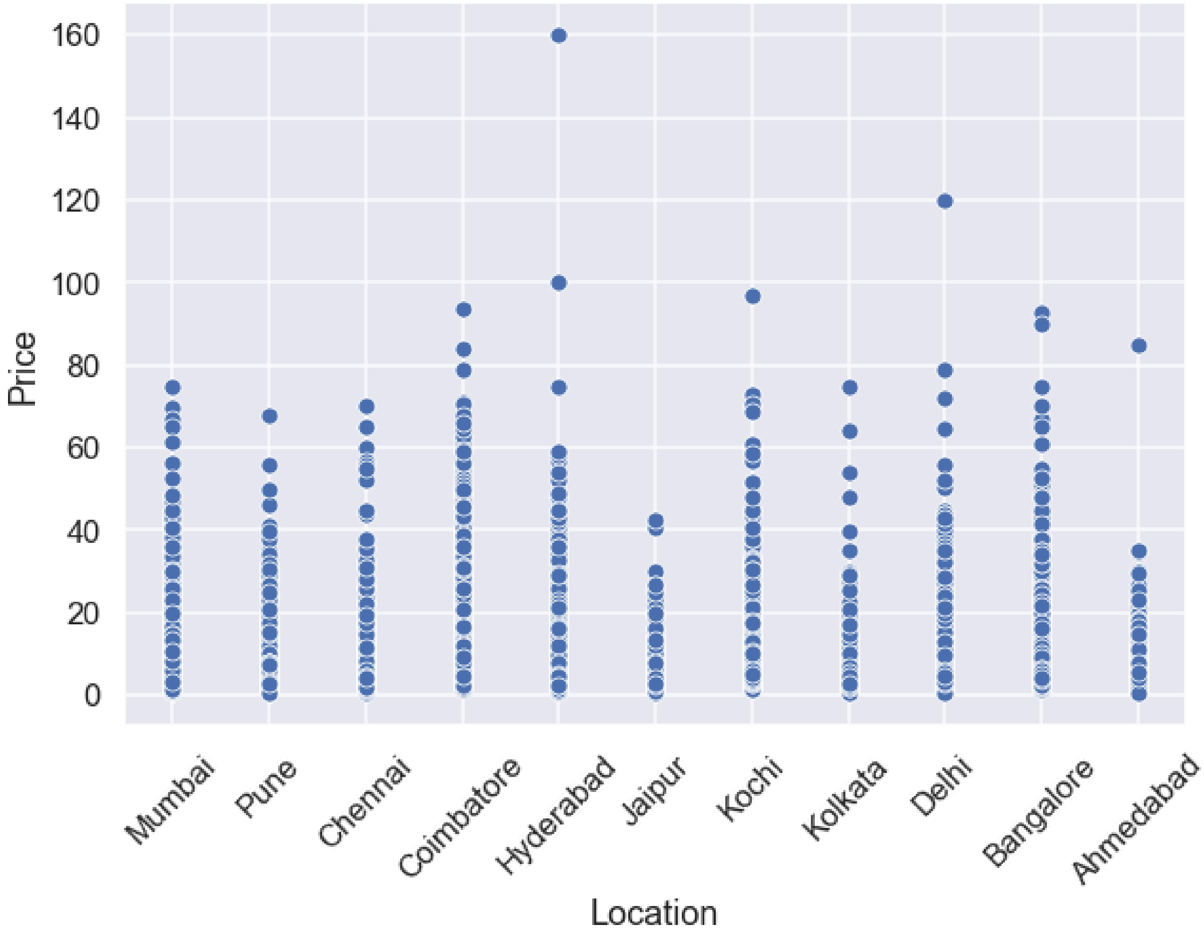
Out[37]: <AxesSubplot:xlabel='Year', ylabel='Price'>




- The newer the car, the higher its price.

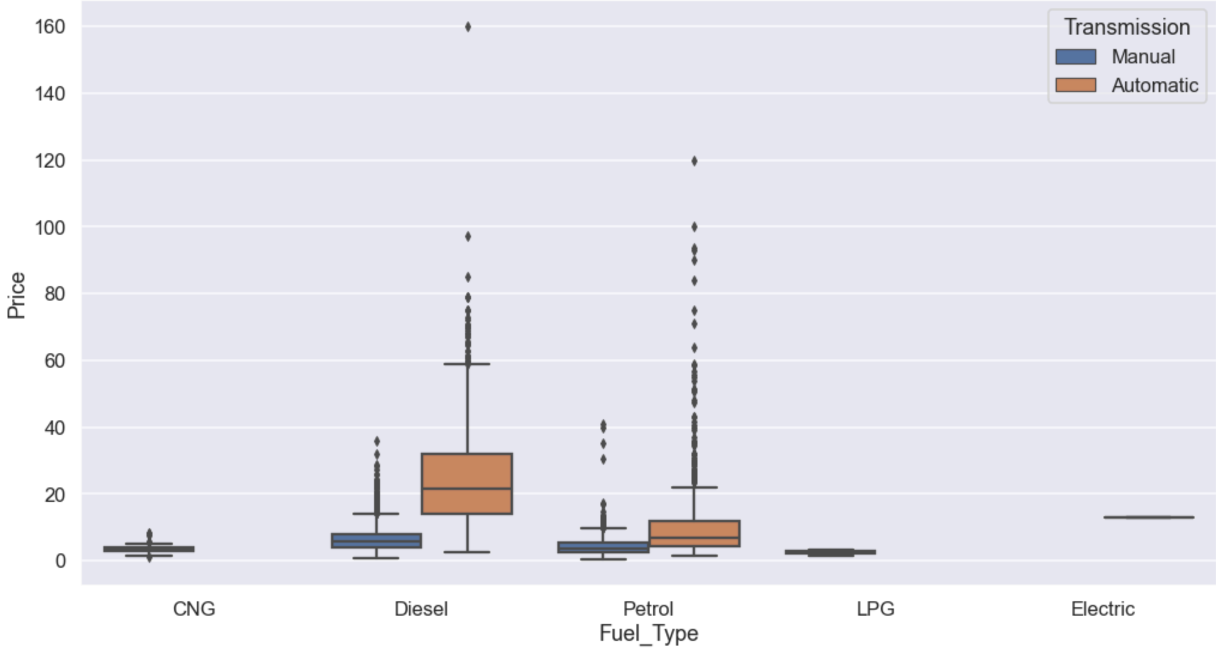
In [38]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="Price", x="Location", data=car)
plt.xticks(rotation=45)
```

Out[38]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```




**Observations** Doesn't seems to have strong correlation between Price and Location

In [39]:
```
plt.figure(figsize=(17, 9))
sns.boxplot(x="Fuel_Type", y="Price", data=car, hue="Transmission")
plt.show()
```




**Observations**

- Price tends to be greater on Diesel type and Automatic transmission.
- There is some outliers that needs some attention.

# Data Pre-processing

## 1. Dropping Name and keeping Brand instead

In [40]:
```
# We will use only brand and set as categorical type;
temp = car["Name"].str.split(" ", n=1, expand=True)
car.drop(["Name"], axis=1, inplace=True)
car["Brand"] = temp[0]
```

In [41]:
```
# verify unique values
car["Brand"].unique()
```

Out[41]:
```
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'], dtype=object)
```

### Observations:

- Isuzu was written in different ways, We gonna use `title` to fix it.

In [42]:
```
# using .title() in case of capitalization issues
car["Brand"] = car["Brand"].str.title()
```

In [43]:
```
car["Brand"].unique()
```

Out[43]:
```
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', 'Force', 'Bentley', 'Lamborghini'],
      dtype=object)
```

In [44]:
```
car["Brand"].value_counts()
```

Out[44]:
```
Maruti           1211
Hyundai          1107
Honda             608
Toyota            411
Mercedes-Benz     318
Volkswagen        315
Ford              300
Mahindra          272
Bmw               267
Audi              236
Tata              186
Skoda             173
Renault           145
Chevrolet         121
Nissan             91
Land               60
Jaguar             40
Fiat               28
Mitsubishi         27
Mini               26
Volvo              21
Porsche            18
Jeep               15
Datsun             13
Force               3
Isuzu               3
Lamborghini         1
Bentley             1
Ambassador          1
Smart               1
Name: Brand, dtype: int64
```

## Observation

- Our data doesn't have enought data point per brand, and this will reflect in our model.
- Brands with less than 3 datapoints, I'll group them and call it as `Others'

In [45]:
```
# Replacing Brands with less than 10 data point to Others (grouping them)
car["Brand"] = car["Brand"].replace("Force", "Others")
car["Brand"] = car["Brand"].replace("Isuzu", "Others")
car["Brand"] = car["Brand"].replace("Lamborghini", "Others")
car["Brand"] = car["Brand"].replace("Smart", "Others")
car["Brand"] = car["Brand"].replace("Ambassador", "Others")
car["Brand"] = car["Brand"].replace("Bentley", "Others")
```

In [46]:
```
car["Brand"].value_counts()
```

Out[46]:
```
Maruti           1211
Hyundai          1107
Honda             608
Toyota            411
Mercedes-Benz     318
Volkswagen        315
Ford              300
Mahindra          272
Bmw               267
Audi              236
```

```
Tata             186
Skoda            173
Renault          145
Chevrolet        121
Nissan            91
Land              60
Jaguar            40
Fiat              28
Mitsubishi        27
Mini              26
Volvo             21
Porsche           18
Jeep              15
Datsun            13
Others            10
Name: Brand, dtype: int64
```

## 2. Converting `categorical variables`

> We already know that the data-type of these columns (*Brand, Location, Fuel_Type, Transmission, Owner_Type, Seats*) is object. So, we need to convert them to categorical type for further processing in the next steps.

In [47]:
```
car["Brand"] = car["Brand"].astype("category")
car["Location"] = car["Location"].astype("category")
car["Fuel_Type"] = car["Fuel_Type"].astype("category")
car["Transmission"] = car["Transmission"].astype("category")
car["Owner_Type"] = car["Owner_Type"].astype("category")
car["Seats"] = car["Seats"].astype("category")
```

In [48]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 0 to 6018
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   Location           6019 non-null   category
 1   Year               6019 non-null   int64   
 2   Kilometers_Driven  6019 non-null   int64   
 3   Fuel_Type          6019 non-null   category
 4   Transmission       6019 non-null   category
 5   Owner_Type         6019 non-null   category
 6   Mileage            6019 non-null   float64 
 7   Engine             6019 non-null   float64 
 8   Power              6019 non-null   float64 
 9   Seats              6019 non-null   category
 10  Price              6019 non-null   float64 
 11  Brand              6019 non-null   category
dtypes: category(6), float64(4), int64(2)
memory usage: 526.5 KB
```

In [49]:
```
car.head()
```

Out[49]:

| | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | Engin |
|---|---|---|---|---|---|---|---|---|
| **0** | Mumbai | 2010 | 72000 | CNG | Manual | First | 26.60 | 998. |
| **1** | Pune | 2015 | 41000 | Diesel | Manual | First | 19.67 | 1582. |
| **2** | Chennai | 2011 | 46000 | Petrol | Manual | First | 18.20 | 1199. |
| **3** | Chennai | 2012 | 87000 | Diesel | Manual | First | 20.77 | 1248. |

| | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | Engin |
|---|---|---|---|---|---|---|---|---|
| 4 | Coimbatore | 2013 | 40670 | Diesel | Automatic | Second | 15.20 | 1968. |

## 3. Binning `Seats`

> Sometimes binning is necessary, it tends to improve the performance of the model. Seats is mostly concentrated on 5, so we'll do here for `Seats` .

In [50]:
```
car["Seats"].value_counts()
```

Out[50]:
```
5.0     5056
7.0      674
8.0      134
4.0       99
6.0       31
2.0       16
10.0       5
9.0        3
0.0        1
Name: Seats, dtype: int64
```

In [51]:
```
# can add custom labels
car['Seats_bin'] = pd.cut(
    car['Seats'], [-np.inf, 4, 7, np.inf],
    labels = ["Under 5", "5 to 7", "Over 7"]
)
car.drop(['Seats'], axis=1, inplace=True)
car['Seats_bin'].value_counts(dropna=False)
```

Out[51]:
```
5 to 7     5761
Over 7      142
Under 5     116
Name: Seats_bin, dtype: int64
```

In [52]:
```
car.tail(5)
```

Out[52]:

| | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | En |
|---|---|---|---|---|---|---|---|---|
| **6014** | Delhi | 2014 | 27365 | Diesel | Manual | First | 28.40 | 12 |
| **6015** | Jaipur | 2015 | 100000 | Diesel | Manual | First | 24.40 | 11 |
| **6016** | Jaipur | 2012 | 55000 | Diesel | Manual | Second | 14.00 | 24 |
| **6017** | Kolkata | 2013 | 46000 | Petrol | Manual | First | 18.90 | 9 |
| **6018** | Hyderabad | 2011 | 47000 | Diesel | Manual | First | 25.44 | 9 |

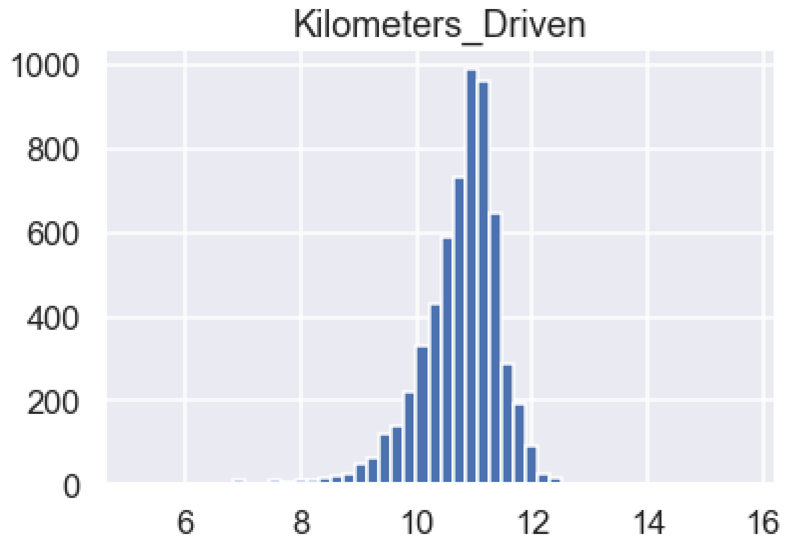
## Checking for duplicated rows

## 4. Log transformation

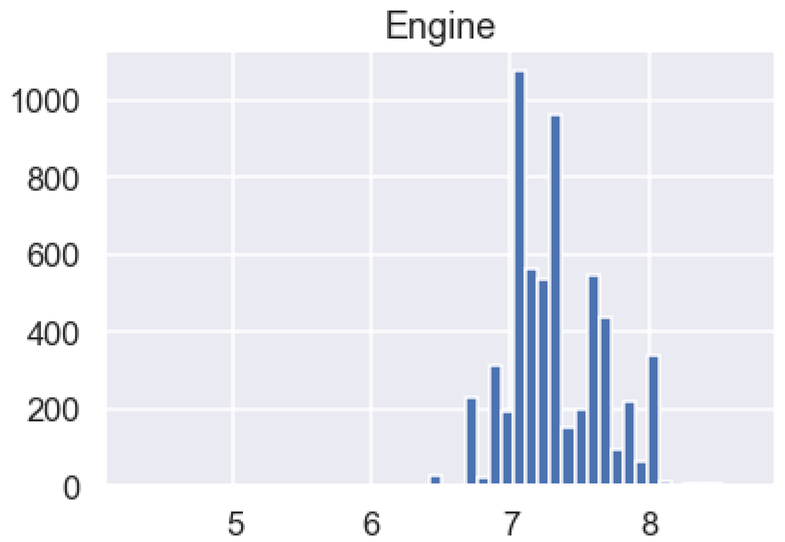
> Some features are very skewed and will likely behave better on the log scale.

> I'll transform `Kilometers_Driven` , `Engine` , `Power` and `Price` .

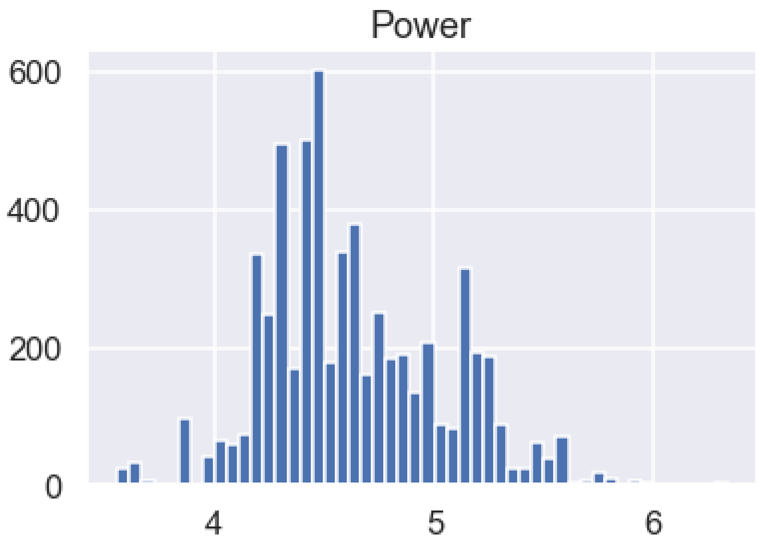
In [53]:
```
cols_to_log = ["Kilometers_Driven", "Engine", "Power", "Price"]

for colname in cols_to_log:
    car[colname] = np.log(car[colname] + 1)


for colname in cols_to_log:
    plt.hist(car[colname], bins=50)
    plt.title(colname)
    plt.show()
    print(np.sum(car[colname] <= 0))
```




```
0
```




```
0
```




0


0

## Observation

- After appling Log to transform skewed data to approximately conform to normality, we can observe that it reduce skewness.

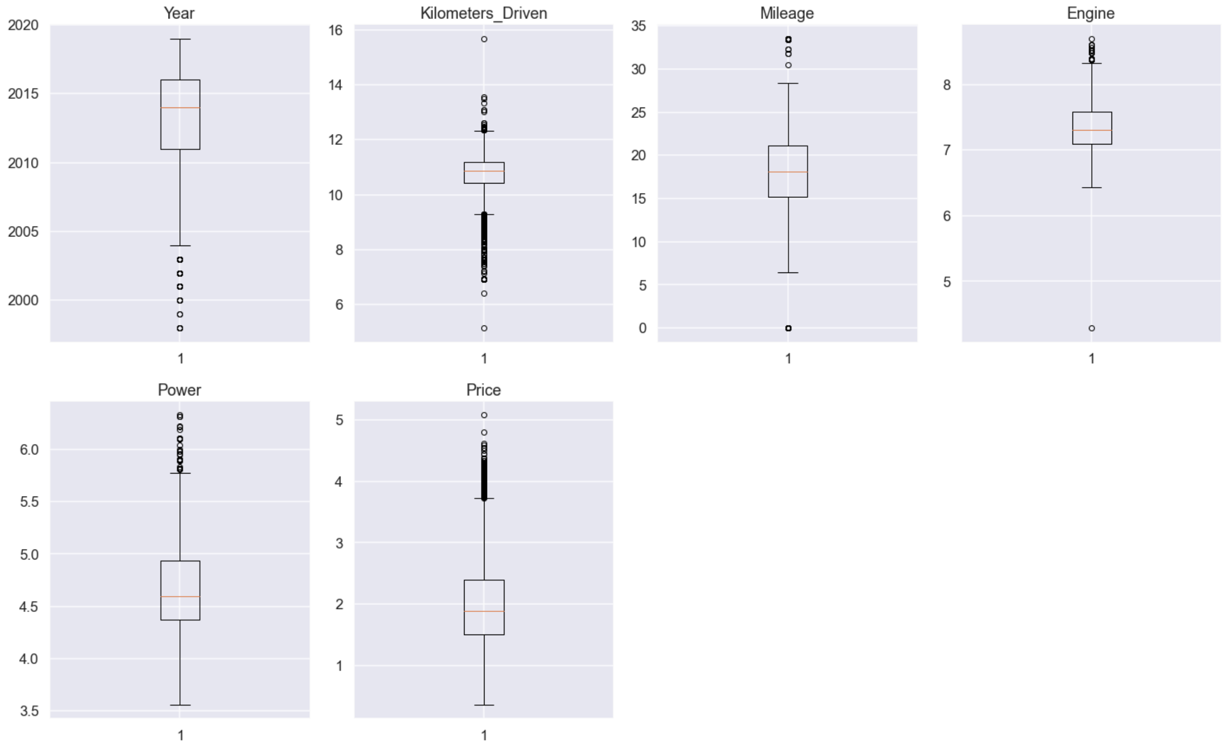
In [54]:
```
car.describe().T
```

Out[54]:

| | count | mean | std | min | 25% | 50% | |
|---|---|---|---|---|---|---|---|
| **Year** | 6019.0 | 2013.358199 | 3.269742 | 1998.000000 | 2011.000000 | 2014.000000 | 20 |
| **Kilometers_Driven** | 6019.0 | 10.758812 | 0.715736 | 5.147494 | 10.434145 | 10.878066 | |
| **Mileage** | 6019.0 | 18.134966 | 4.581528 | 0.000000 | 15.170000 | 18.150000 | |
| **Engine** | 6019.0 | 7.331420 | 0.339041 | 4.290459 | 7.089243 | 7.309212 | |
| **Power** | 6019.0 | 4.646652 | 0.407664 | 3.561046 | 4.369448 | 4.592085 | |
| **Price** | 6019.0 | 2.018429 | 0.748221 | 0.364643 | 1.504077 | 1.893112 | |

## 5. Checking at outliers in every numeric column

In [55]:
```python
# let's plot the boxplots of all columns to check for outliers
numeric_columns1 = car.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numeric_columns1):
    plt.subplot(5, 4, i + 1)
    plt.boxplot(car[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```




## 5.1 Outlier Treatment

In [56]:
```python
# Let's treat outliers by flooring and capping
def treat_outliers(df, col):
    """
    treats outliers in a variable
    col: str, name of the numerical variable
    df: dataframe
    col: name of the column
    """
    Q1 = df[col].quantile(0.25)  # 25th quantile
    Q3 = df[col].quantile(0.75)  # 75th quantile
    IQR = Q3 - Q1
    Lower_Whisker = Q1 - 1.5 * IQR
    Upper_Whisker = Q3 + 1.5 * IQR

    # all the values smaller than Lower_Whisker will be assigned the value of
    # all the values greater than Upper_Whisker will be assigned the value of
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker)

    return df


def treat_outliers_all(df, col_list):
    """
    treat outlier in all numerical variables
```

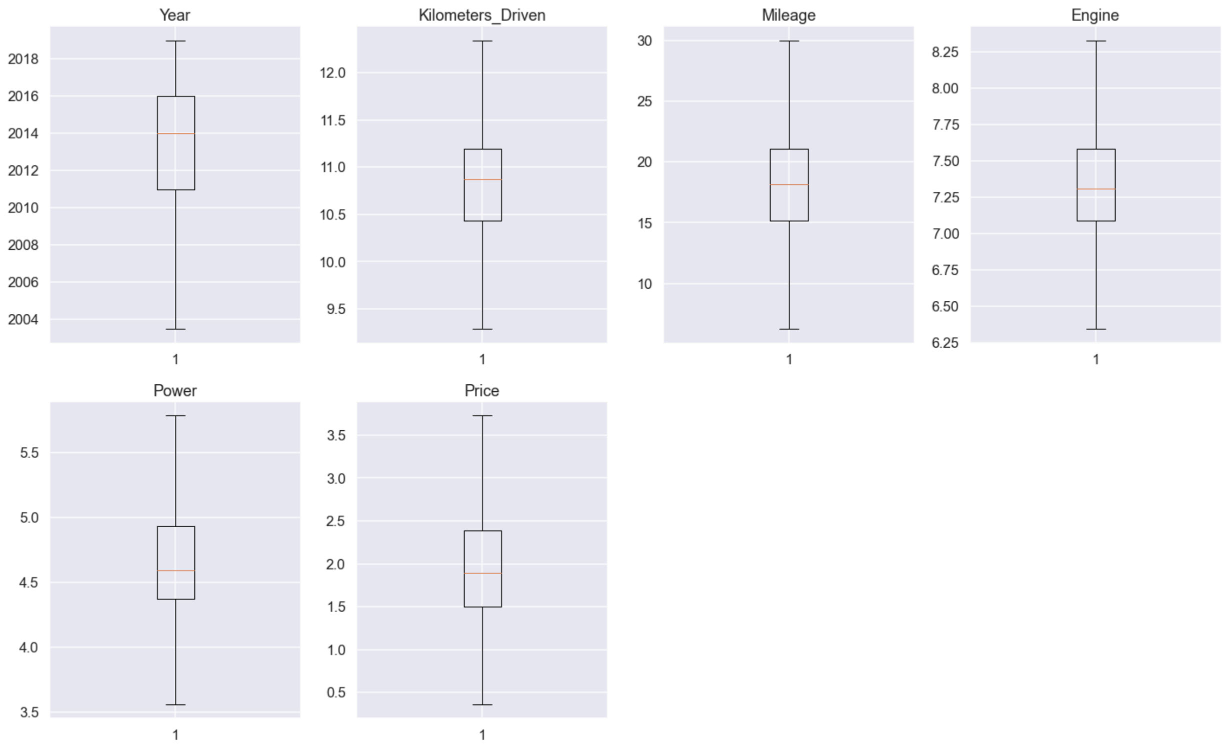
```
    col_list: list of numerical variables
    df: data frame
    """
    for c in col_list:
        df = treat_outliers(df, c)

    return df
```

In [57]:

```
# Treating the outliers
numerical_col1 = car.select_dtypes(include=np.number).columns.tolist()
car = treat_outliers_all(car, numerical_col1)
```

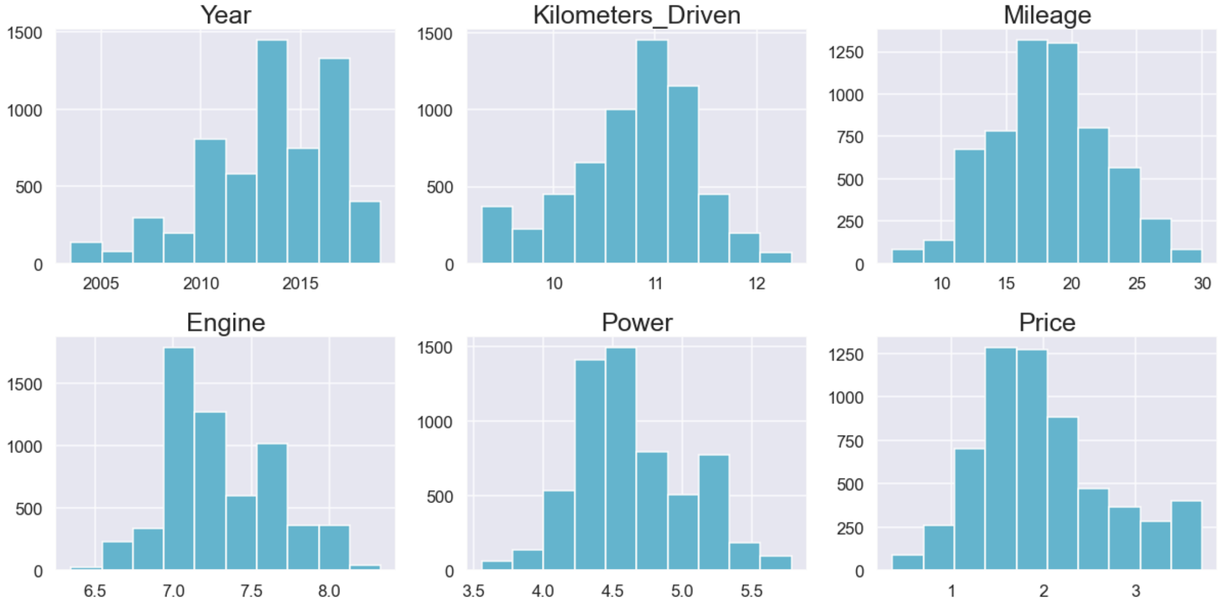
In [58]:

```
# let's look at box plot to see if outliers have been treated or not
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numeric_columns1):
    plt.subplot(5, 4, i + 1)
    plt.boxplot(car[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```




In [59]:

```
all_col = car.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(17, 75))

for i in range(len(all_col)):
    plt.subplot(18, 3, i + 1)
    plt.hist(car[all_col[i]], color="c")
    # sns.histplot(car[all_col[i]], kde=True)  # you can comment the previous
    plt.tight_layout()
    plt.title(all_col[i], fontsize=25)

plt.show()
```




In [60]:
```
car[car.columns[:]].corr()["Price"][:]
```

Out[60]:
```
Year                 0.477991
Kilometers_Driven   -0.215410
Mileage             -0.294237
Engine               0.703444
Power                0.784418
Price                1.000000
Name: Price, dtype: float64
```

In [61]:
```
car.describe().T
```

Out[61]:

| | count | mean | std | min | 25% | 50% |
|---|---|---|---|---|---|---|
| **Year** | 6019.0 | 2013.374149 | 3.213540 | 2003.500000 | 2011.000000 | 2014.000000 |
| **Kilometers_Driven** | 6019.0 | 10.783243 | 0.623619 | 9.288020 | 10.434145 | 10.878066 |
| **Mileage** | 6019.0 | 18.199198 | 4.322077 | 6.275000 | 15.170000 | 18.150000 |
| **Engine** | 6019.0 | 7.331264 | 0.335376 | 6.344425 | 7.089243 | 7.309212 |
| **Power** | 6019.0 | 4.645526 | 0.404051 | 3.561046 | 4.369448 | 4.592085 |
| **Price** | 6019.0 | 2.010468 | 0.727227 | 0.364643 | 1.504077 | 1.893112 |

In [62]:
```
car.head()
```

Out[62]:

| | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | E |
|---|---|---|---|---|---|---|---|---|
| **0** | Mumbai | 2010.0 | 11.184435 | CNG | Manual | First | 26.60 | 6.9( |
| **1** | Pune | 2015.0 | 10.621352 | Diesel | Manual | First | 19.67 | 7.3( |
| **2** | Chennai | 2011.0 | 10.736418 | Petrol | Manual | First | 18.20 | 7.0( |
| **3** | Chennai | 2012.0 | 11.373675 | Diesel | Manual | First | 20.77 | 7.13 |
| **4** | Coimbatore | 2013.0 | 10.613271 | Diesel | Automatic | Second | 15.20 | 7.5 |

In [63]:
```
car.shape
```

Out[63]:
```
(6019, 12)
```

# Data Preparation for Modeling

In [64]:
```
# defining X and y variables
X = car.drop(["Price"], axis=1)
y = car[["Price"]]

print(X.head())
print(y.head())
```

```
     Location    Year  Kilometers_Driven Fuel_Type Transmission Owner_Type  \
0      Mumbai  2010.0          11.184435       CNG       Manual      First
1        Pune  2015.0          10.621352    Diesel       Manual      First
2     Chennai  2011.0          10.736418    Petrol       Manual      First
3     Chennai  2012.0          11.373675    Diesel       Manual      First
4  Coimbatore  2013.0          10.613271    Diesel    Automatic     Second

   Mileage    Engine     Power    Brand Seats_bin
0    26.60  6.906755  4.080246   Maruti    5 to 7
1    19.67  7.367077  4.845761  Hyundai    5 to 7
2    18.20  7.090077  4.496471    Honda    5 to 7
3    20.77  7.130099  4.497139   Maruti    5 to 7
4    15.20  7.585281  4.954418     Audi    5 to 7
      Price
0  1.011601
1  2.602690
2  1.704748
3  1.945910
4  2.930660
```

In [65]:
```
print(X.shape)
print(y.shape)
```

```
(6019, 11)
(6019, 1)
```

In [66]:
```
# creating dummy variables
X = pd.get_dummies(
    X,
    columns=[
        "Brand",
        "Location",
        "Fuel_Type",
        "Transmission",
        "Owner_Type",
        "Seats_bin",
    ],
    drop_first=True,
)
X.head()
```

Out[66]:

| | Year | Kilometers_Driven | Mileage | Engine | Power | Brand_Bmw | Brand_Chevrolet | Bra |
|---|---|---|---|---|---|---|---|---|
| **0** | 2010.0 | 11.184435 | 26.60 | 6.906755 | 4.080246 | 0 | 0 | |
| **1** | 2015.0 | 10.621352 | 19.67 | 7.367077 | 4.845761 | 0 | 0 | |
| **2** | 2011.0 | 10.736418 | 18.20 | 7.090077 | 4.496471 | 0 | 0 | |

| | Year | Kilometers_Driven | Mileage | Engine | Power | Brand_Bmw | Brand_Chevrolet | Bra |
|---|---|---|---|---|---|---|---|---|
| **3** | 2012.0 | 11.373675 | 20.77 | 7.130099 | 4.497139 | 0 | 0 | |
| **4** | 2013.0 | 10.613271 | 15.20 | 7.585281 | 4.954418 | 0 | 0 | |

In [67]:
```
X.shape
```

Out[67]: (6019, 49)

In [68]:
```
# split the data into train and test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

In [69]:
```
X_train.head()
```

Out[69]:

| | Year | Kilometers_Driven | Mileage | Engine | Power | Brand_Bmw | Brand_Chevrolet |
|---|---|---|---|---|---|---|---|
| **4201** | 2011.0 | 11.251574 | 22.07 | 7.090077 | 4.316154 | 0 | 0 |
| **4383** | 2016.0 | 9.900884 | 20.36 | 7.088409 | 4.380776 | 0 | 0 |
| **1779** | 2014.0 | 11.169928 | 15.10 | 7.687080 | 4.948760 | 0 | 0 |
| **4020** | 2013.0 | 11.654390 | 25.20 | 7.130099 | 4.317488 | 0 | 0 |
| **3248** | 2011.0 | 10.981097 | 16.47 | 7.089243 | 4.316154 | 0 | 0 |

# Choose, train and evaluate the model

In [70]:
```
# fitting the model on the train data (70% of the whole data)
from sklearn.linear_model import LinearRegression

linearregression = LinearRegression()
linearregression.fit(X_train, y_train)
```

Out[70]: LinearRegression()

In [71]:
```
# predictions on the test set
pred = linearregression.predict(X_test)

df = pd.DataFrame({"Actual": y_test.values.flatten(), "Predicted": pred.flatt
df
```

Out[71]:

| | Actual | Predicted |
|---|---|---|
| **0** | 1.909543 | 1.772028 |
| **1** | 2.405142 | 2.457937 |

| | Actual | Predicted |
|---|---|---|
| **2** | 2.180417 | 2.162881 |
| **3** | 1.223775 | 1.494487 |
| **4** | 0.955511 | 1.416018 |
| **...** | ... | ... |
| **1801** | 2.348514 | 2.509778 |
| **1802** | 1.435085 | 1.325882 |
| **1803** | 1.658228 | 1.531062 |
| **1804** | 2.012233 | 2.074766 |
| **1805** | 1.932970 | 2.066552 |

1806 rows × 2 columns

- We can observe that the model has returned good prediction results, and the actual and predicted values are a little bit different, but closer to each other.

In [72]:
```
# let us check the coefficients and intercept of the model

coef_car = pd.DataFrame(
    np.append(linearregression.coef_[0], linearregression.intercept_[0]),
    index=X_train.columns.tolist() + ["Intercept"],
    columns=["Coefficients"],
)

coef_car
```

Out[72]:

| | Coefficients |
|---|---|
| **Year** | 0.094236 |
| **Kilometers_Driven** | -0.075441 |
| **Mileage** | -0.012928 |
| **Engine** | 0.286198 |
| **Power** | 0.558538 |
| **Brand_Bmw** | -0.048642 |
| **Brand_Chevrolet** | -0.806758 |
| **Brand_Datsun** | -0.831162 |
| **Brand_Fiat** | -0.751407 |
| **Brand_Ford** | -0.611105 |
| **Brand_Honda** | -0.584284 |
| **Brand_Hyundai** | -0.577294 |
| **Brand_Jaguar** | 0.054469 |
| **Brand_Jeep** | -0.371983 |
| **Brand_Land** | 0.249507 |
| **Brand_Mahindra** | -0.664240 |

| | Coefficients |
|---|---|
| Brand_Maruti | -0.518786 |
| Brand_Mercedes-Benz | -0.040846 |
| Brand_Mini | 0.249277 |
| Brand_Mitsubishi | -0.365001 |
| Brand_Nissan | -0.609744 |
| Brand_Others | -0.506298 |
| Brand_Porsche | 0.251120 |
| Brand_Renault | -0.593479 |
| Brand_Skoda | -0.561615 |
| Brand_Tata | -0.912179 |
| Brand_Toyota | -0.356701 |
| Brand_Volkswagen | -0.614767 |
| Brand_Volvo | -0.137102 |
| Location_Bangalore | 0.105262 |
| Location_Chennai | 0.008942 |
| Location_Coimbatore | 0.082088 |
| Location_Delhi | -0.078622 |
| Location_Hyderabad | 0.081992 |
| Location_Jaipur | -0.036585 |
| Location_Kochi | -0.040128 |
| Location_Kolkata | -0.202746 |
| Location_Mumbai | -0.053760 |
| Location_Pune | -0.034739 |
| Fuel_Type_Diesel | 0.155795 |
| Fuel_Type_Electric | 1.320921 |
| Fuel_Type_LPG | -0.004938 |
| Fuel_Type_Petrol | -0.046071 |
| Transmission_Manual | -0.094239 |
| Owner_Type_Fourth & Above | -0.076740 |
| Owner_Type_Second | -0.044371 |
| Owner_Type_Third | -0.102263 |
| Seats_bin_5 to 7 | -0.162816 |
| Seats_bin_Over 7 | -0.148451 |
| Intercept | -190.695029 |

In [73]:

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

```python
# defining common function for all metrics
def model_perf(model, inp, out):
    """
    model: model
    inp: independent variables
    out: dependent variable
    """
    y_pred = model.predict(inp).flatten()
    y_act = out.values.flatten()

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

In [74]:

```python
# Checking model performance on train set (seen 70% data)
print("Train Performance\n")
model_perf(linearregression, X_train, y_train)
```

```
Train Performance
```

Out[74]:

| | MAE | MAPE | RMSE | R^2 |
|---|---|---|---|---|
| **0** | 0.139745 | 7.987014 | 0.182886 | 0.93581 |

In [75]:

```python
# Checking model performance on test set (unseen 30% data)
print("Test Performance\n")
model_perf(linearregression, X_test, y_test)
```

```
Test Performance
```

Out[75]:

| | MAE | MAPE | RMSE | R^2 |
|---|---|---|---|---|
| **0** | 0.147489 | 8.620238 | 0.20303 | 0.924601 |

**Observations**

- The training and testing scores are 93.5% and 92.4% respectively, and both the scores are comparable. Hence, the model is a good fit.
- R-squared is 0.925 on the test set, i.e., the model explains 92.4% of total variation in the test dataset. So, overall the model is very satisfactory.
- MAE indicates that our current model is able to predict Price within a mean error of 0.14 Lakhs on the test data.
- MAPE on the test set suggests we can predict within 8.6% of the Price.

# Linear Regression using statsmodels

In [76]:
```
# Let's build linear regression model using statsmodel
# unlike sklearn, statsmodels does not add a constant to the data on its own
# we have to add the constant manually
# import pdb; pdb.set_trace()

import statsmodels.api as sm

X = sm.add_constant(X)

X_train1, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

olsmod0 = sm.OLS(y_train, X_train1)
olsres0 = olsmod0.fit()
print(olsres0.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  Price   R-squared:                       0.93
6
Model:                            OLS   Adj. R-squared:                  0.93
5
Method:                 Least Squares   F-statistic:                     123
9.
Date:                Sat, 19 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        01:34:49   Log-Likelihood:                 1179.
4
No. Observations:                4213   AIC:                            -225
9.
Df Residuals:                    4163   BIC:                            -194
2.
Df Model:                          49
Covariance Type:            nonrobust
==============================================================================
================
                            coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
----------------
const                  -190.6950      2.728    -69.895      0.000     -19
6.044    -185.346
Year                      0.0942      0.001     70.134      0.000
0.092       0.097
Kilometers_Driven        -0.0754      0.006    -11.895      0.000      -
0.088      -0.063
Mileage                  -0.0129      0.001    -10.262      0.000      -
0.015      -0.010
Engine                    0.2862      0.028     10.048      0.000
0.230       0.342
Power                     0.5585      0.021     26.652      0.000
0.517       0.600
Brand_Bmw                -0.0486      0.020     -2.420      0.016      -
0.088      -0.009
Brand_Chevrolet          -0.8068      0.027    -29.735      0.000      -
0.860      -0.754
Brand_Datsun             -0.8312      0.073    -11.408      0.000      -
0.974      -0.688
Brand_Fiat               -0.7514      0.047    -15.927      0.000      -
0.844      -0.659
Brand_Ford               -0.6111      0.022    -28.143      0.000      -
0.654      -0.569
Brand_Honda              -0.5843      0.020    -29.456      0.000      -
0.623      -0.545
Brand_Hyundai            -0.5773      0.019    -29.934      0.000      -
0.615      -0.539
```

```
Brand_Jaguar                 0.0545      0.037      1.457      0.145      -
0.019       0.128
Brand_Jeep                  -0.3720      0.058     -6.378      0.000      -
0.486      -0.258
Brand_Land                   0.2495      0.033      7.582      0.000
0.185       0.314
Brand_Mahindra              -0.6642      0.023    -29.439      0.000      -
0.708      -0.620
Brand_Maruti                -0.5188      0.020    -25.735      0.000      -
0.558      -0.479
Brand_Mercedes-Benz         -0.0408      0.019     -2.143      0.032      -
0.078      -0.003
Brand_Mini                   0.2493      0.049      5.128      0.000
0.154       0.345
Brand_Mitsubishi            -0.3650      0.046     -7.862      0.000      -
0.456      -0.274
Brand_Nissan                -0.6097      0.029    -20.912      0.000      -
0.667      -0.553
Brand_Others                -0.5063      0.077     -6.534      0.000      -
0.658      -0.354
Brand_Porsche                0.2511      0.061      4.128      0.000
0.132       0.370
Brand_Renault               -0.5935      0.026    -22.562      0.000      -
0.645      -0.542
Brand_Skoda                 -0.5616      0.023    -24.413      0.000      -
0.607      -0.517
Brand_Tata                  -0.9122      0.025    -36.145      0.000      -
0.962      -0.863
Brand_Toyota                -0.3567      0.021    -16.957      0.000      -
0.398      -0.315
Brand_Volkswagen            -0.6148      0.021    -28.695      0.000      -
0.657      -0.573
Brand_Volvo                 -0.1371      0.048     -2.834      0.005      -
0.232      -0.042
Location_Bangalore           0.1053      0.019      5.573      0.000
0.068       0.142
Location_Chennai             0.0089      0.018      0.493      0.622      -
0.027       0.045
Location_Coimbatore          0.0821      0.018      4.691      0.000
0.048       0.116
Location_Delhi              -0.0786      0.018     -4.458      0.000      -
0.113      -0.044
Location_Hyderabad           0.0820      0.017      4.809      0.000
0.049       0.115
Location_Jaipur             -0.0366      0.019     -1.965      0.049      -
0.073   -9.13e-05
Location_Kochi              -0.0401      0.017     -2.303      0.021      -
0.074      -0.006
Location_Kolkata            -0.2027      0.018    -11.347      0.000      -
0.238      -0.168
Location_Mumbai             -0.0538      0.017     -3.145      0.002      -
0.087      -0.020
Location_Pune               -0.0347      0.017     -1.994      0.046      -
0.069      -0.001
Fuel_Type_Diesel             0.1558      0.031      5.061      0.000
0.095       0.216
Fuel_Type_Electric           1.3209      0.189      6.990      0.000
0.950       1.691
Fuel_Type_LPG               -0.0049      0.072     -0.068      0.946      -
0.147       0.137
Fuel_Type_Petrol            -0.0461      0.031     -1.479      0.139      -
0.107       0.015
Transmission_Manual         -0.0942      0.009     -9.947      0.000      -
0.113      -0.076
Owner_Type_Fourth & Above   -0.0767      0.083     -0.926      0.354      -
0.239       0.086
Owner_Type_Second           -0.0444      0.008     -5.285      0.000      -
0.061      -0.028
Owner_Type_Third            -0.1023      0.022     -4.732      0.000      -
```

```
0.145      -0.060
Seats_bin_5 to 7           -0.1628      0.023     -6.967      0.000      -
0.209      -0.117
Seats_bin_Over 7           -0.1485      0.031     -4.764      0.000      -
0.210      -0.087
==============================================================================
=
Omnibus:                    278.469   Durbin-Watson:                   1.96
7
Prob(Omnibus):                0.000   Jarque-Bera (JB):              1314.30
4
Skew:                        -0.065   Prob(JB):                     4.00e-28
6
Kurtosis:                     5.733   Cond. No.                     1.94e+0
6
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.94e+06. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

**Observations**

- Negative values of the coefficient show that *Price* decreases with the increase of corresponding attribute value.
- Positive values of the coefficient show that *Price* increases with the increase of corresponding attribute value.
- p-value of a variable indicates if the variable is significant or not. we gonna consider the significance level to be 0.05 (5%).
- But these variables might contain multicollinearity, which will affect the p-values.
- So, lets deal with multicollinearity and check the other assumptions of linear regression first, and then look at the p-values.

# Checking Linear Regression Assumptions

Checking the following 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**

## 1. TEST FOR MULTICOLLINEARITY

In [77]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
vif_series1 = pd.Series(
    [variance_inflation_factor(X.values, i) for i in range(X.shape[1])], inde
)
print("VIF Scores: \n\n{}\n".format(vif_series1))
```

```
VIF Scores:

const                          920364.486869
Year                                2.303025
Kilometers_Driven                   1.943818
Mileage                             3.534225
Engine                             11.050082
Power                               8.847576
Brand_Bmw                           2.105615
Brand_Chevrolet                     1.787446
Brand_Datsun                        1.123072
Brand_Fiat                          1.190172
Brand_Ford                          2.825584
Brand_Honda                         4.424763
Brand_Hyundai                       6.872355
Brand_Jaguar                        1.176361
Brand_Jeep                          1.095515
Brand_Land                          1.273144
Brand_Mahindra                      2.701392
Brand_Maruti                        8.119727
Brand_Mercedes-Benz                 2.256220
Brand_Mini                          1.205947
Brand_Mitsubishi                    1.182377
Brand_Nissan                        1.602841
Brand_Others                        1.069538
Brand_Porsche                       1.168474
Brand_Renault                       1.944096
Brand_Skoda                         1.853299
Brand_Tata                          2.321093
Brand_Toyota                        3.499852
Brand_Volkswagen                    2.854993
Brand_Volvo                         1.089394
Location_Bangalore                  2.485958
Location_Chennai                    3.006042
Location_Coimbatore                 3.541483
Location_Delhi                      3.179997
Location_Hyderabad                  3.832402
Location_Jaipur                     2.693769
Location_Kochi                      3.590784
Location_Kolkata                    3.155256
Location_Mumbai                     4.030378
Location_Pune                       3.443976
Fuel_Type_Diesel                   29.082733
Fuel_Type_Electric                  1.051184
Fuel_Type_LPG                       1.196637
Fuel_Type_Petrol                   29.591562
Transmission_Manual                 2.294648
Owner_Type_Fourth & Above           1.016666
Owner_Type_Second                   1.179123
Owner_Type_Third                    1.113021
Seats_bin_5 to 7                    2.726364
Seats_bin_Over 7                    2.778880
dtype: float64
```

- *Engine* have a VIF score greater than 10, let's dropped it and check the model R2

## Removing Multicollinearity

To remove multicollinearity

1. Drop every column one by one that has VIF score greater than 10.
2. Look at the adjusted R-squared of all these models.
3. Drop the variable that makes least change in adjusted R-squared.
4. Check the VIF scores again.
5. Continue till you get all VIF scores under 10.

In [78]:
```
# we drop the one with the highest vif values (Engine) and check the adjusted
X_train2 = X_train1.drop("Engine", axis=1)
vif_series2 = pd.Series(
    [variance_inflation_factor(X_train2.values, i) for i in range(X_train2.sh
    index=X_train2.columns,
)
print("VIF Scores: \n\n{}\n".format(vif_series2))
```

```
VIF Scores:

const                          920075.285563
Year                                2.277008
Kilometers_Driven                   1.930246
Mileage                             3.116889
Power                               4.149090
Brand_Bmw                           2.088459
Brand_Chevrolet                     1.769466
Brand_Datsun                        1.095832
Brand_Fiat                          1.177468
Brand_Ford                          2.849609
Brand_Honda                         4.373398
Brand_Hyundai                       6.997639
Brand_Jaguar                        1.187239
Brand_Jeep                          1.101465
Brand_Land                          1.267555
Brand_Mahindra                      2.635627
Brand_Maruti                        8.113328
Brand_Mercedes-Benz                 2.238045
Brand_Mini                          1.179493
Brand_Mitsubishi                    1.168991
Brand_Nissan                        1.594368
Brand_Others                        1.055005
Brand_Porsche                       1.186635
Brand_Renault                       1.897713
Brand_Skoda                         1.865378
Brand_Tata                          2.296495
Brand_Toyota                        3.275574
Brand_Volkswagen                    2.845089
Brand_Volvo                         1.100997
Location_Bangalore                  2.598754
Location_Chennai                    3.024490
Location_Coimbatore                 3.607807
Location_Delhi                      3.265957
Location_Hyderabad                  3.908331
Location_Jaipur                     2.702022
Location_Kochi                      3.668755
Location_Kolkata                    3.229806
Location_Mumbai                     4.086660
Location_Pune                       3.529917
Fuel_Type_Diesel                   29.329026
Fuel_Type_Electric                  1.044866
Fuel_Type_LPG                       1.229275
Fuel_Type_Petrol                   29.562725
Transmission_Manual                 2.276387
Owner_Type_Fourth & Above           1.012832
Owner_Type_Second                   1.176127
Owner_Type_Third                    1.122574
Seats_bin_5 to 7                    2.838834
Seats_bin_Over 7                    2.842221
```

```
dtype: float64
```

- It seemed to have helped, VIF has come down, and there is no other variable greater than 10 besides dummies(categorical).

In [79]:
```
olsmod1 = sm.OLS(y_train, X_train2)
olsres1 = olsmod1.fit()
print(olsres1.summary())
```

```
                            OLS Regression Results                            
==============================================================================
=
Dep. Variable:                  Price   R-squared:                       0.93
4
Model:                            OLS   Adj. R-squared:                  0.93
3
Method:                 Least Squares   F-statistic:                     123
3.
Date:                Sat, 19 Jun 2021   Prob (F-statistic):               0.0
0
Time:                        01:34:50   Log-Likelihood:                 1129.
0
No. Observations:                4213   AIC:                            -216
0.
Df Residuals:                    4164   BIC:                            -184
9.
Df Model:                          48                                         
Covariance Type:            nonrobust                                         
==============================================================================
================
                             coef    std err          t      P>|t|      
[0.025      0.975]
------------------------------------------------------------------------------
----------------
const                   -188.4166      2.751    -68.482      0.000    -19
3.811    -183.023
Year                       0.0938      0.001     69.037      0.000       
0.091       0.096
Kilometers_Driven         -0.0739      0.006    -11.522      0.000      -
0.087      -0.061
Mileage                   -0.0180      0.001    -15.400      0.000      -
0.020      -0.016
Power                      0.7124      0.014     49.213      0.000       
0.684       0.741
Brand_Bmw                 -0.0477      0.020     -2.346      0.019      -
0.088      -0.008
Brand_Chevrolet           -0.8075      0.027    -29.411      0.000      -
0.861      -0.754
Brand_Datsun              -0.8304      0.074    -11.264      0.000      -
0.975      -0.686
Brand_Fiat                -0.7655      0.048    -16.042      0.000      -
0.859      -0.672
Brand_Ford                -0.5847      0.022    -26.805      0.000      -
0.627      -0.542
Brand_Honda               -0.5675      0.020    -28.375      0.000      -
0.607      -0.528
Brand_Hyundai             -0.5765      0.020    -29.540      0.000      -
0.615      -0.538
Brand_Jaguar               0.0661      0.038      1.749      0.080      -
0.008       0.140
Brand_Jeep                -0.3910      0.059     -6.628      0.000      -
0.507      -0.275
Brand_Land                 0.2502      0.033      7.514      0.000       
0.185       0.316
Brand_Mahindra            -0.6168      0.022    -27.627      0.000      -
0.661      -0.573
```

```
Brand_Maruti               -0.5145      0.020    -25.225      0.000      -
0.554      -0.474
Brand_Mercedes-Benz        -0.0288      0.019     -1.495      0.135      -
0.067       0.009
Brand_Mini                  0.2708      0.049      5.511      0.000
0.174       0.367
Brand_Mitsubishi           -0.2849      0.046     -6.156      0.000      -
0.376      -0.194
Brand_Nissan               -0.5834      0.029    -19.854      0.000      -
0.641      -0.526
Brand_Others               -0.4403      0.078     -5.636      0.000      -
0.593      -0.287
Brand_Porsche               0.3139      0.061      5.127      0.000
0.194       0.434
Brand_Renault              -0.5976      0.027    -22.454      0.000      -
0.650      -0.545
Brand_Skoda                -0.5401      0.023    -23.304      0.000      -
0.586      -0.495
Brand_Tata                 -0.9025      0.026    -35.365      0.000      -
0.953      -0.852
Brand_Toyota               -0.2945      0.020    -14.474      0.000      -
0.334      -0.255
Brand_Volkswagen           -0.6043      0.022    -27.908      0.000      -
0.647      -0.562
Brand_Volvo                -0.1545      0.049     -3.157      0.002      -
0.250      -0.059
Location_Bangalore          0.1054      0.019      5.514      0.000
0.068       0.143
Location_Chennai            0.0103      0.018      0.561      0.575      -
0.026       0.046
Location_Coimbatore         0.0842      0.018      4.752      0.000
0.049       0.119
Location_Delhi             -0.0767      0.018     -4.298      0.000      -
0.112      -0.042
Location_Hyderabad          0.0834      0.017      4.835      0.000
0.050       0.117
Location_Jaipur            -0.0349      0.019     -1.851      0.064      -
0.072       0.002
Location_Kochi             -0.0386      0.018     -2.187      0.029      -
0.073      -0.004
Location_Kolkata           -0.2017      0.018    -11.155      0.000      -
0.237      -0.166
Location_Mumbai            -0.0497      0.017     -2.876      0.004      -
0.084      -0.016
Location_Pune              -0.0305      0.018     -1.732      0.083      -
0.065       0.004
Fuel_Type_Diesel            0.1675      0.031      5.382      0.000
0.106       0.229
Fuel_Type_Electric          1.1366      0.190      5.972      0.000
0.763       1.510
Fuel_Type_LPG              -0.0351      0.073     -0.481      0.631      -
0.178       0.108
Fuel_Type_Petrol           -0.0825      0.031     -2.637      0.008      -
0.144      -0.021
Transmission_Manual        -0.0926      0.010     -9.664      0.000      -
0.111      -0.074
Owner_Type_Fourth & Above  -0.0859      0.084     -1.025      0.306      -
0.250       0.078
Owner_Type_Second          -0.0441      0.008     -5.193      0.000      -
0.061      -0.027
Owner_Type_Third           -0.1019      0.022     -4.660      0.000      -
0.145      -0.059
Seats_bin_5 to 7           -0.1612      0.024     -6.817      0.000      -
0.208      -0.115
Seats_bin_Over 7           -0.1271      0.031     -4.040      0.000      -
0.189      -0.065
=============================================================================
=
Omnibus:                    270.115   Durbin-Watson:                    1.97
```

```
1
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1256.65
3
Skew:                          -0.026   Prob(JB):                    1.32e-27
3
Kurtosis:                       5.675   Cond. No.                    1.93e+0
6
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.93e+06. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

- Earlier adj. R-squared was 0.931, now it is reduced to 0.929.
- Now the above model has no multicollinearity, so we can look at p-values of predictor variables to check their significance.

**Observations**

- There are no p-value greater than 0.05, so they are significant, we'll not drop them.

- On categorical variables, p-value greater than 0.05 doesn't mean we'll drop it, because it is from a categorical variable and there are other levels of this category that are significant.**

## Now we'll check the rest of the assumptions on model *olsres1*

1. Mean of residuals should be 0
2. Linearity of variables
3. Normality of error terms
4. No Heteroscedasticity

## MEAN OF RESIDUALS SHOULD BE 0

In [80]:
```
residual = olsres1.resid
np.mean(residual)
```

Out[80]:
```
-8.475647590997978e-14
```

- Mean of redisuals is very close to 0.

## TEST FOR LINEARITY

**Why the test?**

- Linearity describes a straight-line relationship between two variables, predictor variables must have a linear relation with the dependent variable.

**How to check linearity?**

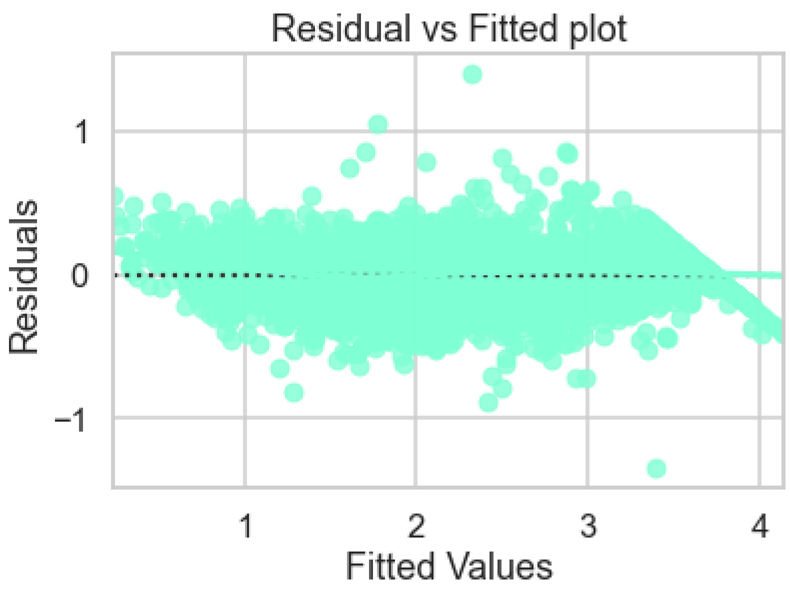
- Make a plot of fitted values vs residuals, if they don't follow any pattern, they we say the model is linear, otherwise model is showing signs of non-linearity.

**How to fix if this assumption is not followed?**

- We can try to transform the variables and make the relationships linear.

In [81]:
```
residual = olsres1.resid
fitted = olsres1.fittedvalues  # predicted values
```

In [82]:
```
sns.set_style("whitegrid")
sns.residplot(fitted, residual, color="aquamarine", lowess=True)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residual vs Fitted plot")
plt.show()
```




- Scatter plot shows the distribution of residuals (errors) vs fitted values (predicted values).
- If there exist any pattern in this plot, we consider it as signs of non-linearity in the data and a pattern means that the model doesn't capture non-linear effects.
- **We see no pattern in the plot above. Hence, the assumption is satisfied.**

## TEST FOR NORMALITY

**What is the test?**

- Error terms/Residuals should be normally distributed
- If the error terms are non- normally distributed, confidence intervals may become too wide or narrow. Once confidence interval becomes unstable, it leads to difficulty in estimating coefficients based on minimization of least squares.

**What do non-normality indicate?**

- It suggests that there are a few unusual data points which must be studied closely to make a better model.

**How to Check the Normality?**

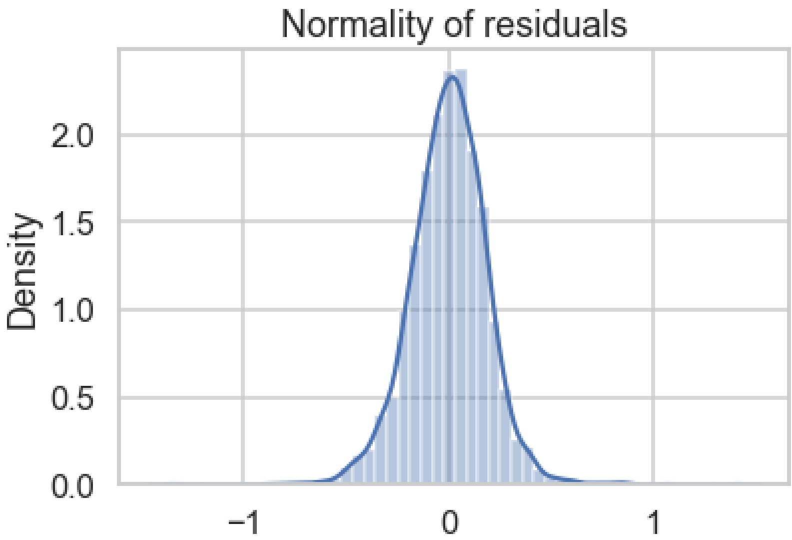
- It can be checked via QQ Plot, Residuals following normal distribution will make a straight line plot otherwise not.

- Other test to check for normality : Shapiro-Wilk test.

**What is the residuals are not-normal?**

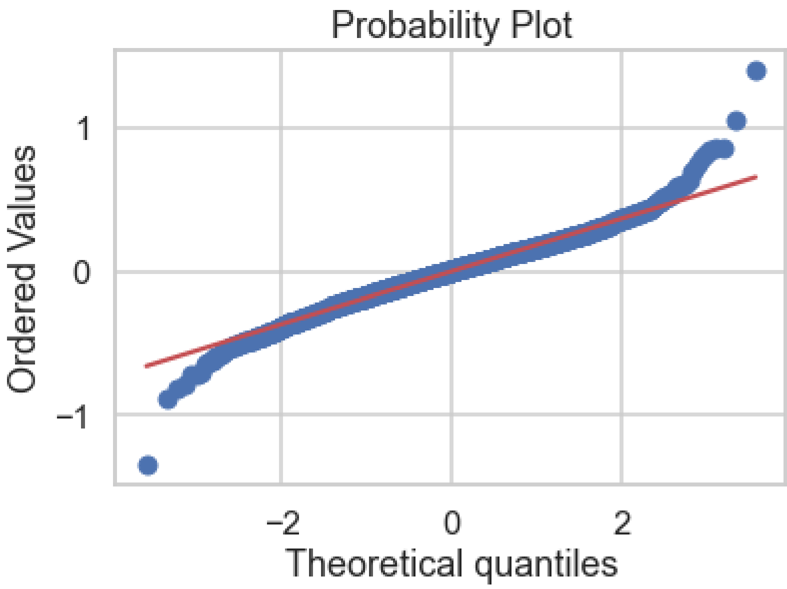
- We can apply transformations like log, exponential, arcsinh, etc. as per our data.

In [83]:
```
sns.distplot(residual)
plt.title("Normality of residuals")
plt.show()
```




## The QQ plot of residuals can be used to visually check the normality assumption. The normal probability plot of residuals should approximately follow a straight line.

In [84]:
```
import pylab
import scipy.stats as stats

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




In [85]:
```
stats.shapiro(residual)
```

Out[85]:
```
ShapiroResult(statistic=0.981600284576416, pvalue=5.493219029554584e-23)
```

- The residuals are not normal as per shapiro test, but as per QQ plot they are approximately normal.
- The issue with shapiro test is when dataset is big, even for small deviations, it shows data as not normal.
- Hence we go with the QQ plot and say that residuals are normal.
- We can try to treat data for outliers and see if that helps in further normalizing the residual curve.

## TEST FOR HOMOSCEDASTICITY

- Test - goldfeldquandt test

- **Homoscedacity**: If the variance of the residuals are symmetrically distributed across the regression line, then the data is said to homoscedastic.

- **Heteroscedacity**: If the variance is unequal for the residuals across the regression line, then the data is said to be heteroscedastic. In this case the residuals can form an arrow shape or any other non symmetrical shape.

For goldfeldquandt test, the null and alternate hypotheses are as follows:

- Null hypothesis : Residuals are homoscedastic
- Alternate hypothesis : Residuals have heteroscedasticity

In [86]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(residual, X_train2)
lzip(name, test)
```

Out[86]:
```
[('F statistic', 1.0389582863285913), ('p-value', 0.19303288531465843)]
```

**Since p-value = 0.193 > 0.05, we can say that the residuals are homoscedastic. This assumption is therefore valid in the data.**

## Predicting on the test data

In [87]:
```
X_train2.columns
```

Out[87]:
```
Index(['const', 'Year', 'Kilometers_Driven', 'Mileage', 'Power', 'Brand_Bmw',
       'Brand_Chevrolet', 'Brand_Datsun', 'Brand_Fiat', 'Brand_Ford',
       'Brand_Honda', 'Brand_Hyundai', 'Brand_Jaguar', 'Brand_Jeep',
       'Brand_Land', 'Brand_Mahindra', 'Brand_Maruti', 'Brand_Mercedes-Benz',
       'Brand_Mini', 'Brand_Mitsubishi', 'Brand_Nissan', 'Brand_Others',
       'Brand_Porsche', 'Brand_Renault', 'Brand_Skoda', 'Brand_Tata',
       'Brand_Toyota', 'Brand_Volkswagen', 'Brand_Volvo', 'Location_Bangalor
e',
       'Location_Chennai', 'Location_Coimbatore', 'Location_Delhi',
       'Location_Hyderabad', 'Location_Jaipur', 'Location_Kochi',
       'Location_Kolkata', 'Location_Mumbai', 'Location_Pune',
       'Fuel_Type_Diesel', 'Fuel_Type_Electric', 'Fuel_Type_LPG',
       'Fuel_Type_Petrol', 'Transmission_Manual', 'Owner_Type_Fourth & Abov
e',
       'Owner_Type_Second', 'Owner_Type_Third', 'Seats_bin_5 to 7',
       'Seats_bin_Over 7'],
      dtype='object')
```

In [88]:
```
# Selecting columns from test data that we used to create our final model
X_test_final = X_test[X_train2.columns]
```

In [89]:
```
X_test_final.head()
```

Out[89]:

| | const | Year | Kilometers_Driven | Mileage | Power | Brand_Bmw | Brand_Chevrolet | Br |
|---|---|---|---|---|---|---|---|---|
| **2868** | 1.0 | 2013.0 | 11.141876 | 23.400 | 4.317488 | 0 | 0 | |
| **5924** | 1.0 | 2017.0 | 10.193991 | 15.400 | 4.795791 | 0 | 0 | |
| **3764** | 1.0 | 2014.0 | 11.362114 | 15.100 | 4.948760 | 0 | 0 | |
| **4144** | 1.0 | 2016.0 | 10.859018 | 25.000 | 4.248638 | 0 | 0 | |
| **2780** | 1.0 | 2009.0 | 11.512935 | 6.275 | 4.592085 | 0 | 0 | |

In [90]:
```
# Checking model performance on train set (seen 70% data)
print("Train Performance\n")
model_perf(olsres1, X_train2.values, y_train)
```

```
Train Performance
```

Out[90]:

| | MAE | MAPE | RMSE | R^2 |
|---|---|---|---|---|
| **0** | 0.141071 | 8.081216 | 0.185091 | 0.934253 |

In [91]:
```
# Checking model performance on test set (seen 70% data)
print("Test Performance\n")
model_perf(olsres1, X_test_final.values, y_test)
```

```
Test Performance
```

Out[91]:

| | MAE | MAPE | RMSE | R^2 |
|---|---|---|---|---|
| **0** | 0.149894 | 8.803791 | 0.206935 | 0.921673 |

- Now we can see that the model has low test and train RMSE and MAE, and both the errors are comparable. So, our model is not suffering from overfitting.
- The model is able to explain 92% of the variation on the test set, which is very good.
- The MAPE on the test set suggests we can predict within 8.8% of the Price.

**Hence, we can conclude the model *olsres1* is good for prediction as well as inference purposes.**

In [92]:

```
# let us print the model summary

olsmod1 = sm.OLS(y_train, X_train2)
olsres1 = olsmod1.fit()
print(olsres1.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  Price   R-squared:                       0.93
4
Model:                            OLS   Adj. R-squared:                  0.93
3
Method:                 Least Squares   F-statistic:                     123
3.
Date:                Sat, 19 Jun 2021   Prob (F-statistic):              0.0
0
Time:                        01:34:53   Log-Likelihood:                1129.
0
No. Observations:                4213   AIC:                           -216
0.
Df Residuals:                    4164   BIC:                           -184
9.
Df Model:                          48
Covariance Type:            nonrobust
==============================================================================
================
                         coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
----------------
const               -188.4166      2.751    -68.482      0.000    -19
3.811    -183.023
Year                   0.0938      0.001     69.037      0.000
0.091       0.096
Kilometers_Driven     -0.0739      0.006    -11.522      0.000       -
0.087      -0.061
Mileage               -0.0180      0.001    -15.400      0.000       -
0.020      -0.016
Power                  0.7124      0.014     49.213      0.000
0.684       0.741
Brand_Bmw             -0.0477      0.020     -2.346      0.019       -
0.088      -0.008
Brand_Chevrolet       -0.8075      0.027    -29.411      0.000       -
0.861      -0.754
Brand_Datsun          -0.8304      0.074    -11.264      0.000       -
0.975      -0.686
Brand_Fiat            -0.7655      0.048    -16.042      0.000       -
```

```
0.859      -0.672
Brand_Ford                  -0.5847      0.022     -26.805      0.000      -
0.627      -0.542
Brand_Honda                 -0.5675      0.020     -28.375      0.000      -
0.607      -0.528
Brand_Hyundai               -0.5765      0.020     -29.540      0.000      -
0.615      -0.538
Brand_Jaguar                 0.0661      0.038       1.749      0.080      -
0.008       0.140
Brand_Jeep                  -0.3910      0.059      -6.628      0.000      -
0.507      -0.275
Brand_Land                   0.2502      0.033       7.514      0.000
0.185       0.316
Brand_Mahindra              -0.6168      0.022     -27.627      0.000      -
0.661      -0.573
Brand_Maruti                -0.5145      0.020     -25.225      0.000      -
0.554      -0.474
Brand_Mercedes-Benz         -0.0288      0.019      -1.495      0.135      -
0.067       0.009
Brand_Mini                   0.2708      0.049       5.511      0.000
0.174       0.367
Brand_Mitsubishi            -0.2849      0.046      -6.156      0.000      -
0.376      -0.194
Brand_Nissan                -0.5834      0.029     -19.854      0.000      -
0.641      -0.526
Brand_Others                -0.4403      0.078      -5.636      0.000      -
0.593      -0.287
Brand_Porsche                0.3139      0.061       5.127      0.000
0.194       0.434
Brand_Renault               -0.5976      0.027     -22.454      0.000      -
0.650      -0.545
Brand_Skoda                 -0.5401      0.023     -23.304      0.000      -
0.586      -0.495
Brand_Tata                  -0.9025      0.026     -35.365      0.000      -
0.953      -0.852
Brand_Toyota                -0.2945      0.020     -14.474      0.000      -
0.334      -0.255
Brand_Volkswagen            -0.6043      0.022     -27.908      0.000      -
0.647      -0.562
Brand_Volvo                 -0.1545      0.049      -3.157      0.002      -
0.250      -0.059
Location_Bangalore           0.1054      0.019       5.514      0.000
0.068       0.143
Location_Chennai             0.0103      0.018       0.561      0.575      -
0.026       0.046
Location_Coimbatore          0.0842      0.018       4.752      0.000
0.049       0.119
Location_Delhi              -0.0767      0.018      -4.298      0.000      -
0.112      -0.042
Location_Hyderabad           0.0834      0.017       4.835      0.000
0.050       0.117
Location_Jaipur             -0.0349      0.019      -1.851      0.064      -
0.072       0.002
Location_Kochi              -0.0386      0.018      -2.187      0.029      -
0.073      -0.004
Location_Kolkata            -0.2017      0.018     -11.155      0.000      -
0.237      -0.166
Location_Mumbai             -0.0497      0.017      -2.876      0.004      -
0.084      -0.016
Location_Pune               -0.0305      0.018      -1.732      0.083      -
0.065       0.004
Fuel_Type_Diesel             0.1675      0.031       5.382      0.000
0.106       0.229
Fuel_Type_Electric           1.1366      0.190       5.972      0.000
0.763       1.510
Fuel_Type_LPG               -0.0351      0.073      -0.481      0.631      -
0.178       0.108
Fuel_Type_Petrol            -0.0825      0.031      -2.637      0.008      -
0.144      -0.021
```

```
Transmission_Manual          -0.0926      0.010     -9.664      0.000      -
0.111      -0.074
Owner_Type_Fourth & Above    -0.0859      0.084     -1.025      0.306      -
0.250       0.078
Owner_Type_Second            -0.0441      0.008     -5.193      0.000      -
0.061      -0.027
Owner_Type_Third             -0.1019      0.022     -4.660      0.000      -
0.145      -0.059
Seats_bin_5 to 7             -0.1612      0.024     -6.817      0.000      -
0.208      -0.115
Seats_bin_Over 7             -0.1271      0.031     -4.040      0.000      -
0.189      -0.065
==============================================================================
=
Omnibus:                      270.115   Durbin-Watson:                   1.97
1
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1256.65
3
Skew:                          -0.026   Prob(JB):                    1.32e-27
3
Kurtosis:                       5.675   Cond. No.                    1.93e+0
6
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.93e+06. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

# Conclusions

***olsres1* is our final model which follows all the assumptions, and can be used for interpretations.**

1. Power come out to be very significant, as expected. As Power increase, the Price also increase, as is visible in the positive coefficient sign.
2. Kilometers Driven come out to weak significant, it was a surprise. As Kilometers increase, the Price decrease, as is visible in the negative coefficient sign.
3. 1 unit increase in Year (year Manufacturing) leads to a decrease in Price by 0.0938 Lakhs.
4. Diesel fuel type tend to have higher prices compared to Petrol.

## Business Recommendations

- Model improvement can be done with more Data points, more informations abou the characteristics of the car, more data points to compare patterns and make better predictions.
- Not enough training data. This can be solved by training with more data (Eventhough this may not always succeed. Sometimes it may give noise towards data).
- Maximize the profit but also be aware to be sold for a reasonable price for someone who would want to own it.
- First owners cars, manually transmission and Diesel are most popular cars available on market.