```
# Importing required libraries for data analysis and visualization
import pandas as pd                      # Pandas for data manipulation and analysis
import numpy as np                       # NumPy for numerical operations
import matplotlib.pyplot as plt          # Matplotlib for basic plotting
import seaborn as sns                    # Seaborn for statistical data visualization
import plotly.express as px              # Plotly Express for interactive visualizations

# Setting display options for Pandas to show three decimal places for floating-point numbers
pd.set_option('display.float_format', lambda x: '%.3f' % x)


# Display all columns without truncation
pd.set_option('display.max_columns', None)

# Load car-related dataset from URL into 'car' DataFrame
url = 'https://raw.githubusercontent.com/ArijitDhali/PrepInsta-DA-Week-6/main/cars_ds_final.csv'
car = pd.read_csv(url, encoding='unicode_escape')

# Display first two rows of the loaded DataFrame
car.head(2)
```

	Unnamed: 0	Make	Model	Variant	Ex-Showroom_Price	Displacement	Cylinders	Val
0	0	Tata	Nano Genx	Xt	Rs. 2,92,667	624 cc	2.000	
1	1	Tata	Nano Genx	Xe	Rs. 2,36,447	624 cc	2.000	

```
# Remove 'Unnamed: 0' column from 'car' DataFrame
del car["Unnamed: 0"]

# Display first two rows
car.head(2)
```

	Make	Model	Variant	Ex-Showroom_Price	Displacement	Cylinders	Valves_Per_Cy
0	Tata	Nano Genx	Xt	Rs. 2,92,667	624 cc	2.000	
1	Tata	Nano Genx	Xe	Rs. 2,36,447	624 cc	2.000	

```
car.dtypes       # Display data types of columns in the 'car' DataFrame
```

```
Make                 object
Model                object
Variant              object
Ex-Showroom_Price    object
Displacement         object
                     ...
USB_Ports           float64
Heads-Up_Display     object
Welcome_Lights       object
Battery              object
Electric_Range       object
Length: 140, dtype: object
```

```
car.info(verbose=True)       # Display concise information about 'car' DataFrame
```

```
 87   Lane_Watch_Camera/_Side_Mirror_Camera      object
 88   Passenger_Side_Seat-Belt_Reminder          object
 89   Seat_Back_Pockets                          object
 90   Voice_Recognition                          object
 91   Walk_Away_Auto_Car_Lock                    object
 92   ABS_(Anti-lock_Braking_System)             object
 93   Headlight_Reminder                         object
 94   Adjustable_Headrests                       object
 95   Gross_Vehicle_Weight                       object
 96   Airbags                                    object
 97   Door_Ajar_Warning                          object
 98   EBD_(Electronic_Brake-force_Distribution)  object
 99   Fasten_Seat_Belt_Warning                   object
 100  Gear_Shift_Reminder                        object
 101  Number_of_Airbags                          float64
 102  Compression_Ratio                          object
 103  Adjustable_Steering_Column                 object
 104  Other_Specs                                object
 105  Other_specs                                object
 106  Parking_Assistance                         object
 107  Key_Off_Reminder                           object
 108  USB_Compatibility                          object
 109  Android_Auto                               object
 110  Apple_CarPlay                              object
 111  Cigarette_Lighter                          object
 112  Infotainment_Screen                        object
 113  Multifunction_Steering_Wheel               object
 114  Average_Speed                              object
 115  EBA_(Electronic_Brake_Assist)              object
 116  Seat_Height_Adjustment                     object
 117  Navigation_System                          object
 118  Second_Row_AC_Vents                        object
 119  Tyre_Pressure_Monitoring_System            object
 120  Rear_Center_Armrest                        object
 121  iPod_Compatibility                         object
 122  ESP_(Electronic_Stability_Program)         object
 123  Cooled_Glove_Box                           object
 124  Recommended_Tyre_Pressure                  object
 125  Heated_Seats                               object
 126  Turbocharger                               object
 127  ISOFIX_(Child-Seat_Mount)                  object
 128  Rain_Sensing_Wipers                        object
 129  Paddle_Shifters                            object
 130  Leather_Wrapped_Steering                   object
 131  Automatic_Headlamps                        object
 132  Engine_Type                                object
 133  ASR_/_Traction_Control                     object
 134  Cruise_Control                             object
 135  USB_Ports                                  float64
 136  Heads-Up_Display                           object
 137  Welcome_Lights                             object
 138  Battery                                    object
 139  Electric_Range                             object
dtypes: float64(6), object(134)
memory usage: 1.4+ MB
```

```
car.shape
```

```
(1276, 140)
```

```
car.describe()
```

	Cylinders	Valves_Per_Cylinder	Doors	Seating_Capacity	Number_of_Ai
count	1210.000	1174.000	1272.000	1270.000	11
mean	4.381	3.978	4.550	5.270	
std	1.661	0.834	0.748	1.145	
min	2.000	1.000	2.000	2.000	
25%	4.000	4.000	4.000	5.000	
50%	4.000	4.000	5.000	5.000	
75%	4.000	4.000	5.000	5.000	

Data Cleaning and Trimming

Modifying each and every column accordingly to get a smooth analysis in data vizualization.

1. Handling the large missing values

Now, we will calculate and display the percentage of missing values for each column in the 'car' DataFrame. This information helps in understanding the completeness of the dataset and identifies columns with missing data.

```
row_size = car.shape[0]
for i in car.columns:
    if car[i].isnull().sum() > 0:
        print(i, "----------", (car[i].isnull().sum() / row_size) * 100)
```

```
Ambient_Lightning ---------- 95.61128526645768
Cargo/Boot_Lights ---------- 90.51724137931035
Drive_Modes ---------- 84.71786833855799
Engine_Immobilizer ---------- 4.702194357366771
High_Speed_Alert_System ---------- 83.69905956112854
Lane_Watch_Camera/_Side_Mirror_Camera ---------- 94.82758620689656
Passenger_Side_Seat-Belt_Reminder ---------- 86.59874608150471
Seat_Back_Pockets ---------- 34.71786833855799
Voice_Recognition ---------- 89.65517241379311
Walk_Away_Auto_Car_Lock ---------- 91.92789968652038
ABS_(Anti-lock_Braking_System) ---------- 10.344827586206897
Headlight_Reminder ---------- 20.219435736677116
Adjustable_Headrests ---------- 19.59247648902821
Gross_Vehicle_Weight ---------- 46.63009404388715
Airbags ---------- 10.344827586206897
Door_Ajar_Warning ---------- 11.206896551724139
EBD_(Electronic_Brake-force_Distribution) ---------- 15.752351097178682
Fasten_Seat_Belt_Warning ---------- 14.890282131661442
Gear_Shift_Reminder ---------- 42.554858934169275
Number_of_Airbags ---------- 10.579937304075235
Compression_Ratio ---------- 71.86520376175548
Adjustable_Steering_Column ---------- 14.968652037617556
Other_Specs ---------- 99.21630094043887
Other_specs ---------- 95.0626959247649
Parking_Assistance ---------- 22.727272727272727
Key_Off_Reminder ---------- 26.64576802507837
USB_Compatibility ---------- 20.924764890282134
Android_Auto ---------- 86.44200626959247
Apple_CarPlay ---------- 86.44200626959247
Cigarette_Lighter ---------- 53.44827586206896
Infotainment_Screen ---------- 36.206896551724135
Multifunction_Steering_Wheel ---------- 27.507836990595614
Average_Speed ---------- 40.12539184952978
EBA_(Electronic_Brake_Assist) ---------- 54.075235109717866
Seat_Height_Adjustment ---------- 25.54858934169279
Navigation_System ---------- 46.47335423197492
Second_Row_AC_Vents ---------- 47.17868338557994
Tyre_Pressure_Monitoring_System ---------- 73.35423197492163
Rear_Center_Armrest ---------- 43.808777429467085
iPod_Compatibility ---------- 51.95924764890282
ESP_(Electronic_Stability_Program) ---------- 61.05015673981191
Cooled_Glove_Box ---------- 50.54858934169278
Recommended_Tyre_Pressure ---------- 99.21630094043887
Heated_Seats ---------- 79.15360501567397
Turbocharger ---------- 48.43260188087774
ISOFIX_(Child-Seat_Mount) ---------- 52.19435736677116
Rain_Sensing_Wipers ---------- 62.539184952978054
Paddle_Shifters ---------- 76.17554858934169
Leather_Wrapped_Steering ---------- 53.99686520376176
Automatic_Headlamps ---------- 60.658307210031346
Engine_Type ---------- 97.33542319749216
ASR_/_Traction_Control ---------- 65.36050156739812
Cruise_Control ---------- 56.974921630094045
USB_Ports ---------- 97.72727272727273
Heads-Up_Display ---------- 96.00313479623824
Welcome_Lights ---------- 94.59247648902821
Battery ---------- 98.98119122257053
Electric_Range ---------- 98.66771159874608
```

```
car.duplicated().sum()
```

```
9
```

```
car = car.drop_duplicates()
car.duplicated().sum()
```

```
0
```

```
car.shape
```

```
(1267, 140)
```

```
row_size = car.shape[0]
for i in car.columns:
    if i != 'ARAI_Certified_Mileage_for_CNG':
        if car[i].isnull().sum() * 100 / row_size > 70:
            car.drop(columns=[i], inplace=True)
```

```
car.shape
```

```
(1267, 116)
```

Here, we fill the missing values in the 'Make' column with the corresponding values from the 'Model' column, updating the 'car' DataFrame in place.

```
car['Make'].fillna(car['Model'], inplace=True)
```

```
car['Make'].unique()
```

```
array(['Tata', 'Datsun', 'Renault', 'Maruti Suzuki', 'Hyundai', 'Premier',
       'Toyota', 'Nissan', 'Volkswagen', 'Ford', 'Mahindra', 'Fiat',
       'Honda', 'Jeep', 'Isuzu', 'Skoda', 'Audi', 'Mercedes-Benz B-Class',
       'Mercedes-Benz Cla-Class', 'Dc', 'Mini', 'Volvo', 'Jaguar', 'Bmw',
       'Land Rover', 'Mercedes-Benz E-Class Cabriolet', 'Porsche',
       'Mercedes-Benz Gls', 'Lexus', 'Mercedes-Benz S-Class', 'Maserati',
       'Mercedes-Benz G-Class', 'Mercedes-Benz Maybach',
       'Mercedes-Benz S-Class Cabriolet', 'Lamborghini', 'Bentley',
       'Ferrari', 'Aston Martin', 'Rolls-Royce Ghost Series Ii',
       'Rolls-Royce Wraith', 'Rolls-Royce Cullinan',
       'Rolls-Royce Phantom Coupe', 'Bugatti', 'Bajaj', 'Icml', 'Force',
       'Mg', 'Kia', 'Land Rover Rover', 'Mercedes-Benz E-Class',
       'Mercedes-Benz C-Class Cabriolet', 'Mercedes-Benz V-Class',
       'Mercedes-Benz E-Class All Terrain', 'Mercedes-Benz Amg-Gt',
       'Mercedes-Benz Amg Gt 4-Door Coupe', 'Rolls-Royce Dawn',
       'Rolls-Royce Drophead Coupe', 'Go+', 'Mercedes-Benz A-Class',
       'Mercedes-Benz C-Class', 'Mercedes-Benz Gle', 'Mercedes-Benz Cls',
       'Mitsubishi', 'Mercedes-Benz Gla-Class', 'Maruti Suzuki R',
       'Rolls-Royce Phantom', 'Mercedes-Benz Glc'], dtype=object)
```

```
car.loc[car['Make'].str.contains('Mercedes-Benz', na=False), 'Make'] = 'Mercedes-Benz'
car.loc[car['Make'].str.contains('Rolls-Royce', na=False), 'Make'] = 'Rolls-Royce'
car.loc[car['Make'].str.contains('Go+', na=False), 'Make'] = 'Datsun'
car.loc[car['Make'].str.contains('Maruti Suzuki R', na=False), 'Make'] = 'Maruti Suzuki'
car['Make'] = car['Make'].str.replace('Land Rover Rover', 'Land Rover')
```

```
car['Make'].unique()
```

```
array(['Tata', 'Datsun', 'Renault', 'Maruti Suzuki', 'Hyundai', 'Premier',
       'Toyota', 'Nissan', 'Volkswagen', 'Ford', 'Mahindra', 'Fiat',
       'Honda', 'Jeep', 'Isuzu', 'Skoda', 'Audi', 'Mercedes-Benz', 'Dc',
       'Mini', 'Volvo', 'Jaguar', 'Bmw', 'Land Rover', 'Porsche', 'Lexus',
       'Maserati', 'Lamborghini', 'Bentley', 'Ferrari', 'Aston Martin',
       'Rolls-Royce', 'Bugatti', 'Bajaj', 'Icml', 'Force', 'Mg', 'Kia',
       'Mitsubishi'], dtype=object)
```

```
car.loc[car['Make'].str.contains('Wagon', na=False), 'Make'] = 'Wagon R'
car['Model'] = car['Model'].str.replace('Wagon', 'Wagon R')
car['Model'] = car['Model'].str.replace('Mercedes-Benz ', '')
car['Model'] = car['Model'].str.replace('Rolls-Royce ', '')
car['Variant'] = car['Variant'].str.replace('Datsun ', '')
```

```
car.sample(5)
```

	Make	Model	Variant	Ex-Showroom_Price	Displacement	Cylinders	Val
1167	Jaguar	F-Type	2.0L Convertible	Rs. 1,01,44,987	1997 cc	6.000	
646	Mahindra	Marazzo	M4 7 Str	Rs. 11,56,471	1497 cc	4.000	
773	Skoda	Kodiaq Scout	2.0 Tdi At	Rs. 33,99,000	1968 cc	4.000	
184	Ford	Aspire	1.2 Ti-Vct Trend Plus	Rs. 6,97,400	1194 cc	4.000	
1212	Land Rover	Discovery	3.0 S Diesel	Rs. 88,77,094	2993 cc	6.000	

3. Processing Ex-Showroom_Price and Displacement Dataset

Here, we print the first few rows of specific columns, 'Ex-Showroom_Price' and 'Displacement', from the 'car' DataFrame.

```
specific_1 = ['Ex-Showroom_Price','Displacement']
print(car[specific_1].head())
```

```
  Ex-Showroom_Price Displacement
0      Rs. 2,92,667       624 cc
1      Rs. 2,36,447       624 cc
2      Rs. 2,96,661       624 cc
3      Rs. 3,34,768       624 cc
4      Rs. 2,72,223       624 cc
```

```
car['Ex-Showroom_Price'] = car['Ex-Showroom_Price'].replace(r'\D', '', regex=True)
car.rename(columns={'Ex-Showroom_Price': 'Ex-Showroom_Price_INR'}, inplace=True)
car['Ex-Showroom_Price_INR'] = car['Ex-Showroom_Price_INR'].astype(int)
```

```
car['Displacement'] = car['Displacement'].replace(r'\D', '', regex=True)
car.rename(columns={'Displacement': 'Displacement_cc'}, inplace=True)
car['Displacement_cc'] = car['Displacement_cc'].astype(float)
```

```
specific_1 = ['Ex-Showroom_Price_INR', 'Displacement_cc']
print(car[specific_1].head())
```

```
   Ex-Showroom_Price_INR  Displacement_cc
0                 292667          624.000
1                 236447          624.000
2                 296661          624.000
3                 334768          624.000
4                 272223          624.000
```

4. Ensuring Datatypes of Cylinders and Valves_Per_Cylinder Dataset

Here, we convert the 'Cylinders' and 'Valves_Per_Cylinder' column to float type in the 'car' DataFrame.

```
car['Cylinders'] = car['Cylinders'].astype(float)


car['Valves_Per_Cylinder'] = car['Valves_Per_Cylinder'].astype(float)
```

5. Processing Drivetrain Dataset

Now, we examine the unique values in the 'Drivetrain' column of the 'car' DataFrame to understand the different drivetrain configurations present in the dataset.

```
car['Drivetrain'].unique()
```

```
array(['RWD (Rear Wheel Drive)', 'FWD (Front Wheel Drive)',
       'AWD (All Wheel Drive)', '4WD', nan], dtype=object)
```

```
car['Drivetrain'] = car['Drivetrain'].str.replace(r'RearWheelDrive', '', regex=True)
car['Drivetrain'] = car['Drivetrain'].str.replace(r'FrontWheelDrive', '', regex=True)
car['Drivetrain'] = car['Drivetrain'].str.replace(r'AllWheelDrive', '', regex=True)
```

```
car['Drivetrain'].unique()
```

```
array(['RWD (Rear Wheel Drive)', 'FWD (Front Wheel Drive)',
       'AWD (All Wheel Drive)', '4WD', nan], dtype=object)
```

```
car['Emission_Norm'].unique()
```

```
array(['BS IV', 'BS 6', nan, 'BS III', 'BS VI'], dtype=object)
```

```
car['Emission_Norm'] = car['Emission_Norm'].str.replace('BS 6', 'BS VI')
```

```
car['Emission_Norm'].unique()
```

```
array(['BS IV', 'BS VI', nan, 'BS III'], dtype=object)
```

```
car['Engine_Location'].unique()
```

```
array(['Rear, Transverse', 'Front, Transverse', 'Front, Longitudinal',
       nan, 'Rear Mid, Transverse', 'Mid, Longitudinal',
       'Mid, Transverse', 'Rear, Longitudinal'], dtype=object)
```

```
car['Engine_Location'] = car['Engine_Location'].str.replace(', ', ' - ')
```

```
car['Engine_Location'].unique()
```

```
array(['Rear - Transverse', 'Front - Transverse', 'Front - Longitudinal',
       nan, 'Rear Mid - Transverse', 'Mid - Longitudinal',
       'Mid - Transverse', 'Rear - Longitudinal'], dtype=object)
```

7. Processing Fuel_Tank_Capacity, Height, Length, Width, Kerb_Weight and Ground_Clearance Dataset

Here, we print the first few rows of the 'Fuel_Tank_Capacity' column from the 'car' DataFrame to inspect the fuel tank capacity values.

```
fuel = ['Fuel_Tank_Capacity']
print(car[fuel].head())
```

```
  Fuel_Tank_Capacity
0          24 litres
1          24 litres
2          15 litres
3          24 litres
4          24 litres
```

```
car['Fuel_Tank_Capacity'] = car['Fuel_Tank_Capacity'].replace(r'\D', '', regex=True)
car.rename(columns={'Fuel_Tank_Capacity': 'Fuel_Tank_Capacity_litres'}, inplace=True)
car['Fuel_Tank_Capacity_litres'] = car['Fuel_Tank_Capacity_litres'].astype(float)
```

```
fuel = ['Fuel_Tank_Capacity_litres']
print(car[fuel].head())
```

```
   Fuel_Tank_Capacity_litres
0                     24.000
1                     24.000
2                     15.000
3                     24.000
4                     24.000
```

```
dimension = ['Height', 'Length', 'Width', 'Kerb_Weight', 'Ground_Clearance']
print(car[dimension].head())
```

```
     Height   Length    Width Kerb_Weight Ground_Clearance
0  1652 mm  3164 mm  1750 mm      660 kg           180 mm
1  1652 mm  3164 mm  1750 mm      725 kg           180 mm
2  1652 mm  3164 mm  1750 mm      710 kg           180 mm
3  1652 mm  3164 mm  1750 mm      725 kg           180 mm
4  1652 mm  3164 mm  1750 mm      725 kg           180 mm
```

```
car['Height'] = car['Height'].replace(r'\D', '', regex=True)
car['Length'] = car['Length'].replace(r'\D', '', regex=True)
car['Width'] = car['Width'].replace(r'\D', '', regex=True)
car['Kerb_Weight'] = car['Kerb_Weight'].replace(r'\D', '', regex=True)
car['Ground_Clearance'] = car['Ground_Clearance'].replace(r'\D', '', regex=True)

car.rename(columns={'Height': 'Height_mm','Length': 'Length_mm','Width': 'Width_mm'}, inplace=True)
car.rename(columns={'Kerb_Weight': 'Kerb_Weight_kg','Ground_Clearance':'Ground_Clearance_mm'}, inplace=True)

car['Height_mm'] = car['Height_mm'].astype(float)
car['Length_mm'] = car['Length_mm'].astype(float)
car['Width_mm'] = car['Width_mm'].astype(float)
car['Kerb_Weight_kg'] = car['Kerb_Weight_kg'].astype(float)
car['Ground_Clearance_mm'] = car['Ground_Clearance_mm'].astype(float)
```

```
dimension = ['Height_mm', 'Length_mm', 'Width_mm', 'Kerb_Weight_kg', 'Ground_Clearance_mm']
car[dimension].head()
```

	Height_mm	Length_mm	Width_mm	Kerb_Weight_kg	Ground_Clearance_mm
0	1652.000	3164.000	1750.000	660.000	180.000
1	1652.000	3164.000	1750.000	725.000	180.000
2	1652.000	3164.000	1750.000	710.000	180.000
3	1652.000	3164.000	1750.000	725.000	180.000
4	1652.000	3164.000	1750.000	725.000	180.000

8. Processing Body_Type, Gears, and Body_Type Dataset

Now, we explore the unique values in the 'Body_Type' column of the 'car' DataFrame to understand the different types of car bodies recorded in the dataset.

```
car['Body_Type'].unique()
```

```
array(['Hatchback', 'MPV', 'MUV', 'SUV', 'Sedan', 'Crossover', nan,
       'Coupe', 'Convertible', 'Sports, Hatchback', 'Sedan, Coupe',
       'Sports', 'Crossover, SUV', 'SUV, Crossover', 'Sedan, Crossover',
       'Sports, Convertible', 'Pick-up', 'Coupe, Convertible'],
      dtype=object)
```

```
car['Gears'].value_counts()
```

```
5                              614
6                              224
8                              139
7                              137
9                               30
4                               16
7 Dual Clutch                    1
Single Speed Reduction Gear      1
Name: Gears, dtype: int64
```

```
car['Gears'] = car['Gears'].replace({'Single Speed Reduction Gear': '1', '7 Dual Clutch': '7'})
car['Gears'] = car['Gears'].astype(float)
```

```
car['Gears'].value_counts()
```

```
5.000    614
6.000    224
8.000    139
7.000    138
9.000     30
4.000     16
1.000      1
Name: Gears, dtype: int64
```

```
car['Body_Type'] = car['Body_Type'].str.replace(', ', '-')
```

```
car['Body_Type'].unique()
```

```
array(['Hatchback', 'MPV', 'MUV', 'SUV', 'Sedan', 'Crossover', nan,
       'Coupe', 'Convertible', 'Sports-Hatchback', 'Sedan-Coupe',
       'Sports', 'Crossover-SUV', 'SUV-Crossover', 'Sedan-Crossover',
       'Sports-Convertible', 'Pick-up', 'Coupe-Convertible'], dtype=object)
```

9. Processing Mileage Datasets

Here, we count the number of missing values in the 'ARAI_Certified_Mileage' column of the 'car' DataFrame to assess the extent of missing mileage information.

```
car['ARAI_Certified_Mileage'].isnull().sum()
```

```
114
```

```
car.loc[car['Fuel_Type'] != 'CNG', 'ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].fillna(car['Highway_Mile
```

```
car['ARAI_Certified_Mileage'].isnull().sum()
```

```
101
```

```
car['ARAI_Certified_Mileage'].unique()
```

```
       '13.5 km/litre', '7.8 km/litre', '14 km/litre', '7.9 km/litre',
       '10.63 km/litre', '6.71 km/litre', '8.6 km/litre', '9 km/litre',
       '8.77 km/litre', '17.3 km/litre', '7.29 km/litre', '10.2 km/litre',
       '9.5 km/litre', '22.4-21.9 km/litre', '22.8 km/litre',
       '5.95 km/litre', '35 km/litre', '22.05 km/litre', '21.4 km/litre',
       '21.7 km/litre', '19.77 km/litre', '25.32 km/litre',
       '25.2 km/litre', '25 km/litre', '11.12 km/litre', '17.6 km/litre',
```

```
       '22.71 km/litre', '17.97 km/litre', '19.34 km/litre',
       '18.69 km/litre', '24.2 km/litre', '17.5 km/litre',
       '23.65 km/litre', '13.86 km/litre', '20.37 km/litre',
       '13.9 km/litre', '14.02 km/litre', '16.1 km/litre',
       '20.45 km/litre', '13.24 km/litre', '17.41 km/litre',
       '14.6 km/litre', '18.42 km/litre', '13.03 km/litre',
       '16.38 km/litre', '12.95 km/litre', '17.42 km/litre',
       '15.5 km/litre', '14.41 km/litre', '19.19 km/litre',
       '16.05 km/litre', '1449 km/litre', '17.8 km/litre',
       '21.27 km/litre', '8.9 km/litre', '9.1 km/litre', '24.7 km/litre',
       '20.7 km/litre', '17.36 km/litre', '23.08 km/litre',
       '28.09 km/litre', '21.56 km/litre', '20.28 km/litre',
       '26.82 km/litre', '26.32 km/litre', '21.38 km/litre',
       '15.29 km/litre', '19.67 km/litre', '10.3 km/litre',
       '9.62 km/litre', '18.12 km/litre', '14.69 km/litre',
       '15.01 km/litre', '16.46 km/litre', '7.94 km/litre',
       '21.76 km/litre', '14.3 km/litre', '15.3 km/litre',
       '10.7 km/litre', '10.85 km/litre', '23.84 km/litre',
       '27.28 km/litre', '16.21 km/litre', '14.23 km/litre',
       '23.2 km/litre', '9.17 km/litre', '21.2 km/litre',
       '13.51 km/litre', '18.18 km/litre', '10.77 km/litre', '4 km/litre',
       '17.4 km/litre', '25.6 km/litre', '25.1 km/litre', '22.6 km/litre',
       '11.56 km/litre'], dtype=object)
```

```
car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].str.replace(r'\s*km/litre', '', regex=True)
car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].str.replace('9.8-10.0', '10.0')
car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].str.replace('22.4-21.9', '22.4')
car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].str.replace(r'\s*kmpl', '', regex=True)

car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].astype(float)


car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].replace(142.0, 14.2)
car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].replace(1449.0, 14.49)

car.rename(columns={'ARAI_Certified_Mileage': 'ARAI_Certified_Mileage_kmpl'}, inplace=True)
```

```
<ipython-input-52-da7edab2ae80>:2: FutureWarning: The default value of regex will change from True to False in a
  car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].str.replace('9.8-10.0', '10.0')
<ipython-input-52-da7edab2ae80>:3: FutureWarning: The default value of regex will change from True to False in a
  car['ARAI_Certified_Mileage'] = car['ARAI_Certified_Mileage'].str.replace('22.4-21.9', '22.4')
```

```
car['ARAI_Certified_Mileage_kmpl'].unique()                  # Display unique values in the cleaned 'ARAI_Certified_
```

```
array([23.6 ,   nan, 21.9 , 25.17, 22.5 , 23.  , 23.01, 24.04, 15.  ,
       24.07, 20.1 , 23.1 , 20.3 , 24.  , 20.89, 20.5 , 16.  , 23.7 ,
       18.16, 23.59, 18.97, 19.49, 22.95, 17.57, 15.7 , 20.14, 20.  ,
       28.4 , 22.  , 18.6 , 22.54, 18.15, 25.35, 20.4 , 25.4 , 18.78,
       17.21, 19.91, 24.4 , 19.  , 21.66, 21.73, 26.1 , 19.4 , 16.78,
       17.71, 20.08, 17.1 , 23.87, 21.01, 19.56, 18.2 , 27.3 , 16.3 ,
       13.8 , 12.35, 12.05, 17.06, 14.4 , 19.5 , 18.3 , 13.7 , 18.19,
       20.38, 19.2 , 14.81, 15.04, 17.9 , 10.  , 15.73, 16.9 , 21.15,
       14.59, 16.47, 11.  , 13.12, 19.33, 14.11, 18.56, 13.57, 13.38,
       11.24, 13.  , 12.8 ,  8.61,  7.4 , 11.6 , 12.  , 11.13, 18.  ,
       10.8 , 17.66, 11.86,  7.96, 13.5 ,  7.8 , 14.  ,  7.9 , 10.63,
        6.71,  8.6 ,  9.  ,  8.77, 17.3 ,  7.29, 10.2 ,  9.5 , 22.4 ,
       22.8 ,  5.95, 35.  , 22.05, 21.4 , 21.7 , 19.77, 25.32, 25.2 ,
       25.  , 11.12, 17.6 , 20.65, 27.4 , 23.8 , 11.9 , 17.52, 18.27,
       23.97, 16.5 , 15.96, 24.3 , 17.19, 21.19, 17.  , 14.8 , 13.87,
       19.6 , 17.7 , 23.9 , 23.5 , 19.1 , 21.04, 19.98, 16.09, 20.64,
       21.5 , 15.4 , 19.01, 17.01, 18.49, 10.22, 21.72, 14.84, 21.13,
       15.41, 13.85, 15.1 , 10.83, 10.75, 11.25, 14.1 , 16.7 , 21.  ,
       16.8 , 26.8 , 14.12, 14.67, 16.81, 12.9 , 14.24, 10.26, 10.01,
       10.91, 12.62, 15.75, 16.25, 20.68, 17.05, 18.5 , 19.62, 16.13,
       12.44, 15.6 , 14.2 ,  8.5 , 18.8 , 16.4 , 13.32,  8.4 ,  7.6 ,
       14.75, 11.68, 12.63, 12.06, 17.2 , 11.5 ,  3.4 , 16.2 ,  9.8 ,
        9.6 , 10.6 ,  8.  ,  5.5 ,  5.  ,  8.8 , 20.6 , 25.5 , 27.39,
       18.9 , 17.49, 14.9 , 16.95, 22.71, 17.97, 19.34, 18.69, 24.2 ,
       17.5 , 23.65, 13.86, 20.37, 13.9 , 14.02, 16.1 , 20.45, 13.24,
       17.41, 14.6 , 18.42, 13.03, 16.38, 12.95, 17.42, 15.5 , 14.41,
       19.19, 16.05, 14.49, 17.8 , 21.27,  8.9 ,  9.1 , 24.7 , 20.7 ,
       17.36, 23.08, 28.09, 21.56, 20.28, 26.82, 26.32, 21.38, 15.29,
       19.67, 10.3 ,  9.62, 18.12, 14.69, 15.01, 16.46,  7.94, 21.76,
       14.3 , 15.3 , 10.7 , 10.85, 23.84, 27.28, 16.21, 14.23, 23.2 ,
        9.17, 21.2 , 13.51, 18.18, 10.77,  4.  , 17.4 , 25.6 , 25.1 ,
       22.6 , 11.56])
```

```
car['ARAI_Certified_Mileage_for_CNG'].unique()
```

```
array([nan, '36 km/kg', '20 km/kg', '32.26 km/kg', '31.76 km/kg',
       '30.48 km/kg', '25 km/kg', '26.6 km/kg', '20.5 km/kg',
       '20.4 km/kg', '43 km/kg', '31.79 km/kg', '18.9 km/kg',
       '14.16 km/kg', '13.96 km/kg', '10.8 km/kg', '33.54 km/kg'],
      dtype=object)
```

```
car['ARAI_Certified_Mileage_for_CNG'] = car['ARAI_Certified_Mileage_for_CNG'].str.replace(r'\s*km/kg', '', regex=True)
car['ARAI_Cecar[['Front_Track', 'Rear_Track']].head(rtified_Mileage_for_CNG'] = car['ARAI_Certified_Mileage_for_CNG'].
```

```
car.rename(columns={'ARAI_Certified_Mileage_for_CNG': 'ARAI_Certified_Mileage_for_CNG_kmpkg'}, inplace=True)
```

```
car['ARAI_Certified_Mileage_for_CNG_kmpkg'].unique()
```

```
array([  nan, 36.  , 20.  , 32.26, 31.76, 30.48, 25.  , 26.6 , 20.5 ,
       20.4 , 43.  , 31.79, 18.9 , 14.16, 13.96, 10.8 , 33.54])
```

```
car['ARAI_Certified_Mileage_kmpl'].fillna(car['ARAI_Certified_Mileage_for_CNG_kmpkg'], inplace=True)
```

```
mileage_drop = ['City_Mileage', 'Highway_Mileage','ARAI_Certified_Mileage_for_CNG_kmpkg']
car.drop(columns=mileage_drop, inplace=True)
```

10. Processing Suspension Datasets

Here, we examine the unique values in the 'Front_Suspension' column of the 'car' DataFrame to understand the different front suspension configurations recorded in the dataset.

```
car['Front_Suspension'].unique()
```

```
       'AIRMATIC', 'McPherson spring strut suspension',
       'Front independent double wishbone design coil springs, anti-roll bar and adaptive damping',
       'Rigid leaf spring', 'Double joint spring-strut axle',
       'Aluminium double-joint springs strut axle with anti-roll bar',
       'Aluminium double track control arm axle with separate lower track arm level, small steering roll
  radius, traverse force compensation, anti-dive',
       'Fully independent double wishbone suspension',
       'Sports Suspension with Adaptive Damping',
       'Double wishbone, coil springs, single-tube gas-filled shock absorber, stabiliser bar',
       'Double-Joint Spring Strut Front Axle', 'Mcpherson strut Type',
       'Mc Pherson suspension with lower triangular links and torsion stabiliser',
       'Double Wishbone with coil spring suspension',
       'McPherson strut,coil spring'], dtype=object)
```

```
car.loc[car['Front_Suspension'].str.contains('MacPherson|Mac Pherson|Mc Pherson|McPherson|Macpherson|Mcpherson', na=F
car.loc[car['Front_Suspension'].str.contains('Double Wishbone|Double wishbone|double-wishbone|double wishbone|Double
car.loc[car['Front_Suspension'].str.contains('link|Link', na=False), 'Front_Suspension'] = 'Multi Link Suspension'
car.loc[car['Front_Suspension'].str.contains('Adaptive|adaptive', na=False), 'Front_Suspension'] = 'Adaptive Suspensi
car.loc[car['Front_Suspension'].str.contains('Leaf spring|leaf spring|Leaf Spring', na=False), 'Front_Suspension'] =
car.loc[car['Front_Suspension'].str.contains('Air|air|AIRMATIC', na=False), 'Front_Suspension'] = 'Air Suspension'
car.loc[car['Front_Suspension'].str.contains('Damper|damper|damping|CONTROL', na=False), 'Front_Suspension'] = 'Dampe
car.loc[car['Front_Suspension'].str.contains('Independent|independent', na=False), 'Front_Suspension'] = 'Independent
car.loc[car['Front_Suspension'].str.contains('Double joint|Double-Joint', na=False), 'Front_Suspension'] = 'Double Jo
car.loc[car['Front_Suspension'].str.contains('single joint ', na=False), 'Front_Suspension'] = 'Single Joint Suspensi
car.loc[~car['Front_Suspension'].str.contains('MacPherson|Double Wishbone|Multi Link|Adaptive|Leaf Spring|Air|Damper|
```

```
car['Front_Suspension'].unique()
```

```
array(['MacPherson Suspension', 'Double Pivot Arm',
       'Aluminium Suspension', 'Independent Suspension',
       'Double Wishbone Suspension', 'Multi Link Suspension',
       'Adaptive Suspension', 'Damper Suspension', 'Air Suspension',
       'Leaf Spring Suspension',
       'Hydraulic Double Acting, Telescopic Shock Absorber',
       'Double Joint Suspension'], dtype=object)
```

```
car['Rear_Suspension'].unique()
```

```
automatic self-leveling',
       'Self-tracking trapezoidal-link suspension',
       'Sports Suspension with Adaptive Damping',
       'Double wishbone, Coil Spring, Gas damper, Anti roll bar',
       'Integral rear axle and M specific set-up', 'Multi Link Setup',
       'Torsion Beam link with coil spring',
       'Isolated trailing link with coil spring',
       'Five-link, Coil springs',
       'Double parabolic leaf spring, Hydraulic telescopic',
       'Double wishbones',
       'Torsion Beam Axle with Stabilizer, Coil Spring',
       'compound link crank axle', '3-link Type coil spring suspension',
       'Torsion Beam Axle,Coil Spring',
       'Multi-link coil springs with stabilizer bar'], dtype=object)
```

```
car.loc[car['Rear_Suspension'].str.contains('multi-link|multilink', case=False, na=False), 'Rear_Suspension'] = 'Mult
car.loc[car['Rear_Suspension'].str.contains('torsion beam|torsion bar', case=False, na=False), 'Rear_Suspension'] = '
car.loc[car['Rear_Suspension'].str.contains('double wishbone|wishbone', case=False, na=False), 'Rear_Suspension'] = '
car.loc[car['Rear_Suspension'].str.contains('coil spring|coil-over|coilover', case=False, na=False), 'Rear_Suspensior
car.loc[car['Rear_Suspension'].str.contains('independent|macpherson|mcpherson', case=False, na=False), 'Rear_Suspensi
car.loc[car['Rear_Suspension'].str.contains('leaf spring|leaf-sprung|leafsprung', case=False, na=False), 'Rear_Susper
car.loc[~car['Rear_Suspension'].str.contains('Multi-link|Torsion Beam|Double Wishbone|Coil Spring|Independent|Leaf Sp
```

```
car['Rear_Suspension'].unique()
```

```
array(['Coil Spring Suspension', 'Torsion Beam Suspension',
       'Other Suspension', 'Independent Suspension',
       'Multi-link Suspension', 'Double Wishbone Suspension',
       'Leaf Spring Suspension'], dtype=object)
```

11. Processing Track Datasets

Here, we show the first few rows of the 'Front_Track' and 'Rear_Track' columns from the 'car' DataFrame for inspection.

```
car[['Front_Track', 'Rear_Track']].head()
```

	Front_Track	Rear_Track
0	1325 mm	1315 mm
1	1325 mm	1315 mm
2	1325 mm	1315 mm
3	1325 mm	1315 mm
4	1325 mm	1315 mm

```
# Clean and standardize 'Front_Track' and 'Rear_Track' columns in 'car' DataFrame
car['Front_Track'] = car['Front_Track'].replace(r'\D', '', regex=True)
car['Rear_Track'] = car['Rear_Track'].replace(r'\D', '', regex=True)

# Rename columns to reflect units and data type
car.rename(columns={'Front_Track': 'Front_Track_mm','Rear_Track': 'Rear_Track_mm'}, inplace=True)

# Convert cleaned columns to float type
car['Front_Track_mm'] = car['Front_Track_mm'].astype(float)
car['Rear_Track_mm'] = car['Rear_Track_mm'].astype(float)


# Display the cleaned and standardized 'Front_Track_mm' and 'Rear_Track_mm' columns from the 'car' DataFrame
car[['Front_Track_mm', 'Rear_Track_mm']].head()
```

	Front_Track_mm	Rear_Track_mm
0	1325.000	1315.000
1	1325.000	1315.000
2	1325.000	1315.000
3	1325.000	1315.000
4	1325.000	1315.000

12. Processing Tire Datasets

This code removes spaces from the 'Front_Tyre_&_Rim' column in the 'car' DataFrame and displays unique values, ensuring consistent formatting.

```
# Remove spaces from the 'Front_Tyre_&_Rim' column in 'car' DataFrame and display unique values
car['Front_Tyre_&_Rim'] = car['Front_Tyre_&_Rim'].str.replace(' ','',regex=False)
car['Front_Tyre_&_Rim'].unique()
```

```
array(['135/70R12', '80/155R13', '155/80R13', '165/70R14', '155R13LT',
       '155/65R13', '155/70R13', nan, '175/65R14', '175/60R15',
       '175/65R15', '185/65R14', '185/65R15', '205/70R15', '205/70',
       '185/60R15', '165/65R14', '165/80R14', '185/70R14', '195/55R16',
       '175/70R14', '185/60R16', '195/55R15', '175/70R15', '205/55R16',
       '175/65', '225/60R17', '265/60R17', '245/70R17', '255/60R18',
       '215/65R17', '235/55R18', '235/65R18', '215/55R17', '205/55R17',
       '225/45R17', '255/35R20', '215/65R16', '235/55R17', '235/55R19',
       '225/50R1794Y', '43.66cm/17"LightAlloyWheelsJCWTrackSpoke,Black',
       '225/55R18', '225/60R19', '245/50R19', '235/60R18', '245/45R18',
       '245/40R18', '255/50R19', '255/40R19', '265/60R18', '265/50R20',
       '295/45R20', '255/40R18', '245/35ZR19', '235/40R18', '275/55R19',
       '295/40R21', '235/60VR18', '265/35R19', '(245/50R18', '245/45R19',
       '245/40R20', '275/50R20', '255/45ZR19', '255/60ZR18', '285/45R21',
       '255/45R19', '245/35ZR20', '245/40R21', '285/60R18', '235/35R19',
       '245/35R19', '285/45ZR21', '275/40R20', '245/40ZR19', '255/40ZR20',
       '255/45R20', '265/45R20', '255/50R21', '285/30ZR20', 'R12',
       '145/80R12', '145/80R13', '185/80R14', '205/65R15', '195/65R15',
       '215/60R16', '195/65R15Steel', '195/60R16', '195/60R15',
       '215/75R15', '185/75R16', '205/60R16', '205/50R17', '205/65R16',
       '185/65R16', '165/60/R14', '235/70R16', 'P235/70R16', '245/70R16',
       '215/70R15', '21565R16', '215/60R17', 'P215/75R15', '245/75R16',
       'P235/65R17', '235/65R17', 'P235/65R15', 'P235/60R18', '265/60R16',
       '215/55R16', '215/50R17', '115/90R17', '265/65R17', '265/65R18',
       '225/55R17', '225/50R18', '225/40R18', '225/50R17', '245/45R17',
       '245/35R17', '245/75R15', '255/335R19', '255/55R19', '255/60R19',
       '155/85R18', '235/65R20', '275/45R21', '255/50', '255/55ZR19',
       '285/40ZR21', '275/35R20', '275/40R22', '265/35ZR19', '275/35R19',
       '255/35R19', '245/30R20', '255/35', '255/30ZR20', '145R12LT6PR',
       '205/60/R16', 'R16', '115/90R16', '205/55', '205/55R18',
       '235/45R18', '225/60R18', '245/45R20', '255/50R20', '225/45R18',
       '255/40R20', '255/40/20', '275/35ZR20', '215/70', '235/50R18',
       '235/45R19', '235/45R18TubelessRadials', '245/40', '245/40R19',
       '155/65R14', '195R15LT,8PRRadial', '255/55R20', '245/45RF20',
       '275/45ZR20', '275/40R19', '185/55R16'], dtype=object)
```

```
# Split and extract information from 'Front_Tyre_&_Rim' column, and clean relevant columns
car[['F_Tire_Width', 'F_Tire']] = car['Front_Tyre_&_Rim'].str.split('/', n=1, expand=True)
car[['F_Tire_Aspect_Ratio', 'F_Tire_Diameter']] = car['F_Tire'].str.split('R', n=1, expand=True)

f_drop = ['F_Tire']
car.drop(columns=f_drop, inplace=True)          # Drop the original 'F_Tire' column

# Clean and replace specific values in the extracted columns
car['F_Tire_Width'] = car['F_Tire_Width'].replace({'155R13LT':'155','(245':'245','R12':np.nan,'R16':np.nan,'43.66cm':
car['F_Tire_Aspect_Ratio'] = car['F_Tire_Aspect_Ratio'].replace({'17"LightAlloyWheelsJCWTrackSpoke,Black':'43','35Z':
car['F_Tire_Diameter'] = car['F_Tire_Diameter'].replace({'15Steel':'15','18TubelessRadials':'18','F20':'20','1794Y':r

# Convert the cleaned columns to float type
car['F_Tire_Width'] = car['F_Tire_Width'].astype(float)
car['F_Tire_Aspect_Ratio'] = car['F_Tire_Aspect_Ratio'].astype(float)
car['F_Tire_Diameter'] = car['F_Tire_Diameter'].astype(float)
```

```
car['F_Tire_Width'].unique()
```

```
array([135.,  80., 155., 165.,  nan, 175., 185., 205., 195., 225., 265.,
       245., 255., 215., 235., 295., 275., 285., 145., 115.])
```

```
car['F_Tire_Aspect_Ratio'].unique()
```

```
array([ 70.,  nan,  80.,  65.,  60.,  55.,  45.,  35.,  50.,  43.,  40.,
        30.,  75.,  90., 335.,  85.])
```

```
car['F_Tire_Diameter'].unique()
```

```
array([12., 13., 14., nan, 15., 16., 17., 18., 20., 19., 21., 22.])
```

```
width = car.columns.get_loc('F_Tire_Width')
aspect = car.columns.get_loc('F_Tire_Aspect_Ratio')
diameter = car.columns.get_loc('F_Tire_Diameter')
front = car.columns.get_loc('Front_Tyre_&_Rim')

column_to_move = car.pop('F_Tire_Width')
car.insert(front, 'F_Tire_Width', column_to_move)         # Move the columns to the desired positions

column_to_move = car.pop('F_Tire_Aspect_Ratio')
car.insert(front, 'F_Tire_Aspect_Ratio', column_to_move)

column_to_move = car.pop('F_Tire_Diameter')
car.insert(front, 'F_Tire_Diameter', column_to_move)

# Drop the original 'Front_Tyre_&_Rim' column and rename the reordered columns
car.drop(columns=['Front_Tyre_&_Rim'], inplace=True)
car.rename(columns={'F_Tire_Width': 'F_Tire_Width_mm', 'F_Tire_Diameter': 'F_Tire_Diameter_inch'}, inplace=True)



# Remove spaces from the 'Rear_Tyre_&_Rim' column in 'car' DataFrame and display unique values
car['Rear_Tyre_&_Rim'] = car['Rear_Tyre_&_Rim'].str.replace(' ','', regex=False)
car['Rear_Tyre_&_Rim'].unique()
```

```
array(['155/65R12', '80/155R13', '155/80R13', '165/70R14', '155R13LT',
       '155/65R13', '155/70R13', nan, '175/65R14', '175/60R15',
       '175/65R15', '185/65R14', '185/65R15', '205/70R15', '205/70',
       '185/60R15', '165/65R14', '165/80R14', '185/70R14', '195/55R16',
       '175/70R14', '185/60R16', '195/55R15', '175/70R15', '205/55R16',
       '175/65', '225/60R17', '265/60R17', '245/70R17', '255/60R18',
       '215/65R17', '235/55R18', '235/60R18', '215/55R17', '205/55R17',
       '225/45R17', '295/30R20', '215/65R16', '235/55R17', '235/55R19',
       '225/50R1794Y', '43.66cm/17"LightAlloyWheelsJCWTrackSpoke,Black',
       '225/55R18', '225/60R19', '245/50R19', '245/45R18', '245/40R18',
       '255/50R19', '275/40R19', '265/60R18', '265/50R20', '295/45R20',
       '255/35R18', '265/35ZR19', '235/40R18', '275/55R19', '295/40R21',
       '235/60VR18', '265/35R19', '(245/50R18', '275/35R20', '275/50R20',
       '245/45R19', '285/40ZR19', '255/60ZR18', '285/45R21', '255/45R19',
       '285/35ZR20', '245/40R21', '245/40R20', '285/60R18', '235/35R19',
       '295/35R19', '315/40ZR21', '275/40R20', '295/35ZR20', '305/30ZR20',
       '285/45R20', '285/40R20', '265/45R20', '285/30ZR20', 'R12',
       '145/80R12', '145/80R13', '185/80R14', '205/65R15', '195/65R15',
       '215/60R16', '195/65R15Steel', '195/60R16', '195/60R15',
       '215/75R15', '185/75R16', '205/60R16', '205/50R17', '205/65R16',
       '185/65R16', '165/60/R14', '235/70R16', 'P235/70R16', '245/70R16',
       '215/70R15', '21565R16', '215/60R17', 'P215/75R15', '245/75R16',
       'P235/65R17', '235/65R17', 'P235/65R15', 'P235/60R18', '265/60R16',
       '215/55R16', '215/50R17', '115/90R17', '265/65R17', '265/65R18',
       '225/55R17', '225/50R18', '225/40R18', '225/50R17', '245/45R17',
       '245/35R17', '285/35R18', '245/75R15', '255/45R17', '255/335R19',
       '255/55R19', '255/60R19', '155/85R18', '235/65R20', '275/45R21',
       '255/50', '275/50ZR19', '315/35ZR21', '275/40R22', '285/35R20',
       '295/30ZR20', '325/30R20', '305/30R20', '305/35R19', '335/30',
       '255/35R19', '355/25ZR21', '145R12LT6PR', '205/60/R16', 'R16',
       '115/90R16', '205/55', '205/55R18', '245/35R19', '235/45R18',
       '225/60R18', '245/45R20', '255/50R20', '275/40R18', '305/30ZR21',
       '255/35R20', '295/35/20', '295/35R20', '315/35ZR20', '215/70',
       '235/50R18', '235/45R19', '235/45R18TubelessRadials', '245/40',
       '255/55R18', '275/35R19', '305/35ZR20', '155/65R14',
       '195R15LT,8PRRadial', '255/55R20', '245/45RF20', '305/40ZR20',
       '185/55R16'], dtype=object)
```

```
# Split and extract information from 'Rear_Tyre_&_Rim' column, and clean relevant columns
car[['R_Tire_Width', 'R_Tire']] = car['Rear_Tyre_&_Rim'].str.split('/', n=1, expand=True)
car[['R_Tire_Aspect_Ratio', 'R_Tire_Diameter']] = car['R_Tire'].str.split('R', n=1, expand=True)

r_drop = ['R_Tire']
car.drop(columns=r_drop, inplace=True)                # Drop the original 'R_Tire' column

# Clean and replace specific values in the extracted columns
car['R_Tire_Width'] = car['R_Tire_Width'].replace({'155R13LT':'155','(245':'245','R12':np.nan,'R16':np.nan,'43.66cm':
car['R_Tire_Aspect_Ratio'] = car['R_Tire_Aspect_Ratio'].replace({'17"LightAlloyWheelsJCWTrackSpoke,Black':'43','35/20
car['R_Tire_Diameter'] = car['R_Tire_Diameter'].replace({None:np.nan,'1794Y':np.nan,'15Steel':'15','18TubelessRadials

# Convert the cleaned columns to float type
car['R_Tire_Width'] = car['R_Tire_Width'].astype(float)
car['R_Tire_Aspect_Ratio'] = car['R_Tire_Aspect_Ratio'].astype(float)
car['R_Tire_Diameter'] = car['R_Tire_Diameter'].astype(float)
```

```
car['R_Tire_Width'].unique()
```

```
array([155.,  80., 165.,  nan, 175., 185., 205., 195., 225., 265., 245.,
       255., 215., 235., 295., 275., 285., 315., 305., 145., 115., 325.,
       335., 355.])
```

```
car['R_Tire_Aspect_Ratio'].unique()
```

```
array([65., nan, 80., 70., 60., 55., 45., 30., 50., 43., 40., 35., 75.,
       90., 85., 25.])
```

```
car['R_Tire_Diameter'].unique()
```

```
array([12., 13., 14., nan, 15., 16., 17., 18., 20., 19., 21., 22.])
```

```
# Reorder and rename columns in 'car' DataFrame for better organization
width = car.columns.get_loc('R_Tire_Width')
aspect = car.columns.get_loc('R_Tire_Aspect_Ratio')
diameter = car.columns.get_loc('R_Tire_Diameter')
rear = car.columns.get_loc('Rear_Tyre_&_Rim')

# Move the columns to the desired positions
column_to_move = car.pop('R_Tire_Width')
car.insert(rear, 'R_Tire_Width', column_to_move)

column_to_move = car.pop('R_Tire_Aspect_Ratio')
car.insert(rear, 'R_Tire_Aspect_Ratio', column_to_move)

column_to_move = car.pop('R_Tire_Diameter')
car.insert(rear, 'R_Tire_Diameter', column_to_move)

# Drop the original 'Rear_Tyre_&_Rim' column and rename the reordered columns
car.drop(columns=['Rear_Tyre_&_Rim'], inplace=True)
car.rename(columns={'R_Tire_Width': 'R_Tire_Width_mm', 'R_Tire_Diameter': 'R_Tire_Diameter_inch'}, inplace=True)
```

13. Processing Power and Torque Datasets

Here, we show the first five rows of the 'Power' column in the 'car' DataFrame to provide a glimpse of the data

```
car['Power'].head()
```

```
0    38PS@5500rpm
1    38PS@5500rpm
2    38PS@5500rpm
3    38PS@5500rpm
4    38PS@5500rpm
Name: Power, dtype: object
```

```
car[['Power_PS', 'Power_RPM']] = car['Power'].str.split('@', n=1, expand=True)
car[['Power_PS', 'Power_RPM']].head()
```

	Power_PS	Power_RPM
0	38PS	5500rpm
1	38PS	5500rpm
2	38PS	5500rpm
3	38PS	5500rpm
4	38PS	5500rpm

```
# Clean and convert 'Power_PS' column to float type
car['Power_PS'] = car['Power_PS'].replace(r'[A-Za-z]', '', regex=True)
car['Power_PS'] = car['Power_PS'].astype(float)

# Display unique values in the cleaned 'Power_PS' column of the 'car' DataFrame
car['Power_PS'].unique()
```

```
array([  38.  ,   54.  ,   68.  ,   73.  ,   74.  ,   62.  ,   59.  ,
         78.  ,   69.  ,   86.  ,   83.  ,   72.  ,   76.6 ,   64.  ,
         80.  ,   75.  ,   90.  ,   81.8 ,   70.  ,   71.  ,   84.  ,
        100.  ,   76.  ,  105.  ,  110.  ,   84.3 ,   96.  ,   65.  ,
         93.  ,  140.  ,   89.  ,   82.  ,  173.  ,  177.  ,  178.  ,
```

```
       141.  ,  154.  ,  120.  ,  180.  ,  143.  ,  150.  ,  122.  ,
       136.  ,  183.  ,  148.  ,  250.  ,  116.  ,  192.  ,  190.  ,
       231.  ,  247.  ,  245.  ,  248.  ,  179.  ,  333.  ,  265.  ,
       340.  ,  401.  ,  240.  ,  468.  ,  286.  ,  334.  ,  410.  ,
       300.  ,  258.  ,  557.  ,  262.  ,  450.  ,  609.  ,  275.  ,
       350.  ,  430.  ,  585.  ,  289.  ,  460.  ,  299.  ,  630.  ,
       453.  ,  550.  ,  261.  ,  455.  ,  610.  ,  650.  ,  590.  ,
       560.  ,  608.  ,  605.  ,  563.  ,  570.  ,  625.  ,  512.  ,
      1479.  , 1600.  ,   13.  ,   12.  ,   48.  ,   47.  ,   67.  ,
        77.  ,  101.4 ,  117.  ,  118.  ,  114.  ,   63.  ,   63.9 ,
       123.  ,  106.  ,   85.  ,  128.  ,  109.  ,  115.  ,   25.8 ,
        41.5 ,  119.  ,  104.  ,  121.  ,  155.  ,   41.  ,  156.  ,
       166.  ,  164.  ,  174.  ,  162.  ,  163.  ,   88.4 ,  142.76,
       200.  ,  160.  ,  187.  ,  372.  ,  181.  ,  184.  ,  268.7 ,
       549.  ,  233.  ,  188.  ,  268.  ,  197.  ,  249.  ,  252.  ,
       194.  ,  225.  ,  407.  ,  320.  ,  235.  ,  567.  ,  296.  ,
       335.  ,  518.  ,  254.  ,  274.  ,  302.  ,  558.  ,  510.  ,
       476.  ,  639.  ,  580.  ,  571.  ,  602.  ,  631.  ,  670.  ,
       681.  ,  700.  ,  740.  ,   34.7 ,   99.  ,  101.  ,  103.  ,
        95.  ,  102.  ,  168.  ,  129.  ,  152.  ,  185.  ,  241.  ,
       503.  ,   98.  ,  176.  ,  330.  ,  431.  ,  505.  ,  789.  ,
        94.  ,  145.  ,  126.  ,  134.  ,  167.  ,  360.  ,  170.  ,
       375.  ,  349.  ,  495.  ,  297.  ,  380.  ,  669.  ,   73.5 ,
       394.  ,  107.  ,  507.  ,  616.  ,  202.  ])
```

```
# Clean and convert 'Power_RPM' column to float type
car['Power_RPM'] = car['Power_RPM'].replace(r'[A-Za-z]', '', regex=True)

# Replace specific values and convert to float type
car.loc[car['Power_RPM'].str.contains('-4', case=False, na=False), 'Power_RPM'] = '4000'
car.loc[car['Power_RPM'].str.contains('-6', case=False, na=False), 'Power_RPM'] = '6000'
car.loc[car['Power_RPM'].str.contains('-7', case=False, na=False), 'Power_RPM'] = '7000'

# Display unique values in the cleaned 'Power_RPM' column of the 'car' DataFrame
car['Power_RPM'] = car['Power_RPM'].astype(float)
car['Power_RPM'].unique()
```

```
array([ 5500.,  5678.,  6000.,  6200.,  5000., 62050.,  5800.,  4000.,
        5600.,  3800.,  4200.,  4400.,  3750.,  6500.,  3700.,  6300.,
        3600.,   400.,    nan,  5100.,  5200.,  4250.,  6550.,  6250.,
        6350.,  7000.,  5250.,  3400.,  5750.,  1200.,  6600.,  4800.,
        8250.,  7500.,  9000.,  5350.,  6700.,  3200.,  6400.,  3500.,
        4500.,   150.,  3000.,  5700.,   500.,  5300.,  8000.,  8400.,
        3850.,  3900.,  5850.,  7300.,  6800.,  6650.,  8500.,  5400.,
        6100.])
```

```
# Reorder and rename columns in 'car' DataFrame for better organization
index_of_Power_RPM = car.columns.get_loc('Power_RPM')
index_of_Power_PS = car.columns.get_loc('Power_PS')
index_of_Power = car.columns.get_loc('Power')

# Move the columns to the desired positions
column_to_move = car.pop('Power_PS')
car.insert(index_of_Power, 'Power_PS', column_to_move)

column_to_move = car.pop('Power_RPM')
car.insert(index_of_Power, 'Power_RPM', column_to_move)

# Drop the original 'Power' column
car.drop(columns=['Power'], inplace=True)

# Display the first five rows of 'Power_PS' and 'Power_RPM' columns in 'car' DataFrame
car[['Power_PS', 'Power_RPM']].head()
```

	Power_PS	Power_RPM
0	38.000	5500.000
1	38.000	5500.000
2	38.000	5500.000
3	38.000	5500.000
4	38.000	5500.000

```
car['Torque'].head()
```

```
0    51Nm@4000rpm
1    51Nm@4000rpm
```

```
2    51Nm@4000rpm
3    51Nm@4000rpm
4    51Nm@4000rpm
Name: Torque, dtype: object
```

```
# Split and extract information from 'Torque' column into 'Torque_Nm' and 'Torque_RPM' columns
car[['Torque_Nm', 'Torque_RPM']] = car['Torque'].str.split('@', n=1, expand=True)
car[['Torque_Nm', 'Torque_RPM']].head()
```

	Torque_Nm	Torque_RPM
0	51Nm	4000rpm
1	51Nm	4000rpm
2	51Nm	4000rpm
3	51Nm	4000rpm
4	51Nm	4000rpm

```
# Clean and convert 'Torque_Nm' column to float type
car['Torque_Nm'] = car['Torque_Nm'].replace(r'[A-Za-z]', '', regex=True)
car['Torque_Nm'] = car['Torque_Nm'].astype(float)

# Display unique values in the cleaned 'Torque_Nm' column of the 'car' DataFrame
car['Torque_Nm'].unique()
```

```
array([  51.  ,   72.  ,   91.  ,  101.  ,   85.  ,   90.  ,   78.  ,
        104.  ,   99.04,   10.1 ,    8.3 ,  114.  ,  113.  ,   96.  ,
        103.9 ,  152.  ,  183.  ,  170.  ,  190.  ,  140.  ,  115.  ,
        220.  ,   95.  ,  172.  ,  230.  ,  175.  ,  250.  ,  215.  ,
        120.  ,  132.  ,  160.  ,  209.  ,  210.  ,  110.  ,  200.  ,
        350.  ,  380.  ,  420.  ,  340.  ,  189.  ,  300.  ,  320.  ,
        270.  ,  280.  ,  400.  ,  365.  ,  430.  ,  370.  ,  500.  ,
        620.  ,  450.  ,  515.  ,  570.  ,  624.  ,  347.  ,  480.  ,
        550.  ,  619.  ,  760.  ,  410.  ,  335.  ,  600.  ,  850.  ,
        900.  ,  580.  ,  520.  , 1000.  ,  700.  ,  770.  ,  650.  ,
        560.  ,  720.  ,  755.  ,  540.  ,  780.  ,  800.  , 1020.  ,
       1600.  , 1479.  ,   18.9 ,   16.1 ,   69.  ,   16.3 ,  240.  ,
        103.  ,  112.7 , 1712.  ,   22.4 ,  260.  ,  207.  ,  195.  ,
          nan,  150.  ,  205.  ,  142.  ,  245.  ,  151.  ,  155.  ,
        133.  ,  153.  ,   70.  ,  105.  ,  145.  ,  247.  ,  321.  ,
        138.  ,  330.  ,  360.  ,  343.  ,  173.  ,  174.  ,  353.  ,
        440.  ,  470.  ,  385.  ,  640.  ,  625.  ,  740.  ,  689.  ,
        510.  ,  630.  ,  697.  ,  690.  ,   59.  ,  197.  ,  134.  ,
        130.  ,  180.  ,  225.  ,  218.  ,  144.  ,  242.  ,  248.  ,
        196.  ,  192.  ,  221.  ,  369.  ,  530.  ,  637.  ,  685.  ,
        718.  ,  212.  ,  222.  ,  475.  ,  680.  ,  750.  ,  546.  ,
       1130.  ,  395.  ,  660.  ,  441.  ])
```

```
# Clean and convert 'Torque_RPM' column to float type
car['Torque_RPM'] = car['Torque_RPM'].replace(r'[A-Za-z]', '', regex=True)

# Replace specific values and convert to float type
car.loc[car['Torque_RPM'].str.contains('-1', case=False, na=False), 'Torque_RPM'] = '1000'
car.loc[car['Torque_RPM'].str.contains('-2', case=False, na=False), 'Torque_RPM'] = '2000'
car.loc[car['Torque_RPM'].str.contains('-3|- 3', case=False, na=False), 'Torque_RPM'] = '3000'
car.loc[car['Torque_RPM'].str.contains('-4|- 4', case=False, na=False), 'Torque_RPM'] = '4000'
car.loc[car['Torque_RPM'].str.contains('-5|- 5', case=False, na=False), 'Torque_RPM'] = '5000'
car.loc[car['Torque_RPM'].str.contains('-6', case=False, na=False), 'Torque_RPM'] = '6000'

car['Torque_RPM'] = car['Torque_RPM'].astype(float)

# Display unique values in the cleaned 'Torque_RPM' column of the 'car' DataFrame
car['Torque_RPM'].unique()
```

```
array([4000., 4386., 4250., 3000., 3500., 4500., 3300., 4200., 2500.,
       2250., 2000., 3100., 6000., 1500., 4400., 4800., 1750., 4300.,
       5000.,   nan, 1250., 1450., 4100., 5250., 4600., 1370., 4750.,
       5100., 6500., 1800., 1350., 1700., 1600., 6700., 2200., 4850.,
       1900., 3800., 1000., 1550., 2400., 1740., 5300., 5500., 5506.,
       3750., 3600., 3200., 7000., 1360.])
```

```
# Reorder and rename columns in 'car' DataFrame for better organization
index_of_Torque_RPM = car.columns.get_loc('Torque_RPM')
index_of_Torque_Nm = car.columns.get_loc('Torque_Nm')
index_of_Torque = car.columns.get_loc('Torque')

# Move the columns to the desired positions
column_to_move = car.pop('Torque_Nm')
car.insert(index_of_Torque, 'Torque_Nm', column_to_move)

column_to_move = car.pop('Torque_RPM')
car.insert(index_of_Torque, 'Torque_RPM', column_to_move)

# Drop the original 'Torque' column
car.drop(columns=['Torque'], inplace=True)

# Display the first five rows of 'Torque_Nm' and 'Torque_RPM' columns in 'car' DataFrame
car[['Torque_Nm', 'Torque_RPM']].head()
```

	Torque_Nm	Torque_RPM
0	51.000	4000.000
1	51.000	4000.000
2	51.000	4000.000
3	51.000	4000.000
4	51.000	4000.000

14. Processing Power_Steering, Power_Windows, Power_Seats and Keyless_Entry Datasets

This code snippet displays the unique values in the 'Power_Steering' column of the 'car' DataFrame.

```
car['Power_Steering'].unique()
```

```
array(['Electric Power', nan, 'Yes', 'Hydraulic Power',
       'Electric Power, Hydraulic Power', 'Electro-Hydraulic'],
      dtype=object)
```

```
# Replace values in the 'Power_Steering' column for better categorization
car['Power_Steering'] = car['Power_Steering'].replace({'Electric Power, Hydraulic Power':'Electro-Hydraulic','Yes':'l

# Display unique values in the modified 'Power_Steering' column of the 'car' DataFrame
car['Power_Steering'].unique()
```

```
array(['Electric Power', 'Non Powered', 'Undefined Powered',
       'Hydraulic Power', 'Electro-Hydraulic'], dtype=object)
```

```
car['Power_Windows'].unique()
```

```
array(['Only Front Windows', nan, 'All Windows'], dtype=object)
```

```
# Replace values in the 'Power_Windows' column for better categorization
car['Power_Windows'] = car['Power_Windows'].replace({'Only Front Windows':'Front Windows',np.nan:'Non Powered'})

# Display unique values in the modified 'Power_Windows' column of the 'car' DataFrame
car['Power_Windows'].unique()
```

```
array(['Front Windows', 'Non Powered', 'All Windows'], dtype=object)
```

```
car['Power_Seats'].unique()
```

```
array([nan, 'Yes', 'Power seats', 'Yes, with memory'], dtype=object)
```

```
# Modify values in the 'Power_Seats' column for better categorization
car.loc[car['Power_Seats'].str.contains('Yes|Power', case=False, na=False), 'Power_Seats'] = 'Powered Seats'
car['Power_Seats'] = car['Power_Seats'].replace(np.nan,'Non Powered')

# Display unique values in the modified 'Power_Seats' column of the 'car' DataFrame
car['Power_Seats'].unique()
```

```
array(['Non Powered', 'Powered Seats'], dtype=object)
```

```
car['Keyless_Entry'].unique()
```

```
array(['Remote', nan, 'Yes', 'Smart Key', 'Remote, Smart Key',
       'Smart Key, Remote'], dtype=object)
```

```
# Modify values in the 'Keyless_Entry' column for better categorization
car.loc[car['Keyless_Entry'].str.contains('Yes|Key|Remote', case=False, na=False), 'Keyless_Entry'] = 'Yes'
car['Keyless_Entry'] = car['Keyless_Entry'].replace(np.nan,'No')

# Display unique values in the modified 'Keyless_Entry' column of the 'car' DataFrame
car['Keyless_Entry'].unique()
```

```
array(['Yes', 'No'], dtype=object)
```

15. Processing Other Features - Datasets

Now, we are displaying the unique values in the 'Odometer' column of the 'car' DataFrame.

```
car['Odometer'].unique()
```

```
array(['Digital', 'Analog', nan, 'Digital, Analog', 'Yes'], dtype=object)
```

```
# Modify values in the 'Odometer' column for better categorization
car['Odometer'] = car['Odometer'].replace({'Digital, Analog':'Digi-Analog','Yes':'Unspecified'})

# Display unique values in the modified 'Odometer' column of the 'car' DataFrame
car['Odometer'].unique()
```

```
array(['Digital', 'Analog', nan, 'Digi-Analog', 'Unspecified'],
      dtype=object)
```

```
car['Speedometer'].unique()
```

```
array(['Analog', 'Digital', 'Analog, Digital', 'Digital, Analog', nan,
       'Yes'], dtype=object)
```

```
# Modify values in the 'Speedometer' column for better categorization
car['Speedometer'] = car['Speedometer'].replace({'Analog, Digital':'Digi-Analog','Digital, Analog':'Digi-Analog','Yes
```

```
# Display unique values in the modified 'Speedometer' column of the 'car' DataFrame
car['Speedometer'].unique()
```

```
array(['Analog', 'Digital', 'Digi-Analog', nan, 'Unspecified'],
      dtype=object)
```

```
car['Tachometer'].unique()
```

```
array(['Not on offer', 'Digital', 'Analog', 'Analog, Digital',
       'Digital, Analog', 'Yes', nan, 'Analog, Not on offer'],
      dtype=object)
```

```
# Update values in the 'Tachometer' column for better categorization
car.loc[car['Tachometer'].str.contains('Yes|Analog|Digital', case=False, na=False), 'Tachometer'] = 'Yes'
car.loc[car['Tachometer'].str.contains('Not', case=False, na=False), 'Tachometer'] = 'No'
car['Tachometer'] = car['Tachometer'].replace(np.nan,'No')

# Display unique values in the modified 'Tachometer' column of the 'car' DataFrame
car['Tachometer'].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

```
car['Tripmeter'].value_counts()
```

```
Yes     978
2       197
1        26
1, 2      7
Name: Tripmeter, dtype: int64
```

```
# Update 'Tripmeter' column: If not null, set to 'Yes'; otherwise, set to 'No'
car['Tripmeter'] = np.where(car['Tripmeter'].notnull(), 'Yes', car['Tripmeter'])

# Replace remaining NaN values with 'No' in the 'Tripmeter' column
car['Tripmeter'] = car['Tripmeter'].replace(np.nan, 'No')

# Display updated value counts for the 'Tripmeter' column in the 'car' DataFrame
car['Tripmeter'].value_counts()
```

```
Yes    1208
No       59
Name: Tripmeter, dtype: int64
```

```
car['Wheelbase'].sample(5)
```

```
615    2258 mm
534    2680 mm
560    2519 mm
418    2450 mm
956    2600 mm
Name: Wheelbase, dtype: object
```

```
# Clean and convert the 'Wheelbase' column to numerical values in millimeters
car['Wheelbase'] = car['Wheelbase'].replace(r'\D', '', regex=True)
car.rename(columns={'Wheelbase': 'Wheelbase_mm'}, inplace=True)
car['Wheelbase_mm'] = car['Wheelbase_mm'].astype(float)
```

```
car['Wheelbase_mm'].unique()
```

```
array([2230., 2348., 2422., 2442., 2350., 2360., 2450., 2425., 2400.,
       2435., 2636., 2420., 2460., 2470., 2430., 2525., 2570., 2469.,
       2490., 2550., 2630., 2510., 2520., 2530., 2845., 2865., 2677.,
       2660., 2841., 2637., 2699., 2700., 2603.,   nan, 2702., 2670.,
       2820., 2495., 2835., 2960., 3157., 2595., 2912., 2810., 2741.,
       2941., 2811., 2575., 2720., 2915., 2760., 2693., 2475., 3075.,
       2790., 3210., 3110., 2994., 3035., 3165., 2945., 3004., 2850.,
       2890., 3128., 2942., 3171., 2870., 3365., 2950., 3100., 2650.,
       3003., 2746., 2669., 2992., 2995., 2600., 3295., 3112., 3266.,
       3320., 2710., 1925., 2380., 2385., 2501., 2541., 2375., 2500.,
       2498., 2680., 2794., 2519., 2579., 2673., 2553., 2258., 2662.,
       2750., 2740., 3040., 2552., 2585., 3060., 2647., 2745., 2791.,
       2776., 2774., 2807., 2681., 3079., 2874., 2864., 3008., 2840.,
       3430., 3200., 2984., 2923., 2895., 2998., 2922., 3120., 2951.,
       2620., 2990., 3570., 1840., 2555., 2610., 2786., 2646., 3105.,
       2812., 2780., 2704., 2989., 2590., 3095., 2968., 2855., 3070.,
       2622., 2964., 3488., 2825., 3125., 3820., 2688., 2873., 2800.,
       3066.])
```

```
car['Basic_Warranty'].unique()
```

```
array(['2 years /75000 Kms (years/distance whichever comes first)',
       '2 years / Unlimited Kms',
       '2 Years / 50,000 Kms (Whichever comes earlier)', nan,
       '2 years /40000 Kms (years/distance whichever comes first)',
       '2 Years / 75,000 KM (whichever is earlier)',
       '3 years /100000 Kms (years/distance whichever comes first)',
       '24 months /50000 Kms (whichever comes first)',
       '2 Years / 100,000 Kms (whichever comes first)',
       '3 Years/1,00,000 Kms (Whichever comes earlier)',
       '2 Years / 40,000 Kms (Whichever comes earlier)',
       '3 years / Unlimited Kms',
       '3 Years / 100000 km (whichever comes first)',
       '2 Years Unlimited Kilometres',
       '3rd years /Unlimited Kms (years/distance whichever comes first)',
       '3 years/40,000 km', '2 Years Warranty',
       '4 years / 80000 Kms (years/distance whichever comes first)',
       '4 years /Unlimited Kms (years/distance whichever comes first)',
       '2 years /1,00,000 Kms (years/distance whichever comes first)',
       '3 years/1 lakh kms (whichever is earlier)',
       '3 years / 100,000 Kms',
       '2 years /50000 Kms (years/distance whichever comes first)',
       '3 years 60000 kms', '1 Year /Unlimited KMs', '3 Years Warranty',
       '8 yrs/1.6L kmBattery Warranty',
       '3 years / 100,000 Km of warranty**',
       '2 years without mileage limit'], dtype=object)
```

```
# Update the 'Basic_Warranty' column in the 'car' DataFrame based on the specified conditions
car.loc[car['Basic_Warranty'].str.contains('1 Year', na=False), 'Basic_Warranty'] = '1'
car.loc[car['Basic_Warranty'].str.contains('2 years|2 Years|24 months', na=False), 'Basic_Warranty'] = '2'
car.loc[car['Basic_Warranty'].str.contains('3 years|3 Years|3rd years', na=False), 'Basic_Warranty'] = '3'
car.loc[car['Basic_Warranty'].str.contains('4 years', na=False), 'Basic_Warranty'] = '4'
car.loc[car['Basic_Warranty'].str.contains('8 yrs', na=False), 'Basic_Warranty'] = '8'

# Rename the 'Basic_Warranty' column to 'Basic_Warranty_years' and convert to float
car.rename(columns={'Basic_Warranty': 'Basic_Warranty_years'}, inplace=True)
car['Basic_Warranty_years'] = car['Basic_Warranty_years'].astype(float)
```

```
car['Basic_Warranty_years'].value_counts()
```

```
2.000    486
3.000    323
4.000     12
1.000      3
8.000      3
Name: Basic_Warranty_years, dtype: int64
```

```
car['Boot_Space'].unique()
```

```
array(['110 litres', '94 litres', '222 litres', '300 litres',
       '400 litres', '250 litres', '177 litres', '265 litres',
       '235 litres', nan, '242 litres', '240 litres', '260 litres',
       '625 litres', '454 litres', '251 litres', '210 litres',
       '407 litres', '320 litres', '316 litres', '285 litres',
       '280 litres', '378 litres', '257 litres', '330 litres',
       '592 litres', '339 litres', '354 litres', '438 litres',
       '615 litres', '522 litres', '150 litres', '425 litres',
       '488 litres', '470 litres', '460 litres', '278 litres',
       '215 litres', '432 litres', '360 litres', '480 litres',
       '455 litres', '505 litres', '520 litres', '560 litres',
       '550 litres', '650 litres', '1025 litres', '540 litres',
       '465 litres', '515 litres', '500 litres', '605 litres',
       '530 litres', '510 litres', '580 litres', '261 litres',
       '133 litres', '616 litres', '358 litres', '430 litres',
       '230 litres', '490 litres', '443 litres', '395 litres',
       '20 litres', '256 litres', '243 litres', '268 litres',
       '345 litres', '419 litres', '390 litres', '420 litres',
       '350 litres', '1702 litres', '328 litres', '352 litres',
       '475 litres', '259 litres', '207 litres', '759 litres',
       '494 litres', '135 litres', '255 litres', '223 litres',
       '600 litres', '209 litres', '384 litres', '96 litres',
       '128 litres', '448 litres', '324 litres', '296 litres',
       '211 litres', '270 litres', '380 litres', '495 litres',
       '1400 litres', '281 litres', '295 litres', '909 litres',
       '1761 litres', '478 litres', '645 litres', '745 litres',
       '770 litres', '621 litres', '535 litres', '173 litres',
       '165 litres', '70 litres', '450 litres', '525 litres',
       '347 litres',
       '209(All3RowsUp).550(3rdRowFolded)&803(2ndRowand3rdRowFolded) litres',
       '363 litres', '375 litres', '392 litres', '433 litres',
       '981 litres', '586 litres', '341 litres', '435 litres',
       '825 litres', '326 litres', '368 litres', '132 litres',
       '249 litres', '421 litres', '610 litres', '200.5 litres',
       '310 litres', '412 litres', '54 litres', '258 litres',
       '476 litres', '590 litres', '598 litres', '1050 litres'],
      dtype=object)
```

```
# Clean and convert 'Boot_Space' column to float type
car['Boot_Space'] = car['Boot_Space'].replace('209(All3RowsUp).550(3rdRowFolded)&803(2ndRowand3rdRowFolded) litres',r
car['Boot_Space'] = car['Boot_Space'].replace(r'\D', '', regex=True)
car.rename(columns={'Boot_Space': 'Boot_Space_litres'}, inplace=True)
car['Boot_Space_litres'] = car['Boot_Space_litres'].astype(float)
```

```
car['Boot_Space_litres'].unique()                # Display unique values in the 'Boot_Space_litres' column of the 'c
```

```
array([ 110.,   94.,  222.,  300.,  400.,  250.,  177.,  265.,  235.,
         nan,  242.,  240.,  260.,  625.,  454.,  251.,  210.,  407.,
        320.,  316.,  285.,  280.,  378.,  257.,  330.,  592.,  339.,
        354.,  438.,  615.,  522.,  150.,  425.,  488.,  470.,  460.,
        278.,  215.,  432.,  360.,  480.,  455.,  505.,  520.,  560.,
        550.,  650., 1025.,  540.,  465.,  515.,  500.,  605.,  530.,
        510.,  580.,  261.,  133.,  616.,  358.,  430.,  230.,  490.,
```

```
       443.,  395.,   20.,  256.,  243.,  268.,  345.,  419.,  390.,
       420.,  350., 1702.,  328.,  352.,  475.,  259.,  207.,  759.,
       494.,  135.,  255.,  223.,  600.,  209.,  384.,   96.,  128.,
       448.,  324.,  296.,  211.,  270.,  380.,  495., 1400.,  281.,
       295.,  909., 1761.,  478.,  645.,  745.,  770.,  621.,  535.,
       173.,  165.,   70.,  450.,  525.,  347.,  363.,  375.,  392.,
       433.,  981.,  586.,  341.,  435.,  825.,  326.,  368.,  132.,
       249.,  421.,  610., 2005.,  310.,  412.,   54.,  258.,  476.,
       590.,  598., 1050.])
```

```
car.drop(columns='Extended_Warranty', inplace=True)            # Drop the 'Extended_Warranty' column from the 'car'
```

```
car['Minimum_Turning_Radius'].unique()       # Display unique values in the 'Minimum_Turning_Radius' column of the '
```

```
array(['4 meter', '4.7 meter', nan, '4.5 meter', '4.6 meter', '4.9 meter',
       '4.8 meter', '4.65 meter', '5.1 meter', '5.2 meter', '4.97 meter',
       '5 meter', '5.25 meter', '6.2 meter', '5.5 meter', '5.75 meter',
       '5.4 meter', '11.1 meter', '11 meter', '6 meter', '5.9 meter',
       '5.7 meter', '11.22 meter', '11.61 meter', '5.95 meter',
       '11.6 meter', '5.8 meter', '5.6 meter', '7.5 meter', '6.15 meter',
       '6.45 meter', '13.5 meter', '6.35 meter', '10.7 meter',
       '11.8 meter', '5.3 meter', '6.9 meter', '3.5 meter', '5.05 meter',
       '5.55 meter', '4.35 meter', '6.25 meter', '5.65 meter',
       '5.35 meter', '11.7 meter', '10.8 meter', '6.1 meter', '6.3 meter',
       '12.1 meter', '5.85 meter', '11.9 meter', '6.55 meter',
       '4.1 meter', '5.45 meter', '5.61 meter', '5.64 meter',
       '11.87 meter', '11.84 meter', '11.4 meter', '10.66 meter',
       '5.33 meter', '6750 meter'], dtype=object)
```

```
# Removing non-digit characters from the 'Minimum_Turning_Radius' column
car['Minimum_Turning_Radius'] = car['Minimum_Turning_Radius'].replace(r'\D', '', regex=True)

# Renaming the column to provide a more descriptive name
car.rename(columns={'Minimum_Turning_Radius': 'Minimum_Turning_Radius_meter'}, inplace=True)

# Converting the values in 'Minimum_Turning_Radius_meter' to float data type
car['Minimum_Turning_Radius_meter'] = car['Minimum_Turning_Radius_meter'].astype(float)
```

Handling Null Values

Dealing with the NULL values present over the dataset.

Now, we produce summary statistics for the numerical columns in the updated 'car' DataFrame,

```
# Generate descriptive statistics for numerical columns in the updated 'car' DataFrame
car.describe()
```

	Ex-Showroom_Price_INR	Displacement_cc	Cylinders	Valves_Per_Cylinder	Fu
count	1267.000	1255.000	1201.000	1165.000	
mean	4617330.301	1856.508	4.384	3.978	
std	12187926.657	1066.613	1.667	0.837	
min	236447.000	72.000	2.000	1.000	
25%	740553.500	1198.000	4.000	4.000	
50%	1055000.000	1497.000	4.000	4.000	
75%	2991400.000	1998.000	4.000	4.000	
max	212155397.000	7993.000	16.000	16.000	

```
# Display the data type of each column in the 'car' DataFrame
for i in car.columns:
  print(i ,'-------', car[i].dtype)
```

```
Door_Pockets ------- object
Engine_Malfunction_Light ------- object
FM_Radio ------- object
Fuel-lid_Opener ------- object
Fuel_Gauge ------- object
Handbrake ------- object
Instrument_Console ------- object
Low_Fuel_Warning ------- object
Minimum_Turning_Radius_meter ------- float64
Multifunction_Display ------- object
Sun_Visor ------- object
Third_Row_AC_Vents ------- object
Ventilation_System ------- object
Auto-Dimming_Rear-View_Mirror ------- object
Hill_Assist ------- object
Gear_Indicator ------- object
Engine_Immobilizer ------- object
Seat_Back_Pockets ------- object
ABS_(Anti-lock_Braking_System) ------- object
Headlight_Reminder ------- object
Adjustable_Headrests ------- object
Gross_Vehicle_Weight ------- object
Airbags ------- object
Door_Ajar_Warning ------- object
EBD_(Electronic_Brake-force_Distribution) ------- object
Fasten_Seat_Belt_Warning ------- object
Gear_Shift_Reminder ------- object
Number_of_Airbags ------- float64
Adjustable_Steering_Column ------- object
Parking_Assistance ------- object
Key_Off_Reminder ------- object
USB_Compatibility ------- object
Cigarette_Lighter ------- object
Infotainment_Screen ------- object
Multifunction_Steering_Wheel ------- object
Average_Speed ------- object
EBA_(Electronic_Brake_Assist) ------- object
Seat_Height_Adjustment ------- object
Navigation_System ------- object
Second_Row_AC_Vents ------- object
Rear_Center_Armrest ------- object
iPod_Compatibility ------- object
ESP_(Electronic_Stability_Program) ------- object
Cooled_Glove_Box ------- object
Turbocharger ------- object
ISOFIX_(Child-Seat_Mount) ------- object
Rain_Sensing_Wipers ------- object
Leather_Wrapped_Steering ------- object
Automatic_Headlamps ------- object
ASR_/_Traction_Control ------- object
Cruise_Control ------- object
```

```
car.isnull().sum().sum()          # Sum of all missing values in the 'car' DataFrame
```

```
26207
```

```
# Impute missing values with the median for numeric columns in the 'car' DataFrame
numeric=["int32","float64"]

for j in car.columns:
    if car[j].dtype in numeric:
        if car[j].isnull().sum()>0:
            car[j]=car[j].fillna(car[j].median())


# Display the count of missing values in numeric columns of the 'car' DataFrame
for j in car.columns:
    if car[j].dtype in numeric:
        print(j,'--------',car[j].isnull().sum())
```

```
Displacement_cc -------- 0
Cylinders -------- 0
Valves_Per_Cylinder -------- 0
Fuel_Tank_Capacity_litres -------- 0
Height_mm -------- 0
Length_mm -------- 0
Width_mm -------- 0
Doors -------- 0
ARAI_Certified_Mileage_kmpl -------- 0
Kerb_Weight_kg -------- 0
Gears -------- 0
Ground_Clearance_mm -------- 0
Front_Track_mm -------- 0
```

```
Rear_Track_mm -------- 0
F_Tire_Diameter_inch -------- 0
F_Tire_Aspect_Ratio -------- 0
F_Tire_Width_mm -------- 0
R_Tire_Diameter_inch -------- 0
R_Tire_Aspect_Ratio -------- 0
R_Tire_Width_mm -------- 0
Power_RPM -------- 0
Power_PS -------- 0
Torque_RPM -------- 0
Torque_Nm -------- 0
Seating_Capacity -------- 0
Wheelbase_mm -------- 0
Basic_Warranty_years -------- 0
Boot_Space_litres -------- 0
Minimum_Turning_Radius_meter -------- 0
Number_of_Airbags -------- 0
```

```
car.isnull().sum().sum()      # Sum of all remaining missing values in the 'car' DataFrame
```

```
22065
```

```
# Selecting a subset of vital columns from the 'car' DataFrame
vital = ['Make', 'Model', 'Variant', 'Ex-Showroom_Price_INR', 'Drivetrain', 'Cylinder_Configuration',
 'Emission_Norm', 'Engine_Location', 'Fuel_System', 'Fuel_Type', 'Body_Type', 'Front_Brakes', 'Rear_Brakes',
 'Front_Suspension', 'Rear_Suspension', 'Power_Steering', 'Power_Windows', 'Power_Seats', 'Odometer', 'Speedometer',
 'Seats_Material', 'Type']
```

```
# Fill missing values in vital columns with 'unspecified' in the 'car' DataFrame
for i in vital:
    car[i]=car[i].fillna('unspecified')
```

```
car.isnull().sum().sum()          # Sum of all remaining missing values in the 'car' DataFrame after filling with 'ur
```

```
21828
```

```
# Extract non-numeric features not included in the vital columns
features = []
for i in car.columns:
    if i not in vital:
        if car[i].dtype not in numeric:
            features.append(i)
```

```
features[:5]      # Display the first 5 non-numeric features
```

```
['Keyless_Entry',
 'Tachometer',
 'Tripmeter',
 'Start_/_Stop_Button',
 '12v_Power_Outlet']
```

```
# Convert non-numeric features to binary indicators ('Yes' or 'No') in the 'car' DataFrame
for k in features:
    car[k] = np.where(car[k].notnull(), 'Yes', car[k])
    car[k] = car[k].fillna('No')
```

```
car.isnull().sum().sum()  # Sum of all remaining missing values in the 'car' DataFrame after converting to binary inc
```

```
0
```

Viewing & Saving Clean Data

Viewing the final and cleaned data, saving it into .csv format

Here, the column names in the 'car' DataFrame are modified by replacing spaces with underscores and converting them to lowercase for consistency.

```
car.columns = [col.replace(' ', '_').lower() for col in car.columns]    # Rename columns by replacing spaces with unde
```

```
car.sample(5)                 # Display a random sample of 5 rows from the modified 'car' DataFrame
```

	make	model	variant	ex-showroom_price_inr	displacement_cc	cylinders	v
111	Tata	Bolt	Xms Petrol	614515	1193.000	4.000	
374	Maruti Suzuki	S-Presso	Vxi+	456000	998.000	3.000	
529	Fiat	Linea	Dynamic Multijet	932025	1248.000	4.000	
259	Skoda	Kodiaq	Style 2.0 Tdi 4X4 At	3299599	1968.000	4.000	
139	Hyundai	Aura	S 1.2 Cng Petrol (Cng +	728900	1197.000	4.000	

```
print(car.dtypes)       # Display data types of each column in the modified 'car' DataFrame
```

```
make                        object
model                       object
variant                     object
ex-showroom_price_inr        int64
displacement_cc            float64
                            ...
rain_sensing_wipers         object
leather_wrapped_steering    object
automatic_headlamps         object
asr_/_traction_control      object
cruise_control              object
Length: 117, dtype: object
```

```
print(car.isnull().sum().sum())              # Sum of all remaining missing values in the modified 'car' DataFrame
```

```
0
```

```
print(car.shape)            # Display the current shape (rows, columns) of the modified 'car' DataFrame
```

```
(1267, 117)
```

```
print(car.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1267 entries, 0 to 1275
Columns: 117 entries, make to cruise_control
dtypes: float64(30), int64(1), object(86)
memory usage: 1.2+ MB
None
```

```
car.to_csv('car_data_cleaned.csv', index=False)        # Save the modified 'car' DataFrame to a CSV file named 'car_
```

Statistics & Data Vizualization Data Analysis

1.1. Descriptive Numeric Analysis

This code generates descriptive statistics, such as mean, standard deviation, minimum, and maximum values, for numerical columns in the 'car' DataFrame.

```
# Generate descriptive statistics for numerical columns in the 'stores' DataFrame
car.describe()
```

	ex-showroom_price_inr	displacement_cc	cylinders	valves_per_cylinder	fu
count	1267.000	1267.000	1267.000	1267.000	
mean	4617330.301	1853.103	4.364	3.979	
std	12187926.657	1062.118	1.625	0.803	
min	236447.000	72.000	2.000	1.000	
25%	740553.500	1198.000	4.000	4.000	
50%	1055000.000	1497.000	4.000	4.000	
75%	2991400.000	1998.000	4.000	4.000	
max	212155397.000	7993.000	16.000	16.000	

Observation: Here we get a statistical data for all the numerics value such as the price, dimensions of body and tire, capacity, power and torque. 1.2. Descriptive Numeric Analysis

This code generates descriptive statistics for categorical columns in the 'car' DataFrame, including count, unique values, top value, and frequency.

```
# Generate descriptive statistics for categorical columns in the 'stores' DataFrame
car.describe(include=['object'])
```

	make	model	variant	drivetrain	cylinder_configuration	emission_no
count	1267	1267	1267	1267	1267	12
unique	39	263	1055	5	5	
top	Maruti Suzuki	Nexon	Lxi	FWD (Front Wheel Drive)	In-line	BS
freq	163	24	9	878	1068	8

Observation: This type of description shows the frequency of each column which is object, their frequency and the leader for every column.

2. Top Car Models

Here, we calculate the average ex-showroom price for each car model, group the data by make and model, and then display the top 5 car models based on the highest average ex-showroom price.

```
# Find and display the top 5 car models based on the ex-showroom price
top_car_models = car.groupby(['make', 'model'])['ex-showroom_price_inr'].mean().sort_values(ascending=False).head(5)
print("Top Car Models:\n", top_car_models)
```

```
Top Car Models:
 make         model
Bugatti      Chiron           202149167.000
Rolls-Royce  Phantom           85200000.000
             Drophead Coupe     83755383.000
             Phantom Coupe      77312661.000
             Cullinan           69500000.000
Name: ex-showroom_price_inr, dtype: float64
```

Observation: In India, the leading car Bugatti is a super sports car, and the other 4 are known for luxury. Maybe they were bought by Ambani, Adani and Tata XD

3. Cheapest Car Models

Here, we calculate the average ex-showroom price for each car model, group the data by make and model, and then display the 5 cheapest car models based on the lowest average ex-showroom price.

```
# Find and display the cheap 5 car models based on the ex-showroom price
cheapest_car_models = car.groupby(['make', 'model'])['ex-showroom_price_inr'].mean().sort_values(ascending=True).head
print("Top Car Models:\n", cheapest_car_models)
```

```
Top Car Models:
 make           model
Bajaj          Qute (Re60)     273000.000
Maruti Suzuki  Omni            283631.500
Tata           Nano Genx       291263.500
Maruti Suzuki  Alto 800 Tour   355397.000
Datsun         Redi-Go         358994.333
Name: ex-showroom_price_inr, dtype: float64
```

Observation: Bajaj Qute is a most pocket friendly car. Omni is best known to kidnap people (or ambulance) XP. Nano Genx and Datsun are now obsolete. However Alto is know popularly for affordable cab services.

4. Popular Fuel Types

In this step, we identify and display the top 5 most popular fuel types based on their count in the 'car' DataFrame.

```
# Find and display the most popular fuel types based on the count
popular_fuel_types = car['fuel_type'].value_counts().head(5).reset_index()
print("Popular Fuel Types:\n", popular_fuel_types)
```

```
Popular Fuel Types:
       index  fuel_type
0    Petrol        642
1    Diesel        574
2       CNG         16
3    Hybrid         15
4  Electric         14
```

Observation: Seems like people of India sticks with more Pterol and Diesel Variants. Howevver it is concerning as we really need to switch over more renewable source.

5. Seating Capacity Insights

Here, we analyze seating capacity characteristics by calculating the average ex-showroom price for each seating capacity category in the 'car' DataFrame.

```
# Explore seating capacity characteristics
seating_capacity_insights = car.groupby('seating_capacity')['ex-showroom_price_inr'].mean()
print("Seating Capacity Insights:\n", seating_capacity_insights)
```

```
Seating Capacity Insights:
 seating_capacity
2.000    37209609.436
4.000    18033302.614
5.000     2485557.853
6.000     1576018.231
7.000     4393010.776
8.000     1175828.588
9.000     1099911.737
16.000    1205000.000
Name: ex-showroom_price_inr, dtype: float64
```

Observation: 2 Seater always means a sports car. so it will always be high. 16 seater seems like more suspicious, maybe its a Traveller

6. Fuel Efficiency Analysis

In this analysis, we calculate the average ARAI certified mileage for each fuel type in the 'car' DataFrame to understand fuel efficiency characteristics.

```
# Analyze fuel efficiency based on ARAI certified mileage
fuel_efficiency_analysis = car.groupby('fuel_type')['arai_certified_mileage_kmpl'].mean()
print("Fuel Efficiency Analysis:\n", fuel_efficiency_analysis)
```

```
Fuel Efficiency Analysis:
 fuel_type
CNG            28.672
CNG + Petrol   22.323
Diesel         19.423
Electric       18.200
Hybrid         16.792
Petrol         17.201
Name: arai_certified_mileage_kmpl, dtype: float64
```

Observation: Its a surprise that Indian Dads still stick with petrol or diesel for better mileage. Rather they should shitch to CNG for better mileage.

7. Correlation Analysis

Here, we compute and showcase the correlation matrix between numerical columns in the 'car' DataFrame, providing insights into the relationships between different numerical features.

```
# Calculate and display the correlation matrix between numerical columns
car.corr()
```

	showroom_price_inr		
ex-showroom_price_inr	1.000	0.794	0.817
displacement_cc	0.794	1.000	0.879
cylinders	0.817	0.879	1.000
valves_per_cylinder	0.030	0.032	-0.002
fuel_tank_capacity_litres	0.211	0.379	0.341
height_mm	-0.195	-0.026	-0.176
length_mm	0.406	0.657	0.541
width_mm	0.449	0.664	0.565
doors	-0.460	-0.475	-0.501
arai_certified_mileage_kmpl	-0.390	-0.599	-0.498
kerb_weight_kg	-0.031	-0.046	-0.022
gears	0.387	0.511	0.471
ground_clearance_mm	0.112	0.143	0.170
front_track_mm	0.011	0.027	0.034
rear_track_mm	0.010	0.026	0.033
f_tire_diameter_inch	0.492	0.708	0.649
f_tire_aspect_ratio	-0.361	-0.452	-0.462
f_tire_width_mm	0.397	0.661	0.570
r_tire_diameter_inch	0.493	0.710	0.650
r_tire_aspect_ratio	-0.483	-0.614	-0.625
r_tire_width_mm	0.477	0.729	0.654
power_rpm	0.065	0.004	0.030
power_ps	0.855	0.896	0.881
torque_rpm	0.222	0.138	0.210
torque_nm	0.703	0.839	0.765
seating_capacity	-0.307	-0.165	-0.313
wheelbase_mm	0.424	0.648	0.532
basic_warranty_years	0.164	0.254	0.202
boot_space_litres	0.055	0.172	0.107
minimum_turning_radius_meter	0.033	0.088	0.024

Observation: Matrix showing the co-relation between the numeric columns

8. Outliers Detection

This code identifies and examines outliers in all numeric columns of the 'car' DataFrame by comparing values that exceed the mean plus 3 times the standard deviation. It then prints the values of outliers for each numeric column.

```
# Identify and examine outliers in all numeric columns
numeric_columns = car.select_dtypes(include=['number']).columns

outliers = {}
for column in numeric_columns:
    # Filter values where they exceed the mean plus 3 times the standard deviation
    column_outliers = car.loc[car[column] > car[column].mean() + 3 * car[column].std(), column].values
    outliers[column] = column_outliers

# Print values of outliers for each numeric column
for column, values in outliers.items():
    print(f"Values of Outliers in {column}:\n", values)
```

```
 [2218. 2218. 2218. 2226. 2220. 2220. 2220. 2220. 2220. 2220. 2220. 2220.
 2220. 2220. 2220. 2220. 2220. 2220.]
Values of Outliers in doors:
 []
Values of Outliers in arai_certified_mileage_kmpl:
 [36.   35.   43.   33.54 33.54]
Values of Outliers in kerb_weight_kg:
 [10161043. 10161043. 10161043. 10531080. 10531080. 10531080. 10531080.
 10531080. 10161043. 10161043. 10161043. 10531080.]
Values of Outliers in gears:
 []
Values of Outliers in ground_clearance_mm:
 [2955. 2955. 2955. 2955. 2955. 2955. 2955. 2955. 2955. 2955. 2955. 2955.
 2955. 2955.]
Values of Outliers in front_track_mm:
 [147613.  16925.  16925.  16924.  16924.  16924.  16924.  16924.  16924.
  16924.]
Values of Outliers in rear_track_mm:
 [149413.  16864.  16864.  16866.  16866.  16866.  16866.  16866.  16866.
  16866.]
Values of Outliers in f_tire_diameter_inch:
 [22.]
Values of Outliers in f_tire_aspect_ratio:
 [335.]
Values of Outliers in f_tire_width_mm:
 []
Values of Outliers in r_tire_diameter_inch:
 [22.]
Values of Outliers in r_tire_aspect_ratio:
 []
Values of Outliers in r_tire_width_mm:
 [325. 335. 355.]
Values of Outliers in power_rpm:
 [62050. 62050. 62050. 62050.]
Values of Outliers in power_ps:
 [ 609.  585.  630.  610.  650.  590.  608.  608.  605.  570.  625. 1479.
 1600.  585.  639.  610.  580.  610.  639.  571.  602.  631.  631.  610.
  670.  670.  610.  681.  700.  700.  740.  571.  570.  789.  625.  669.
  616.]
Values of Outliers in torque_rpm:
 [6700. 7000.]
Values of Outliers in torque_nm:
 [ 850.  900.  850. 1000.  850.  900. 1020.  850. 1600. 1479. 1712. 1130.]
Values of Outliers in seating_capacity:
 [ 9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.
  9. 16.]
Values of Outliers in wheelbase_mm:
 [3365. 3295. 3295. 3266. 3295. 3320. 3430. 3570. 3488. 3570. 3820.]
Values of Outliers in basic_warranty_years:
 [4. 4. 4. 4. 4. 4. 4. 4. 8. 8. 8. 4. 4. 4. 4.]
Values of Outliers in boot_space_litres:
 [1025. 1025. 1025. 1025. 1702. 1400. 1400. 1400.  909.  909. 1761. 1761.
  909.  909.  909.  909.  981.  981.  981.  981. 1702. 2005. 1050.]
Values of Outliers in minimum_turning_radius_meter:
 [1122. 1122. 1122. 1122. 1161. 1161. 1161. 1161. 1161. 1187. 1187. 1184.
 1184. 1184. 1184. 1184. 1184. 1066. 6750.]
Values of Outliers in number_of_airbags:
 [14. 14. 14.]
```

Observation: Some cars might have an outlying feature, but that doesn't mean they doesn't exit. They exist! But they are special.

9. Average Price by Make

In this step, we calculate and display the average ex-showroom price for each car make in the 'car' DataFrame, providing insights into pricing trends across different makes.

```
# Calculate and display the average ex-showroom price for each car make
average_price_by_make = car.groupby('make')['ex-showroom_price_inr'].mean()
print("Average Ex-showroom Price by Make:\n", average_price_by_make)
```

```
Average Ex-showroom Price by Make:
 make
Aston Martin     36267442.000
Audi              7240430.645
Bajaj              273000.000
Bentley          40565854.333
Bmw               8256216.216
Bugatti         202149167.000
Datsun             496075.417
Dc                3407407.000
Ferrari          42872495.625
Fiat               779556.348
Force             1150500.000
Ford              1123853.814
Honda             1206136.562
Hyundai            987476.546
Icml               970082.909
Isuzu             2225392.800
Jaguar           10464503.045
Jeep              3203428.071
Kia               1701142.857
Lamborghini      39053919.231
Land Rover       14294469.541
Lexus            15015300.000
Mahindra          1059604.300
Maruti Suzuki      673891.092
Maserati         15696889.889
Mercedes-Benz     9479081.544
Mg                1610307.692
Mini              3792000.000
Mitsubishi        3456192.857
Nissan            1695369.172
Porsche          15346000.000
Premier            637603.833
Renault            801107.750
Rolls-Royce      67796334.556
Skoda             1973879.465
Tata               937114.570
Toyota            1653056.098
Volkswagen        1321923.529
Volvo             5834750.000
Name: ex-showroom_price_inr, dtype: float64
```

Observation: This showcases the average price that a brand offers to a customer.

10. Price Segmentation

Reference: Pd.Cut Reference: Unstack

Here, we define price ranges for segmentation based on ex-showroom prices, create a new column 'price_segment' in the 'car' DataFrame, and then calculate and display the average ex-showroom price for each car make within each price segment.

```
# Define price ranges for segmentation
super_expensive_range = (car['ex-showroom_price_inr'] > 5000000)
expensive_range = ((car['ex-showroom_price_inr'] <= 5000000) & (car['ex-showroom_price_inr'] > 2000000))
affordable_range = (car['ex-showroom_price_inr'] <= 2000000)

# Create a new column 'price_segment' based on the defined ranges
car['price_segment'] = pd.cut(car['ex-showroom_price_inr'], bins=[0, 2000000, 5000000, float('inf')],
                              labels=['Affordable', 'Expensive', 'Super Expensive'], right=False)

# Calculate and display the average ex-showroom price for each car make within each price segment
car.groupby(['price_segment', 'make'])['ex-showroom_price_inr'].mean().unstack().transpose()
```

Dc	NaN	3407407.000	NaN
Ferrari	NaN	NaN	42872495.625
Fiat	779556.348	NaN	NaN
Force	1150500.000	NaN	NaN
Ford	801043.949	3207666.667	7462000.000
Honda	1005014.035	2843848.571	NaN
Hyundai	889158.926	2309302.333	NaN
Icml	802013.500	2650777.000	NaN
Isuzu	1820198.667	2833184.000	NaN
Jaguar	NaN	4647816.000	12175293.353
Jeep	1786555.556	2249521.429	8424737.200
Kia	1339312.500	2859000.000	NaN
Lamborghini	NaN	NaN	39053919.231
Land Rover	NaN	NaN	14294469.541
Lexus	NaN	NaN	15015300.000
Mahindra	1025149.750	2920150.000	NaN
Maruti Suzuki	673891.092	NaN	NaN
Maserati	NaN	NaN	15696889.889
Mercedes-Benz	NaN	3623889.722	12181477.769
Mg	1498909.091	2223000.000	NaN
Mini	NaN	3792000.000	NaN
Mitsubishi	NaN	2888465.000	6862560.000
Nissan	997336.929	NaN	21240272.000
Porsche	NaN	NaN	15346000.000
Premier	637603.833	NaN	NaN
Renault	801107.750	NaN	NaN
Rolls-Royce	NaN	NaN	67796334.556
Skoda	1311918.125	2810041.158	NaN
Tata	937114.570	NaN	NaN
Toyota	1066236.364	2917857.143	12164500.000
Volkswagen	969157.143	2968166.667	NaN
Volvo	NaN	3650222.222	8019277.778

Observation: Call the Indian Dads! They need this, before they dream about buying super-expensive cars. This will help them classify their dreams.

11. Model manufactured by Each Maker

This code creates a new DataFrame, unique_models_per_make, that lists unique car models for each make. The models are sorted and presented as comma-separated strings. This provides a concise overview of distinct car models associated with each make.

```
# Create a new DataFrame with unique models per make
unique_models_per_make = car.groupby('make')['model'].unique().apply(lambda x: ', '.join(sorted(x))).reset_index()

# Display the unique models per make
print("Unique Models per Make:\n", unique_models_per_make)
```

```
Unique Models per Make:
                make                                              model
0    Aston Martin                                Db 11, Rapide, Vantage
1            Audi  A3, A3 Cabriolet, A4, A5, A5 Cabriolet, A6, A8...
2           Bajaj                                          Qute (Re60)
3         Bentley    Bentayga, Continental Gt, Flying Spur, Mulsanne
4             Bmw  3-Series, 5-Series, 6-Series, 7-Series, M2 Com...
5         Bugatti                                             Chiron
6          Datsun                                  Go, Go+, Redi-Go
7              Dc                                             Avanti
8         Ferrari  458 Speciale, 458 Spider, 488 Gtb, 812 Superfa...
9            Fiat  Abarth Avventura, Abarth Punto, Avventura, Lin...
10          Force                                           Gurkha
11           Ford  Aspire, Ecosport, Endeavour, Figo, Freestyle, ...
12          Honda  Accord Hybrid, Amaze, Brv, City, Civic, Cr-V, ...
13        Hyundai  Aura, Creta, Elantra, Elite I20, Grand I10, Gr...
14           Icml                                          Extreme
15          Isuzu                            Dmax V-Cross, Mu-X
16         Jaguar                       F-Pace, F-Type, Xe, Xf, Xj
17           Jeep  Compass, Compass Trailhawk, Grand Cherokee, Wr...
18            Kia                                Carnival, Seltos
19    Lamborghini                           Aventador, Huracan, Urus
20     Land Rover  Discovery, Discovery Sport, Range, Range Evoqu...
21          Lexus  Es, Lc 500H, Ls 500H, Lx 450D, Lx 570, Nx 300H...
22       Mahindra  Alturas G4, Bolero, Bolero Power Plus, E Verit...
23  Maruti Suzuki  Alto, Alto 800 Tour, Alto K10, Baleno, Baleno ...
24       Maserati  Ghibli, Grancabrio, Granturismo, Levante, Quat...
25  Mercedes-Benz  A-Class, Amg Gt 4-Door Coupe, Amg-Gt, B-Class,...
26             Mg                                  Hector, Zs Ev
27           Mini  Clubman, Convertible, Cooper 3 Door, Cooper 5 ...
28     Mitsubishi                  Montero, Outlander, Pajero Sport
29         Nissan    Gtr, Kicks, Micra, Micra Active, Sunny, Terrano
30        Porsche  718, 911, Cayenne, Cayenne Coupe, Macan, Panamera
31        Premier                                             Rio
32        Renault               Captur, Duster, Kwid, Lodgy, Triber
33   Rolls-Royce  Cullinan, Dawn, Drophead Coupe, Ghost Series I...
34          Skoda  Kodiaq, Kodiaq Scout, Monte Carlo, Octavia, Ra...
35           Tata  Altroz, Bolt, Harrier, Hexa, Nano Genx, Nexon,...
36         Toyota  Camry, Corolla Altis, Etios Cross, Etios Liva,...
37     Volkswagen                 Ameo, Passat, Polo, Tiguan, Vento
38          Volvo  S60, S60 Cross Country, S90, V40, V40 Cross Co...
```

Observation: This showcases the different models that every car brand offers to customer.

12. Feature-Rich Cars

This code creates a 'num_features' column in the 'car' DataFrame, counting the 'Yes' values across a list of car features. This allows for a concise numerical representation of the number of features in each car.

```
# List of features (lowercased)
features = [
    'keyless_entry', 'tachometer', 'tripmeter', 'start_/_stop_button', '12v_power_outlet', 'audiosystem',
    'aux-in_compatibility', 'average_fuel_consumption', 'bluetooth', 'boot-lid_opener', 'cd_/_mp3_/_dvd_player',
    'central_locking', 'child_safety_locks', 'clock', 'cup_holders', 'distance_to_empty', 'door_pockets',
    'engine_malfunction_light', 'fm_radio', 'fuel-lid_opener', 'fuel_gauge', 'handbrake', 'instrument_console',
    'low_fuel_warning', 'multifunction_display', 'sun_visor', 'third_row_ac_vents', 'ventilation_system',
    'auto-dimming_rear-view_mirror', 'hill_assist', 'gear_indicator', 'engine_immobilizer', 'seat_back_pockets',
    'abs_(anti-lock_braking_system)', 'headlight_reminder', 'adjustable_headrests', 'gross_vehicle_weight', 'airbags'
    'door_ajar_warning', 'ebd_(electronic_brake-force_distribution)', 'fasten_seat_belt_warning', 'gear_shift_reminde
    'adjustable_steering_column', 'parking_assistance', 'key_off_reminder', 'usb_compatibility', 'cigarette_lighter',
    'infotainment_screen', 'multifunction_steering_wheel', 'average_speed', 'eba_(electronic_brake_assist)',
    'seat_height_adjustment', 'navigation_system', 'second_row_ac_vents', 'rear_center_armrest', 'ipod_compatibility'
    'esp_(electronic_stability_program)', 'cooled_glove_box', 'turbocharger', 'isofix_(child-seat_mount)',
    'rain_sensing_wipers', 'leather_wrapped_steering', 'automatic_headlamps', 'asr_/_traction_control', 'cruise_contr
]

# Create a new column 'num_features' with the count of 'Yes' values for each row
car['num_features'] = car[features].apply(lambda row: row.eq('Yes').sum(), axis=1)
```

```
# Find and display the top 20 cars with more features
top_cars_more_features = car.sort_values(by='num_features', ascending=False).head(20)[['make', 'model', 'ex-showroom_
print("Top 20 Cars with More Features:\n", top_cars_more_features)
```

```
Top 20 Cars with More Features:
                make             model  ex-showroom_price_inr  num_features
247            Skoda  Superb Sportline                3149599            65
792       Land Rover      Range Evoque                5985000            65
278           Jaguar                Xe                4633000            65
277           Jaguar                Xe                4498000            65
839           Jaguar                Xj               11129599            65
266             Audi                Q3                3496750            65
758            Skoda            Superb                2849599            65
265             Audi                Q3                4361000            65
817             Audi                Q7                7711500            65
261            Skoda            Kodiaq                3299599            65
260            Skoda            Kodiaq                3678599            65
259            Skoda            Kodiaq                3299599            65
246            Skoda  Superb Sportline                2899599            65
816             Audi                Q7                8111500            65
791       Land Rover      Range Evoque                5494000            65
790       Land Rover      Range Evoque                5985000            65
789       Land Rover      Range Evoque                5494000            65
880     Rolls-Royce    Drophead Coupe               83755383            64
1048  Mercedes-Benz               Gle               12500000            64
318             Bmw          7-Series               13510000            64
```

Observation: Are you rich? Want more features in a car? Go for Jaguar Xj Else you can go for Skoda Superb!

13. Cars with Less Features

Here, we identify and display the last 5 cars in the 'car' DataFrame based on the number of features, showcasing cars with a lower number of features

```
# Find and display the last 5 cars with less features
cars_less_features = car.sort_values(by='num_features', ascending=False).tail(5)[['make', 'model', 'ex-showroom_price
print("5 Cars with Less Features:\n", cars_less_features)
```

```
5 Cars with Less Features:
                make               model  ex-showroom_price_inr  num_features
536        Mahindra    Bolero Power Plus                 859497            14
82          Premier                  Rio                 568623            12
353     Rolls-Royce             Cullinan               69500000             8
1162         Jaguar               F-Type               23659454             7
862   Mercedes-Benz  Amg Gt 4-Door Coupe               24200000             6
```

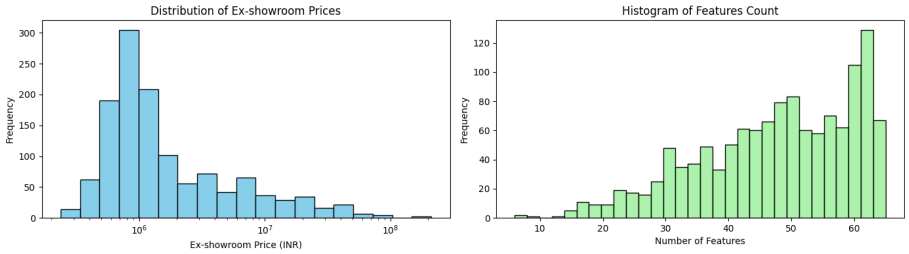
Observation: If you want to waste money with less features, regret buying Rolls Royce Cullinan. Else buy Mahindra Bolero.

Data Vizualization: Univariate

1. Histogram

Here, we create a side-by-side histogram subplot for the distribution of ex-showroom prices and the count of features in the 'car' DataFrame. The first subplot uses a log scale for better visualization of ex-showroom prices.

```
# Create a side-by-side histogram subplot for sales and days to ship
fig, axes = plt.subplots(1, 2, figsize=(14, 4))

# First subplot for sales distribution
log_bins = np.logspace(np.log10(car['ex-showroom_price_inr'].min()), np.log10(car['ex-showroom_price_inr'].max()), 20
axes[0].hist(car['ex-showroom_price_inr'], bins=log_bins, color='skyblue', edgecolor='black')
axes[0].set_title('Distribution of Ex-showroom Prices')
axes[0].set_xlabel('Ex-showroom Price (INR)')
axes[0].set_ylabel('Frequency')
axes[0].set_xscale('log')  # Set x-axis to log scale

# Second subplot for the count of features
sns.histplot(car['num_features'], bins=30, color='lightgreen', edgecolor='black', ax=axes[1])
axes[1].set_title('Histogram of Features Count')
axes[1].set_xlabel('Number of Features')
axes[1].set_ylabel('Frequency')

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```



Observation:

```
Plot 1 shows that the Indian car market is more focused on affordable range which is 10 Lacs.
Plot 2 shows that most of the Indian cars have more than 30 features in their car. Yes, India do want to get an industry ready mc
```

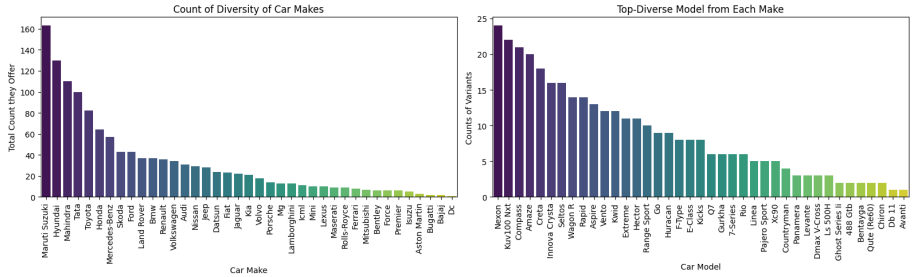
2. Bar Charts

Reference: .idmax()

This code creates a side-by-side bar plot with two subplots. The first subplot shows the count of car makes to visualize the diversity of car makes. The second subplot displays the count of variants for the top-diverse model from each make.

```
# Create a side-by-side bar plot for car makes and top-diverse models
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# First subplot for car makes
sns.countplot(data=car, x='make', palette='viridis', order=car['make'].value_counts().index, ax=axes[0])
axes[0].set_title('Count of Diversity of Car Makes')
axes[0].set_xlabel('Car Make')
axes[0].set_ylabel('Total Count they Offer')
axes[0].tick_params(axis='x', rotation=90)

# Identify the top-diverse model for each make
top_models_by_make = car.groupby('make')['model'].apply(lambda x: x.value_counts().idxmax())

# Filter the car DataFrame to include only the rows corresponding to the top-selling models
top_diverse_cars = car[car['model'].isin(top_models_by_make)]

# Second subplot for top-diverse models
sns.countplot(data=top_diverse_cars, x='model', order=top_diverse_cars['model'].value_counts().index, palette='viridi
axes[1].set_title('Top-Diverse Model from Each Make')
axes[1].set_xlabel('Car Model')
axes[1].set_ylabel('Counts of Variants')
axes[1].tick_params(axis='x', rotation=90)

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```

```
<ipython-input-161-ab20d5d321fc>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.countplot(data=car, x='make', palette='viridis', order=car['make'].valu
<ipython-input-161-ab20d5d321fc>:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.countplot(data=top_diverse_cars, x='model', order=top_diverse_cars['mod
```

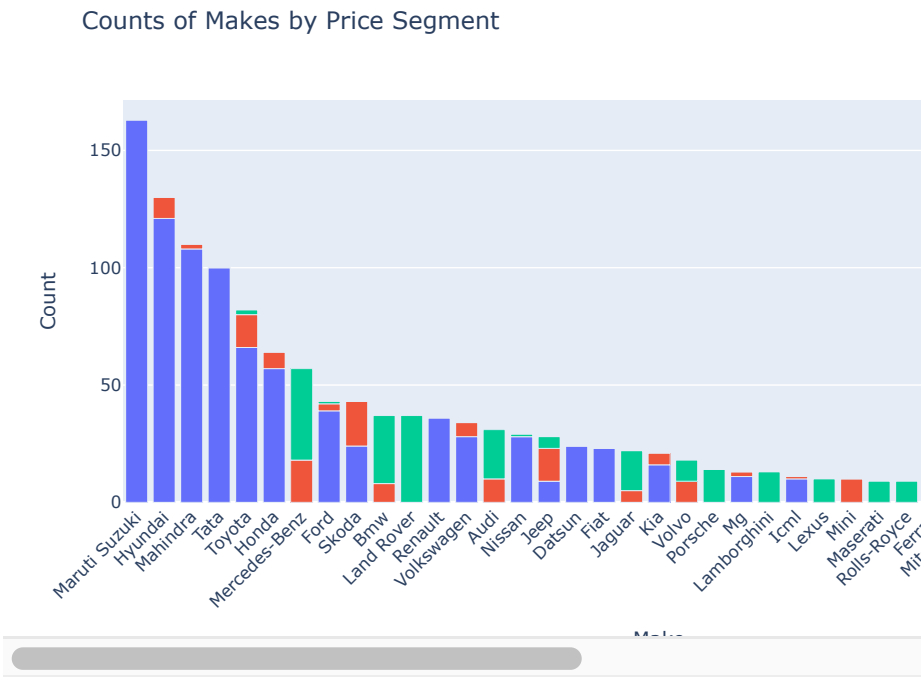


Observation:

```
This was expected! Indian dad's trust is always Maruti Suzuki! Who's DC anyway? The one who created Tarzan the wonder car?
Well Never know Ranjit sir would offer so much variants in Nexon. Anyways Avanti is an overrated modified car for me.
```

Now we calculats the counts of car makes and price segments, then create a stacked bar chart using Plotly to visualize the distribution of makes across different price segments.

```
# Calculate the counts of makes and price segments
make_counts = car.groupby(['make', 'price_segment']).size().reset_index(name='count')

# Sort the makes in descending order based on the total count
sorted_makes = make_counts.groupby('make')['count'].sum().sort_values(ascending=False).index

# Create a stacked bar chart using plotly
fig = px.bar(make_counts, x='make', y='count', color='price_segment', title='Counts of Makes by Price Segment',
             labels={'make': 'Make', 'count': 'Count', 'price_segment': 'Price Segment'}, height=500,
             category_orders={'make': sorted_makes})

# Customize the layout
fig.update_layout(xaxis=dict(tickangle=-45, tickmode='array', tickvals=list(range(len(sorted_makes))),
                             ticktext=sorted_makes))

# Show the chart
fig.show()
```

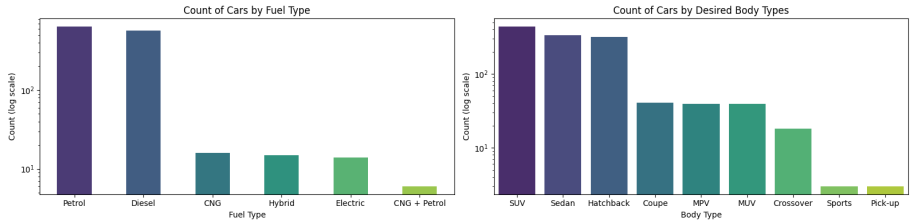


Observation: Well we can now see, why Indians prefer Maruti, Hyundai, Mahindra, Tata, Toyota and Honda! Its super affordable! Also we can see them everywhere on Indian roads.

Here, we define the desired body types, filter the DataFrame to include only those body types, and then create side-by-side bar charts for fuel type and the distribution of desired body types. The y-axis is displayed on a log scale for better visualization.

```
# Define the desired body types
desired_body_types = ['Coupe', 'Crossover', 'Hatchback', 'MPV', 'MUV', 'Pick-up', 'Sedan', 'Sports', 'SUV']

# Filter the DataFrame to include only the desired body types
filtered_car = car[car['body_type'].isin(desired_body_types)]

# Create side-by-side bar charts for fuel type and body type distribution
fig, axes = plt.subplots(1, 2, figsize=(16, 4))

# Bar chart for fuel type distribution
sns.countplot(data=car, x='fuel_type',width = 0.5, order=car['fuel_type'].value_counts().index, palette='viridis', a>
axes[0].set_yscale('log')
axes[0].set_title('Count of Cars by Fuel Type')
axes[0].set_xlabel('Fuel Type')
axes[0].set_ylabel('Count (log scale)')

# Bar chart for filtered body type distribution
sns.countplot(data=filtered_car, x='body_type', order=filtered_car['body_type'].value_counts().index, palette='viridi
axes[1].set_yscale('log')
axes[1].set_title('Count of Cars by Desired Body Types')
axes[1].set_xlabel('Body Type')
axes[1].set_ylabel('Count (log scale)')

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```

```
<ipython-input-164-af4def5b4f17>:11: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i

<ipython-input-164-af4def5b4f17>:18: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i
```



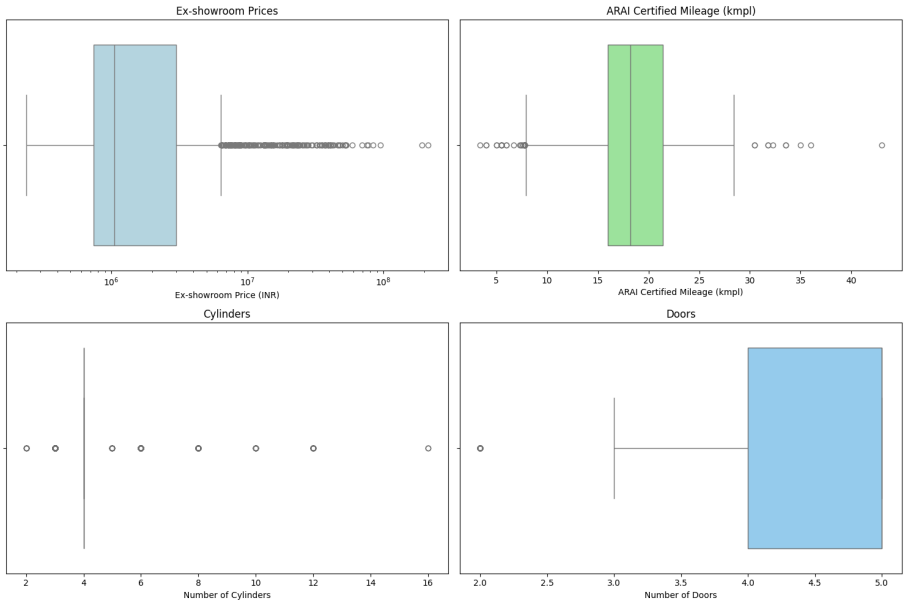
Observation:

```
Petrol and diesel is still preferred in India by lot.
Nowadays there is hype of SUV and Sedan, people of India prefere more spacious car. I rather see more SUVs and sedans nowadays
```

3. Box Plots

Here, we create a 2x2 matrix of box plots for different features in the 'car' DataFrame, showcasing the distribution and variability of ex-showroom prices, ARAI certified mileage, number of cylinders, and number of doors. The x-axis for ex-showroom prices is displayed on a log scale for better visualization.

```
# Create a 2x2 matrix of box plots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Box plot for ex-showroom prices
sns.boxplot(data=car, x='ex-showroom_price_inr', color='lightblue', ax=axes[0, 0])
axes[0, 0].set_xscale('log')
axes[0, 0].set_title('Ex-showroom Prices')
axes[0, 0].set_xlabel('Ex-showroom Price (INR)')

# Box plot for ARAI certified mileage
sns.boxplot(data=car, x='arai_certified_mileage_kmpl', color='lightgreen', ax=axes[0, 1])
axes[0, 1].set_title('ARAI Certified Mileage (kmpl)')
axes[0, 1].set_xlabel('ARAI Certified Mileage (kmpl)')

# Box plot for cylinders
sns.boxplot(data=car, x='cylinders', color='lightcoral', ax=axes[1, 0])
axes[1, 0].set_title('Cylinders')
axes[1, 0].set_xlabel('Number of Cylinders')

# Box plot for doors
sns.boxplot(data=car, x='doors', color='lightskyblue', ax=axes[1, 1])
axes[1, 1].set_title('Doors')
axes[1, 1].set_xlabel('Number of Doors')

# Adjust layout
plt.tight_layout()
plt.show()
```



Observation:

Yes, prices of car, have extensive outliers depending of rich class of people buying super expensive sports or luxury cars.
An average car in India gives mileage of 17-22 kmpl.
Cylinders of Indian cars remain fixed and distributed.
Most car of India has 4 doors except the sports car having 2 doors.

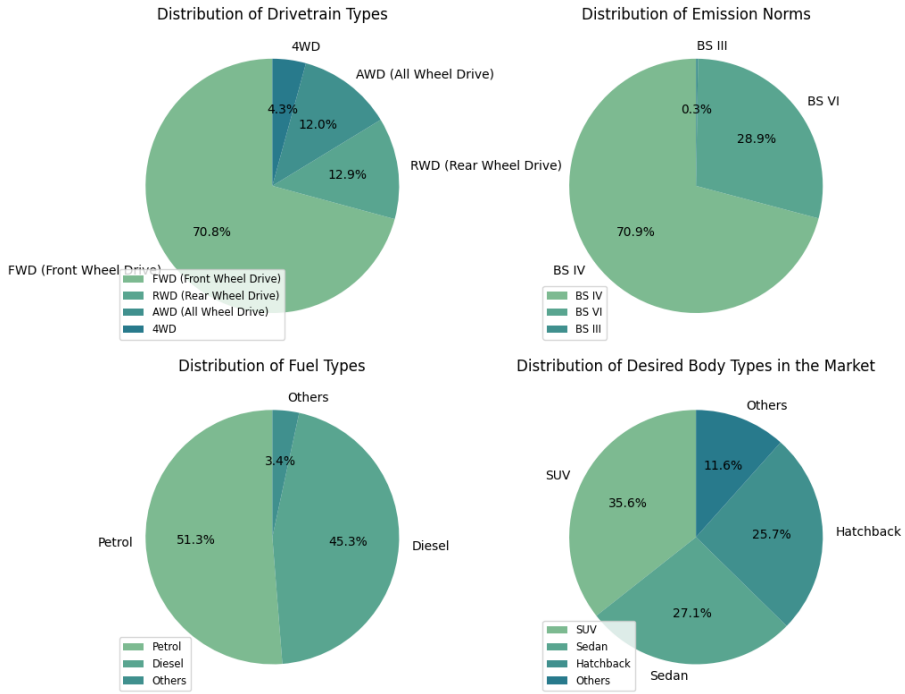
4. Pie Charts

Reference: Function In this function, we take a data series and a threshold percentage as inputs. The function calculates the percentages of each category in the series and creates a mask to identify categories exceeding the specified threshold percentage. Categories below the threshold are grouped into 'Others,' and the resulting grouped series is returned.

```
# Function to group small percentages into 'Other'
def group_small_percentages(data_series, threshold_percent=10):
    percentages = data_series.value_counts(normalize=True) * 100
    mask = percentages >= threshold_percent
    grouped_series = data_series.apply(lambda x: x if mask.get(x, False) else 'Others')
    return grouped_series
```

In this code, we remove 'unspecified' values from each column in the 'car' DataFrame and then create a 2x2 matrix of pie charts to visualize the distribution of drivetrain types, emission norms, fuel types, and desired body types in the market. The 'group_small_percentages' function is utilized to handle small percentages and group them into 'Others' for better visualization.

```
# Remove 'unspecified' values from each column
car_no_unspecified = car.replace('unspecified', np.nan).dropna()

# Create a 2x2 matrix for pie charts
plt.figure(figsize=(10, 8))

# Pie chart for Drivetrain
plt.subplot(2, 2, 1)
drivetrain_distribution = car_no_unspecified['drivetrain'].value_counts()
plt.pie(drivetrain_distribution, labels=drivetrain_distribution.index, autopct='%1.1f%%', startangle=90, colors=sns.c
plt.title('Distribution of Drivetrain Types')
plt.legend(loc='lower left', fontsize='small')

# Pie chart for Emission Norm
plt.subplot(2, 2, 2)
emission_norm_distribution = car_no_unspecified['emission_norm'].value_counts()
plt.pie(emission_norm_distribution, labels=emission_norm_distribution.index, autopct='%1.1f%%', startangle=90, colors
plt.title('Distribution of Emission Norms')
plt.legend(loc='lower left', fontsize='small')

# Pie chart for Fuel Type
plt.subplot(2, 2, 3)
fuel_type_distribution = group_small_percentages(car_no_unspecified['fuel_type'])
plt.pie(fuel_type_distribution.value_counts(), labels=fuel_type_distribution.value_counts().index, autopct='%1.1f%%',
plt.title('Distribution of Fuel Types')
plt.legend(loc='lower left', fontsize='small')

# Pie chart for Desired Body Types
plt.subplot(2, 2, 4)
body_type_distribution = group_small_percentages(filtered_car['body_type'])
plt.pie(body_type_distribution.value_counts(), labels=body_type_distribution.value_counts().index, autopct='%1.1f%%',
plt.title('Distribution of Desired Body Types in the Market')
plt.legend(loc='lower left', fontsize='small')

plt.tight_layout()
plt.show()
```

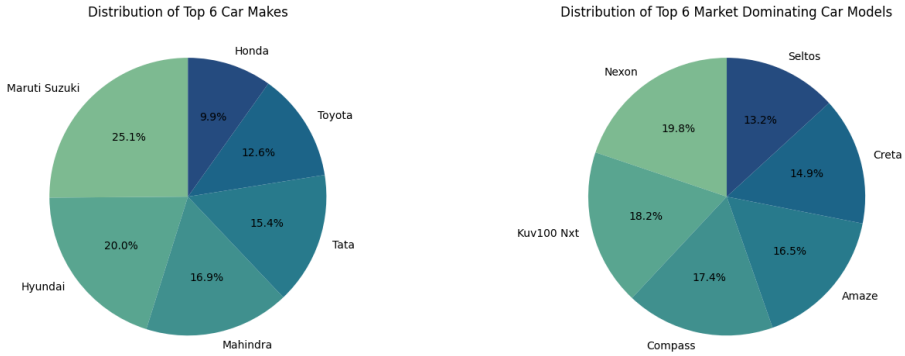


Observation:

```
Most cars in India are Forward and Rear wheel drive due to even terrain. 4 Wheel Drive is only required for long trekking such as
Most cars of India is BS4, which is more harmful for nature, however its preferred to use BS6.
Petrols and diesels are always first priority of Indian Dads!
India grows the hype of SUVs and Sedans, more than Hatchbacks.
```

Here, we are visualizing the distribution of the top 6 car makes and market-dominating car models using pie charts. The first subplot shows the distribution of the top 6 car makes, while the second subplot illustrates the distribution of the top 6 market-dominating car models.

```
# Select the top 6 makes and market-dominating models based on counts
top_makes = car['make'].value_counts().head(6)
top_models = car['model'].value_counts().head(6)

# Create a 1x2 matrix of subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Plot the pie chart for top makes in the first subplot (axes[0])
axes[0].pie(top_makes, labels=top_makes.index, autopct='%1.1f%%', startangle=90, colors=sns.color_palette('crest'))
axes[0].set_title('Distribution of Top 6 Car Makes')

# Plot the pie chart for top models in the second subplot (axes[1])
axes[1].pie(top_models, labels=top_models.index, autopct='%1.1f%%', startangle=90, colors=sns.color_palette('crest'))
axes[1].set_title('Distribution of Top 6 Market Dominating Car Models')

# Adjust layout for better spacing
plt.tight_layout()

# Show the plots
plt.show()
```



Observation:

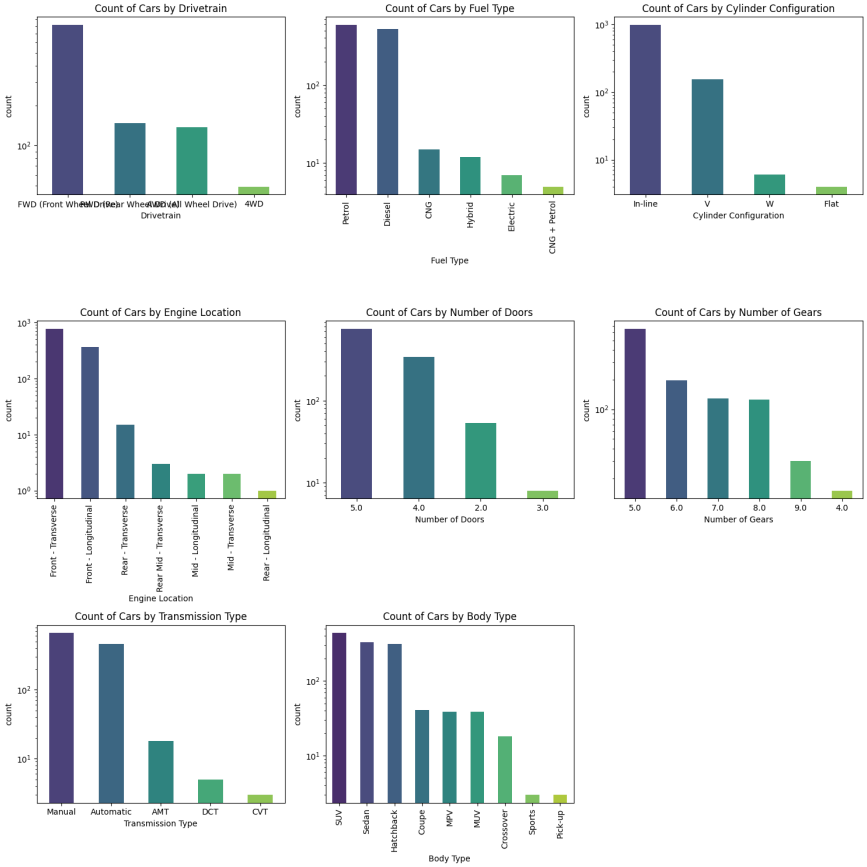
```
Maruti Suzuki and Hyundai shares more market in India
Tata Nexon didn't let us leave with a doubt of the toughest competitor in Indian Market
```

5. Count Plots

Here, we create a 3x3 matrix of subplots to visualize count plots for various categorical columns in the 'car' DataFrame. Each subplot focuses on different aspects like drivetrain, fuel type, cylinder configuration, engine location, number of doors, number of gears, transmission type, and body type. The use of logarithmic scale on the y-axis enhances visibility.

```
fig, axes = plt.subplots(3, 3, figsize=(15, 15))

# Count plot for drivetrain
sns.countplot(data=car_no_unspecified, x='drivetrain', ax=axes[0, 0], palette='viridis', order=car_no_unspecified['dr
axes[0, 0].set_title('Count of Cars by Drivetrain')
axes[0, 0].set_xlabel('Drivetrain')
axes[0, 0].set_yscale('log')

# Count plot for fuel type
sns.countplot(data=car_no_unspecified, x='fuel_type', ax=axes[0, 1], palette='viridis', order=car_no_unspecified['fue
axes[0, 1].set_title('Count of Cars by Fuel Type')
axes[0, 1].tick_params(axis='x', rotation=90)
axes[0, 1].set_xlabel('Fuel Type')
axes[0, 1].set_yscale('log')

# Count plot for cylinder configuration
sns.countplot(data=car_no_unspecified, x='cylinder_configuration', ax=axes[0, 2], palette='viridis', order=car_no_uns
axes[0, 2].set_title('Count of Cars by Cylinder Configuration')
axes[0, 2].set_xlabel('Cylinder Configuration')
axes[0, 2].set_yscale('log')

# Count plot for engine location
sns.countplot(data=car_no_unspecified, x='engine_location', ax=axes[1, 0], palette='viridis', order=car_no_unspecifie
axes[1, 0].set_title('Count of Cars by Engine Location')
axes[1, 0].tick_params(axis='x', rotation=90)
axes[1, 0].set_xlabel('Engine Location')
axes[1, 0].set_yscale('log')

# Count plot for doors
sns.countplot(data=car_no_unspecified, x='doors', ax=axes[1, 1], palette='viridis', order=car_no_unspecified['doors']
axes[1, 1].set_title('Count of Cars by Number of Doors')
axes[1, 1].set_xlabel('Number of Doors')
axes[1, 1].set_yscale('log')

# Count plot for gears
sns.countplot(data=car_no_unspecified, x='gears', ax=axes[1, 2], palette='viridis', order=car_no_unspecified['gears']
axes[1, 2].set_title('Count of Cars by Number of Gears')
axes[1, 2].set_xlabel('Number of Gears')
axes[1, 2].set_yscale('log')

# Count plot for type
sns.countplot(data=car_no_unspecified, x='type', ax=axes[2, 0], palette='viridis', order=car_no_unspecified['type'].v
axes[2, 0].set_title('Count of Cars by Transmission Type')
axes[2, 0].set_xlabel('Transmission Type')
axes[2, 0].set_yscale('log')

# Count plot for body type
sns.countplot(data=filtered_car, x='body_type', ax=axes[2, 1], palette='viridis', order=filtered_car['body_type'].val
axes[2, 1].set_title('Count of Cars by Body Type')
axes[2, 1].tick_params(axis='x', rotation=90)
axes[2, 1].set_xlabel('Body Type')
axes[2, 1].set_yscale('log')

# Hide the empty subplot in the last row and last column
axes[2, 2].axis('off')

plt.tight_layout()
plt.show()
```

```
Passing `palette` without assigning `hue` is deprecated and will be remov

<ipython-input-170-16a9c84177fa>:42: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remov

<ipython-input-170-16a9c84177fa>:48: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remov
```



Observation:

```
Due to even terrains (except pothholes), all cars are suitable to use Forward and Rear wheel drive.
Petrol and Diesel dominates Indian market
Most cars in India are required for city drive, they are rarely used for long drives. In-line is most suitable cylinder.
Almost every affordable car got their engine located in front.
5 doors ? Really ? I think they are counting the trunk.
Most manual cars use 5 Geared transmission.
Slowly the automatic transmission is rising, which shows that Indian people are lazy to hit the clutch even.
Is SUV the new hatchback? Interesting!
```

Data Vizualization: Bivariate

1. Correlation Matrix

Here, we select specific dimensions (height, length, width, kerb weight, and ground clearance) from the 'car' DataFrame and calculate the correlation matrix. The resulting matrix shows the correlation coefficients between these dimensions.

```
# Select the dimensions and calculate the correlation matrix
dimension_subset = car[['height_mm', 'length_mm', 'width_mm', 'kerb_weight_kg', 'ground_clearance_mm']].corr()

# Display the correlation matrix
dimension_subset
```

	height_mm	length_mm	width_mm	kerb_weight_kg	ground_cle
height_mm	1.000	0.145	0.197	-0.043	
length_mm	0.145	1.000	0.757	-0.062	
width_mm	0.197	0.757	1.000	-0.058	
kerb_weight_kg	-0.043	-0.062	-0.058	1.000	

Next steps: Generate code with dimension_subset View recommended plots

Observation: Dimesions of car that is really required

In this section, we choose specific columns related to tire features from the 'car' DataFrame and compute the correlation matrix. The resulting matrix displays the correlation coefficients between these tire-related dimensions.

```
# Select the specified columns and calculate the correlation matrix
correlation_matrix_tire = car[['front_track_mm', 'rear_track_mm', 'f_tire_diameter_inch', 'f_tire_aspect_ratio',
                               'f_tire_width_mm', 'r_tire_diameter_inch', 'r_tire_aspect_ratio', 'r_tire_width_mm']].

# Display the correlation matrix
correlation_matrix_tire
```

	front_track_mm	rear_track_mm	f_tire_diameter_inch	f_tire
front_track_mm	1.000	1.000	0.047	
rear_track_mm	1.000	1.000	0.045	
f_tire_diameter_inch	0.047	0.045	1.000	
f_tire_aspect_ratio	-0.022	-0.021	-0.567	
f_tire_width_mm	0.045	0.043	0.867	
r_tire_diameter_inch	0.047	0.045	0.999	
r_tire_aspect_ratio	-0.025	-0.024	-0.762	
r_tire_width_mm	0.034	0.033	0.878	

Next steps: Generate code with correlation_matrix_tire View recommended plots

Observation: Tires also matter for the roads full of potholes.

Here, we choose specific columns related to power and torque from the 'car' DataFrame and compute the correlation matrix. The resulting matrix shows the correlation coefficients between these power and torque-related dimensions.

```
# Select the specified columns and calculate the correlation matrix
correlation_matrix_power_torque = car[['power_rpm', 'power_ps', 'torque_rpm', 'torque_nm']].corr()

# Display the correlation matrix
correlation_matrix_power_torque
```

	power_rpm	power_ps	torque_rpm	torque_nm
power_rpm	1.000	0.064	0.281	-0.066
power_ps	0.064	1.000	0.245	0.855
torque_rpm	0.281	0.245	1.000	-0.070
torque_nm	-0.066	0.855	-0.070	1.000

Next steps: Generate code with correlation_matrix_power_torque View recommended plots

Observation: Indian Dads are less interested in this! Why did I find this even? Its a more genz matter.

In this part, we choose specific columns related to gears, doors, and additional features from the 'car' DataFrame and compute the correlation matrix.

```
# Select the specified columns and calculate the correlation matrix
correlation_matrix_gears_doors_features = car[['gears', 'doors', 'num_features']].corr()

# Display the correlation matrix
correlation_matrix_gears_doors_features
```

	gears	doors	num_features
gears	1.000	-0.213	0.507
doors	-0.213	1.000	-0.181
num_features	0.507	-0.181	1.000

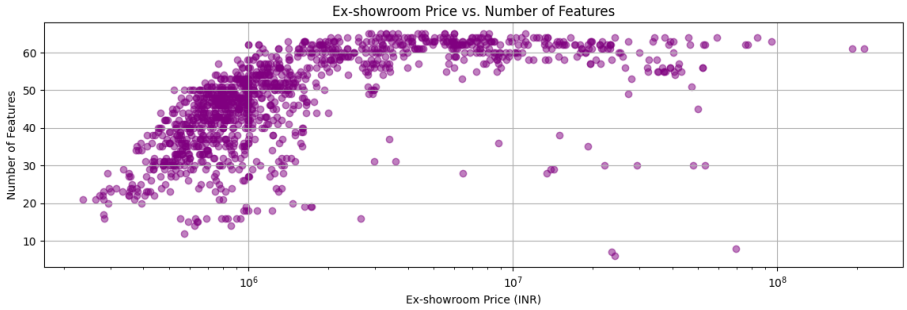
Next steps: Generate code with correlation_matrix_gears_doors_features View recommended plots

Observation: Yes! Features, door and gears are the important factors for Indians.

2. Scatter Plots

Here, we create a scatter plot to visualize the relationship between the ex-showroom price and the number of features for each car. The x-axis is scaled logarithmically for better clarity.

```
# Scatter plot of ex-showroom price vs. number of features
plt.figure(figsize=(14, 4))
plt.scatter(car['ex-showroom_price_inr'], car['num_features'], alpha=0.5, c='purple')
plt.title('Ex-showroom Price vs. Number of Features')
plt.xlabel('Ex-showroom Price (INR)')
plt.ylabel('Number of Features')
plt.xscale('log')
plt.grid(True)
plt.show()
```

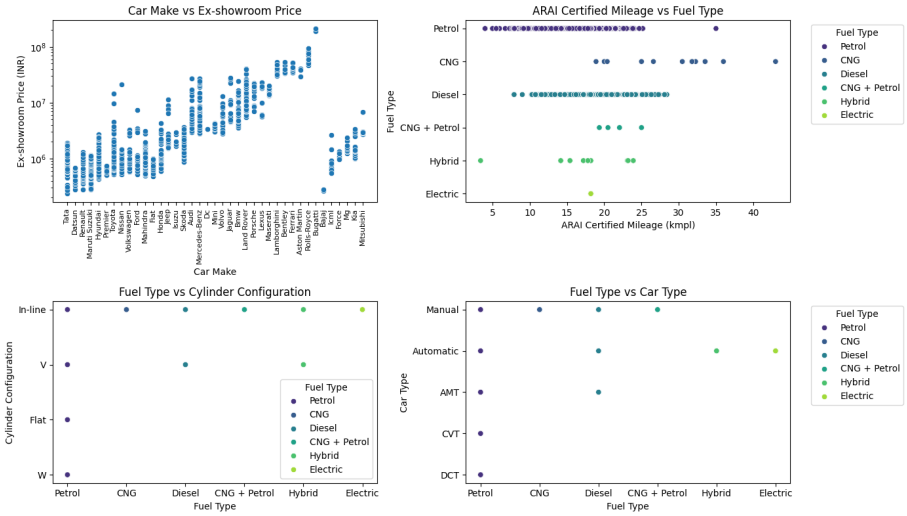


Observation: Gives an informative insight as we see the number of features are growing upto 10 Lakhs, as the customers of India prefer more features in more affordable car.

The plots include comparisons between car make and ex-showroom price, ARAI certified mileage and fuel type, fuel type and cylinder configuration, and fuel type and car type. Legends and log scale on the y-axis are applied for better visualization.

```
fig, axs = plt.subplots(2, 2, figsize=(14, 8))

# Scatter plot between car make and ex-showroom price
sns.scatterplot(x='make', y='ex-showroom_price_inr', data=car, palette='viridis', ax=axs[0, 0])
axs[0, 0].set_title('Car Make vs Ex-showroom Price')
axs[0, 0].set_xlabel('Car Make')
axs[0, 0].set_ylabel('Ex-showroom Price (INR)')
axs[0, 0].set_yscale('log')
axs[0, 0].tick_params(axis='x', rotation=90,labelsize=8)

# Scatter plot between ARAI certified mileage and fuel type
sns.scatterplot(x='arai_certified_mileage_kmpl', y='fuel_type', data=car, hue='fuel_type', palette='viridis', ax=axs
axs[0, 1].set_title('ARAI Certified Mileage vs Fuel Type')
axs[0, 1].set_xlabel('ARAI Certified Mileage (kmpl)')
axs[0, 1].set_ylabel('Fuel Type')
axs[0, 1].legend(title='Fuel Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Scatter plot between fuel type and cylinder configuration
sns.scatterplot(x='fuel_type', y='cylinder_configuration', data=car_no_unspecified, hue='fuel_type', palette='viridis
axs[1, 0].set_title('Fuel Type vs Cylinder Configuration')
axs[1, 0].set_xlabel('Fuel Type')
axs[1, 0].set_ylabel('Cylinder Configuration')
axs[1, 0].legend(title='Fuel Type')

# Scatter plot between fuel type and car type
sns.scatterplot(x='fuel_type', y='type', data=car_no_unspecified, hue='fuel_type', palette='viridis', ax=axs[1, 1])
axs[1, 1].set_title('Fuel Type vs Car Type')
axs[1, 1].set_xlabel('Fuel Type')
axs[1, 1].set_ylabel('Car Type')
axs[1, 1].legend(title='Fuel Type', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```

```
<ipython-input-176-9b08a900e29c>:4: UserWarning:

Ignoring `palette` because no `hue` variable has been assigned.
```



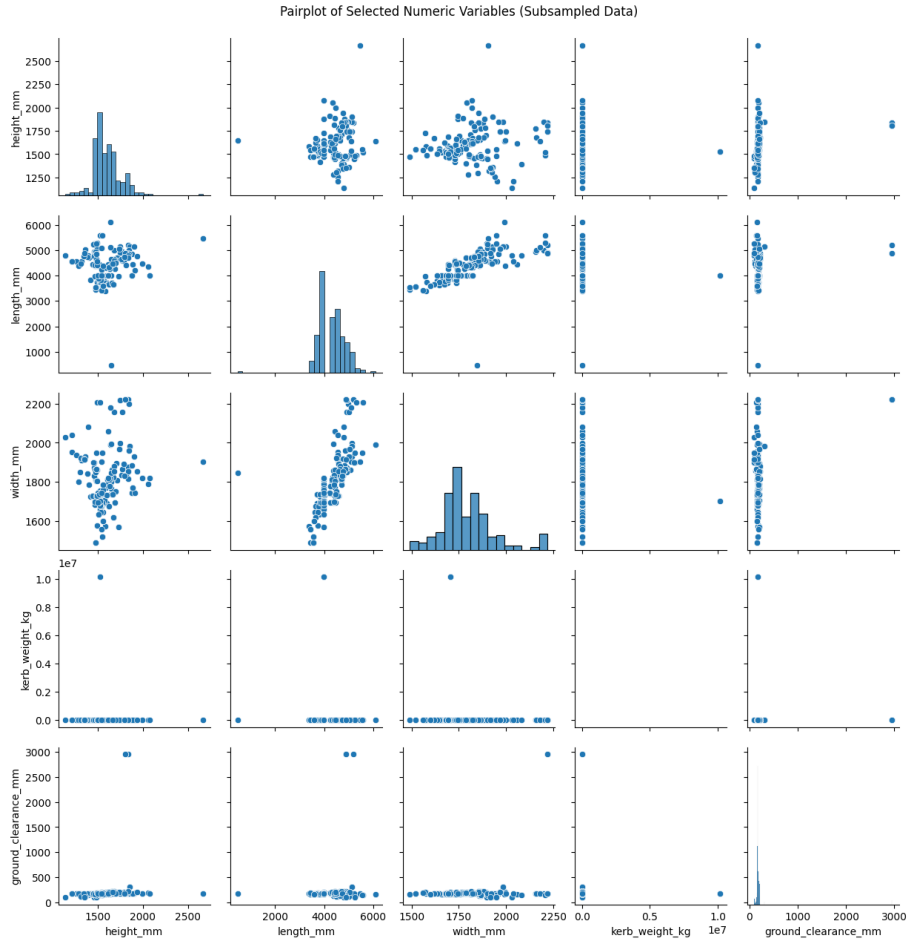
Observation:

```
Tata, Maruti, Hundai, Mahindra, Kia, Bajaj and Honda provides the afforadable cars focusing more customer.
CNG provides really good mileage as compare petrol and diesel.
In-line transmission supports all fuel type, making it more universal. V type is for sports
Never knew that manual isn't adjustable with electric. Also Automatic isn't adjustable with CNG.
```

3. Pair Plots

We create a pairplot to visualize relationships between selected numeric variables (height, length, width, kerb weight, and ground clearance) in the 'car' DataFrame. The data is subsampled to improve plot readability. The title is added for clarity.

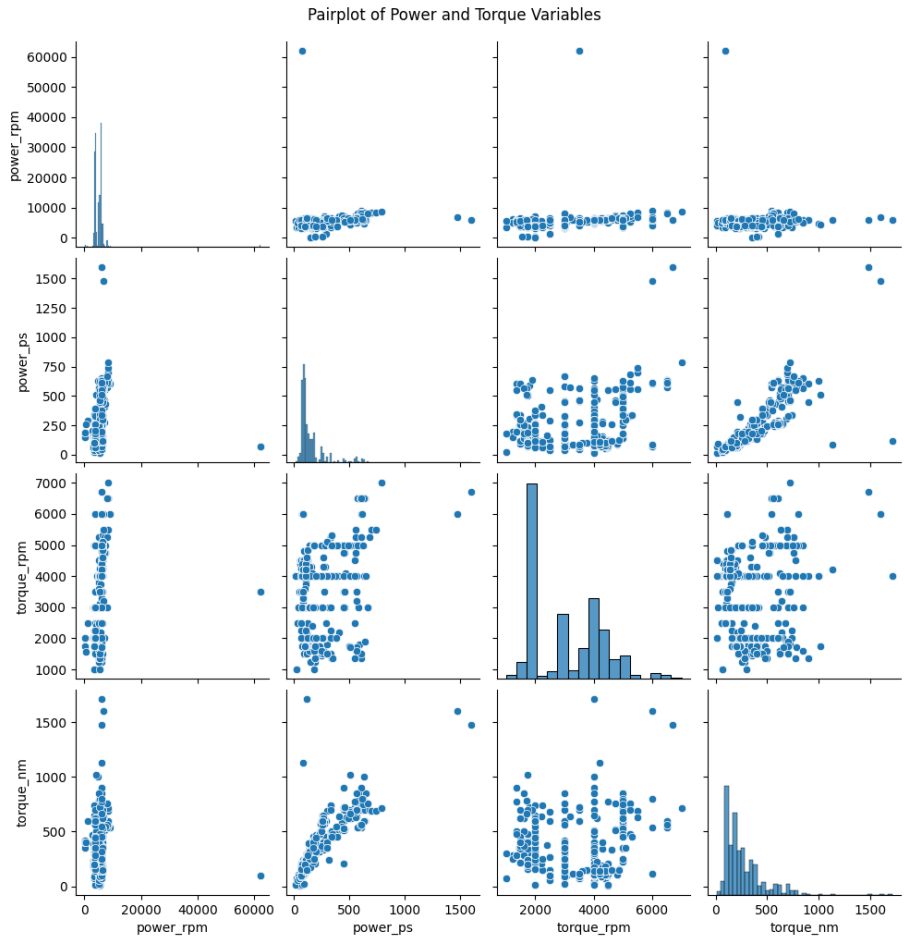
```
# Pairplot for selected numeric variables
selected_numeric_variables = ['height_mm', 'length_mm', 'width_mm', 'kerb_weight_kg', 'ground_clearance_mm']
# Subsample the data
subsampled_data = car.sample(frac=0.2, random_state=42)

# Pairplot for selected numeric variables with subsampled data
sns.pairplot(subsampled_data[selected_numeric_variables])
plt.suptitle('Pairplot of Selected Numeric Variables (Subsampled Data)', y=1.02)
plt.show()
```



Observation: Pairplots give an insight of the dimensions that one looks before buying, to save taxes.

To visualize relationships between selected power and torque variables (power RPM, power PS, torque RPM, and torque NM) in the 'car' DataFrame. The title is included for clarity.

```
# Selected numeric variables
selected_power_torque_variables = ['power_rpm', 'power_ps', 'torque_rpm', 'torque_nm']

# Pairplot for selected power and torque variables
sns.pairplot(car[selected_power_torque_variables])
plt.suptitle('Pairplot of Power and Torque Variables', y=1.02)
plt.show()
```



Observation: Pairplot for the car lovers, to compare the power and torque then, buy the best car.

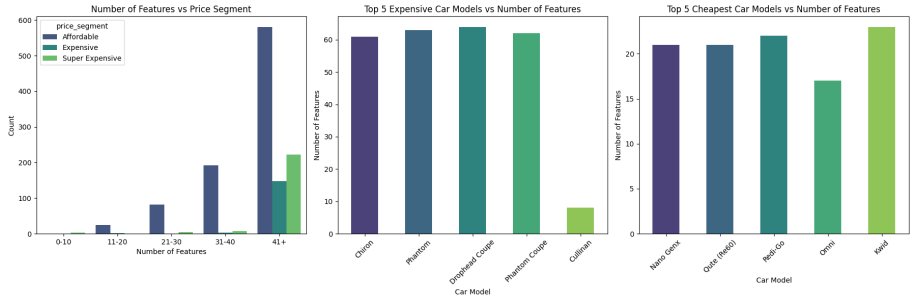
4. Bar Charts

Reference: pd.cut

To explore the relationship between the number of features, price segment, and specific car models. The first chart displays the count of cars based on the number of features and price segment. The second and third charts showcase the top 5 expensive and cheapest car models along with the corresponding number of features. Legends, titles, and axis labels are provided for better interpretation.

```
# Create bins for the number of features
feature_bins = [0, 10, 20, 30, 40, float('inf')]
feature_labels = ['0-10', '11-20', '21-30', '31-40', '41+']
car['feature_bins'] = pd.cut(car['num_features'], bins=feature_bins, labels=feature_labels, right=False)

# Find the top 5 unique car models with the highest ex-showroom prices
top_expensive_models = car.sort_values(by='ex-showroom_price_inr', ascending=False).drop_duplicates(subset='model').h

# Find the top 5 unique car models with the lowest ex-showroom prices
top_cheapest_models = car.sort_values(by='ex-showroom_price_inr', ascending=True).drop_duplicates(subset='model').hea

# Create a 1x3 matrix of subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot the first chart (Number of Features vs Price Segment)
sns.countplot(data=car, x='feature_bins', hue='price_segment', palette='viridis', ax=axes[0])
axes[0].set_title('Number of Features vs Price Segment')
axes[0].set_xlabel('Number of Features')
axes[0].set_ylabel('Count')

# Plot the second chart (Top 5 Expensive Car Models vs Number of Features)
sns.barplot(data=top_expensive_models, x='model', y='num_features', palette='viridis', width=0.5 , ax=axes[1])
axes[1].set_title('Top 5 Expensive Car Models vs Number of Features')
axes[1].set_xlabel('Car Model')
axes[1].set_ylabel('Number of Features')
axes[1].tick_params(axis='x', rotation=45)

# Plot the third chart (Top 5 Cheapest Car Models vs Number of Features)
sns.barplot(data=top_cheapest_models, x='model', y='num_features', palette='viridis', width=0.5, ax=axes[2])
axes[2].set_title('Top 5 Cheapest Car Models vs Number of Features')
axes[2].set_xlabel('Car Model')
axes[2].set_ylabel('Number of Features')
axes[2].tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout()
plt.show()
```

```
<ipython-input-179-762492a13830>:22: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i


<ipython-input-179-762492a13830>:29: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i
```



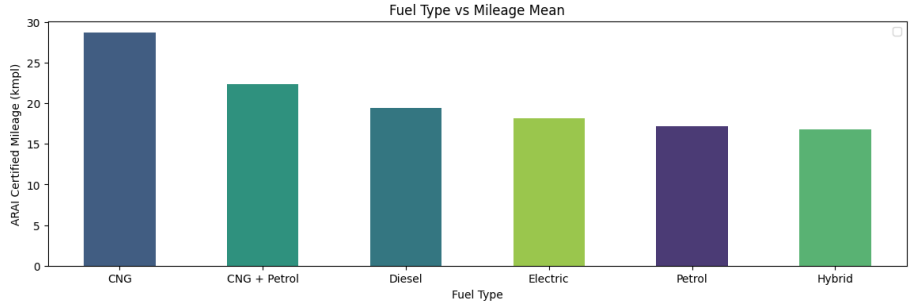
Observation:

```
Seems like that affordable cars provide more features, however super expensive car provides more power.
Chiron provides most feature and cullinan with least feature, being super expensive.
Nano Genx and Kwid are quiet impressive to provide such great features even being cheap.
```

Here, we sort the data by the mean mileage in descending order and create a bar chart to compare the ARAI certified mileage mean across different fuel types.

```
# Sort the data by mean mileage in descending order
sorted_data = car.groupby('fuel_type')['arai_certified_mileage_kmpl'].mean().sort_values(ascending=False).index

# Bar chart with hue for Fuel Type vs Mileage mean
plt.figure(figsize=(14, 4))
sns.barplot(data=car, x='fuel_type', y='arai_certified_mileage_kmpl', hue='fuel_type', ci=False, palette='viridis', c
plt.title('Fuel Type vs Mileage Mean')
plt.xlabel('Fuel Type')
plt.ylabel('ARAI Certified Mileage (kmpl)')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0))
plt.show()
```

```
<ipython-input-180-c20cae0f17d0>:6: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same e

WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
```

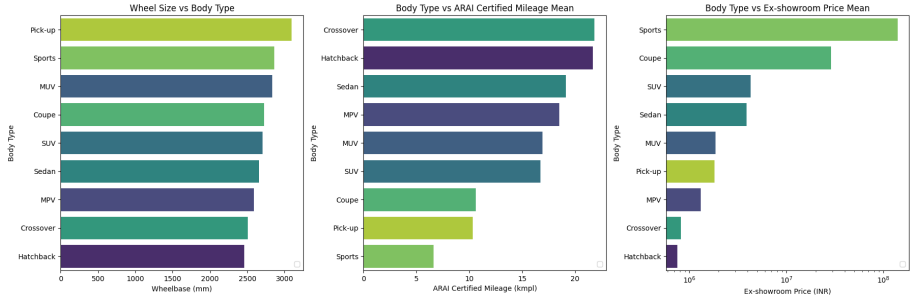


Observation: Some one please call and make Indian Dads undertand to buy CNG cars.

Now we explore the relationships between selected body types, wheelbase, ARAI certified mileage, and ex-showroom price. Three bar charts are created, each focusing on a different aspect and displaying the mean values.

```
# Define the desired body types
desired_body_types = ['Coupe', 'Crossover', 'Hatchback', 'MPV', 'MUV', 'Pick-up', 'Sedan', 'Sports', 'SUV']

# Filter the DataFrame to include only the desired body types
filtered_car = car[car['body_type'].isin(desired_body_types)]

# Order the body types by mean wheelbase in descending order
order_body_types_wheelbase = filtered_car.groupby('body_type')['wheelbase_mm'].mean().sort_values(ascending=False).ir

# Order the body types by mean ARAI Certified Mileage in descending order
order_body_types_mileage = filtered_car.groupby('body_type')['arai_certified_mileage_kmpl'].mean().sort_values(ascenc

# Order the body types by mean Ex-showroom Price in descending order
order_body_types_price = filtered_car.groupby('body_type')['ex-showroom_price_inr'].mean().sort_values(ascending=Fals

# Create a 1x3 matrix for the subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Bar chart with hue for Wheel Size vs Body Type
sns.barplot(ax=axes[0], data=filtered_car, x='wheelbase_mm', y='body_type', hue='body_type', ci=False, palette='viric
axes[0].set_title('Wheel Size vs Body Type')
axes[0].set_xlabel('Wheelbase (mm)')
axes[0].set_ylabel('Body Type')
axes[0].legend(loc='lower right', bbox_to_anchor=(1.0, 0.0))

# Bar chart for Body Type vs ARAI Certified Mileage mean
sns.barplot(ax=axes[1], data=filtered_car, x='arai_certified_mileage_kmpl', y='body_type', hue='body_type', ci=False,
axes[1].set_title('Body Type vs ARAI Certified Mileage Mean')
axes[1].set_xlabel('ARAI Certified Mileage (kmpl)')
axes[1].set_ylabel('Body Type')
axes[1].legend(loc='lower right', bbox_to_anchor=(1.0, 0.0))

# Bar chart for Body Type vs Ex-showroom Price mean with log scale on x-axis
sns.barplot(ax=axes[2], data=filtered_car, x='ex-showroom_price_inr', y='body_type',hue='body_type', ci=False, paletl
axes[2].set_xscale('log')
axes[2].set_title('Body Type vs Ex-showroom Price Mean')
axes[2].set_xlabel('Ex-showroom Price (INR)')
axes[2].set_ylabel('Body Type')
axes[2].legend(loc='lower right', bbox_to_anchor=(1.0, 0.0))

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.show()
```

```
<ipython-input-182-5a4b579485da>:20: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same e


WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
<ipython-input-182-5a4b579485da>:27: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same e


WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
<ipython-input-182-5a4b579485da>:34: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same e


WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
```



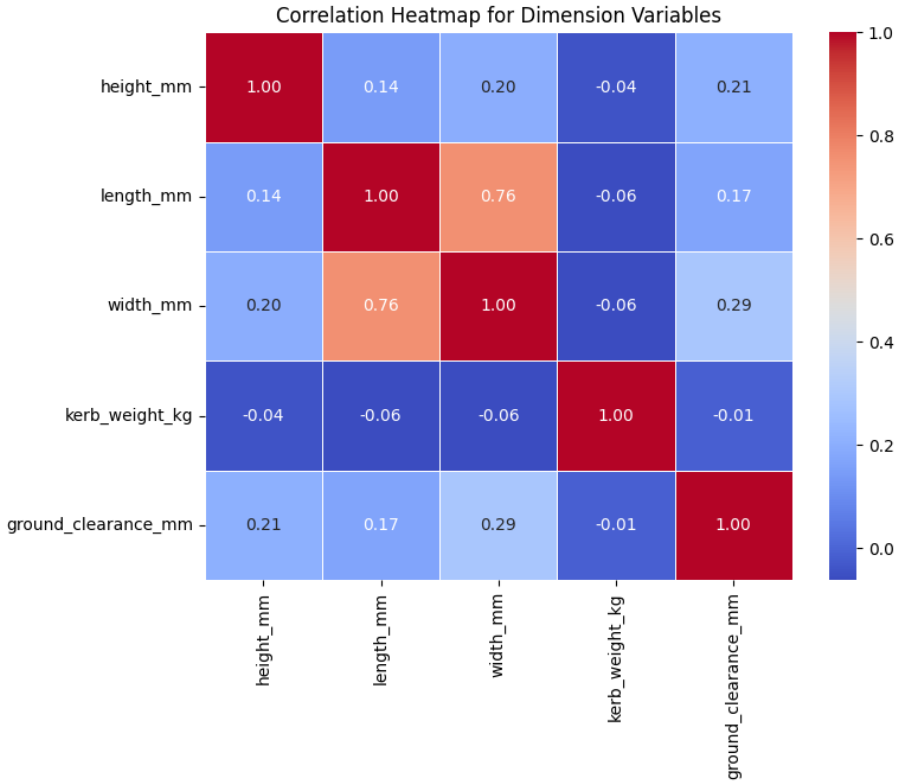
Observation:

```
Hatch back, Crossover and MPV is used for comfort/daily riding, however sport and SUV and pickup are heavy duty cars, that requir
Sports having fastest engine, consumes fiul faster as compared to crossover being used for comfort ride.
Sports car having costliest engine tends to be more expensive.
```

5. Heatmaps

Here, we choose specific dimensions from the 'car' DataFrame and compute the correlation matrix. The resulting heatmap visually represents the correlation coefficients between these dimension variables.

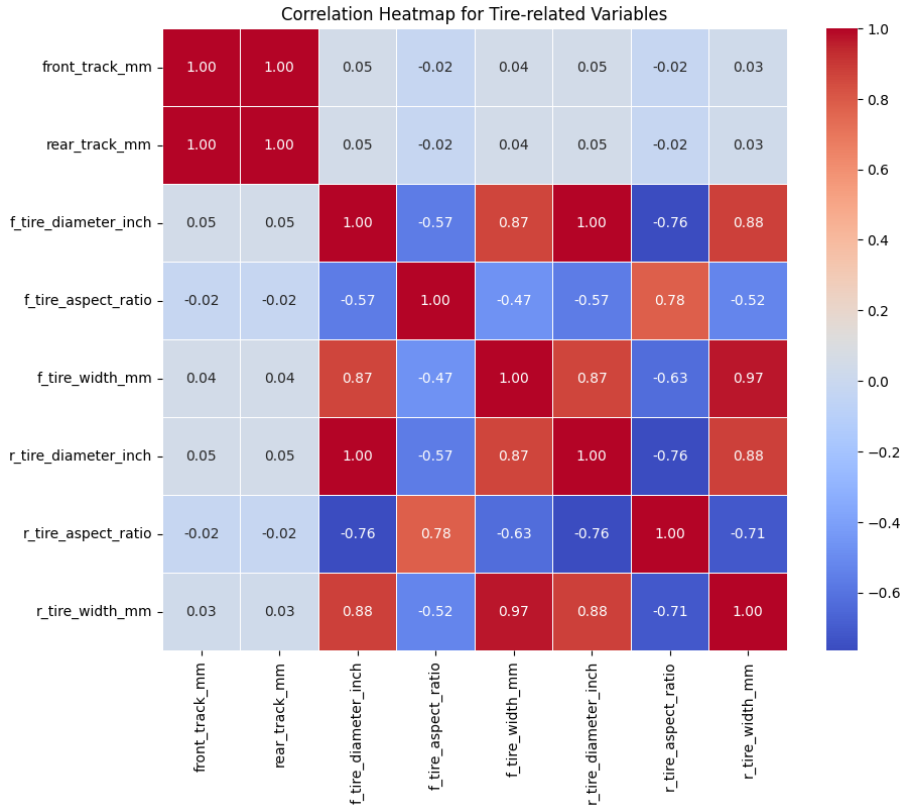
```
# Select the dimensions and calculate the correlation matrix
dimension_subset = car[['height_mm', 'length_mm', 'width_mm', 'kerb_weight_kg', 'ground_clearance_mm']].corr()

# Create a heatmap for the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(dimension_subset, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap for Dimension Variables')
plt.show()
```



Observation: Dimesions of car that is really required

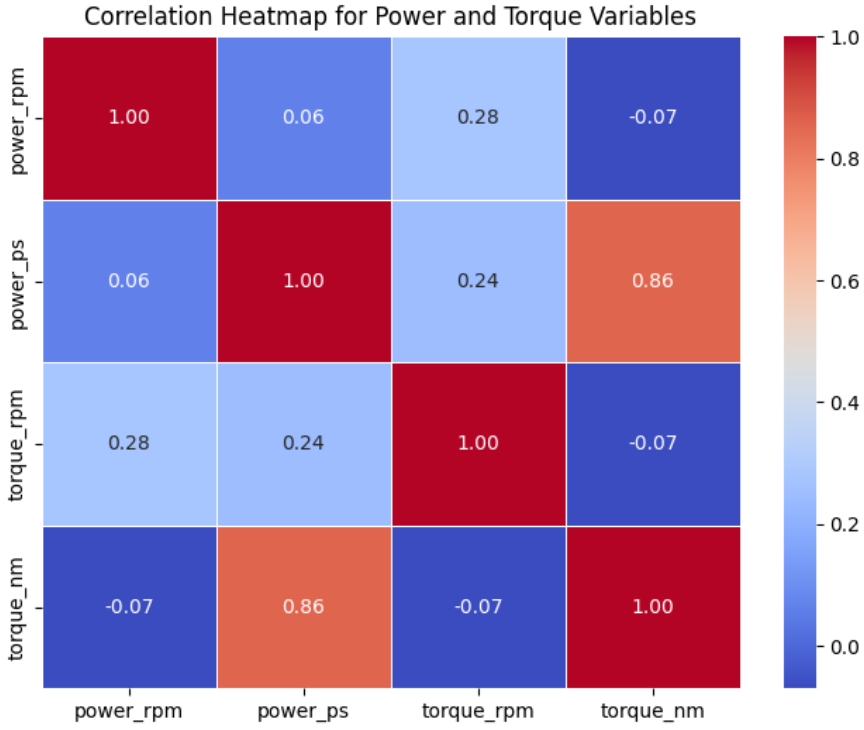
Now we focus on specific tire-related variables by selecting columns from the 'car' DataFrame. The resulting heatmap illustrates the correlation coefficients among these tire-related dimensions, providing insights into their relationships.

```
#Select the specified columns and calculate the correlation matrix
correlation_matrix_tire = car[['front_track_mm', 'rear_track_mm', 'f_tire_diameter_inch', 'f_tire_aspect_ratio',
                               'f_tire_width_mm', 'r_tire_diameter_inch', 'r_tire_aspect_ratio', 'r_tire_width_mm']].

# Create a heatmap for the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_tire, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap for Tire-related Variables')
plt.show()
```



Observation: Tires also matter for the roads full of potholes.

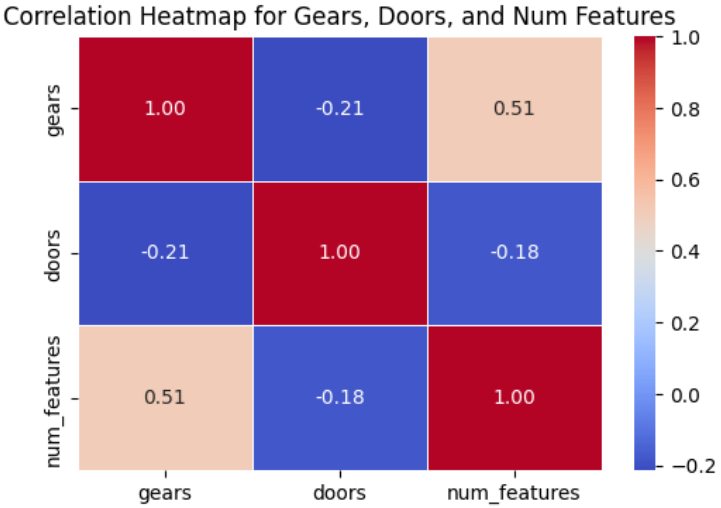
```
# Select the specified columns and calculate the correlation matrix
correlation_matrix_power_torque = car[['power_rpm', 'power_ps', 'torque_rpm', 'torque_nm']].corr()

# Create a heatmap for the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix_power_torque, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap for Power and Torque Variables')
plt.show()
```



Observation: Indian Dads are less interested in this! Why did I find this even? Its a more genz matter.

In this section, we focus on specific power and torque variables by selecting columns from the 'car' DataFrame. The resulting heatmap illustrates the correlation coefficients among these variables, providing insights into their relationships.

```
# Select the specified columns and calculate the correlation matrix
correlation_matrix_gears_doors_features = car[['gears', 'doors', 'num_features']].corr()

# Create a heatmap for the correlation matrix
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix_gears_doors_features, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap for Gears, Doors, and Num Features')
plt.show()
```



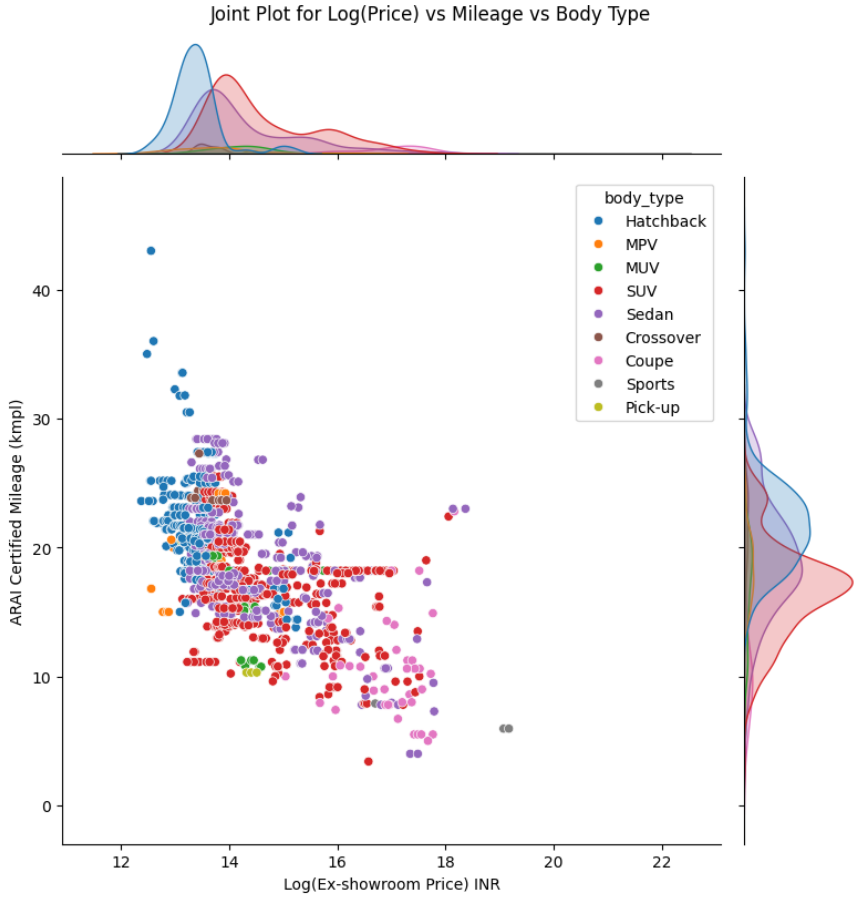
Observation: Yes! Features, door and gears are the important factors for Indians.

6. Joint Plots

Now we explore the relationships between the logarithm of ex-showroom price, ARAI certified mileage, and body type for the desired body types. The joint plot visualizes these correlations.

```
# Define the desired body types
desired_body_types = ['Coupe', 'Crossover', 'Hatchback', 'MPV', 'MUV', 'Pick-up', 'Sedan', 'Sports', 'SUV']

# Filter the DataFrame to include only the desired body types
filtered_car = car[car['body_type'].isin(desired_body_types)]

# Joint plot for Price vs Mileage vs Body type
plt.figure(figsize=(8, 6))
filtered_car['log_price'] = np.log1p(filtered_car['ex-showroom_price_inr'])

joint_plot = sns.jointplot(data=filtered_car, x='log_price', y='arai_certified_mileage_kmpl', hue='body_type', height
joint_plot.set_axis_labels('Log(Ex-showroom Price) INR', 'ARAI Certified Mileage (kmpl)')
joint_plot.fig.suptitle('Joint Plot for Log(Price) vs Mileage vs Body Type', y=1.02)
plt.show()

# Remove the temporary log_price column
filtered_car.drop('log_price', axis=1, inplace=True)
```

```
<ipython-input-187-50831a0a59d1>:9: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do

<Figure size 800x600 with 0 Axes>
```



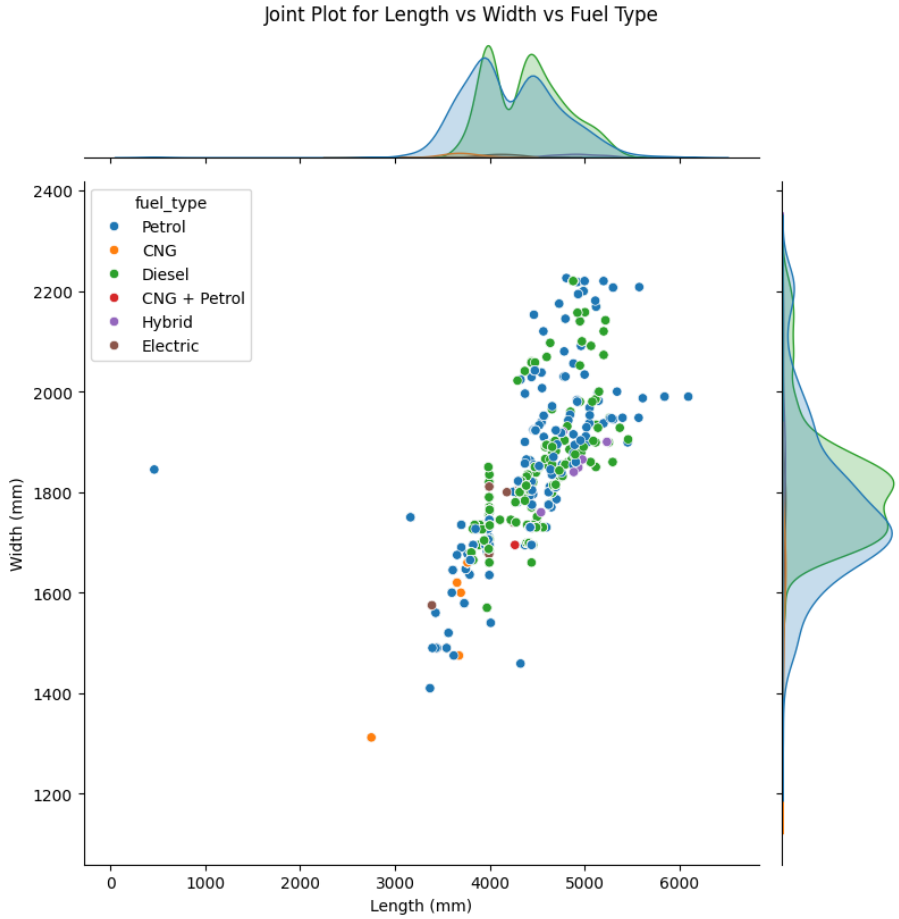

```
<ipython-input-187-50831a0a59d1>:17: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
```

Observation: Gives a brief idea that bigger bodytype consumes more fuel.

Here, we generate a joint plot to explore the relationships between car length and width, considering different fuel types. The plot provides insights into how these dimensions vary across various fuel types.

```
# Joint plot for Length vs Width vs Fuel type
plt.figure(figsize=(12, 8))

joint_plot = sns.jointplot(data=car, x='length_mm', y='width_mm', hue='fuel_type', height=8)
joint_plot.set_axis_labels('Length (mm)', 'Width (mm)')
joint_plot.fig.suptitle('Joint Plot for Length vs Width vs Fuel Type', y=1.02)
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

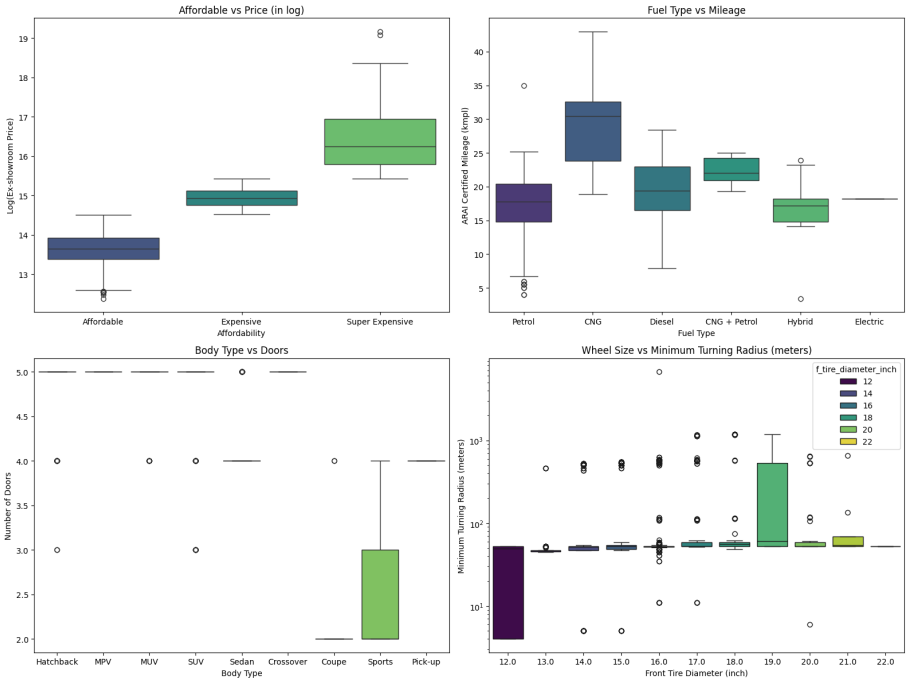


Observation: Gives synopsis about the dimension that requires to carry various types of fuel.

7. Box Plots

Here, we use a 2x2 matrix of subplots to compare different aspects of car data. We explore affordability vs price, fuel type vs mileage, body type vs doors, and wheel size vs minimum turning radius.

```
# Create a 2x2 matrix of subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16, 12))

desired_body_types = ['Coupe', 'Crossover', 'Hatchback', 'MPV', 'MUV', 'Pick-up', 'Sedan', 'Sports', 'SUV']

# Filter the DataFrame to include only the desired body types
filtered_car = car[car['body_type'].isin(desired_body_types)]

# Order the body types by mean wheelbase in descending order
order_body_types_wheelbase = filtered_car.groupby('body_type')['wheelbase_mm'].mean().sort_values(ascending=False).ir

# 1. Affordable vs Price (in log)
sns.boxplot(data=car, x='price_segment', y=np.log1p(car['ex-showroom_price_inr']), hue='price_segment', palette='viri
axes[0, 0].set_title('Affordable vs Price (in log)')
axes[0, 0].set_xlabel('Affordability')
axes[0, 0].set_ylabel('Log(Ex-showroom Price)')

# 2. Fuel Type vs Mileage
sns.boxplot(data=car, x='fuel_type', y='arai_certified_mileage_kmpl', hue='fuel_type', palette='viridis', dodge=False
axes[0, 1].set_title('Fuel Type vs Mileage')
axes[0, 1].set_xlabel('Fuel Type')
axes[0, 1].set_ylabel('ARAI Certified Mileage (kmpl)')

# 3. Body Type vs Doors (corrected line)
sns.boxplot(data=filtered_car, x='body_type', y='doors', hue='body_type', palette='viridis', dodge=False, ax=axes[1,
axes[1, 0].set_title('Body Type vs Doors')
axes[1, 0].set_xlabel('Body Type')
axes[1, 0].set_ylabel('Number of Doors')

# 4. Wheel Size vs minimum_turning_radius_meter (with y-axis in log scale)
sns.boxplot(data=car, x='f_tire_diameter_inch', y='minimum_turning_radius_meter', hue='f_tire_diameter_inch', palette
axes[1, 1].set_title('Wheel Size vs Minimum Turning Radius (meters)')
axes[1, 1].set_xlabel('Front Tire Diameter (inch)')
axes[1, 1].set_ylabel('Minimum Turning Radius (meters)')
axes[1, 1].set_yscale('log')

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



Observation:

```
Price segment provide crisp data about the affordability of any car.
All segments of fuel type tend to have same mileage. (less outliers)
Number of doors are more or less same for every bodytype car in India, however there are 4 door super cars along with 2 door supe
Tire size seems to have same turning radius except for some outliers.
```

8. Categorical Plots

Here, we use a 3x2 matrix of subplots to display categorical plots. The plots include counts of cars by body type, number of doors, number of gears, affordability vs price, fuel type, and engine type.