MACHINE LEARNING MODULE PROJECT

Problem Description:

A Chinese automobile company aspires to enter the US market by setting up their manufacturing unit there and producing cars locally to give competition to their US and European counterparts. They have contracted an automobile consulting company to understand the factors on which the pricing of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the American market, since those may be very different from the Chinese market. Essentially, the company wants to know:

- **Which variables are significant in predicting the price of a car**
- **How well those variables describe the price of a car**

Based on various market surveys, the consulting firm has gathered a large dataset of different types of cars across the American market.

Dataset: https://drive.google.com/file/d/1FHmYNLs9v0Enc-UExEMpitOFGsWvB2dP/view?usp=drive_link

Business Goal:

You are required to model the price of cars with the available independent variables. It will be used by the management to understand how exactly the prices vary with the independent variables. They can accordingly manipulate the design of the cars, the business strategy etc. to meet certain price levels. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

Loading and Pre-processing:

- **Load the dataset and perform necessary preprocessing steps.**

In [1]:
```
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

In [3]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv("CarPrice_Assignment.csv")
df
```

Out[3]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...
...	...	...	...	...	...	...	...	...	...	...	...
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...

205 rows × 26 columns

In [5]:
```
# Inspecting the dataset
print("Dataset Shape:", df.shape)
print("First 5 Rows:\n", df.head())
print("Data Types:\n", df.dtypes)
```

```
Dataset Shape: (205, 26)
First 5 Rows:
    car_ID  symboling                   CarName fueltype aspiration doornumber  \
0       1          3        alfa-romero giulia      gas        std        two
1       2          3       alfa-romero stelvio      gas        std        two
2       3          1  alfa-romero Quadrifoglio      gas        std        two
3       4          2               audi 100 ls      gas        std       four
4       5          2                audi 100ls      gas        std       four

       carbody drivewheel enginelocation  wheelbase  ...  enginesize  \
0  convertible        rwd          front       88.6  ...         130
1  convertible        rwd          front       88.6  ...         130
2    hatchback        rwd          front       94.5  ...         152
3        sedan        fwd          front       99.8  ...         109
4        sedan        4wd          front       99.4  ...         136

   fuelsystem  boreratio  stroke compressionratio horsepower  peakrpm citympg  \
0        mpfi       3.47    2.68              9.0        111     5000      21
1        mpfi       3.47    2.68              9.0        111     5000      21
2        mpfi       2.68    3.47              9.0        154     5000      19
3        mpfi       3.19    3.40             10.0        102     5500      24
4        mpfi       3.19    3.40              8.0        115     5500      18

   highwaympg    price
0          27  13495.0
1          27  16500.0
2          26  16500.0
3          30  13950.0
4          22  17450.0

[5 rows x 26 columns]
Data Types:
 car_ID               int64
symboling             int64
CarName              object
fueltype             object
aspiration           object
doornumber           object
carbody              object
drivewheel           object
enginelocation       object
wheelbase           float64
```

```
carlength           float64
carwidth            float64
carheight           float64
curbweight            int64
enginetype           object
cylindernumber       object
enginesize            int64
fuelsystem           object
boreratio           float64
stroke              float64
compressionratio    float64
horsepower            int64
peakrpm               int64
citympg               int64
highwaympg            int64
price               float64
dtype: object
```

In [7]:
```
print("Feature properties of the dataset...")
print("\t")
df.info()
```

```
Feature properties of the dataset...

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [9]:
```
print("Statistical Analysis of the dataset...")
print("\t")
df.describe()
```

```
Statistical Analysis of the dataset...
```

Out[9]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000

In [11]:

```
print("Checking for missing values...")
print(df.isnull().sum())
```

```
Checking for missing values...
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

In [13]:
```
print("Checking duplicate values in the dataset...")
df.duplicated().sum()
```

```
Checking duplicate values in the dataset...
```

Out[13]: 0

In [15]:
```
# Checking variable named CarName (now 'carname') which is comprised of two parts - the first word is the name of 'ca
df.CarName.head()
```

Out[15]:
```
0          alfa-romero giulia
1         alfa-romero stelvio
2    alfa-romero Quadrifoglio
3                 audi 100 ls
4                  audi 100ls
Name: CarName, dtype: object
```

In [17]:
```
# Retaining only the 'car company' (first half) under a new column 'name' and dropping off the 'carname' column
df['name'] = df['CarName'].apply(lambda x: x.split(' ')[0])
df = df.drop('CarName', axis=1)
df.head()
```

Out[17]:

	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	fuel
0	1	3	gas	std	two	convertible	rwd	front	88.6	168.8	...	
1	2	3	gas	std	two	convertible	rwd	front	88.6	168.8	...	
2	3	1	gas	std	two	hatchback	rwd	front	94.5	171.2	...	
3	4	2	gas	std	four	sedan	fwd	front	99.8	176.6	...	
4	5	2	gas	std	four	sedan	4wd	front	99.4	176.6	...	

5 rows × 26 columns

In [23]:
```
# Converting all the entries of 'name' to lower case and checking unique entries of 'name'
df.name = df.name.str.lower()
df.name.unique()
```

Out[23]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

In [25]:
```
# Correcting the irregularities in the names of unique entries of 'companyname'
'''
defining a 'name_rep' function to expedite the process of replacement
'a' is the old value, 'b' is the replaced (corrected) value
'''
def name_rep(df,a,b):
    return df.name.replace(a,b, inplace=True)

name_rep(df,'maxda','mazda')
name_rep(df,'porcshce','porsche')
name_rep(df,'toyouta','toyota')
name_rep(df,'vokswagen','volkswagen')
name_rep(df,'vw','volkswagen')
name_rep(df,'alfa-romero','alfa-romeo')

print(df.name.unique())
print("\nNumber of unique car companies: ",df.name.nunique())
```

```
['alfa-romeo' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
 'mazda' 'buick' 'mercury' 'mitsubishi' 'nissan' 'peugeot' 'plymouth'
 'porsche' 'renault' 'saab' 'subaru' 'toyota' 'volkswagen' 'volvo']

Number of unique car companies:  22
```

In [27]:
```
df.head()
```

Out[27]:

	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	fuel
0	1	3	gas	std	two	convertible	rwd	front	88.6	168.8	...	
1	2	3	gas	std	two	convertible	rwd	front	88.6	168.8	...	
2	3	1	gas	std	two	hatchback	rwd	front	94.5	171.2	...	
3	4	2	gas	std	four	sedan	fwd	front	99.8	176.6	...	
4	5	2	gas	std	four	sedan	4wd	front	99.4	176.6	...	

5 rows × 26 columns

In [29]:
```
df.shape
```

Out[29]: (205, 26)

Univariate Analysis

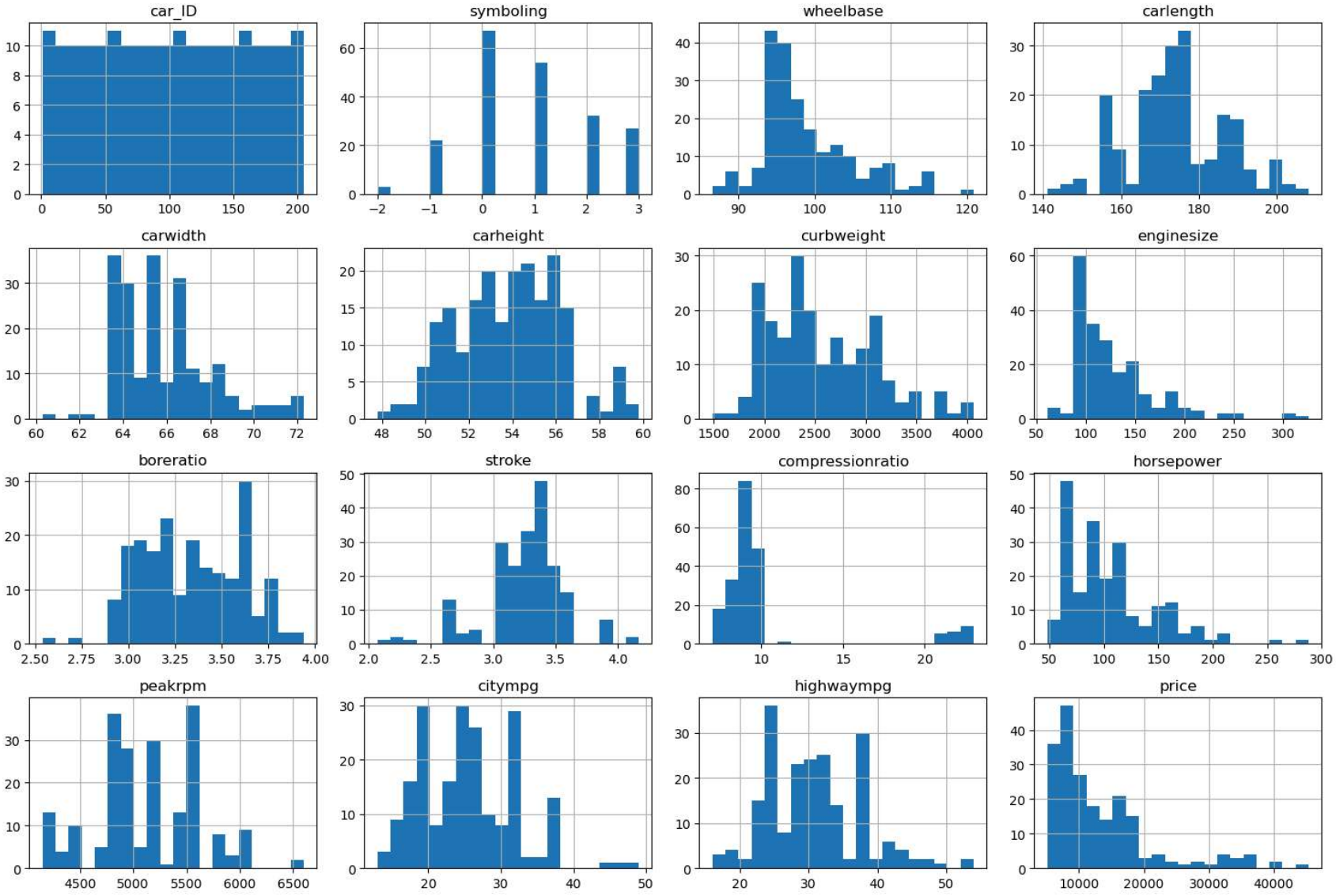
In [32]:
```
# Value counts for categorical columns
for col in df.select_dtypes(include='object').columns:
    print(f"Value counts for {col}:\n{df[col].value_counts()}\n")
```

```
Value counts for fueltype:
fueltype
gas       185
diesel     20
Name: count, dtype: int64

Value counts for aspiration:
aspiration
std      168
turbo     37
Name: count, dtype: int64

Value counts for doornumber:
doornumber
four    115
two      90
Name: count, dtype: int64

Value counts for carbody:
carbody
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: count, dtype: int64

Value counts for drivewheel:
drivewheel
fwd    120
rwd     76
4wd      9
Name: count, dtype: int64

Value counts for enginelocation:
enginelocation
front    202
rear       3
Name: count, dtype: int64

Value counts for enginetype:
enginetype
```

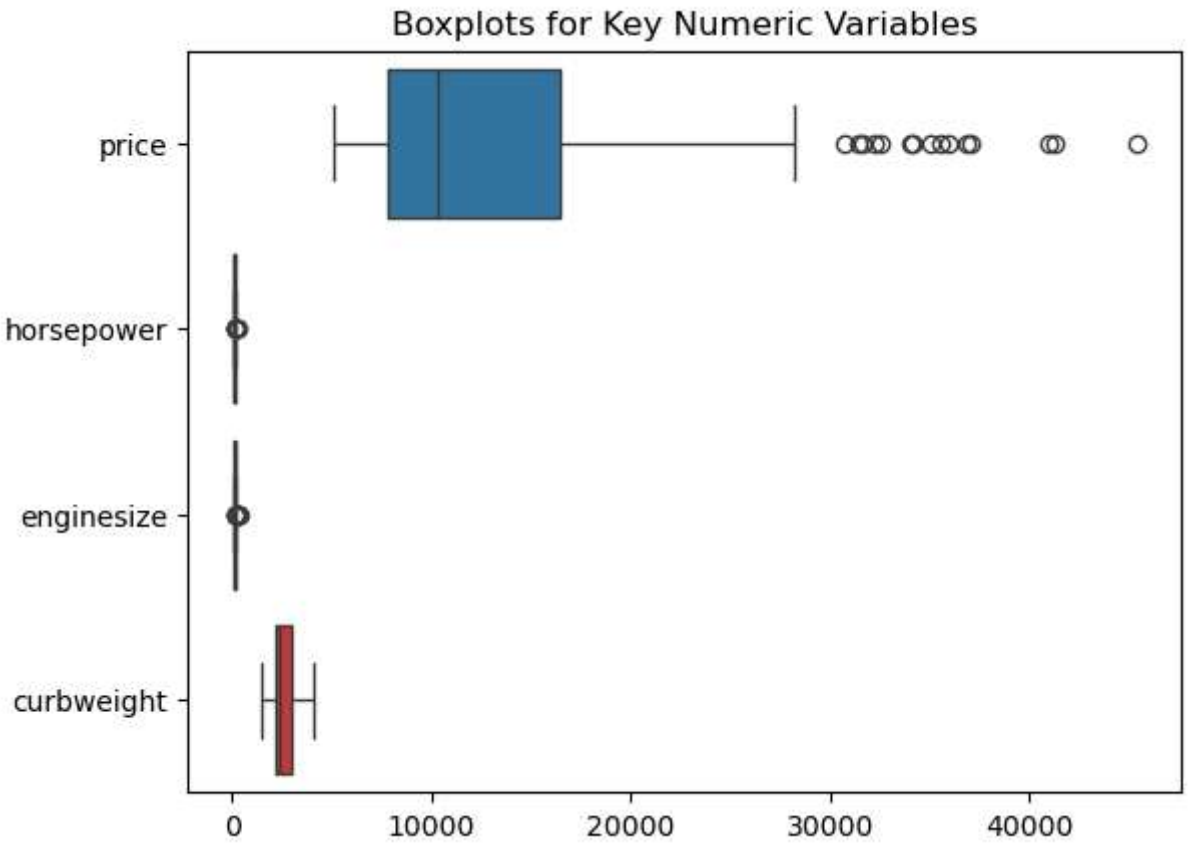
```
ohc      148
ohcf      15
ohcv      13
dohc      12
l         12
rotor      4
dohcv      1
Name: count, dtype: int64

Value counts for cylindernumber:
cylindernumber
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: count, dtype: int64

Value counts for fuelsystem:
fuelsystem
mpfi    94
2bbl    66
idi     20
1bbl    11
spdi     9
4bbl     3
mfi      1
spfi     1
Name: count, dtype: int64

Value counts for name:
name
toyota        32
nissan        18
mazda         17
mitsubishi    13
honda         13
volkswagen    12
subaru        12
peugeot       11
```

```
volvo         11
dodge          9
buick          8
bmw            8
audi           7
plymouth       7
saab           6
porsche        5
isuzu          4
jaguar         3
chevrolet      3
alfa-romeo     3
renault        2
mercury        1
Name: count, dtype: int64
```

In [34]:
```python
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Histograms for numeric features
df.hist(figsize=(15, 10), bins=20)
plt.tight_layout()
plt.show()
```

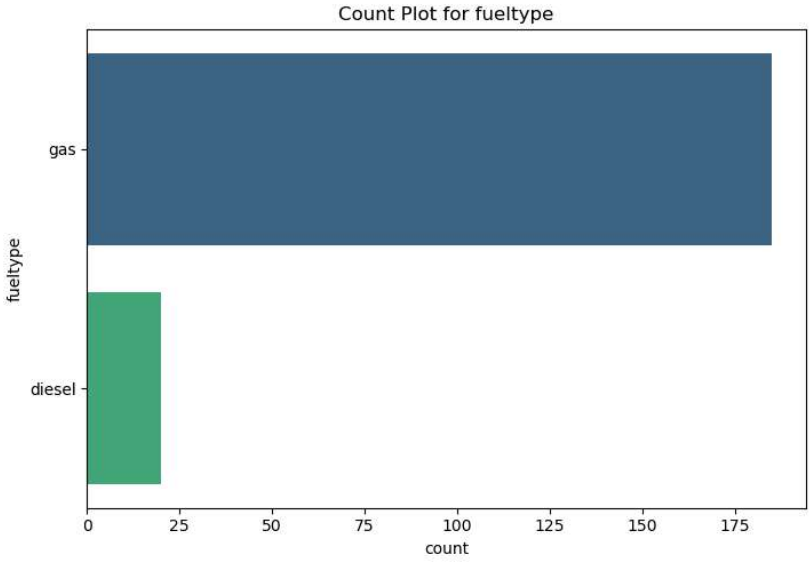


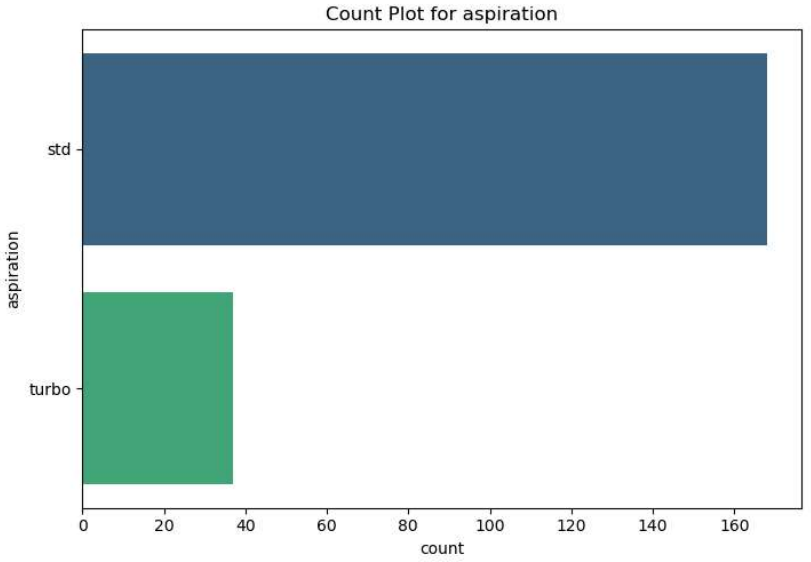
In [36]:
```
# Boxplots for price and numeric features
sns.boxplot(data=df[['price', 'horsepower', 'enginesize', 'curbweight']], orient='h')
plt.title("Boxplots for Key Numeric Variables")
plt.show()
```

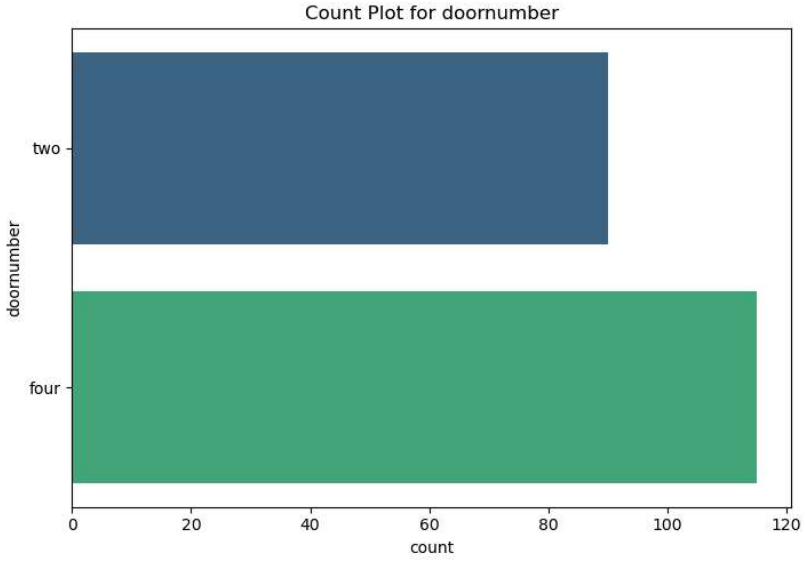


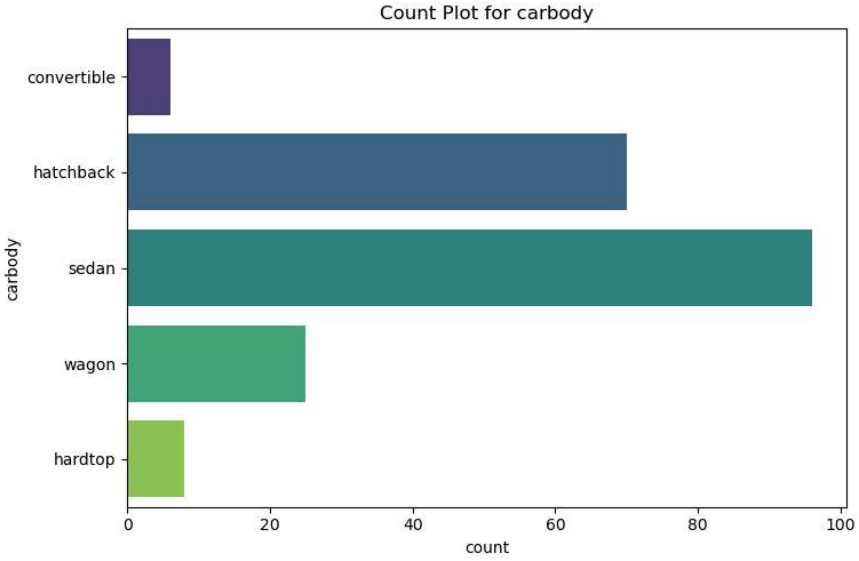
In [42]:
```python
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Create subplots for bar plots of categorical variables
categorical_cols = df.select_dtypes(include='object').columns
n_cols = 2  # Number of columns for the subplot grid
n_rows = (len(categorical_cols) + 1) // n_cols  # Calculate the number of rows needed

fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))
axes = axes.flatten()  # Flatten the 2D axes array into 1D for easier indexing

for i, col in enumerate(categorical_cols):
    sns.countplot(y=col, data=df, palette='viridis', ax=axes[i])
    axes[i].set_title(f"Count Plot for {col}")
```

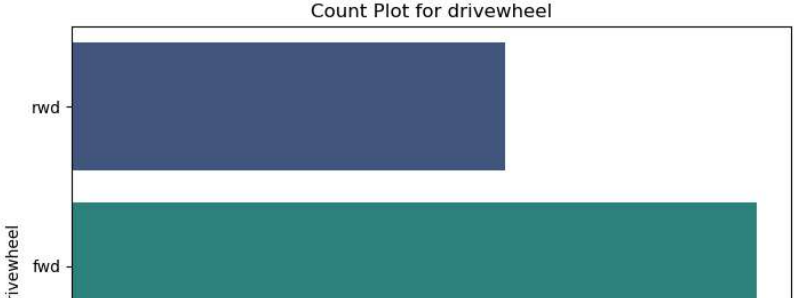
```
# Remove unused subplots
for j in range(len(categorical_cols), len(axes)):
    fig.delaxes(axes[j])

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

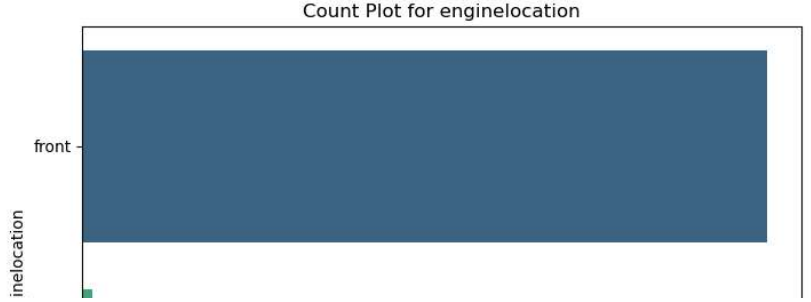

Count Plot for fueltype
gas
diesel
fueltype
0
25
50
75
100
125
150
175
count

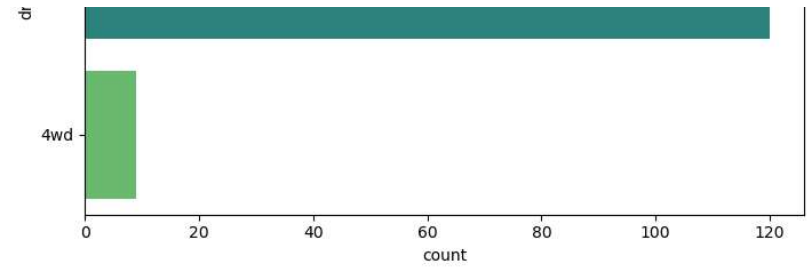

Count Plot for aspiration
std
turbo
aspiration
0
20
40
60
80
100
120
140
160
count

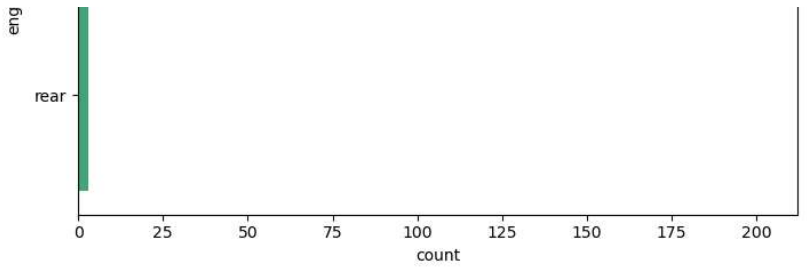

Count Plot for doornumber
two
four
doornumber
0
20
40
60
80
100
120
count

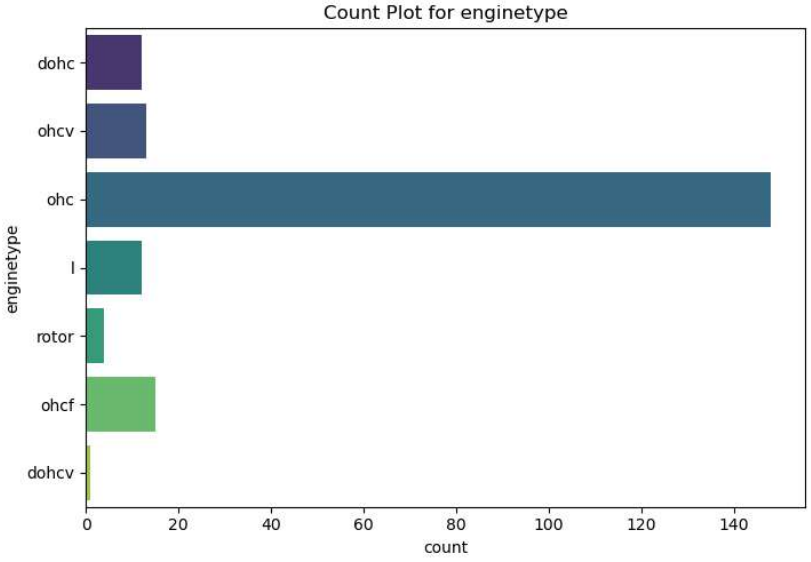

Count Plot for carbody
convertible
hatchback
sedan
wagon
hardtop
carbody
0
20
40
60
80
100
count

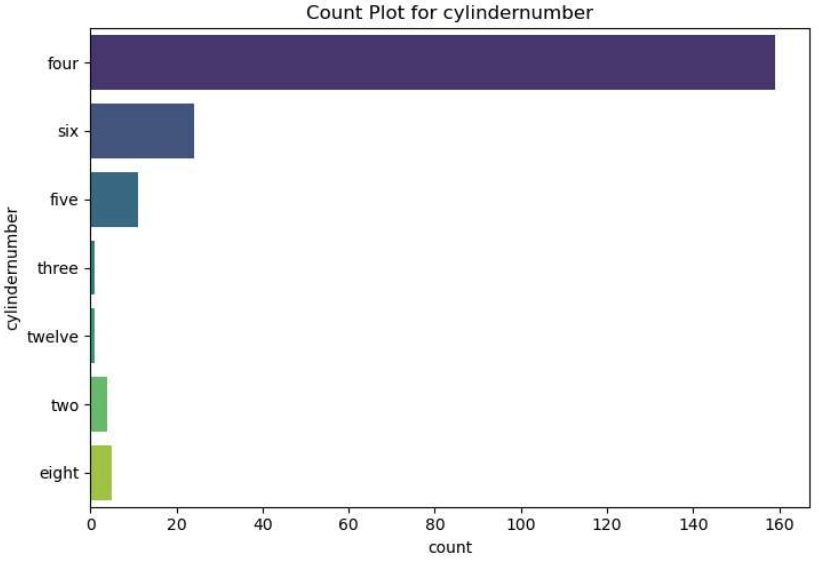

Count Plot for drivewheel
rwd
fwd

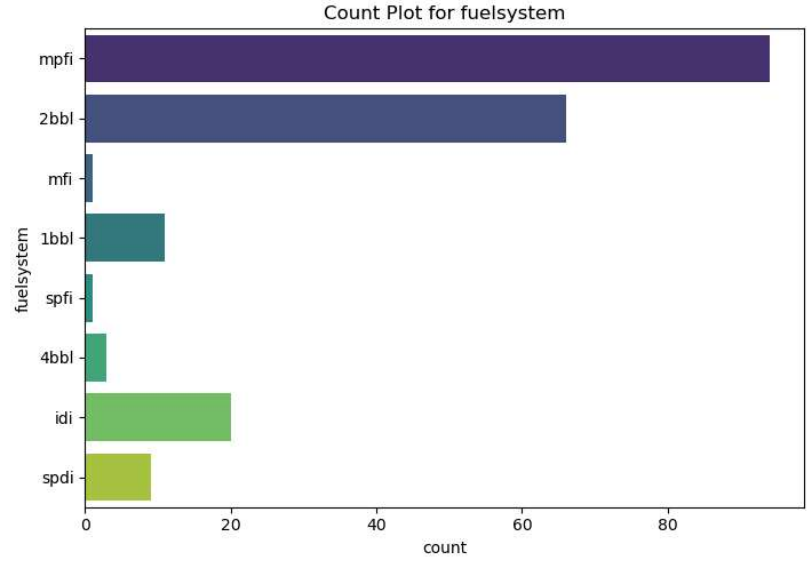

Count Plot for enginelocation
front

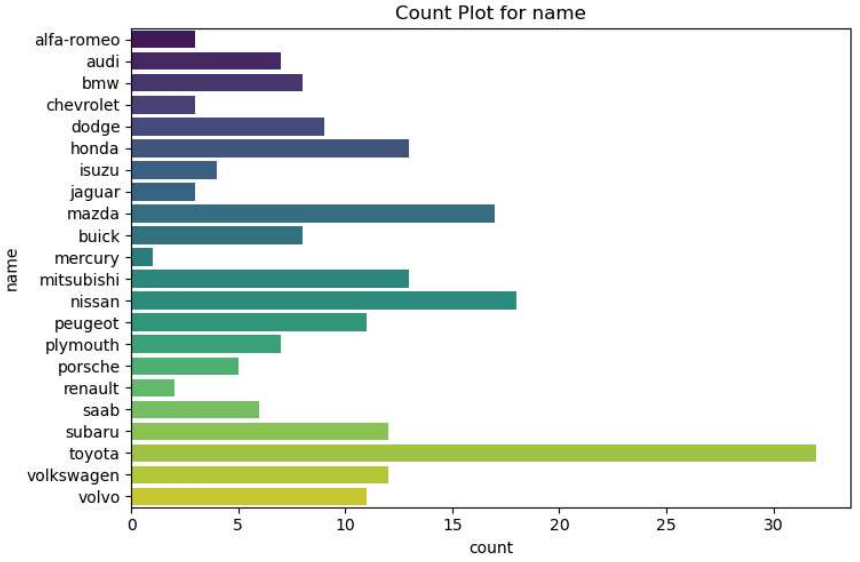

dr
4wd
0
20
40
60
80
100
120
count


eng
rear
0
25
50
75
100
125
150
175
200
count


Count Plot for enginetype
enginetype
dohc
ohcv
ohc
l
rotor
ohcf
dohcv
0
20
40
60
80
100
120
140
count

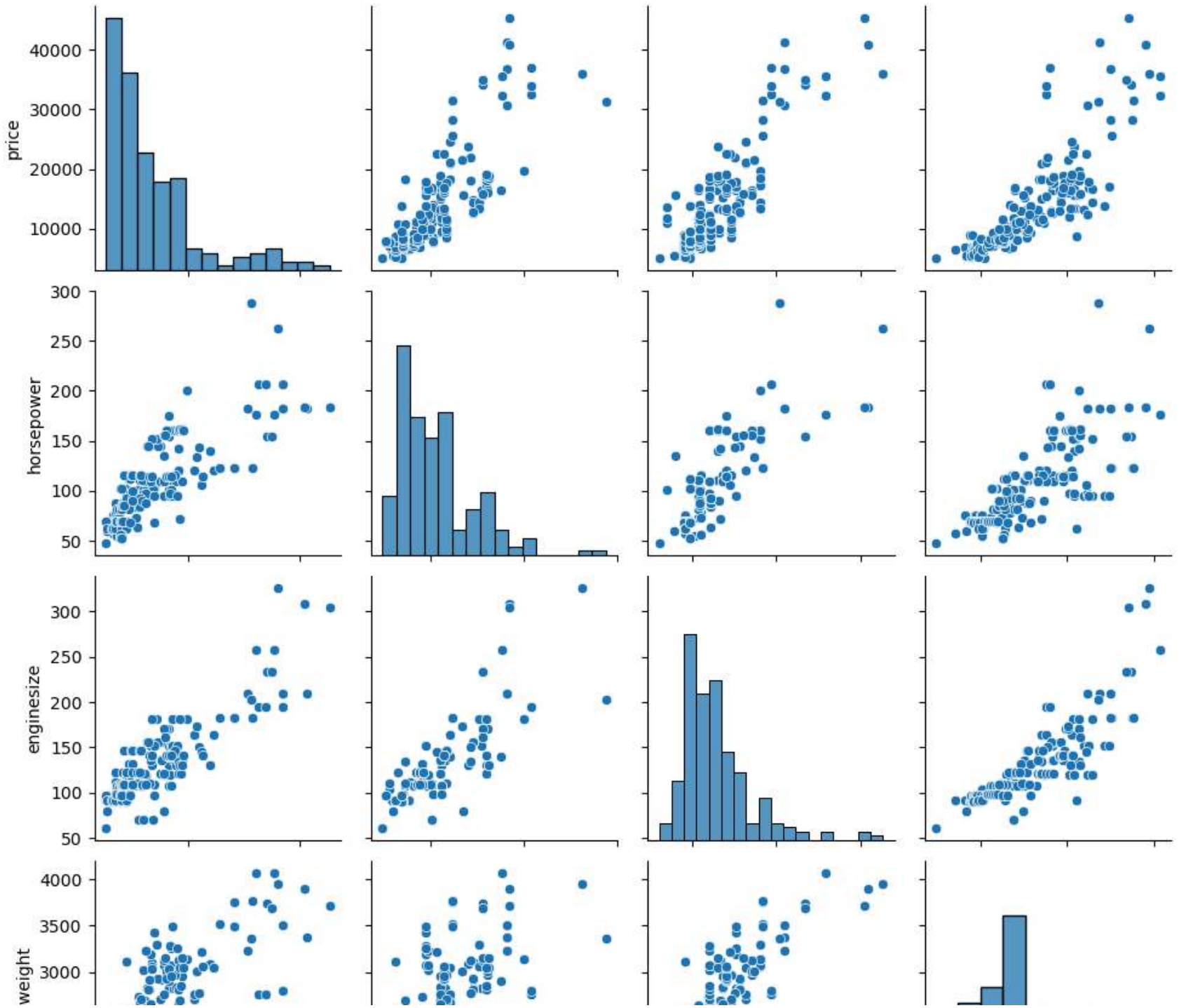

Count Plot for cylindernumber
cylindernumber
four
six
five
three
twelve
two
eight
0
20
40
60
80
100
120
140
160
count

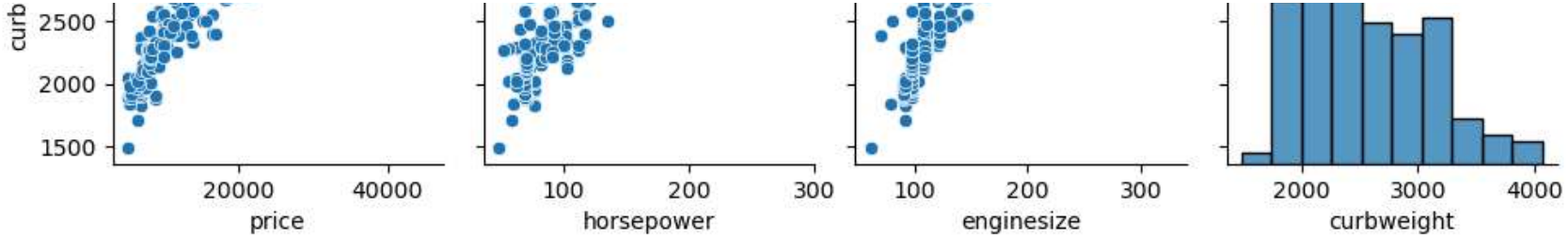

Count Plot for fuelsystem
fuelsystem
mpfi
2bbl
mfi
1bbl
spfi
4bbl
idi
spdi
0
20
40
60
80
count


Count Plot for name
name
alfa-romeo
audi
bmw
chevrolet
dodge
honda
isuzu
jaguar
mazda
buick
mercury
mitsubishi
nissan
peugeot
plymouth
porsche
renault
saab
subaru
toyota
volkswagen
volvo
0
5
10
15
20
25
30
count

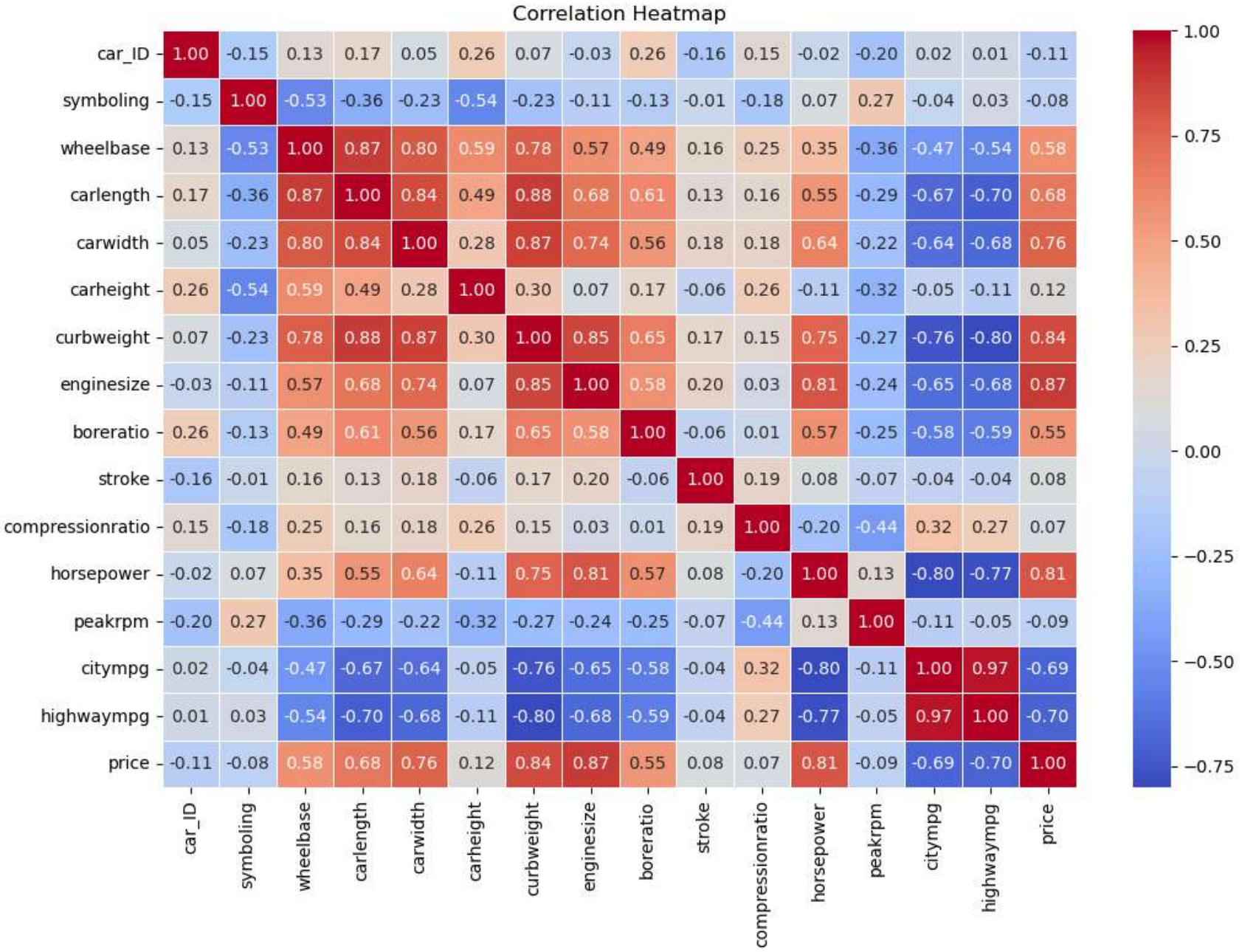
Bivariate Analysis

In [47]:
```
# Scatter plots for numeric variables
sns.pairplot(df, vars=['price', 'horsepower', 'enginesize', 'curbweight'], kind='scatter')
plt.show()
```


price
horsepower
enginesize
weight
40000
30000
20000
10000
300
250
200
150
100
50
4000
3500
3000

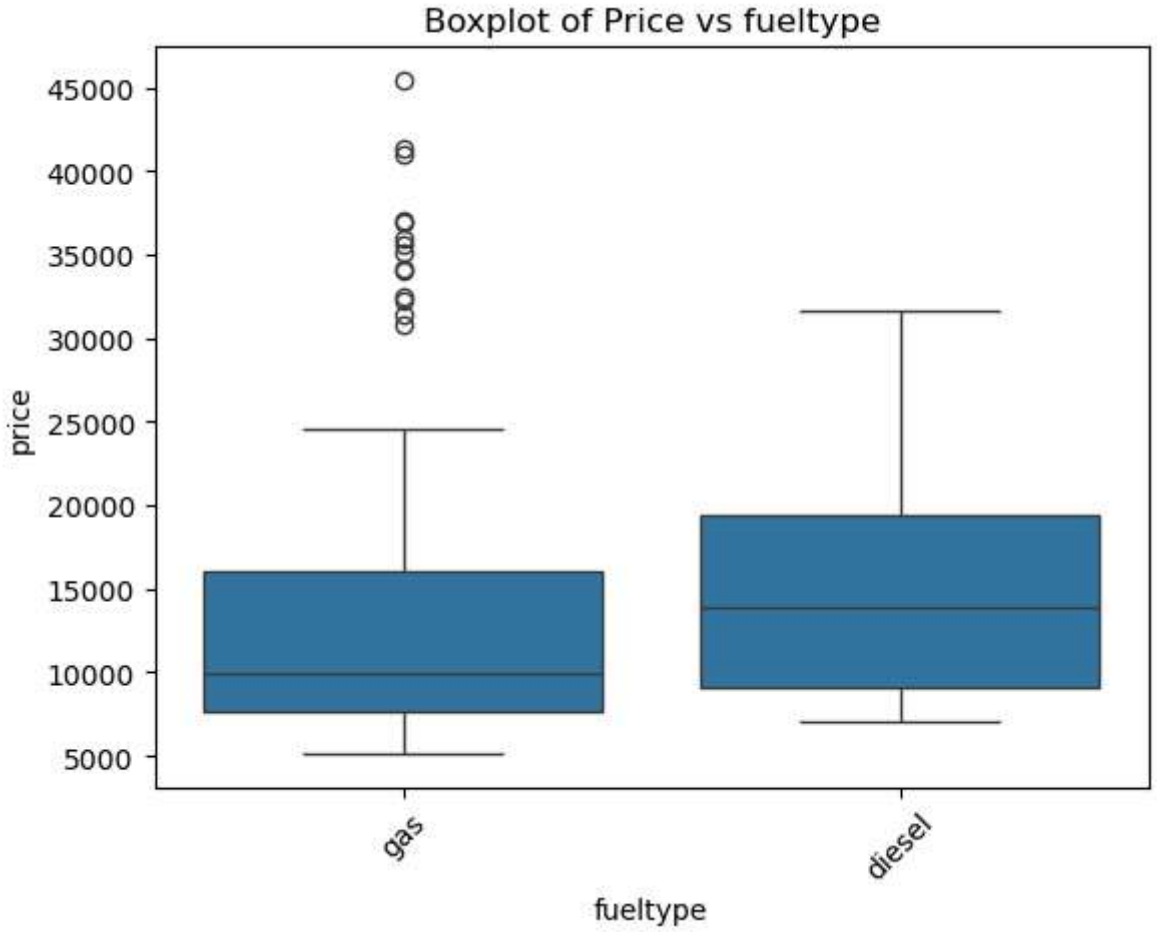


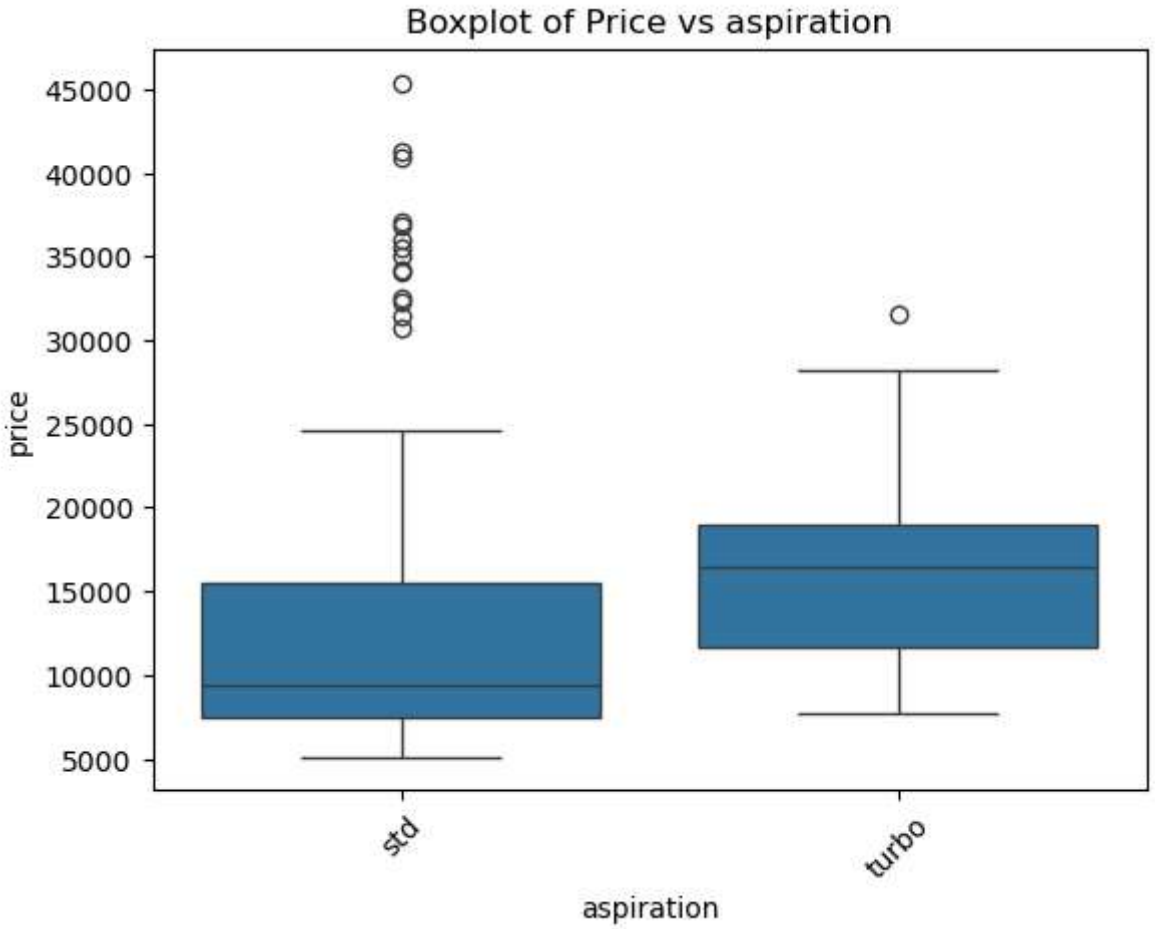
In [61]:
```
# Correlation heatmap for numeric features

# Select only numeric columns for correlation calculation
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix
correlation = numeric_df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

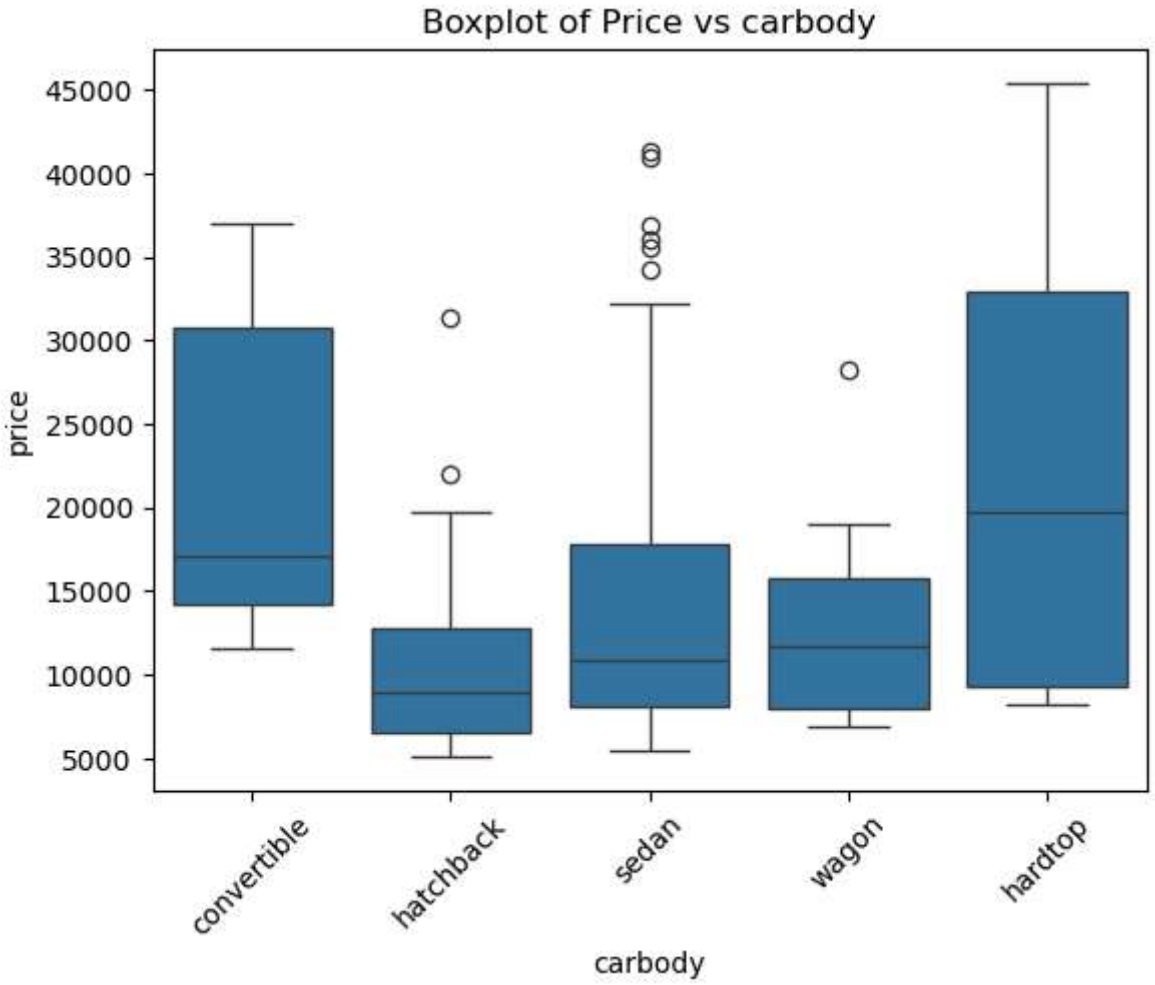


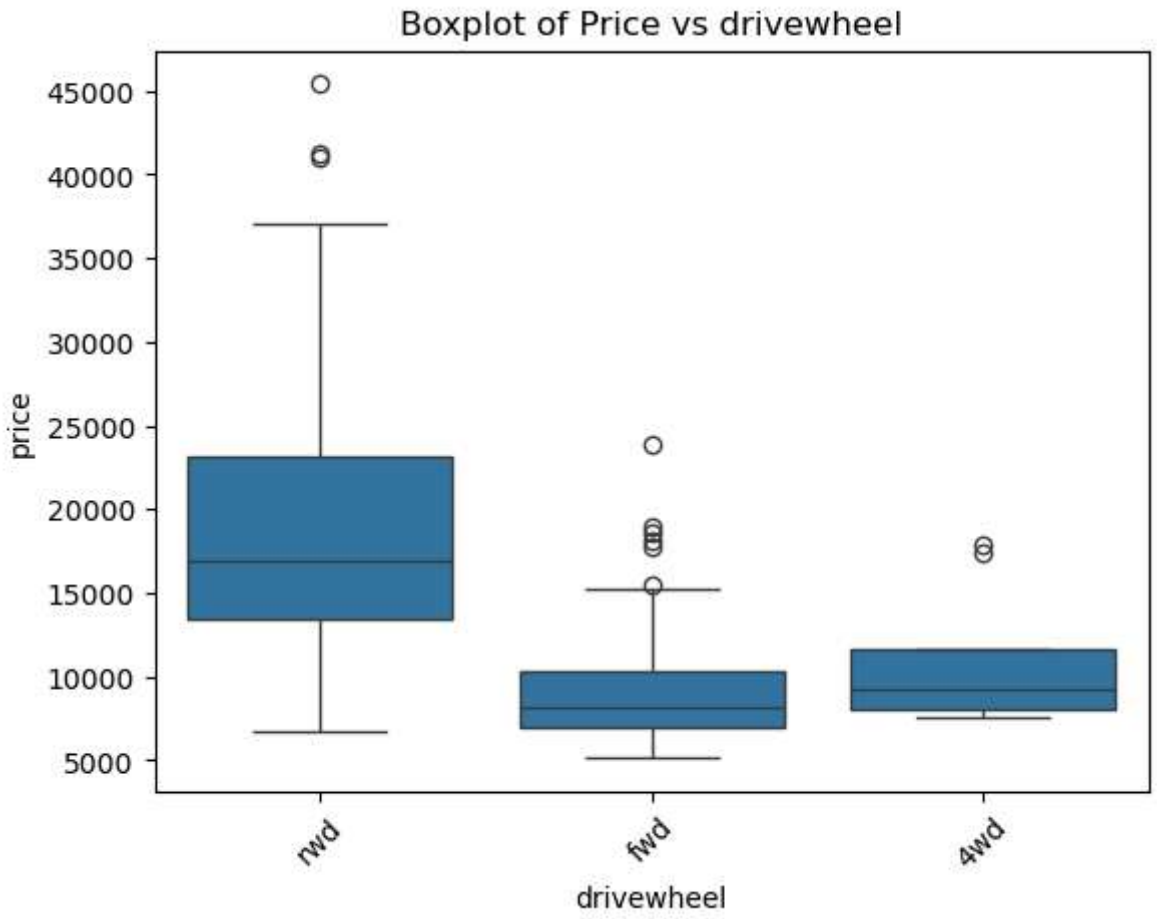
In [51]:
```
# Boxplot of price against categorical variables
for col in ['fueltype', 'aspiration', 'carbody', 'drivewheel']:
```

```
    sns.boxplot(x=col, y='price', data=df)
    plt.title(f"Boxplot of Price vs {col}")
    plt.xticks(rotation=45)
    plt.show()
```

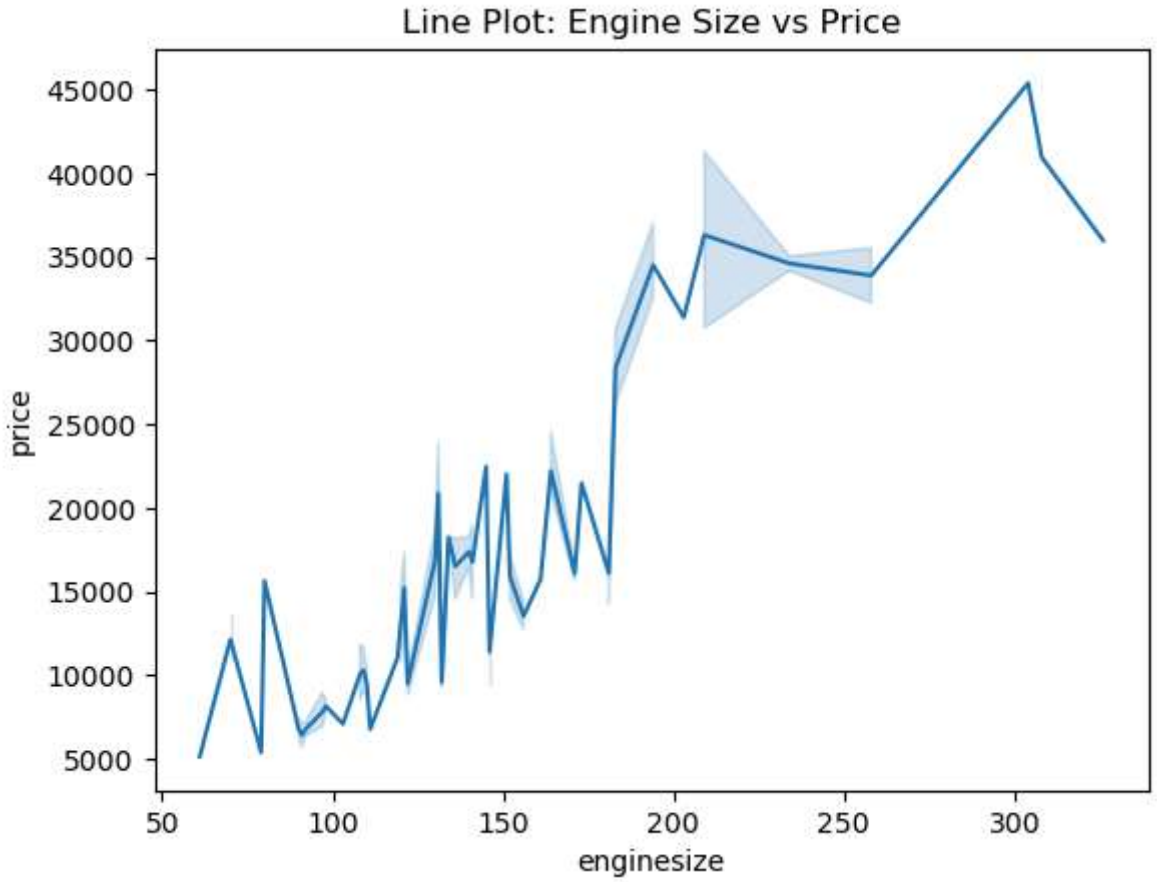



Boxplot of Price vs aspiration
price
45000
40000
35000
30000
25000
20000
15000
10000
5000
std
turbo
aspiration


Boxplot of Price vs carbody
price
45000
40000
35000
30000
25000
20000
15000
10000
5000
convertible
hatchback
sedan
wagon
hardtop
carbody

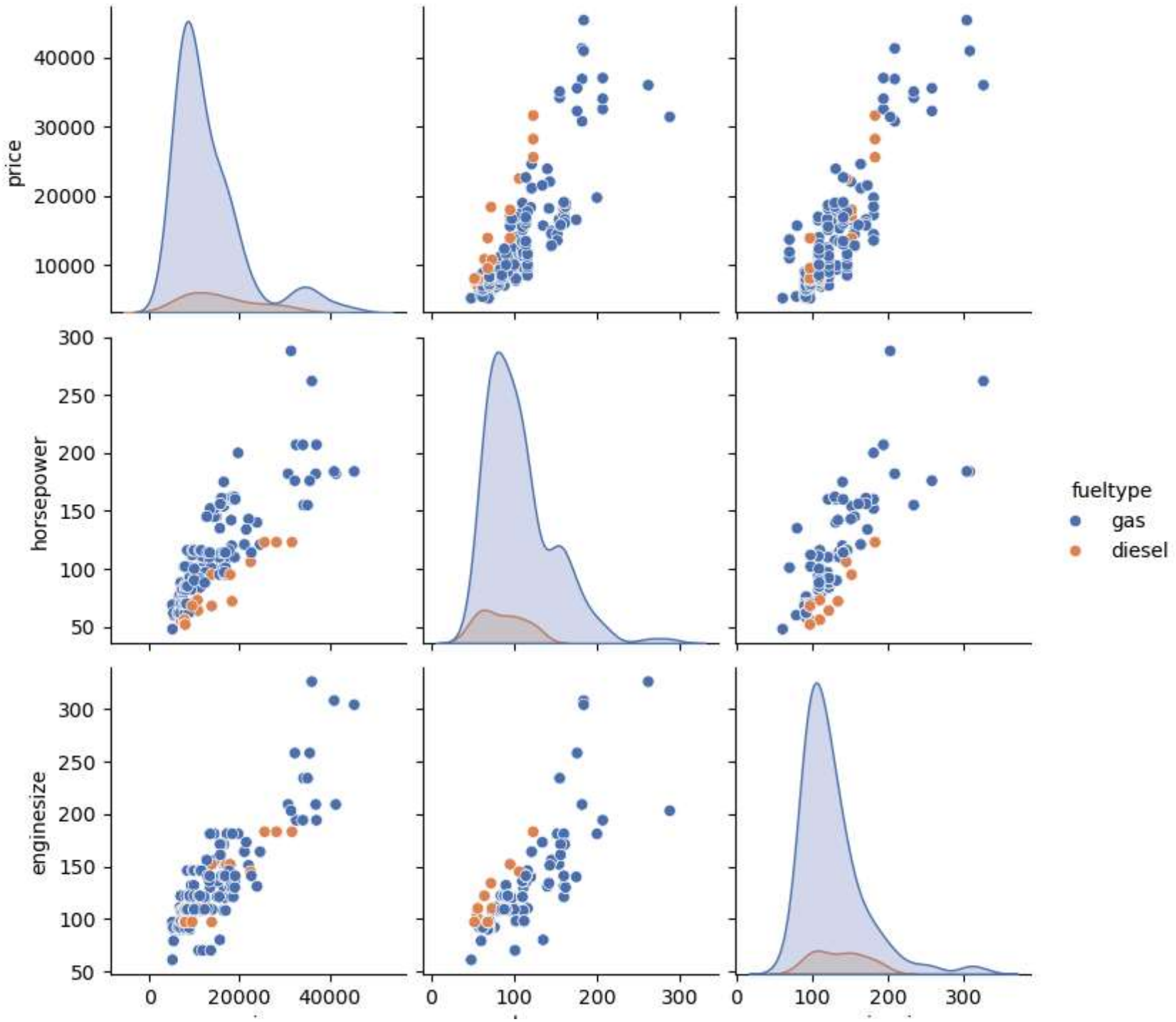


In [53]:
```
# Line plot for engine size and price
sns.lineplot(data=df, x='enginesize', y='price')
plt.title("Line Plot: Engine Size vs Price")
plt.show()
```



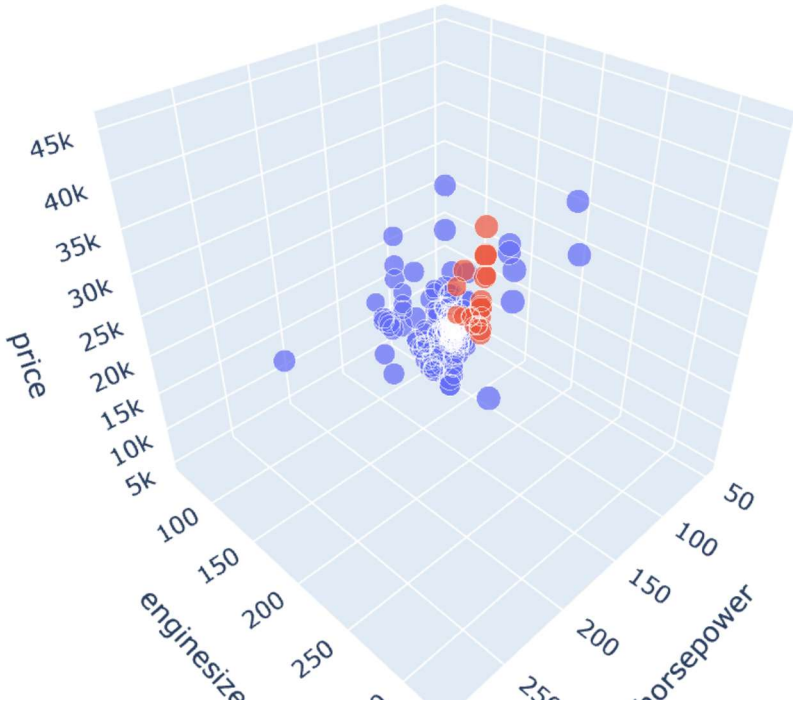
Multivariate Analysis

In [55]:
```
# Pairplot including categorical hue
sns.pairplot(df, vars=['price', 'horsepower', 'enginesize'], hue='fueltype', palette='deep')
plt.show()
```

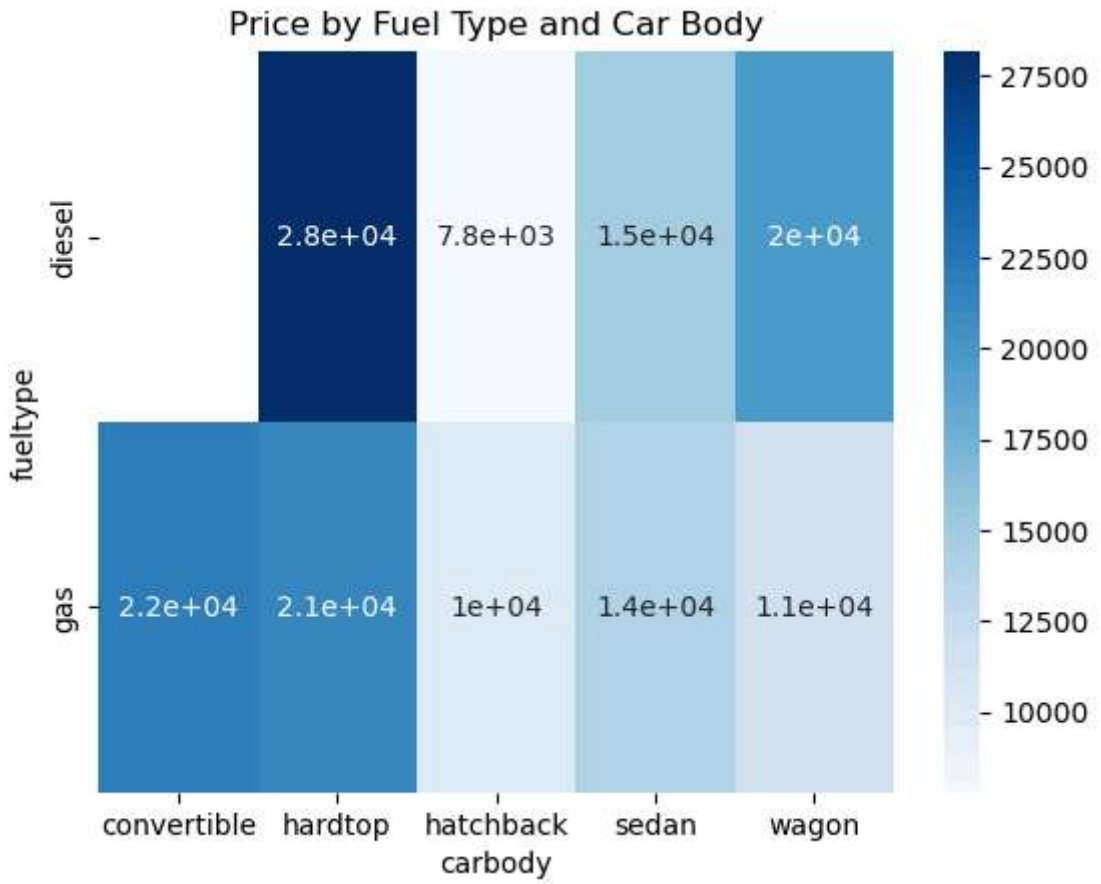

price
horsepower
enginesize
fueltype
gas
diesel
40000
30000
20000
10000
300
250
200
150
100
50
0
20000
40000
0
100
200
300
0
100
200
300

price horsepower enginesize

In [57]:
```
# 3D Scatter plot (requires plotly)
import plotly.express as px
fig = px.scatter_3d(df, x='horsepower', y='enginesize', z='price', color='fueltype', size='curbweight')
fig.show()
```



In [59]:
```
# Heatmap with interactions
pivot_table = pd.pivot_table(df, values='price', index='fueltype', columns='carbody', aggfunc='mean')
sns.heatmap(pivot_table, annot=True, cmap='Blues')
plt.title("Price by Fuel Type and Car Body")
plt.show()
```



Data Cleaning

In [64]:
```
# Feature encoding
# Binary columns: 'fueltype', 'aspiration', 'doornumber', 'enginelocation'
binary_cols = ['fueltype', 'aspiration', 'doornumber', 'enginelocation']
df[binary_cols] = df[binary_cols].apply(lambda x: LabelEncoder().fit_transform(x))
```

```
# Categorical columns: 'name', 'carbody', 'drivewheel', 'enginetype', 'cylindernumber', 'fuelsystem'
categorical_cols = ['name', 'carbody', 'drivewheel', 'enginetype', 'cylindernumber', 'fuelsystem']

# Using one-hot encoding for categorical variables
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

In [66]:
```
# Handling outliers (using Z-score method)
from scipy.stats import zscore

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns  # Numeric columns
z_scores = np.abs(zscore(df[numeric_cols]))
df = df[(z_scores < 3).all(axis=1)]  # Keep rows where z-score < 3
```

In [68]:
```
# Feature scaling (Standard Scaling)
scaler = StandardScaler()
scaled_columns = numeric_cols.drop('price')  # Exclude the target column ('price') from scaling
df[scaled_columns] = scaler.fit_transform(df[scaled_columns])
```

In [70]:
```
df.columns
```

Out[70]:
```
Index(['car_ID', 'symboling', 'fueltype', 'aspiration', 'doornumber',
       'enginelocation', 'wheelbase', 'carlength', 'carwidth', 'carheight',
       'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
       'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price', 'name_audi',
       'name_bmw', 'name_buick', 'name_chevrolet', 'name_dodge', 'name_honda',
       'name_isuzu', 'name_jaguar', 'name_mazda', 'name_mercury',
       'name_mitsubishi', 'name_nissan', 'name_peugeot', 'name_plymouth',
       'name_porsche', 'name_renault', 'name_saab', 'name_subaru',
       'name_toyota', 'name_volkswagen', 'name_volvo', 'carbody_hardtop',
       'carbody_hatchback', 'carbody_sedan', 'carbody_wagon', 'drivewheel_fwd',
       'drivewheel_rwd', 'enginetype_dohcv', 'enginetype_l', 'enginetype_ohc',
       'enginetype_ohcf', 'enginetype_ohcv', 'enginetype_rotor',
       'cylindernumber_five', 'cylindernumber_four', 'cylindernumber_six',
       'cylindernumber_three', 'cylindernumber_twelve', 'cylindernumber_two',
       'fuelsystem_2bbl', 'fuelsystem_4bbl', 'fuelsystem_idi',
       'fuelsystem_mfi', 'fuelsystem_mpfi', 'fuelsystem_spdi',
       'fuelsystem_spfi'],
      dtype='object')
```

In [72]:
```
# Splitting the dataset into features and target variable
X = df.drop(columns=['price'])  # Features
y = df['price']  # Target variable

# 9. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check processed dataset
print("Processed Dataset Shape:", df.shape)
print("Processed Dataset Preview:\n", df.head())
```

```
Processed Dataset Shape: (181, 66)
Processed Dataset Preview:
      car_ID  symboling  fueltype  aspiration  doornumber  enginelocation  \
0 -1.705395   1.692758         1           0           1               0
1 -1.688383   1.692758         1           0           1               0
2 -1.671371   0.113432         1           0           1               0
3 -1.654360   0.903095         1           0           0               0
4 -1.637348   0.903095         1           0           0               0

   wheelbase  carlength  carwidth  carheight  ...  cylindernumber_three  \
0  -1.777805  -0.399899 -0.851313  -1.980334  ...                 False
1  -1.777805  -0.399899 -0.851313  -1.980334  ...                 False
2  -0.706407  -0.190325 -0.136763  -0.520622  ...                 False
3   0.256036   0.281217  0.220512   0.249782  ...                 False
4   0.183399   0.281217  0.322591   0.249782  ...                 False

   cylindernumber_twelve  cylindernumber_two  fuelsystem_2bbl  \
0                  False               False            False
1                  False               False            False
2                  False               False            False
3                  False               False            False
4                  False               False            False

   fuelsystem_4bbl  fuelsystem_idi  fuelsystem_mfi  fuelsystem_mpfi  \
0            False           False           False             True
1            False           False           False             True
2            False           False           False             True
3            False           False           False             True
4            False           False           False             True

   fuelsystem_spdi  fuelsystem_spfi
0            False            False
1            False            False
2            False            False
3            False            False
4            False            False

[5 rows x 66 columns]
```

Model Implementation:

- **Implement the following five regression algorithms:**

1. Linear Regression

2. Decision Tree Regressor

3. Random Forest Regressor

4. Gradient Boosting Regressor

5. Support Vector Regressor

Model Evaluation:

• Compare the performance of all the models based on R-squared, Mean Squared Error (MSE), and Mean Absolute Error (MAE).

• Identify the best performing model and justify why it is the best.

In [74]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import pandas as pd

# Initialize the regression models
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree Regressor": DecisionTreeRegressor(random_state=42),
    "Random Forest Regressor": RandomForestRegressor(random_state=42, n_jobs=-1),
    "Gradient Boosting Regressor": GradientBoostingRegressor(random_state=42),
    "Support Vector Regressor": SVR(kernel='rbf')
}

# Dictionary to store results
results = {}
```

```python
# Train and evaluate each model
for name, model in models.items():
    print(f"Training and evaluating {name}...")
    # Train the model
    model.fit(X_train, y_train)
    # Predict on test data
    y_pred = model.predict(X_test)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Save results
    results[name] = {"R-squared": r2, "MSE": mse, "MAE": mae}
    print(f"{name} - R-squared: {r2:.3f}, MSE: {mse:.3f}, MAE: {mae:.3f}")
    print("-" * 50)

# Display results in a DataFrame
results_df = pd.DataFrame(results).T  # Transpose for better readability
print("\nComparison of Model Performance:")
print(results_df)

# Sort the results by R-squared for better visualization
sorted_results = results_df.sort_values(by="R-squared", ascending=False)
print("\nSorted Results by R-squared:")
print(sorted_results)
```

```
Training and evaluating Linear Regression...
Linear Regression - R-squared: -0.597, MSE: 79979879.040, MAE: 4303.487
------------------------------------------------
Training and evaluating Decision Tree Regressor...
Decision Tree Regressor - R-squared: 0.862, MSE: 6901322.654, MAE: 1709.306
------------------------------------------------
Training and evaluating Random Forest Regressor...
Random Forest Regressor - R-squared: 0.930, MSE: 3493466.544, MAE: 1286.072
------------------------------------------------
Training and evaluating Gradient Boosting Regressor...
Gradient Boosting Regressor - R-squared: 0.930, MSE: 3489361.476, MAE: 1334.132
------------------------------------------------
Training and evaluating Support Vector Regressor...
Support Vector Regressor - R-squared: -0.055, MSE: 52868726.512, MAE: 4870.108
------------------------------------------------

Comparison of Model Performance:
                           R-squared           MSE          MAE
Linear Regression          -0.596554  7.997988e+07  4303.487327
Decision Tree Regressor     0.862236  6.901323e+06  1709.306297
Random Forest Regressor     0.930264  3.493467e+06  1286.071703
Gradient Boosting Regressor 0.930346  3.489361e+06  1334.132024
Support Vector Regressor   -0.055363  5.286873e+07  4870.107576

Sorted Results by R-squared:
                           R-squared           MSE          MAE
Gradient Boosting Regressor 0.930346  3.489361e+06  1334.132024
Random Forest Regressor     0.930264  3.493467e+06  1286.071703
Decision Tree Regressor     0.862236  6.901323e+06  1709.306297
Support Vector Regressor   -0.055363  5.286873e+07  4870.107576
Linear Regression          -0.596554  7.997988e+07  4303.487327
```

1. R-squared (Coefficient of Determination):

__Interpretation:__ R-squared measures how well the model explains the variability of the target variable. Higher values (closer to 1) are better.

__Ranking (Highest to Lowest):__

1. Gradient Boosting Regressor: 0.930346

2. Random Forest Regressor: 0.930264
3. Decision Tree Regressor: 0.862236
4. Support Vector Regressor: -0.055363 (negative value indicates a poor fit)
5. Linear Regression: -0.596554 (negative value indicates the model performs worse than the mean of the target variable)

__Conclusion:__ **Gradient Boosting Regressor** has the best R-squared value, meaning it explains the variability in the data most effectively.

2. Mean Squared Error (MSE):

__Interpretation:__ MSE measures the average squared difference between predicted and actual values. Lower values are better.

__Ranking (Lowest to Highest):__

1. Gradient Boosting Regressor: 3.489361 * 10e6
2. Random Forest Regressor: 3.493467 * 10e6
3. Decision Tree Regressor: 6.901323 * 10e6
4. Support Vector Regressor: 5.286873 * 10e7
5. Linear Regression: 7.997988 * 10e7

__Conclusion:__ **Gradient Boosting Regressor** has the lowest MSE, indicating it produces the smallest prediction errors on average.

3. Mean Absolute Error (MAE):

__Interpretation:__ MAE measures the average absolute difference between predicted and actual values. Lower values are better.

__Ranking (Lowest to Highest):__

1. Random Forest Regressor: 1286.071703
2. Gradient Boosting Regressor: 1334.132024
3. Decision Tree Regressor: 1709.306297
4. Linear Regression: 4303.487327
5. Support Vector Regressor: 4870.107576

__Conclusion:__ **Random Forest Regressor** has the lowest MAE, meaning its predictions are the closest to actual values, on average.

Final Recommendation:

Considering R-squared and MSE, Gradient Boosting Regressor performs the best overall, as it explains the variance in the data effectively and has the smallest squared errors. If minimizing absolute errors (MAE) is critical, Random Forest Regressor is slightly better. However, the differences between Gradient Boosting and Random Forest are minimal. **GRADIENT BOOSTING REGRESSOR** is the best choice as it balances all metrics well.

Feature Importance Analysis:

- **Identify the significant variables affecting car prices (feature selection).**

In [76]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate correlation of price with each feature
correlation_with_price = df.corr()["price"].sort_values(ascending=False)
correlation_table = correlation_with_price.to_frame(name="Correlation with Price")

# Filter the table to display only positive correlation values
positive_correlation_table = correlation_table[correlation_table["Correlation with Price"] > 0]

# Display the filtered table
print("Features with Positive Correlation with Price:")
display(positive_correlation_table)
```

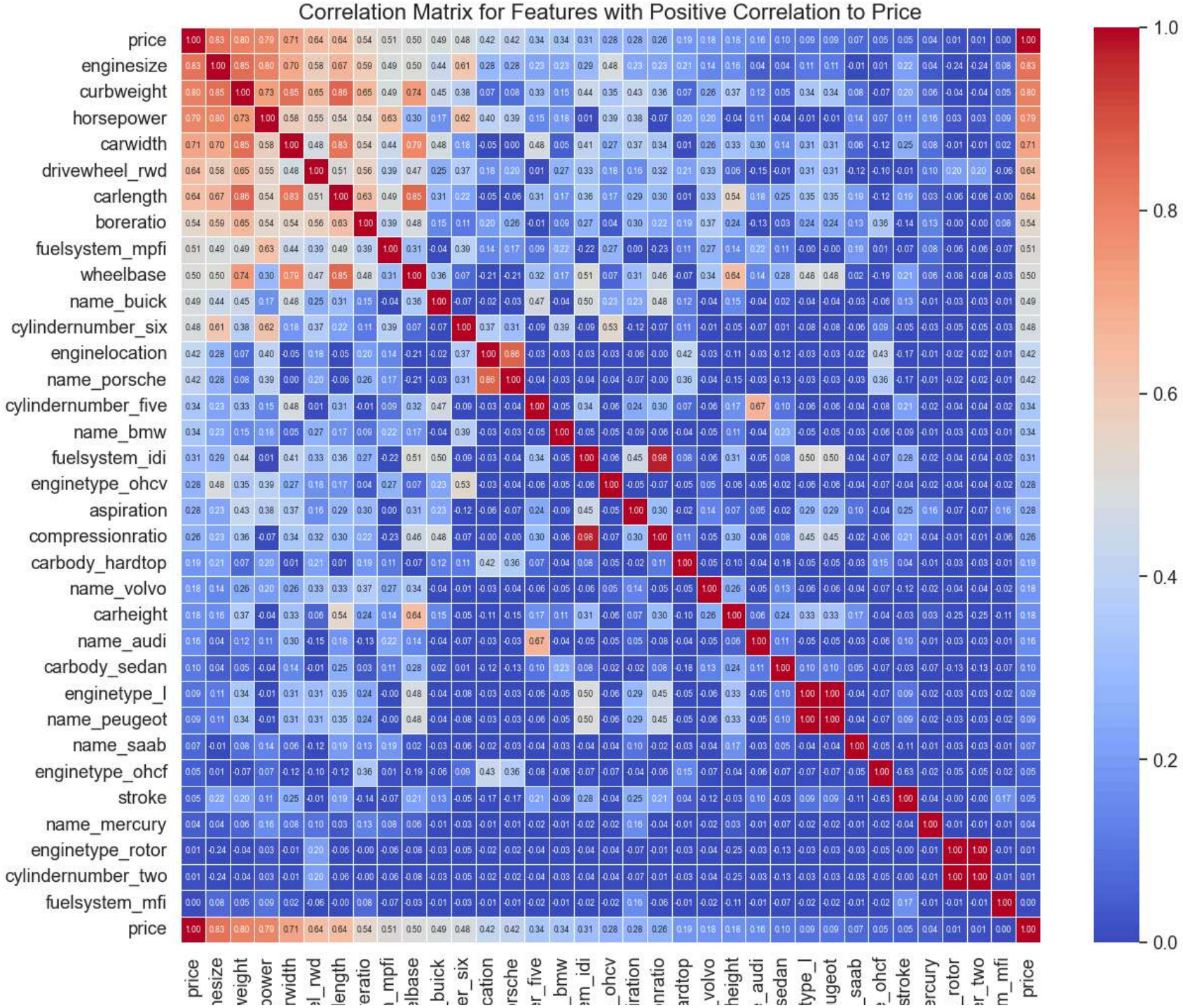
```
Features with Positive Correlation with Price:
```

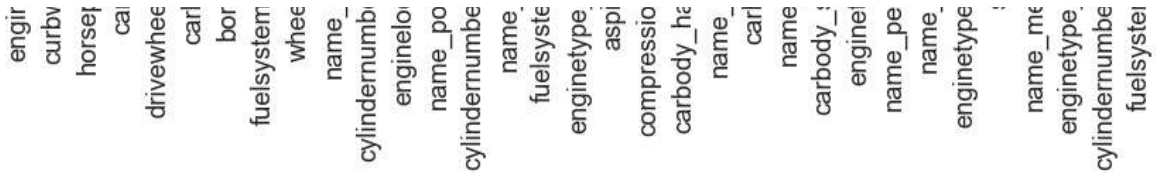
	Correlation with Price
price	1.000000
enginesize	0.830473
curbweight	0.802365
horsepower	0.792024
carwidth	0.711771
drivewheel_rwd	0.638593
carlength	0.638258
boreratio	0.536939
fuelsystem_mpfi	0.509319
wheelbase	0.499827
name_buick	0.491968
cylindernumber_six	0.484293
enginelocation	0.423143
name_porsche	0.420193
cylindernumber_five	0.342106
name_bmw	0.338747
fuelsystem_idi	0.305847
enginetype_ohcv	0.280770
aspiration	0.279749
compressionratio	0.263098
carbody_hardtop	0.188320
name_volvo	0.181358

	Correlation with Price
carheight	0.175122
name_audi	0.157468
carbody_sedan	0.103060
enginetype_l	0.094218
name_peugeot	0.094218
name_saab	0.067297
enginetype_ohcf	0.052134
stroke	0.051856
name_mercury	0.043512
enginetype_rotor	0.010037
cylindernumber_two	0.010037
fuelsystem_mfi	0.004357

In [78]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Filter the correlation table for positive correlation values
positive_correlations = correlation_table[correlation_table["Correlation with Price"] > 0]

# Extract the feature names with positive correlation
positive_features = positive_correlations.index.tolist()

# Create a subset of the dataframe with only positive correlation features and price
positive_correlation_matrix = df[positive_features + ["price"]].corr()

# Adjust the font scale and create the heatmap
sns.set(font_scale=1.2)  # Adjust font scale for better readability
plt.figure(figsize=(14, 12))  # Adjust the figure size for larger boxes
sns.heatmap(
    positive_correlation_matrix,
```

```
    annot=True,
    fmt=".2f",  # Format the annotations to 2 decimal places
    annot_kws={"size": 6},  # Adjust the size of annotation text
    cmap="coolwarm",
    linewidths=0.5,
    vmin=0,  # Minimum correlation value (only positive correlations are shown)
)
plt.title("Correlation Matrix for Features with Positive Correlation to Price", fontsize=16)
plt.show()
```


Correlation Matrix for Features with Positive Correlation to Price
price
enginesize
curbweight
horsepower
carwidth
drivewheel_rwd
carlength
boreratio
fuelsystem_mpfi
wheelbase
name_buick
cylindernumber_six
enginelocation
name_porsche
cylindernumber_five
name_bmw
fuelsystem_idi
enginetype_ohcv
aspiration
compressionratio
carbody_hardtop
name_volvo
carheight
name_audi
carbody_sedan
enginetype_l
name_peugeot
name_saab
enginetype_ohcf
stroke
name_mercury
enginetype_rotor
cylindernumber_two
fuelsystem_mfi
price
1.0
0.8
0.6
0.4
0.2
0.0



To determine the main features with a strong correlation to price, we typically use the following thresholds as a guideline:

Strong correlation: Absolute correlation value ≥ 0.7

Moderate correlation: Absolute correlation value between 0.5 and 0.7

Weak correlation: Absolute correlation value < 0.5

Based on the values, the features with strong correlation (≥ 0.7) to price are:

1. *__enginesize: 0.830473__*
2. *__curbweight: 0.802365__*
3. *__horsepower: 0.792024__*
4. *__carwidth: 0.711771__*

Hyperparameter Tuning:

- **Perform hyperparameter tuning and check whether the performance of the model has increased.**

In [80]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Define parameter grids for each model
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4]
}

param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

param_grid_dt = {
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

param_grid_svr = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'epsilon': [0.1, 0.2, 0.5],
    'gamma': ['scale', 'auto']
}

# Linear Regression has no hyperparameters to tune in this case
param_grid_lr = {}

# Initialize models
models = {
    "Random Forest": (RandomForestRegressor(random_state=42), param_grid_rf),
    "Gradient Boosting": (GradientBoostingRegressor(random_state=42), param_grid_gb),
    "Decision Tree": (DecisionTreeRegressor(random_state=42), param_grid_dt),
    "Support Vector Regressor": (SVR(), param_grid_svr),
    "Linear Regression": (LinearRegression(), param_grid_lr)
}

# Perform GridSearchCV and evaluate performance
best_estimators = {}
for model_name, (model, param_grid) in models.items():
    print(f"Performing GridSearchCV for {model_name}...")

    if param_grid:  # Skip GridSearchCV for models with no hyperparameters to tune
        grid_search = GridSearchCV(model, param_grid, cv=3, scoring='r2', n_jobs=-1)
```

```
        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_
        print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    else:
        model.fit(X_train, y_train)  # Train model directly for Linear Regression
        best_model = model

    # Store the best estimator
    best_estimators[model_name] = best_model

    # Evaluate performance on the test set
    y_pred = best_model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    print(f"{model_name} Performance after Hyperparameter Tuning:")
    print(f"R-squared: {r2:.4f}")
    print(f"Mean Squared Error: {mse:.4f}")
    print(f"Mean Absolute Error: {mae:.4f}")
    print("-" * 50)
```

```
Performing GridSearchCV for Random Forest...
Best parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1
00}
Random Forest Performance after Hyperparameter Tuning:
R-squared: 0.9282
Mean Squared Error: 3595887.4571
Mean Absolute Error: 1295.1399
--------------------------------------------------
Performing GridSearchCV for Gradient Boosting...
Best parameters for Gradient Boosting: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'subsample': 0.8}
Gradient Boosting Performance after Hyperparameter Tuning:
R-squared: 0.9094
Mean Squared Error: 4538433.5470
Mean Absolute Error: 1451.8173
--------------------------------------------------
Performing GridSearchCV for Decision Tree...
Best parameters for Decision Tree: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5}
Decision Tree Performance after Hyperparameter Tuning:
R-squared: 0.8518
Mean Squared Error: 7426333.2763
Mean Absolute Error: 1689.4226
--------------------------------------------------
Performing GridSearchCV for Support Vector Regressor...
Best parameters for Support Vector Regressor: {'C': 10, 'epsilon': 0.5, 'gamma': 'scale', 'kernel': 'linear'}
Support Vector Regressor Performance after Hyperparameter Tuning:
R-squared: 0.6652
Mean Squared Error: 16772056.3722
Mean Absolute Error: 2135.2316
--------------------------------------------------
Performing GridSearchCV for Linear Regression...
Linear Regression Performance after Hyperparameter Tuning:
R-squared: -0.5966
Mean Squared Error: 79979879.0405
Mean Absolute Error: 4303.4873
--------------------------------------------------
```

1. Gradient Boosting Regressor

Before Tuning: R-squared: 0.9303 MSE: 3.489361 * 10e6 MAE: 1334.1320

After Tuning: R-squared: 0.9094 (↓) MSE: 4.538433 * 10e6 (↑) MAE: 1451.8173 (↑)

Observation: Performance **decreased** after tuning. The R-squared dropped slightly, while both MSE and MAE increased, suggesting the tuning did not improve the model.

2. Random Forest Regressor

Before Tuning: R-squared: 0.9303 MSE: 3.493467 * 10e6 MAE: 1286.0717

After Tuning: R-squared: 0.9282 (↓) MSE: 3.595887 * 10e6 (↑) MAE: 1295.1399 (↑)

Observation: Performance **slightly decreased** after tuning. The changes in metrics are minor, but they indicate the model's performance worsened slightly.

3. Decision Tree Regressor

Before Tuning: R-squared: 0.8622 MSE: 6.901323 * 10e6 MAE: 1709.3063

After Tuning: R-squared: 0.8518 (↓) MSE: 7.426333 * 10e6 (↑) MAE: 1689.4226 (↓)

Observation: Performance **slightly decreased** after tuning. R-squared dropped a bit, and MSE increased, but MAE improved marginally.

4. Support Vector Regressor

Before Tuning: R-squared: -0.0554 MSE: 5.286873 * 10e7 MAE: 4870.1076

After Tuning: R-squared: 0.6652 (↑) MSE: 1.677206 * 10e7 (↓) MAE: 2135.2316 (↓)

Observation: Performance **significantly improved** after tuning. R-squared turned positive and increased substantially, while MSE and MAE both decreased significantly, making the model much better than before.

5. Linear Regression

Before Tuning: R-squared: -0.5966 MSE: 7.997988 * 10e7 MAE: 4303.4873

After Tuning: R-squared: -0.5966 (no change) MSE: 7.997988 * 10e7 (no change) MAE: 4303.4873 (no change)

Observation: Performance **did not change** because hyperparameter tuning likely wasn't applied to Linear Regression (no parameters to tune).

Conclusion

Support Vector Regressor showed the most improvement after tuning, with a substantial increase in R-squared and significant reductions in MSE and MAE. While it still doesn't outperform Gradient Boosting or Random Forest, it became a much more viable model. Other models, especially Gradient Boosting and Random Forest, showed slightly worse performance after tuning. This could be due to overfitting or suboptimal parameter combinations. Linear Regression remained unchanged.