d9vuyaffr

December 3, 2024

1 Car Price Prediction Model

1.1 Name: Ajo Babu A

1.2 1. Overview of Problem Statement:

A Chinese automobile company aspires to establish its presence in the US market by manufacturing cars locally to compete with US and European brands. Understanding the factors influencing car prices in the US, which differ from the Chinese market, is critical for success. The company has contracted a consulting firm to analyze a dataset of various cars from the American market, with the goal of identifying the variables that significantly affect car prices and understanding how these factors influence pricing. This insight will guide the company's market entry strategy and product design to achieve competitive pricing.

1.3 2. Objective

The objective of this project is to develop a regression model to predict car prices in the US market using various independent variables. This model will help identify significant factors influencing car prices and explain their relationship to pricing, enabling the company to design competitive cars and devise effective business strategies.

1.4 3. Data description

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

Features: car_ID ,symboling, CarName, fueltype, aspiration, doornumber, carbody, drivewheel, enginelocation, wheelbase, carlength, carwidth, carheight, curbweight, enginetype, cylindernumber, enginesize, fuelsystem, boreratio, stroke, compressionratio, horsepower, peakrpm,citympg, highwaympg, price

1.5 Importing Libraries

[271]:
```python
# Importing Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
from sklearn.feature_selection import VarianceThreshold


from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.model_selection import train_test_split
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 r2_score, mean_absolute_error, mean_squared_error

from sklearn.model_selection import GridSearchCV

import joblib
import warnings
warnings.filterwarnings("ignore")
```

1.6 4. Importing Data

[20]:
```
data = pd.read_csv('CarPrice_Assignment.csv')
data
```

[20]:
```
     car_ID  symboling                   CarName fueltype aspiration  \
0         1          3        alfa-romero giulia      gas        std
1         2          3       alfa-romero stelvio      gas        std
2         3          1  alfa-romero Quadrifoglio      gas        std
3         4          2               audi 100 ls      gas        std
4         5          2                audi 100ls      gas        std
..      …        …                       …      …        …
200     201         -1           volvo 145e (sw)      gas        std
201     202         -1               volvo 144ea      gas      turbo
202     203         -1               volvo 244dl      gas        std
203     204         -1                 volvo 246   diesel      turbo
204     205         -1               volvo 264gl      gas      turbo

    doornumber      carbody drivewheel enginelocation  wheelbase  …  \
0          two  convertible        rwd          front       88.6  …
1          two  convertible        rwd          front       88.6  …
2          two    hatchback        rwd          front       94.5  …
3         four        sedan        fwd          front       99.8  …
4         four        sedan        4wd          front       99.4  …
..         …          …        …            …        …  …
200       four        sedan        rwd          front      109.1  …
201       four        sedan        rwd          front      109.1  …
202       four        sedan        rwd          front      109.1  …
203       four        sedan        rwd          front      109.1  …
```

```
204       four        sedan        rwd          front       109.1  …

     enginesize  fuelsystem  boreratio  stroke compressionratio horsepower  \
0           130        mpfi       3.47    2.68              9.0        111
1           130        mpfi       3.47    2.68              9.0        111
2           152        mpfi       2.68    3.47              9.0        154
3           109        mpfi       3.19    3.40             10.0        102
4           136        mpfi       3.19    3.40              8.0        115
..          …          …         …      …              …        …
200         141        mpfi       3.78    3.15              9.5        114
201         141        mpfi       3.78    3.15              8.7        160
202         173        mpfi       3.58    2.87              8.8        134
203         145         idi       3.01    3.40             23.0        106
204         141        mpfi       3.78    3.15              9.5        114

     peakrpm citympg  highwaympg    price
0       5000      21          27  13495.0
1       5000      21          27  16500.0
2       5000      19          26  16500.0
3       5500      24          30  13950.0
4       5500      18          22  17450.0
..       …     …         …      …
200     5400      23          28  16845.0
201     5300      19          25  19045.0
202     5500      18          23  21485.0
203     4800      26          27  22470.0
204     5400      19          25  22625.0

[205 rows x 26 columns]
```

[22]:
```
df = pd.DataFrame(data)
```

[24]:
```
df.head()
```

[24]:
```
   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]
```

[26]:
```
df.info()
```

```
<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
```

[28]:
```
df.dtypes
```

[28]:
```
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
```

[30]:
```
df.describe()
```

[30]:
```
           car_ID   symboling   wheelbase   carlength    carwidth   carheight  \
count  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
std     59.322565    1.245307    6.021776   12.337289    2.145204    2.443522
min      1.000000   -2.000000   86.600000  141.100000   60.300000   47.800000
25%     52.000000    0.000000   94.500000  166.300000   64.100000   52.000000
50%    103.000000    1.000000   97.000000  173.200000   65.500000   54.100000
75%    154.000000    2.000000  102.400000  183.100000   66.900000   55.500000
max    205.000000    3.000000  120.900000  208.100000   72.300000   59.800000
```

```
       curbweight  enginesize   boreratio      stroke  compressionratio  \
count   205.000000  205.000000  205.000000  205.000000        205.000000
mean   2555.565854  126.907317    3.329756    3.255415         10.142537
std     520.680204   41.642693    0.270844    0.313597          3.972040
min    1488.000000   61.000000    2.540000    2.070000          7.000000
25%    2145.000000   97.000000    3.150000    3.110000          8.600000
50%    2414.000000  120.000000    3.310000    3.290000          9.000000
75%    2935.000000  141.000000    3.580000    3.410000          9.400000
max    4066.000000  326.000000    3.940000    4.170000         23.000000

       horsepower      peakrpm     citympg  highwaympg         price
count  205.000000   205.000000  205.000000  205.000000    205.000000
mean   104.117073  5125.121951   25.219512   30.751220  13276.710571
std     39.544167   476.985643    6.542142    6.886443   7988.852332
min     48.000000  4150.000000   13.000000   16.000000   5118.000000
25%     70.000000  4800.000000   19.000000   25.000000   7788.000000
50%     95.000000  5200.000000   24.000000   30.000000  10295.000000
75%    116.000000  5500.000000   30.000000   34.000000  16503.000000
max    288.000000  6600.000000   49.000000   54.000000  45400.000000
```

[34]:
```
df.shape
```

[34]:
```
(205, 26)
```

[36]:
```
df.columns
```

[36]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

1.7 5.Data preprocessing and Data cleaning

[39]:
```
# Checking for duplicate
df.duplicated()
```

[39]:
```
0      False
1      False
2      False
3      False
4      False
       …
200    False
```

```
201    False
202    False
203    False
204    False
Length: 205, dtype: bool
```

[43]:
```
df.duplicated().sum()
```

[43]:
```
0
```

No duplicates found

[46]:
```
# Checking for null values
df.isnull()
```

[46]:
```
     car_ID  symboling  CarName  fueltype  aspiration  doornumber  carbody  \
0     False      False    False     False       False       False    False
1     False      False    False     False       False       False    False
2     False      False    False     False       False       False    False
3     False      False    False     False       False       False    False
4     False      False    False     False       False       False    False
..      …          …        …        …          …           …        …
200   False      False    False     False       False       False    False
201   False      False    False     False       False       False    False
202   False      False    False     False       False       False    False
203   False      False    False     False       False       False    False
204   False      False    False     False       False       False    False

     drivewheel  enginelocation  wheelbase  …  enginesize  fuelsystem  \
0         False           False      False  …       False       False
1         False           False      False  …       False       False
2         False           False      False  …       False       False
3         False           False      False  …       False       False
4         False           False      False  …       False       False
..          …               …         …  …        …           …
200       False           False      False  …       False       False
201       False           False      False  …       False       False
202       False           False      False  …       False       False
203       False           False      False  …       False       False
204       False           False      False  …       False       False

     boreratio  stroke  compressionratio  horsepower  peakrpm  citympg  \
0        False   False             False       False    False    False
1        False   False             False       False    False    False
2        False   False             False       False    False    False
3        False   False             False       False    False    False
4        False   False             False       False    False    False
```

```
..        …      …                …         …      …      …
200      False  False            False     False  False  False
201      False  False            False     False  False  False
202      False  False            False     False  False  False
203      False  False            False     False  False  False
204      False  False            False     False  False  False

     highwaympg  price
0         False  False
1         False  False
2         False  False
3         False  False
4         False  False
..          …      …
200       False  False
201       False  False
202       False  False
203       False  False
204       False  False

[205 rows x 26 columns]
```

[48]:
```
df.isnull().sum()
```

[48]:
```
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
```

No null vaalues found

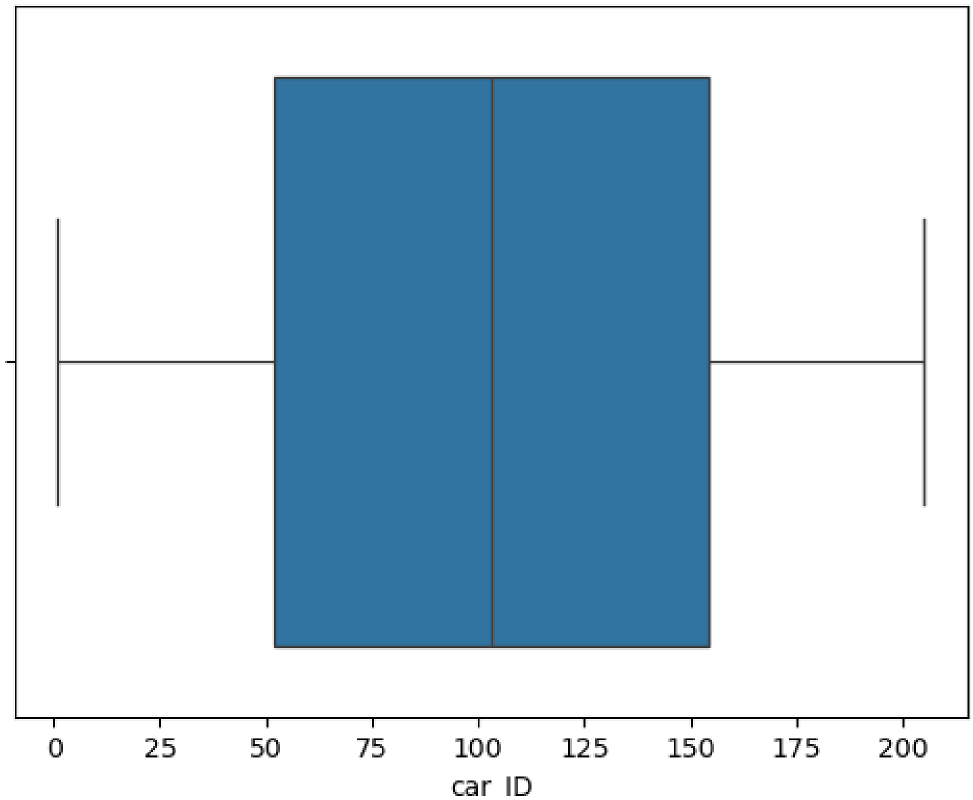
1.7.1 Classifying columns

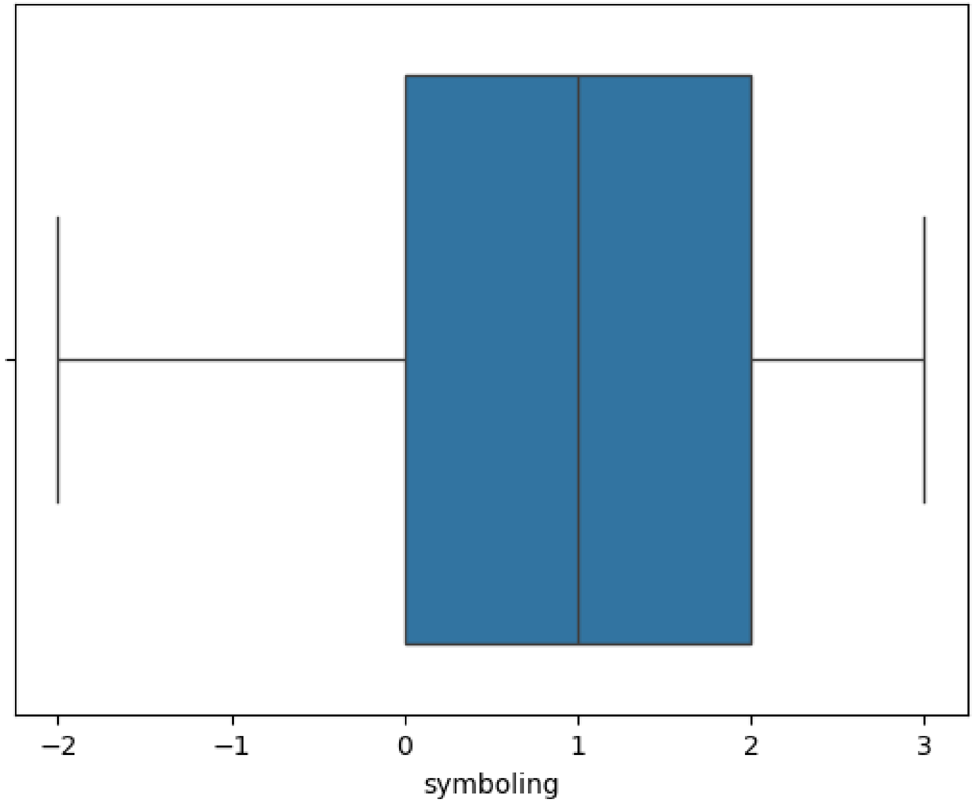
[52]:
```
numerical_columns = df.select_dtypes(include=['number']).columns
categorical_columns = df.select_dtypes(include=['object']).columns

print("numerical columns: ",numerical_columns)
print("Categorical columns: ",categorical_columns)
```

```
numerical columns:  Index(['car_ID', 'symboling', 'wheelbase', 'carlength',
'carwidth',
       'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
Categorical columns:  Index(['CarName', 'fueltype', 'aspiration', 'doornumber',
'carbody',
       'drivewheel', 'enginelocation', 'enginetype', 'cylindernumber',
       'fuelsystem'],
      dtype='object')
```

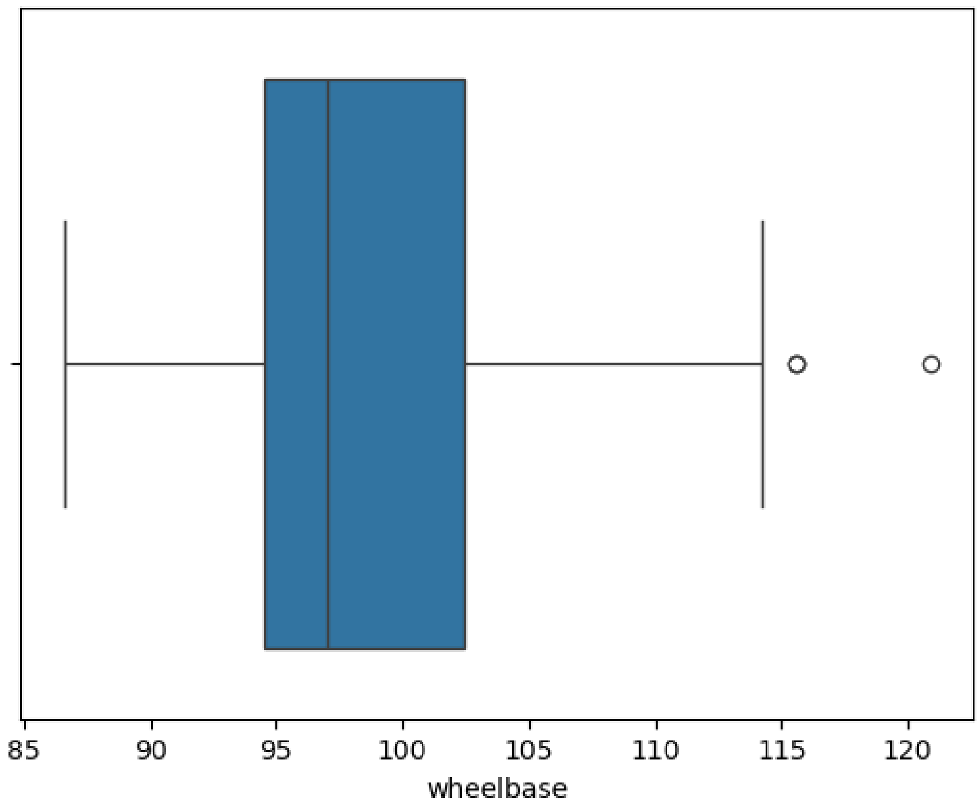
1.7.2 Outlier Detection

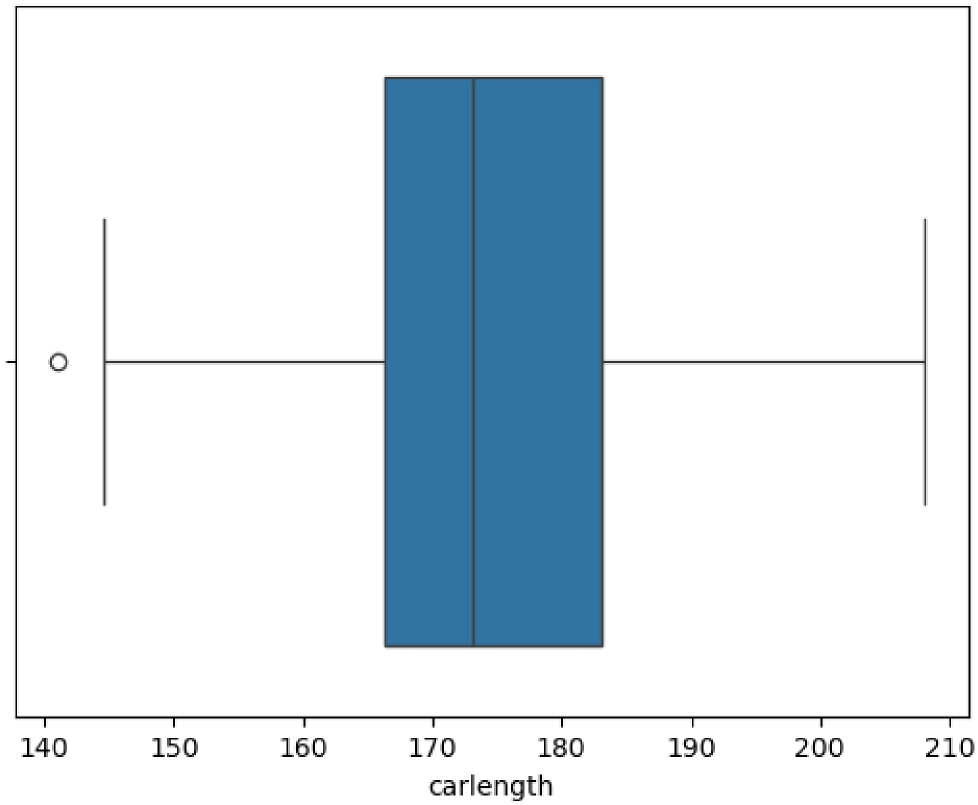
[57]:
```
# boxplot to identify outliers
for i in df.select_dtypes(include='number').columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```

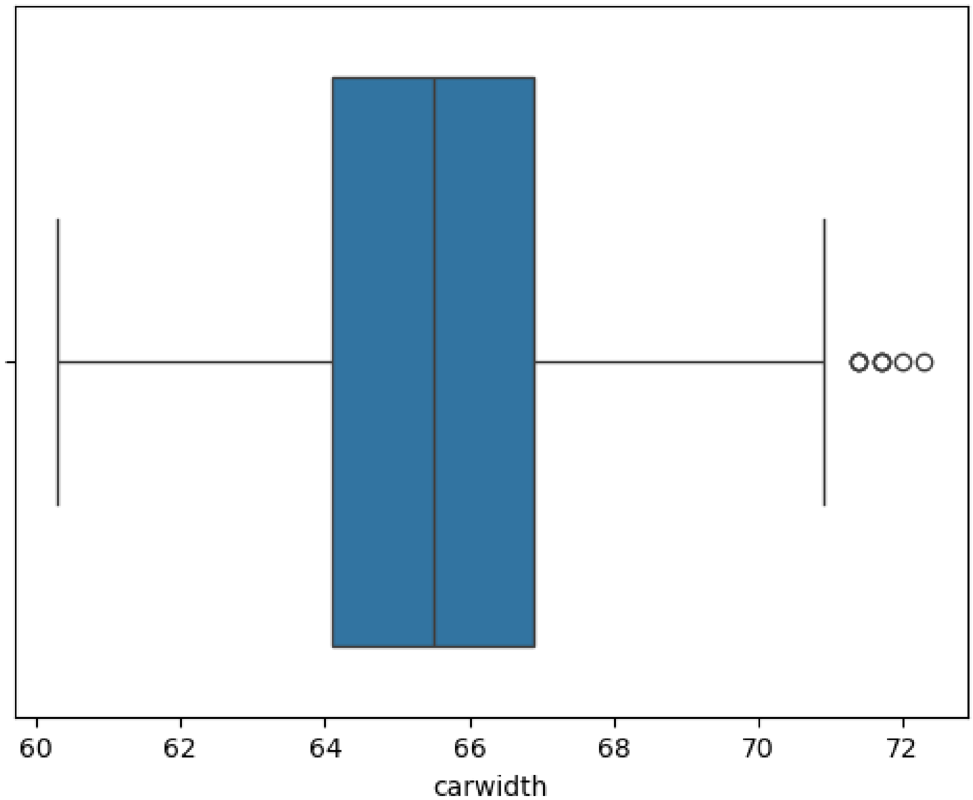

0
25
50
75
100
125
150
175
200
car_ID

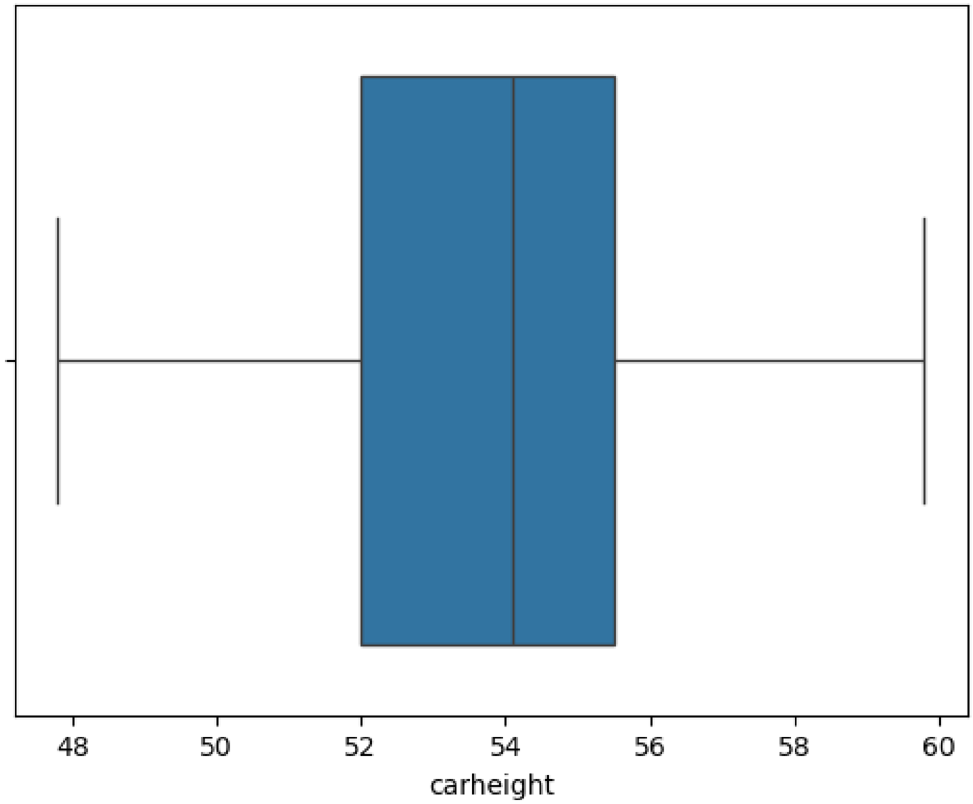


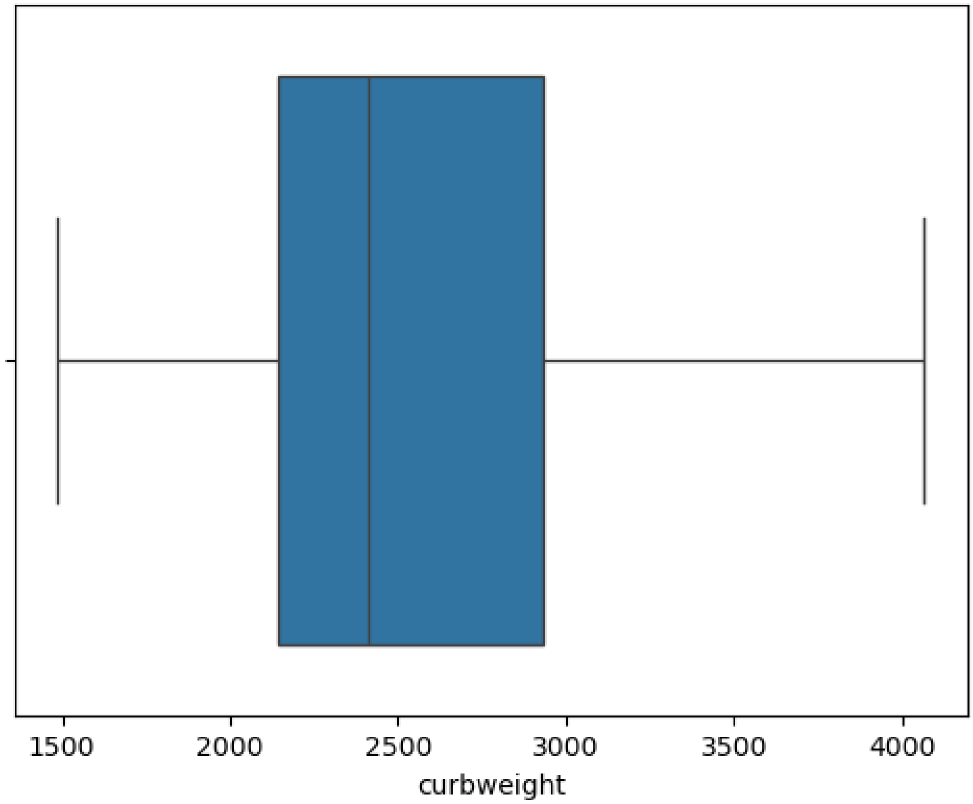
−2
−1
0
1
2
3
symboling

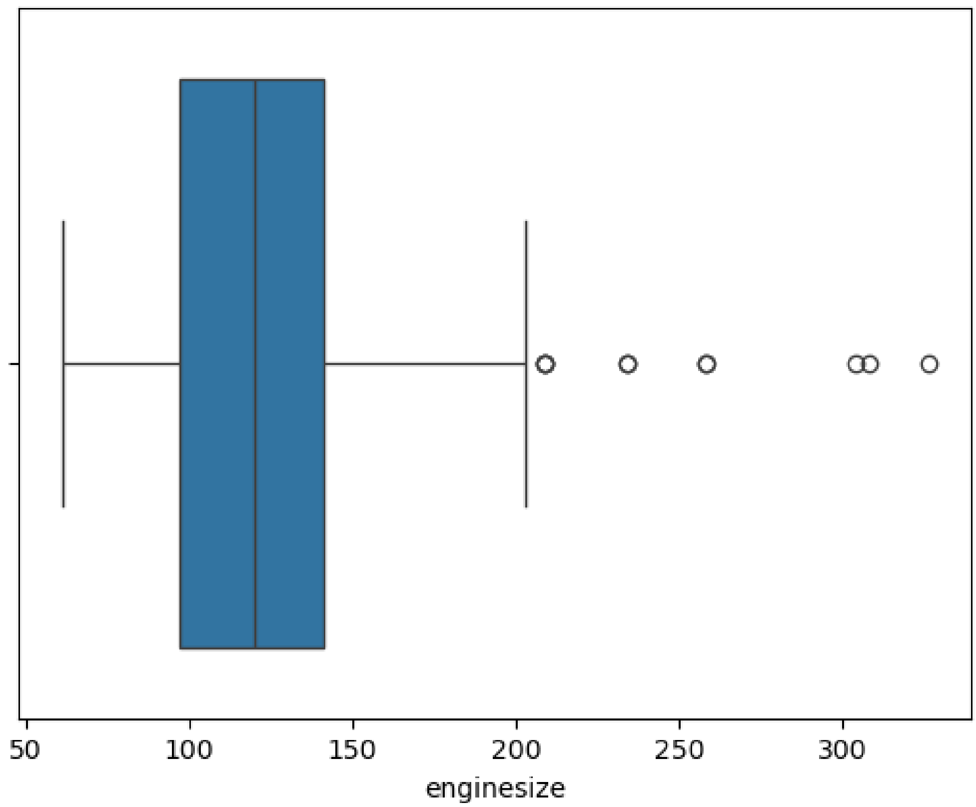

85
90
95
100
105
110
115
120
wheelbase


140
150
160
170
180
190
200
210
carlength

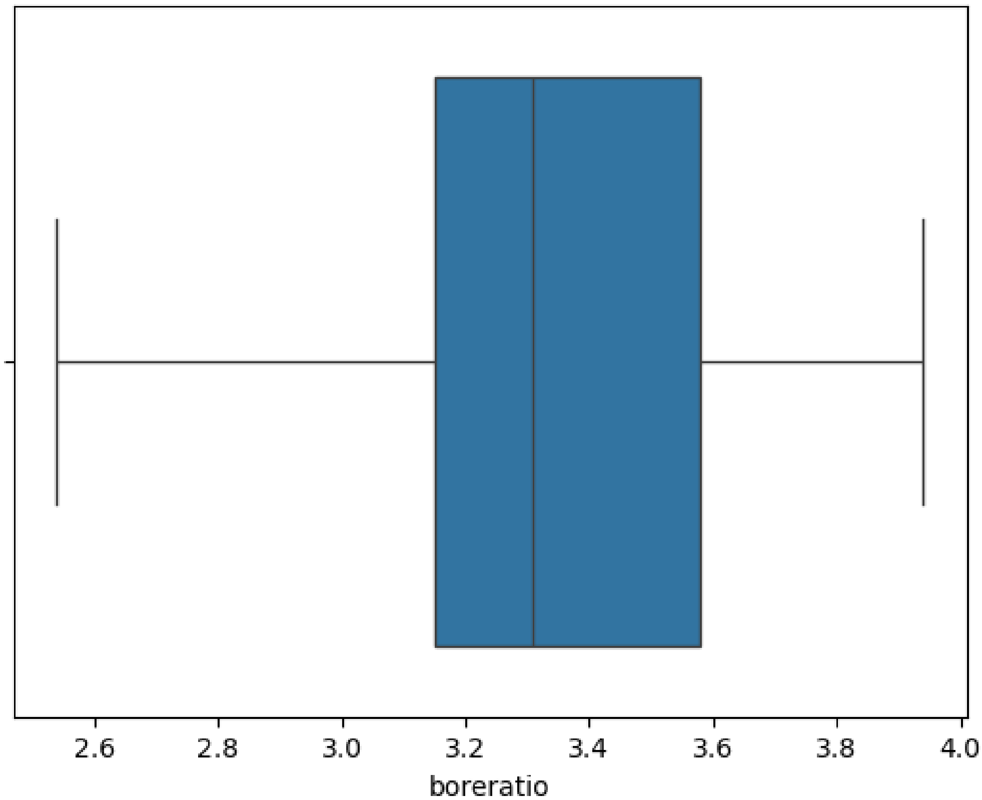

60
62
64
66
68
70
72
carwidth

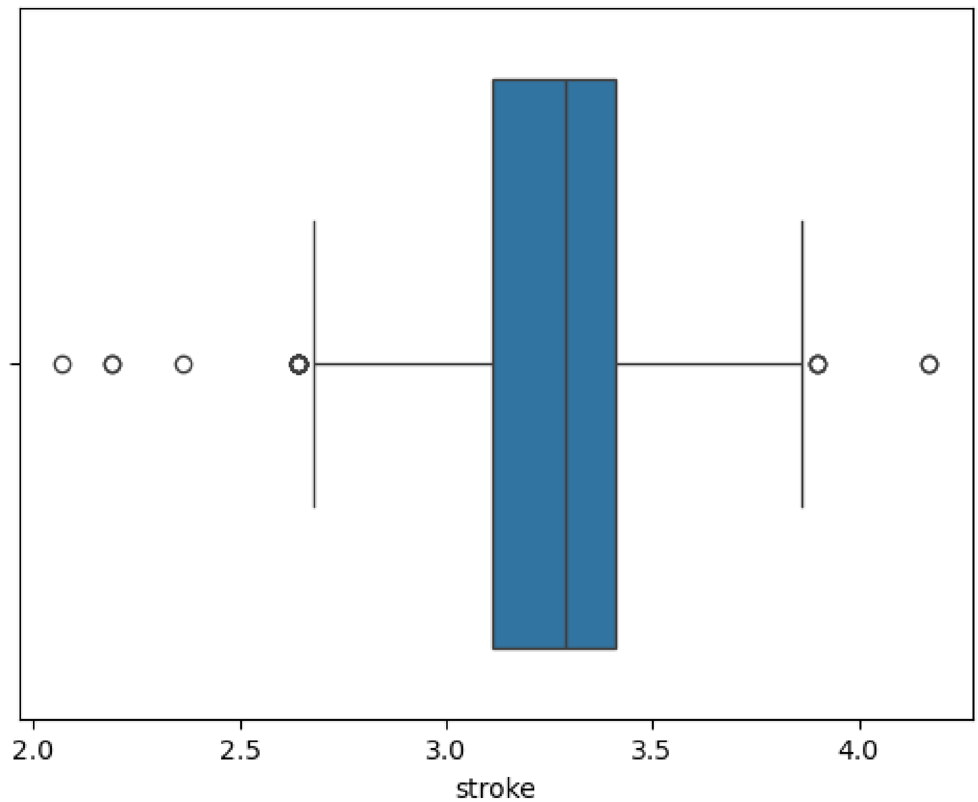

48
50
52
54
56
58
60
carheight

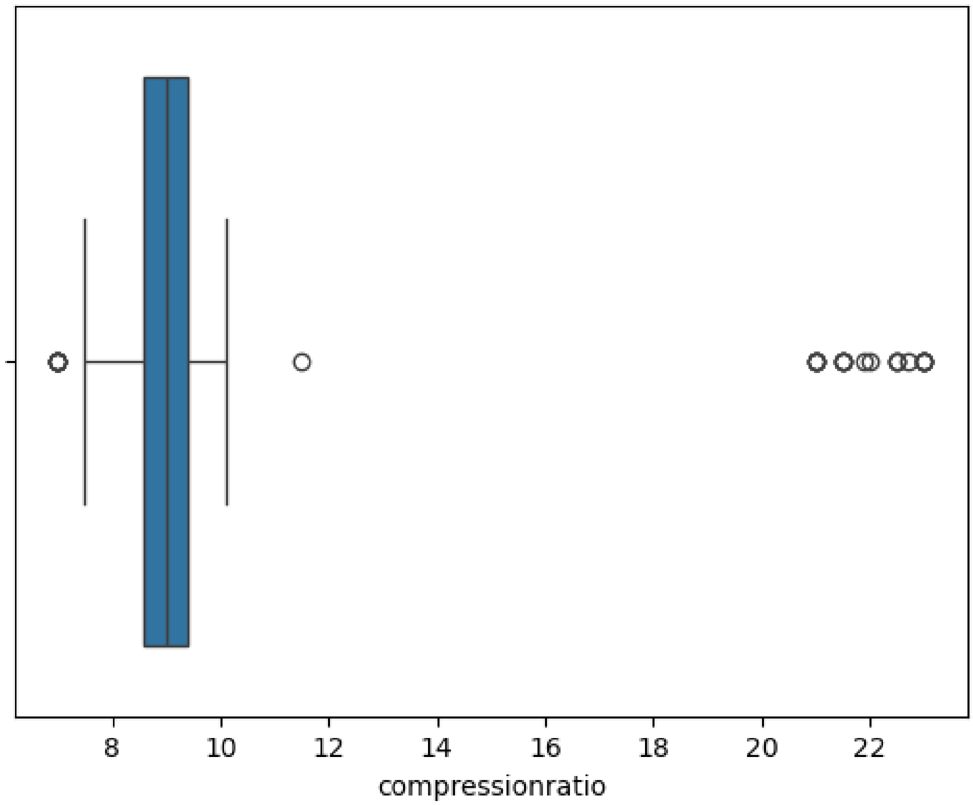

1500
2000
2500
3000
3500
4000
curbweight



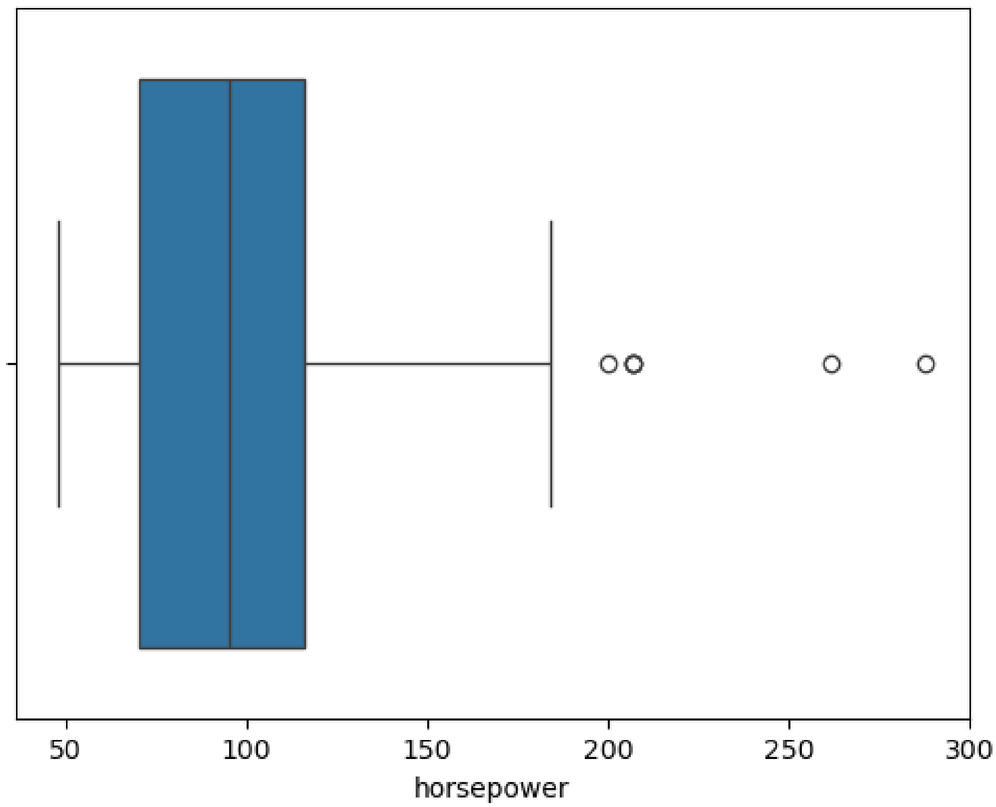
50
100
150
200
250
300
enginesize

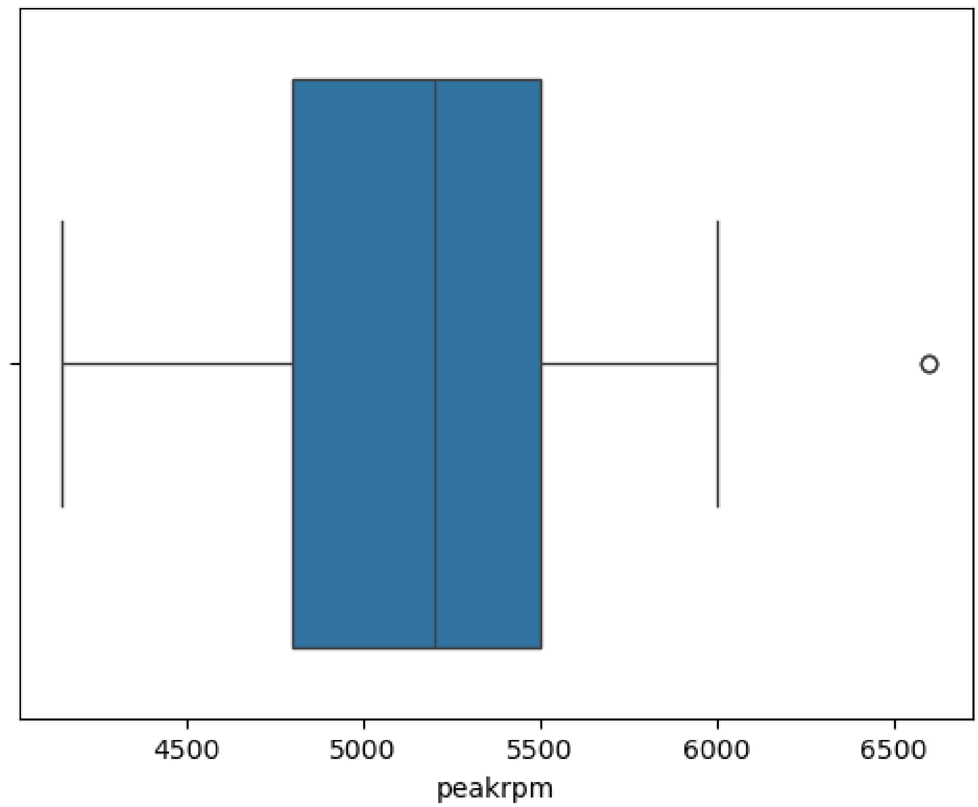

2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
boreratio

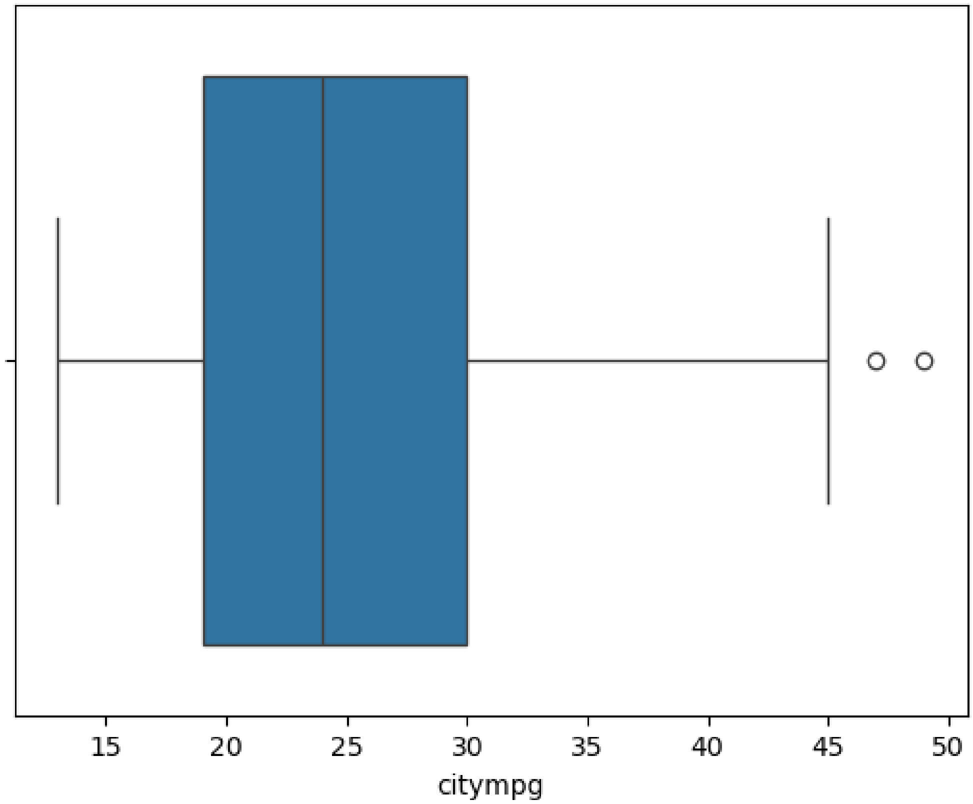

2.0
2.5
3.0
3.5
4.0
stroke

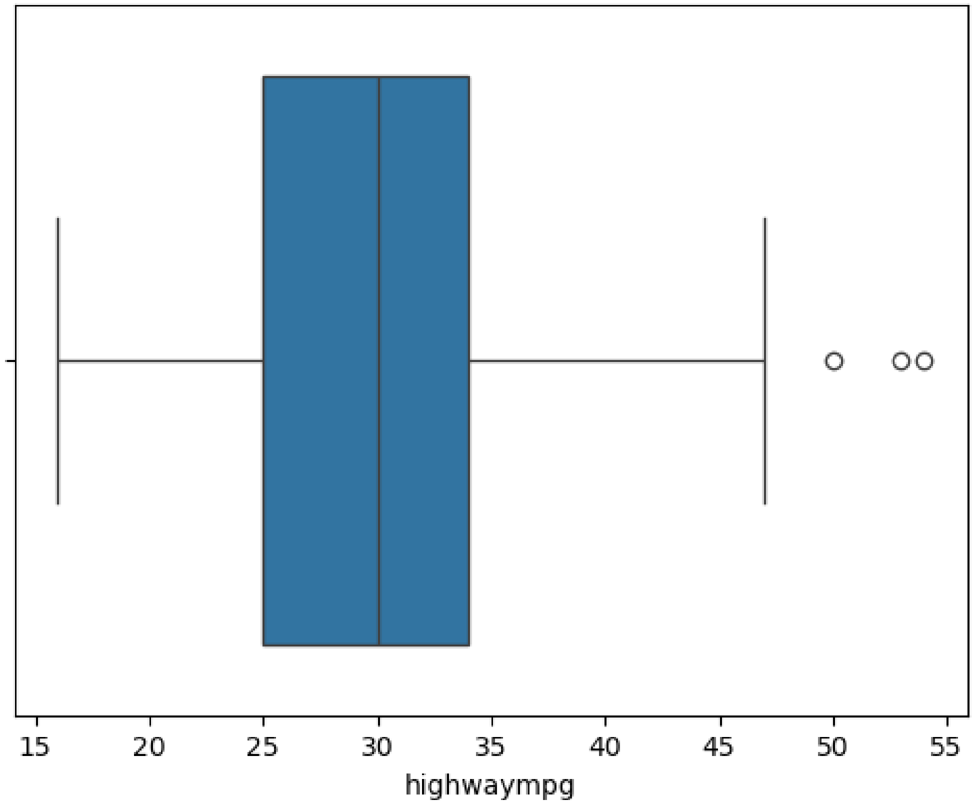


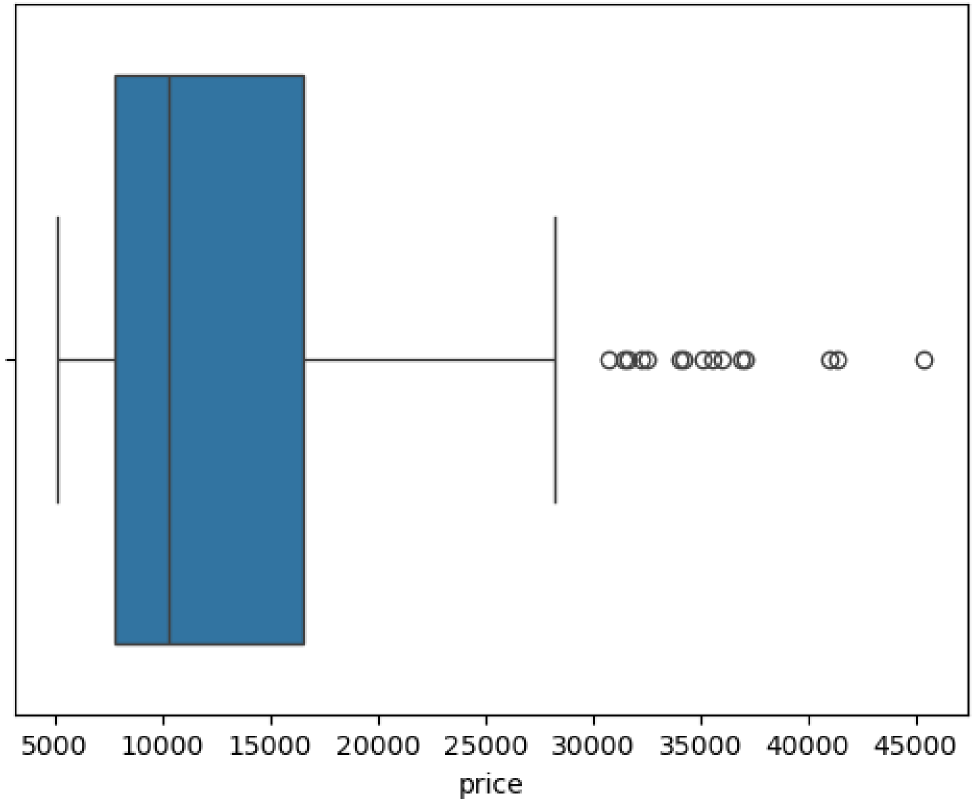
8
10
12
14
16
18
20
22
compressionratio


50
100
150
200
250
300
horsepower


4500
5000
5500
6000
6500
peakrpm


15
20
25
30
35
40
45
50
citympg

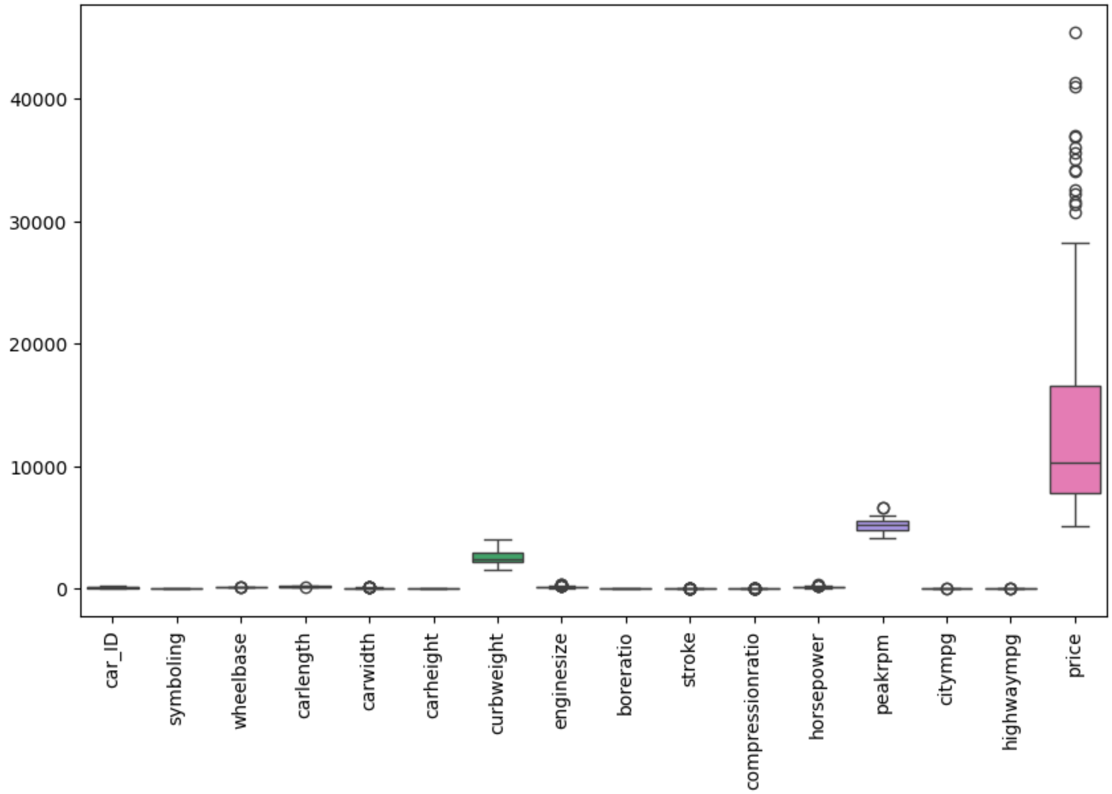

15
20
25
30
35
40
45
50
55
highwaympg



[59]:
```
numerical_columns = df.select_dtypes(include=['number']).columns
plt.figure(figsize=(10,6))
sns.boxplot(data = df[numerical_columns])
plt.xticks(rotation=90)
```

[59]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
 [Text(0, 0, 'car_ID'),
  Text(1, 0, 'symboling'),
  Text(2, 0, 'wheelbase'),
  Text(3, 0, 'carlength'),
  Text(4, 0, 'carwidth'),
  Text(5, 0, 'carheight'),
  Text(6, 0, 'curbweight'),
  Text(7, 0, 'enginesize'),
  Text(8, 0, 'boreratio'),
  Text(9, 0, 'stroke'),
  Text(10, 0, 'compressionratio'),
  Text(11, 0, 'horsepower'),
  Text(12, 0, 'peakrpm'),
  Text(13, 0, 'citympg'),
  Text(14, 0, 'highwaympg'),
```

```
Text(15, 0, 'price')])
```



Ouliers Found

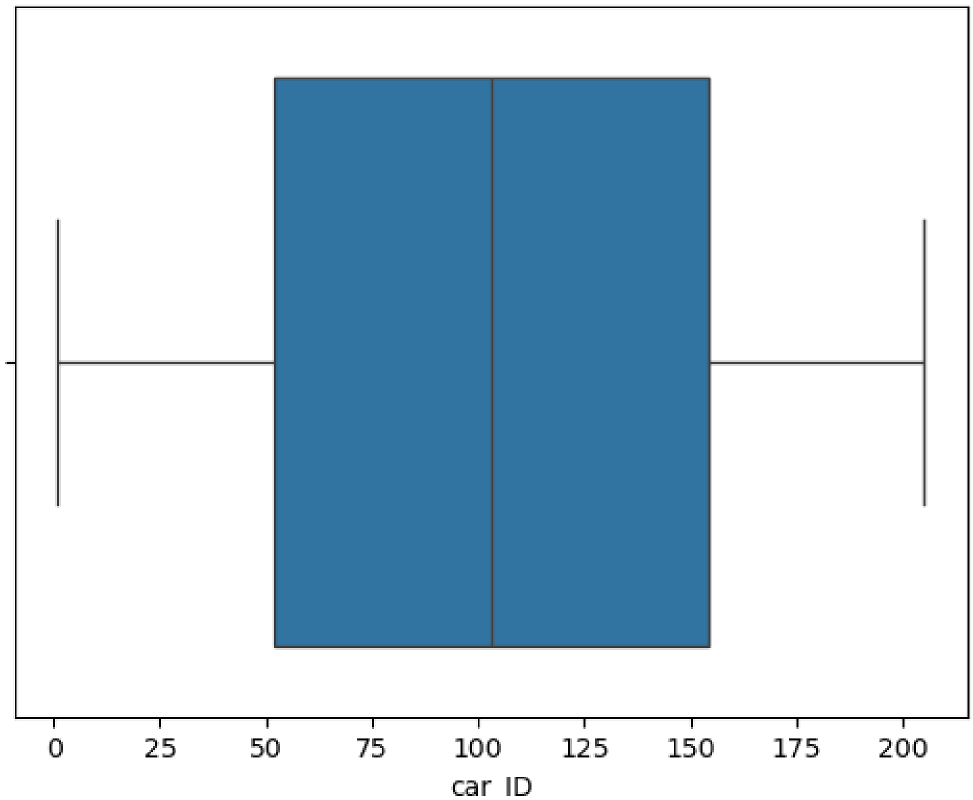
1.7.3 IQR Method

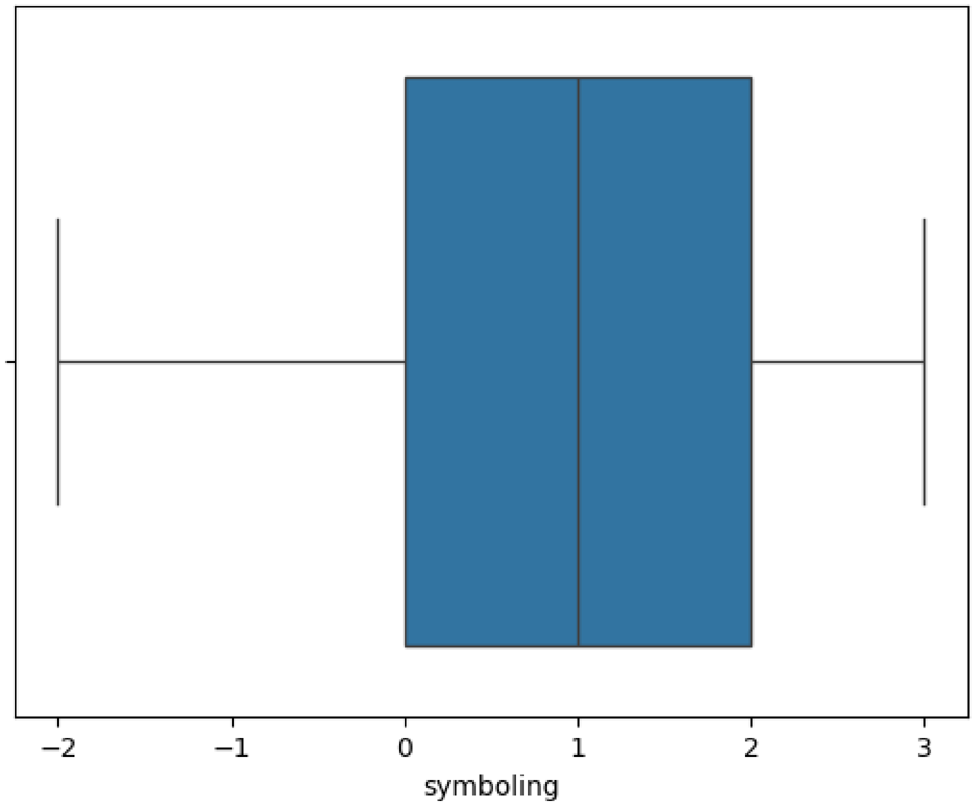
[64]:
```
# List of features with potential outliers
features = ['wheelbase', 'carlength', 'carwidth', 'enginesize', 'stroke',
            'compressionratio', 'horsepower', 'peakrpm', 'citympg',␣
 ↪'highwaympg', 'price']

# Function to apply IQR method to fix outliers
def fix_outliers_iqr(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Replace outliers with the respective bounds
```

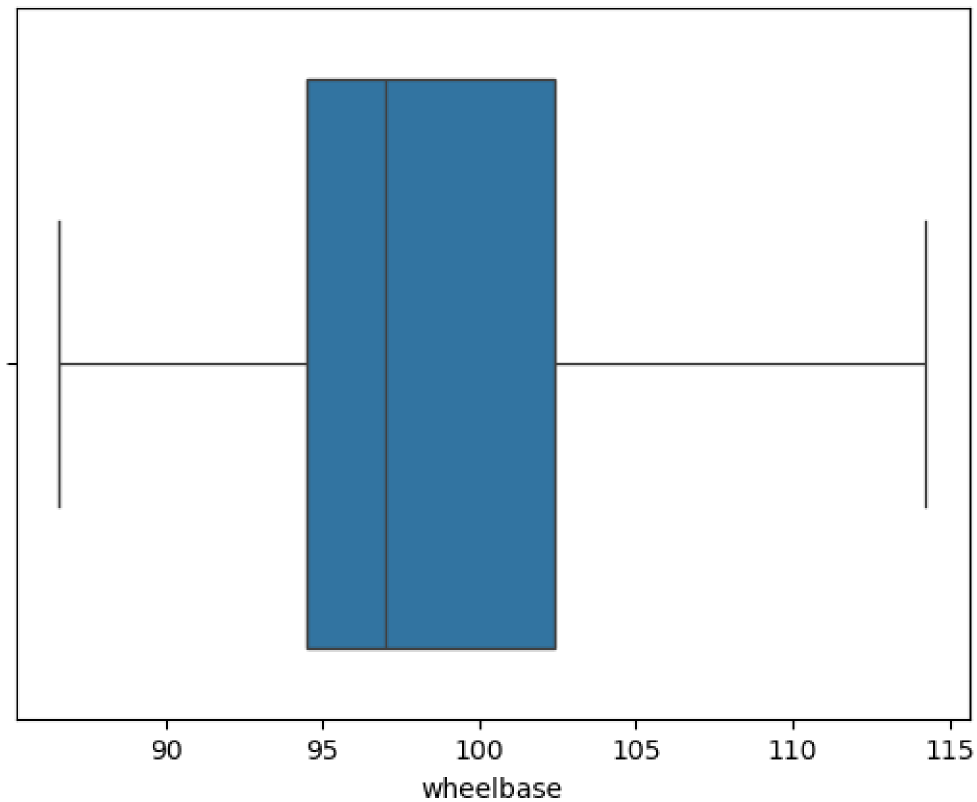
```
        df[col] = df[col].apply(lambda x: lower_bound if x < lower_bound else␣
 ↪upper_bound if x > upper_bound else x)

    return df


df = fix_outliers_iqr(df, features)
```

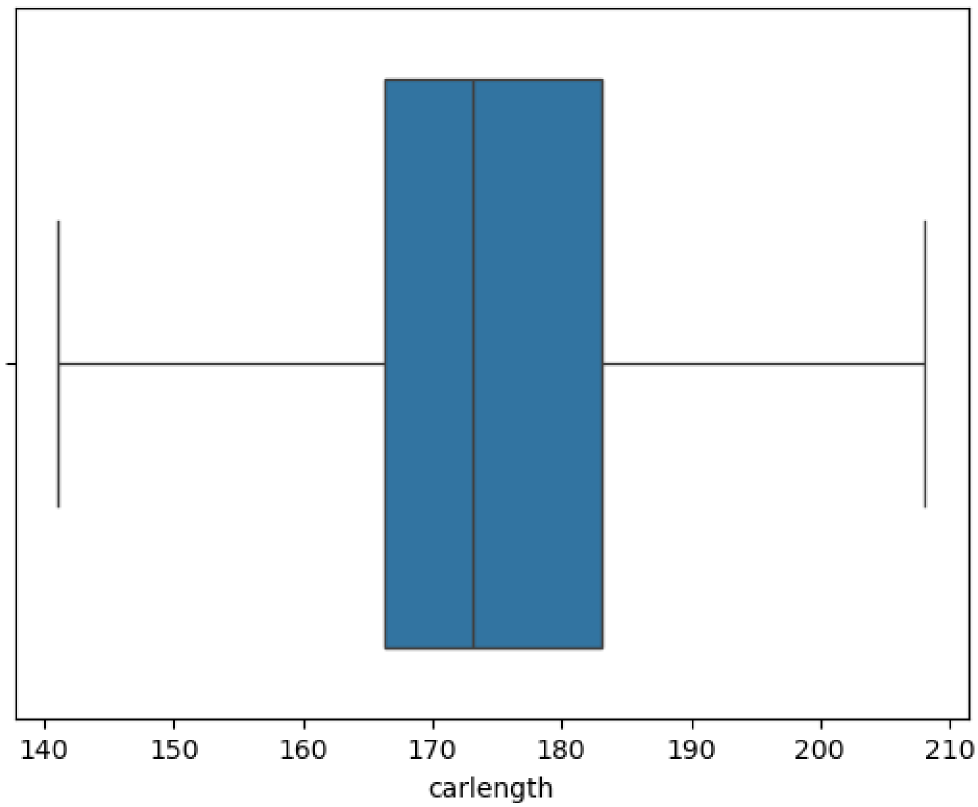
1.7.4 Visualising after Outlier detection

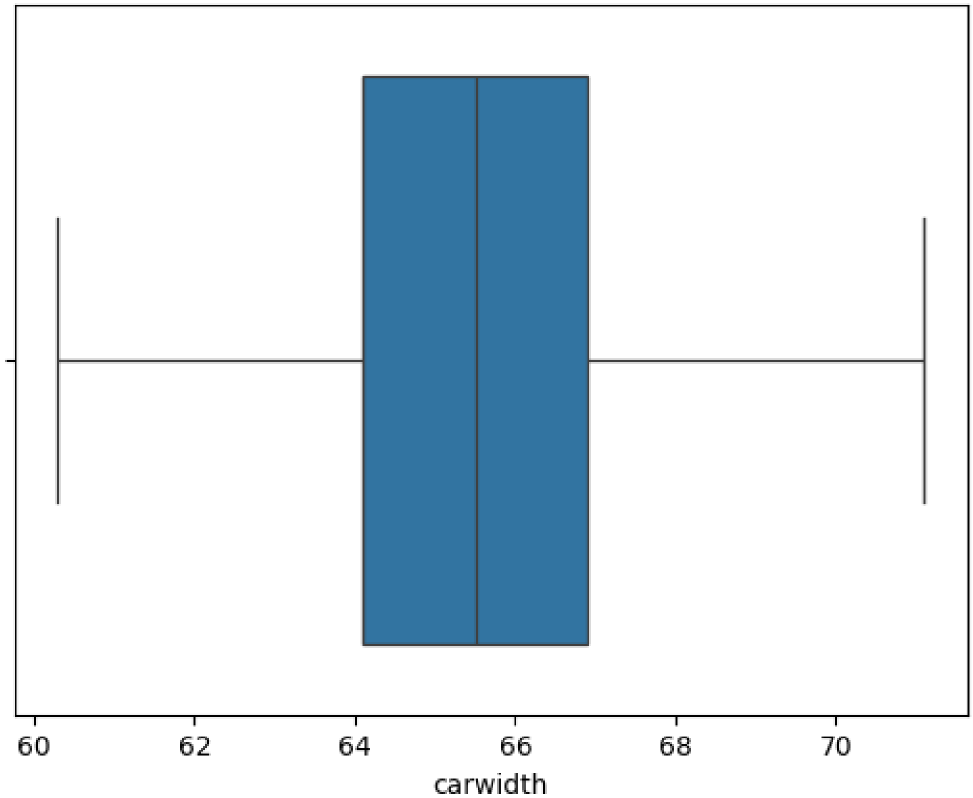
[67]:
```
# boxplot to identify outliers
for i in df.select_dtypes(include='number').columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```



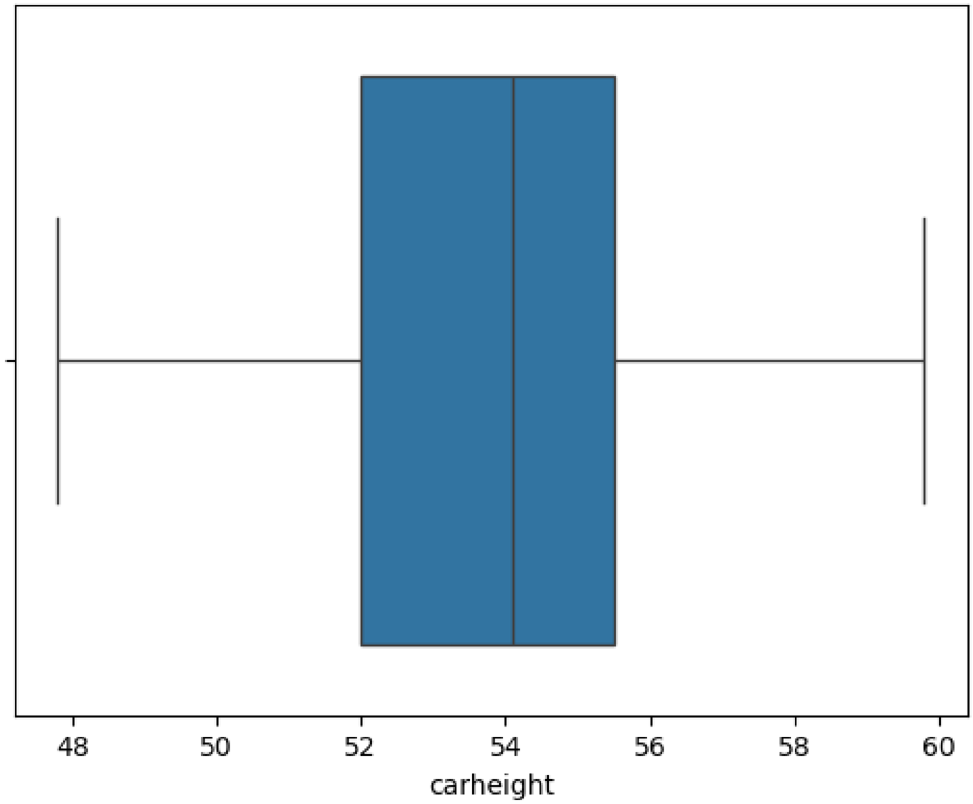

−2
−1
0
1
2
3
symboling

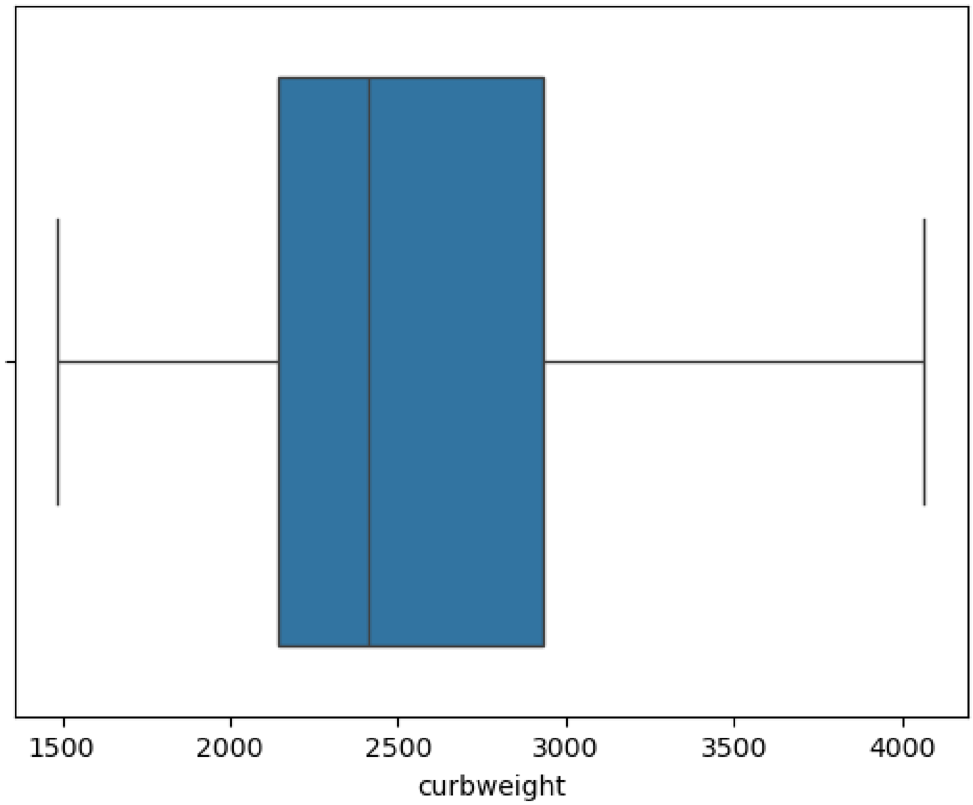

90
95
100
105
110
115
wheelbase

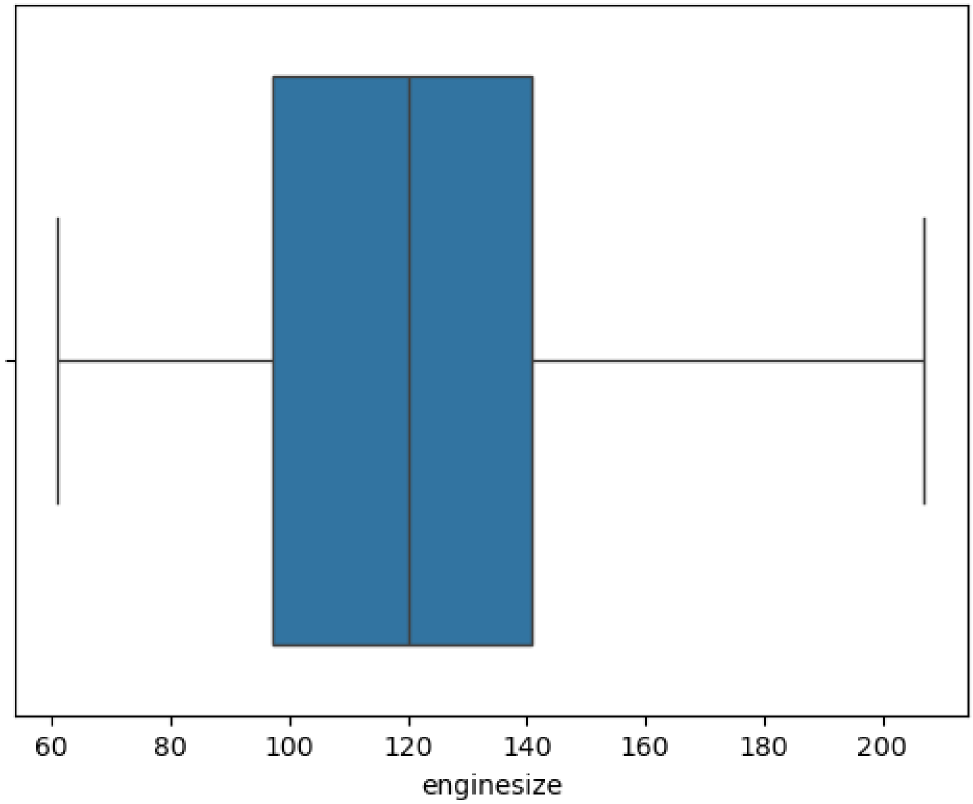

140
150
160
170
180
190
200
210
carlength

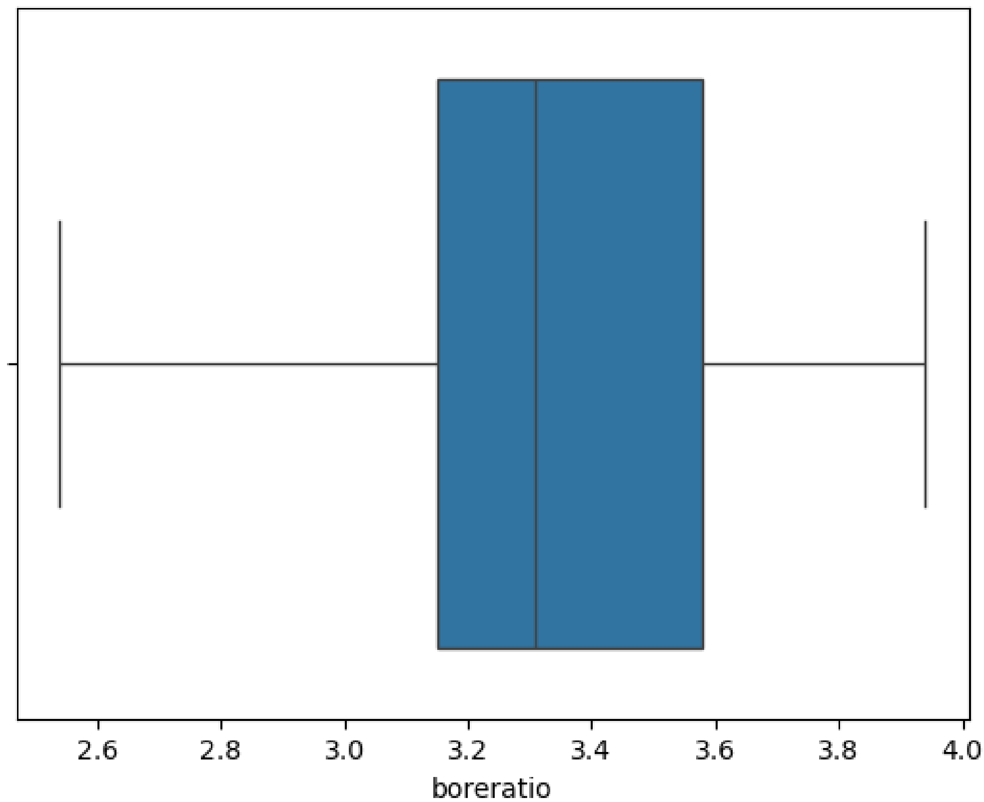


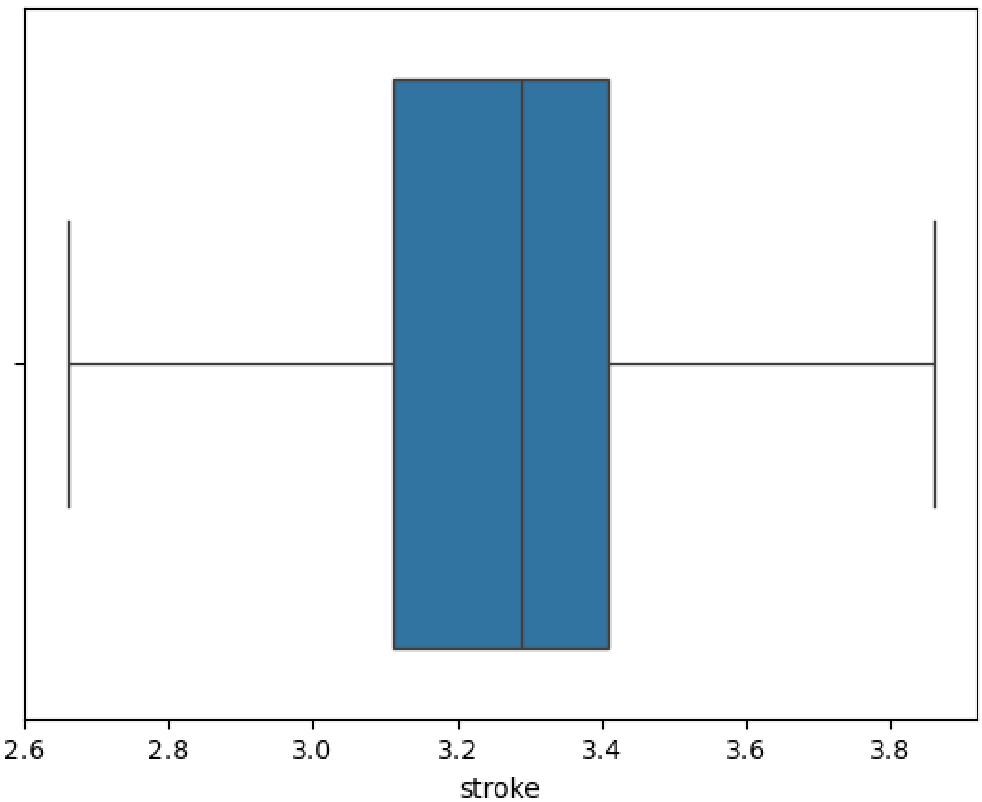
60
62
64
66
68
70
carwidth

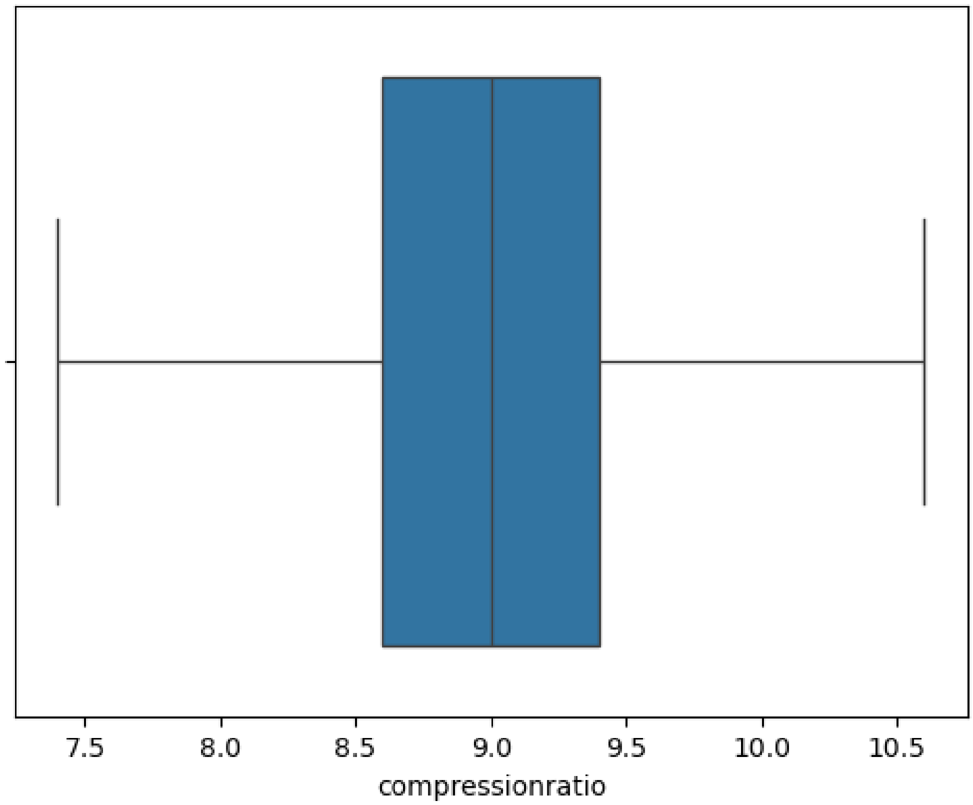

48
50
52
54
56
58
60
carheight

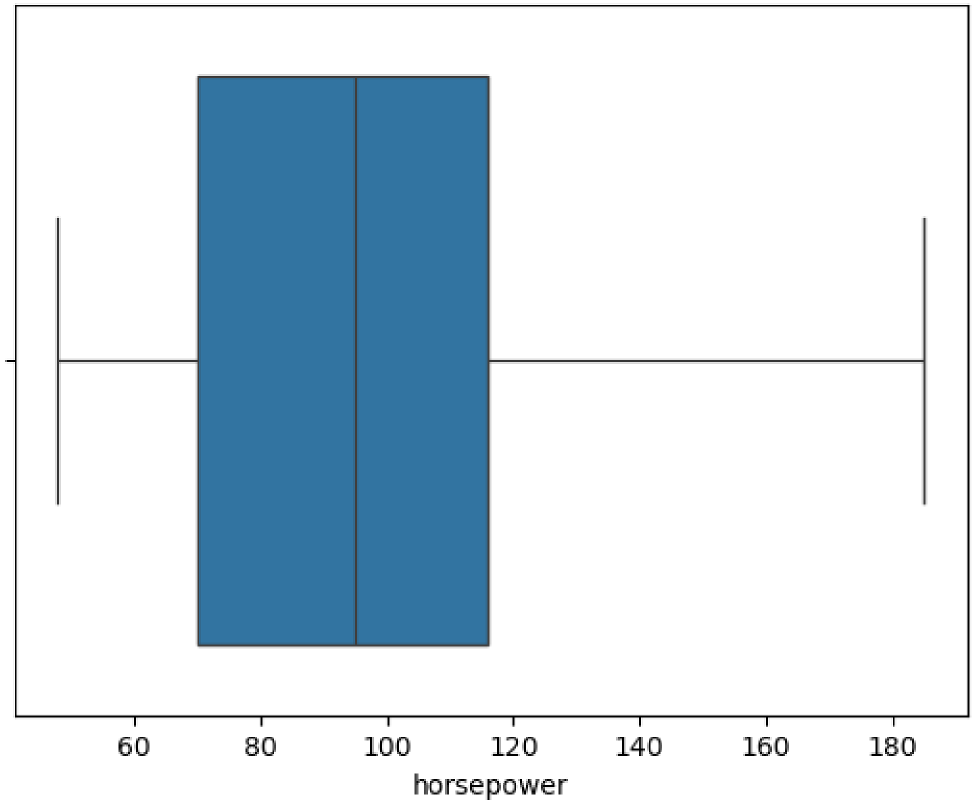

1500
2000
2500
3000
3500
4000
curbweight

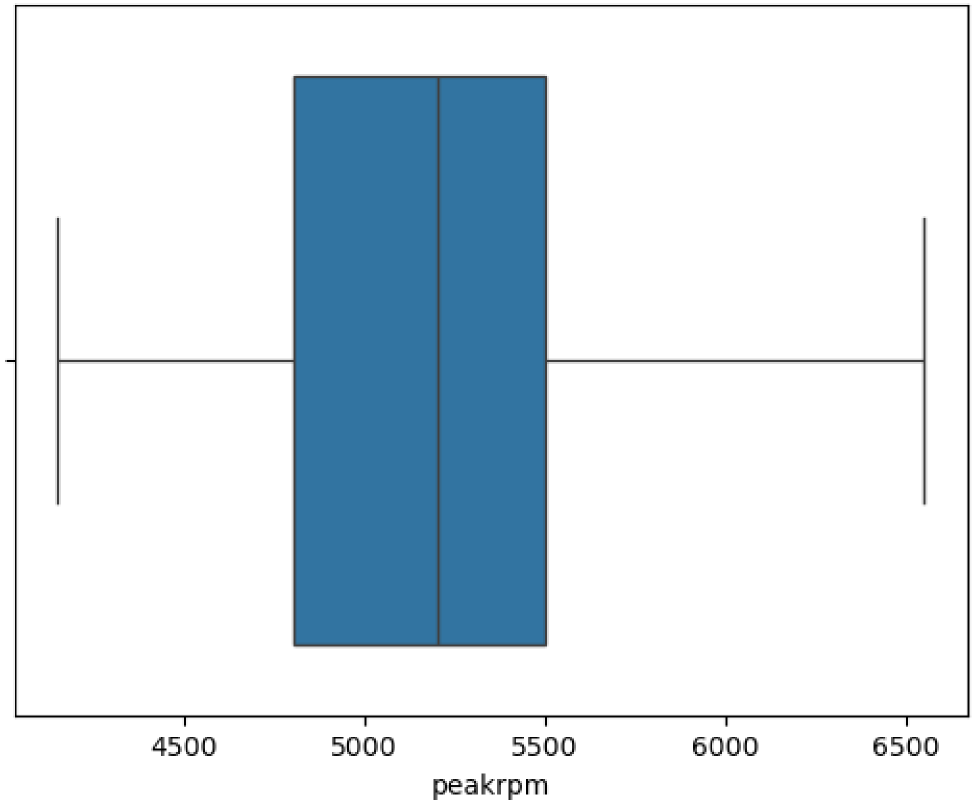

60
80
100
120
140
160
180
200
enginesize


2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
boreratio

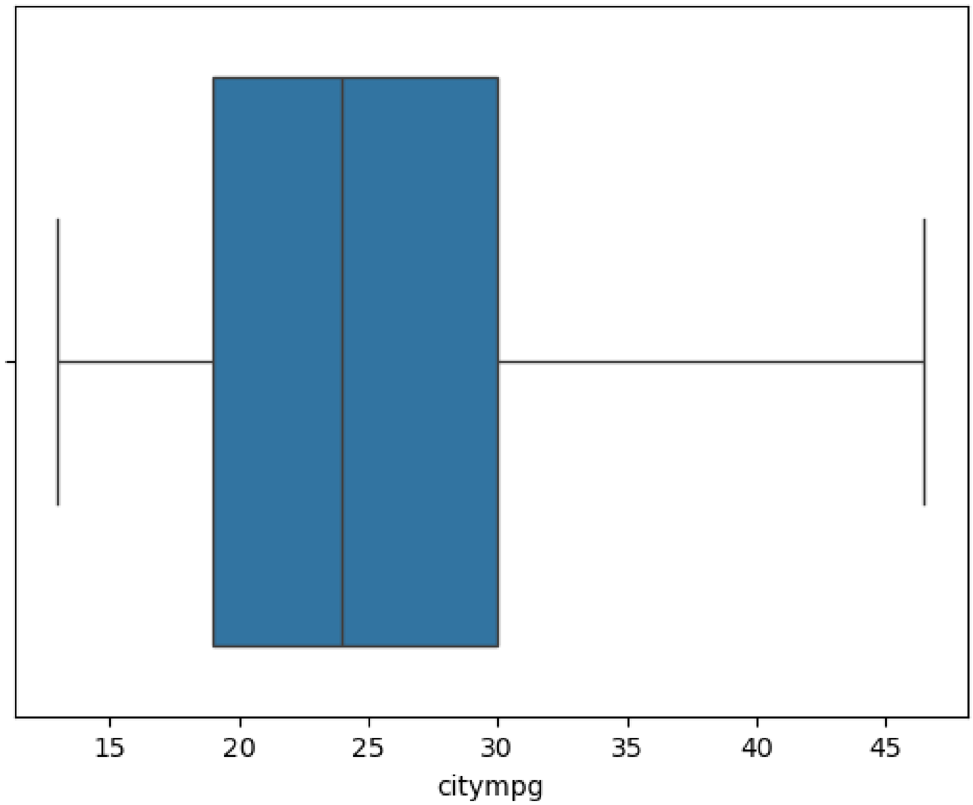

2.6
2.8
3.0
3.2
3.4
3.6
3.8
stroke

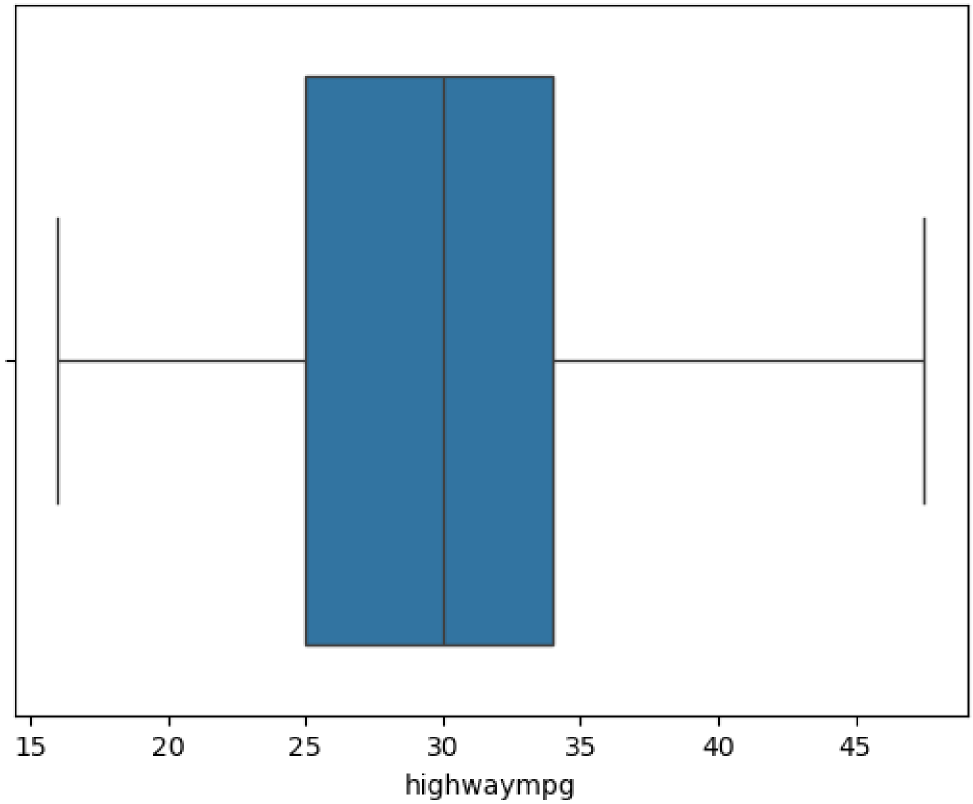

7.5
8.0
8.5
9.0
9.5
10.0
10.5
compressionratio

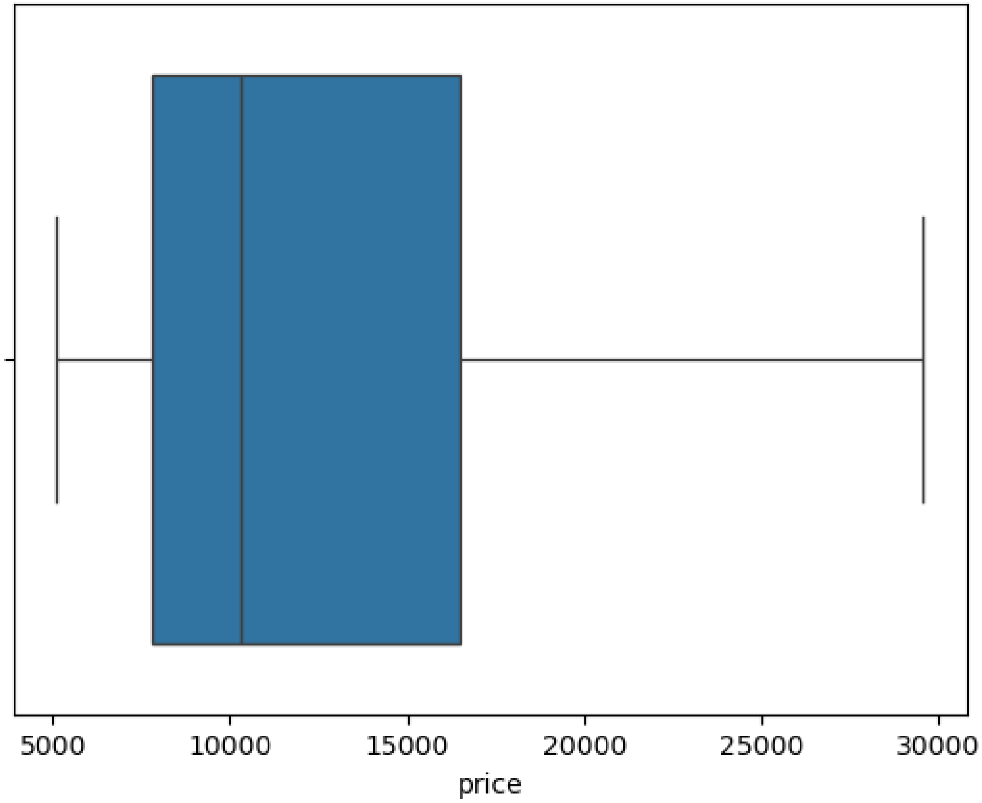

60
80
100
120
140
160
180
horsepower



4500
5000
5500
6000
6500
peakrpm


15
20
25
30
35
40
45
citympg

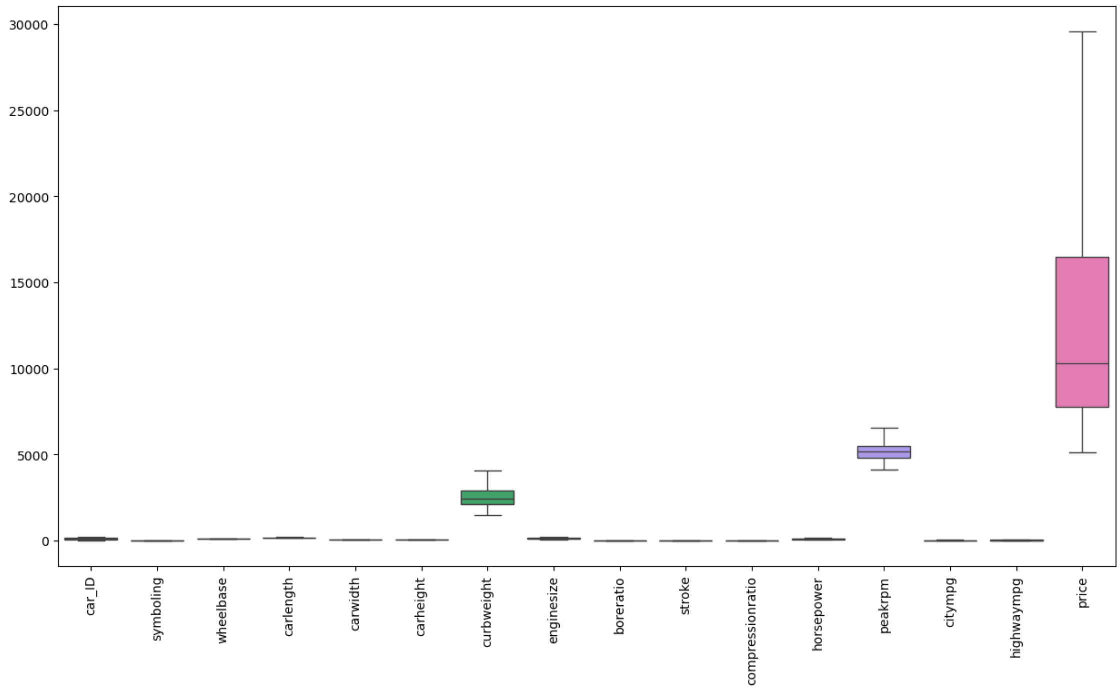

15
20
25
30
35
40
45
highwaympg



[76]:
```
numerical_columns = df.select_dtypes(include=['number']).columns
plt.figure(figsize=(15,8))
sns.boxplot(data = df[numerical_columns])
plt.xticks(rotation=90)
```

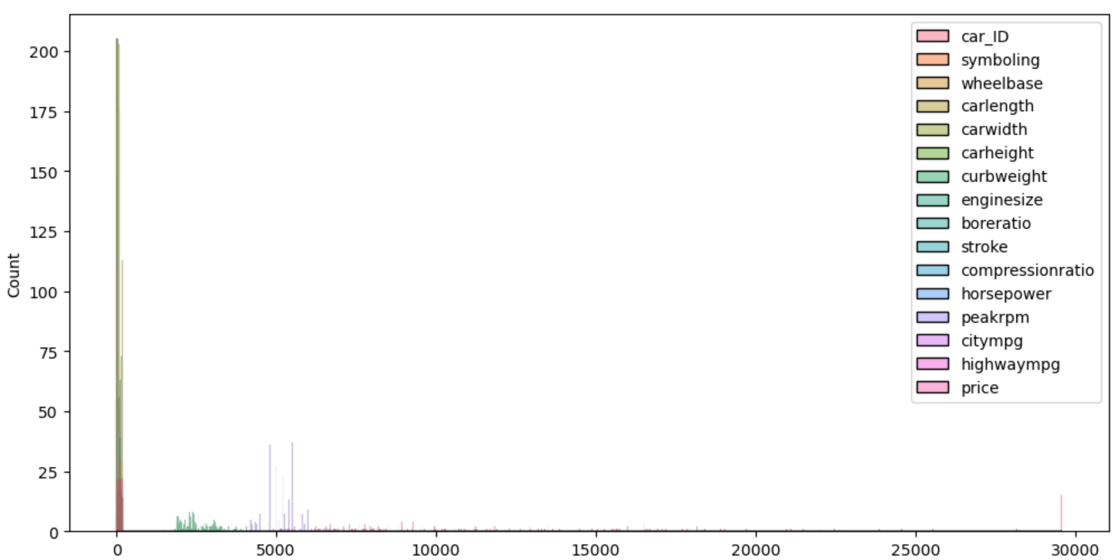
[76]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
 [Text(0, 0, 'car_ID'),
  Text(1, 0, 'symboling'),
  Text(2, 0, 'wheelbase'),
  Text(3, 0, 'carlength'),
  Text(4, 0, 'carwidth'),
  Text(5, 0, 'carheight'),
  Text(6, 0, 'curbweight'),
  Text(7, 0, 'enginesize'),
  Text(8, 0, 'boreratio'),
  Text(9, 0, 'stroke'),
  Text(10, 0, 'compressionratio'),
  Text(11, 0, 'horsepower'),
  Text(12, 0, 'peakrpm'),
  Text(13, 0, 'citympg'),
  Text(14, 0, 'highwaympg'),
```

```
Text(15, 0, 'price')])
```



1.8 Checking Skew

[90]:
```
# visualising
plt.figure(figsize=(12,6))
sns.histplot(df[numerical_columns])
plt.show()
```

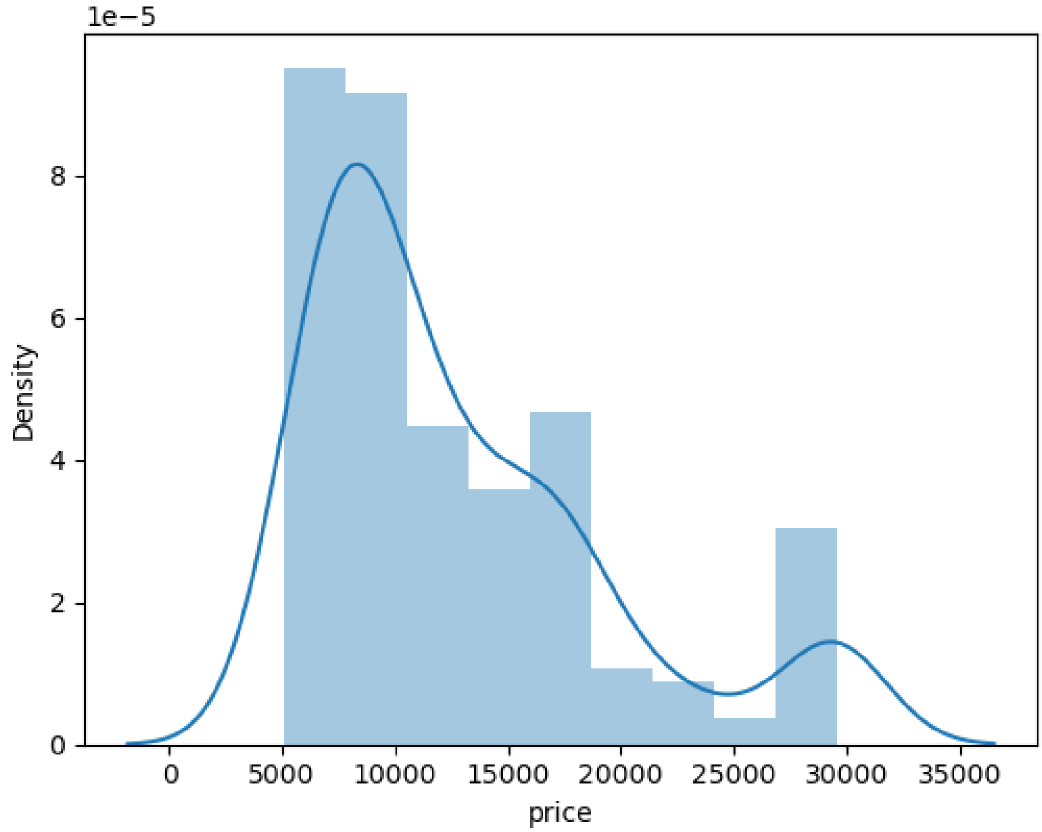


[84]:
```
df[numerical_columns].skew()
```

[84]:
```
car_ID              0.000000
symboling           0.211072
wheelbase           0.924916
carlength           0.155954
carwidth            0.776278
carheight           0.063123
curbweight          0.681398
enginesize          0.908453
boreratio           0.020156
stroke             -0.379130
compressionratio    0.035149
horsepower          0.814957
peakrpm             0.049935
citympg             0.604594
highwaympg          0.347441
price               1.222031
dtype: float64
```

[92]:
```
sns.distplot(df['price'])
```

[92]:
```
<Axes: xlabel='price', ylabel='Density'>
```



Skewness in high for 'price'

1.8.1 Fixing skewness using log transformation

[99]:
```
df['price'] = np.log(df['price'])
df['wheelbase'] = np.log(df['wheelbase'])
df['carwidth'] = np.log(df['carwidth'])
df['enginesize'] = np.log(df['enginesize'])
df['compressionratio'] = np.log(df['compressionratio'])
df['horsepower'] = np.log(df['horsepower'])
```

[107]:
```
df[numerical_columns].skew()
```

[107]:
```
car_ID              0.000000
symboling           0.211072
wheelbase           0.781960
carlength           0.155954
carwidth            0.696287
carheight           0.063123
curbweight          0.681398
```

```
enginesize          0.401418
boreratio           0.020156
stroke             -0.379130
compressionratio   -0.241483
horsepower          0.287093
peakrpm             0.049935
citympg             0.604594
highwaympg          0.347441
price               0.459254
dtype: float64
```

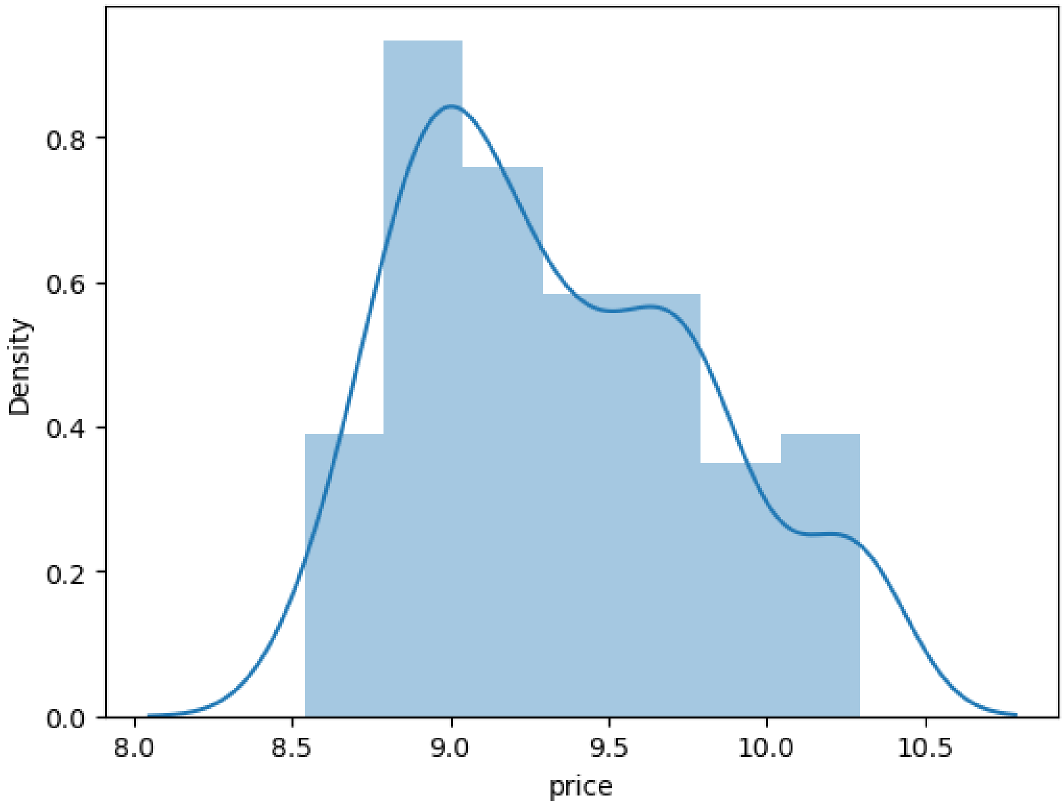
[109]:
```python
df['compressionratio'] = np.sqrt(df['compressionratio'])
df['wheelbase'] = np.sqrt(df['wheelbase'])
```

[111]:
```python
df[numerical_columns].skew()
```

[111]:
```
car_ID              0.000000
symboling           0.211072
wheelbase           0.766142
carlength           0.155954
carwidth            0.696287
carheight           0.063123
curbweight          0.681398
enginesize          0.401418
boreratio           0.020156
stroke             -0.379130
compressionratio   -0.304199
horsepower          0.287093
peakrpm             0.049935
citympg             0.604594
highwaympg          0.347441
price               0.459254
dtype: float64
```

[113]:
```python
sns.distplot(df['price'])
```

[113]:
```
<Axes: xlabel='price', ylabel='Density'>
```



[115]:
```
df[numerical_columns].skew()
```

[115]:
```
car_ID              0.000000
symboling           0.211072
wheelbase           0.766142
carlength           0.155954
carwidth            0.696287
carheight           0.063123
curbweight          0.681398
enginesize          0.401418
boreratio           0.020156
stroke             -0.379130
compressionratio   -0.304199
horsepower          0.287093
peakrpm             0.049935
citympg             0.604594
highwaympg          0.347441
price               0.459254
dtype: float64
```

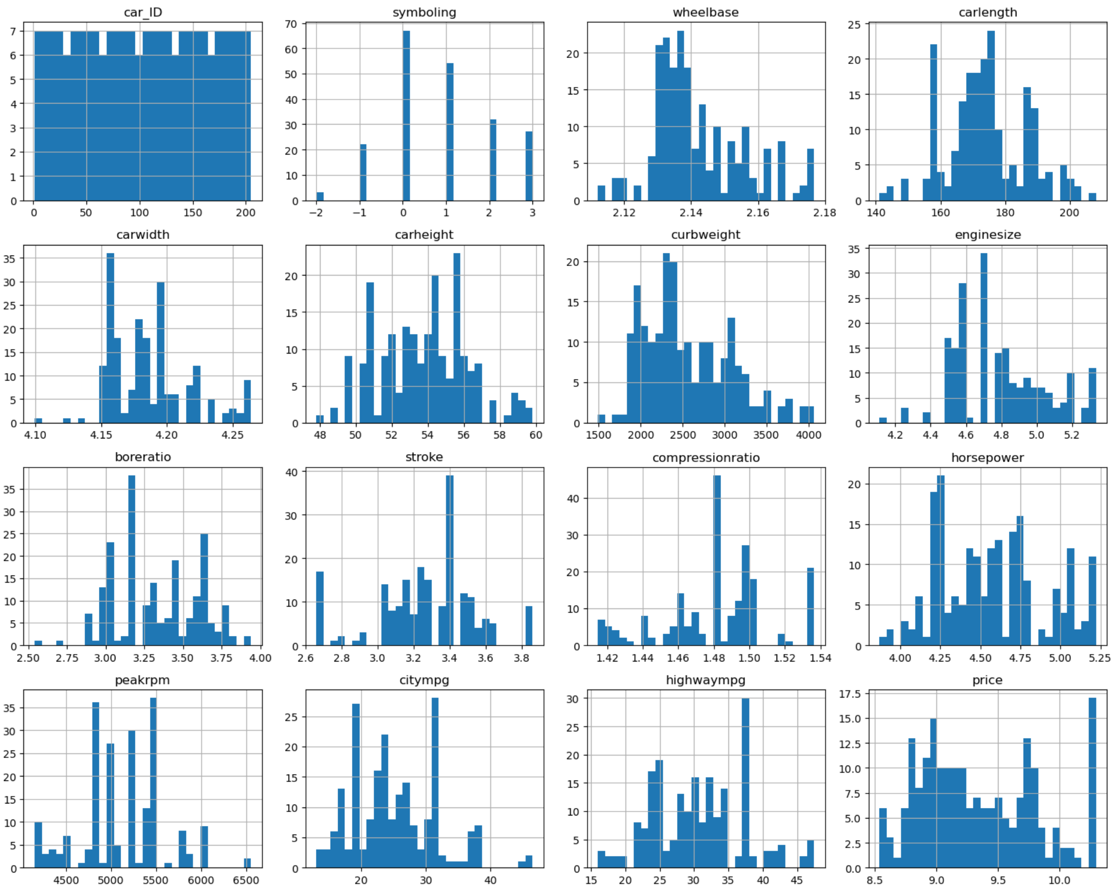
1.9 6. EDA

[118]:
```python
df1 = df.copy()
```

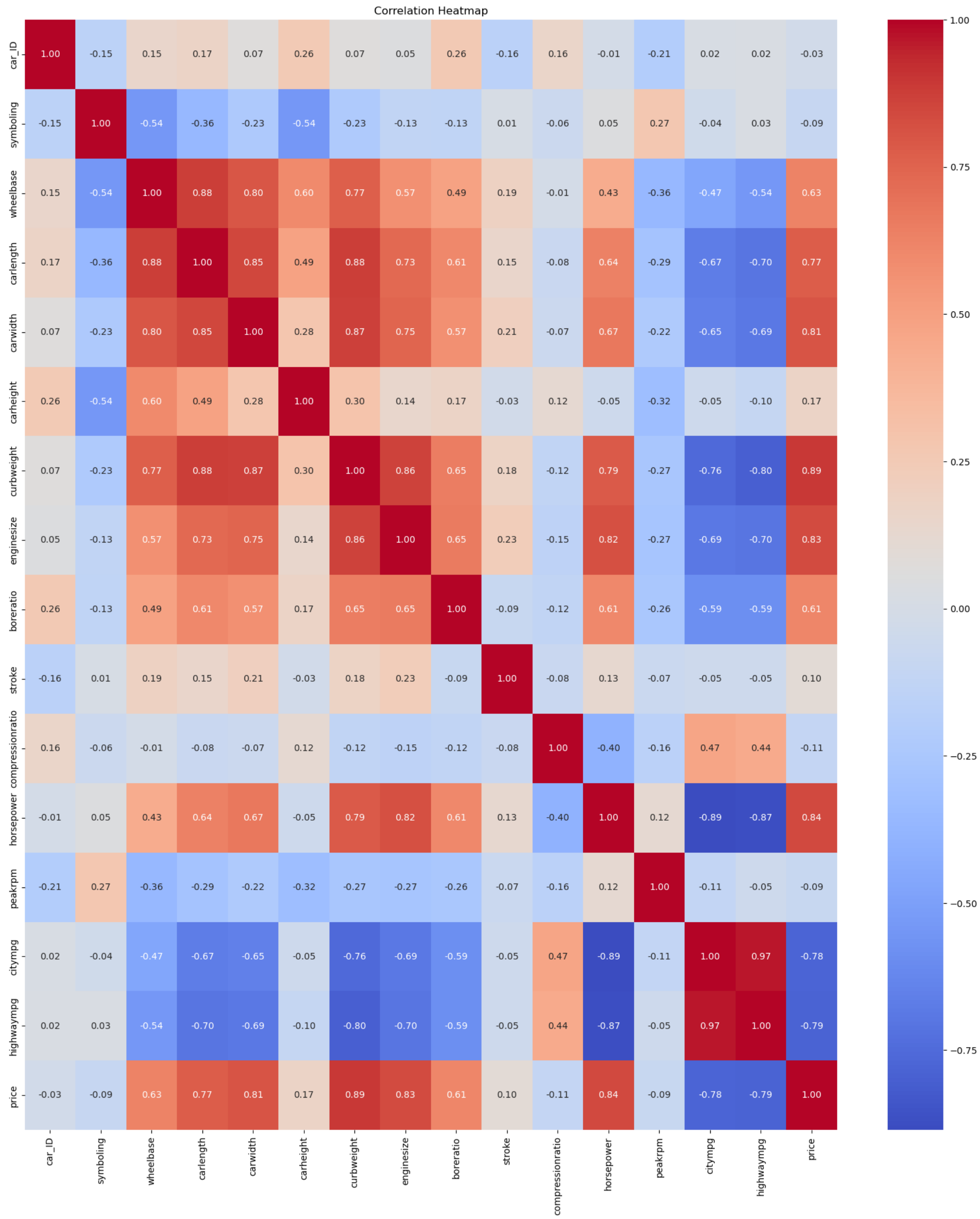
[120]:
```python
df1.head()
```

[120]:
```
   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   2.117577  …    4.867534
1  convertible        rwd          front   2.117577  …    4.867534
2    hatchback        rwd          front   2.132745  …    5.023881
3        sedan        fwd          front   2.145500  …    4.691348
4        sedan        4wd          front   2.144563  …    4.912655

   fuelsystem  boreratio  stroke compressionratio horsepower  peakrpm citympg  \
0        mpfi       3.47    2.68         1.482304   4.709530   5000.0    21.0
1        mpfi       3.47    2.68         1.482304   4.709530   5000.0    21.0
2        mpfi       2.68    3.47         1.482304   5.036953   5000.0    19.0
3        mpfi       3.19    3.40         1.517427   4.624973   5500.0    24.0
4        mpfi       3.19    3.40         1.442027   4.744932   5500.0    18.0

   highwaympg     price
0        27.0  9.510075
1        27.0  9.711116
2        26.0  9.711116
3        30.0  9.543235
4        22.0  9.767095

[5 rows x 26 columns]
```

[122]:
```python
# Histogram
df1.hist(bins=30, figsize=(15, 12))
plt.tight_layout()
plt.show()
```

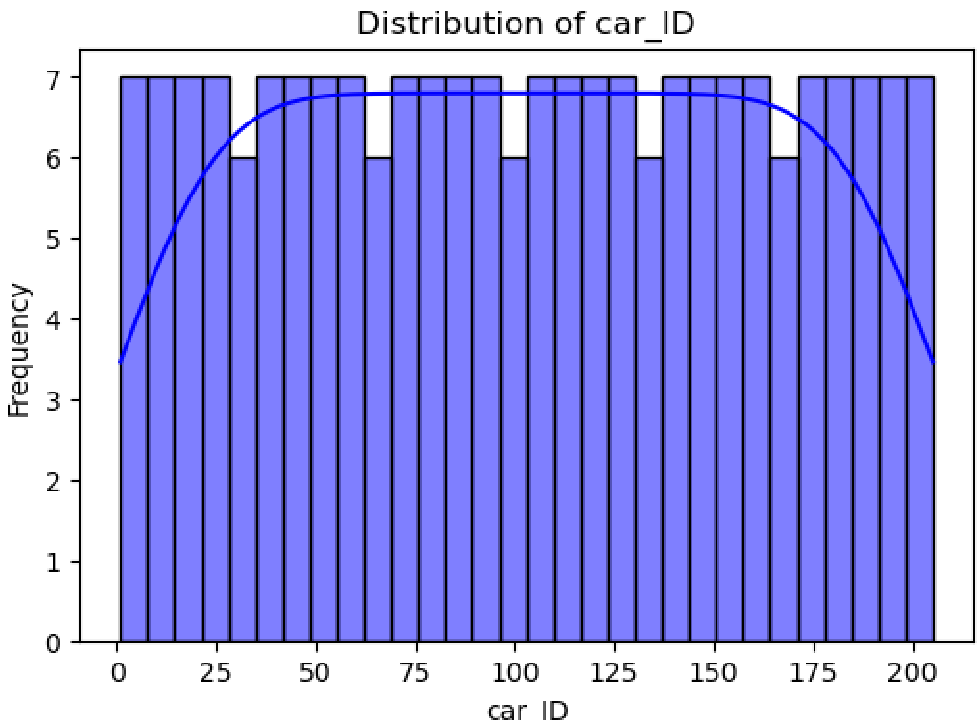


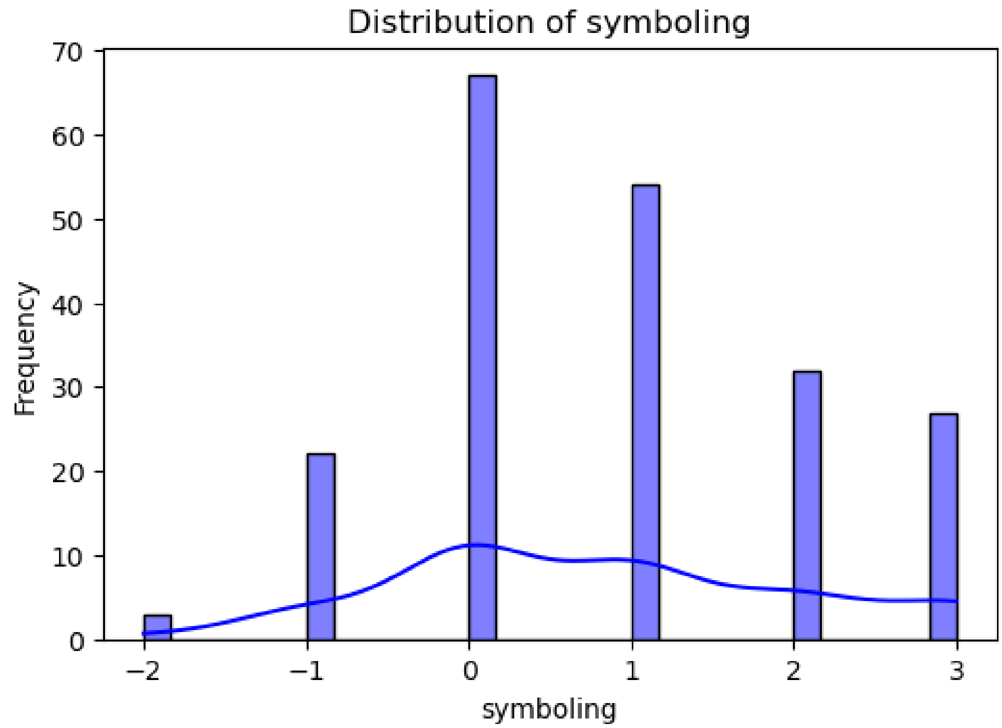
[132]:
```
#Compute correlation matrix
corr_matrix = df1[numerical_columns].corr()

# Heatmap
plt.figure(figsize=(20, 22))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



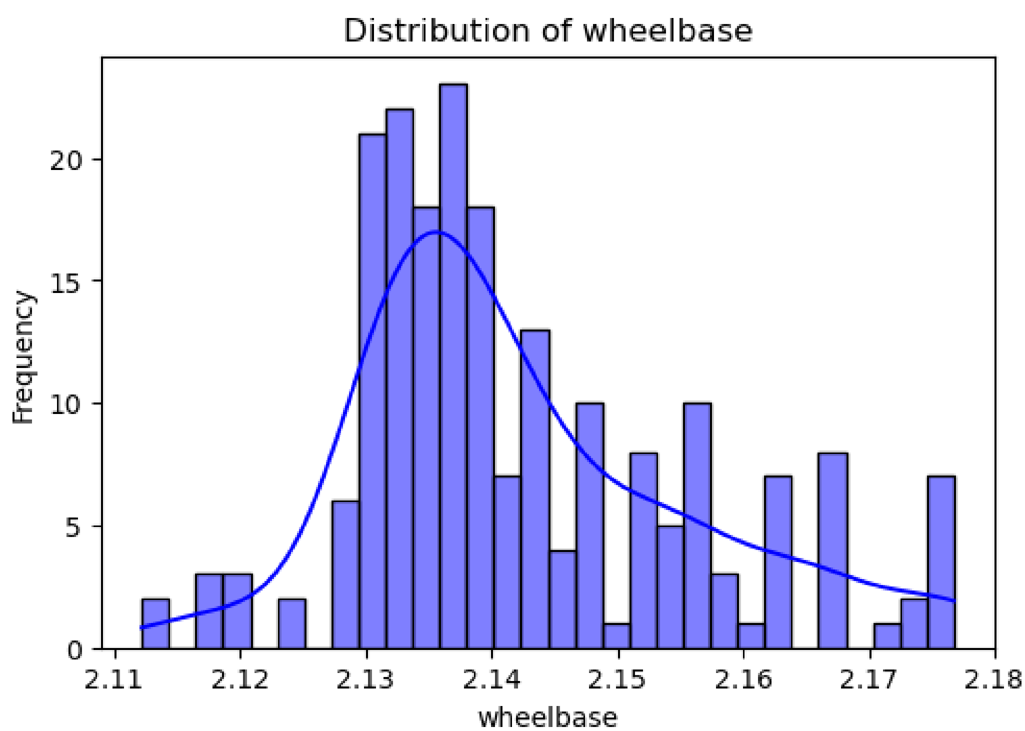
[172]:
```python
# Plot distribution for numerical column
for feature in numerical_columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(df1[feature], kde=True, bins=30, color='blue')
    plt.title(f'Distribution of {feature}')
```

```
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()
```



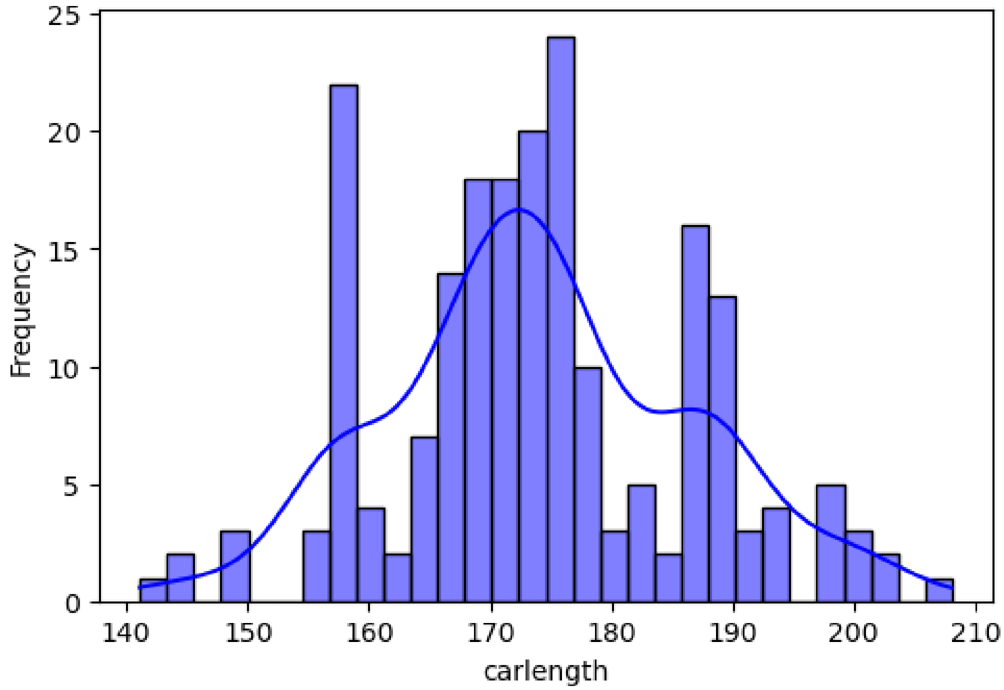


Distribution of symboling
Frequency
symboling
0
10
20
30
40
50
60
70
−2
−1
0
1
2
3

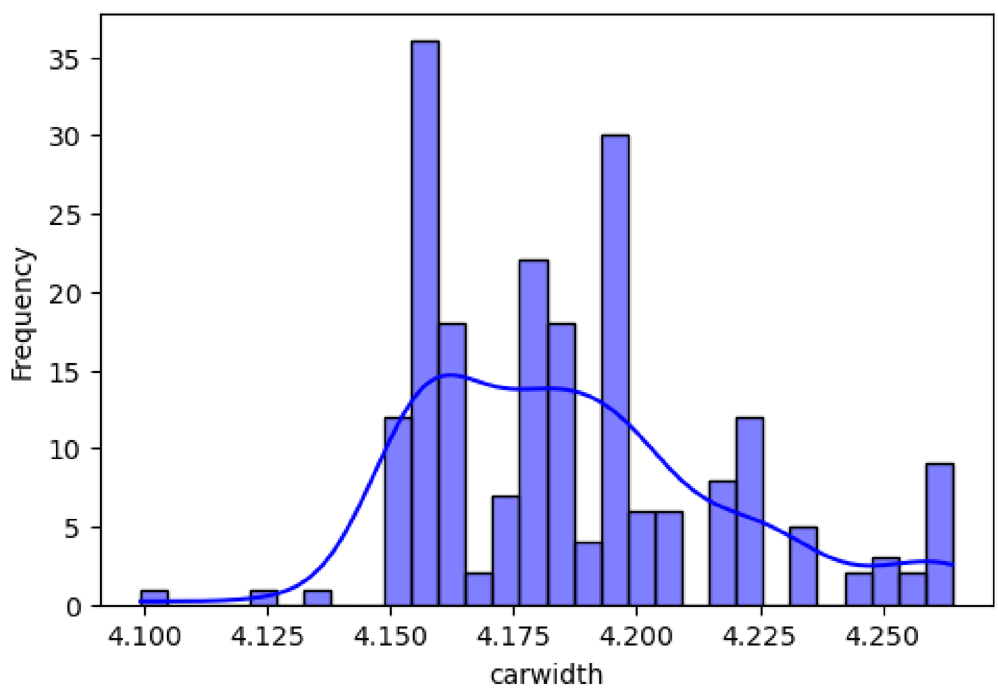


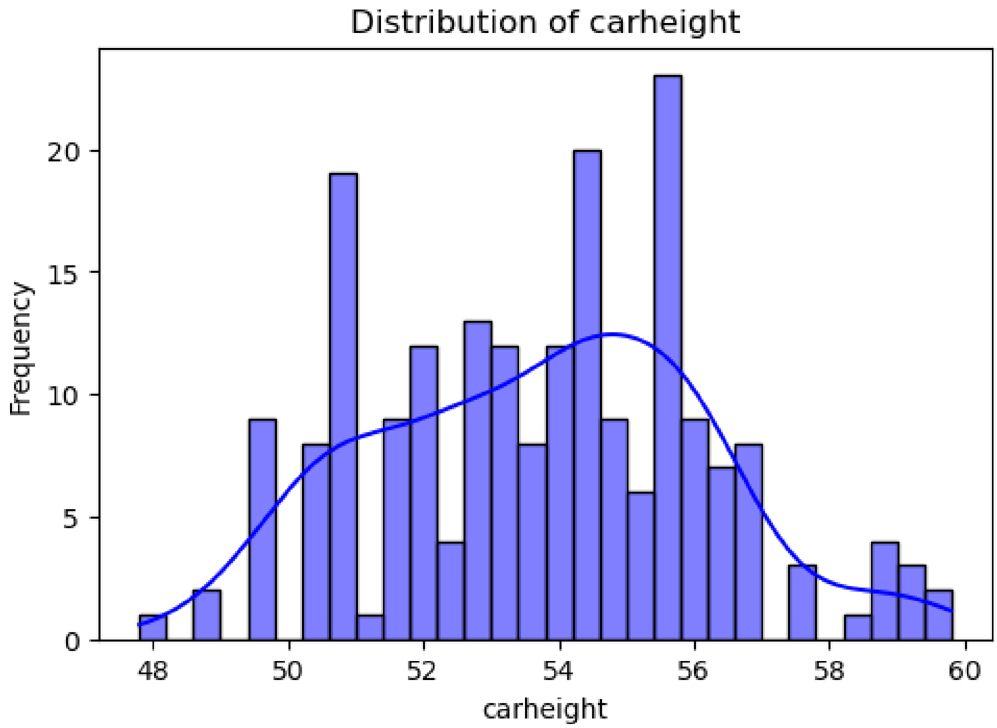
Distribution of wheelbase
Frequency
wheelbase
0
5
10
15
20
2.11
2.12
2.13
2.14
2.15
2.16
2.17
2.18

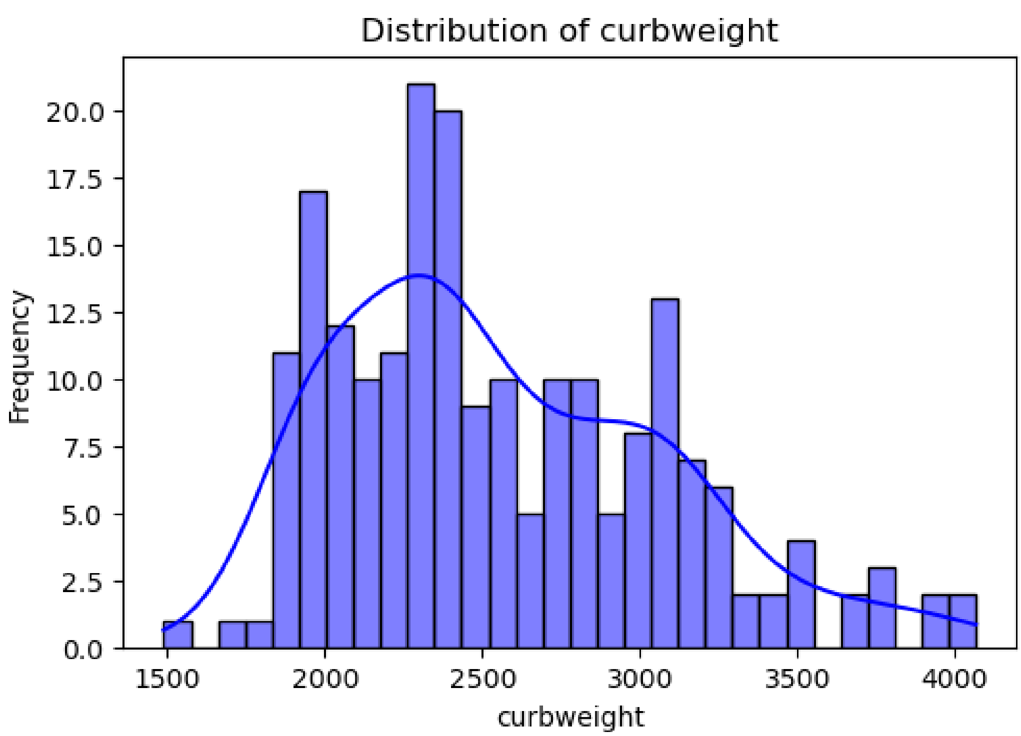
Distribution of carlength

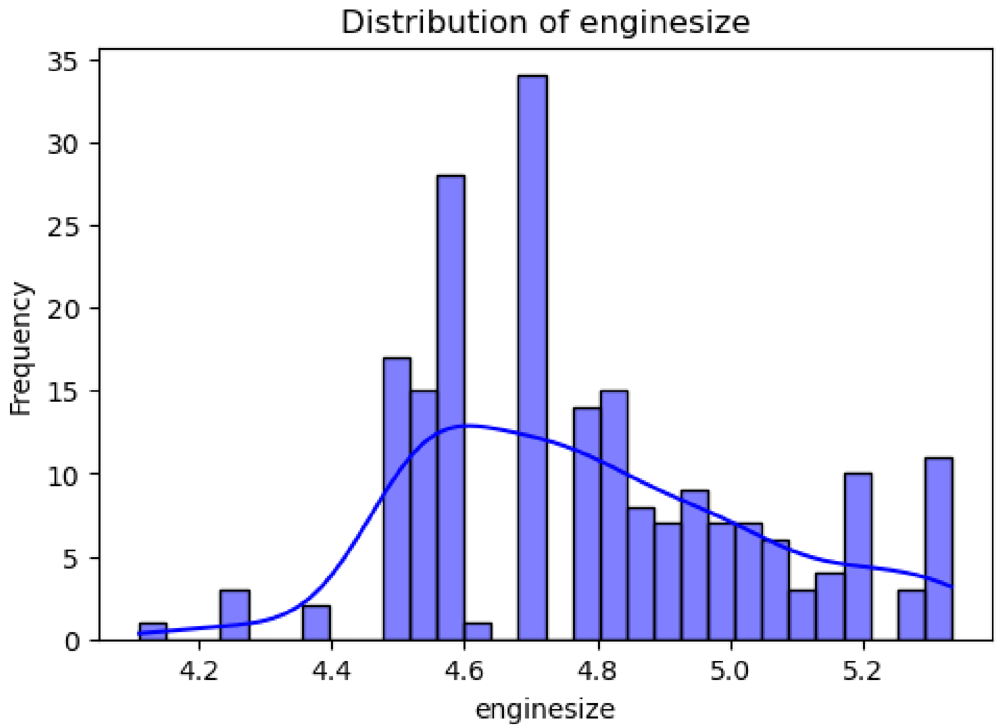


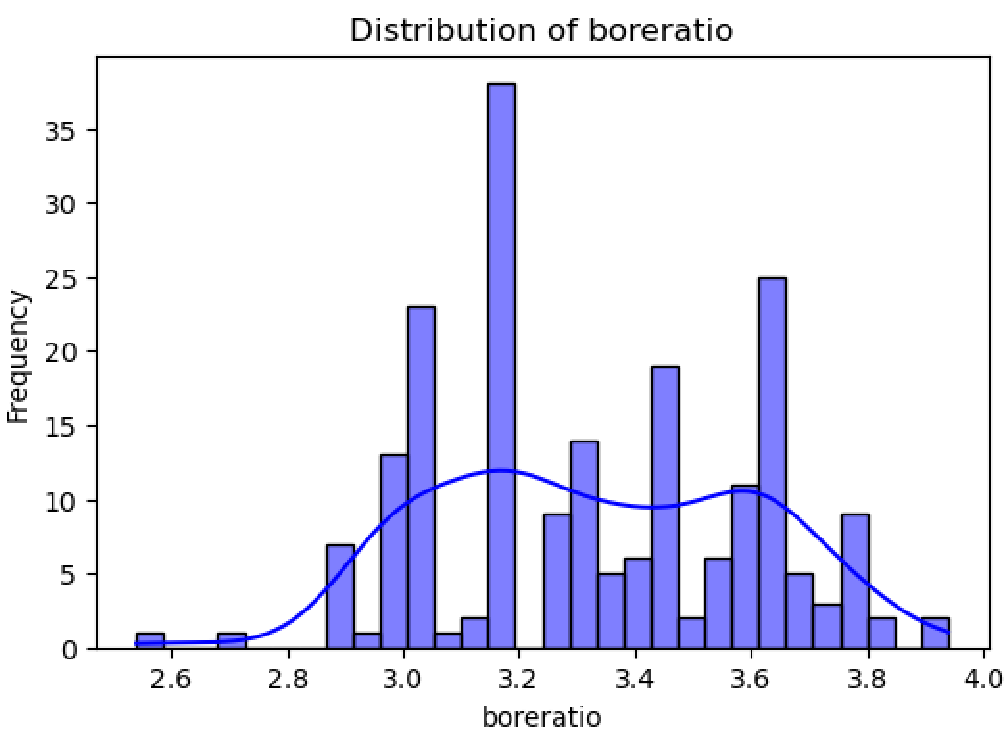
Distribution of carwidth

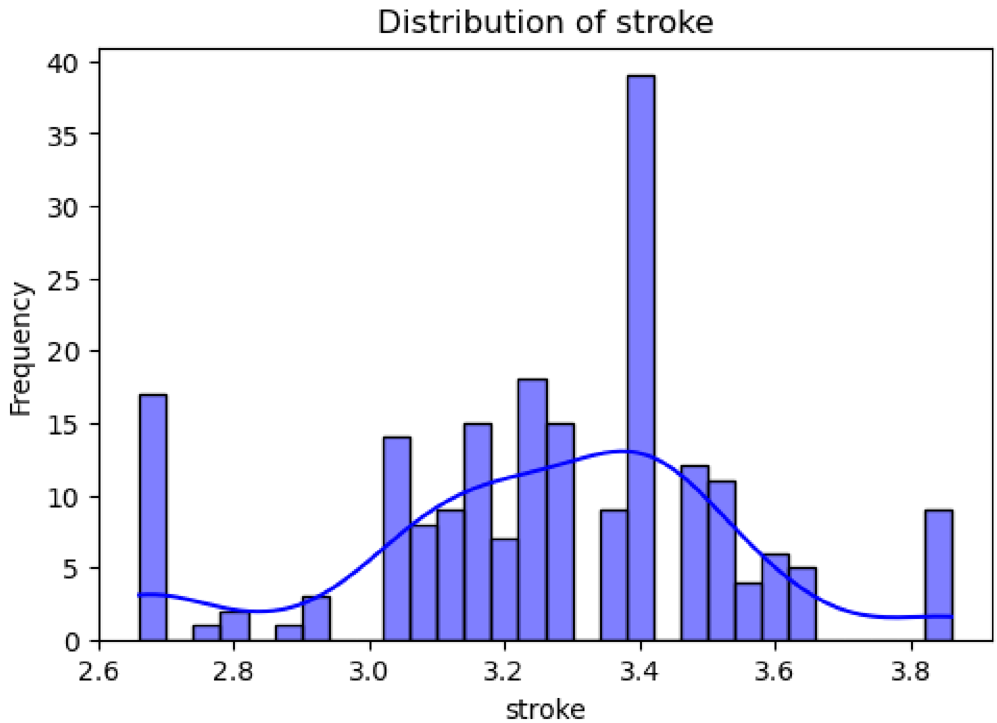



Distribution of carheight
Frequency
carheight
0
5
10
15
20
48
50
52
54
56
58
60

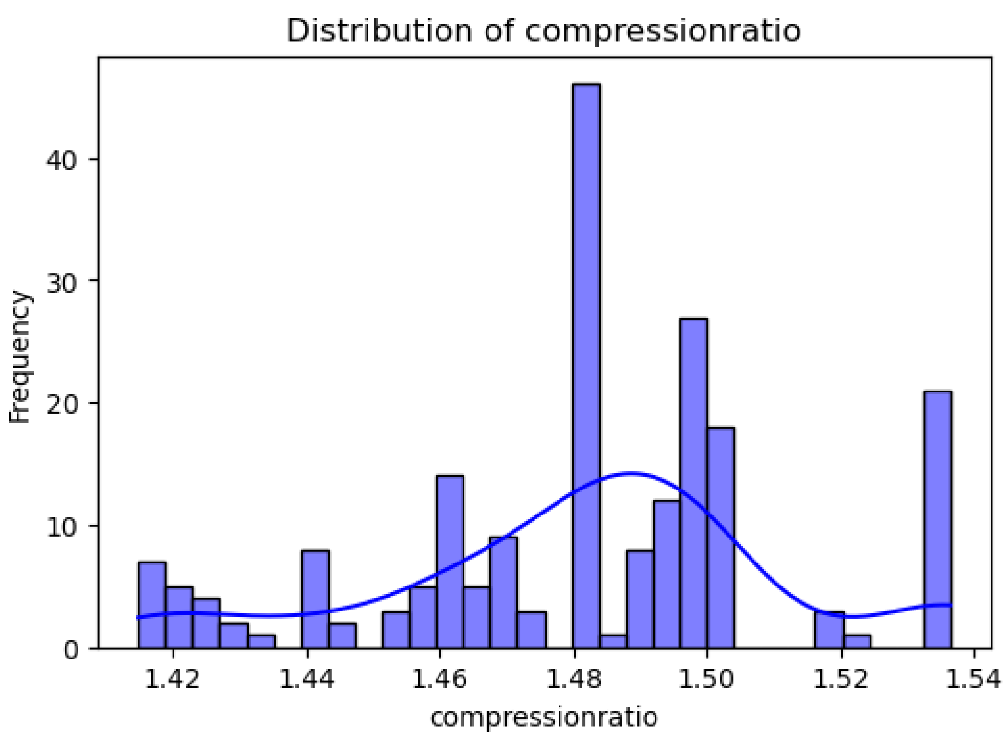

Distribution of curbweight
Frequency
curbweight
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1500
2000
2500
3000
3500
4000


Distribution of enginesize
Frequency
enginesize
0
5
10
15
20
25
30
35
4.2
4.4
4.6
4.8
5.0
5.2

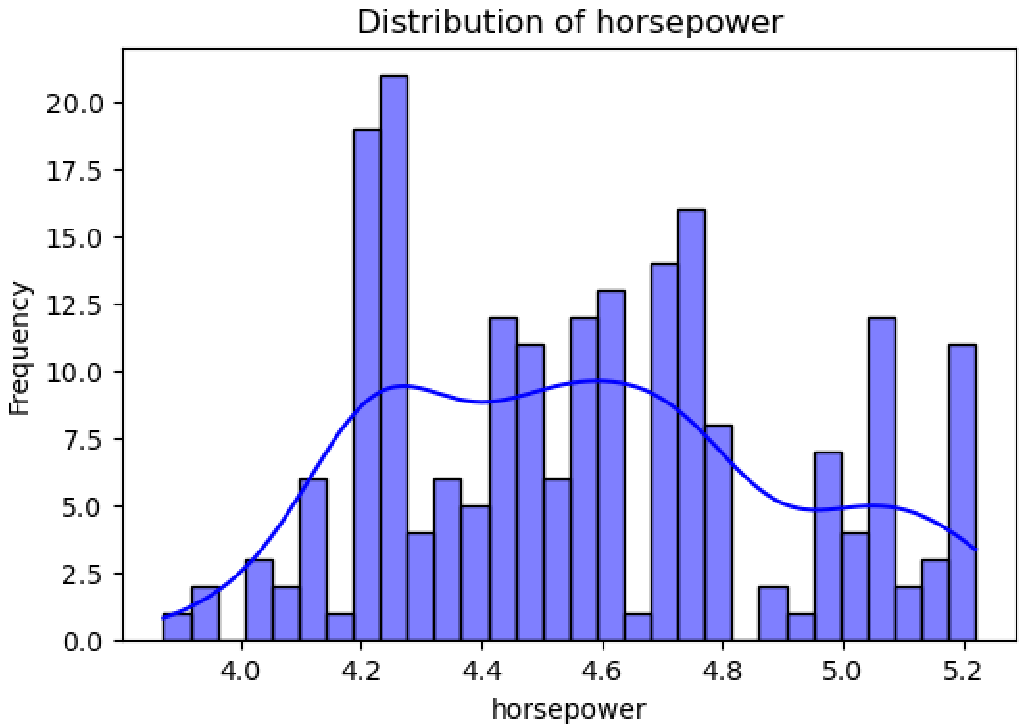

Distribution of boreratio
Frequency
boreratio
0
5
10
15
20
25
30
35
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0

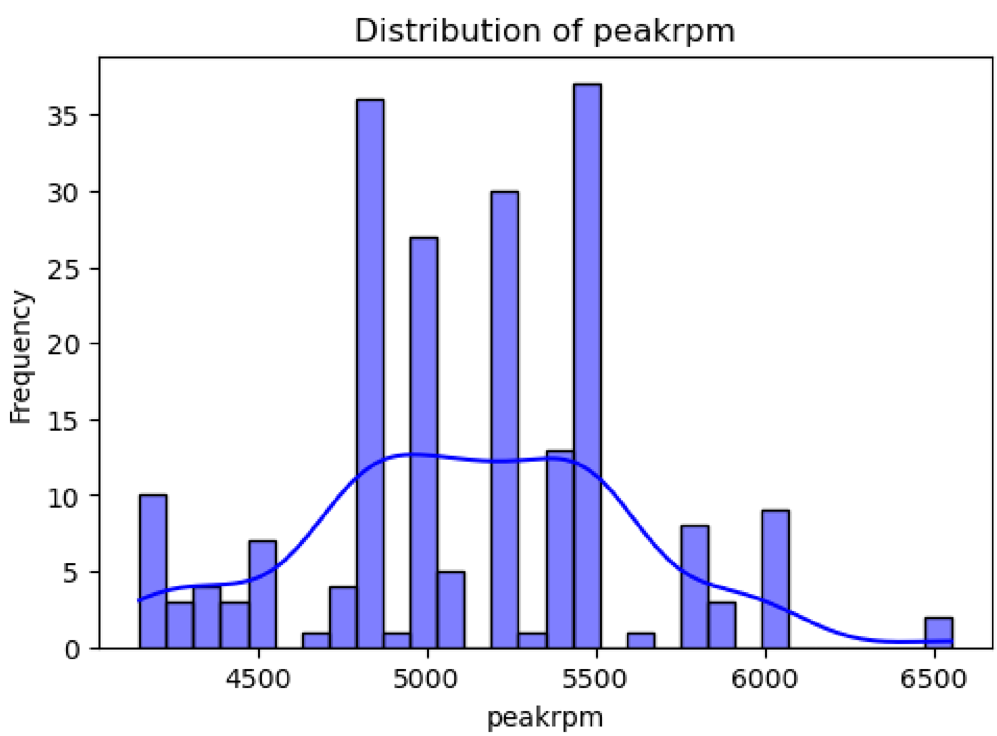


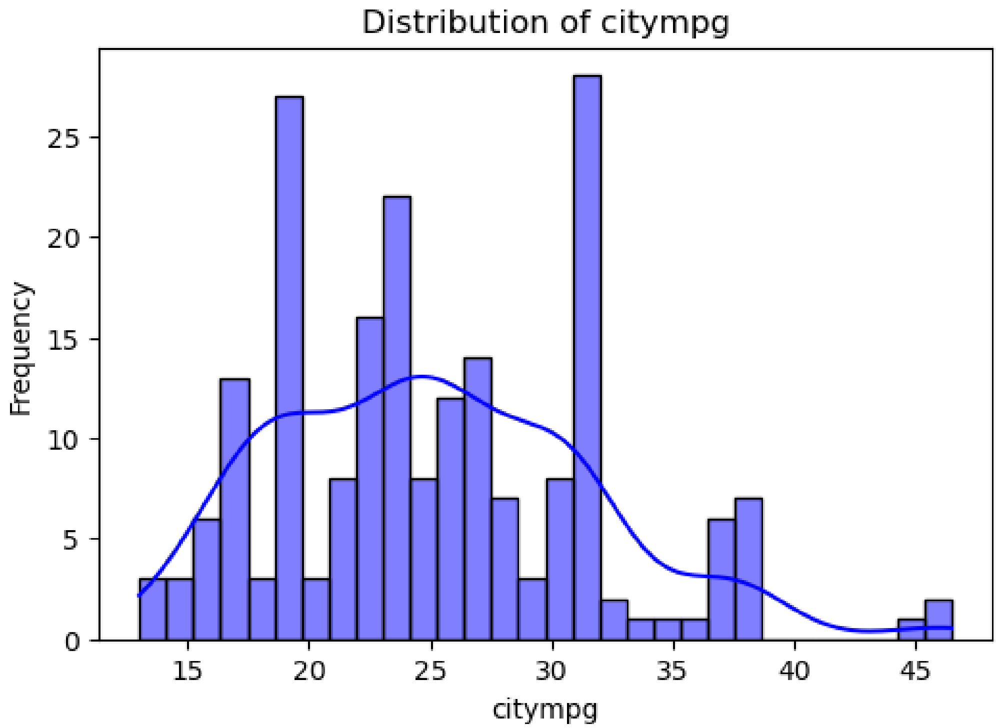
Distribution of stroke
Frequency
stroke
0
5
10
15
20
25
30
35
40
2.6
2.8
3.0
3.2
3.4
3.6
3.8



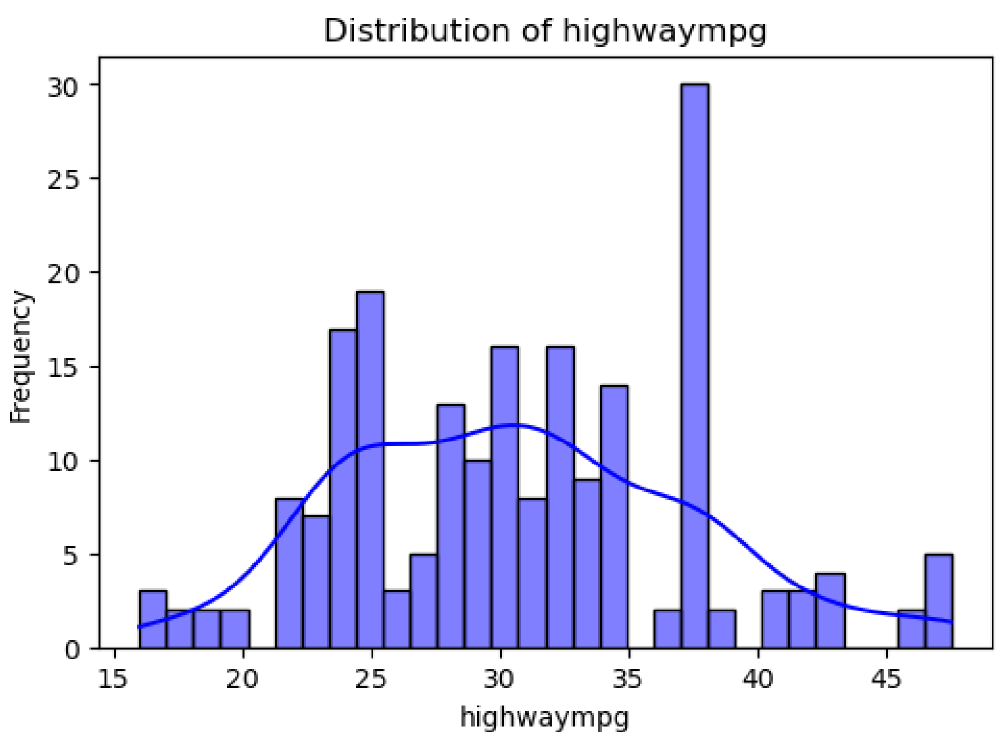
Distribution of compressionratio
Frequency
compressionratio
0
10
20
30
40
1.42
1.44
1.46
1.48
1.50
1.52
1.54


Distribution of horsepower
Frequency
horsepower
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
4.0
4.2
4.4
4.6
4.8
5.0
5.2

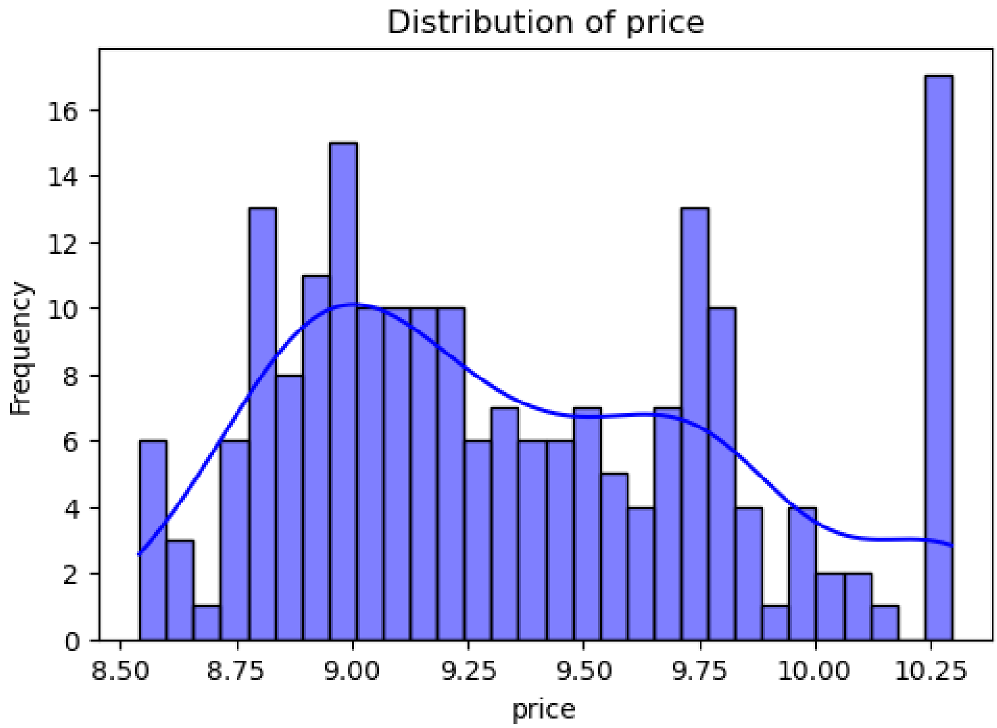

Distribution of peakrpm
Frequency
peakrpm
0
5
10
15
20
25
30
35
4500
5000
5500
6000
6500



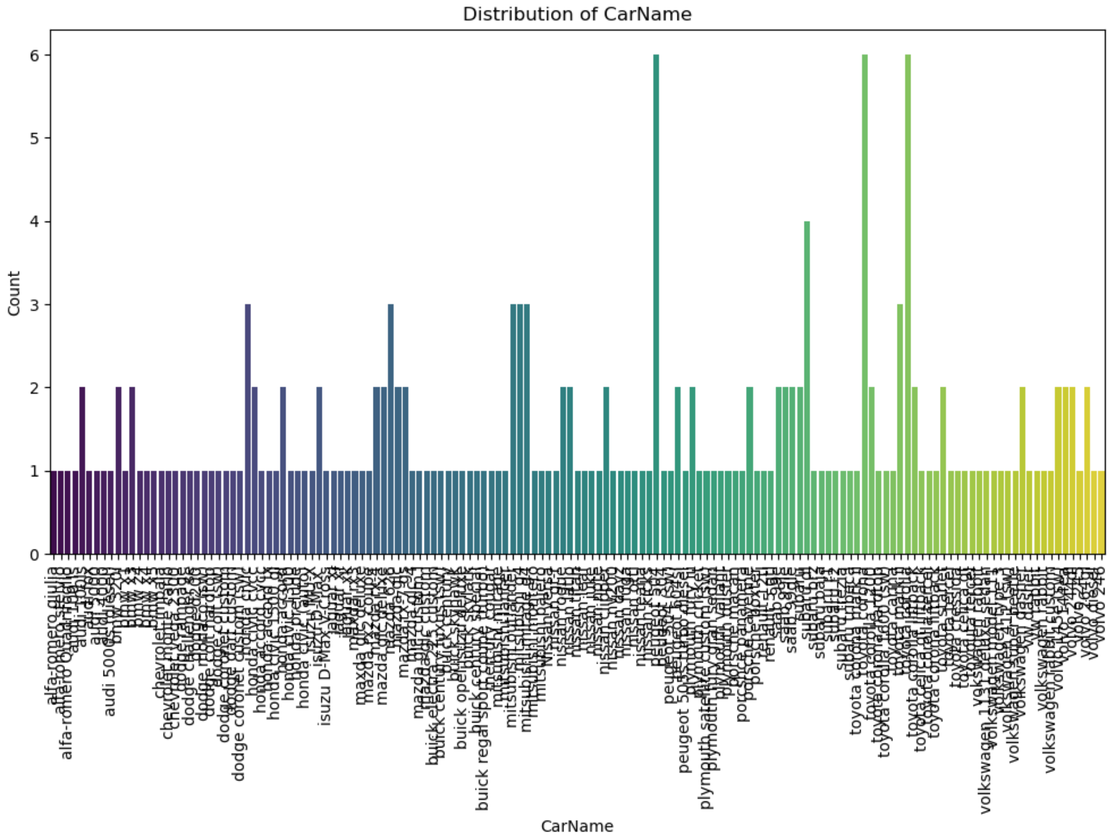
Distribution of citympg
Frequency
0
5
10
15
20
25
15
20
25
30
35
40
45
citympg

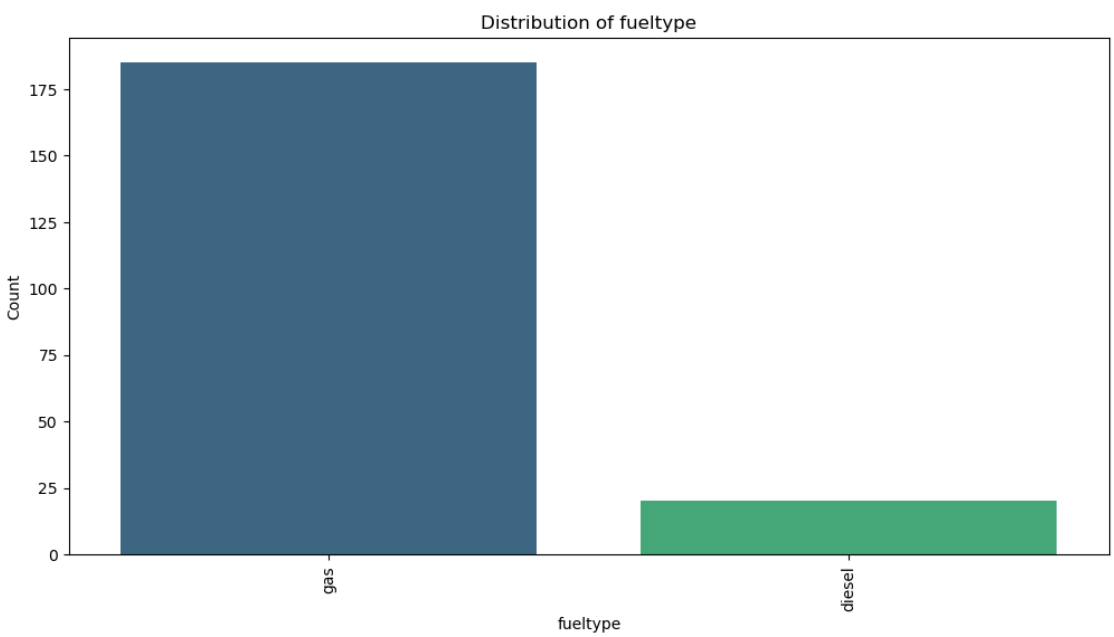


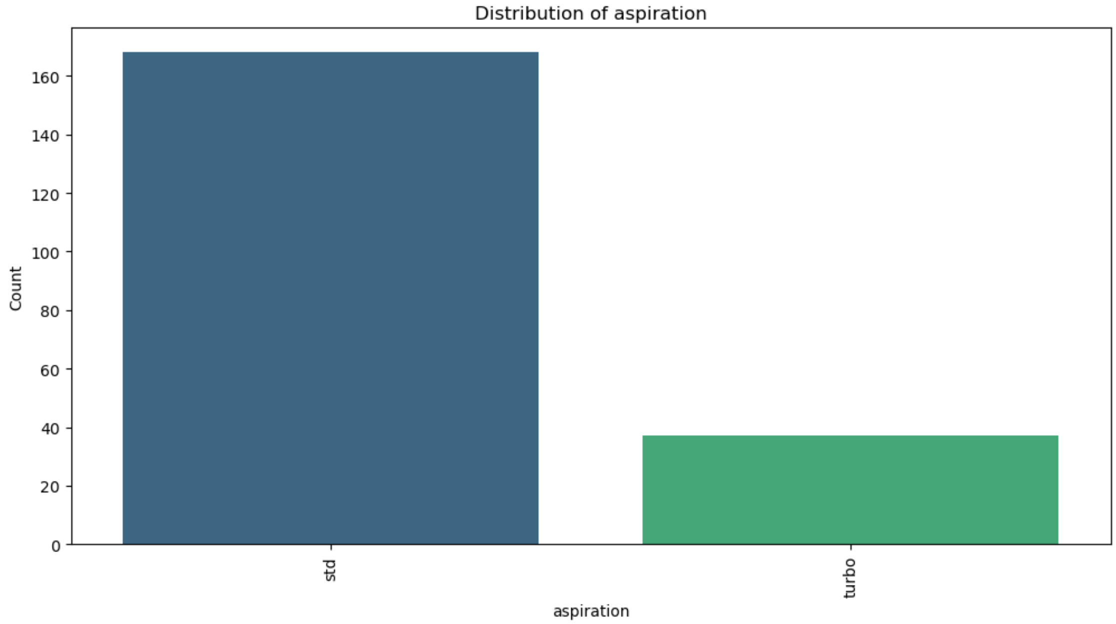
Distribution of highwaympg
Frequency
0
5
10
15
20
25
30
15
20
25
30
35
40
45
highwaympg

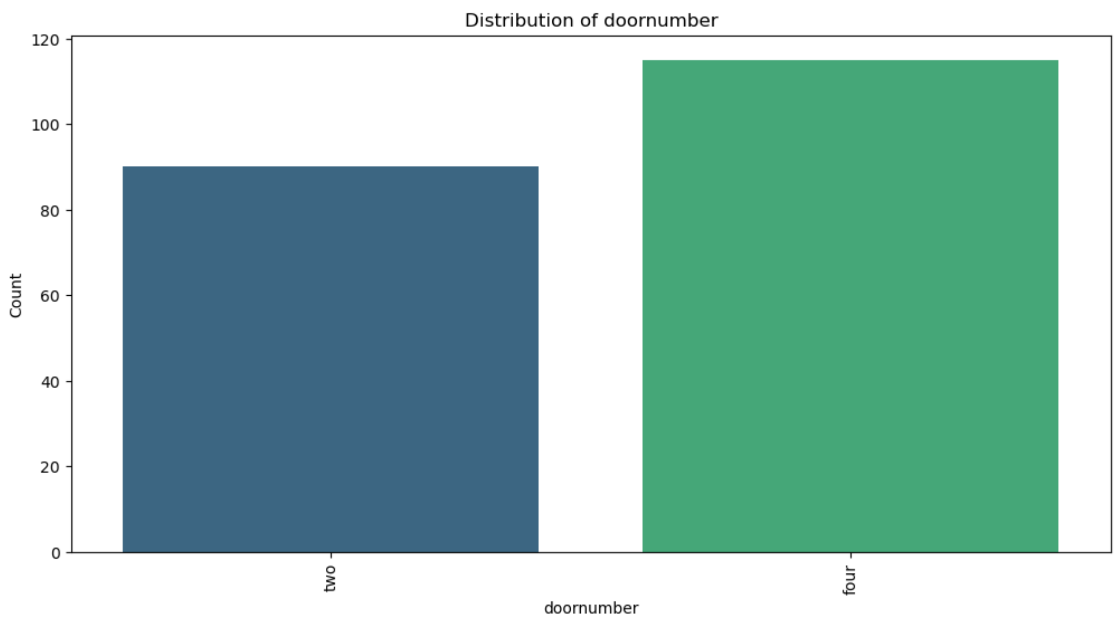


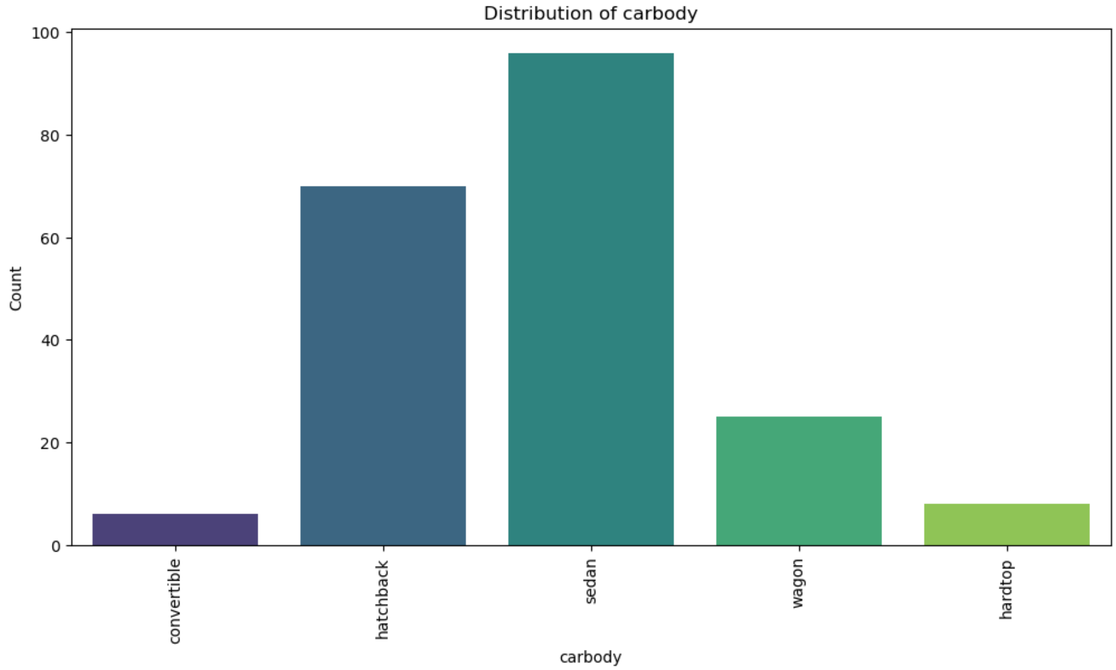
[158]:
```
categorical_feature = df1.drop(numerical_columns, axis=1)

# Plot bar chart for each categorical feature
for feature in categorical_feature:
    plt.figure(figsize=(12, 6))
    sns.countplot(x=feature, data=df, palette='viridis')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.xticks(rotation=90)
    plt.show()
```

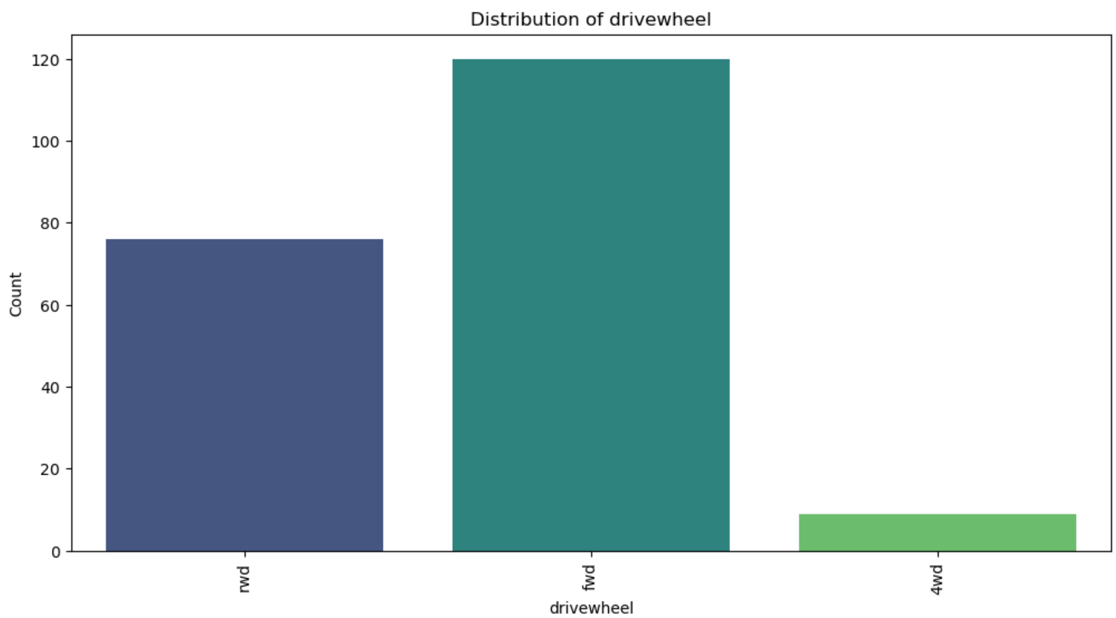

Distribution of CarName
Count
CarName

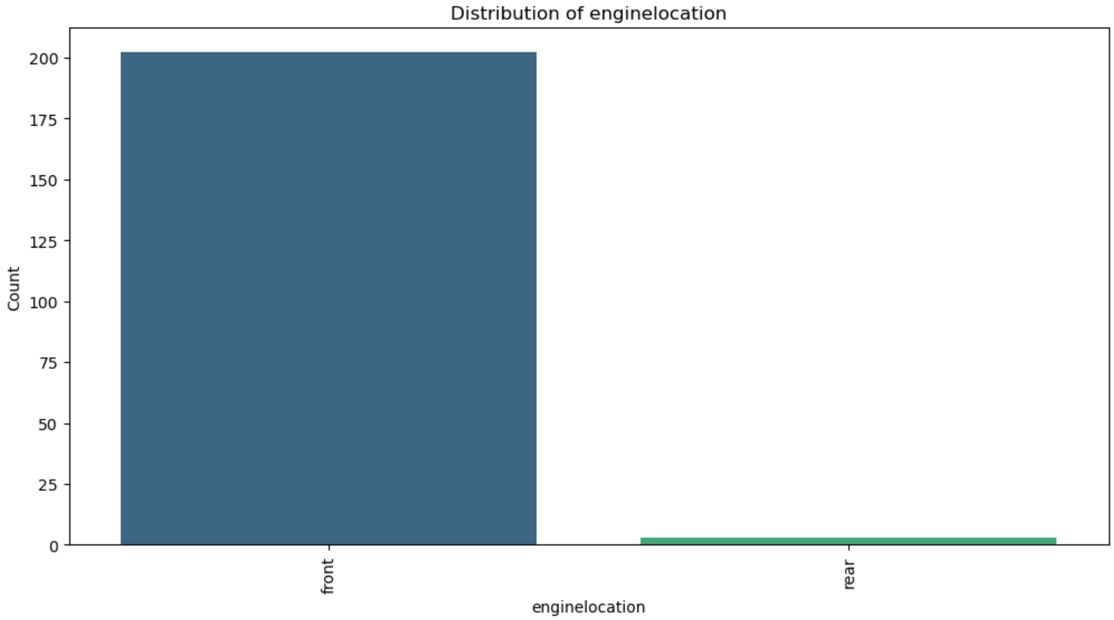

Distribution of fueltype
Count
gas
diesel
fueltype

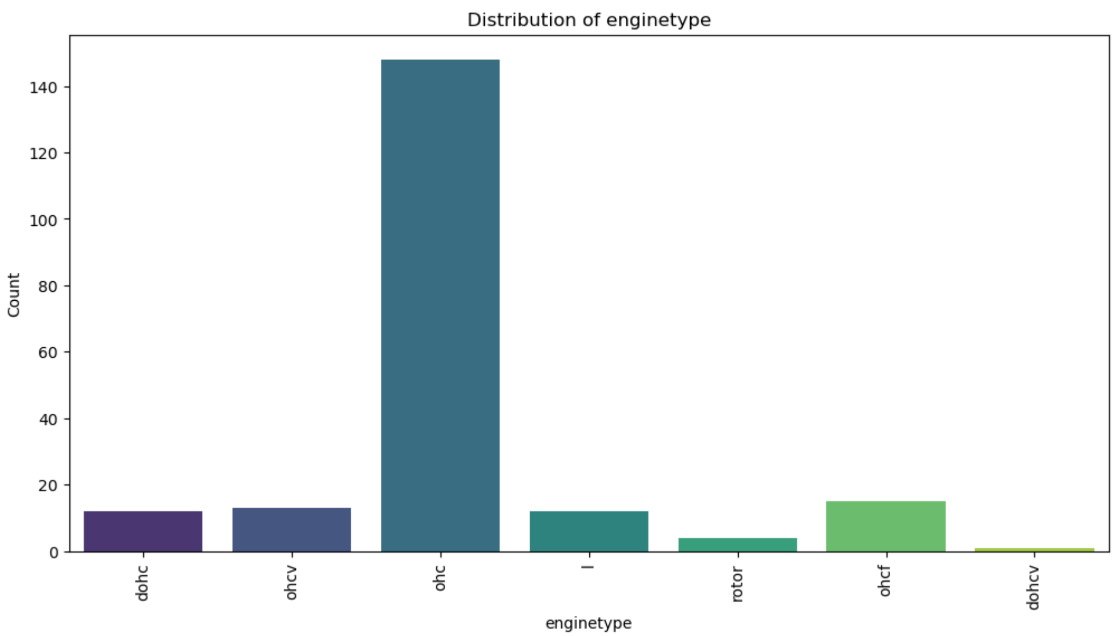

Distribution of aspiration
Count
0
20
40
60
80
100
120
140
160
std
turbo
aspiration

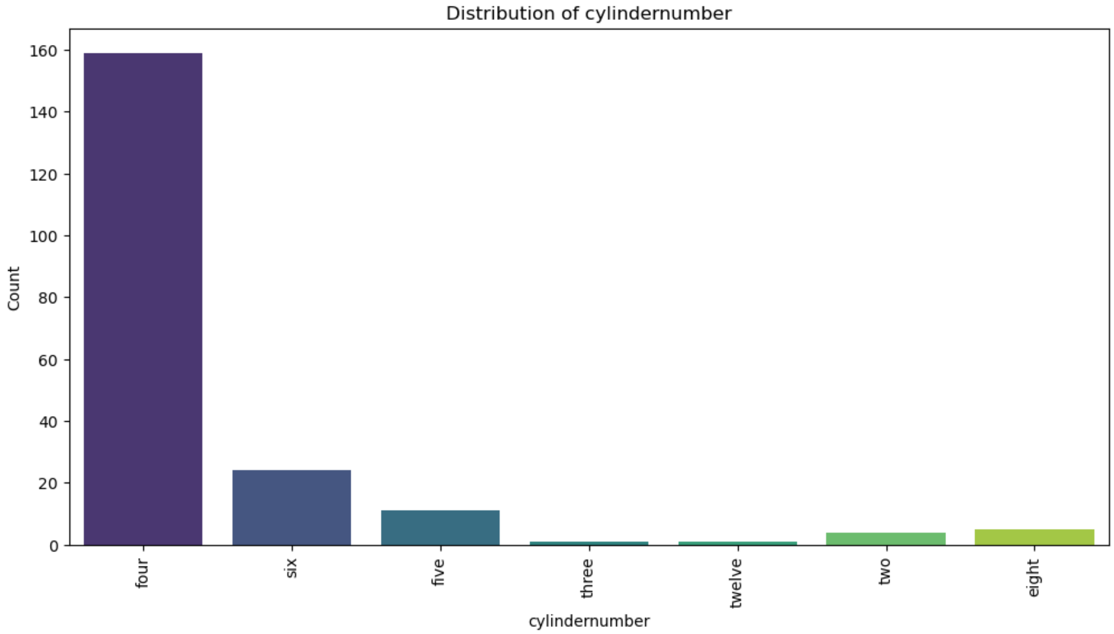

Distribution of doornumber
Count
0
20
40
60
80
100
120
two
four
doornumber

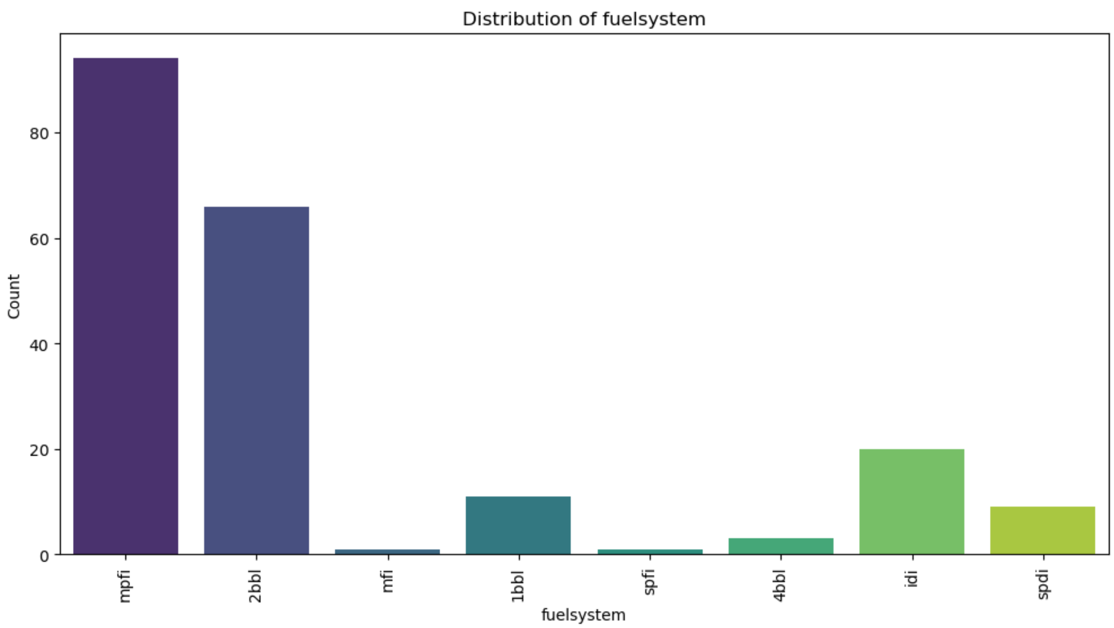

Distribution of carbody
Count
100
80
60
40
20
0
convertible
hatchback
sedan
wagon
hardtop
carbody


Distribution of drivewheel
Count
120
100
80
60
40
20
0
rwd
fwd
4wd
drivewheel


Distribution of enginelocation
Count
200
175
150
125
100
75
50
25
0
front
rear
enginelocation

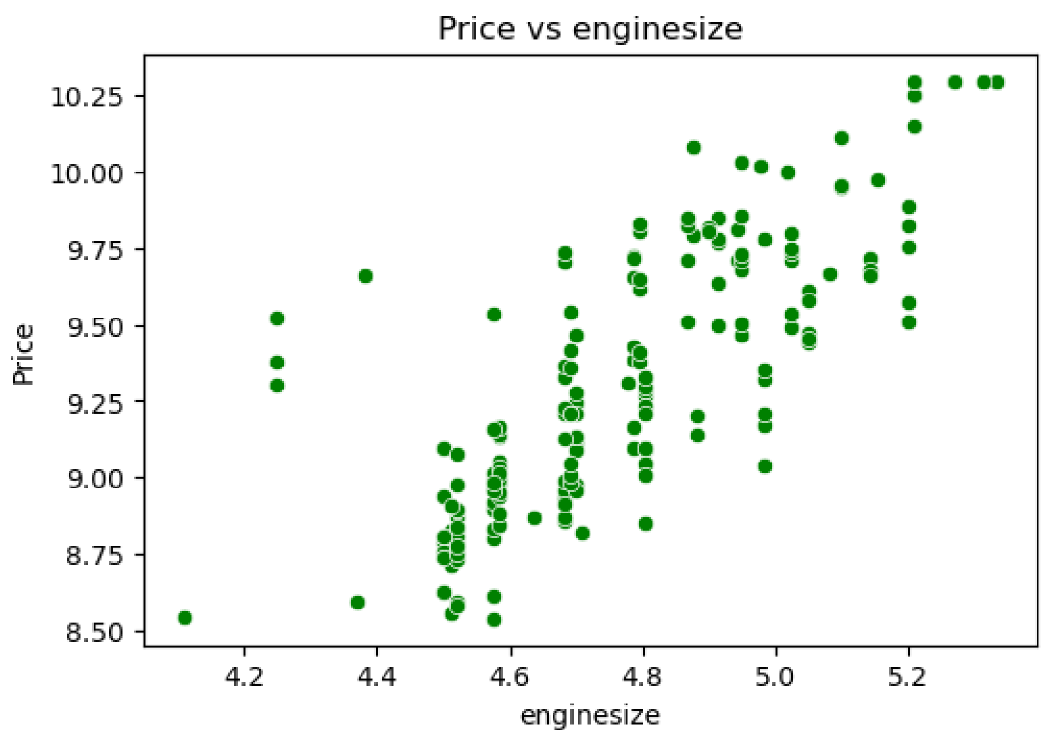

Distribution of enginetype
Count
140
120
100
80
60
40
20
0
dohc
ohcv
ohc
l
rotor
ohcf
dohcv
enginetype

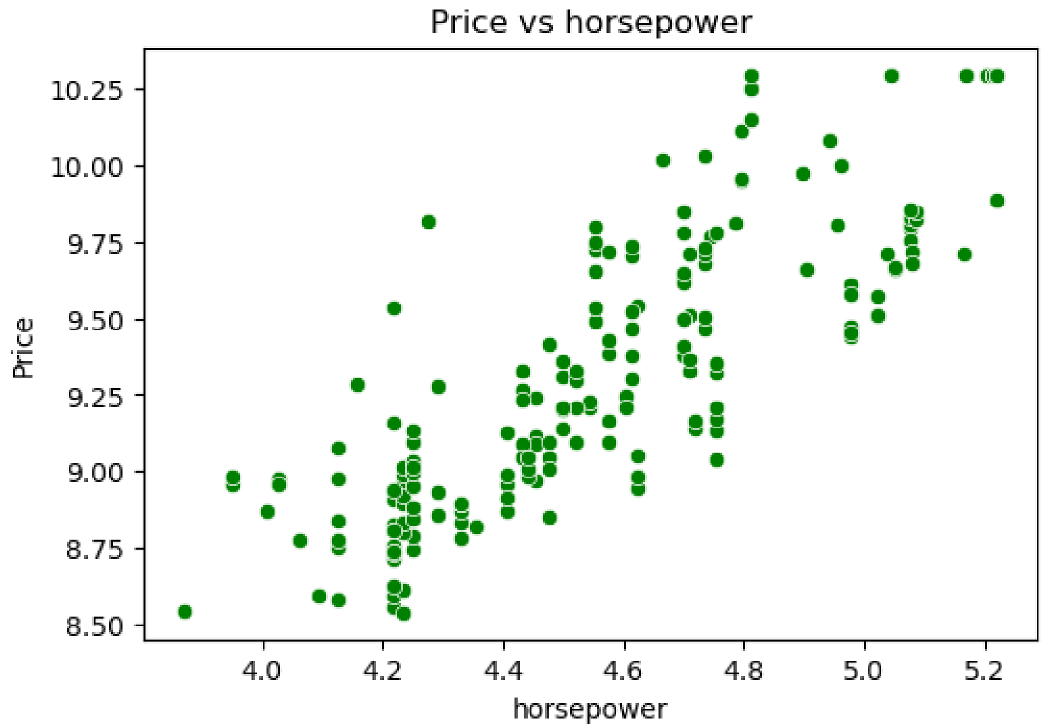


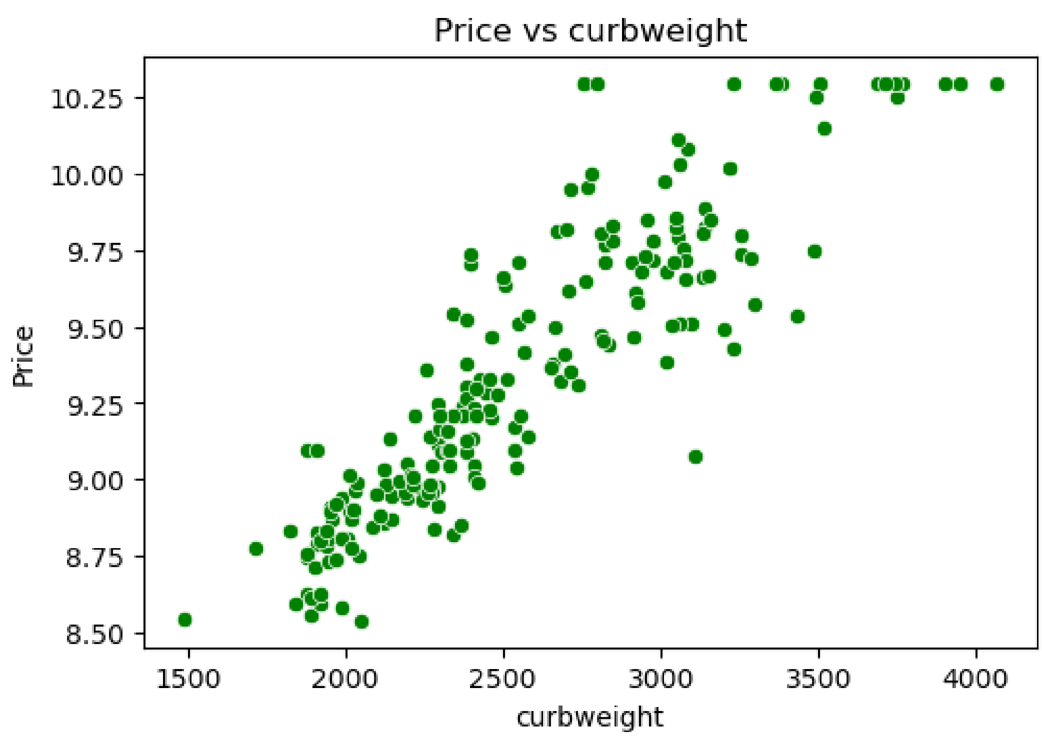


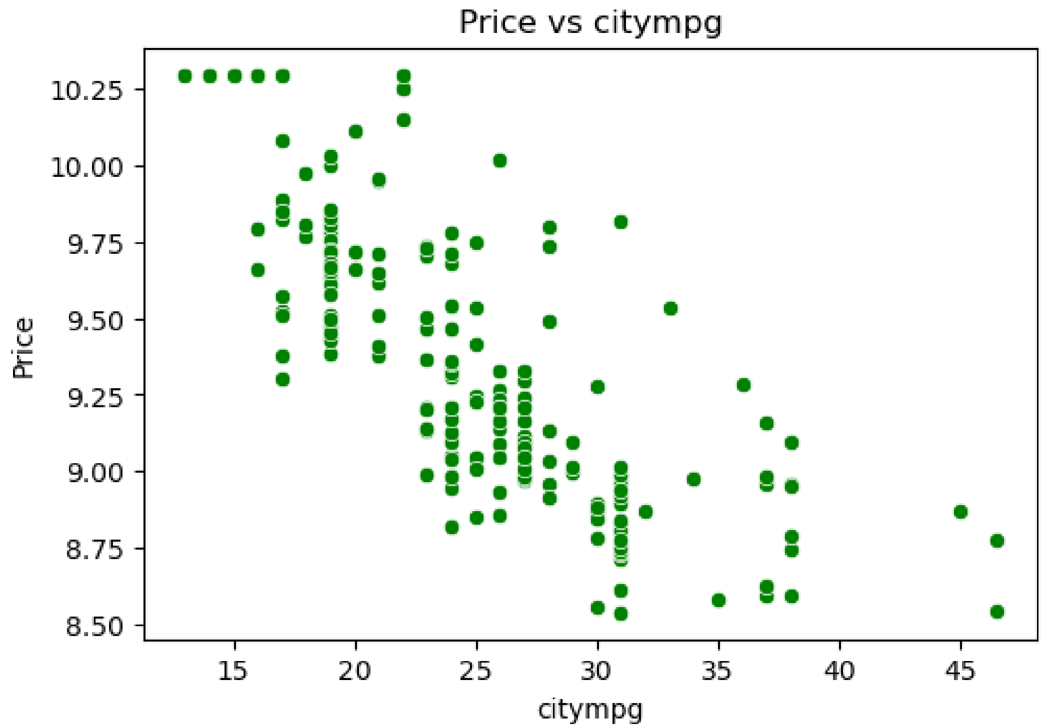
[174]:
```
# Scatter plots for price vs numerical features
features_to_compare = ['enginesize', 'horsepower', 'curbweight', 'citympg',␣
 ↪'highwaympg']

for feature in features_to_compare:
```

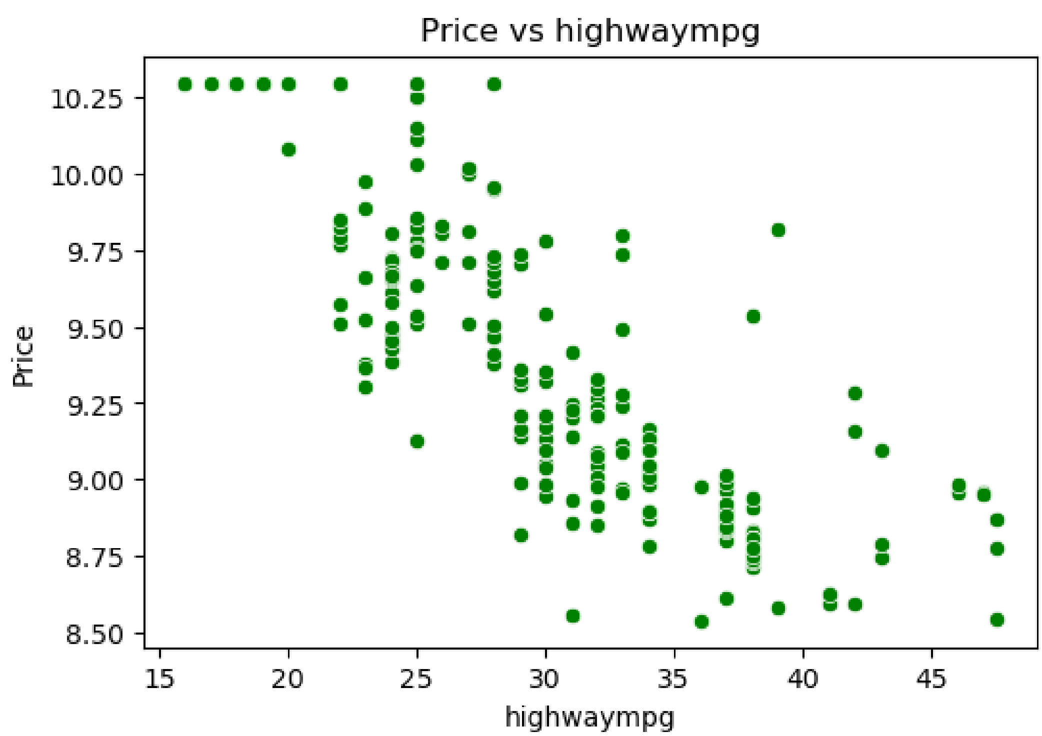
```
plt.figure(figsize=(6, 4))
sns.scatterplot(x=df[feature], y=df['price'], color='green')
plt.title(f'Price vs {feature}')
plt.xlabel(feature)
plt.ylabel('Price')
plt.show()
```




Price vs horsepower
Price
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
4.0
4.2
4.4
4.6
4.8
5.0
5.2
horsepower


Price vs curbweight
Price
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
1500
2000
2500
3000
3500
4000
curbweight


Price vs citympg
Price
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
15
20
25
30
35
40
45
citympg


Price vs highwaympg
Price
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
15
20
25
30
35
40
45
highwaympg

1.10 7. Feature Engineering

Label encoding and Onehot Encoding

[183]:
```
label_encoder = LabelEncoder()
df_label = df1.copy()
df1.head()
```

[183]:
```
   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   2.117577  …    4.867534
1  convertible        rwd          front   2.117577  …    4.867534
2    hatchback        rwd          front   2.132745  …    5.023881
3        sedan        fwd          front   2.145500  …    4.691348
4        sedan        4wd          front   2.144563  …    4.912655

   fuelsystem  boreratio  stroke compressionratio horsepower  peakrpm citympg  \
0        mpfi       3.47    2.68         1.482304   4.709530   5000.0    21.0
1        mpfi       3.47    2.68         1.482304   4.709530   5000.0    21.0
2        mpfi       2.68    3.47         1.482304   5.036953   5000.0    19.0
3        mpfi       3.19    3.40         1.517427   4.624973   5500.0    24.0
4        mpfi       3.19    3.40         1.442027   4.744932   5500.0    18.0

   highwaympg     price
0        27.0  9.510075
1        27.0  9.711116
2        26.0  9.711116
3        30.0  9.543235
4        22.0  9.767095

[5 rows x 26 columns]
```

[185]:
```
#label Encod
df_label['fueltype'] = label_encoder.fit_transform(df1['fueltype'])
df_label['aspiration'] = label_encoder.fit_transform(df1['aspiration'])
df_label['doornumber'] = label_encoder.fit_transform(df1['doornumber'])
df_label['carbody'] = label_encoder.fit_transform(df1['carbody'])
df_label['drivewheel'] = label_encoder.fit_transform(df1['drivewheel'])
df_label['enginelocation'] = label_encoder.fit_transform(df1['enginelocation'])
df_label['enginetype'] = label_encoder.fit_transform(df1['enginetype'])
df_label['cylindernumber'] = label_encoder.fit_transform(df1['cylindernumber'])
df_label['fuelsystem'] = label_encoder.fit_transform(df1['fuelsystem'])
```

```python
df_label.head()
```

[185]:

```
   car_ID  symboling                   CarName  fueltype  aspiration  \
0       1          3        alfa-romero giulia         1           0
1       2          3       alfa-romero stelvio         1           0
2       3          1  alfa-romero Quadrifoglio         1           0
3       4          2               audi 100 ls         1           0
4       5          2                audi 100ls         1           0

   doornumber  carbody  drivewheel  enginelocation  wheelbase  …  \
0           1        0           2               0   2.117577  …
1           1        0           2               0   2.117577  …
2           1        2           2               0   2.132745  …
3           0        3           1               0   2.145500  …
4           0        3           0               0   2.144563  …

   enginesize  fuelsystem  boreratio  stroke  compressionratio  horsepower  \
0    4.867534           5       3.47    2.68          1.482304    4.709530
1    4.867534           5       3.47    2.68          1.482304    4.709530
2    5.023881           5       2.68    3.47          1.482304    5.036953
3    4.691348           5       3.19    3.40          1.517427    4.624973
4    4.912655           5       3.19    3.40          1.442027    4.744932

   peakrpm  citympg  highwaympg     price
0   5000.0     21.0        27.0  9.510075
1   5000.0     21.0        27.0  9.711116
2   5000.0     19.0        26.0  9.711116
3   5500.0     24.0        30.0  9.543235
4   5500.0     18.0        22.0  9.767095

[5 rows x 26 columns]
```

[187]:

```python
# OneHot encoding

onehot = OneHotEncoder(sparse_output=False)
hot_encod = onehot.fit_transform(df_label[['CarName']])
hot_columns = onehot.get_feature_names_out(['CarName'])
```

[189]:

```python
# Creating DataFrame with onehot encoded columns
df_onehot = pd.concat([
    df_label,
    pd.DataFrame(hot_encod, columns=hot_columns)
], axis=1)
```

[191]:

```python
df_onehot.head()
```

[191]:
```
   car_ID  symboling                   CarName  fueltype  aspiration  \
0       1          3        alfa-romero giulia         1           0
1       2          3       alfa-romero stelvio         1           0
2       3          1  alfa-romero Quadrifoglio         1           0
3       4          2               audi 100 ls         1           0
4       5          2                audi 100ls         1           0

   doornumber  carbody  drivewheel  enginelocation  wheelbase  …  \
0           1        0           2               0   2.117577  …
1           1        0           2               0   2.117577  …
2           1        2           2               0   2.132745  …
3           0        3           1               0   2.145500  …
4           0        3           0               0   2.144563  …

   CarName_volkswagen type 3  CarName_volvo 144ea  CarName_volvo 145e (sw)  \
0                        0.0                  0.0                      0.0
1                        0.0                  0.0                      0.0
2                        0.0                  0.0                      0.0
3                        0.0                  0.0                      0.0
4                        0.0                  0.0                      0.0

   CarName_volvo 244dl  CarName_volvo 245  CarName_volvo 246  \
0                  0.0                0.0                0.0
1                  0.0                0.0                0.0
2                  0.0                0.0                0.0
3                  0.0                0.0                0.0
4                  0.0                0.0                0.0

   CarName_volvo 264gl  CarName_volvo diesel  CarName_vw dasher  \
0                  0.0                   0.0                0.0
1                  0.0                   0.0                0.0
2                  0.0                   0.0                0.0
3                  0.0                   0.0                0.0
4                  0.0                   0.0                0.0

   CarName_vw rabbit
0                0.0
1                0.0
2                0.0
3                0.0
4                0.0

[5 rows x 173 columns]
```

[195]:
```
df_onehot.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Columns: 173 entries, car_ID to CarName_vw rabbit
dtypes: float64(160), int32(9), int64(3), object(1)
memory usage: 270.0+ KB
```

[197]:
```
# Dropping categorical values
df_onehot = df_onehot.drop('CarName',axis=1)
```

[199]:
```
df_onehot.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Columns: 172 entries, car_ID to CarName_vw rabbit
dtypes: float64(160), int32(9), int64(3)
memory usage: 268.4 KB
```

1.10.1 Setting y as target

[204]:
```
y = df_onehot['price']
y
x = df_onehot.drop(['price','car_ID'], axis=1)
x
```

[204]:
```
     symboling  fueltype  aspiration  doornumber  carbody  drivewheel  \
0            3         1           0           1        0           2
1            3         1           0           1        0           2
2            1         1           0           1        2           2
3            2         1           0           0        3           1
4            2         1           0           0        3           0
..         …         …           …           …        …           …
200         -1         1           0           0        3           2
201         -1         1           1           0        3           2
202         -1         1           0           0        3           2
203         -1         0           1           0        3           2
204         -1         1           1           0        3           2

     enginelocation  wheelbase  carlength  carwidth  …  \
0                 0   2.117577      168.8  4.160444  …
1                 0   2.117577      168.8  4.160444  …
2                 0   2.132745      171.2  4.182050  …
3                 0   2.145500      176.6  4.192680  …
4                 0   2.144563      176.6  4.195697  …
..              …        …          …        …  …
200               0   2.166164      188.8  4.232656  …
201               0   2.166164      188.8  4.231204  …
202               0   2.166164      188.8  4.232656  …
```

```
203              0   2.166164      188.8  4.232656  …
204              0   2.166164      188.8  4.232656  …

     CarName_volkswagen type 3  CarName_volvo 144ea  CarName_volvo 145e (sw)  \
0                          0.0                  0.0                      0.0
1                          0.0                  0.0                      0.0
2                          0.0                  0.0                      0.0
3                          0.0                  0.0                      0.0
4                          0.0                  0.0                      0.0
..                         …                  …                      …
200                        0.0                  0.0                      1.0
201                        0.0                  1.0                      0.0
202                        0.0                  0.0                      0.0
203                        0.0                  0.0                      0.0
204                        0.0                  0.0                      0.0

     CarName_volvo 244dl  CarName_volvo 245  CarName_volvo 246  \
0                    0.0                0.0                0.0
1                    0.0                0.0                0.0
2                    0.0                0.0                0.0
3                    0.0                0.0                0.0
4                    0.0                0.0                0.0
..                   …                …                …
200                  0.0                0.0                0.0
201                  0.0                0.0                0.0
202                  1.0                0.0                0.0
203                  0.0                0.0                1.0
204                  0.0                0.0                0.0

     CarName_volvo 264gl  CarName_volvo diesel  CarName_vw dasher  \
0                    0.0                   0.0                0.0
1                    0.0                   0.0                0.0
2                    0.0                   0.0                0.0
3                    0.0                   0.0                0.0
4                    0.0                   0.0                0.0
..                   …                   …                …
200                  0.0                   0.0                0.0
201                  0.0                   0.0                0.0
202                  0.0                   0.0                0.0
203                  0.0                   0.0                0.0
204                  1.0                   0.0                0.0

     CarName_vw rabbit
0                  0.0
1                  0.0
2                  0.0
3                  0.0
```

```
4                    0.0
..                   …
200                  0.0
201                  0.0
202                  0.0
203                  0.0
204                  0.0

[205 rows x 170 columns]
```

1.11 8. Feature Selection

1.11.1 Variance threshold

[208]:
```
var_threshold = VarianceThreshold(threshold=0.5)
x_var = var_threshold.fit_transform(x)
var_selected = x.columns[var_threshold.get_support()].tolist()
print("1.Filter method results: ")
print("\na) Variance threshold")
print(f"Features selected: {len(var_selected)}")
print("Selected Features: ", var_selected[:12],"....")
```

```
1.Filter method results:

a) Variance threshold
Features selected: 11
Selected Features:  ['symboling', 'carbody', 'carlength', 'carheight',
'curbweight', 'enginetype', 'cylindernumber', 'fuelsystem', 'peakrpm',
'citympg', 'highwaympg'] …
```

[212]:
```
var_selected
```

[212]:
```
['symboling',
 'carbody',
 'carlength',
 'carheight',
 'curbweight',
 'enginetype',
 'cylindernumber',
 'fuelsystem',
 'peakrpm',
 'citympg',
 'highwaympg']
```

1.12 9. Feature Scaling

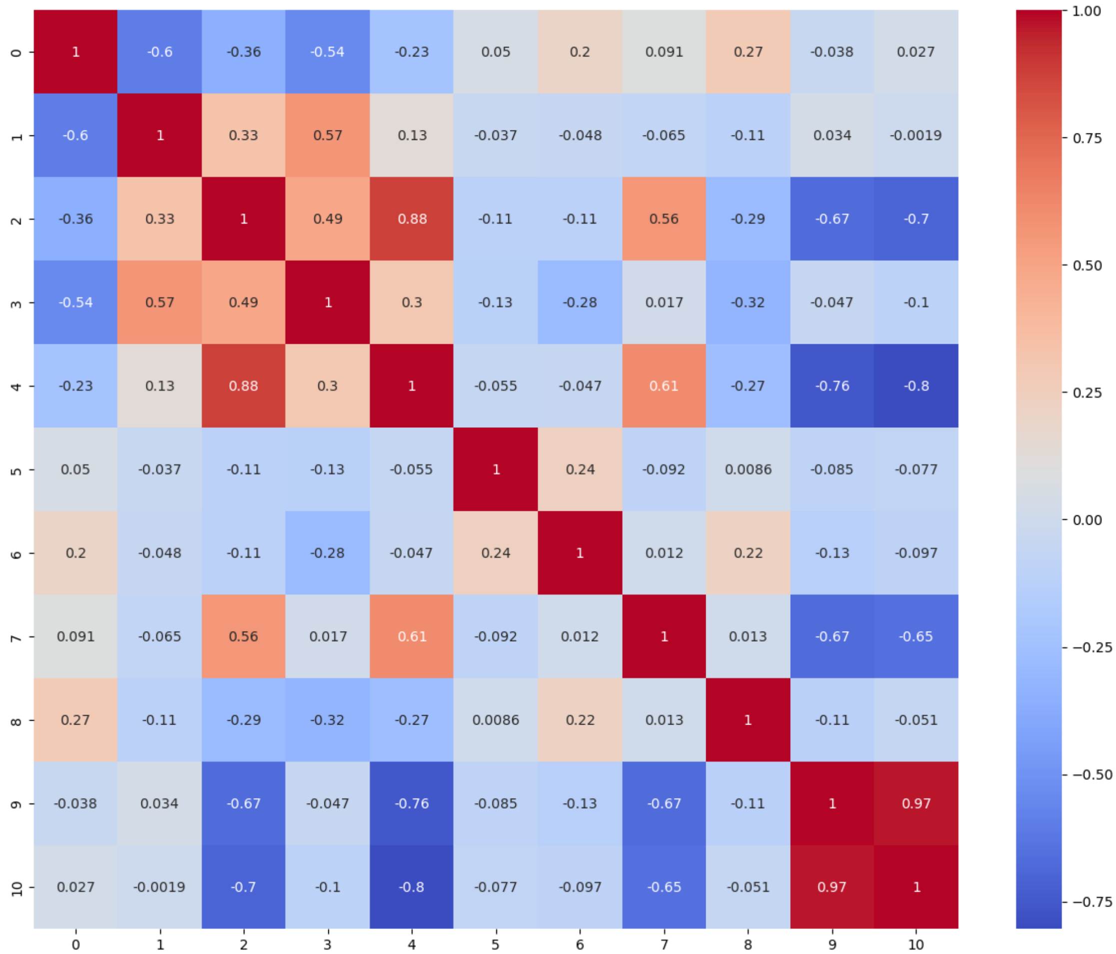
[220]:
```
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()
```

[222]:
```
# Applying  scaling method
x_standardized = standard_scaler.fit_transform(x_var)
```

[226]:
```
# Converting to DataFrame
df_std = pd.DataFrame(x_standardized)
df_std.head()
```

[226]:
```
          0         1         2         3         4         5         6  \
0  1.743470 -3.050975 -0.426521 -2.020417 -0.014566 -2.865105 -0.147475
1  1.743470 -3.050975 -0.426521 -2.020417 -0.014566 -2.865105 -0.147475
2  0.133509 -0.717207 -0.231513 -0.543527  0.514882  1.886890  1.112210
3  0.938490  0.449677  0.207256  0.235942 -0.420797 -0.013908 -0.147475
4  0.938490  0.449677  0.207256  0.235942  0.516807 -0.013908 -1.407161

          7         8         9        10
0  0.869568 -0.262757 -0.649321 -0.552143
1  0.869568 -0.262757 -0.649321 -0.552143
2  0.869568 -0.262757 -0.958163 -0.702161
3  0.869568  0.791357 -0.186058 -0.102086
4  0.869568  0.791357 -1.112584 -1.302237
```

[230]:
```
# Drawing Correlation
correlation = df_std.corr()
plt.figure(figsize=(15, 12))
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.show()
```



1.13 10. Splitting Data to training and testing

[235]:
```
X_train, X_test, y_train, y_test = train_test_split(df_std, y, test_size=0.2,␣
↪random_state=42)
```

1.13.1 Linear Regression

Linear Regression is a simple and interpretable model that establishes a linear relationship between the dependent variable (target) and one or more independent variables (features). It minimizes the sum of squared residuals to find the best-fitting line. While effective for datasets with linear trends, it may underperform when relationships are complex or non-linear.

1.13.2 Decision Tree Regressor

The Decision Tree Regressor is a tree-based model that splits the dataset into subsets based on feature values, creating a tree structure. Each leaf represents a prediction, while branches signify decisions. It's intuitive and can capture non-linear relationships but may overfit the data, requiring pruning or other regularization techniques.

1.13.3 Random Forest Regressor

Random Forest Regressor is an ensemble learning method that combines multiple decision trees (trained on random subsets of the data and features) to improve prediction accuracy. It reduces overfitting and increases robustness but may be computationally expensive for large datasets.

1.13.4 Gradient Boosting Regressor

Gradient Boosting Regressor is another ensemble technique that builds sequential decision trees, where each tree corrects the errors of the previous one. It is highly effective for complex, non-linear relationships and offers flexibility through hyperparameter tuning. However, it is computationally intensive and sensitive to overfitting if not properly tuned.

1.13.5 Support Vector Regressor (SVR)

SVR uses the principles of Support Vector Machines to find a hyperplane that best fits the data within a margin of tolerance. It is powerful for small- to medium-sized datasets with non-linear relationships, especially when paired with kernels. However, it may struggle with large datasets and requires careful parameter selection for optimal performance.

1.14 11. Building Models

[238]:

```
# Define models
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree Regressor": DecisionTreeRegressor(random_state=42),
    "Random Forest Regressor": RandomForestRegressor(random_state=42),
    "Gradient Boosting Regressor": GradientBoostingRegressor(random_state=42),
    "Support Vector Regressor": SVR(kernel='rbf')
}
```

[240]:

```
# Train and evaluate models
results = {}

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Calculate metrics
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    # Store results
    results[name] = {
        "R² Score": r2,
```

```
        "MAE": mae,
        "RMSE": rmse
    }
```

1.15 12. Model Evaluation Result

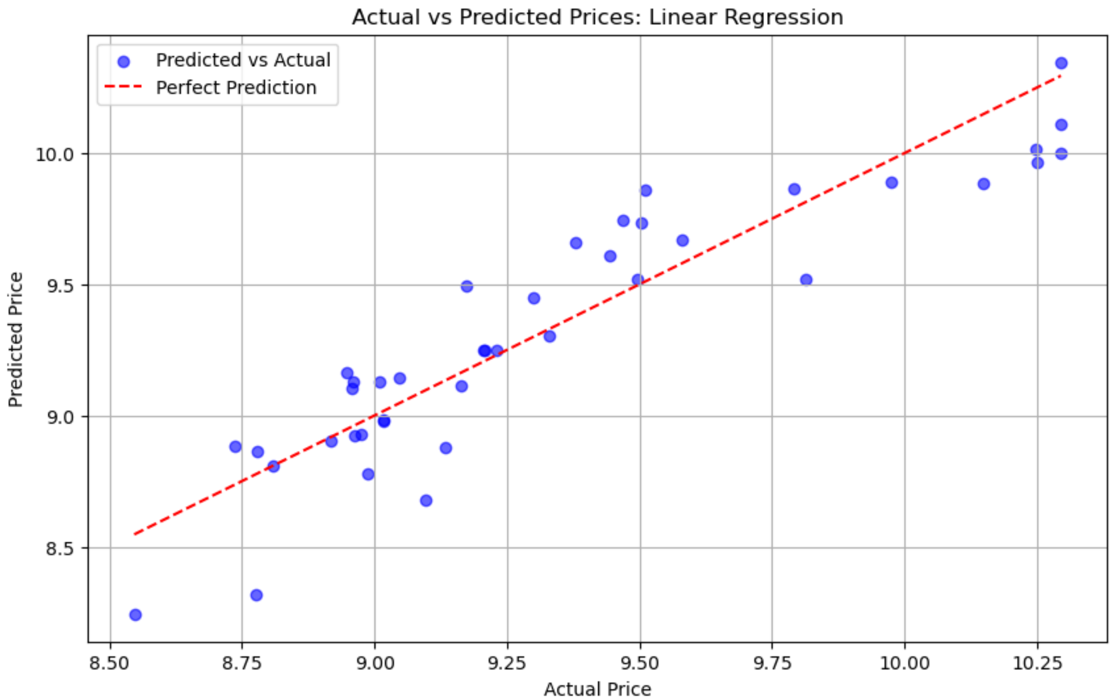
[243]:
```
# Display results
results_df = pd.DataFrame(results).T
print(results_df)
```

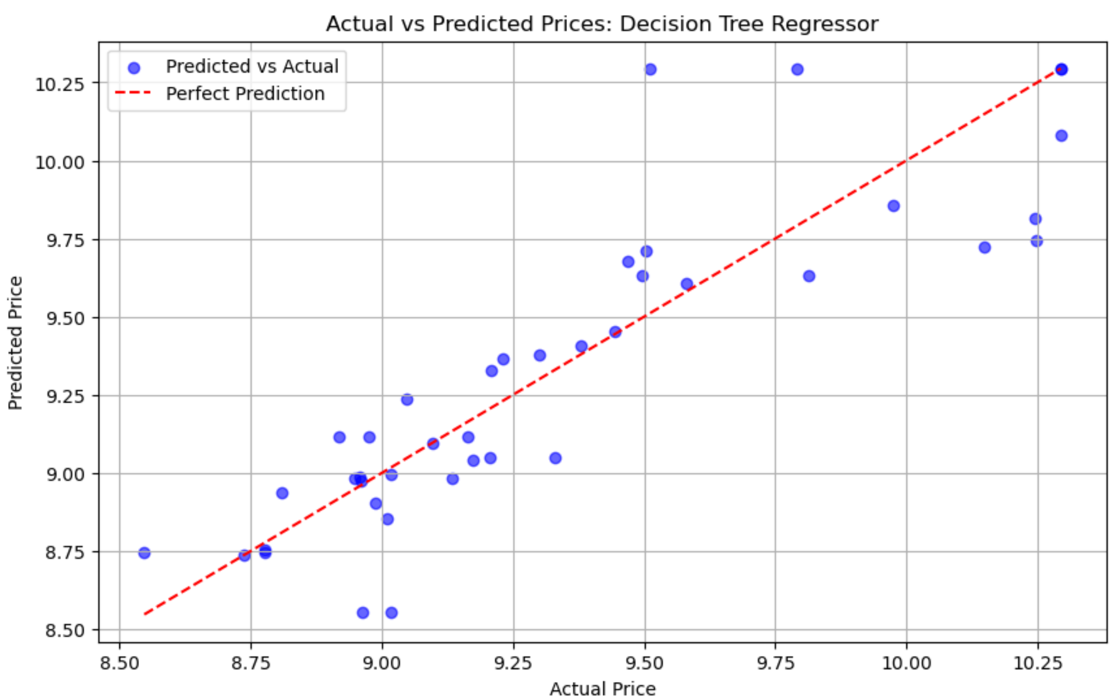
	R² Score	MAE	RMSE
Linear Regression	0.825697	0.161574	0.201772
Decision Tree Regressor	0.744972	0.170912	0.244063
Random Forest Regressor	0.876245	0.136639	0.170016
Gradient Boosting Regressor	0.891191	0.133739	0.159419
Support Vector Regressor	0.790854	0.177862	0.221020

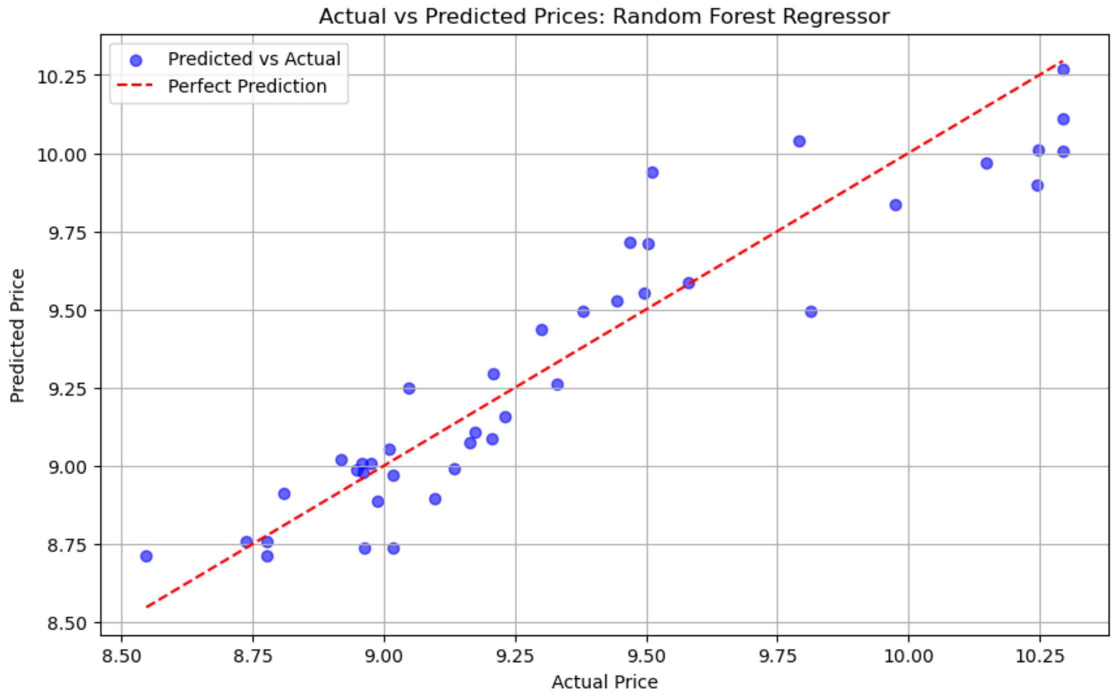
1.16 Visualisation

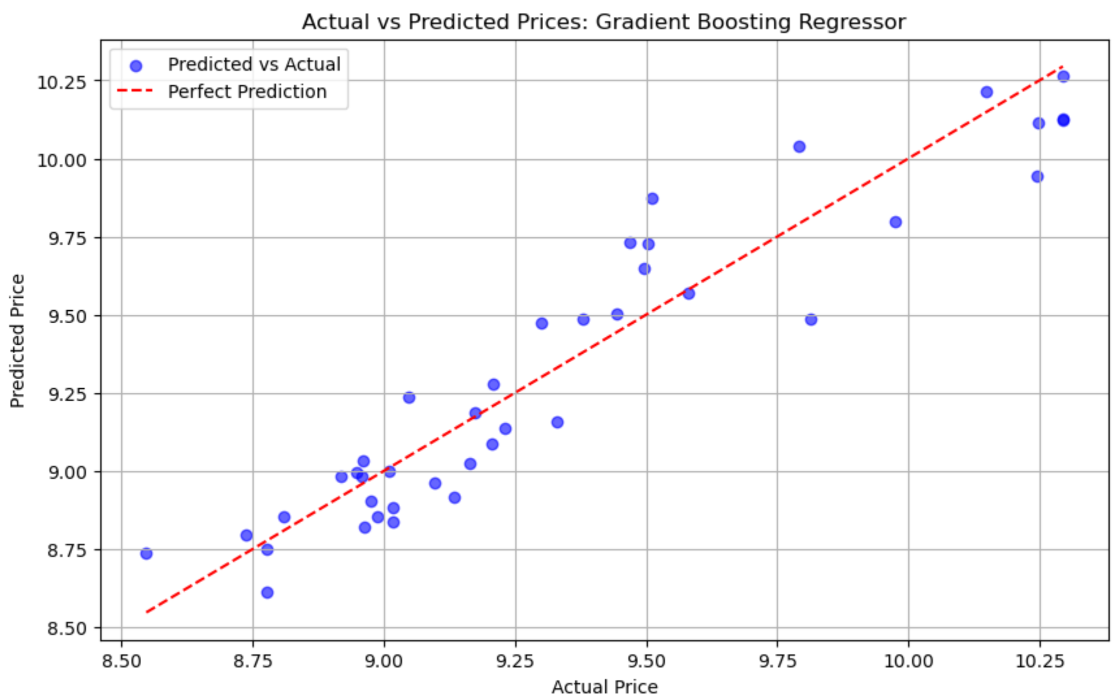
[253]:
```
# Function to plot actual vs predicted prices
def plot_actual_vs_predicted(y_test, y_pred, model_name):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predicted vs␣
 ↪Actual')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color='red', linestyle='--', label='Perfect Prediction')
    plt.title(f'Actual vs Predicted Prices: {model_name}')
    plt.xlabel('Actual Price')
    plt.ylabel('Predicted Price')
    plt.legend()
    plt.grid(True)
    plt.show()
```

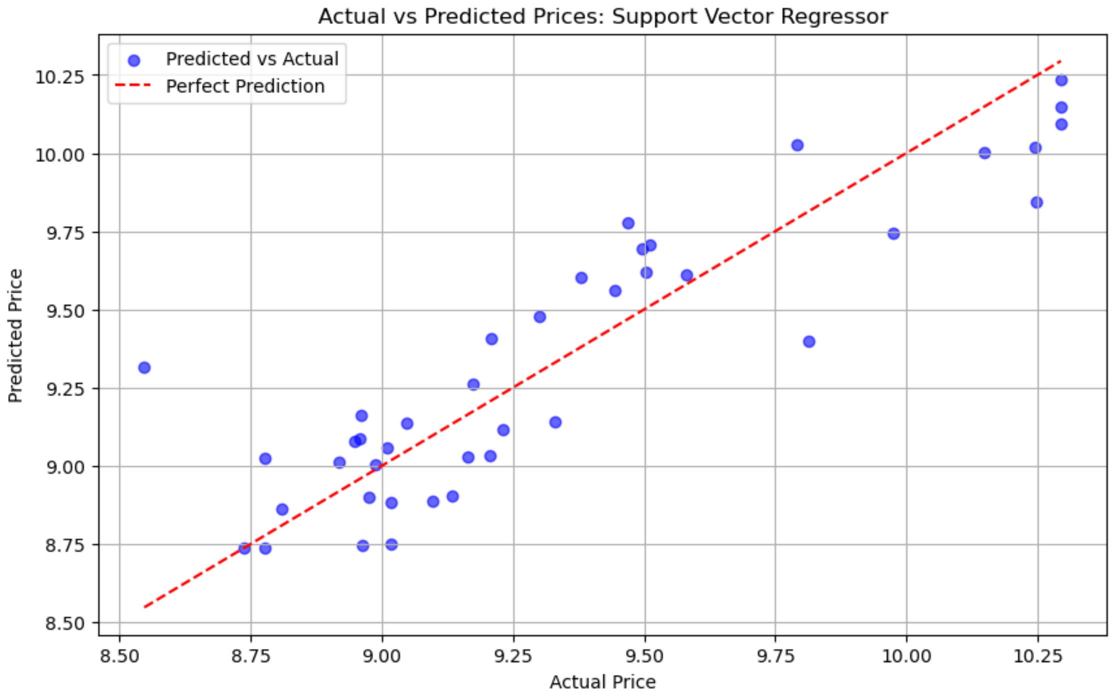
[255]:
```
# Iterate through models to plot predictions
for name, model in models.items():
    # Generate predictions for the test set
    y_pred = model.predict(X_test)

    # Call the plotting function
    plot_actual_vs_predicted(y_test, y_pred, name)
```


Actual vs Predicted Prices: Linear Regression
Predicted vs Actual
Perfect Prediction
Predicted Price
Actual Price
8.5
9.0
9.5
10.0
8.50
8.75
9.00
9.25
9.50
9.75
10.00
10.25


Actual vs Predicted Prices: Decision Tree Regressor
Predicted vs Actual
Perfect Prediction
Predicted Price
Actual Price
8.50
8.75
9.00
9.25
9.50
9.75
10.00
10.25


Actual vs Predicted Prices: Random Forest Regressor
Predicted vs Actual
Perfect Prediction
Predicted Price
Actual Price
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
8.50
8.75
9.00
9.25
9.50
9.75
10.00
10.25


Actual vs Predicted Prices: Gradient Boosting Regressor
Predicted vs Actual
Perfect Prediction
Predicted Price
Actual Price
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
8.50
8.75
9.00
9.25
9.50
9.75
10.00
10.25



1.17 Finding the best model

[257]:
```
# Find the best model based on R² Score
best_model_name = results_df['R² Score'].idxmax()
best_model_metrics = results_df.loc[best_model_name]

print(f"Best Model: {best_model_name}")
print("\nPerformance Metrics:")
print(best_model_metrics)
```

```
Best Model: Gradient Boosting Regressor

Performance Metrics:
R² Score    0.891191
MAE         0.133739
RMSE        0.159419
Name: Gradient Boosting Regressor, dtype: float64
```

[259]:
```
# Rank models by R² Score
ranked_models = results_df.sort_values(by='R² Score', ascending=False)
print("Ranked Models by R² Score:")
print(ranked_models)
```

```
Ranked Models by R² Score:
                             R² Score       MAE      RMSE
```

```
Gradient Boosting Regressor  0.891191  0.133739  0.159419
Random Forest Regressor      0.876245  0.136639  0.170016
Linear Regression            0.825697  0.161574  0.201772
Support Vector Regressor     0.790854  0.177862  0.221020
Decision Tree Regressor      0.744972  0.170912  0.244063
```

1.17.1 Best Model is Gradient Boosting Regressor

1.18 13. Hyperperameter Tuning

[266]:
```
# Define the model
gbr = GradientBoostingRegressor(random_state=42)

# Define hyperparameters to tune
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.6, 0.8, 1.0]
}

# Perform Grid Search with Cross Validation
grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid,
                           scoring='r2', cv=5, verbose=2, n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train)

# Best parameters and performance
print("Best Parameters:", grid_search.best_params_)
print("Best R² Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 1296 candidates, totalling 6480 fits
Best Parameters: {'learning_rate': 0.05, 'max_depth': 6, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 200, 'subsample': 0.6}
Best R² Score: 0.8970872363382394
```

[268]:
```
# Use the best estimator for predictions
best_gbr = grid_search.best_estimator_
y_pred = best_gbr.predict(X_test)

# Evaluate on the test set
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```python
print("\nTest Set Performance:")
print(f"R² Score: {r2:.4f}")
print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
```

```
Test Set Performance:
R² Score: 0.9089
MAE: 0.12
RMSE: 0.15
```

1.19 Saving the Model

[273]:
```python
# Save the model to a file
joblib.dump(best_gbr, 'car_price_prediction_gb_model.joblib')
print("Model saved as 'car_price_prediction_gb_model.joblib'")
```

```
Model saved as 'car_price_prediction_gb_model.joblib'
```

1.20 Conclusion

Final Insights: The Gradient Boosting Regressor model provides actionable insights into the car price dynamics, which can help the business:

Design cars that meet specific price targets. Strategize pricing based on critical features influencing the price. This project demonstrates a structured approach to solving regression problems and highlights the value of machine learning in deriving business insights.

[ ]:
```python

```