In [28]:
```
# 1. Import all the required Python Libraries
import pandas as pd
import numpy as np
```

In [28]:
```
# 1. Import all the required Python Libraries
import pandas as pd
import numpy as np
```

In [29]:
```
# 2. Dataset Description

description = """
Dataset Name: Automobile Dataset
Source: https://www.kaggle.com/datasets/toramky/automobile-dataset

This dataset contains various characteristics of cars such as fuel type, number of doors,
engine size, horsepower, and more. It is commonly used for machine learning projects
like regression, classification, and price prediction.

You uploaded the dataset locally as 'autodata.csv'.
"""
print(description)
```

```
Dataset Name: Automobile Dataset
Source: https://www.kaggle.com/datasets/toramky/automobile-dataset

This dataset contains various characteristics of cars such as fuel type, number of doors,
engine size, horsepower, and more. It is commonly used for machine learning projects
like regression, classification, and price prediction.

You uploaded the dataset locally as 'autodata.csv'.
```

In [31]:
```
# 3. Load the Dataset into pandas dataframe
df = pd.read_csv("C:/Users/prajw/Desktop/Indexs/DSBDA print/GROUP A/Assignment 1 (Data Wranglin I)/autodata.csv")
df.head()
```

Out[31]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	compression-ratio	horsepower	pe r
0	0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	...	9.0	111.0	500
1	1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	...	9.0	111.0	500
2	2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	...	9.0	154.0	500
3	3	2	164	audi	std	four	sedan	fwd	front	99.8	...	10.0	102.0	550
4	4	2	164	audi	std	four	sedan	4wd	front	99.4	...	8.0	115.0	550

5 rows × 30 columns

In [ ]:
```
# 4. Data Preprocessing
```

In [33]:
```
# Check for missing values
print("Missing values per column:\n", df.isnull().sum())
```

```
Missing values per column:
 Unnamed: 0           0
symboling             0
normalized-losses     0
make                  0
aspiration            0
num-of-doors          0
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  0
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 0
city-L/100km          0
horsepower-binned     2
diesel                0
gas                   0
dtype: int64
```

In [34]:
```
# Get initial statistics
print("\nDescriptive statistics:\n", df.describe(include="all"))
```

```
Descriptive statistics:
        Unnamed: 0   symboling  normalized-losses    make aspiration  \
count   201.000000  201.000000          201.00000     201        201
unique         NaN         NaN                NaN      22          2
top            NaN         NaN                NaN  toyota        std
freq           NaN         NaN                NaN      32        165
mean    100.000000    0.840796          122.00000     NaN        NaN
std      58.167861    1.254802           31.99625     NaN        NaN
min       0.000000   -2.000000           65.00000     NaN        NaN
25%      50.000000    0.000000          101.00000     NaN        NaN
50%     100.000000    1.000000          122.00000     NaN        NaN
75%     150.000000    2.000000          137.00000     NaN        NaN
max     200.000000    3.000000          256.00000     NaN        NaN

       num-of-doors body-style drive-wheels engine-location  wheel-base  ...  \
count           201        201          201             201  201.000000  ...
unique            2          5            3               2         NaN  ...
top            four      sedan          fwd           front         NaN  ...
freq            115         94          118             198         NaN  ...
mean            NaN        NaN          NaN             NaN   98.797015  ...
std             NaN        NaN          NaN             NaN    6.066366  ...
min             NaN        NaN          NaN             NaN   86.600000  ...
25%             NaN        NaN          NaN             NaN   94.500000  ...
50%             NaN        NaN          NaN             NaN   97.000000  ...
75%             NaN        NaN          NaN             NaN  102.400000  ...
max             NaN        NaN          NaN             NaN  120.900000  ...

        compression-ratio  horsepower     peak-rpm    city-mpg highway-mpg  \
count          201.000000  199.000000   199.000000  201.000000  201.000000
unique                NaN         NaN          NaN         NaN         NaN
top                   NaN         NaN          NaN         NaN         NaN
freq                  NaN         NaN          NaN         NaN         NaN
mean            10.164279  103.396985  5117.587940   25.179104   30.686567
std              4.004965   37.553843   480.521824    6.423220    6.815150
min              7.000000   48.000000  4150.000000   13.000000   16.000000
25%              8.600000   70.000000  4800.000000   19.000000   25.000000
50%              9.000000   95.000000  5200.000000   24.000000   30.000000
75%              9.400000  116.000000  5500.000000   30.000000   34.000000
max             23.000000  262.000000  6600.000000   49.000000   54.000000

               price  city-L/100km horsepower-binned      diesel         gas
count     201.000000    201.000000               199  201.000000  201.000000
unique           NaN           NaN                 3         NaN         NaN
top              NaN           NaN               Low         NaN         NaN
freq             NaN           NaN               151         NaN         NaN
mean    13207.129353      9.944145               NaN    0.099502    0.900498
std      7947.066342      2.534599               NaN    0.300083    0.300083
min      5118.000000      4.795918               NaN    0.000000    0.000000
25%      7775.000000      7.833333               NaN    0.000000    1.000000
50%     10295.000000      9.791667               NaN    0.000000    1.000000
75%     16500.000000     12.368421               NaN    0.000000    1.000000
max     45400.000000     18.076923               NaN    1.000000    1.000000

[11 rows x 30 columns]
```

In [35]:
```python
# Check dimensions
print("\nShape of the dataset:", df.shape)
```

```
Shape of the dataset: (201, 30)
```

In [36]:
```python
# Column names
print("\nColumns in dataset:\n", df.columns)
```

```
Columns in dataset:
 Index(['Unnamed: 0', 'symboling', 'normalized-losses', 'make', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price', 'city-L/100km', 'horsepower-binned', 'diesel',
       'gas'],
      dtype='object')
```

In [39]:
```python
# 5. Data Formatting and Normalization

# Check data types of each column
print("\nData types before conversion:\n", df.dtypes)

# Example conversions: ensure numeric columns are numeric
# (force conversion if needed using errors='coerce')
df["price"] = pd.to_numeric(df["price"], errors='coerce')
df["horsepower"] = pd.to_numeric(df["horsepower"], errors='coerce')
df["peak-rpm"] = pd.to_numeric(df["peak-rpm"], errors='coerce')

# Fill missing values with mean
df["price"] = df["price"].fillna(df["price"].mean())
df["horsepower"] = df["horsepower"].fillna(df["horsepower"].mean())
df["peak-rpm"] = df["peak-rpm"].fillna(df["peak-rpm"].mean())
```

```
# Fill 'num-of-doors' missing values with mode
df["num-of-doors"] = df["num-of-doors"].fillna(df["num-of-doors"].mode()[0])

# Drop rows where target or crucial column is still missing (if any)
df.dropna(subset=["price"], axis=0, inplace=True)

print("\nData types after conversion:\n", df.dtypes)
```

```
Data types before conversion:
 Unnamed: 0             int64
symboling               int64
normalized-losses       int64
make                   object
aspiration             object
num-of-doors           object
body-style             object
drive-wheels           object
engine-location        object
wheel-base            float64
length                float64
width                 float64
height                float64
curb-weight             int64
engine-type            object
num-of-cylinders       object
engine-size             int64
fuel-system            object
bore                  float64
stroke                float64
compression-ratio     float64
horsepower            float64
peak-rpm              float64
city-mpg                int64
highway-mpg             int64
price                 float64
city-L/100km          float64
horsepower-binned      object
diesel                  int64
gas                     int64
dtype: object

Data types after conversion:
 Unnamed: 0             int64
symboling               int64
normalized-losses       int64
make                   object
aspiration             object
num-of-doors           object
body-style             object
drive-wheels           object
engine-location        object
wheel-base            float64
length                float64
width                 float64
height                float64
curb-weight             int64
engine-type            object
num-of-cylinders       object
engine-size             int64
fuel-system            object
bore                  float64
stroke                float64
compression-ratio     float64
horsepower            float64
peak-rpm              float64
city-mpg                int64
highway-mpg             int64
price                 float64
city-L/100km          float64
horsepower-binned      object
diesel                  int64
gas                     int64
dtype: object
```

In [40]:
```
# 6. Turn categorical variables into quantitative variables

# Select object (categorical) columns
categorical_columns = df.select_dtypes(include=['object']).columns
print("Categorical columns:\n", categorical_columns)

# One-hot encode categorical columns
df_encoded = pd.get_dummies(df, columns=categorical_columns)
df_encoded.head()
```

```
Categorical columns:
 Index(['make', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels',
       'engine-location', 'engine-type', 'num-of-cylinders', 'fuel-system',
       'horsepower-binned'],
      dtype='object')
```

Out[40]:

	Unnamed: 0	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	...	fuel-system_2bbl	fuel-system_4bbl	fuel system_id
0	0	3	122	88.6	0.811148	0.890278	48.8	2548	130	3.47	...	False	False	Fals
1	1	3	122	88.6	0.811148	0.890278	48.8	2548	130	3.47	...	False	False	Fals
2	2	1	122	94.5	0.822681	0.909722	52.4	2823	152	2.68	...	False	False	Fals
3	3	2	164	99.8	0.848630	0.919444	54.3	2337	109	3.19	...	False	False	Fals
4	4	2	164	99.4	0.848630	0.922222	54.3	2824	136	3.19	...	False	False	Fals

5 rows × 80 columns

In [41]:
```
# Save final cleaned and encoded dataset (optional)
df_encoded.to_csv("cleaned_autodata.csv", index=False)
print("✅ Cleaned dataset saved as 'cleaned_autodata.csv'")
```

```
✅ Cleaned dataset saved as 'cleaned_autodata.csv'
```

In [ ]: