In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
data = pd.read_csv('D:/New folder/AutoData.csv')
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
data.head()
```

Out[4]:

ıboling	make	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	whee
3	alfa-romero giulia	gas	std	two	convertible	rwd	front	
3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	
1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	
2	audi 100 ls	gas	std	four	sedan	fwd	front	
2	audi 100ls	gas	std	four	sedan	4wd	front	

× 25 columns

In [5]:
```
data.tail()
```

Out[5]:

	symboling	make	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	\
200	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	
201	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	
202	-1	volvo 244dl	gas	std	four	sedan	rwd	front	
203	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	
204	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	

5 rows × 25 columns

In [6]:
```
data.shape
```

Out[6]:
```
(205, 25)
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   symboling         205 non-null    int64  
 1   make              205 non-null    object 
 2   fueltype          205 non-null    object 
 3   aspiration        205 non-null    object 
 4   doornumber        205 non-null    object 
 5   carbody           205 non-null    object 
 6   drivewheel        205 non-null    object 
 7   enginelocation    205 non-null    object 
 8   wheelbase         205 non-null    float64
 9   carlength         205 non-null    float64
 10  carwidth          205 non-null    float64
 11  carheight         205 non-null    float64
 12  curbweight        205 non-null    int64  
 13  enginetype        205 non-null    object 
 14  cylindernumber    205 non-null    object 
 15  enginesize        205 non-null    int64  
 16  fuelsystem        205 non-null    object 
 17  boreratio         205 non-null    float64
 18  stroke            205 non-null    float64
 19  compressionratio  205 non-null    float64
 20  horsepower        205 non-null    int64  
 21  peakrpm           205 non-null    int64  
 22  citympg           205 non-null    int64  
 23  highwaympg        205 non-null    int64  
 24  price             205 non-null    float64
dtypes: float64(8), int64(7), object(10)
memory usage: 40.2+ KB
```

In [8]:
```
data.describe()
```

Out[8]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	l
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	20
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	

In [9]:
```
data.nunique()
```

Out[9]:
```
symboling             6
make                147
fueltype              2
aspiration            2
doornumber            2
carbody               5
drivewheel            3
enginelocation        2
wheelbase            53
carlength            75
carwidth             44
carheight            49
curbweight          171
enginetype            7
cylindernumber        7
enginesize           44
fuelsystem            8
boreratio            38
stroke               37
compressionratio     32
horsepower           59
peakrpm              23
citympg              29
highwaympg           30
price               189
dtype: int64
```

In [10]:
```
print(data['fueltype'].value_counts())
print(data['aspiration'].value_counts())
print(data['doornumber'].value_counts())
print(data['carbody'].value_counts())
print(data['drivewheel'].value_counts())
print(data['enginelocation'].value_counts())
```

```
gas       185
diesel     20
Name: fueltype, dtype: int64
std      168
turbo     37
Name: aspiration, dtype: int64
four    115
two      90
Name: doornumber, dtype: int64
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: carbody, dtype: int64
fwd    120
rwd     76
4wd      9
Name: drivewheel, dtype: int64
front    202
rear       3
Name: enginelocation, dtype: int64
```

In [11]:
```
###data cleaning
```

In [12]:
```
#checking duplicates
data[data.duplicated()]
```

Out[12]:

symboling	make	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	whee

0 rows × 25 columns

no duplicate values

In [13]:
```
##checking null values
data.isnull().sum()
```

Out[13]:
```
symboling           0
make                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 values

In [14]:
```
##we see that in make column car company and model given,splitting the company and model
data['car_company'] = data.make.str.split(' ').str.get(0).str.upper()
```

In [15]:
```
data.head()
```

Out[15]:

	symboling	make	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocati
0	3	alfa-romero giulia	gas	std	two	convertible	rwd	fro
1	3	alfa-romero stelvio	gas	std	two	convertible	rwd	fro
2	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	fro
3	2	audi 100 ls	gas	std	four	sedan	fwd	fro
4	2	audi 100ls	gas	std	four	sedan	4wd	fro

5 rows × 26 columns

In [16]:
```
data.car_company.unique()
```

Out[16]:
```
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 [17]:
```
#VOLKSWAGEN has three different values as  VOKSWAGEN and VW
# MAZDA is also spelled as MAXDA
# PORSCHE as PORSCHE and PORCSCHE.
#TOYOTA AS TOYOUTA
```

In [18]:
```
data['car_company'] = data['car_company'].replace(['VOKSWAGEN','VW'],'VOLKSWAG
EN')
data['car_company'] = data['car_company'].replace(['MAXDA'],'MAZDA')
data['car_company'] = data['car_company'].replace(['PORCSHCE'],'PORSCHE')
data['car_company'] = data['car_company'].replace(['TOYOUTA'],'TOYOTA')
```

In [19]:
```
data.car_company.unique()
```

Out[19]:
```
array(['ALFA-ROMERO', 'AUDI', 'BMW', 'CHEVROLET', 'DODGE', 'HONDA',
       'ISUZU', 'JAGUAR', 'MAZDA', 'BUICK', 'MERCURY', 'MITSUBISHI',
       'NISSAN', 'PEUGEOT', 'PLYMOUTH', 'PORSCHE', 'RENAULT', 'SAAB',
       'SUBARU', 'TOYOTA', 'VOLKSWAGEN', 'VOLVO'], dtype=object)
```

In [20]:
```
data = data.drop(['make'], axis =1)
```

In [21]:
```
data.head()
```

Out[21]:

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbas
0	3	gas	std	two	convertible	rwd	front	88
1	3	gas	std	two	convertible	rwd	front	88
2	1	gas	std	two	hatchback	rwd	front	94
3	2	gas	std	four	sedan	fwd	front	99
4	2	gas	std	four	sedan	4wd	front	99

5 rows × 25 columns

Exploratory Data Analysis

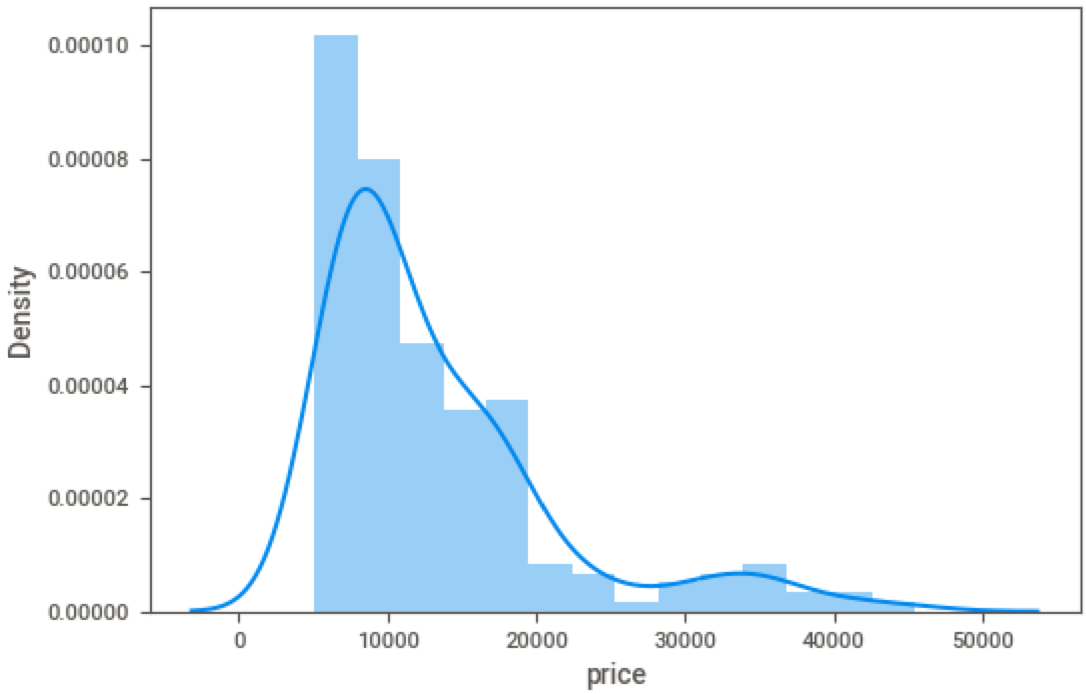
In [22]:
```
import sweetviz
my_report = sweetviz.analyze([data,"Automobile"],target_feat='price')
```

In [23]:
```
my_report.show_html('eda.html')
```

Report eda.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in your notebook/colab files.

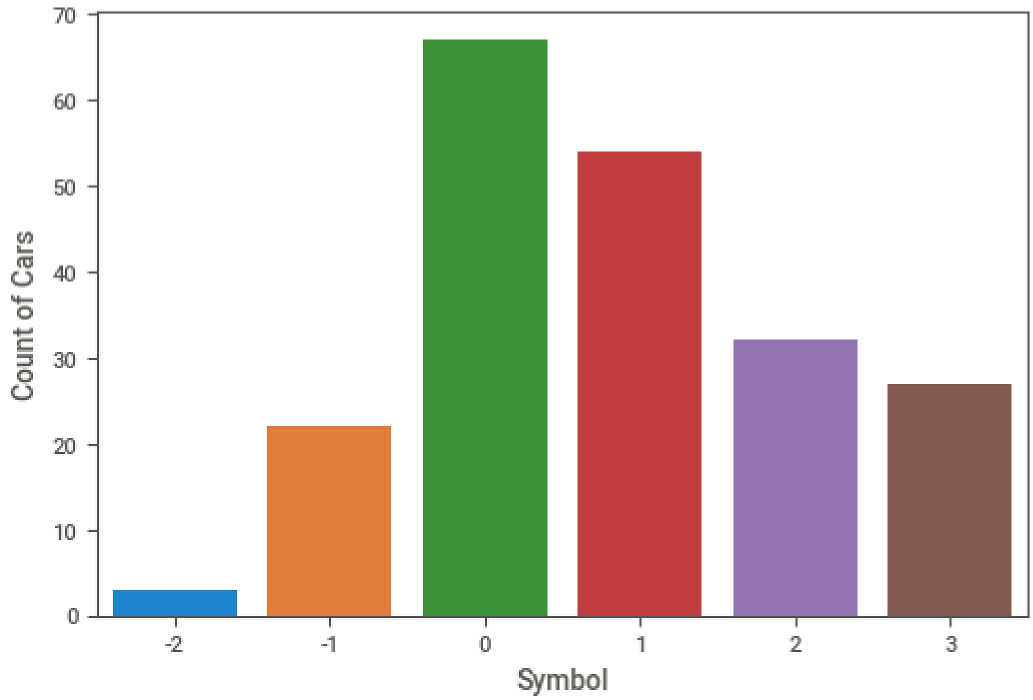
In [24]:
```
sns.distplot(data['price'])
```

Out[24]: <AxesSubplot:xlabel='price', ylabel='Density'>



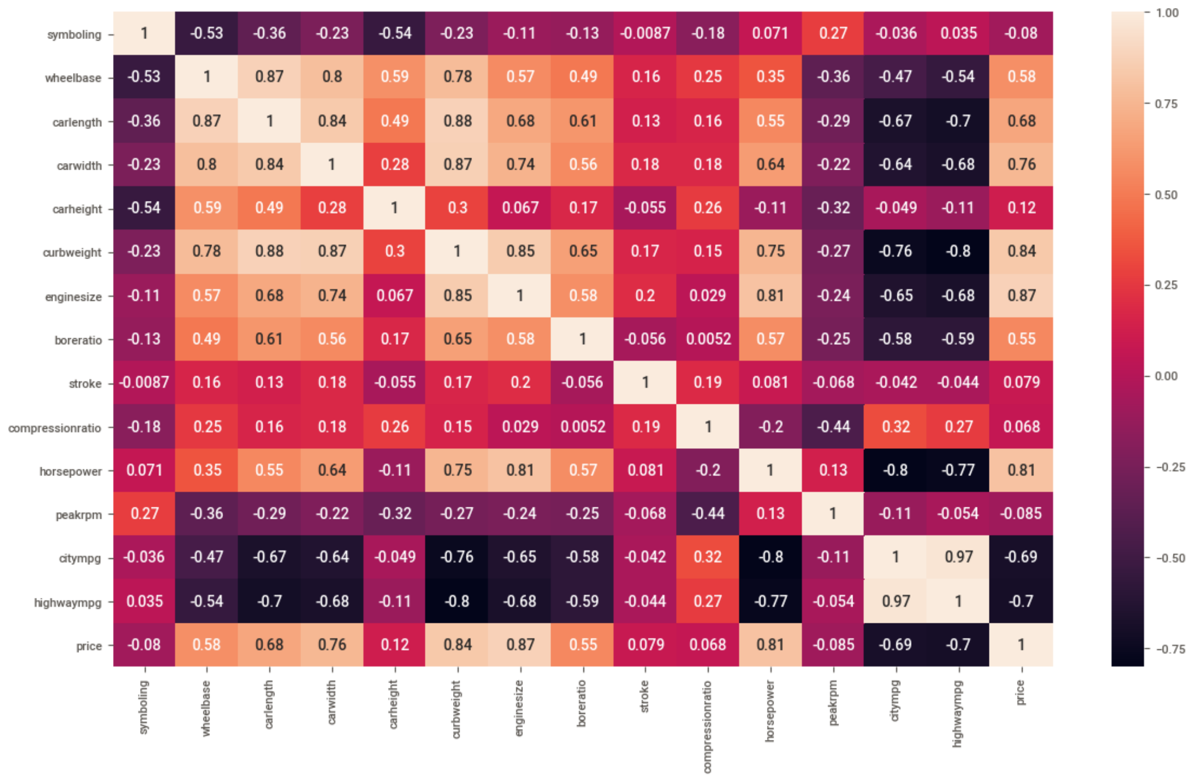
In [25]:
```
##symboling
ex = sns.countplot(data['symboling'])
ex.set(xlabel = 'Symbol',ylabel= 'Count of Cars')
```

Out[25]: [Text(0.5, 0, 'Symbol'), Text(0, 0.5, 'Count of Cars')]



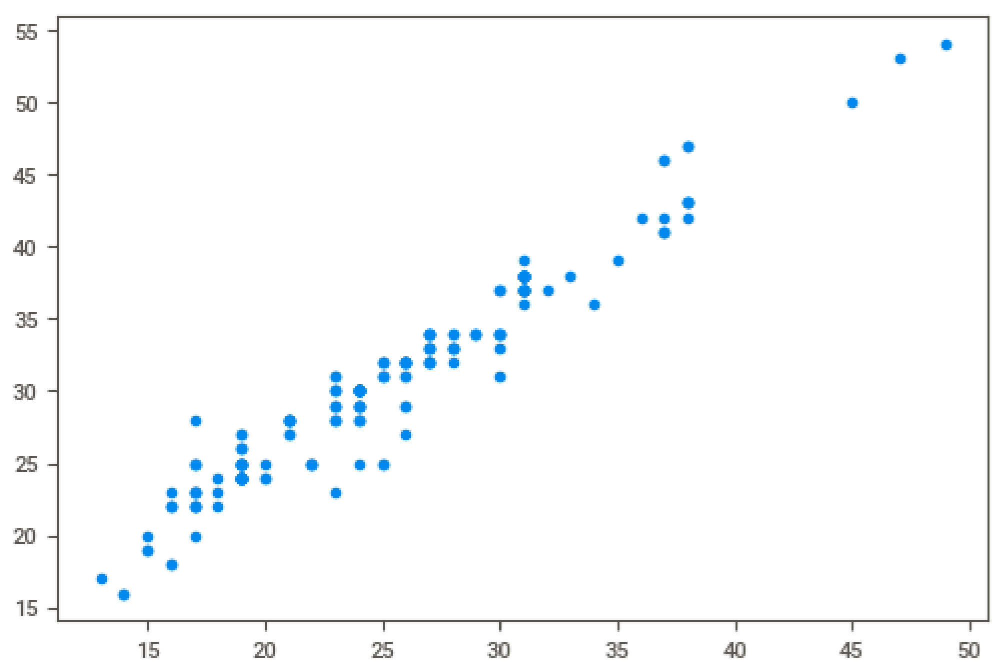
In [26]:
```
##correlation matrix
plt.figure(figsize=(14,8))
corr = data.corr()
sns.heatmap(corr,annot=True)
```

Out[26]: <AxesSubplot:>



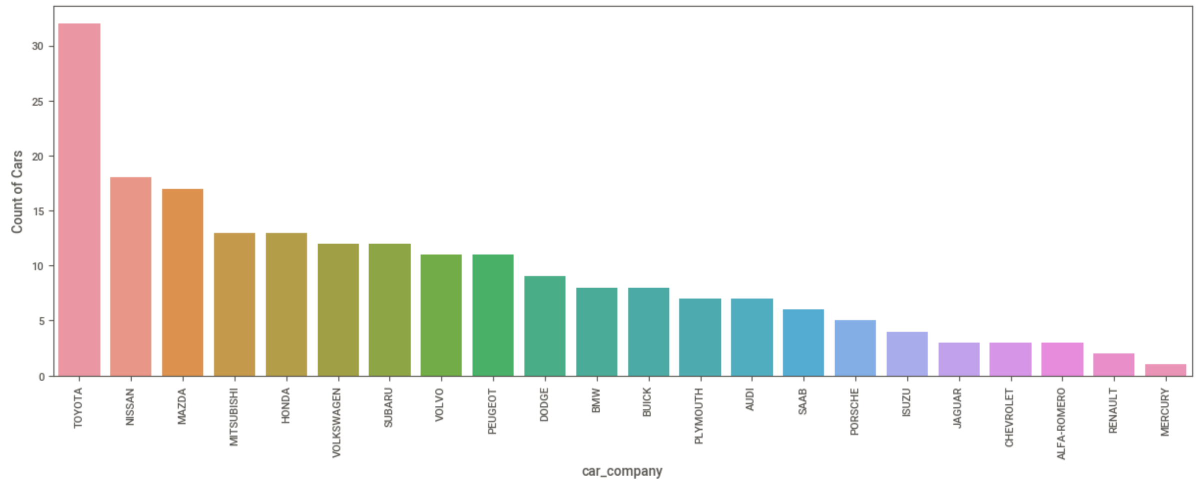
In [27]:
```
plt.scatter(data['citympg'],data['highwaympg'])
```

Out[27]: <matplotlib.collections.PathCollection at 0x228ca1edcd0>



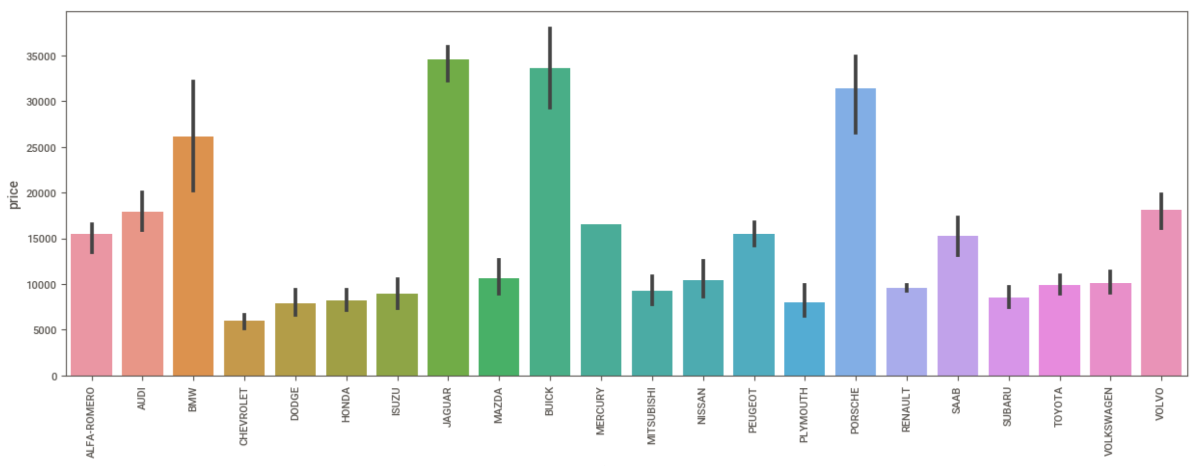
citympg and highwaympg are highly correlated

In [28]:
```
fig, ax = plt.subplots(figsize = (15,5))
plot = sns.countplot(data['car_company'], order=pd.value_counts(data['car_comp
any']).index,)
plot.set(xlabel = 'car_company', ylabel= 'Count of Cars')
plt.xticks(rotation=90)
plt.show()
```



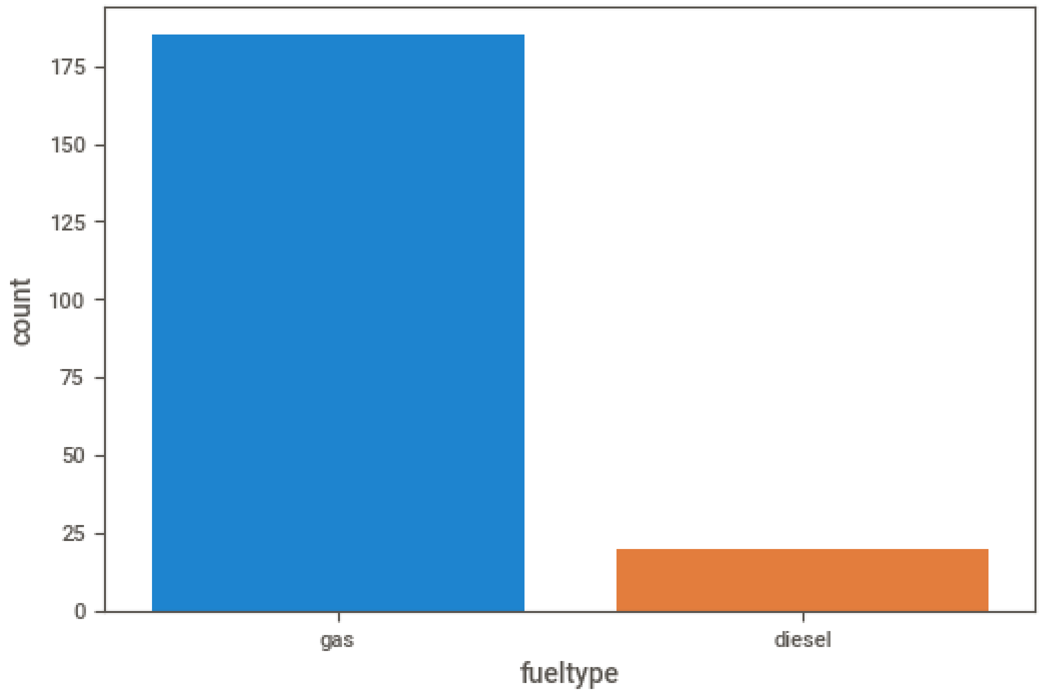
In [29]:
```
fig, ax = plt.subplots(figsize = (15,5))
sns.barplot('car_company','price',data=data)
plt.xticks(rotation=90)
```

Out[29]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]),
 [Text(0, 0, 'ALFA-ROMERO'),
  Text(1, 0, 'AUDI'),
  Text(2, 0, 'BMW'),
  Text(3, 0, 'CHEVROLET'),
  Text(4, 0, 'DODGE'),
  Text(5, 0, 'HONDA'),
  Text(6, 0, 'ISUZU'),
  Text(7, 0, 'JAGUAR'),
  Text(8, 0, 'MAZDA'),
  Text(9, 0, 'BUICK'),
  Text(10, 0, 'MERCURY'),
  Text(11, 0, 'MITSUBISHI'),
  Text(12, 0, 'NISSAN'),
  Text(13, 0, 'PEUGEOT'),
  Text(14, 0, 'PLYMOUTH'),
  Text(15, 0, 'PORSCHE'),
  Text(16, 0, 'RENAULT'),
  Text(17, 0, 'SAAB'),
  Text(18, 0, 'SUBARU'),
  Text(19, 0, 'TOYOTA'),
  Text(20, 0, 'VOLKSWAGEN'),
  Text(21, 0, 'VOLVO')])
```

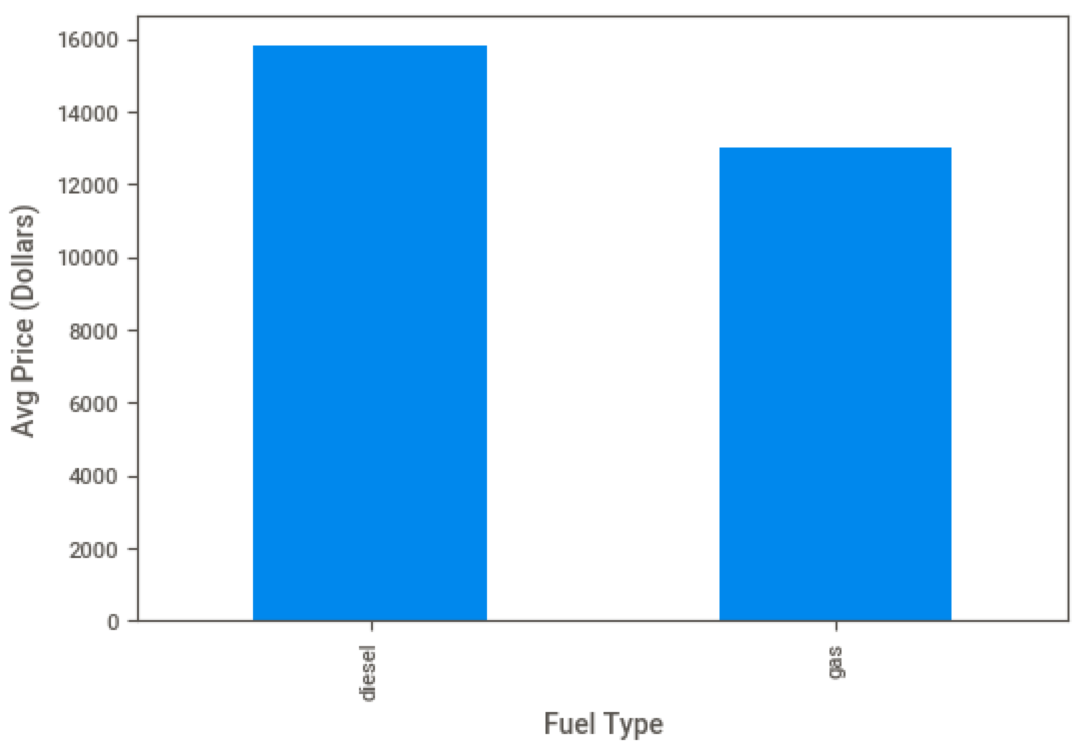


In [30]:
```
sns.countplot(data['fueltype'])
```

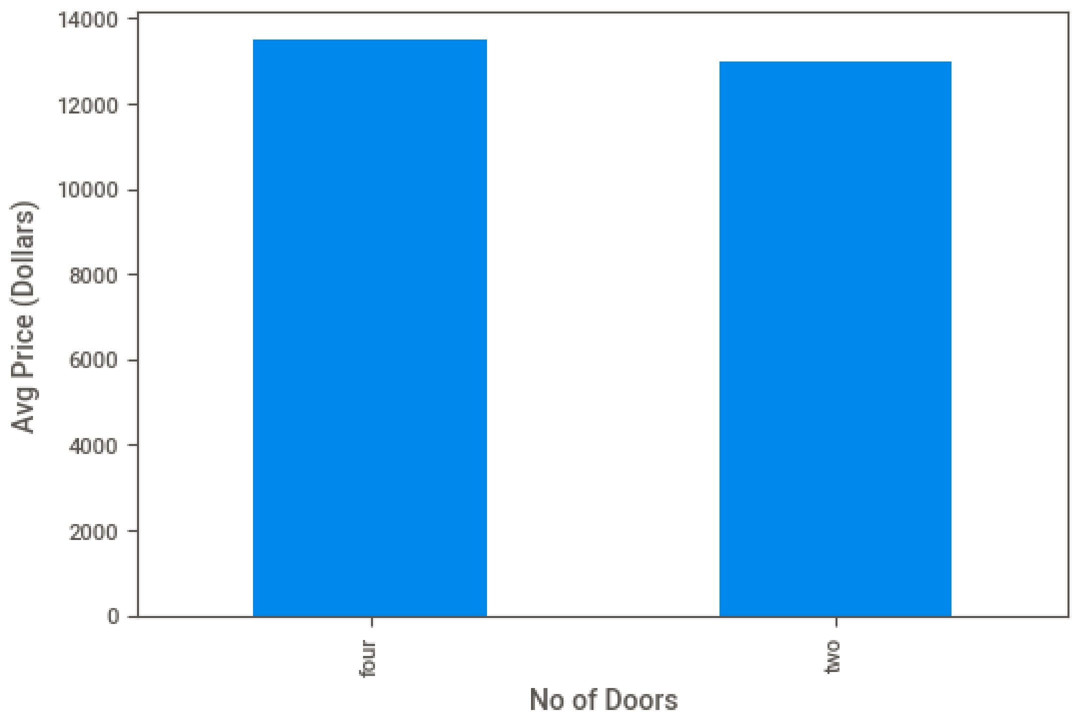
Out[30]: <AxesSubplot:xlabel='fueltype', ylabel='count'>



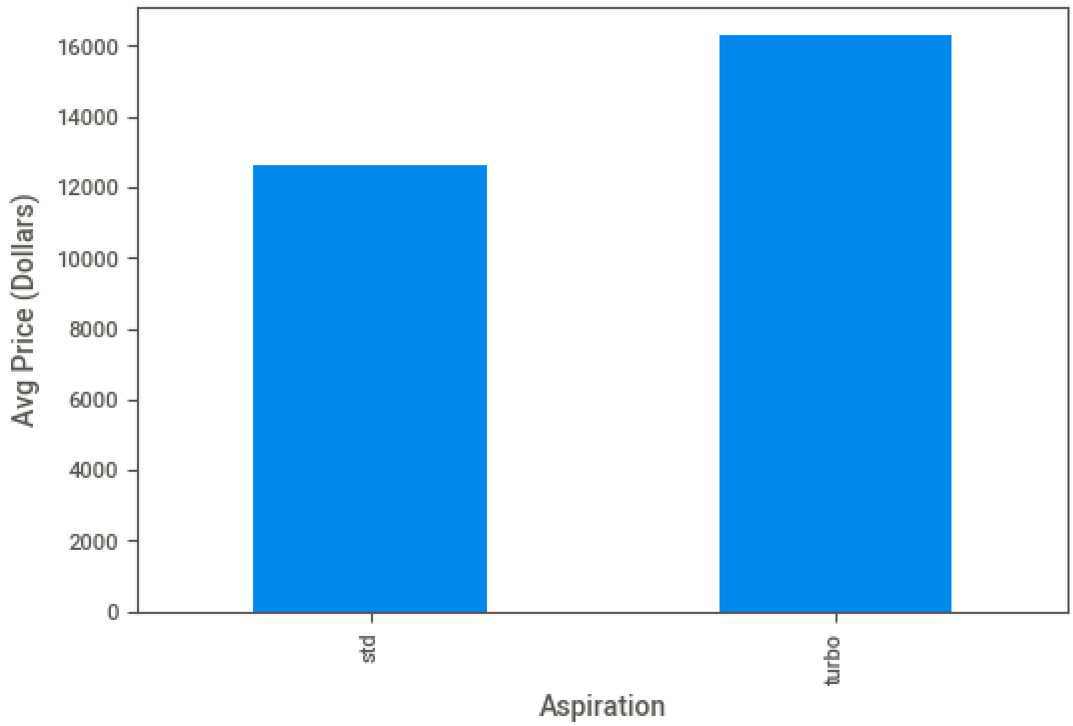
In [31]:
```
fuel_avg_price = data[['fueltype','price']].groupby("fueltype", as_index = False).mean().rename(columns={'price':'fuel_avg_price'})
plt1 = fuel_avg_price.plot(x = 'fueltype', kind='bar',legend = False, sort_columns = True)
plt1.set_xlabel("Fuel Type")
plt1.set_ylabel("Avg Price (Dollars)")
plt.show()
```



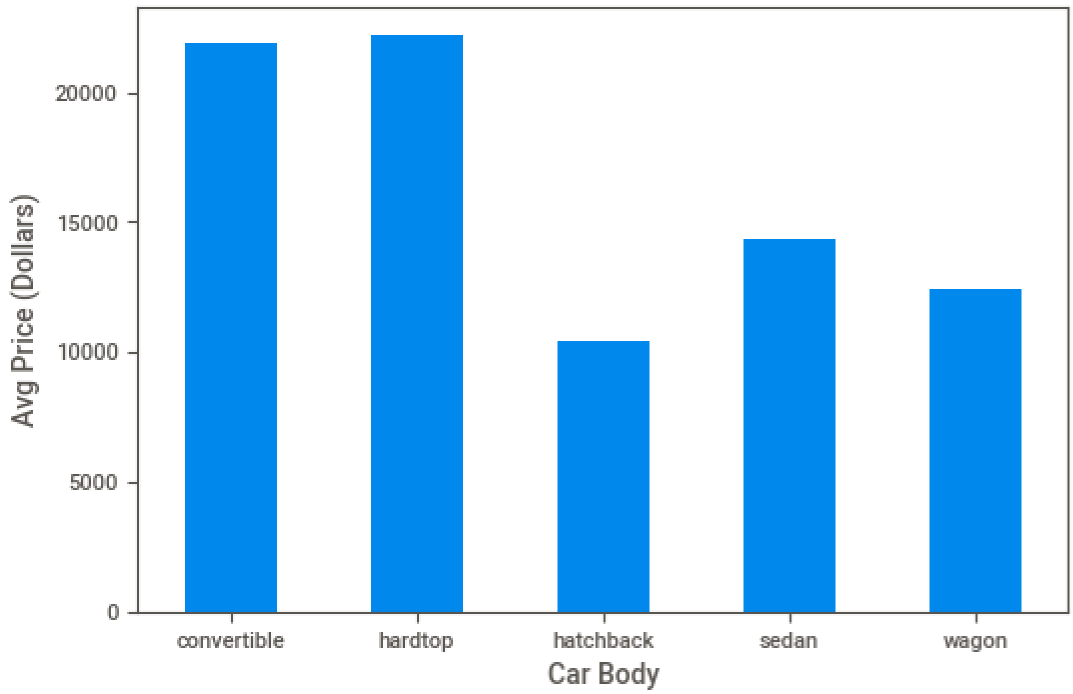
In [32]:
```
door_avg_price = data[['doornumber','price']].groupby("doornumber", as_index =
False).mean().rename(columns={'price':'door_avg_price'})
plt1 = door_avg_price.plot(x = 'doornumber', kind='bar',legend = False, sort_c
olumns = True)
plt1.set_xlabel("No of Doors")
plt1.set_ylabel("Avg Price (Dollars)")
plt.show()
```



In [33]:
```
aspir_avg_price = data[['aspiration','price']].groupby("aspiration", as_index
= False).mean().rename(columns={'price':'aspir_avg_price'})
plt1 = aspir_avg_price.plot(x = 'aspiration', kind='bar',legend = False, sort_
columns = True)
plt1.set_xlabel("Aspiration")
plt1.set_ylabel("Avg Price (Dollars)")

plt.show()
```

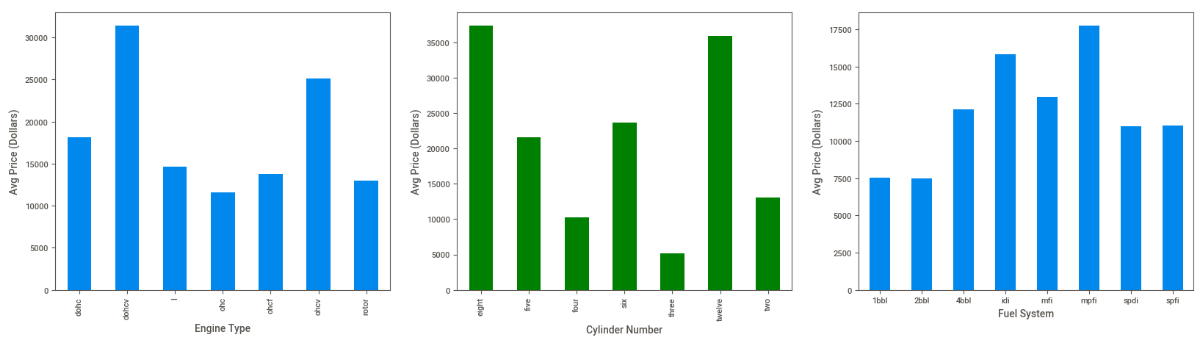


In [34]:
```
df_body_avg_price = data[['carbody','price']].groupby("carbody", as_index = False).mean().rename(columns={'price':'carbody_avg_price'})
plt1 = df_body_avg_price.plot(x = 'carbody', kind='bar',legend = False, sort_columns = True)
plt1.set_xlabel("Car Body")
plt1.set_ylabel("Avg Price (Dollars)")
plt.xticks(rotation = 0)
plt.show()
```

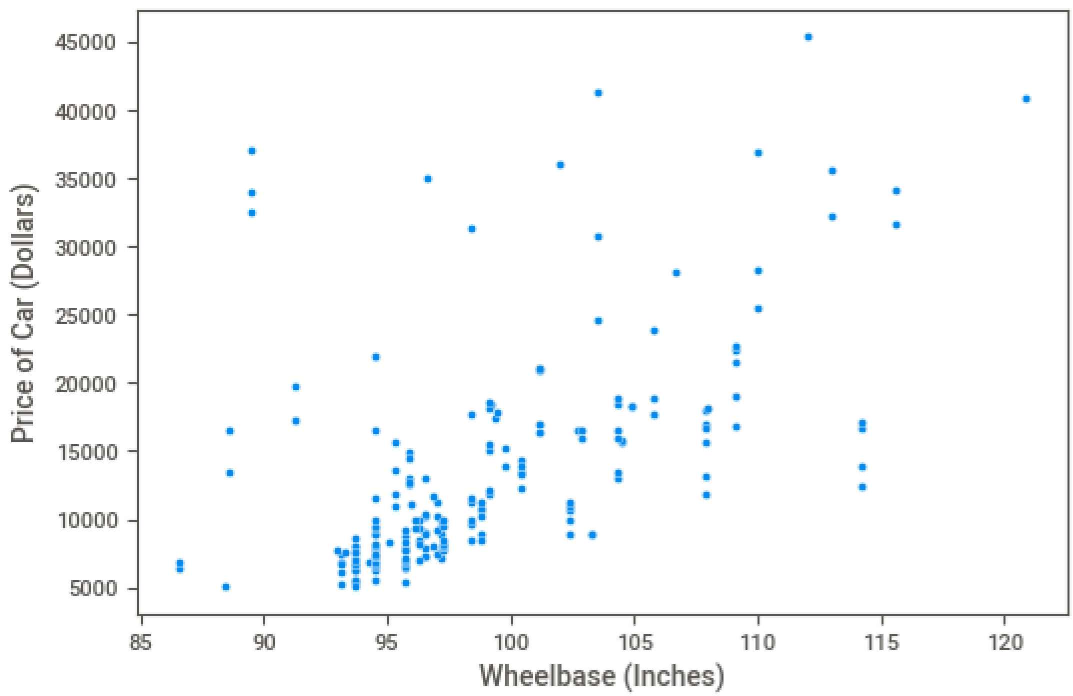


In [35]:

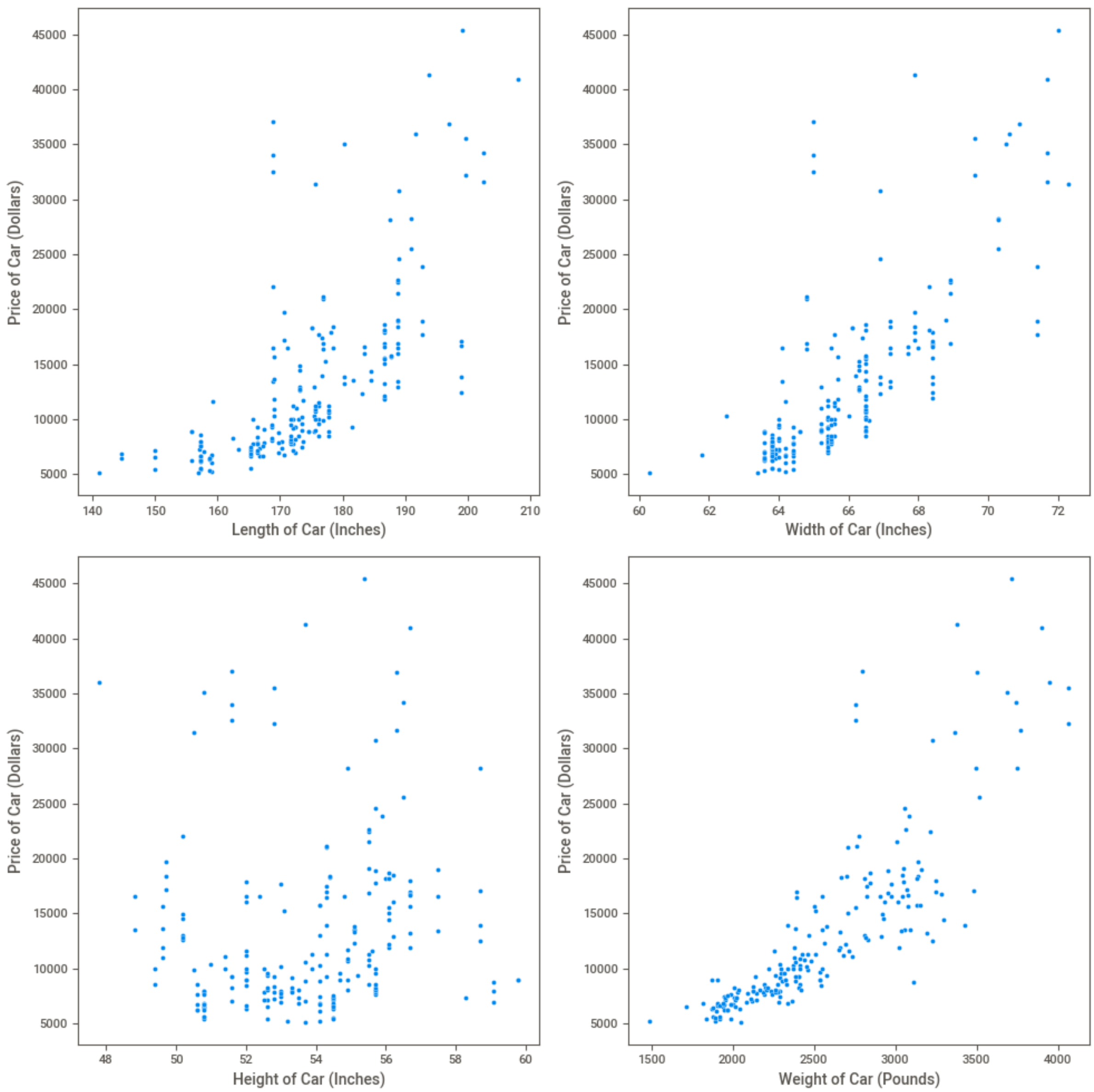
```
fig, axs = plt.subplots(1,3,figsize=(20,5))


df_engine_avg_price = data[['enginetype','price']].groupby("enginetype", as_index = False).mean().rename(columns={'price':'engine_avg_price'})
plt1 = df_engine_avg_price.plot(x = 'enginetype', kind='bar', sort_columns = True, legend = False, ax = axs[0])
plt1.set_xlabel("Engine Type")
plt1.set_ylabel("Avg Price (Dollars)")
plt.xticks(rotation = 0)
df_cylindernumber_avg_price = data[['cylindernumber','price']].groupby("cylindernumber", as_index = False).mean().rename(columns={'price':'cylindernumber_avg_price'})
plt1 = df_cylindernumber_avg_price.plot(x = 'cylindernumber', kind='bar',color='g', sort_columns = True,legend = False, ax = axs[1])
plt1.set_xlabel("Cylinder Number")
plt1.set_ylabel("Avg Price (Dollars)")
plt.xticks(rotation = 0)
df_fuelsystem_avg_price = data[['fuelsystem','price']].groupby("fuelsystem", as_index = False).mean().rename(columns={'price':'fuelsystem_avg_price'})
plt1 = df_fuelsystem_avg_price.plot(x = 'fuelsystem', kind='bar', sort_columns = True,legend = False, ax = axs[2])
plt1.set_xlabel("Fuel System")
plt1.set_ylabel("Avg Price (Dollars)")
plt.xticks(rotation = 0)
plt.show()
```



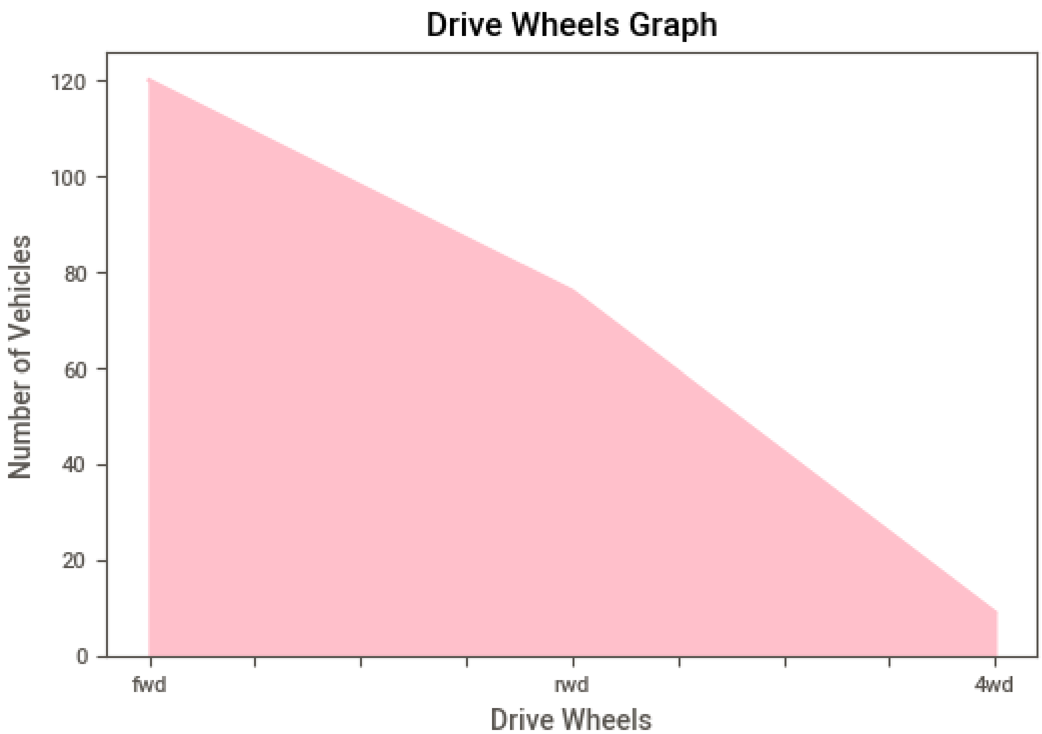
In [36]:
```
plt1 = sns.scatterplot(x = 'wheelbase', y = 'price', data = data)
plt1.set_xlabel('Wheelbase (Inches)')
plt1.set_ylabel('Price of Car (Dollars)')
plt.show()
```



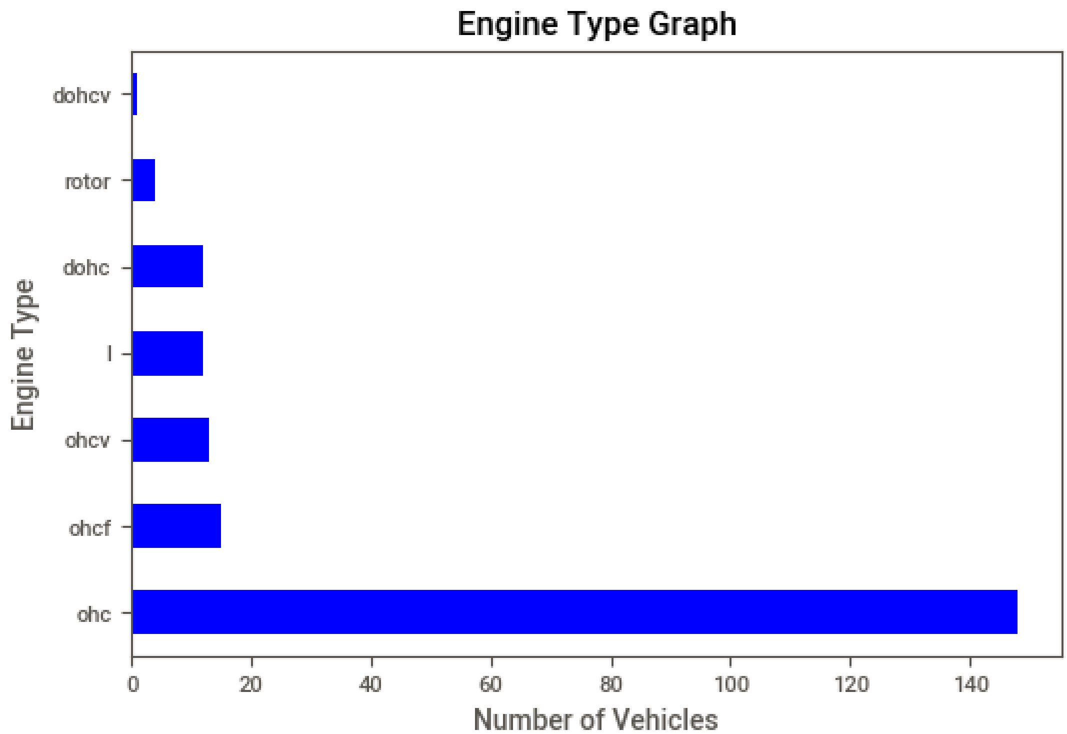
In [37]:
```
fig, axs = plt.subplots(2,2,figsize=(10,10))
plt1 = sns.scatterplot(x = 'carlength', y = 'price', data = data, ax = axs[0,0
])
plt1.set_xlabel('Length of Car (Inches)')
plt1.set_ylabel('Price of Car (Dollars)')
plt2 = sns.scatterplot(x = 'carwidth', y = 'price', data = data, ax = axs[0,1
])
plt2.set_xlabel('Width of Car (Inches)')
plt2.set_ylabel('Price of Car (Dollars)')
plt3 = sns.scatterplot(x = 'carheight', y = 'price', data = data, ax = axs[1,0
])
plt3.set_xlabel('Height of Car (Inches)')
plt3.set_ylabel('Price of Car (Dollars)')
plt3 = sns.scatterplot(x = 'curbweight', y = 'price', data = data, ax = axs[1,
1])
plt3.set_xlabel('Weight of Car (Pounds)')
plt3.set_ylabel('Price of Car (Dollars)')
plt.tight_layout()
```



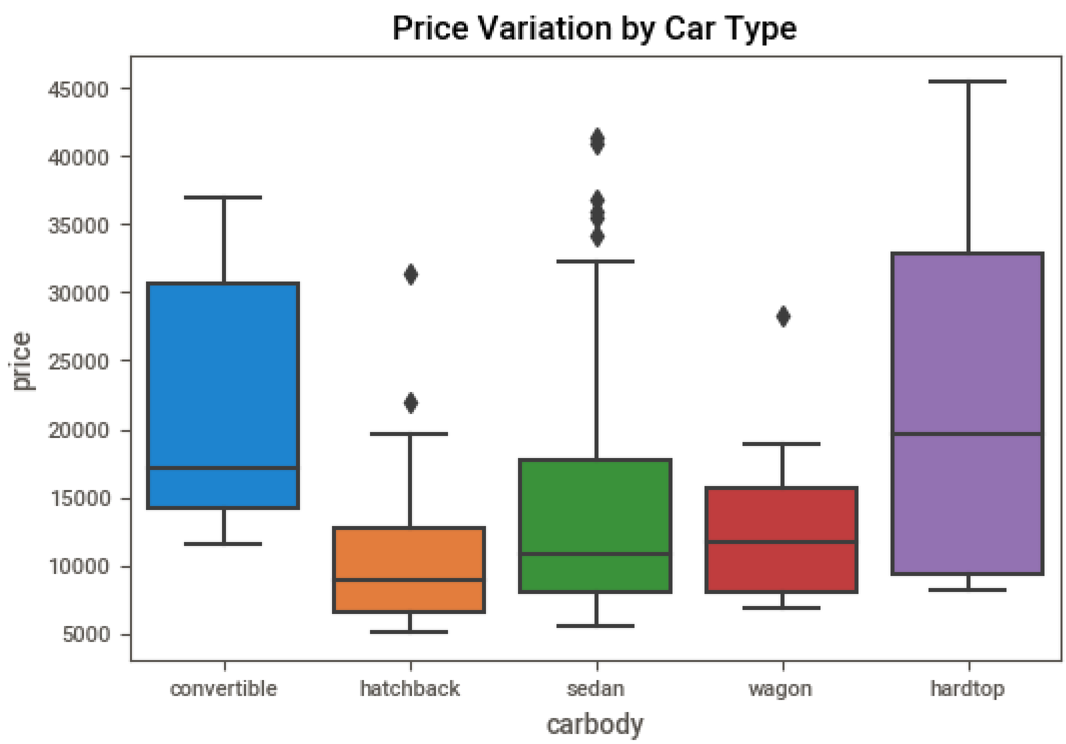
In [38]:
```
plt.title('Drive Wheels Graph')
plt.xlabel('Drive Wheels')
plt.ylabel('Number of Vehicles')
data['drivewheel'].value_counts().plot(kind='area',color='pink')
plt.show()
```



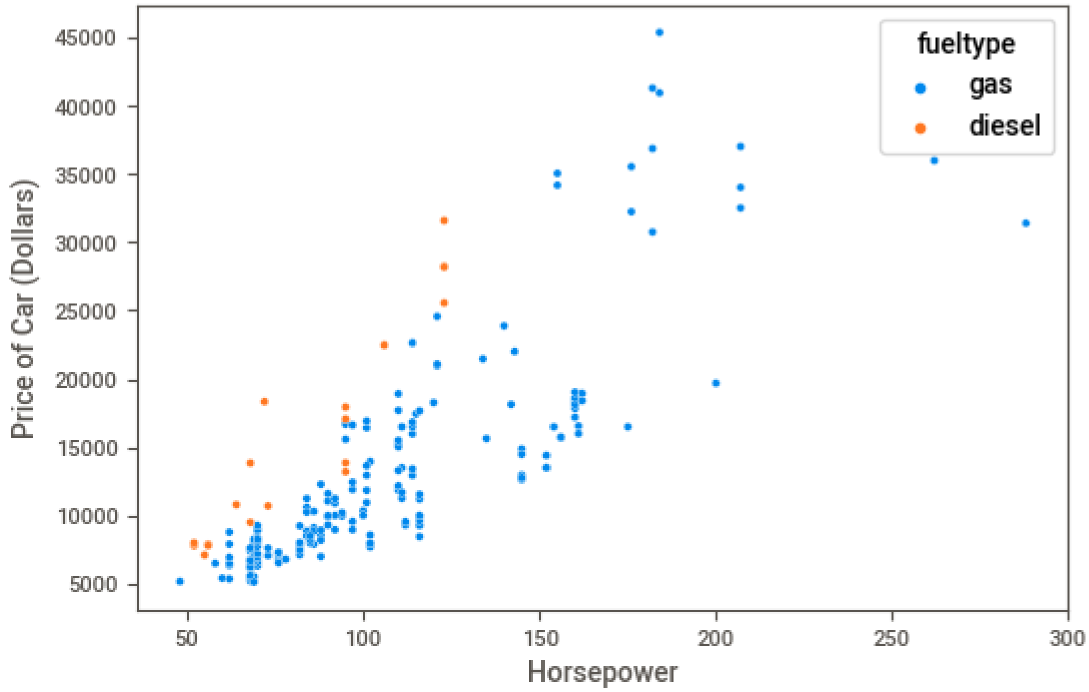
In [39]:
```
# Horizontal Bar Graph
plt.title('Engine Type Graph')
plt.ylabel('Engine Type')
plt.xlabel('Number of Vehicles')
data['enginetype'].value_counts().plot(kind='barh',color='b')
plt.show()
```



In [40]:
```
# Detecting outliers | Boxplot
plt.title('Price Variation by Car Type')
plt.xlabel('Car Type')
plt.ylabel('Price')
sns.boxplot(data['carbody'],data['price'])
plt.show()
```



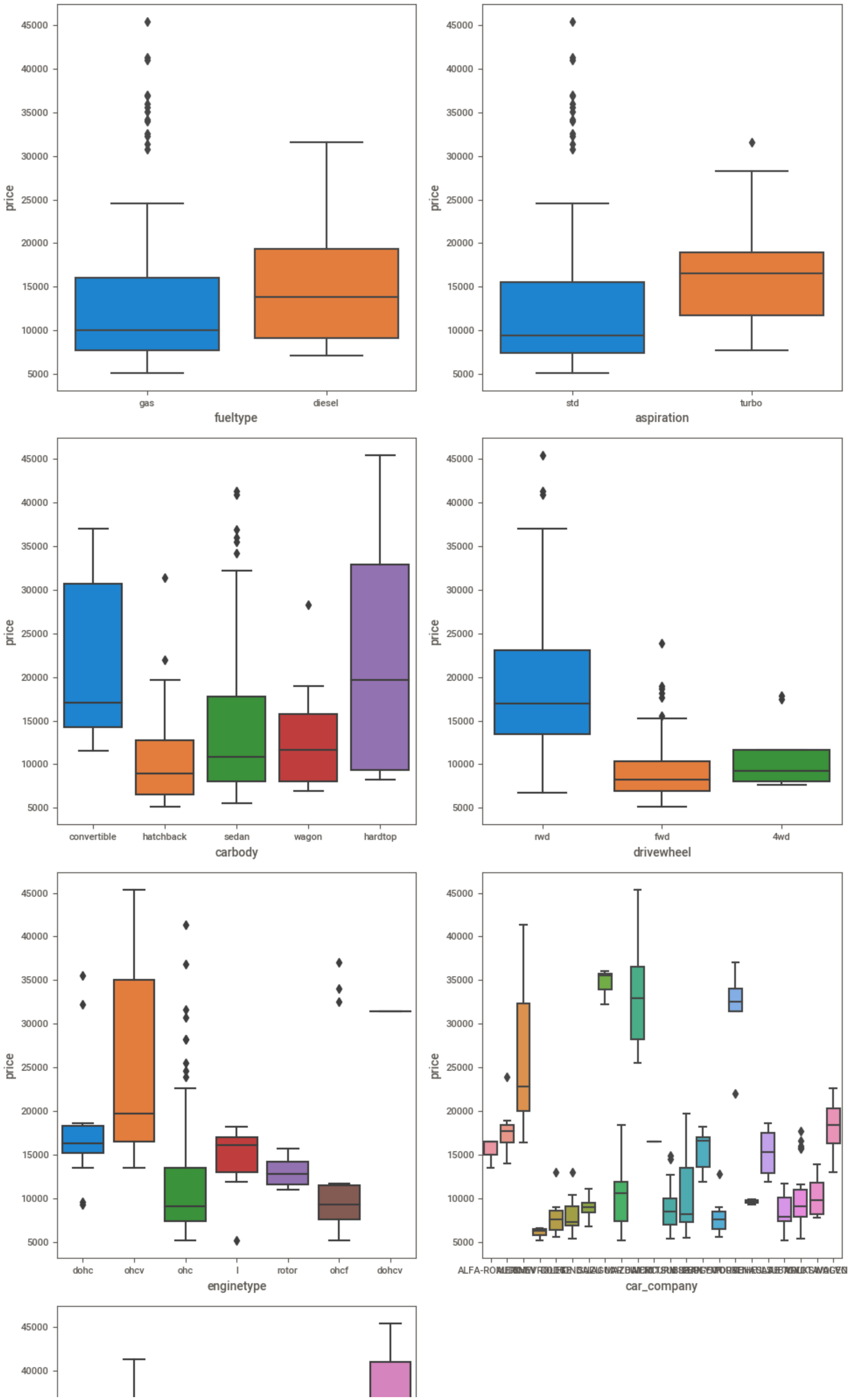
In [41]:
```
plt1 = sns.scatterplot(x = 'horsepower', y = 'price', hue = 'fueltype', data = data)
plt1.set_xlabel('Horsepower')
plt1.set_ylabel('Price of Car (Dollars)')
plt.show()
```

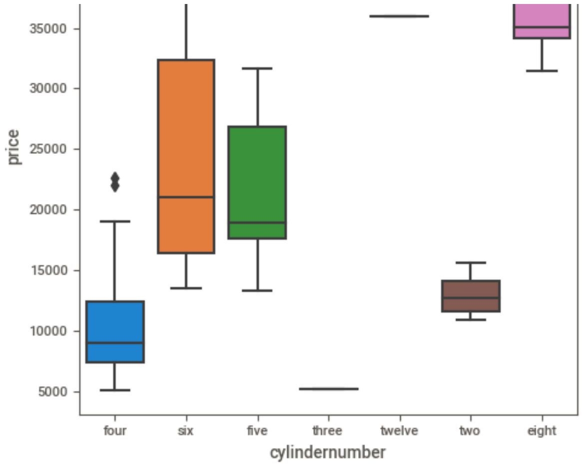


In [42]:
```
from pandas_visual_analysis import VisualAnalysis
```

In [43]:
```
VisualAnalysis(data)
```

In [44]:
```
plt.figure(figsize=(10, 20))
plt.subplot(4,2,1)
sns.boxplot(x = 'fueltype', y = 'price', data = data)
plt.subplot(4,2,2)
sns.boxplot(x = 'aspiration', y = 'price', data = data)
plt.subplot(4,2,3)
sns.boxplot(x = 'carbody', y = 'price', data = data)
plt.subplot(4,2,4)
sns.boxplot(x = 'drivewheel', y = 'price', data = data)
plt.subplot(4,2,5)
sns.boxplot(x = 'enginetype', y = 'price', data = data)
plt.subplot(4,2,6)
sns.boxplot(x = 'car_company', y = 'price', data = data)
plt.subplot(4,2,7)
sns.boxplot(x = 'cylindernumber', y = 'price', data = data)
plt.tight_layout()
plt.show()
```


price
fueltype
gas
diesel
aspiration
std
turbo
carbody
convertible
hatchback
sedan
wagon
hardtop
drivewheel
rwd
fwd
4wd
enginetype
dohc
ohcv
ohc
l
rotor
ohcf
dohcv
car_company



In [45]:
```
import dtale
```

In [46]:
```
dtale.show(data)
```

▸	0
0	

Out[46]:

Feature Engineering

In [47]:
```
# Identifying Categorical & Numerical Cols
cols = data.columns
num_cols = data._get_numeric_data().columns.to_list()
cat_cols = list(set(cols)-set(num_cols))

print('Numerical Columns')
print(num_cols)
print('\nCategorical Columns')
print(cat_cols)
```

```
Numerical Columns
['symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweigh
t', 'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower', 'p
eakrpm', 'citympg', 'highwaympg', 'price']

Categorical Columns
['drivewheel', 'carbody', 'enginelocation', 'car_company', 'enginetype', 'asp
iration', 'fuelsystem', 'cylindernumber', 'doornumber', 'fueltype']
```

In [48]:
```
##label encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for i in data[cat_cols]:
    data[i] = le.fit_transform(data[i])
```

In [49]:
```
data.head()
```

Out[49]:

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase
0	3	1	0	1	0	2	0	88.6
1	3	1	0	1	0	2	0	88.6
2	1	1	0	1	2	2	0	94.5
3	2	1	0	0	3	1	0	99.8
4	2	1	0	0	3	0	0	99.4

5 rows × 25 columns

In [50]:
```
data.describe()
```

Out[50]:

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	0.902439	0.180488	0.439024	2.614634	1.326829	0.014634
std	1.245307	0.297446	0.385535	0.497483	0.859081	0.556171	0.120377
min	-2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	0.000000	0.000000	2.000000	1.000000	0.000000
50%	1.000000	1.000000	0.000000	0.000000	3.000000	1.000000	0.000000
75%	2.000000	1.000000	0.000000	1.000000	3.000000	2.000000	0.000000
max	3.000000	1.000000	1.000000	1.000000	4.000000	2.000000	1.000000

8 rows × 25 columns

In [51]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
```

In [52]:
```
# train-test split
X = data['horsepower']
y = data['price']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.3,random_state
=42)

print(X.head())
print(y.head())
```

```
0    111
1    111
2    154
3    102
4    115
Name: horsepower, dtype: int64
0    13495.0
1    16500.0
2    16500.0
3    13950.0
4    17450.0
Name: price, dtype: float64
```

In [ ]:
```

```

In [53]:
```
# As data is in 1D array, need to convert to 2D array for linear regression
X_train = X_train.values.reshape(-1,1)
X_test = X_test.values.reshape(-1,1)
```

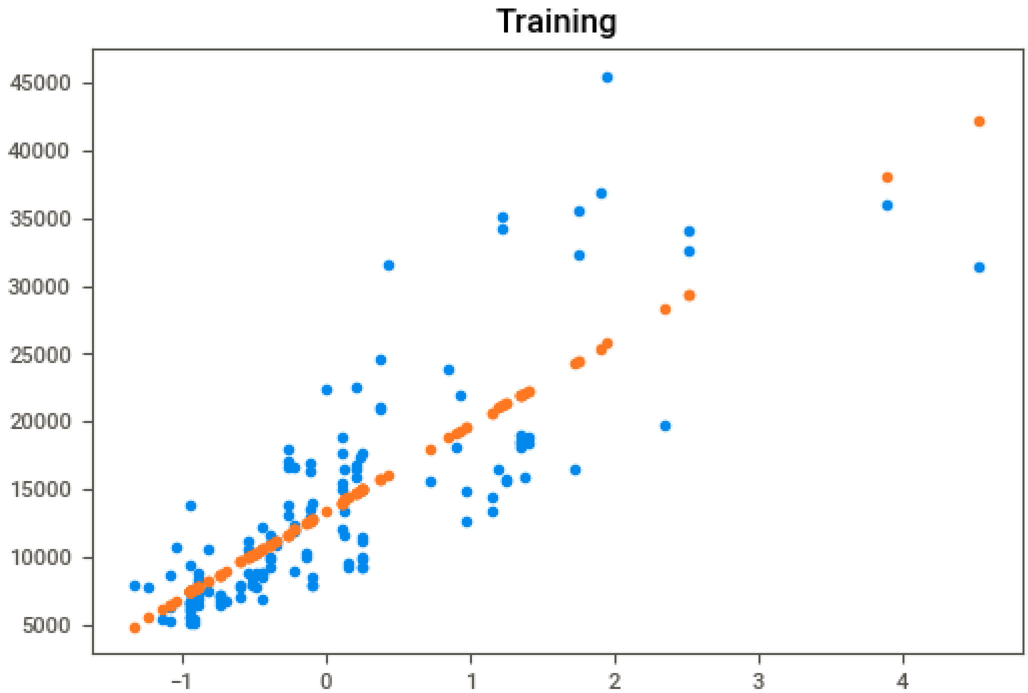
In [54]:
```
# Scaling
# Since Price & Horse Power have large gaps in terms of value, Applying Standa
rd Scaling Scaling to make both of them comparable
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

X_train = ss.fit_transform(X_train)
X_test = ss.fit_transform(X_test)
```

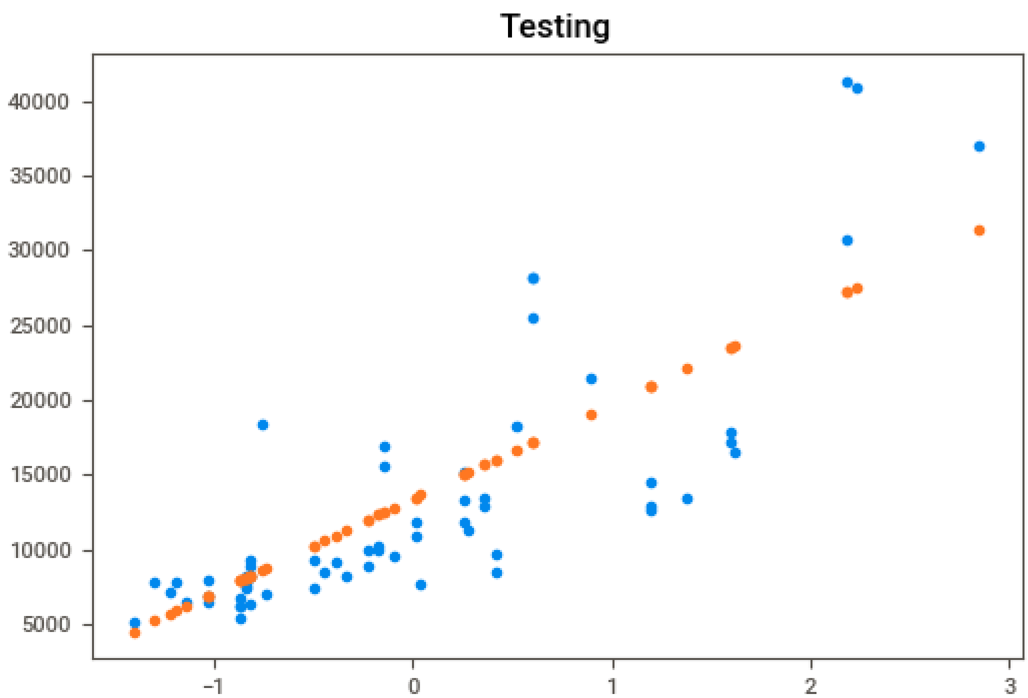
In [55]:
```
# Linear regression fit
lr = LinearRegression()
lr.fit(X_train,y_train)

print('Intercept is',lr.intercept_)
print('Coefficient is',lr.coef_)
```

```
Intercept is 13408.503496503494
Coefficient is [6332.93047557]
```

In [56]:

```
# Predictions
# Blue scatter -> actual-training data
# Orange scatter -> predicted data

# Training
y_train_pred = lr.predict(X_train)
plt.title('Training')
plt.scatter(X_train,y_train)
plt.scatter(X_train,y_train_pred)
plt.show()

# Testing
y_test_pred = lr.predict(X_test)
plt.title('Testing')
plt.scatter(X_test,y_test)
plt.scatter(X_test,y_test_pred)
plt.show()
```


Training
45000
40000
35000
30000
25000
20000
15000
10000
5000
−1
0
1
2
3
4


Testing
40000
35000
30000
25000
20000
15000
10000
5000
−1
0
1
2
3

In [57]:
```
# Let's check how much good fit it is by calculating R-Squared

print('Mean Squared Error for training data is',mean_squared_error(y_train,y_train_pred))
print('R2 Score for training data in LR is',r2_score(y_train,y_train_pred))
print('\n')

print('Mean Squared Error for testing data is',mean_squared_error(y_test,y_test_pred))
print('R2 Score for testing data in LR is',r2_score(y_test,y_test_pred))
```

```
Mean Squared Error for training data is 20843608.761176858
R2 Score for training data in LR is 0.6580190372125265


Mean Squared Error for testing data is 24492365.072824143
R2 Score for testing data in LR is 0.6464951326151096
```

In [58]:
```
##scaling the numeric features
from sklearn.preprocessing import StandardScaler
```

In [59]:
```
from sklearn.feature_selection import VarianceThreshold
```

In [60]:
```
X=data.drop(labels=['price'],axis=1)
```

In [61]:
```
X.head()
```

Out[61]:

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase
0	3	1	0	1	0	2	0	88.6
1	3	1	0	1	0	2	0	88.6
2	1	1	0	1	2	2	0	94.5
3	2	1	0	0	3	1	0	99.8
4	2	1	0	0	3	0	0	99.4

5 rows × 24 columns

In [62]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=101)
X_train.shape,X_test.shape
```

Out[62]:
```
((143, 24), (62, 24))
```

In [63]:
```
from sklearn.feature_selection import mutual_info_regression
```

In [64]:
```
mutual_info=mutual_info_regression(X_train,y_train)
mutual_info
```

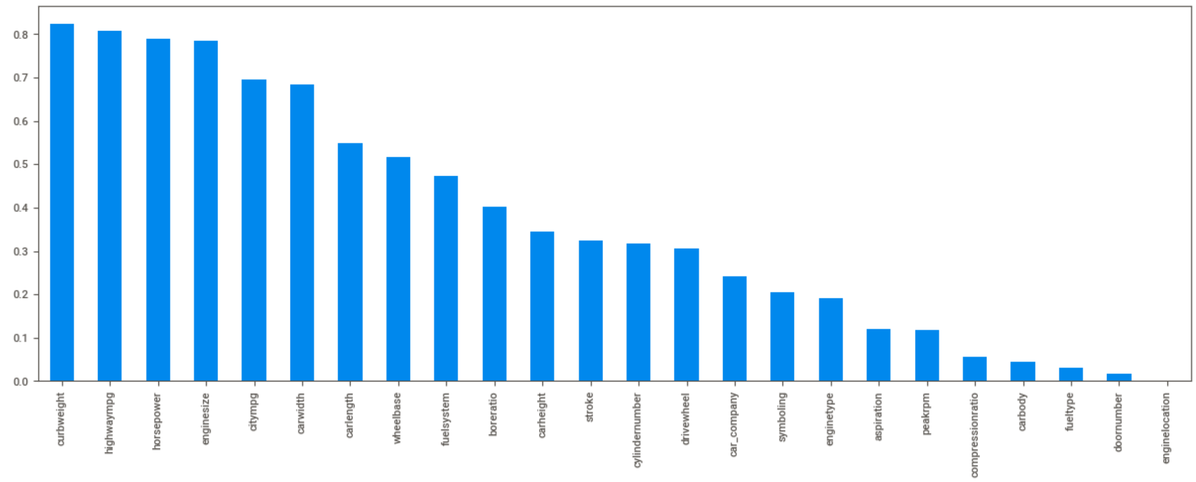
Out[64]:
```
array([0.20465991, 0.03143474, 0.11898343, 0.01667595, 0.04386839,
       0.30628466, 0.        , 0.51739309, 0.54888357, 0.68345924,
       0.34562791, 0.82414561, 0.19232454, 0.31709153, 0.78555772,
       0.47352681, 0.40187809, 0.32469001, 0.05679579, 0.7892827 ,
       0.117621  , 0.69657567, 0.80827938, 0.24179643])
```

In [65]:
```
mutual_info=pd.Series(mutual_info)
mutual_info.index=X_train.columns
mutual_info.sort_values(ascending=False)
```

Out[65]:
```
curbweight          0.824146
highwaympg          0.808279
horsepower          0.789283
enginesize          0.785558
citympg             0.696576
carwidth            0.683459
carlength           0.548884
wheelbase           0.517393
fuelsystem          0.473527
boreratio           0.401878
carheight           0.345628
stroke              0.324690
cylindernumber      0.317092
drivewheel          0.306285
car_company         0.241796
symboling           0.204660
enginetype          0.192325
aspiration          0.118983
peakrpm             0.117621
compressionratio    0.056796
carbody             0.043868
fueltype            0.031435
doornumber          0.016676
enginelocation      0.000000
dtype: float64
```

In [66]:
```
mutual_info.sort_values(ascending=False).plot.bar(figsize=(15,5))
```

Out[66]: <AxesSubplot:>



In [67]:
```
from sklearn.feature_selection import SelectKBest
```

In [68]:
```
##now we select the top 10 imp features
sel_ten_cols = SelectKBest(mutual_info_regression,k=10)
sel_ten_cols.fit(X_train,y_train)
X_train.columns[sel_ten_cols.get_support()]
```

Out[68]:
```
Index(['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize',
       'fuelsystem', 'boreratio', 'horsepower', 'citympg', 'highwaympg'],
      dtype='object')
```

In [69]:
```
X_features=X[['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize'
,
       'fuelsystem', 'boreratio', 'horsepower', 'citympg', 'highwaympg']]
```

In [70]:
```
X_features.head()
```

Out[70]:

	wheelbase	carlength	carwidth	curbweight	enginesize	fuelsystem	boreratio	horsepower	c
0	88.6	168.8	64.1	2548	130	5	3.47	111	
1	88.6	168.8	64.1	2548	130	5	3.47	111	
2	94.5	171.2	65.5	2823	152	5	2.68	154	
3	99.8	176.6	66.2	2337	109	5	3.19	102	
4	99.4	176.6	66.4	2824	136	5	3.19	115	

In [71]:
```
X_train_f,X_test_f,y_train,y_test=train_test_split(X_features,y,test_size=0.3,
random_state=42)
X_train_f.shape,X_test_f.shape
```

Out[71]: ((143, 10), (62, 10))

In [72]:
```
lr.fit(X_train_f,y_train)
y_pred_f=lr.predict(X_test_f)
```

In [73]:
```
print('Mean Squared Error for testing data is',mean_squared_error(y_test,y_pre
d_f))
print('R2 Score for testing data in LR is',r2_score(y_test,y_pred_f))
```

```
Mean Squared Error for testing data is 16258191.516167236
R2 Score for testing data in LR is 0.7653411657570831
```

In [74]:
```
scaler=StandardScaler()
```

In [75]:
```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [76]:
```
from sklearn.decomposition import PCA

pca = PCA()
```

In [77]:
```
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

In [78]:
```
pca.explained_variance_ratio_
```

Out[78]:
```
array([3.18441012e-01, 1.86313722e-01, 9.47407760e-02, 6.41294318e-02,
       5.41606783e-02, 4.67038821e-02, 3.86339669e-02, 3.51085999e-02,
       2.74097884e-02, 2.29736425e-02, 2.09937344e-02, 1.72155330e-02,
       1.48719664e-02, 1.37510741e-02, 1.13929660e-02, 9.11907665e-03,
       7.13410453e-03, 5.97989705e-03, 4.28344212e-03, 2.51628267e-03,
       2.21025445e-03, 1.23221993e-03, 5.06724349e-04, 1.77224814e-04])
```

In [79]:
```
pca = PCA(n_components=5)

X_train_5 = pca.fit_transform(X_train_scaled)
X_test_5 = pca.transform(X_test_scaled)
```

In [80]:
```
lr.fit(X_train_5,y_train)
```

Out[80]:
```
LinearRegression()
```

In [81]:
```
y_pred = lr.predict(X_test_5)
```

In [82]:
```
print('Mean Squared Error for testing data is',mean_squared_error(y_test,y_pre
d))
print('R2 Score for testing data in LR is',r2_score(y_test,y_pred))
```

```
Mean Squared Error for testing data is 66749275.64663582
R2 Score for testing data in LR is 0.03658982032392677
```

In [ ]:
```

```

In [83]:
```
from sklearn.linear_model import SGDRegressor
```

In [84]:
```
sgd_regressor=SGDRegressor(alpha=0.0005,learning_rate='optimal',eta0=0.001)
```

In [85]:
```
sgd_regressor.fit(X_train_f,y_train)
```

Out[85]:
```
SGDRegressor(alpha=0.0005, eta0=0.001, learning_rate='optimal')
```

In [86]:
```
sgd_pred = sgd_regressor.predict(X_test_f)
```

In [87]:
```
mean_squared_error(sgd_pred,y_test)
```

Out[87]:
```
6.091051072290543e+34
```

In [88]:
```
r2_score(sgd_pred,y_test)
```

Out[88]:
```
-18.844770554825438
```

In [ ]:

In [89]:
```
X_feature_scaled=scaler.fit_transform(X_features)
```

In [90]:
```
X_feature_scaled=pd.DataFrame(X_feature_scaled)
```

In [91]:
```
X_feature_scaled.head()
```

Out[91]:

	0	1	2	3	4	5	6	7	
0	-1.690772	-0.426521	-0.844782	-0.014566	0.074449	0.869568	0.519071	0.174483	-0.64655
1	-1.690772	-0.426521	-0.844782	-0.014566	0.074449	0.869568	0.519071	0.174483	-0.64655
2	-0.708596	-0.231513	-0.190566	0.514882	0.604046	0.869568	-2.404880	1.264536	-0.95301
3	0.173698	0.207256	0.136542	-0.420797	-0.431076	0.869568	-0.517266	-0.053668	-0.18686
4	0.107110	0.207256	0.230001	0.516807	0.218885	0.869568	-0.517266	0.275883	-1.10624

In [92]:
```
X_train_s,X_test_s,y_train,y_test=train_test_split(X_feature_scaled,y,test_siz
e=0.3,random_state=1)
X_train_s.shape,X_test_s.shape
```

Out[92]:
```
((143, 10), (62, 10))
```

In [93]:
```
SGDRegressor().fit(X_train_s,y_train)
```

Out[93]:
```
SGDRegressor()
```

In [94]:
```
pred=sgd_regressor.predict(X_test_s)
```

In [95]:
```
print('Mean Squared Error for testing data is',mean_squared_error(y_test,pred
))
print('R2 Score for testing data in sgdregressor is',r2_score(y_test,pred))
```

```
Mean Squared Error for testing data is 6.047272323553665e+28
R2 Score for testing data in sgdregressor is -1.0016912754459494e+21
```

In [96]:
```
lr.fit(X_train_s,y_train)
```

Out[96]:
```
LinearRegression()
```

In [97]:
```
predicts=lr.predict(X_test_s)
```

In [98]:
```
print('Mean Squared Error for testing data is',mean_squared_error(y_test,predi
cts))
print('R2 Score for testing data in LR is',r2_score(y_test,predicts))
```

```
Mean Squared Error for testing data is 11511898.099945687
R2 Score for testing data in LR is 0.8093129054958448
```

In [ ]:

In [99]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn import svm
```

In [100]:
```
models = []
models.append(('DTR', DecisionTreeRegressor()))
models.append(('svr', svm.SVR()))
models.append(('RFr', RandomForestRegressor()))
```

In [101]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

In [102]:
```
results = []
for name,model in models:
    kfold = KFold(n_splits=10,random_state=1,shuffle=True)
    cvresults = cross_val_score(model,X_train_s,y_train,cv=kfold)
    results.append(cvresults)
    output = "%s: %f(%f)" % (name,cvresults.mean(),cvresults.std())
    print(output)
```

```
DTR: 0.804075(0.079758)
svr: -0.204919(0.223145)
RFr: 0.882346(0.069454)
```

In [103]:
```
rfr=RandomForestRegressor(n_estimators=50,max_depth=5,min_samples_leaf=5,min_s
amples_split=4,n_jobs=1,random_state=1)
```

In [104]:
```
rfr.fit(X_train_s,y_train)
```

Out[104]:
```
RandomForestRegressor(max_depth=5, min_samples_leaf=5, min_samples_split=4,
                      n_estimators=50, n_jobs=1, random_state=1)
```

In [105]:
```
rfr_pred=rfr.predict(X_test_s)
```

In [106]:
```
print('Mean Squared Error for testing data is',mean_squared_error(y_test,rfr_p
red))
print('R2 Score for testing data in random forest regressor is',r2_score(y_tes
t,rfr_pred))
```

```
Mean Squared Error for testing data is 6103456.741882865
R2 Score for testing data in random forest regressor is 0.8989002141578258
```

In [ ]:
```

```

In [107]:
```
X_features.head()
```

Out[107]:

	wheelbase	carlength	carwidth	curbweight	enginesize	fuelsystem	boreratio	horsepower	c
0	88.6	168.8	64.1	2548	130	5	3.47	111	
1	88.6	168.8	64.1	2548	130	5	3.47	111	
2	94.5	171.2	65.5	2823	152	5	2.68	154	
3	99.8	176.6	66.2	2337	109	5	3.19	102	
4	99.4	176.6	66.4	2824	136	5	3.19	115	

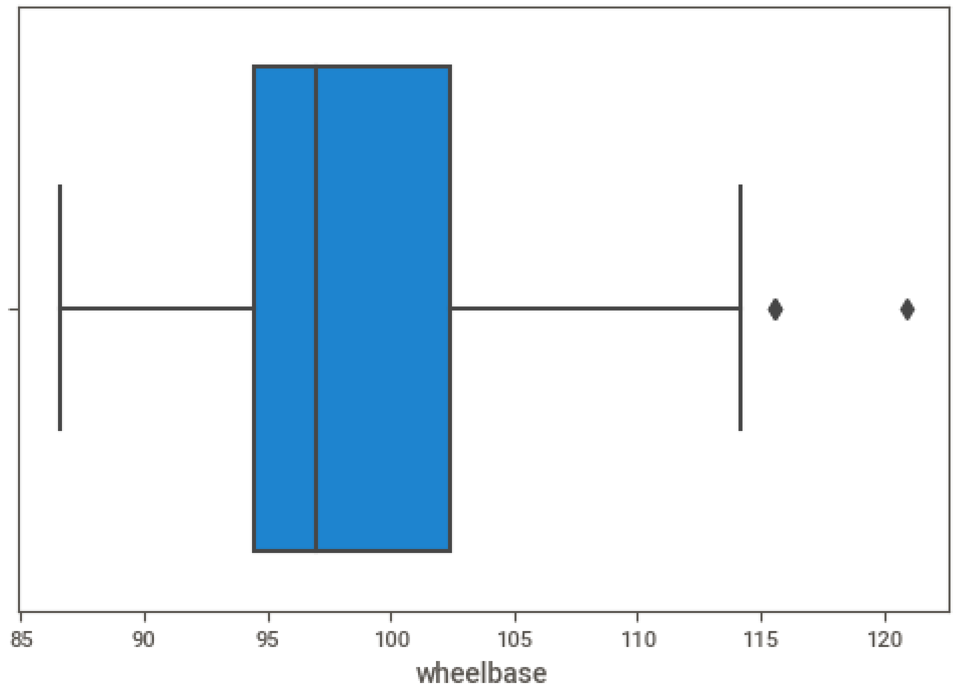
In [108]:
```
X_features=X_features.drop('highwaympg',axis=1)
```

In [109]:
```
X_features.head()
```

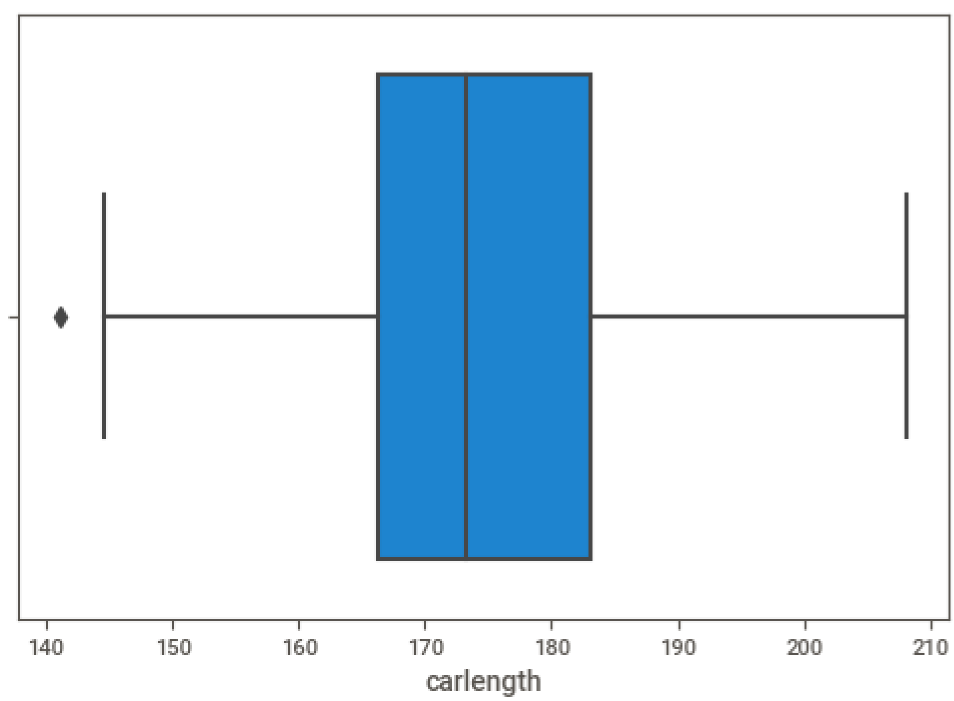
Out[109]:

	wheelbase	carlength	carwidth	curbweight	enginesize	fuelsystem	boreratio	horsepower	c
0	88.6	168.8	64.1	2548	130	5	3.47	111	
1	88.6	168.8	64.1	2548	130	5	3.47	111	
2	94.5	171.2	65.5	2823	152	5	2.68	154	
3	99.8	176.6	66.2	2337	109	5	3.19	102	
4	99.4	176.6	66.4	2824	136	5	3.19	115	

In [110]:
```
sns.boxplot('wheelbase',data=X_features,orient='h')
```
Out[110]:
```
<AxesSubplot:xlabel='wheelbase'>
```

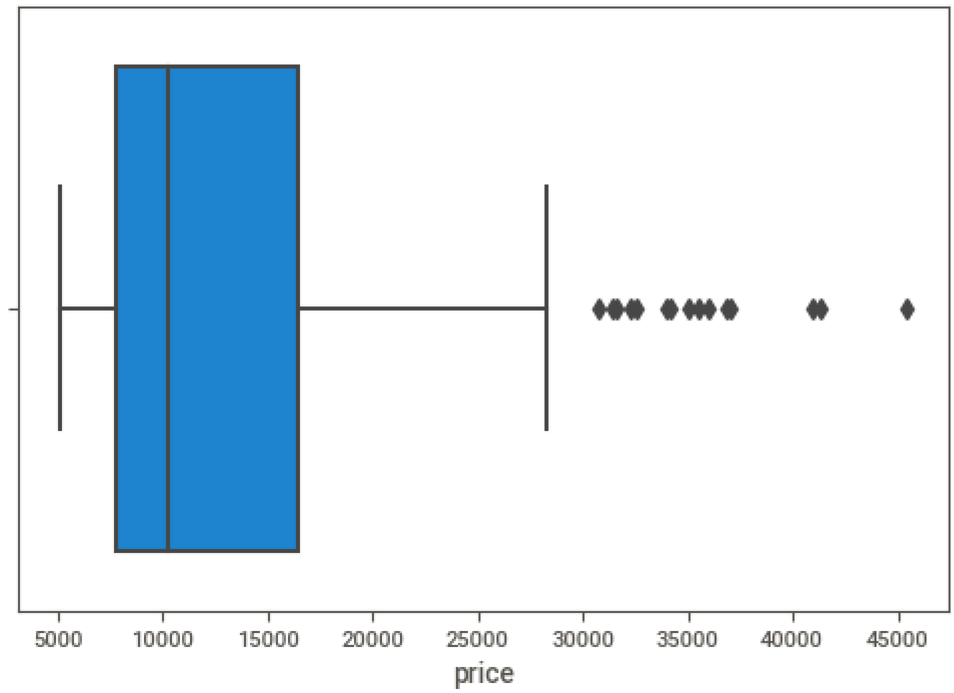


In [111]:
```
sns.boxplot('carlength',data=X_features)
```
Out[111]:
```
<AxesSubplot:xlabel='carlength'>
```



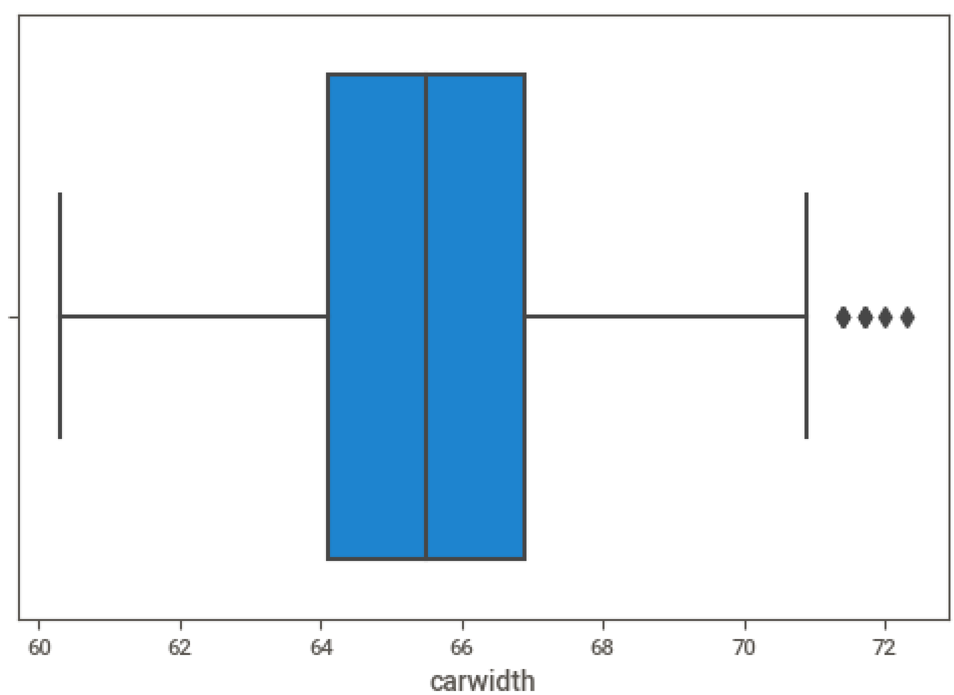
In [112]:
```
sns.boxplot(y)
```

Out[112]: <AxesSubplot:xlabel='price'>



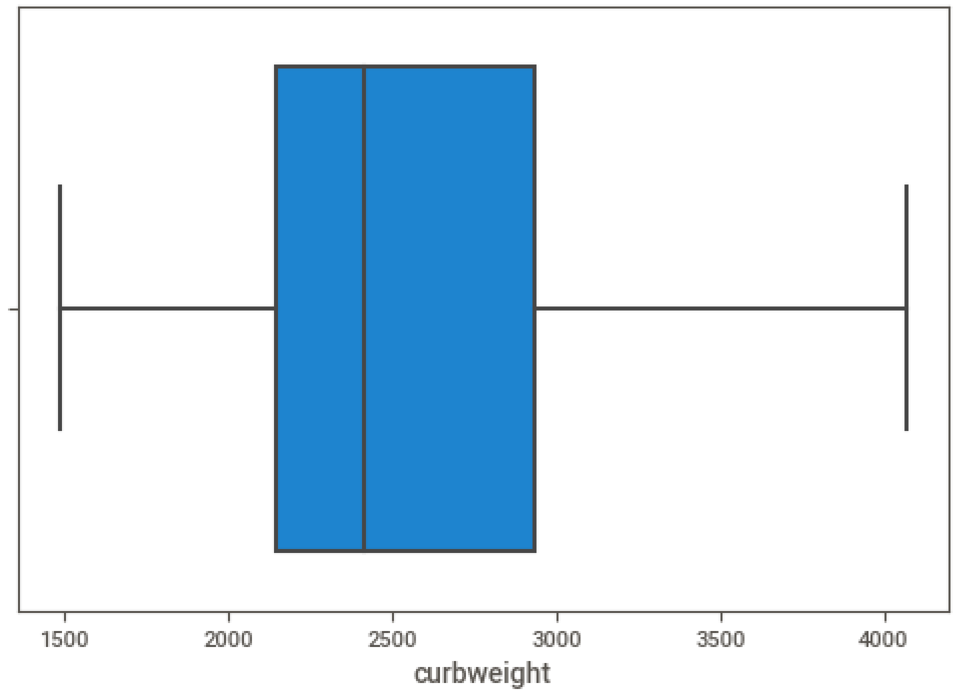
In [113]:
```
sns.boxplot('carwidth',data=X_features)
```

Out[113]: <AxesSubplot:xlabel='carwidth'>



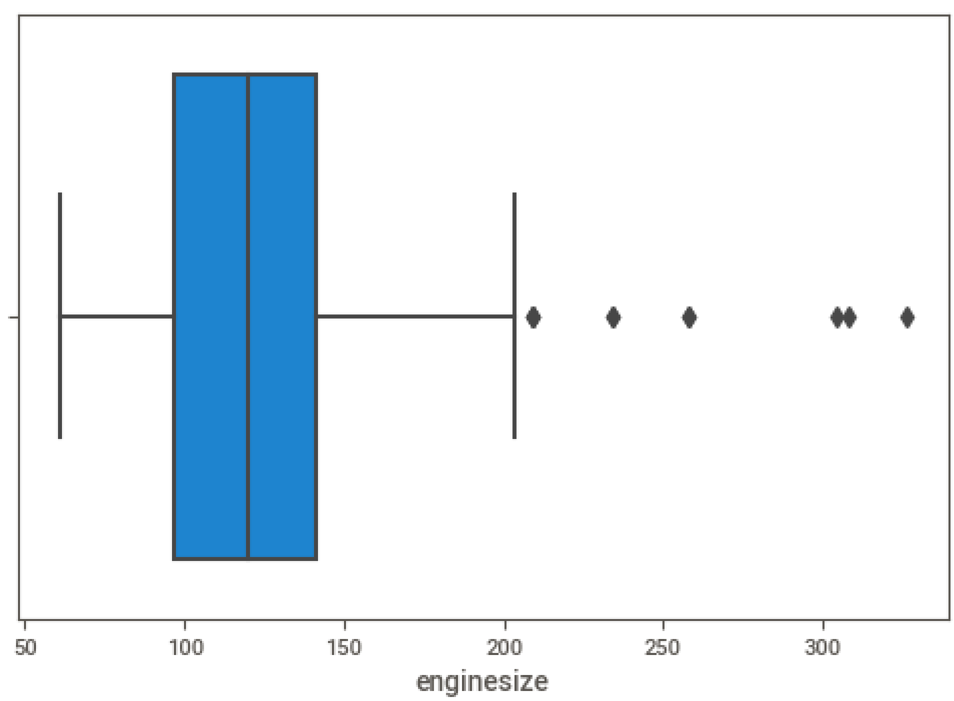
In [114]:
```
sns.boxplot('curbweight',data=X_features)
```

Out[114]: <AxesSubplot:xlabel='curbweight'>



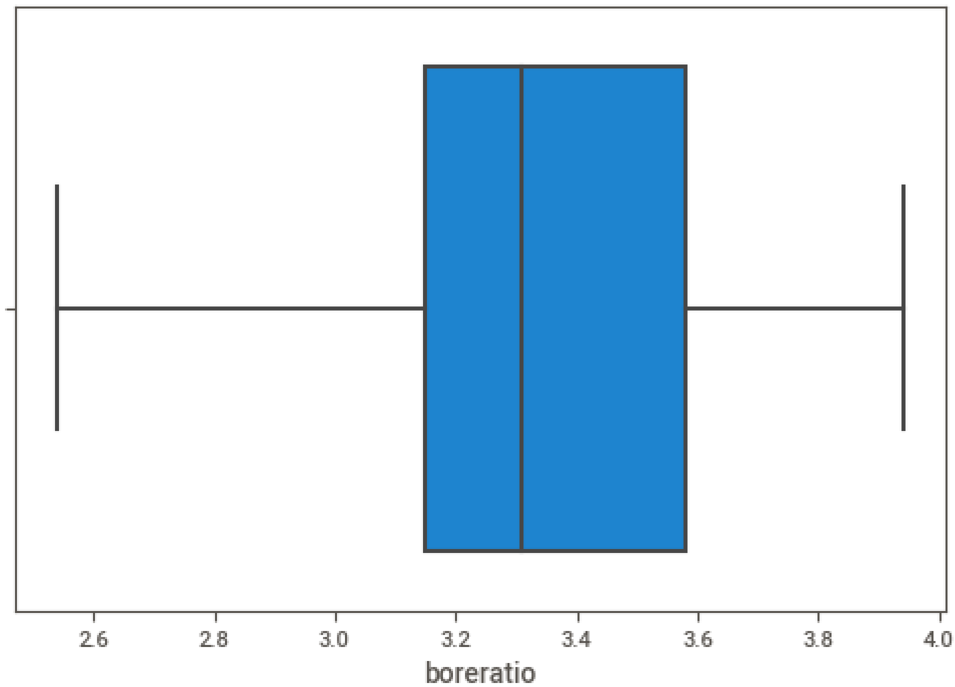
In [115]:
```
sns.boxplot('enginesize',data=X_features)
```

Out[115]: <AxesSubplot:xlabel='enginesize'>



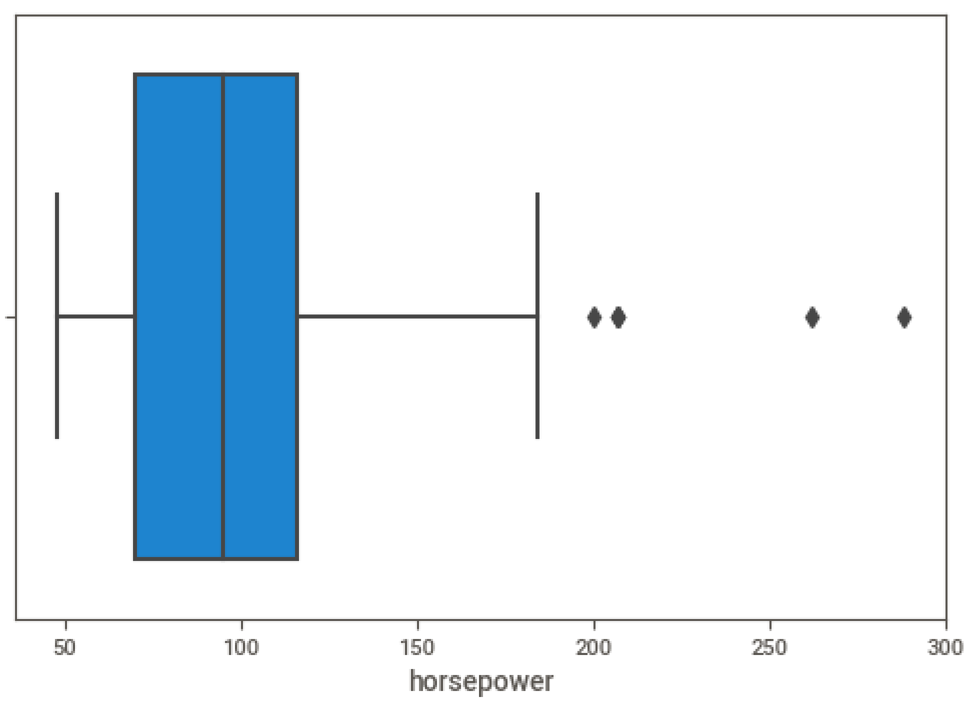
In [116]:
```
sns.boxplot('boreratio',data=X_features)
```

Out[116]: <AxesSubplot:xlabel='boreratio'>



In [117]:
```
sns.boxplot('horsepower',data=X_features)
```

Out[117]: <AxesSubplot:xlabel='horsepower'>



In [118]:
```
sns.boxplot('citympg',data=X_features)
```

Out[118]:
```
<AxesSubplot:xlabel='citympg'>
```