# Data Science Major Project at INTERNSELITE (Oct Batch 2023)

(Submitted by Rajvardhan Mall
Chalamalla Nikhil, Aditya Gupta,
Neeharika PM,Devisetti VDSSS
Desish and Muhammed Bilal p)

**PROBLEM STATEMENT**

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 year and is larger than the new car market now. Consider this: In 2018-19, while new car sales were recorded at 3.6 million units, around 4 million second-hand cars were bought and sold. There is a slowdown in new car sales and that could mean that the demand is shifting towards the pre-owned market. In fact, some car sellers replace their old cars with pre-owned cars instead of buying new ones.

The goal of the case is as follows:

## i.) Perform EDA :-

(This part is performed by Rajvardhan Mall)

**Solution** :-

```
import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline
```

```
sns.set(color_codes = True)

import timeit

from warnings import filterwarnings

filterwarnings('ignore')from datetime import date
```

a.) data = pd.read_csv('Cars.csv')

# EDA (Exploratory data analysis)

b.) data.head()

**head()** will display the top 5 observations of the dataset

| | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | Engine | Power | Colour | Seats | No. of Doors | New_Price | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Mahindra Scorpio | Pune | 2012.0 | 98000.0 | Diesel | Manual | Third | 12.05 kmpl | 2179 CC | 120 bhp | Black/Silver | 8.0 | 5.0 | NaN | 6.00 |
| **1** | Maruti Baleno | Kochi | 2018.0 | 18678.0 | Petrol | Manual | First | 21.1 kmpl | 998 CC | 100 bhp | Others | 5.0 | 4.0 | NaN | 8.32 |
| **2** | Mahindra Xylo | Bangalore | 2013.0 | 197000.0 | Diesel | Manual | First | 11.68 kmpl | 2498 CC | 112 bhp | White | 7.0 | 5.0 | NaN | 4.00 |
| **3** | Hyundai Grand | Delhi | 2014.0 | 45000.0 | Diesel | Manual | First | 24.0 kmpl | 1120 CC | 70 bhp | White | 5.0 | 4.0 | NaN | 3.49 |
| **4** | Toyota Innova | Delhi | 2011.0 | 65000.0 | Diesel | Manual | First | 12.8 kmpl | 2494 CC | 102 bhp | Others | 8.0 | 5.0 | NaN | 6.40 |

c.) data.tail()

**tail()** will display the last 5 observations of the dataset

| | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | Engine | Power | Colour | Seats | No. of Doors | New_Price | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **5956** | Honda Civic | Pune | 2011.0 | 47000.0 | Petrol | Automatic | Second | 13.9 kmpl | 1799 CC | 130.3 bhp | Others | 5.0 | 4.0 | NaN | |
| **5957** | Hyundai i20 | Delhi | 2013.0 | 63777.0 | Petrol | Manual | First | 18.5 kmpl | 1197 CC | 82.9 bhp | Black/Silver | 5.0 | 4.0 | NaN | |
| **5958** | Maruti Swift | Coimbatore | 2016.0 | 37806.0 | Petrol | Manual | First | 20.4 kmpl | 1197 CC | 81.80 bhp | Black/Silver | 5.0 | 4.0 | NaN | |
| **5959** | Mercedes-Benz SLK-Class | Coimbatore | 2016.0 | 22732.0 | Petrol | Automatic | First | 18.1 kmpl | 3498 CC | 306 bhp | Black/Silver | 2.0 | 2.0 | NaN | |
| **5960** | Hyundai i10 | Kolkata | 2016.0 | 7000.0 | Petrol | Manual | First | 20.36 kmpl | 1197 CC | 78.9 bhp | White | 5.0 | 4.0 | NaN | |

d.) data.shape

(5961, 15)

e.) data.dtypes

```
Name                 object
Location             object
Kilometers_Driven   float64
Fuel_Type            object
Transmission         object
Owner_Type           object
Mileage              object
Engine               object
Power                object
Colour               object
Seats               float64
No. of Doors        float64
New_Price            object
Price               float64
dtype: object
```

## f.) print(data.info())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5961 entries, 0 to 5960
Data columns (total 15 columns):
#   Column             Non-Null Count  Dtype
---  ------             --------------  -----
0   Name               5961 non-null   object
1   Location           5950 non-null   object
2   Year               5959 non-null   float64
3   Kilometers_Driven  5953 non-null   float64
4   Fuel_Type          5961 non-null   object
5   Transmission       5934 non-null   object
6   Owner_Type         5946 non-null   object
7   Mileage            5959 non-null   object
8   Engine             5944 non-null   object
9   Power              5929 non-null   object
10  Colour             5950 non-null   object
11  Seats              5956 non-null   float64
12  No. of Doors       5960 non-null   float64
13  New_Price          824 non-null    object
14  Price              5961 non-null   float64
dtypes: float64(5), object(10)
memory usage: 698.7+ KB
None
```

## g.) data.describe()

| | Year | Kilometers_Driven | Seats | No. of Doors | Price |
|---|---|---|---|---|---|
| **count** | 5959.000000 | 5.953000e+03 | 5956.000000 | 5960.000000 | 5961.000000 |
| **mean** | 2013.389159 | 5.871110e+04 | 5.269140 | 4.114933 | 9.528103 |
| **std** | 3.243051 | 9.171221e+04 | 0.789048 | 0.344757 | 11.214382 |
| **min** | 1998.000000 | 1.710000e+02 | 2.000000 | 2.000000 | 0.440000 |
| **25%** | 2011.500000 | 3.393100e+04 | 5.000000 | 4.000000 | 3.500000 |
| **50%** | 2014.000000 | 5.300000e+04 | 5.000000 | 4.000000 | 5.660000 |
| **75%** | 2016.000000 | 7.300000e+04 | 5.000000 | 4.000000 | 10.000000 |
| **max** | 2019.000000 | 6.500000e+06 | 10.000000 | 5.000000 | 160.000000 |

h.) data.nunique()

**nunique() based on** several unique values in each column and the data description, we can identify the continuous and categorical columns in the data. Duplicated data can be handled or removed based on further analysis

```
Name                 212
Location              11
Year                  22
Kilometers_Driven   3068
Fuel_Type              5
Transmission           2
Owner_Type             4
Mileage              439
Engine               143
Power                369
Colour                 3
Seats                  8
No. of Doors           3
New_Price            540
Price               1369
dtype: int64
```

i.) data.isnull().sum()

```
Name                   0
Location              11
Year                   2
Kilometers_Driven      8
Fuel_Type              0
Transmission          27
Owner_Type            15
Mileage                2
Engine                17
Power                 32
Colour                11
Seats                  5
No. of Doors           1
New_Price           5137
Price                  0
dtype: int64
```

j.) (data.isnull().sum()/(len(data)))*100

```
Name                   0.000000
Location               0.184533
Year                   0.033551
Kilometers_Driven      0.134206
Fuel_Type              0.000000
Transmission           0.452944
Owner_Type             0.251636
Mileage                0.033551
Engine                 0.285187
Power                  0.536823
Colour                 0.184533
Seats                  0.083879
No. of Doors           0.016776
New_Price             86.176816
Price                  0.000000
dtype: float64
```

The percentage of missing values for the columns **New_Price** and **Price** is ~86% and 0%, respectively.

k.) data = data.drop(['Year'], axis = 1)

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5961 entries, 0 to 5960
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               5961 non-null   object
 1   Location           5950 non-null   object
 2   Kilometers_Driven  5953 non-null   float64
 3   Fuel_Type          5961 non-null   object
 4   Transmission       5934 non-null   object
 5   Owner_Type         5946 non-null   object
 6   Mileage            5959 non-null   object
 7   Engine             5944 non-null   object
 8   Power              5929 non-null   object
 9   Colour             5950 non-null   object
 10  Seats              5956 non-null   float64
 11  No. of Doors       5960 non-null   float64
 12  New_Price          824 non-null    object
 13  Price              5961 non-null   float64
dtypes: float64(4), object(10)
memory usage: 652.1+ KB
```

l.) data.describe(include='all').T

| | count | unique | top | freq | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | 5961 | 212 | Maruti Swift | 343 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Location** | 5950 | 11 | Mumbai | 781 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Kilometers_Driven** | 5953.0 | NaN | NaN | NaN | 58711.100118 | 91712.207172 | 171.0 | 33931.0 | 53000.0 | 73000.0 | 6500000.0 |
| **Fuel_Type** | 5961 | 5 | Diesel | 3188 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Transmission** | 5934 | 2 | Manual | 4225 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Owner_Type** | 5946 | 4 | First | 4875 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Mileage** | 5959 | 439 | 18.9 kmpl | 172 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Engine** | 5944 | 143 | 1197 CC | 606 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Power** | 5929 | 369 | 74 bhp | 233 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Colour** | 5950 | 3 | White | 2115 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Seats** | 5956.0 | NaN | NaN | NaN | 5.26914 | 0.789048 | 2.0 | 5.0 | 5.0 | 5.0 | 10.0 |
| **No. of Doors** | 5960.0 | NaN | NaN | NaN | 4.114933 | 0.344757 | 2.0 | 4.0 | 4.0 | 4.0 | 5.0 |
| **New_Price** | 824 | 540 | 4.78 Lakh | 6 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Price** | 5961.0 | NaN | NaN | NaN | 9.528103 | 11.214382 | 0.44 | 3.5 | 5.66 | 10.0 | 160.0 |

From the statistics summary, we can infer the below findings :

- Maruti Swift has highest number of counts.
- On average of Kilometers-driven in Used cars are ~58k KM. The range shows a huge difference between min and max as max values show 650000 KM shows the evidence of an outlier. This record can be removed.
- Min value of Mileage shows 0 cars won't be sold with 0 mileage. This sounds like a data entry issue.
- It looks like Engine and Power have outliers, and the data is right-skewed.
- The average number of seats in a car is approx. 5. car seat is an important feature in price contribution.
- The max price of a used car is 160k which is quite weird, such a high price for used cars. There may be an outlier or data entry issue.

m.) data.describe().T

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Kilometers_Driven** | 5953.0 | 58711.100118 | 91712.207172 | 171.00 | 33931.0 | 53000.00 | 73000.0 | 6500000.0 |
| **Seats** | 5956.0 | 5.269140 | 0.789048 | 2.00 | 5.0 | 5.00 | 5.0 | 10.0 |
| **No. of Doors** | 5960.0 | 4.114933 | 0.344757 | 2.00 | 4.0 | 4.00 | 4.0 | 5.0 |
| **Price** | 5961.0 | 9.528103 | 11.214382 | 0.44 | 3.5 | 5.66 | 10.0 | 160.0 |

n.) print(data.Name.unique())

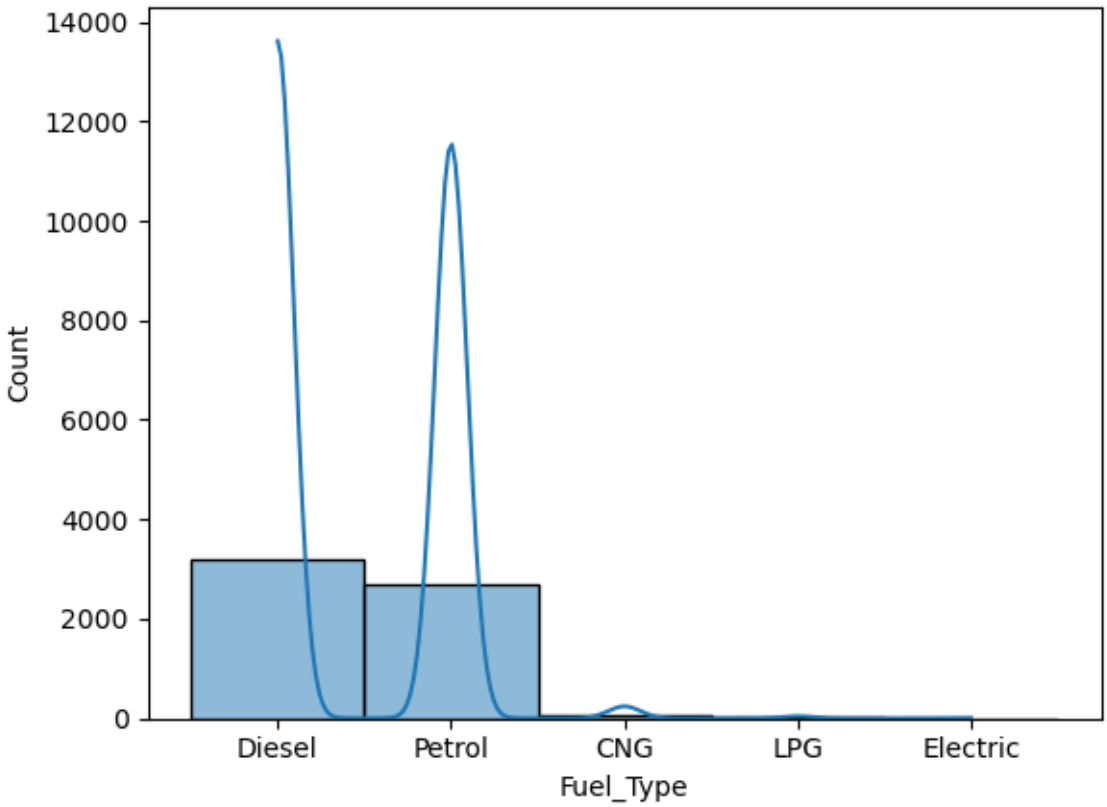
```
['Mahindra Scorpio' 'Maruti Baleno' 'Mahindra Xylo' 'Hyundai Grand'
'Toyota Innova' 'Honda WRV' 'Hyundai Verna' 'Toyota Fortuner'
'Chevrolet Beat' 'Audi A6' 'Hyundai i20' 'Skoda Laura' 'Honda City'
'Audi Q7' 'Renault Duster' 'Maruti Ertiga' 'Land Rover Freelander'
```

```
'Skoda Rapid' 'Maruti Zen' 'BMW 3' 'Maruti Wagon' 'Honda Jazz'
'Maruti Vitara' 'ISUZU D-MAX' 'Maruti SX4' 'Hyundai i10' 'Honda Amaze'
'Hyundai Xcent' 'Jaguar F' 'Mercedes-Benz C-Class' 'Volkswagen Polo'
'BMW 5' 'Mahindra Thar' 'Maruti Alto' 'Audi A3' 'Tata Xenon' 'BMW X1'
'Hyundai Getz' 'Volkswagen Vento' 'Chevrolet Sail' 'Volkswagen Jetta'
'Maruti Ciaz' 'Honda CR-V' 'Mahindra XUV500' 'Mercedes-Benz New'
'Honda Civic' 'Skoda Superb' 'Maruti Ritz' 'Mitsubishi Pajero'
'Toyota Corolla' 'Hyundai Elantra' 'Ford Figo' 'Hyundai EON'
'Nissan Teana' 'Maruti Swift' 'Mercedes-Benz CLA' 'Land Rover Range'
'Audi Q3' 'Ford Fiesta' 'Tata Indica' 'Ford Ecosport' 'Volvo S80'
'Skoda Fabia' 'Audi A4' 'Mercedes-Benz E-Class' 'Honda Brio'
'Land Rover Discovery' 'Ford Ikon' 'Hyundai Santa' 'Mahindra Bolero'
'Maruti Dzire' 'Renault KWID' 'Maruti Celerio' 'Chevrolet Optra'
'Mercedes-Benz GLC' 'BMW X3' 'Nissan Micra' 'Toyota Etios'
'Hyundai Creta' 'Ford Endeavour' 'BMW 7' 'BMW X5' 'Skoda Octavia'
'Tata Zest' 'Renault Scala' 'Mercedes-Benz B' 'Ford EcoSport'
'Mahindra TUV' 'Maruti Grand' 'Tata Hexa' 'Hyundai Elite'
'Hyundai Accent' 'Mercedes-Benz GLE' 'Fiat Linea' 'Maruti Omni' 'Audi Q5'
'Volkswagen Passat' 'Porsche Panamera' 'Honda BRV' 'Mini Cooper'
'Tata Indigo' 'Maruti A-Star' 'Tata Nano' 'Hyundai Santro'
'Volkswagen Tiguan' 'Mercedes-Benz R-Class' 'Honda Accord' 'Datsun GO'
'Ford Aspire' 'Toyota Camry' 'Chevrolet Cruze' 'Nissan Sunny'
'Datsun redi-GO' 'Jeep Compass' 'Audi RS5' 'Volvo S60' 'Fiat Punto'
'Volvo XC60' 'Honda Mobilio' 'Tata Manza' 'Mercedes-Benz S' 'Maruti Eeco'
'Mercedes-Benz GLA' 'Tata Bolt' 'BMW 6' 'Fiat Grande' 'Maruti Esteem'
'Mercedes-Benz M-Class' 'Audi TT' 'Skoda Yeti' 'Volkswagen Ameo'
'Mahindra Logan' 'Renault Pulse' 'Maruti Ignis' 'Mercedes-Benz GL-Class'
'Jaguar XF' 'Mitsubishi Lancer' 'Mercedes-Benz SLC' 'Porsche Cayenne'
'Mercedes-Benz A' 'Fiat Avventura' 'Tata Safari' 'Nissan Terrano'
'Maruti 800' 'Mercedes-Benz S-Class' 'Chevrolet Aveo' 'Ford Mustang'
'Maruti S' 'Volvo V40' 'Audi A7' 'Tata Sumo' 'Force One' 'Tata Tiago'
'Mahindra Verito' 'Mercedes-Benz CLS-Class' 'Toyota Platinum'
'Mercedes-Benz SLK-Class' 'Nissan X-Trail' 'BMW Z4' 'Mitsubishi Cedia'
'Jaguar XJ' 'Chevrolet Enjoy' 'Mahindra XUV300' 'Porsche Cayman'
'Chevrolet Spark' 'Mahindra KUV' 'Hyundai Sonata' 'Mahindra Quanto'
'Mahindra Renault' 'Volkswagen CrossPolo' 'Renault Captur'
'Mercedes-Benz SL-Class' 'Mahindra Jeep' 'Maruti S-Cross'
'Renault Fluence' 'Ford Fusion' 'Isuzu MUX' 'BMW X6' 'Toyota Qualis'
'Maruti 1000' 'Mitsubishi Outlander' 'Mahindra Ssangyong' 'Smart Fortwo'
'Renault Koleos' 'Tata Tigor' 'Mercedes-Benz GLS' 'Nissan Evalia'
'Tata Nexon' 'Honda BR-V' 'Jaguar XE' 'BMW 1' 'Ford Freestyle'
'Fiat Siena' 'Ford Classic' 'Honda WR-V' 'Mahindra NuvoSport' 'Tata New'
'Lamborghini Gallardo' 'Mitsubishi Montero' 'Datsun Redi'
'Mini Countryman' 'Toyota Prius' 'Fiat Petra' 'Volvo XC90'
'Hyundai Tucson' 'Audi A8' 'Volkswagen Beetle' 'Bentley Continental'
'Chevrolet Captiva' 'Mini Clubman' 'Porsche Boxster' 'Mahindra E']
```

o.) print(data.Name.nunique())

212

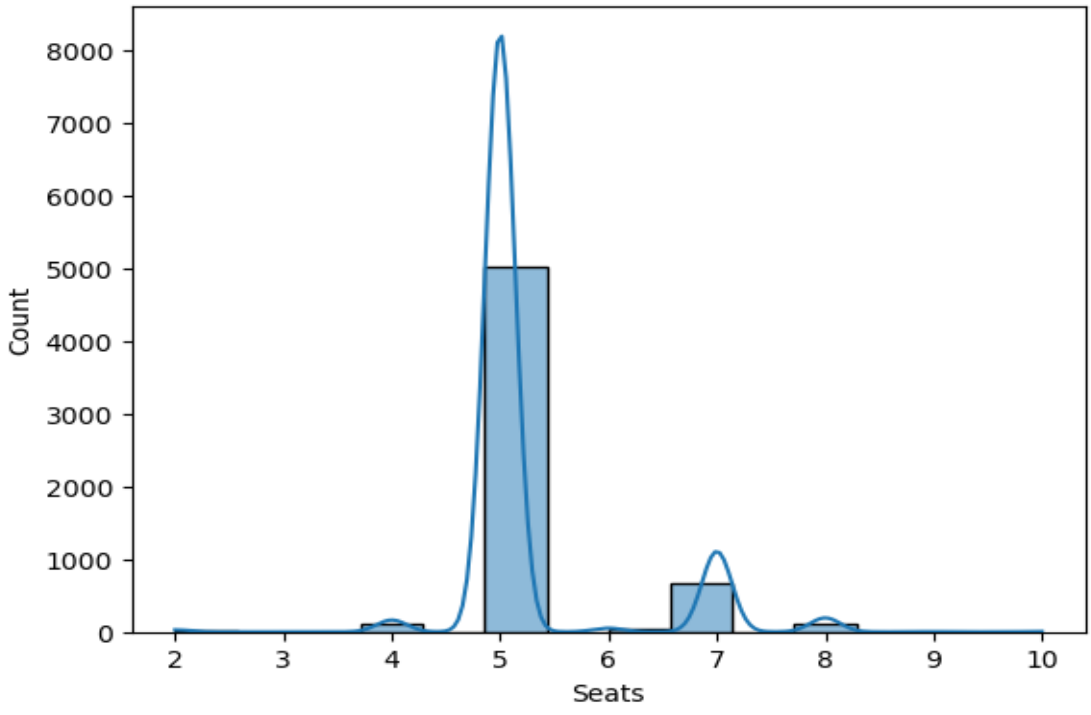
p.) sns.histplot(data.Fuel_Type, kde=True)
plt.show()




From this histogram we can see that Cars with Fuel_Type Diesel has hightest number of count.

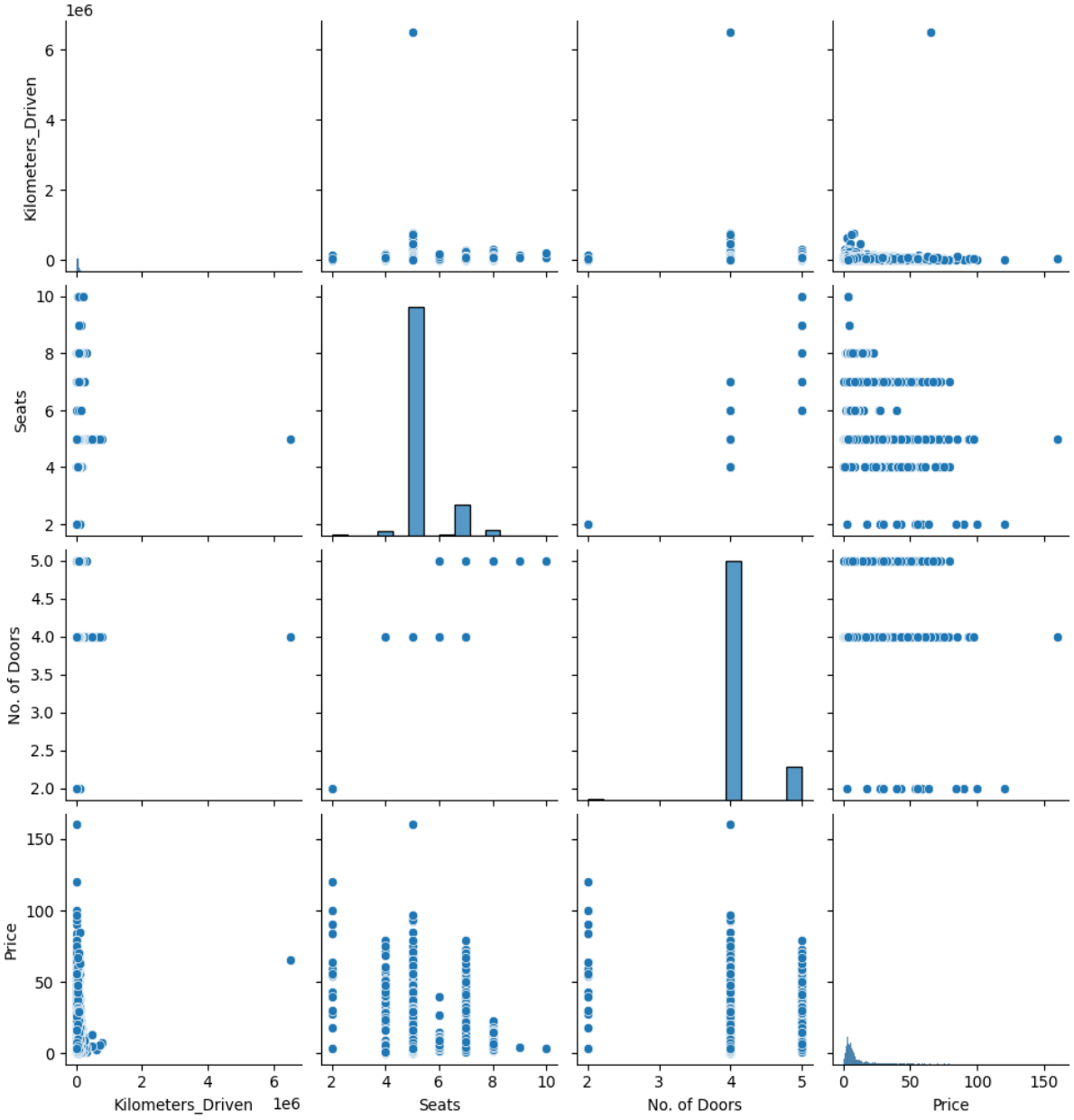
q.) sns.histplot(data.Seats, kde=True)

plt.show()



r.) sns.pairplot(data)

plt.show()




s.) fig, axes = plt.subplots(3, 2, figsize = (20, 35))

print('Bar plot for all categorical variables in the dataset')

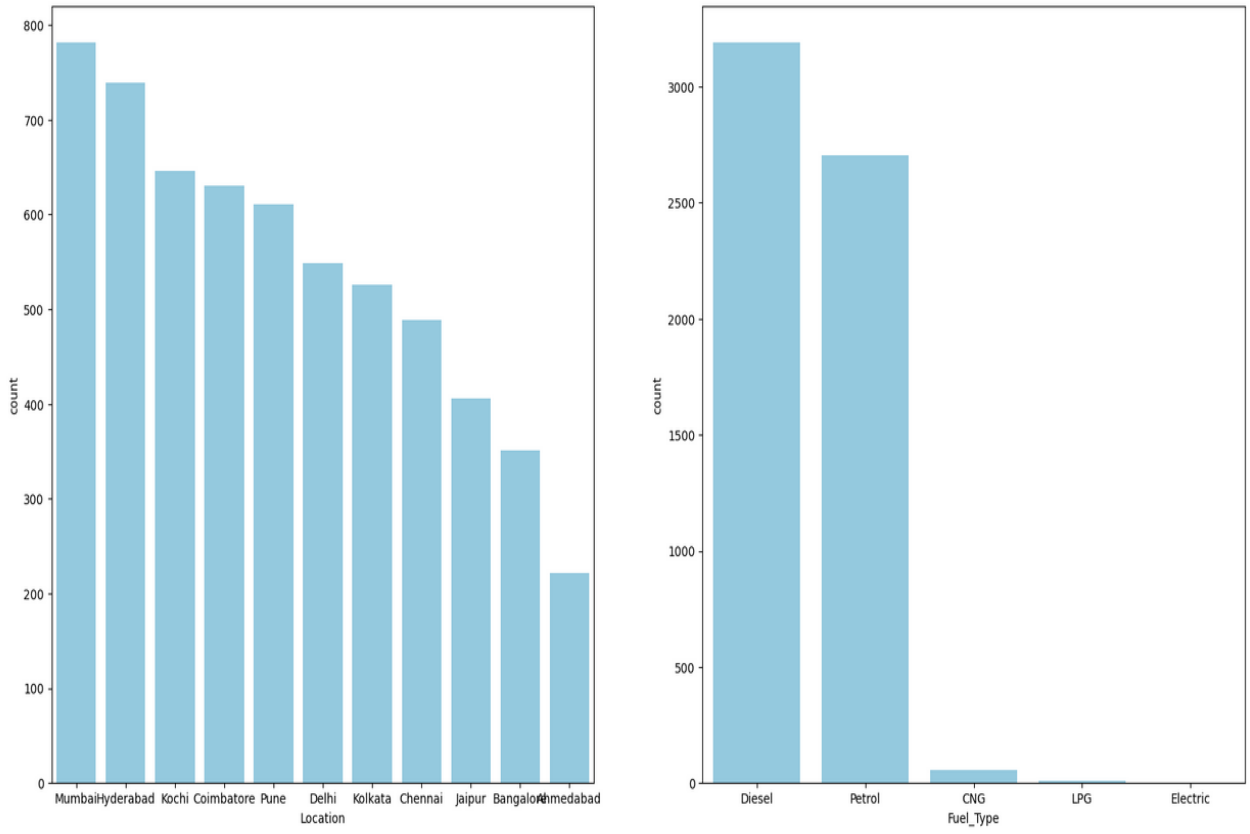
sns.countplot(ax = axes[0, 0], x = 'Location', data = data, color = 'skyblue',

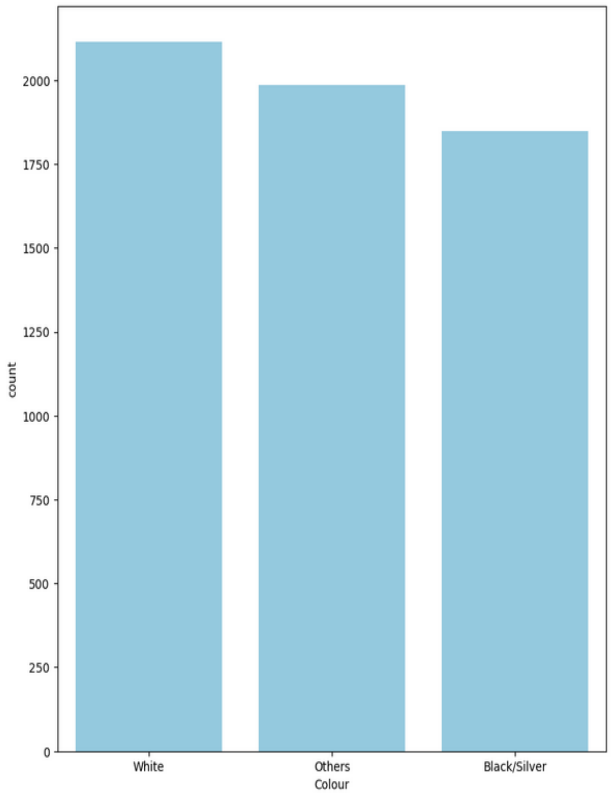
order = data['Location'].value_counts().index);

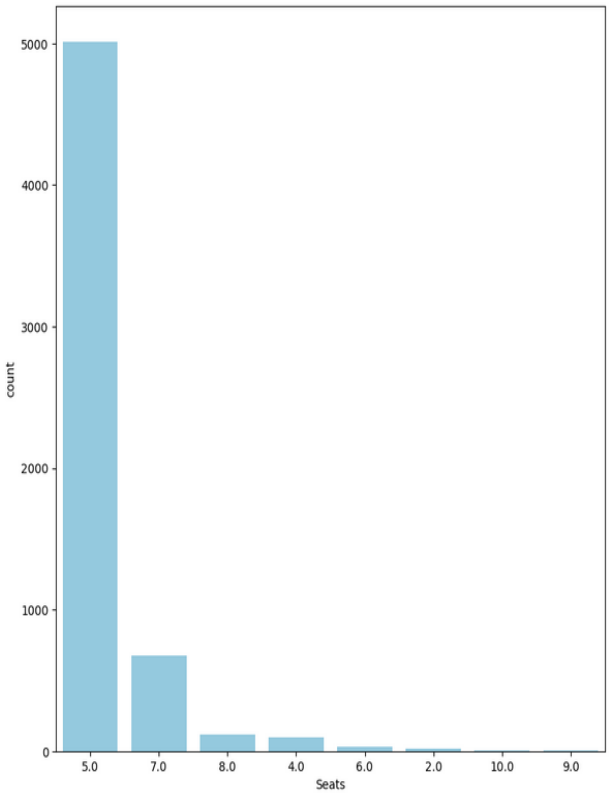
sns.countplot(ax = axes[0, 1], x = 'Fuel_Type', data = data, color = 'skyblue',

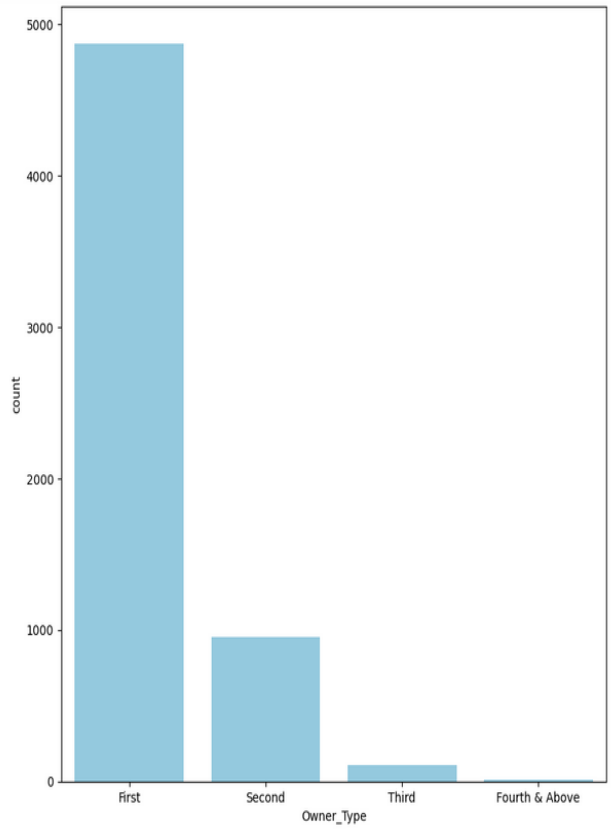
```
            order = data['Fuel_Type'].value_counts().index);
sns.countplot(ax = axes[1, 0], x = 'Owner_Type', data = data, color = 'skyblue',
            order = data['Owner_Type'].value_counts().index);
sns.countplot(ax = axes[1, 1], x = 'Transmission', data = data, color = 'skyblue',
            order = data['Transmission'].value_counts().index);
sns.countplot(ax = axes[2, 0], x = 'Colour', data = data, color = 'skyblue',
            order = data['Colour'].value_counts().index);
sns.countplot(ax = axes[2, 1], x = 'Seats', data = data, color = 'skyblue',
            order = data['Seats'].value_counts().index);
```

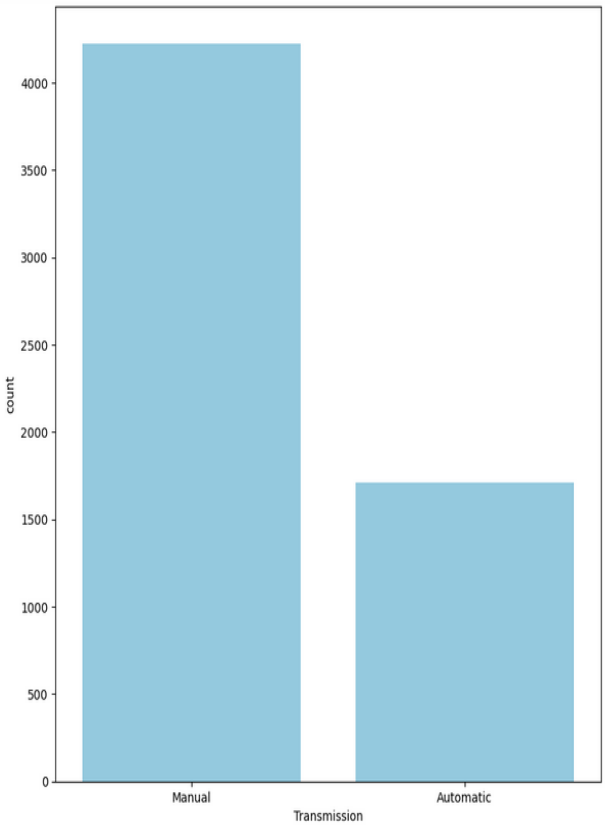
```
Bar plot for all categorical variables in the dataset
```




count
2000
1750
1500
1250
1000
750
500
250
0
White
Others
Black/Silver
Colour


count
5000
4000
3000
2000
1000
0
5.0
7.0
8.0
4.0
6.0
2.0
10.0
9.0
Seats

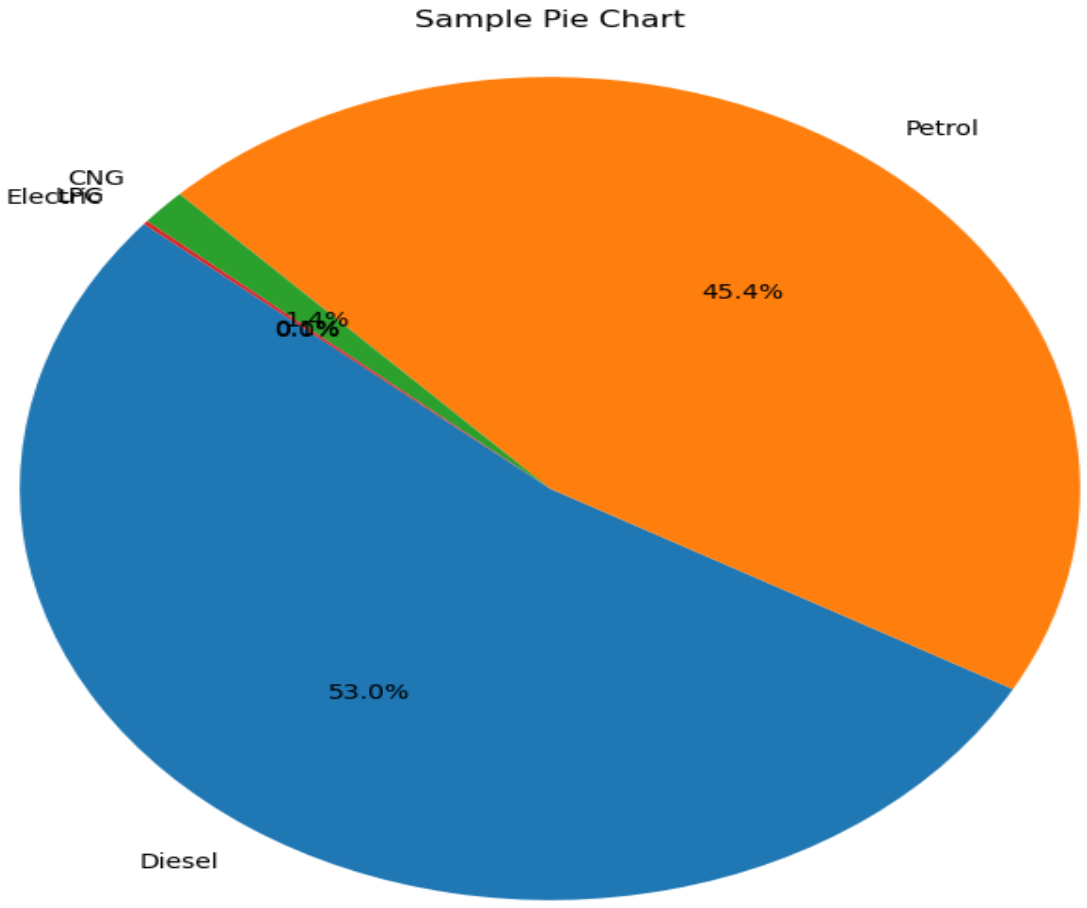

count
5000
4000
3000
2000
1000
0
First
Second
Third
Fourth & Above
Owner_Type


count
4000
3500
3000
2500
2000
1500
1000
500
0
Manual
Automatic
Transmission

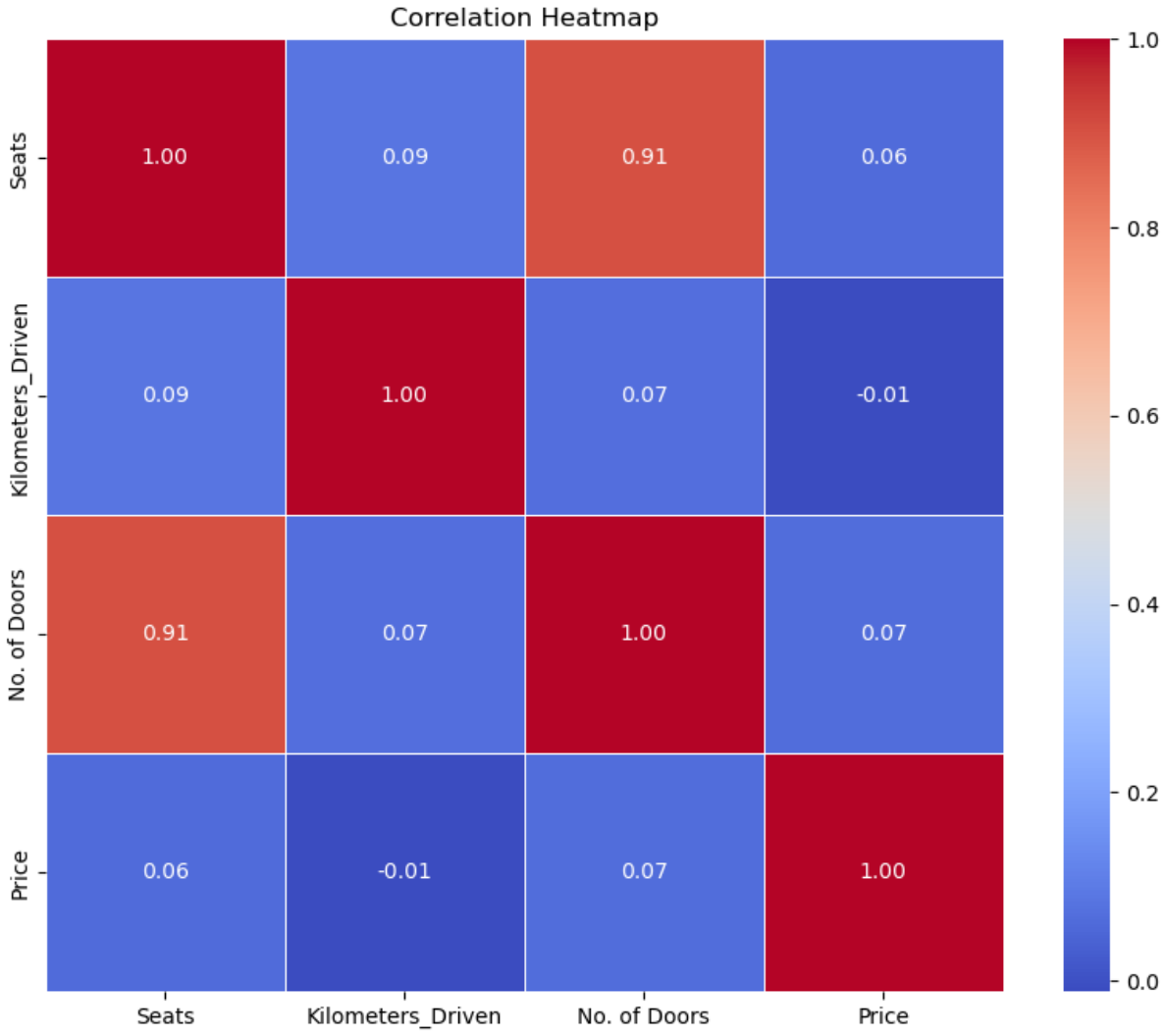
From the count plot, we can have below observations

- Mumbai has the highest number of cars available for purchase, followed by Hyderabad and Coimbatore
- ~53% of cars have fuel type as Diesel this shows diesel cars provide higher performance
- ~72% of cars have manual transmission
- ~75 % of cars are First owned cars. This shows most of the buyers prefer to purchase first-owner cars
- ~35 % of cars are with colour white.
- Cars with seat number 5 are hightest in number.

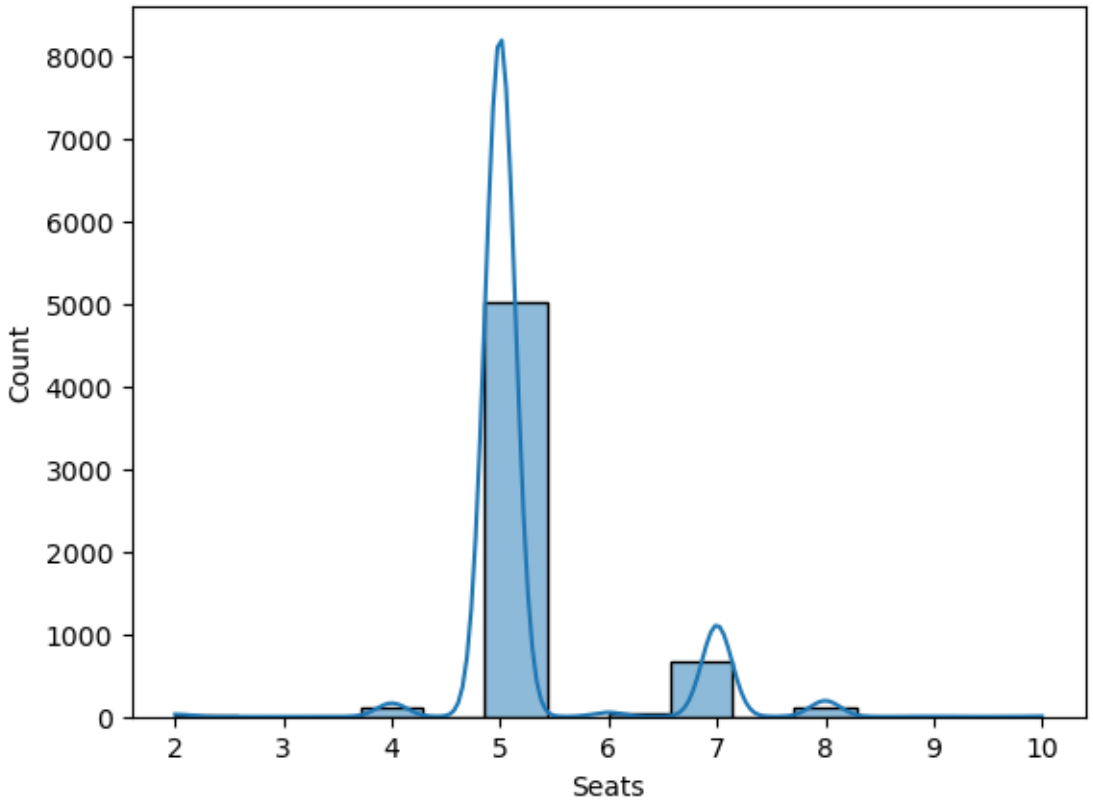
t.)
```
labels = ['Diesel','Petrol',"CNG","LPG","Electric"]
sizes = [3188,2731,86,9,0] #from above chart, we have, total number of males and females
plt.figure(figsize=(8,8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Sample Pie Chart')
plt.show()
```



u.) columns_for_heatmap = ['Seats', 'Kilometers_Driven','No. of Doors','Price']
heatmap_data = data[columns_for_heatmap]
correlation_matrix = heatmap_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
sns.histplot(data.Year, kde=True)
plt.show()



v.)sns.histplot(data.Seats, kde=True)
plt.show()



# ii.) Building various Models to Predict the price:-

(This part is performed by Devisetti VDSSS Desish and Neeharika PM)

## Model 1 :- RANDOM FOREST MODEL CODE

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer

df = pd.read_csv('Cars.csv')



premium_threshold = 20

df['Premium'] = (df['Price'] >= premium_threshold).astype(int)

# Extract features (X) and target variable (y)
X = df[['Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power', 'Seats']]
y = df['Price']

# Extract numeric values from 'Mileage', 'Engine', and 'Power'
X['Mileage'] = X['Mileage'].str.extract('(\d+\.\d+)').astype(float)
X['Engine'] = X['Engine'].str.extract('(\d+)').astype(float)
X['Power'] = X['Power'].str.extract('(\d+\.\d+)').astype(float)

# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')
X.loc[:, :] = imputer.fit_transform(X)
```

```
# Train the Random Forest model
rf_model = RandomForestRegressor()
rf_model.fit(X, y)

# Take user input for premium or non-premium
premium_input = input("Is the car premium or non-premium? (Type 'premium' or 'non-premium'): ").lower()

# Take user input for owner type
owner_type_input = input("Enter the owner type (Type 'first', 'second', or 'third'): ").lower()

# Take user input for the new car features
year = int(input("Enter the year of the car: "))
kilometers_driven = int(input("Enter the kilometers driven: "))
mileage = float(input("Enter the mileage (kmpl): "))
engine = float(input("Enter the engine capacity (cc): "))
power = float(input("Enter the power (bhp): "))
seats = int(input("Enter the number of seats: "))

# Create a DataFrame with user input
new_data = pd.DataFrame({
    'Year': [year],
    'Kilometers_Driven': [kilometers_driven],
    'Mileage': [f'{mileage} kmpl'],
    'Engine': [f'{engine} cc'],
    'Power': [f'{power} bhp'],
    'Seats': [seats]
```

```
})

# Extract and preprocess the features for the new car
new_features = new_data[['Year', 'Kilometers_Driven', 'Mileage',
'Engine', 'Power', 'Seats']]
new_features['Mileage'] =
new_features['Mileage'].str.extract('(\d+\.\d+)').astype(float)
new_features['Engine'] =
new_features['Engine'].str.extract('(\d+)').astype(float)
new_features['Power'] =
new_features['Power'].str.extract('(\d+\.\d+)').astype(float)

# Handle missing values using SimpleImputer
new_features = pd.DataFrame(imputer.transform(new_features),
columns=new_features.columns)

# Make predictions
predicted_price = rf_model.predict(new_features)

# Adjust predicted price based on owner type
if owner_type_input == 'first':
    predicted_price *= 1.1
elif owner_type_input == 'second':
    predicted_price *= 0.9
elif owner_type_input == 'third':
    predicted_price *= 0.7

if seats > 5:
    predicted_price *= 1.1
```

```
print(f"Adjusted Predicted Price of the Car: {predicted_price[0]}")
```

Output :-

```
Is the car premium or non-premium? (Type 'premium' or 'non-premium'):
premium
Enter the owner type (Type 'first', 'second', or 'third'): second
Enter the year of the car: 2009
Enter the kilometers driven: 94000
Enter the mileage (kmpl): 12.07
Enter the engine capacity (cc): 2967
Enter the power (bhp): 241.4
Enter the number of seats: 5
Adjusted Predicted Price of the Car: 12.235770000000002
```

## Model 2 :- LINEAR REGRESSION MODEL CODE

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

df = pd.read_csv('Cars.csv')

premium_threshold = 20

df['Premium'] = (df['Price'] >= premium_threshold).astype(int)

# Extract features (X) and target variable (y)
X = df[['Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power', 'Seats']]
y = df['Price']

# Extract numeric values from 'Mileage', 'Engine', and 'Power'
X['Mileage'] = X['Mileage'].str.extract('(\d+\.\d+)').astype(float)
X['Engine'] = X['Engine'].str.extract('(\d+)').astype(float)
X['Power'] = X['Power'].str.extract('(\d+\.\d+)').astype(float)

# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')
X.loc[:, :] = imputer.fit_transform(X)

# Train the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X, y)
```

```
# Take user input for premium or non-premium
premium_input = input("Is the car premium or non-premium? (Type
'premium' or 'non-premium'): ").lower()

# Take user input for owner type
owner_type_input = input("Enter the owner type (Type 'first', 'second', or
'third'): ").lower()

# Take user input for the new car features
year = int(input("Enter the year of the car: "))
kilometers_driven = int(input("Enter the kilometers driven: "))
mileage = float(input("Enter the mileage (kmpl): "))
engine = float(input("Enter the engine capacity (cc): "))
power = float(input("Enter the power (bhp): "))
seats = int(input("Enter the number of seats: "))

# Create a DataFrame with user input
new_data = pd.DataFrame({
    'Year': [year],
    'Kilometers_Driven': [kilometers_driven],
    'Mileage': [f'{mileage} kmpl'],
    'Engine': [f'{engine} cc'],
    'Power': [f'{power} bhp'],
    'Seats': [seats]
})

# Extract and preprocess the features for the new car
new_features = new_data[['Year', 'Kilometers_Driven', 'Mileage',
'Engine', 'Power', 'Seats']]
new_features['Mileage'] =
new_features['Mileage'].str.extract('(\d+\.\d+)').astype(float)
new_features['Engine'] =
new_features['Engine'].str.extract('(\d+)').astype(float)
new_features['Power'] =
new_features['Power'].str.extract('(\d+\.\d+)').astype(float)

# Handle missing values using SimpleImputer
new_features = pd.DataFrame(imputer.transform(new_features),
columns=new_features.columns)

# Make predictions
predicted_price = linear_model.predict(new_features)
```

```
# Adjust predicted price based on owner type
if owner_type_input == 'first':
    predicted_price *= 1.1
elif owner_type_input == 'second':
    predicted_price *= 0.9
elif owner_type_input == 'third':
    predicted_price *= 0.7

if seats > 5:
    predicted_price *= 1.1

print(f"Adjusted Predicted Price of the Car: {predicted_price[0]}")
```

## Output :-

```
Is the car premium or non-premium? (Type 'premium' or 'non-premium'):
premium
Enter the owner type (Type 'first', 'second', or 'third'): first
Enter the year of the car: 2014
Enter the kilometers driven: 4800
Enter the mileage (kmpl): 21.76
Enter the engine capacity (cc): 1995
Enter the power (bhp): 190
Enter the number of seats: 5
Adjusted Predicted Price of the Car: 19.788796919009428
```

# COMPARING ABOVE TWO MODELS FOR PREDICTING THE PRICES

Comparing the performance of different machine learning models requires thorough evaluation metrics and testing on relevant datasets. In your case, you have used both Random Forest and Linear Regression for predicting car prices. Here are some considerations:

Linear Regression:

Pros: Simplicity and interpretability: Linear regression is easy to understand, and the coefficients provide insights into the relationship between features and the target variable. Less prone to overfitting: Linear regression may be less prone to overfitting compared to more complex models.

Cons: Assumes a linear relationship: Linear regression assumes a linear relationship between the features and the target, which may not always hold in real-world scenarios. Limited flexibility: Linear regression may not capture complex, non-linear relationships in the data.

Random Forest:

Pros: High flexibility: Random Forest is capable of capturing complex relationships and interactions in the data, making it suitable for non-linear problems. Robust to overfitting: Random Forest is less prone to overfitting compared to individual decision trees.

Cons: Less interpretability: Random Forest models are more complex, and the interpretation of individual trees can be challenging. More computational resources: Random Forest may require more computational resources compared to simpler models like Linear Regression. Suggestions:

If your dataset exhibits non-linear relationships and interactions between features, Random Forest might perform better. If interpretability and simplicity are crucial, or if the relationships in your data are primarily linear, Linear Regression might be a better choice. To make a more informed decision, you should perform cross-validation, evaluate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared on both models, and consider the specific characteristics of your dataset. It's recommended to split your dataset into training and testing sets for a fair evaluation.

# iii.) Feature Engineering :-

(This part is performed by Rajvardhan Mall)

a.) data.columns

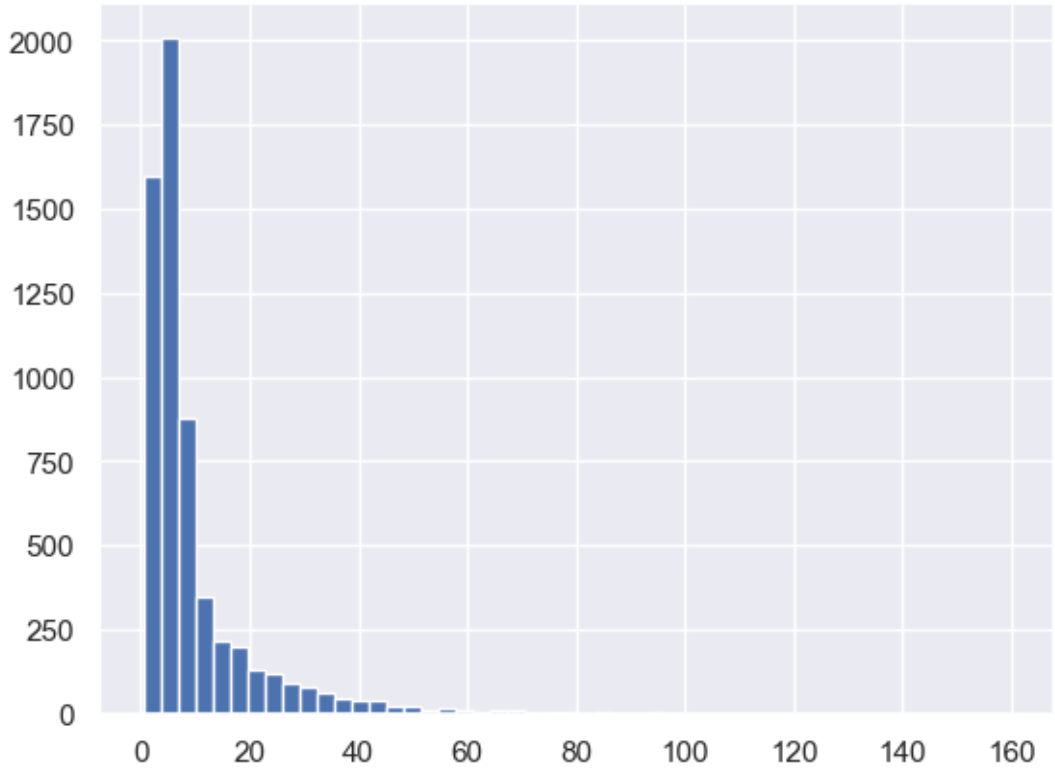
Output:-

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

Price Visualization

b.) data["Price"].hist(bins=50)

Output :-

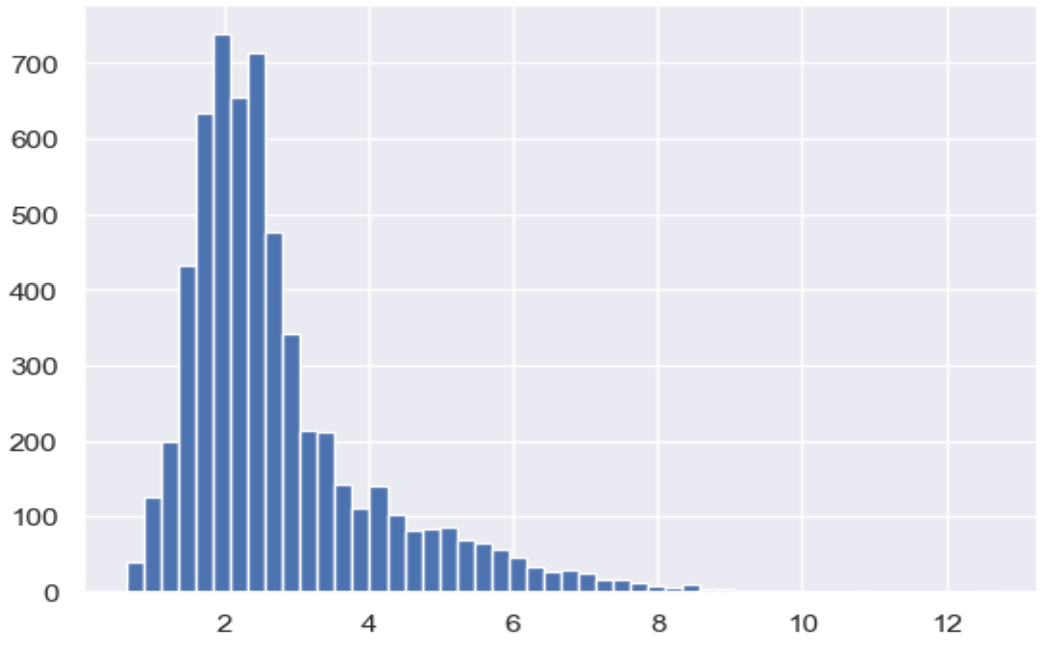



We see that its a right skewed distribution. In this case we can take the root or apply log function.

c.) Price_squareroot=np.sqrt(data['Price'])

Price_squareroot.hist(bins=50)

Output :-



d.) data['Price'].describe()

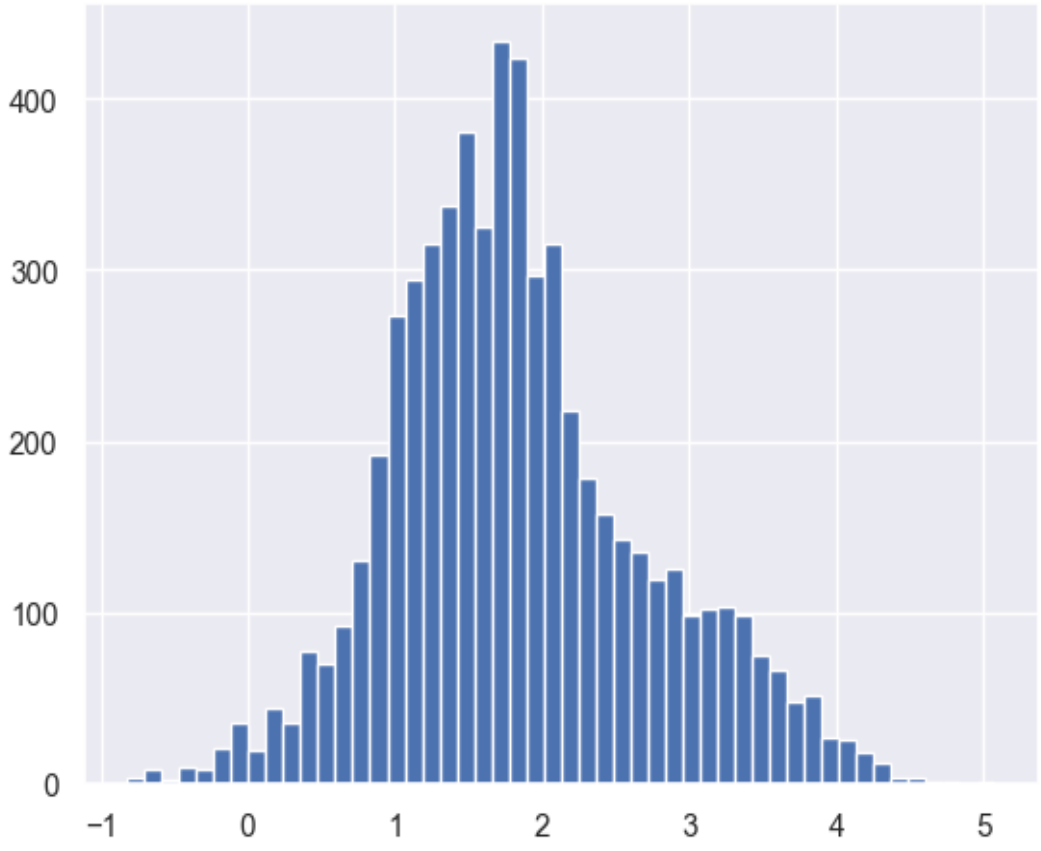
Output :-

```
count    5961.000000
mean        9.528103
std        11.214382
min         0.440000
25%         3.500000
50%         5.660000
75%        10.000000
max       160.000000
Name: Price, dtype: float64
```

We can see that the min value is not zero. Hence,the log is defined

e.) Price_log=np.log(data["Price"])
Price_log.hist(bins=50)

Output :-




f.) data['Price'].value_counts()

Output :-

```
Price
4.50     88
3.50     82
5.50     82
4.25     73
3.25     69
       ..
13.69     1
0.63      1
13.23     1
4.59      1
55.54     1
Name: count, Length: 1369, dtype: int64
```

g.) data_encoded=pd.get_dummies(data)
data_encoded.head()

Output :-

| | Kilometers_Driven | Seats | No. of Doors | Price | Name_Audi A3 | Name_Audi A4 | Name_Audi A6 | Name_Audi A7 | Name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 99000.0 | 8.0 | 5.0 | 6.00 | False | False | False | False | |
| **1** | 18678.0 | 5.0 | 4.0 | 8.32 | False | False | False | False | |
| **2** | 197000.0 | 7.0 | 5.0 | 4.00 | False | False | False | False | |
| **3** | 45000.0 | 5.0 | 4.0 | 3.49 | False | False | False | False | |
| **4** | 65000.0 | 8.0 | 5.0 | 6.40 | False | False | False | False | |

5 rows × 1732 columns