Cars4U

Problem definition

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.

As a senior data scientist at Cars4U, 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. For example, if the business knows the market price, it will never sell anything below it.

Objective

Explore and visualize the dataset, build a linear regression model to predict the prices of used cars, and generate a set of insights and recommendations that will help the business.

Data Description

The data contains the different attributes of used cars sold in different locations in India. The detailed data dictionary is given below.

Data Dictionary

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

Import necessary libraries

In [864…

```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

# 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 split the data into train and test
from sklearn.model_selection import train_test_split

# 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 [865…

```
# loading the dataset
data = pd.read_csv("used_cars_data.csv")
```

In [866…

```
# checking the shape of the data
print(f"There are {data.shape[0]} rows and {data.shape[1]} columns.")  # f-strin
```

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

In [867…

```
# Sample of the data, we can also use Head or Tail function to see the data samp
data.sample(
    10, random_state=2
)
```

Out[867…

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_T
4584	4584	Tata Tigor 1.05 Revotorq XT	Kochi	2018	28973	Diesel	Manual	
6505	6505	Volkswagen Vento Diesel Highline	Chennai	2011	76041	Diesel	Manual	
3675	3675	Maruti Swift VDI	Ahmedabad	2012	65000	Diesel	Manual	
5654	5654	Hyundai i20 Magna Optional 1.2	Kochi	2014	42315	Petrol	Manual	

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_T
4297	4297	Toyota Camry 2.5 G	Mumbai	2014	68400	Petrol	Automatic	
2603	2603	Mercedes-Benz New C-Class 220 CDI AT	Jaipur	2010	74213	Diesel	Automatic	
4337	4337	Volkswagen Vento Petrol Highline AT	Kochi	2014	32283	Petrol	Automatic	Sec
6625	6625	Maruti Swift VDI BSIV	Kolkata	2012	72000	Diesel	Manual	
2846	2846	Skoda Superb Elegance 1.8 TSI AT	Kochi	2011	73783	Petrol	Automatic	Sec
1237	1237	Audi Q3 2.0 TDI Quattro	Hyderabad	2013	60000	Diesel	Automatic	

In [868…

```
# creating a copy of the data so that original data remains unchanged
df = data.copy()
```

In [869…

```
# checking for duplicate values in the data
df.duplicated().sum()
```

Out[869… 0

Observation: There are no duplicate values in the data.

In [870…

```
# checking the names of the columns in the data
print(df.columns)
```

```
Index(['S.No.', 'Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats',
       'New_Price', 'Price'],
      dtype='object')
```

In [871…

```
# checking column datatypes and number of non-null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   S.No.              7253 non-null   int64
 1   Name               7253 non-null   object
 2   Location           7253 non-null   object
 3   Year               7253 non-null   int64
 4   Kilometers_Driven  7253 non-null   int64
```

```
 5   Fuel_Type          7253 non-null   object
 6   Transmission       7253 non-null   object
 7   Owner_Type         7253 non-null   object
 8   Mileage            7251 non-null   object
 9   Engine             7207 non-null   object
 10  Power              7078 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:

- There are many numeric (float and int type) and string (object type) columns in the data.
- Dependent variable is the Price of a car, which is float type.
- New_Price has only 1006 values.

In [872…

```
# checking for missing values in the data.
df.isnull().sum()
```

Out[872…

```
S.No.                   0
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 2
Engine                 46
Power                 175
Seats                  53
New_Price            6247
Price                1234
dtype: int64
```

Observation: There are missing values in many columns. New_Price contains 6247 null values.

In [873…

```
# Let's look at the statistical summary of the data
df.describe(include="all").T
```

Out[873…

	count	unique	top	freq	mean	std	min	25%
S.No.	7253.0	NaN	NaN	NaN	3626.0	2093.905084	0.0	1813.0
Name	7253	2041	Mahindra XUV500 W8 2WD	55	NaN	NaN	NaN	NaN
Location	7253	11	Mumbai	949	NaN	NaN	NaN	NaN
Year	7253.0	NaN	NaN	NaN	2013.365366	3.254421	1996.0	2011.0
Kilometers_Driven	7253.0	NaN	NaN	NaN	58699.063146	84427.720583	171.0	34000.0
Fuel_Type	7253	5	Diesel	3852	NaN	NaN	NaN	NaN
Transmission	7253	2	Manual	5204	NaN	NaN	NaN	NaN
Owner_Type	7253	4	First	5952	NaN	NaN	NaN	NaN

	count	unique	top	freq	mean	std	min	25%
Mileage	7251	450	17.0 kmpl	207	NaN	NaN	NaN	NaN
Engine	7207	150	1197 CC	732	NaN	NaN	NaN	NaN
Power	7078	385	74 bhp	280	NaN	NaN	NaN	NaN
Seats	7200.0	NaN	NaN	NaN	5.279722	0.81166	0.0	5.0
New_Price	1006	625	33.36 Lakh	6	NaN	NaN	NaN	NaN
Price	6019.0	NaN	NaN	NaN	9.479468	11.187917	0.44	3.5

Observation:

- Median value of the sold cars year model is 2014 and Mean is 2013 year
- Median value of the sold cars kilometers is 53416 and Mean is 58699 Kilometers.
- Median of the car seats is 5 and it close mean value as well.
- The Price of the cars in the data has a very wide range (0.44 to 160.0).
- Median Price of the car is 5.64 Lakhs and Mean Price is 9.47 Lakhs.

In [874…

```
# filtering non-numeric columns
car_columns = data.select_dtypes(exclude=np.number).columns
car_columns
```

Out[874…

```
Index(['Name', 'Location', 'Fuel_Type', 'Transmission', 'Owner_Type',
       'Mileage', 'Engine', 'Power', 'New_Price'],
      dtype='object')
```

In [875…

```
# printing the number of occurrences of each unique value in each categorical co

cat_col = ["Location", "Year", "Fuel_Type", "Transmission", "Owner_Type", "Seats

for column in cat_col:
    print(data[column].value_counts())
    print("-" * 50)
```

```
Mumbai        949
Hyderabad     876
Coimbatore    772
Kochi         772
Pune          765
Delhi         660
Kolkata       654
Chennai       591
Jaipur        499
Bangalore     440
Ahmedabad     275
Name: Location, 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
--------------------------------------------------
Diesel      3852
Petrol      3325
CNG           62
LPG           12
Electric       2
Name: Fuel_Type, dtype: int64
--------------------------------------------------
Manual       5204
Automatic    2049
Name: Transmission, dtype: int64
--------------------------------------------------
First             5952
Second            1152
Third              137
Fourth & Above      12
Name: Owner_Type, dtype: int64
--------------------------------------------------
5.0     6047
7.0      796
8.0      170
4.0      119
6.0       38
2.0       18
10.0       8
9.0        3
0.0        1
Name: Seats, dtype: int64
--------------------------------------------------
```

Observation:

- Highest numbers of cars being sold or available for purchase in Mumbai
- Highest numbers of cars being sold are 2015 and 2014 year manufactured cars.
- Highest number of cars being sold are Diesel fuel type.
- 5204 Manual transmission cars being sold
- Most of the sold cars owner type is First
- Most of the cars being sold are 5 seaters

Data Preprocessing

In [876…

```
# dropping S.No Column, since we have pandas default s.no column
df.drop(['S.No.'], axis=1, inplace=True)
```

In [877…

```
# there are 2 different units in the Mileage column so checking the count of occ

kmkg = 0
kmpl = 0
for i in df.Mileage:
    if str(i).endswith("km/kg"):
        kmkg+=1
    elif str(i).endswith("kmpl"):
        kmpl+=1

print('The number of rows with Km/Kg : {} '.format(kmkg))
print('The number of rows with Kmpl : {} '.format(kmpl))
```

```
The number of rows with Km/Kg : 74
The number of rows with Kmpl : 7177
```

In [878…

```
# removing km/kg and kmpl units from the Mileage column

df["Mileage"] = df["Mileage"].str.rstrip(" kmpl")
df["Mileage"] = df["Mileage"].str.rstrip(" km/g")
```

In [879…

```
# Strip CC unit from the Engine column
df["Engine"] = df["Engine"].str.rstrip(" CC")
```

In [880…

```
# Strip bhp unit from the Power column nad replace null values with nan
df["Power"] = df["Power"].str.rstrip(" bhp")
df["Power"]= df["Power"].replace(regex="null", value = np.nan)
```

Feature Engineering - Creating new column using Years column

In [881…

```
# Age of the car based on the manufactured year

Cur_Year = 2021
df['Car_Age']=Cur_Year-df['Year']
df.head()
```

Out[881…

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.(
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.6
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.:
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.7
4	Audi A4 New 2.0	Coimbatore	2013	40670	Diesel	Automatic	Second	15.:

Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
TDI Multitronic							

In [882…

```
# checking 0.0 values since 0.0 not possible for used cars so it should be nan
df.query("Mileage == '0.0'")['Mileage'].count()
```

Out[882… 81

Observation: There are totally 81, 0.0 values occured in the Mileage column which is not valid values

In [883…

```
# updating 0.0 values with nan value
df.loc[df["Mileage"]=='0.0','Mileage']=np.nan
```

In [884…

```
# checking 0.0 values in the Power column
df.loc[df["Power"]=='0.0','Power'].count()
```

Out[884… 0

In [885…

```
# checking 0.0 values in the Seats column - using query function since Seats col
df.query("Seats == 0.0")['Seats']
```

Out[885…

```
3999    0.0
Name: Seats, dtype: float64
```

In [886…

```
# updating 0.0 values with nan value
df.loc[3999,'Seats'] =np.nan
```

In [887…

```
# converting cr to lakhs in the New_Price column
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
            # 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 missin
        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
```

In [888…

```
# updating null values with nan fileds in the New_Price column

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 missin
        new_price_num.append(np.nan)

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

In [889…

```
df.head()
```

Out[889…

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.(
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.6
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.:
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.7
4	Audi A4 New 2.0	Coimbatore	2013	40670	Diesel	Automatic	Second	15.:

Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
TDI Multitronic							

Feature Engineering

In [890…

```
# converting datatypes
df["Fuel_Type"] = df["Fuel_Type"].astype("category")
df["Transmission"] = df["Transmission"].astype("category")
df["Owner_Type"] = df["Owner_Type"].astype("category")
df["Mileage"] = df["Mileage"].astype(float)
df["Power"] = df["Power"].astype(float)
df["Engine"]=df["Engine"].astype(float)
df["Location"] = df["Location"].astype("category")
```

In [891…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               7253 non-null   object
 1   Location           7253 non-null   category
 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  Car_Age            7253 non-null   int64
 14  new_price_num      1006 non-null   float64
dtypes: category(4), float64(6), int64(3), object(2)
memory usage: 652.7+ KB
```

In [892…

```
# dropping null values from the Name column
df['Name'] = df.dropna(subset=['Name'])
```

In [893…

```
# creating brand and model columns using Name
Brand = df['Name'].apply(lambda x : x.split(' ')[0])
Model = df['Name'].apply(lambda x : x.split(' ')[1])

df.insert(1,"Brand",Brand)
df.insert(2,"Model",Model)
df.head()
```

Out[893…

	Name	Brand	Model	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Ow
0	Maruti Wagon R LXI CNG	Maruti	Wagon	Mumbai	2010	72000	CNG	Manual	

	Name	Brand	Model	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Ow
1	Hyundai Creta 1.6 CRDi SX Option	Hyundai	Creta	Pune	2015	41000	Diesel	Manual	
2	Honda Jazz V	Honda	Jazz	Chennai	2011	46000	Petrol	Manual	
3	Maruti Ertiga VDI	Maruti	Ertiga	Chennai	2012	87000	Diesel	Manual	
4	Audi A4 New 2.0 TDI Multitronic	Audi	A4	Coimbatore	2013	40670	Diesel	Automatic	

In [894…

```
# unique brand names from the newly created Brand column just to make sure all t
df.Brand.unique()
```

Out[894…

```
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)
```

Observation:

- Duplicate name occured due to upper and lower case difference. Example: Isuzu and ISUZU
- Due to split command some names looks random like Land(Land Rover) and Mini(Mini cooper) these needs to be corrected

In [895…

```
# correcting wrong name based on the above observations
df.loc[df.Brand == 'ISUZU','Brand']='Isuzu'
df.loc[df.Brand=='Mini','Brand']='Mini Cooper'
df.loc[df.Brand=='Land','Brand']='Land Rover'
```

In [896…

```
# checking null values from the newly created Model column
df.Model.isnull().sum()
```

Out[896… 0

In [897…

```
# checking null values from the newly created Brand column
df.Brand.isnull().sum()
```

Out[897… 0

Missing value Treatment

In [898…

```
# checking for the null values and its count

num_missing = df.isnull().sum(axis=1)
num_missing.value_counts()
```

Out[898…
```
2    5025
3    1113
0     819
1     187
4      57
5      31
6      20
7       1
dtype: int64
```

In [899…
```python
# chekcing missing values based on each row
for n in num_missing.value_counts().sort_index().index:
    if n > 0:
        print(f'For the rows with exactly {n} missing values, NAs are found in:'
        n_miss_per_col = df[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        1113
Price            1013
new_price_num    1113
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
```

Observation: This confirms that certain columns tend to be missing together or all nonmissing together. How exactly we handle this will depend on what we're doing. For visualization we may just drop the missing values, but for modeling we will likely want to either impute them or use a method that can handle missing predictor values.

In [900…

```
# Handling Missing values for Mileage, Power, Engine and Seats
# Choosing Median value to fill the the missing value instead mean value since t
df['Engine']=df.groupby(['Model','Year'])['Engine'].apply(lambda x:x.fillna(x.me
df['Power']=df.groupby(['Model','Year'])['Power'].apply(lambda x:x.fillna(x.medi
df['Mileage']=df.groupby(['Model','Year'])['Mileage'].apply(lambda x:x.fillna(x.
df['Seats']=df.groupby(['Model'])['Seats'].apply(lambda x:x.fillna(x.median()))
```

In [901…

```
col=['Engine','Power','Mileage', 'Seats']
df[col].isnull().sum()
```

Out[901…

```
Engine      7
Power      52
Mileage    21
Seats       3
dtype: int64
```

In [902…

```
# Median and Mean for Seats column is 5 so replacing 5 with null values
df['Seats']=df['Seats'].fillna(5)
```

In [903…

```
# converting newly created columns data type
df['Brand'] =df['Brand'].astype("category")
df['Model'] =df['Model'].astype("category")
```

In [904…

```
# checking column datatypes and number of non-null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   Name               7253 non-null   object  
 1   Brand              7253 non-null   category
 2   Model              7253 non-null   category
 3   Location           7253 non-null   category
 4   Year               7253 non-null   int64   
 5   Kilometers_Driven  7253 non-null   int64   
 6   Fuel_Type          7253 non-null   category
 7   Transmission       7253 non-null   category
 8   Owner_Type         7253 non-null   category
 9   Mileage            7232 non-null   float64 
 10  Engine             7246 non-null   float64 
 11  Power              7201 non-null   float64 
 12  Seats              7253 non-null   float64 
 13  New_Price          1006 non-null   object  
 14  Price              6019 non-null   float64 
 15  Car_Age            7253 non-null   int64   
 16  new_price_num      1006 non-null   float64 
dtypes: category(6), float64(6), int64(3), object(2)
memory usage: 685.0+ KB
```

In [905…

```
# Dropping New_Price Column and Null Values by grouping Brand and Model columns
df['new_price_num']=df.groupby(['Brand', 'Model'])['new_price_num'].apply(lambda
```

In [906…

```
df.new_price_num.isnull().sum()
```

Out[906… 1512

In [907…

```
df.drop(['New_Price'],axis=1,inplace=True)
```

In [908…

```
df['new_price_num']=df.groupby(['Brand'])['new_price_num'].apply(lambda x:x.fill
```

In [909…

```
df.isnull().sum()
```

Out[909…

```
Name                    0
Brand                   0
Model                   0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                21
Engine                  7
Power                  52
Seats                   0
Price                1234
```

```
Car_Age              0
new_price_num      159
dtype: int64
```

Observation: There are still 159 null values in the new_price_num column and 1234 missing values in the price column

In [910…

```
# filling further missing values with median values for Power, Mileage and Engin
pre_cols = ["Power","Mileage","Engine"]

for col in pre_cols:
    df[col] = df[col].fillna(df[col].median())
```

In [911…

```
# drop null values from the data set
df.dropna(inplace=True,axis=0)
```

In [912…

```
df.isnull().sum()
```

Out[912…

```
Name                 0
Brand                0
Model                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
Car_Age              0
new_price_num        0
dtype: int64
```

Observation: There are no missing values in the data set

In [913…

```
df.shape
```

Out[913…

```
(5892, 16)
```

Data Visualization - Univariate Data Analysis

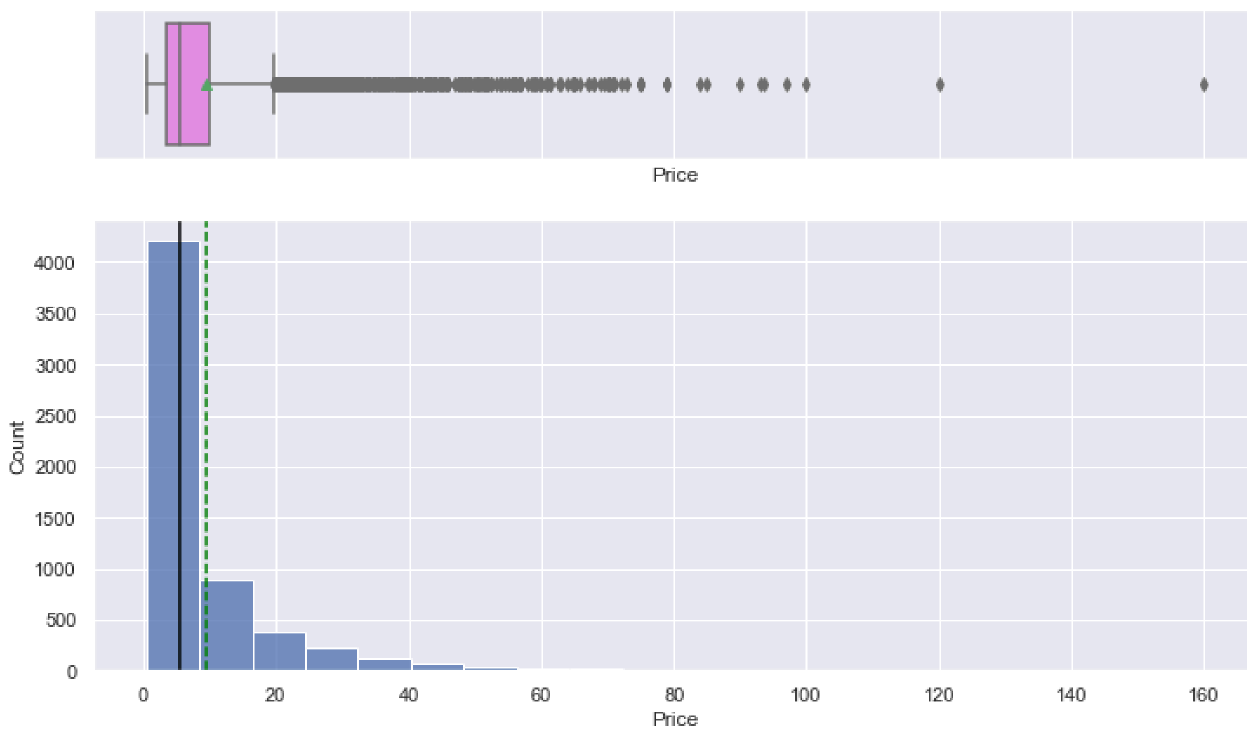
In [914…

```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(df, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
```

```
    bins: number of bins for histogram (default None)
    """
    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(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [915…

```
# creating boxplot for Price column
histogram_boxplot(df, "Price", bins = 20)
```



Observation:

- The distribution is heavily right-skewed, and most of the cars price is less than 10laks
- There is a significant difference between the mean and the median of the price distribution.
- The data points are far spread out from the mean, which indicates a high variance in the car prices.

Handling outliers

Since we have outliers in the Proce column, we have a couple of options to handle this.

- if the point seems truly nonsensical it may be best to treat it as missing
- alternatively, we could drop that observation or we could use statistics that are robust to outliers

It's often a good idea to examine the sensitivity to outliers by running an analysis with and without them.

In [916…

```
quartiles = np.quantile(df['Price'][df['Price'].notnull()], [.25, .75])
price_4iqr = 4 * (quartiles[1] - quartiles[0])
print(f'Q1 = {quartiles[0]}, Q3 = {quartiles[1]}, 4*IQR = {price_4iqr}')
outlier_price = df.loc[np.abs(df['Price'] - df['Price'].median()) > price_4iqr,
outlier_price
```

```
Q1 = 3.5, Q3 = 10.12, 4*IQR = 26.479999999999997
```

Out[916…

```
67       35.67
92       39.58
134      54.00
148      37.00
168      45.00
         ...
5919    100.00
5921     36.00
5927     45.52
5946     48.00
6008     45.00
Name: Price, Length: 302, dtype: float64
```

In [917…

```
price = df['Price'][df['Price'].notnull()]

print(price.mean())  # the mean is being pulled
print(price.median())
```

```
9.59541581805837
5.75
```

In [918…

```
from scipy.stats import tmean

print(tmean(price, limits=np.quantile(price, [.1, .9])))
print(tmean(price, limits=[0,100]))
```

```
7.126651113467657
9.569884569682568
```

In [919…

```
# dropping these rows
#df.drop(outlier_price.index, axis=0, inplace=True)
# if we wanted to make these NA we could just do this
#df.loc[np.abs(df['Price'] - df['Price'].median()) > price_4iqr, 'Price'] = np.n
```

In [920…
```
df.describe()
```

Out[920…

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	
count	5892.000000	5.892000e+03	5892.000000	5892.000000	5892.000000	5892.000000	5
mean	2013.397658	5.865530e+04	18.321224	1624.684572	113.061006	5.278344	
std	3.268687	9.212811e+04	4.170001	600.893519	53.491518	0.797586	
min	1998.000000	1.710000e+02	7.500000	72.000000	34.200000	2.000000	
25%	2012.000000	3.373675e+04	15.300000	1198.000000	75.000000	5.000000	
50%	2014.000000	5.300000e+04	18.190000	1493.000000	93.700000	5.000000	
75%	2016.000000	7.268325e+04	21.100000	1984.000000	138.100000	5.000000	
max	2019.000000	6.500000e+06	33.540000	5998.000000	552.000000	10.000000	

Observation: As max price dropped so much we cant use this dropping Outlier handling technique on the data set

In [921…
```
# Removing outlier from the price if its more than 100 lakhs
#df = df[df['Price']<100.0]
```

In [922…
```
df.describe()
```

Out[922…

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	
count	5892.000000	5.892000e+03	5892.000000	5892.000000	5892.000000	5892.000000	5
mean	2013.397658	5.865530e+04	18.321224	1624.684572	113.061006	5.278344	
std	3.268687	9.212811e+04	4.170001	600.893519	53.491518	0.797586	
min	1998.000000	1.710000e+02	7.500000	72.000000	34.200000	2.000000	
25%	2012.000000	3.373675e+04	15.300000	1198.000000	75.000000	5.000000	
50%	2014.000000	5.300000e+04	18.190000	1493.000000	93.700000	5.000000	
75%	2016.000000	7.268325e+04	21.100000	1984.000000	138.100000	5.000000	
max	2019.000000	6.500000e+06	33.540000	5998.000000	552.000000	10.000000	

Observation: Removed Oultier from the Price column

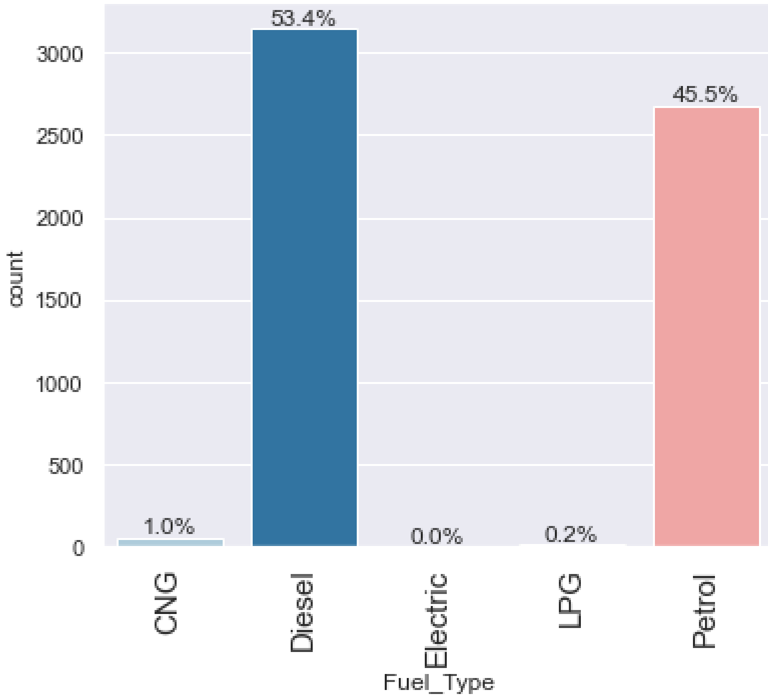
Data Visualization - Categorical Data

In [923…
```
# function to create labeled barplots


def labeled_barplot(df, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
```

```
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all le
    """

    total = len(df[feature])  # length of the column
    count = df[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=df,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

In [924…

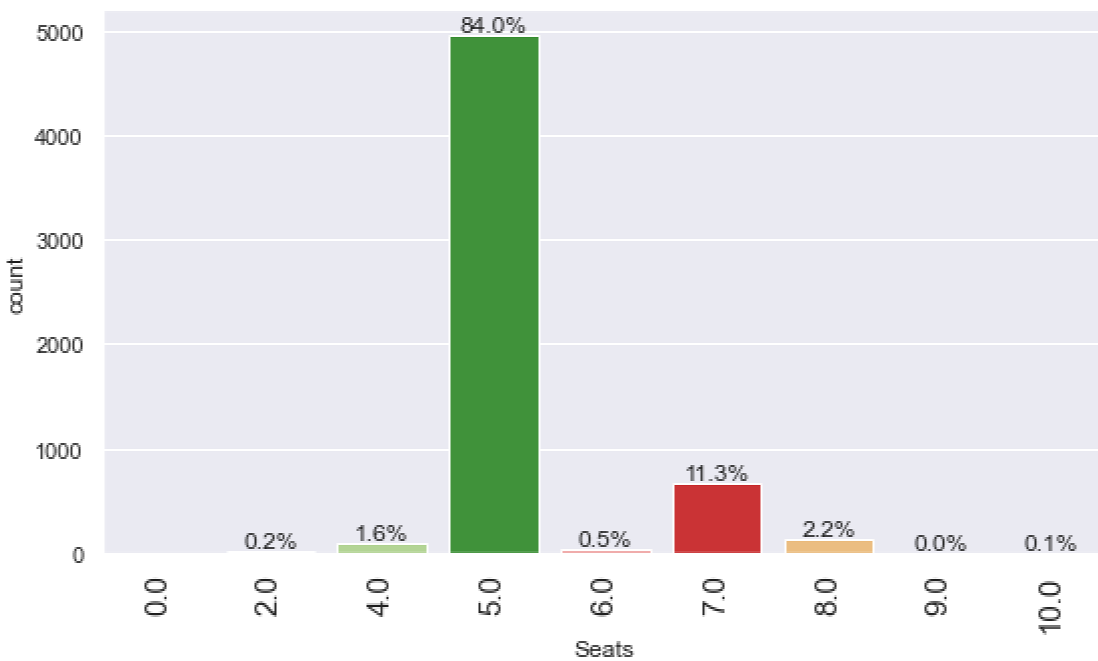
```
# creating barplot for Fuel_Type categorical column
labeled_barplot(df, "Fuel_Type", perc=True)
```



Observation: Most of the sold cars are Diesel and Petrol fuel type

In [925…

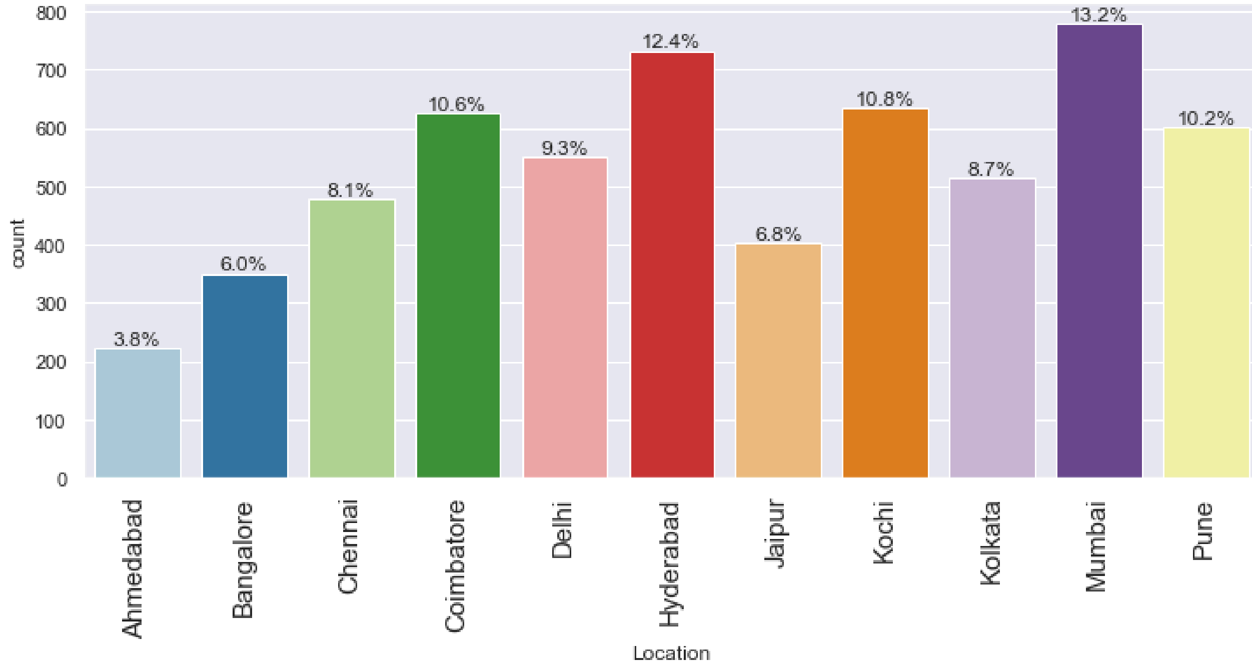
```
# creating barplot for Seats categorical column
labeled_barplot(df, "Seats", perc=True)
```



Observation: More than 80% of the sold cars are 5 seaters

In [926…

```
# creating barplot for Location categorical column
labeled_barplot(df, "Location", perc=True)
```

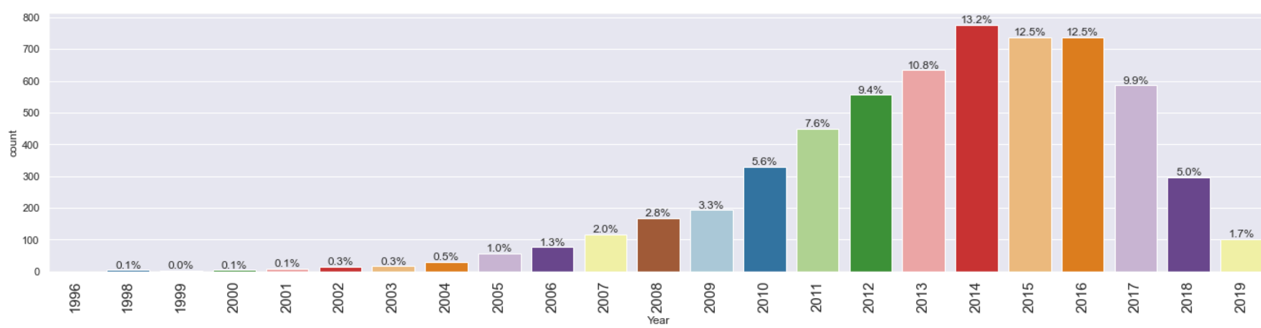


Observation:

- In Mumbai and Hyderabad location, available/sold cars for purchase is more.
- In Ahmedabad available/sold cars are very low.

In [927…

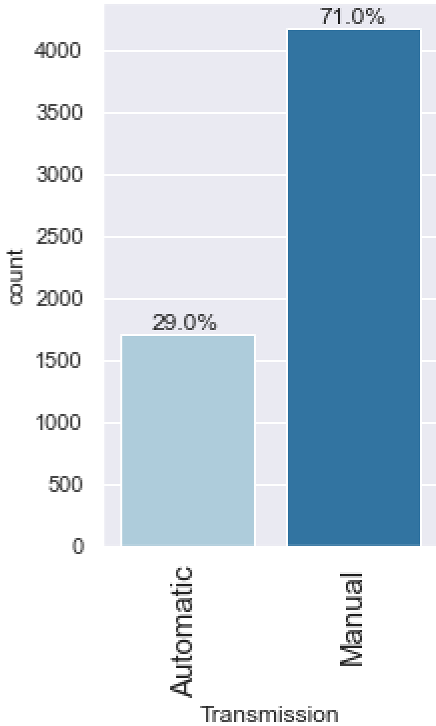
```
# creating barplot for Year categorical column
labeled_barplot(df, "Year", perc=True)
```



Observation: Most of the avilable/sold cars are 2014 and 2015 year models

In [928…

```
# creating barplot for Transmission categorical column
labeled_barplot(df, "Transmission", perc=True)
```

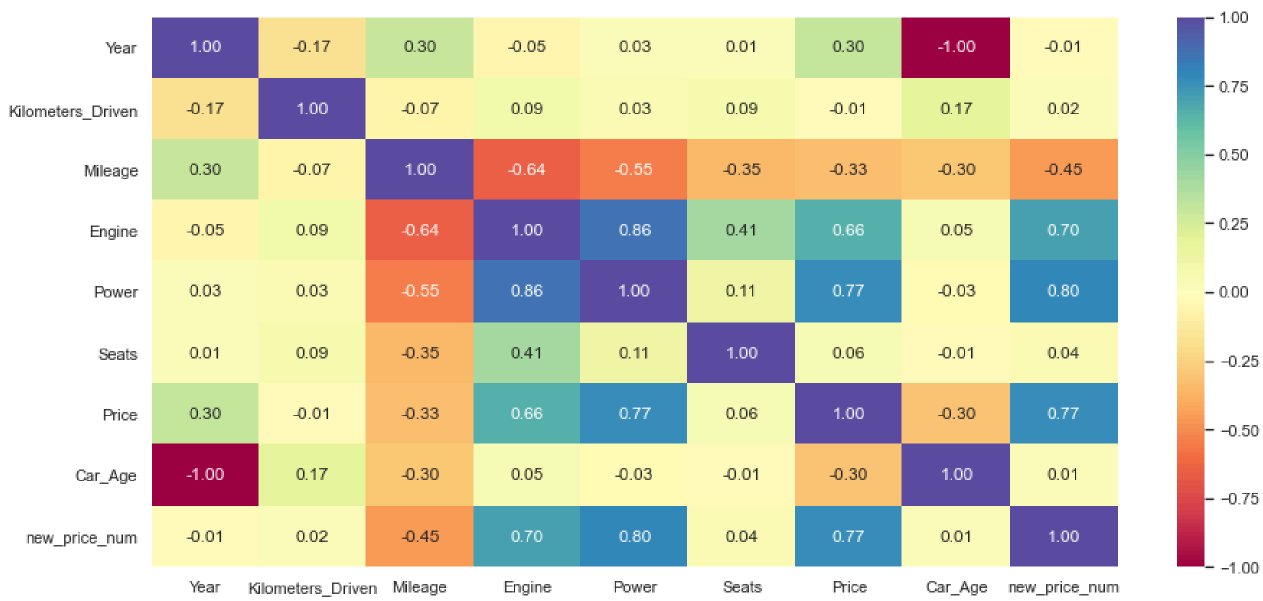


Observation: Most of the avilable/sold cars are manual cars

Multivariate Data Analysis

In [929…

```
# Heatmap

plt.figure(figsize=(15, 7))
sns.heatmap(
    df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



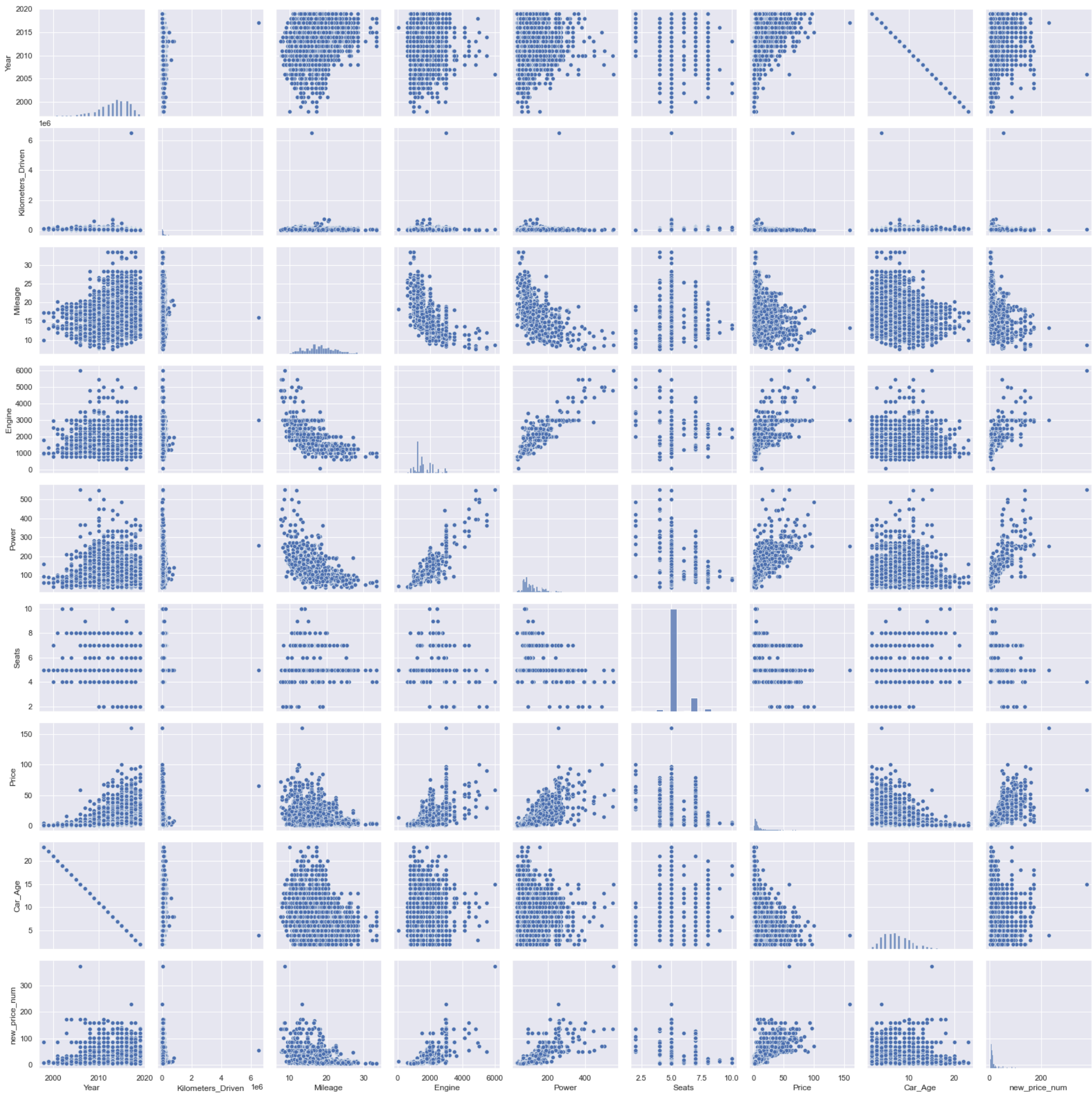
Observation:

- Engine has strong positive correlation to Power [0.86].

- Price has positive correlation to Engine[0.66] as well Power [0.77].
- Price has negative correlation to Kilometers_Driven, Mileage and Car_Age.
- Mileage is negative correlated to Kilometers_Driven,Engine,Power, Seats,Price and Car_Age

In [930…

```
# Pairplot

sns.pairplot(data=df)
plt.show()
```



Observation: Almost similar to Heatmap plot observations

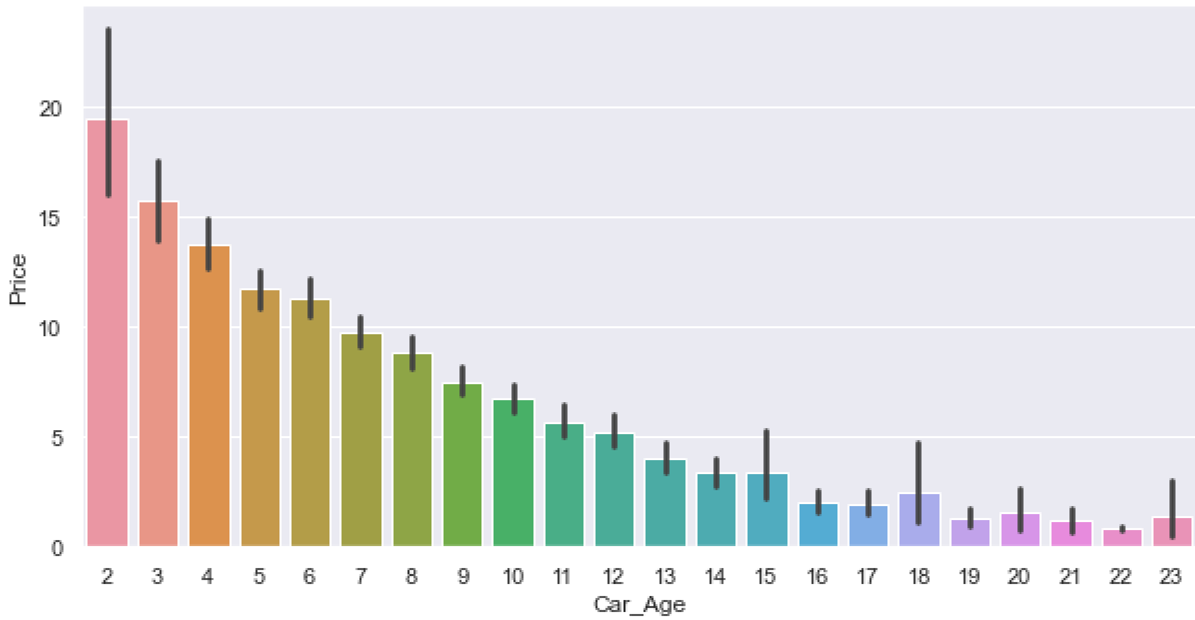
Bivariate Data Analysis

Price Vs Car_Age

In [931…

```
plt.figure(figsize=(10,5))
sns.barplot(x='Car_Age', y='Price', data=df)
```

```
plt.show()
```

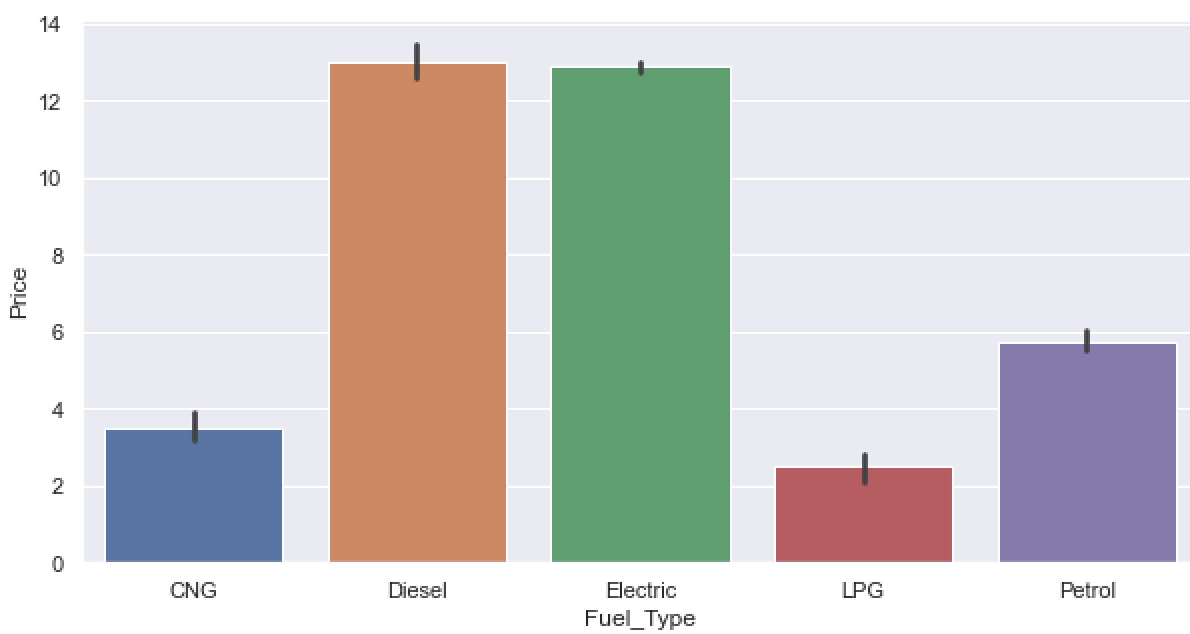


Observation: Latest model cars are expensive than old model cars.

Price Vs Fuel_Type

In [932…

```
plt.figure(figsize=(10,5))
sns.barplot(x='Fuel_Type', y='Price', data=df)
plt.show()
```



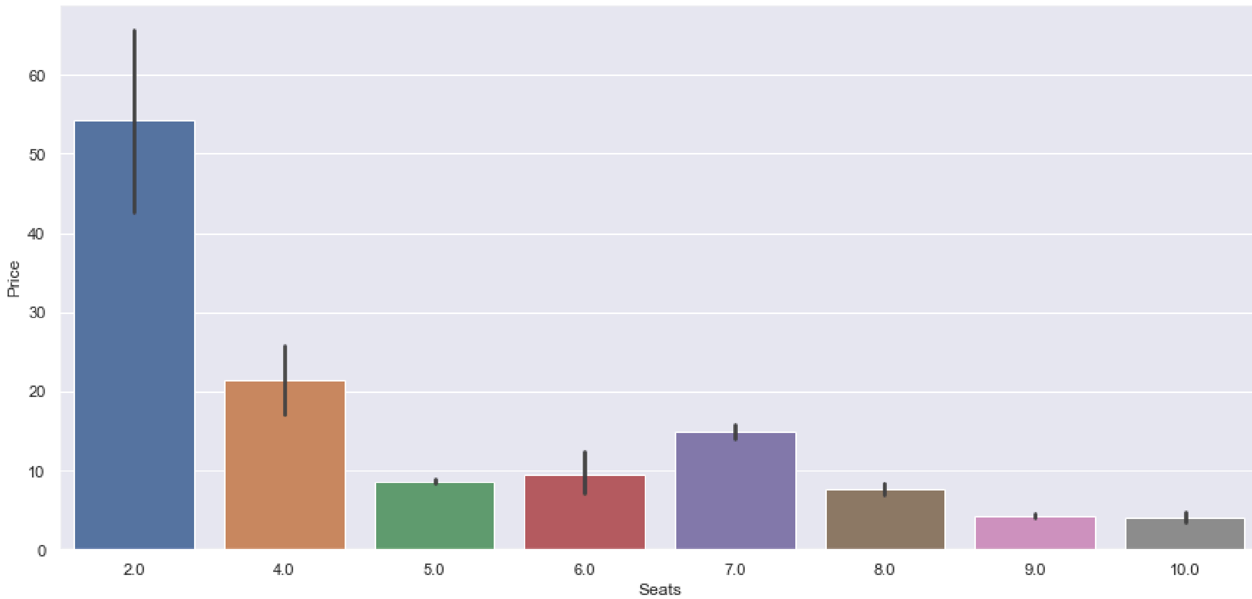
Observation: Diesel and Electric car prices are higher than other fuel types.

Price Vs Seats

In [933…

```
plt.figure(figsize=(15,7))
sns.barplot(x='Seats', y='Price', data=df)
```

```
plt.show()
```

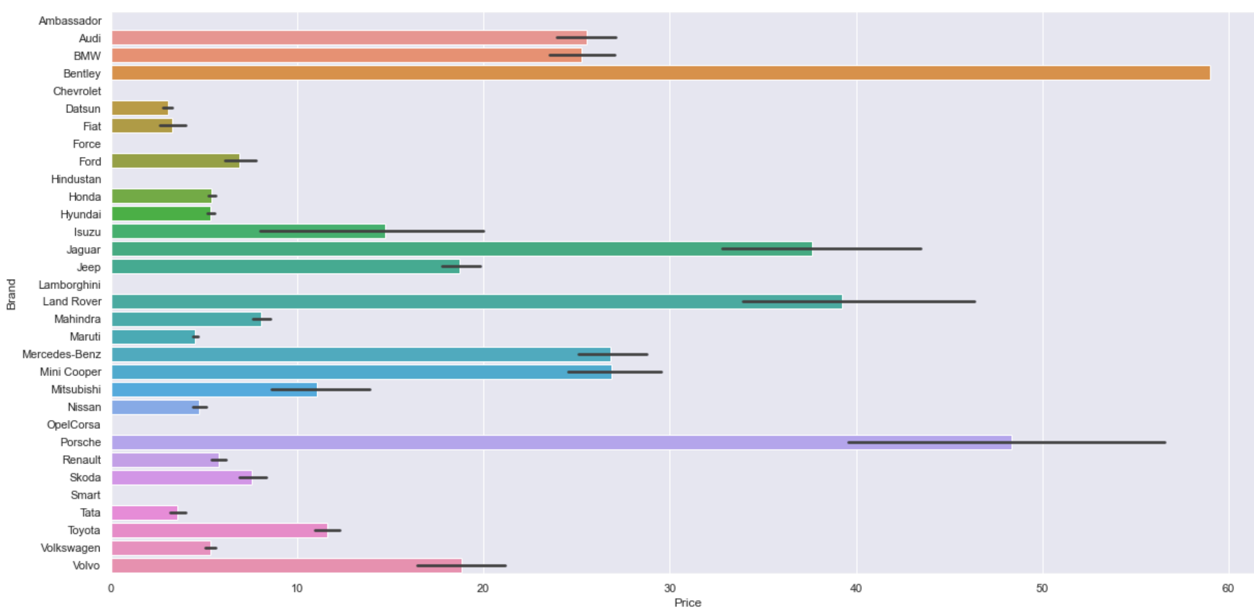


Observation: 2 Seater cars are more expensive.

Price Vs Brand

In [934…

```
plt.figure(figsize=(20,10))
sns.barplot(x='Price', y='Brand', data=df)
sns.set(font_scale=1)
plt.show()
```

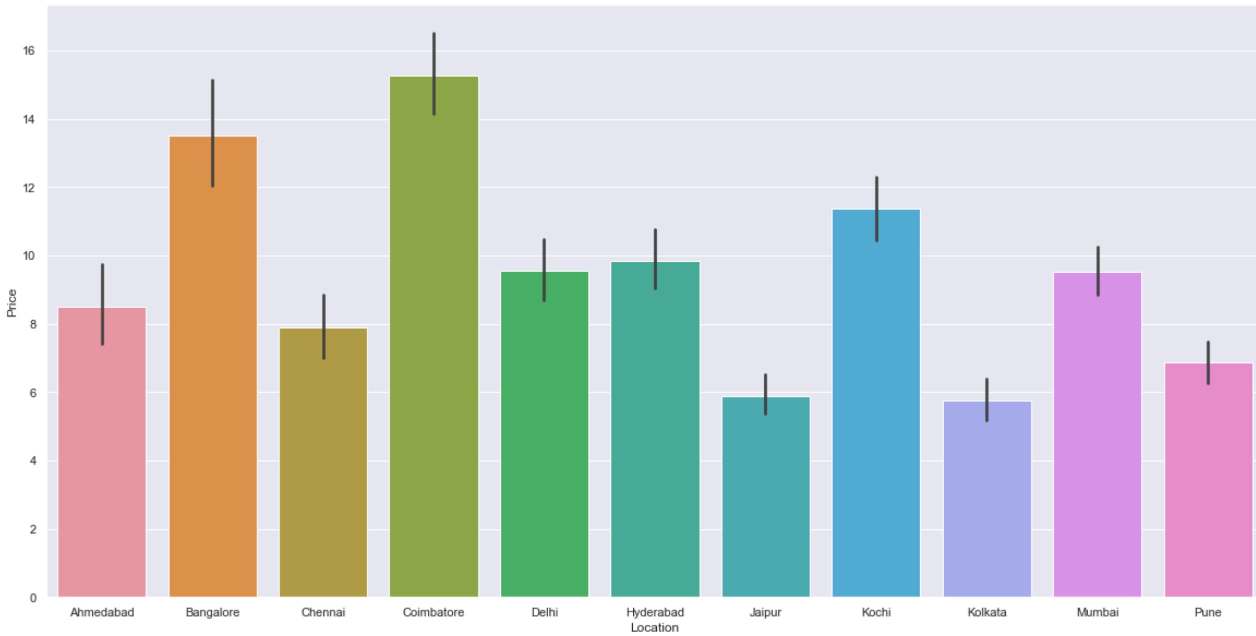


Observation: Bently cars are so expensive than other models and Ambassador cars are much cheaper than other models

Price Vs Location

In [935…

```
plt.figure(figsize=(20,10))
sns.barplot(x='Location', y='Price', data=df)
plt.show()
```

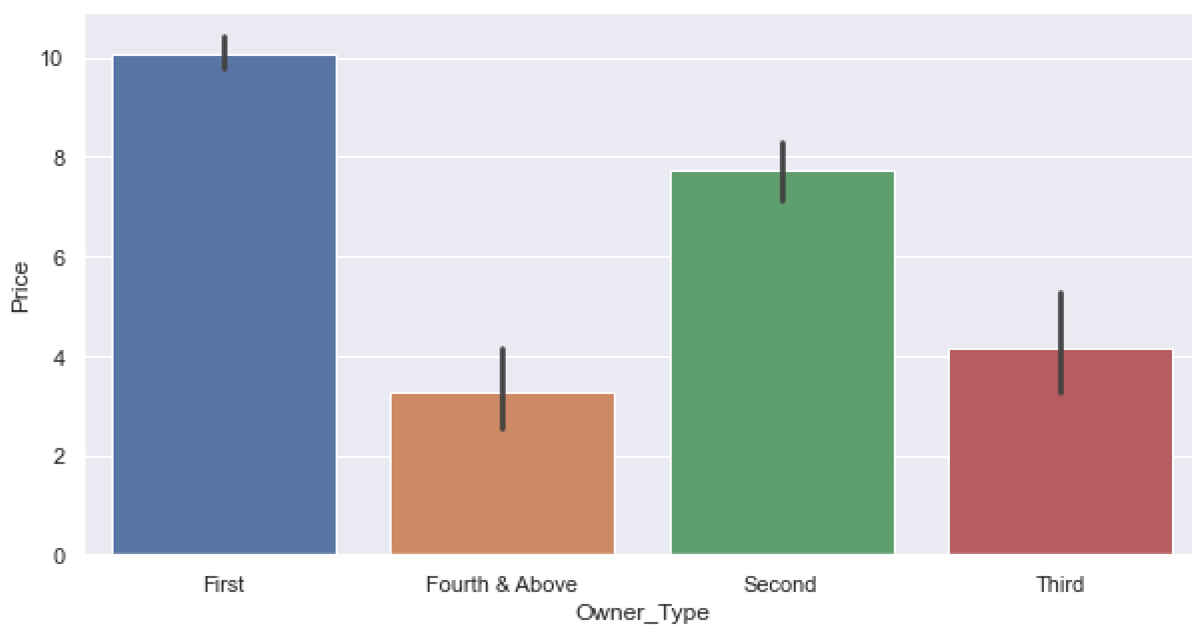


Observation: Expensive cars are in Coimbatore

Price Vs Owner_Type

In [936…

```
plt.figure(figsize=(10,5))
sns.barplot(x='Owner_Type', y='Price', data=df)
plt.show()
```

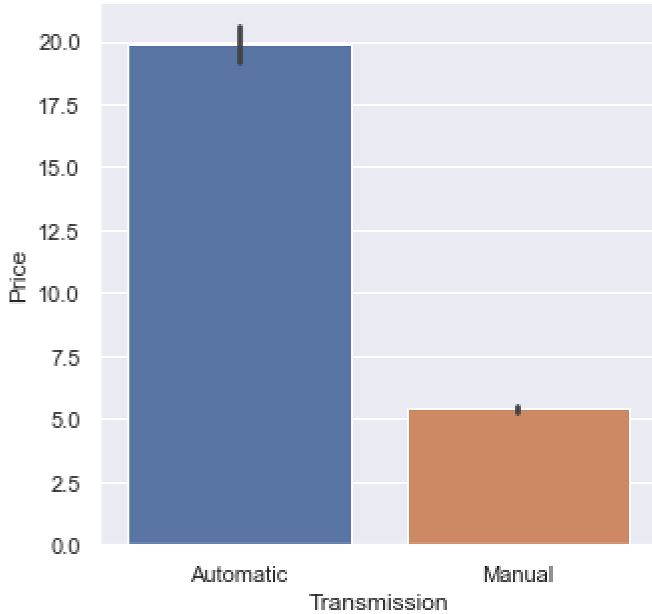


Observation: Price decreases as number of owner increases.

Price Vs Transmission

In [716…

```
plt.figure(figsize=(5,5))
sns.barplot(x='Transmission', y='Price', data=df)
plt.show()
```



Observation: Automatic transmission vehicle cars are expensive than manual transmission vehicles.

In [937…

```
# Performing log transform

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] + 1)
Perform_log_transform(df,['Kilometers_Driven','Price'])
```

In [938…

```
# droping few columns
df.drop(['Name','Model','Year','Brand','new_price_num'],axis=1,inplace=True)
```

In [939…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5892 entries, 0 to 6017
Data columns (total 13 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   float64 
 7   Power                  5892 non-null   float64 
 8   Seats                  5892 non-null   float64 
 9   Price                  5892 non-null   float64 
 10  Car_Age                5892 non-null   int64   
 11  Kilometers_Driven_log  5892 non-null   float64 
 12  Price_log              5892 non-null   float64 
dtypes: category(4), float64(7), int64(2)
memory usage: 613.3 KB
```

Model building - Linear Regression

Define independent and dependent variables

In [940…
```
X = df.drop(["Price","Price_log"], axis=1)
y = df[["Price","Price_log"]]
```

Creating dummy variables

In [941…
```
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["object", "category"]).columns.tolist(),
    drop_first=True,
)
X.head()
```

Out[941…

	Kilometers_Driven	Mileage	Engine	Power	Seats	Car_Age	Kilometers_Driven_log	Location_E
0	72000	26.60	998.0	58.16	5.0	11	11.184435	
1	41000	19.67	1582.0	126.20	5.0	6	10.621352	
2	46000	18.20	1199.0	88.70	5.0	10	10.736418	
3	87000	20.77	1248.0	88.76	7.0	9	11.373675	
4	40670	15.20	1968.0	140.80	5.0	8	10.613271	

In [942…
```
X.shape
```

Out[942…
```
(5892, 25)
```

Split the data into train and test

In [943…
```
# splitting the data in 70:30 ratio for train to test data

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [944…
```
print("Number of rows in train data =", x_train.shape[0])
print("Number of rows in test data =", x_test.shape[0])
```

```
Number of rows in train data = 4124
Number of rows in test data = 1768
```

Fitting a linear model

In [945…
```
lin_reg_model = LinearRegression()
lin_reg_model.fit(x_train, y_train["Price_log"])
```

Out[945…
```
LinearRegression()
```

Checking the coefficients and intercept of the model

In [946…

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

Out[946…

	Coefficients
Kilometers_Driven	1.011740e-07
Mileage	-2.276587e-02
Engine	6.982181e-05
Power	6.611717e-03
Seats	-1.113545e-02
Car_Age	-9.006011e-02
Kilometers_Driven_log	-9.000283e-02
Location_Bangalore	1.154584e-01
Location_Chennai	-8.611132e-03
Location_Coimbatore	7.712955e-02
Location_Delhi	-4.464876e-02
Location_Hyderabad	9.539781e-02
Location_Jaipur	-6.438473e-02
Location_Kochi	-1.715873e-02
Location_Kolkata	-2.263252e-01
Location_Mumbai	-7.747959e-02
Location_Pune	-4.017201e-02
Fuel_Type_Diesel	1.576610e-01
Fuel_Type_Electric	8.262828e-01
Fuel_Type_LPG	-5.800273e-02
Fuel_Type_Petrol	-1.773626e-01
Transmission_Manual	-2.734767e-01
Owner_Type_Fourth & Above	2.076125e-01
Owner_Type_Second	-5.606496e-02
Owner_Type_Third	-1.132480e-01
Intercept	3.513203e+00

Coefficient Interpretations

- Coefficients of Car_Age, Kilometers_Driven, Engine, Power and some of the location, Owner_Type, Fuel_Type column values are positive.
  - Increase in these will lead to an increase in the rating of an anime.
- Coefficients of Mileage, Car_Age and some of the Location, Owner_Type, Fuel_Type columns are negative.
  - Increase in these will lead to a decrease in the rating of an anime.

Model performance check

- We will be using metric functions defined in sklearn for RMSE, MAE, and $R2$.
- We will define functions to calculate adjusted $R2$ and MAPE.
  - The mean absolute percentage error (MAPE) measures the accuracy of predictions as a percentage, and can be calculated as the average absolute percent error for each predicted value minus actual values divided by actual values. It works best if there are no extreme values in the data and none of the actual values are 0.
- We will create a function that will print out all the above metrics in one go.

In [947…

```
# function to compute adjusted R-squared
def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


# function to compute MAPE
def mape_score(targets, predictions):
    return np.mean(np.abs(targets - predictions) / targets) * 100


# function to compute different metrics to check performance of a regression moc
def model_performance_regression(model, predictors, target):
    """
    Function to compute different metrics to check regression model performance

    model: regressor
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    r2 = r2_score(target, pred)  # to compute R-squared
    adjr2 = adj_r2_score(predictors, target, pred)  # to compute adjusted R-squa
    rmse = np.sqrt(mean_squared_error(target, pred))  # to compute RMSE
    mae = mean_absolute_error(target, pred)  # to compute MAE
    mape = mape_score(target, pred)  # to compute MAPE

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "RMSE": rmse,
```

```
                "MAE": mae,
                "R-squared": r2,
                "Adj. R-squared": adjr2,
                "MAPE": mape,
            },
            index=[0],
        )

    return df_perf
```

In [948…

```
# Checking model performance on train set
print("Training Performance\n")
lin_reg_model_train_perf = model_performance_regression(lin_reg_model, x_train,
lin_reg_model_train_perf
```

```
Training Performance
```

Out[948…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.259633	0.189005	0.88103	0.880304	10.427436

In [949…

```
# Checking model performance on test set
print("Test Performance\n")
lin_reg_model_test_perf = model_performance_regression(lin_reg_model, x_test, y_
lin_reg_model_test_perf
```

```
Test Performance
```

Out[949…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.246725	0.185859	0.885729	0.88409	10.363832

Observation:

- The train and test $R2$ are 0.881 and 0.885, indicating that the model explains 88.1% and 88.5% of the total variation in the train and test sets respectively. Also, both scores are comparable.
- RMSE values on the train and test sets are also comparable.
- This shows that the model is not overfitting.
- MAE indicates that our current model is able to predict car price within a mean error of ~0.18.
- MAPE of 10.36 on the test data means that we are able to predict within ~10% of the car price.

Actionable Insights & Recommendations

- Based on our Linear Regression model results, we have 10.4% of MAPE on the training data and 10.3% on the test data, which means that we are able to predict within ~10% of the car price.
- The train and test $R2$ are 0.881 and 0.885, indicating that the model explains 88.1% and 88.5% of the total variation in the train and test sets respectively.

- Automatic cars sell at higher prices so manual cars are selling in high volume so we need to focus this point and invest accordingly.
- Price decreases as number of owner increases. So investing in the multiple owner cars might be risky.
- We have to be more careful when investmenting in the Ahmedabad, Jaipur and kolkatta market.
- Coimbatore, Bangalore , Mumbai and Hyderabad markets are very good to invest.
- Mumbai and Hyderbad seems to be more popular in used car market.
- Diesel and Electrical cars are expensive but Electrical car market is still low in india so we need to focus more on Diesel cars.
- For 2014, 2015 and 2016 model cars has high demand so we need to focus this year model cars.
- 5 seater cars are in high demand so we can invest more on this type of cars.