

In []:

```
#konaparthi niharika
#19bcs6174
```

In [1]:

```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

In [2]:

```
car_data = pd.read_csv("E:\sem 4\machine learning\lab\Carprice_Assignment.csv")
car_data.head()
```

Out[2]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpf
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpf
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpf
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpf
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpf

5 rows × 26 columns



In [3]:

```
car_data.columns
```

Out[3]:

```
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')
```

In [4]:

```
car_data.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

```

In [5]:

```
car_data.describe()
```

Out[5]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compress
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.



In [6]:

```
car_data.shape
```

Out[6]:

```
(205, 26)
```

In [7]:

```

#Split the company name fron carname
CompanyName = car_data['CarName'].apply(lambda x : x.split(' ')[0])
car_data.insert(3,"CompanyName",CompanyName)
car_data.drop(['CarName'],axis=1,inplace=True)
car_data.head()

```

Out[7]:

	car_ID	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	engineLocation	wheelbase	...	enginesize	fuelsys
0	1	3	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	r
1	2	3	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	r
2	3	1	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	r
3	4	2	audi	gas	std	four	sedan	fwd	front	99.8	...	109	r
4	5	2	audi	gas	std	four	sedan	4wd	front	99.4	...	136	r

5 rows × 26 columns



In [8]:

```
car_data['total_mpg']=(55*car_data['citympg']/100)+(45*car_data['highwaympg']/100)
car_data.drop(['car_ID','citympg','highwaympg'],axis=1,inplace=True)
car_data.symboling=car_data.symboling.astype(str)
car_data.head()
```

Out[8]:

	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	engineLocation	wheelbase	carlength	...	cylindernumber
0	3	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	...	four
1	3	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	...	four
2	1	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	...	six
3	2	audi	gas	std	four	sedan	fwd	front	99.8	176.6	...	four
4	2	audi	gas	std	four	sedan	4wd	front	99.4	176.6	...	five

5 rows × 24 columns



In [9]:

```
#to cross check whether the columns have been dropped or not!
car_data.shape
```

Out[9]:

(205, 24)

In [11]:

```
#check all the companies present in the data
car_data.CompanyName.unique()
```

Out[11]:

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

In [13]:

```
car_data.CompanyName.replace('maxda','mazda',inplace=True)
car_data.CompanyName.replace('Nissan','nissan',inplace=True)
car_data.CompanyName.replace('porcshce','porsche',inplace=True)
car_data.CompanyName.replace('toyouta','toyota',inplace=True)
car_data.CompanyName.replace('vokswagen','volkswagen',inplace=True)
car_data.CompanyName.replace('vw','volkswagen',inplace=True)
```

In [14]:

```
#check whether the company names are fixed or not!
car_data.CompanyName.unique()
```

Out[14]:

```
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 [24]:

```
#shows price distribution of a car
```

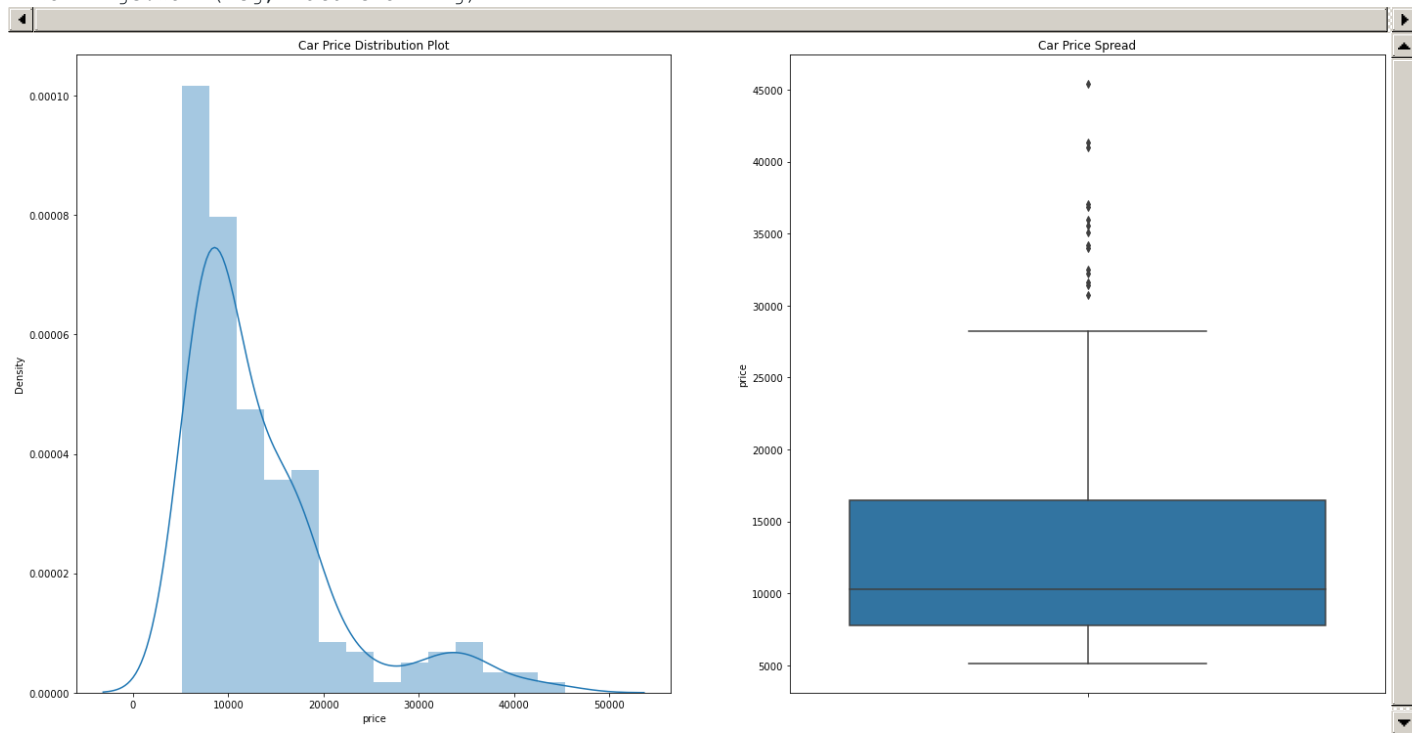
```
plt.figure(figsize=(24,12))

plt.subplot(1,2,1)
plt.title('Car Price Distribution Plot')
sns.distplot(car_data.price)

plt.subplot(1,2,2)
plt.title('Car Price Spread')
sns.boxplot(y=car_data.price)

plt.show()
```

E:\Anaconda\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
warnings.warn(msg, FutureWarning)



In [23]:

```
print(car_data.price.describe(percentiles = [0.35,0.60,0.85,0.95,1]))
```

```
count      205.000000
mean       13276.710571
std        7988.852332
min         5118.000000
35%        8522.600000
50%        10295.000000
60%        12515.600000
85%        18500.000000
95%        32472.400000
100%       45400.000000
max        45400.000000
Name: price, dtype: float64
```

In [26]:

```
#visualizing categorical data
plt.figure(figsize=(25, 6))

plt.subplot(1,3,1)
plt1 = car_data.CompanyName.value_counts().plot(kind='bar')
plt.title('Companies Histogram')
plt1.set(xlabel = 'Car company', ylabel='Frequency of company')

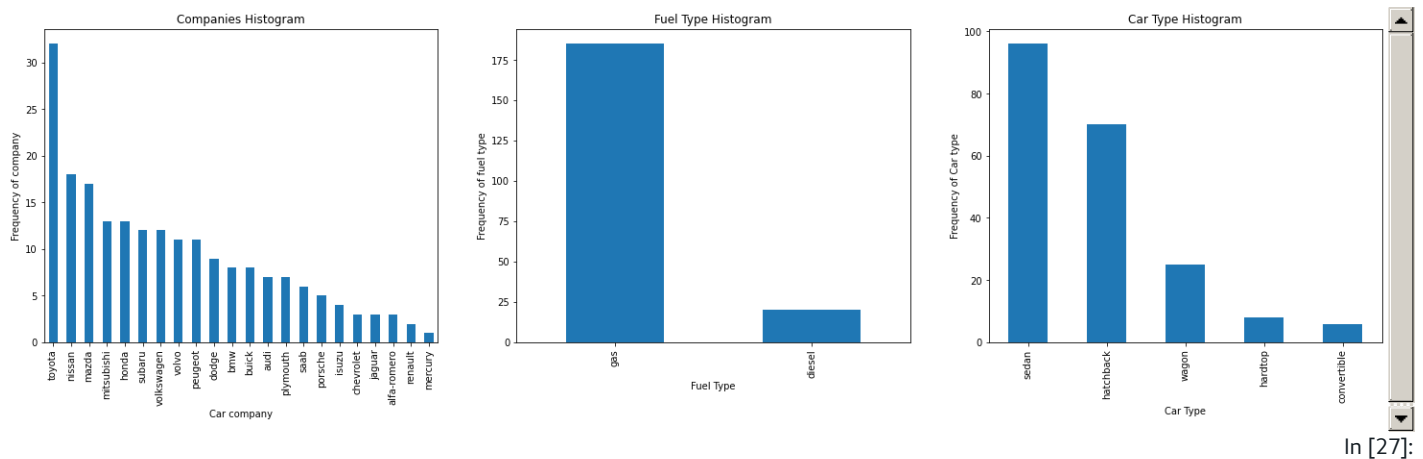
plt.subplot(1,3,2)
plt1 = car_data.fueltype.value_counts().plot(kind='bar')
plt.title('Fuel Type Histogram')
plt1.set(xlabel = 'Fuel Type', ylabel='Frequency of fuel type')

plt.subplot(1,3,3)
plt1 = car_data.carbody.value_counts().plot(kind='bar')
plt.title('Car Type Histogram')
```

```
plt1.set(xlabel = 'Car Type', ylabel='Frequency of Car type')
```

```
plt.show()
```

```
#from the plot we got to know that toyota produces more number of cars.  
#gas cars are more used than diesel cars  
#sedan is more preferred car
```



In [27]:

```
plt.figure(figsize=(20,8))
```

```
plt.subplot(1,2,1)
```

```
plt.title('Symboling Histogram')
```

```
sns.countplot(car_data.symboling, palette=("cubehelix"))
```

```
plt.subplot(1,2,2)
```

```
plt.title('Symboling vs Price')
```

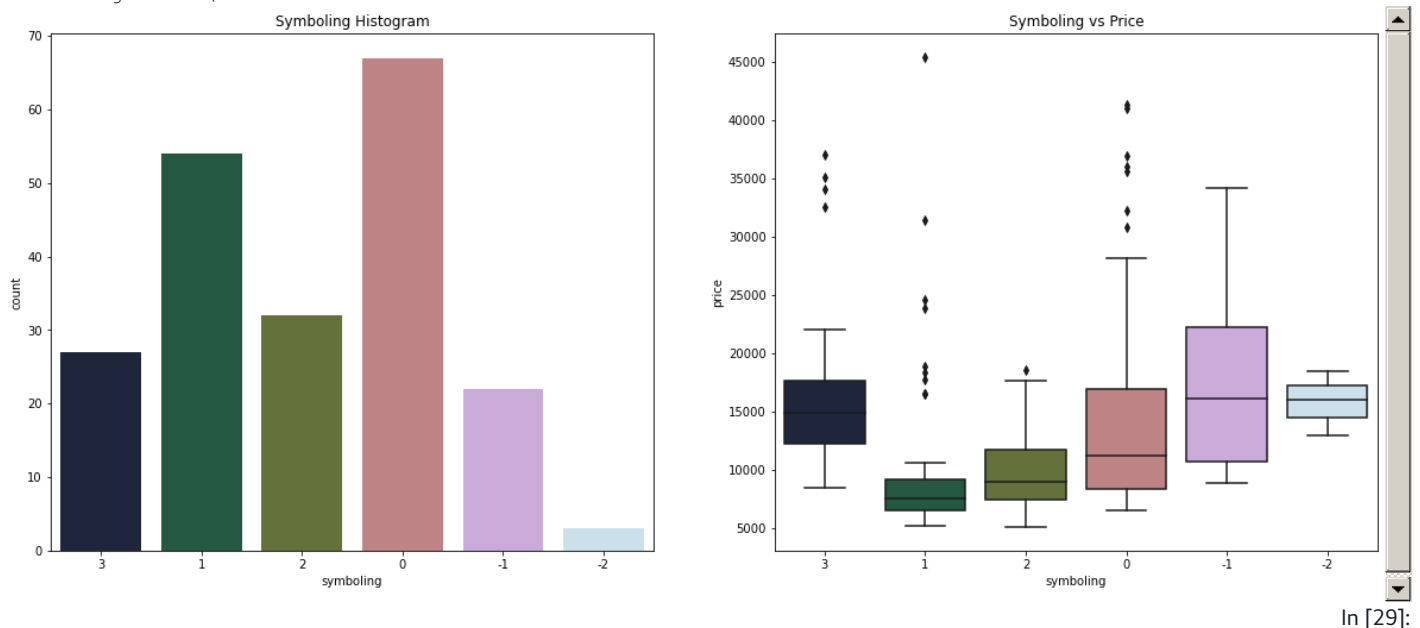
```
sns.boxplot(x=car_data.symboling, y=car_data.price, palette=("cubehelix"))
```

```
plt.show()
```

```
#from the plot, we get to know that  
#symboling having values 0 and 1 more cars are sold  
#symboling having -1 are high priced.
```

E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



In [29]:

```
plt.figure(figsize=(20,8))
```

```
plt.subplot(1,2,1)
```

```
plt.title('Engine Type Histogram')
```

```
sns.countplot(car_data.engine_type, palette=("Blues_d"))
```

```
plt.subplot(1,2,2)
```

```
plt.title('Engine Type vs Price')
```

```
sns.boxplot(x=car_data.enginetype, y=car_data.price, palette=("PuBuGn"))
```

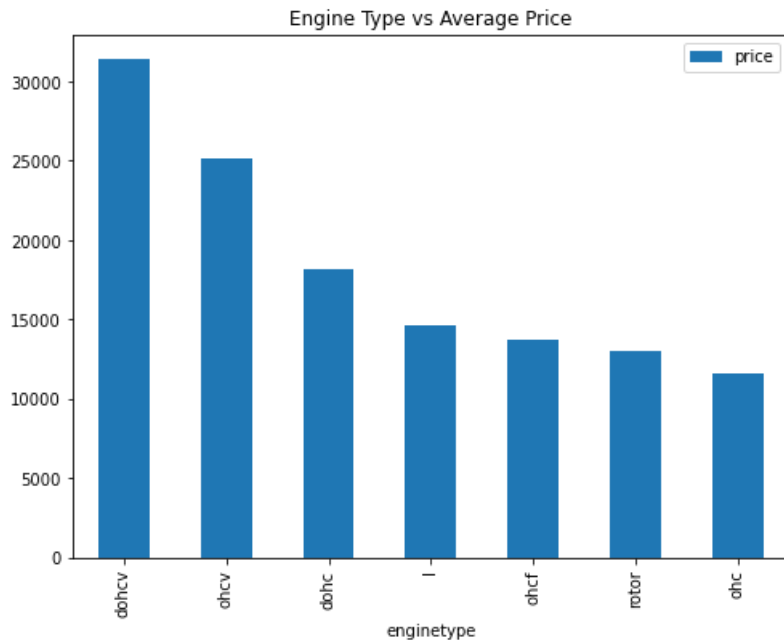
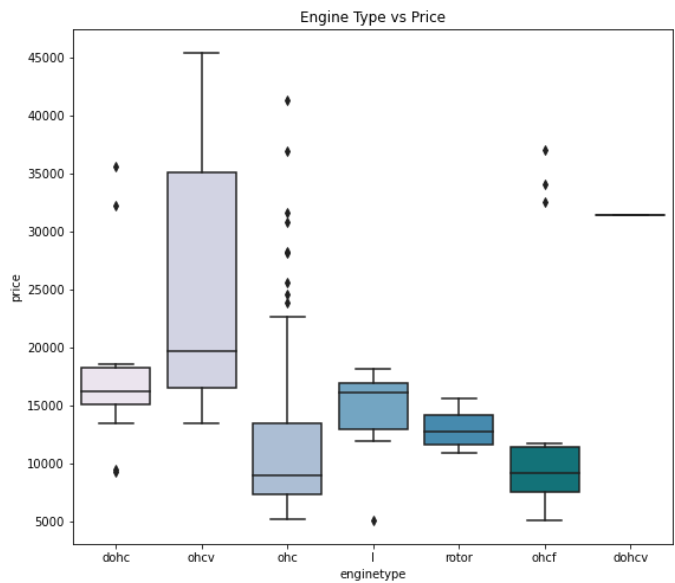
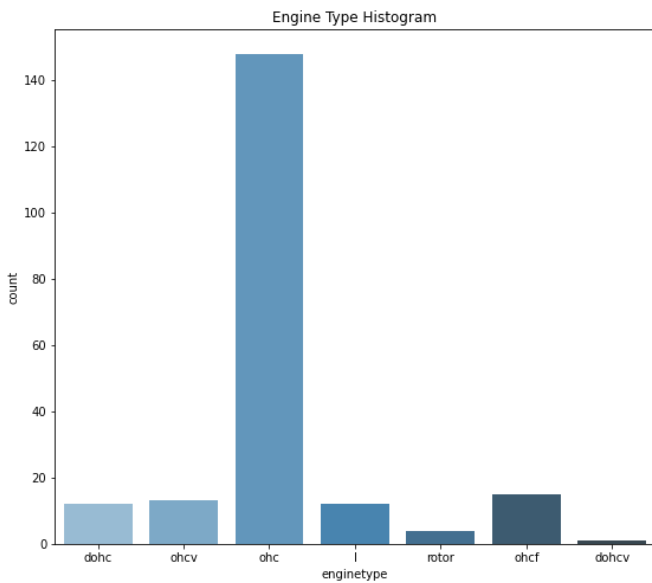
```
plt.show()
```

```
df = pd.DataFrame(car_data.groupby(['enginetype'])['price'].mean().sort_values(ascending = False))
df.plot.bar(figsize=(8,6))
plt.title('Engine Type vs Average Price')
plt.show()
```

*#from the plot,we can get that  
#ohc engine cars are highly sold  
#ohcv engine is highly priced.  
#dohcv is very expensive.*

E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn()
```



In [31]:

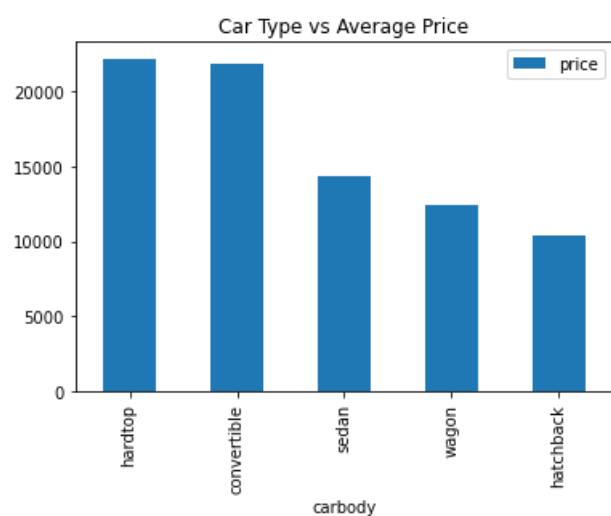
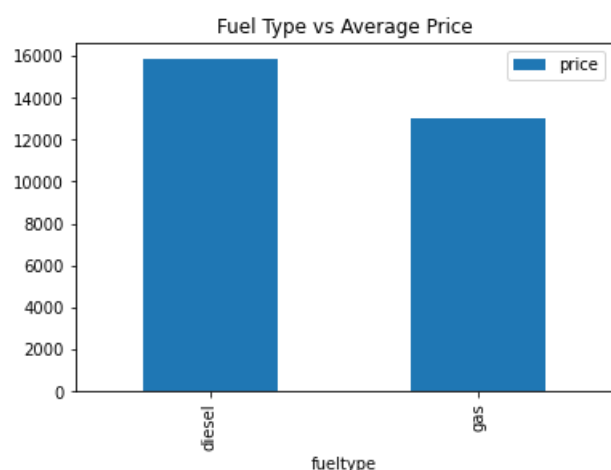
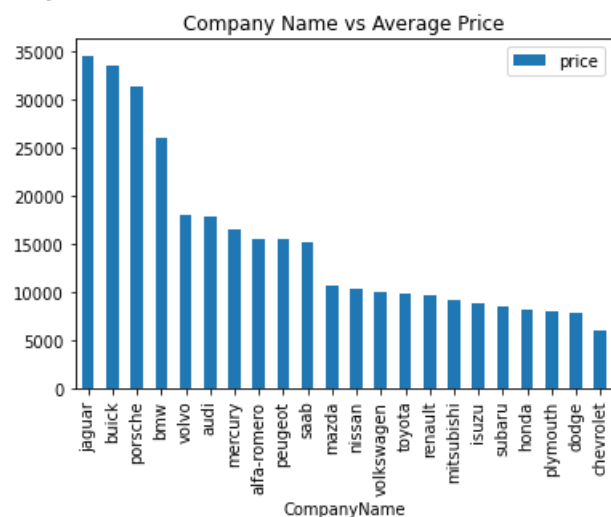
```
plt.figure(figsize=(25, 6))
```

```
df = pd.DataFrame(car_data.groupby(['CompanyName'])['price'].mean().sort_values(ascending = False))
df.plot.bar()
plt.title('Company Name vs Average Price')
plt.show()
```

```
df = pd.DataFrame(car_data.groupby(['fueltype'])['price'].mean().sort_values(ascending = False))
df.plot.bar()
plt.title('Fuel Type vs Average Price')
plt.show()
```

```
df = pd.DataFrame(car_data.groupby(['carbody'])['price'].mean().sort_values(ascending = False))
df.plot.bar()
plt.title('Car Type vs Average Price')
plt.show()
#from the plot we get that,
#the average price of jaguar is high and chevrolet is low
#diesel cars have high average price compared to gas cars
#hardtop has high average price compared to hatch back.
```

<Figure size 1800x432 with 0 Axes>



```
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
plt.title('Door Number Histogram')
sns.countplot(car_data.doornumber, palette=("plasma"))

plt.subplot(1,2,2)
```

In [33]:

```
plt.title('Door Number vs Price')
sns.boxplot(x=car_data.doornumber, y=car_data.price, palette=("plasma"))

plt.show()

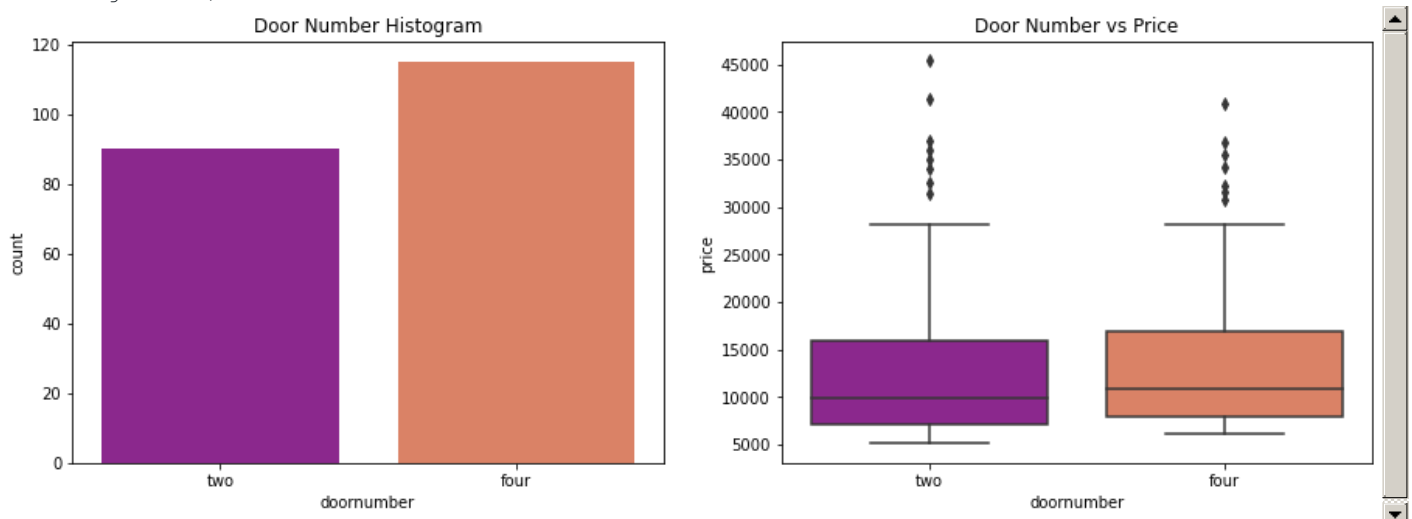
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
plt.title('Aspiration Histogram')
sns.countplot(car_data.aspiration, palette=("plasma"))

plt.subplot(1,2,2)
plt.title('Aspiration vs Price')
sns.boxplot(x=car_data.aspiration, y=car_data.price, palette=("plasma"))
plt.show()
#from the plot we can conclude that
#doornumber variable is not affecting the price much. There is no significant difference between the cat
#turbo has high price range
```

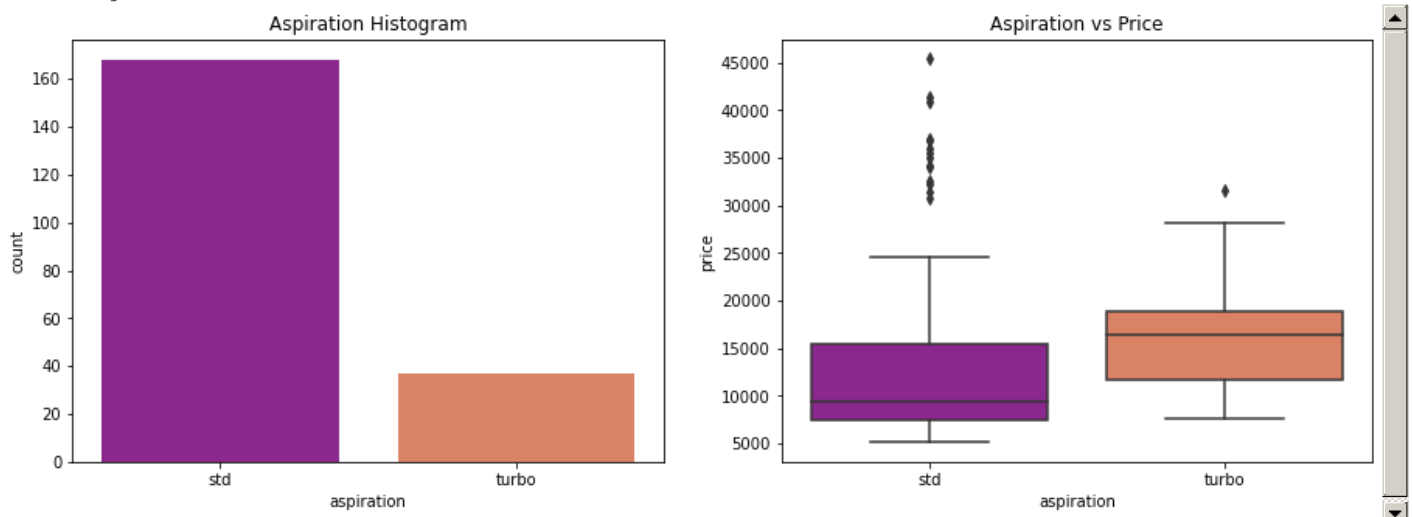
E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



In [34]:

```
def plot_count(x,fig):
    plt.subplot(4,2,fig)
    plt.title(x+' Histogram')
    sns.countplot(car_data[x],palette=("magma"))
    plt.subplot(4,2,(fig+1))
    plt.title(x+' vs Price')
    sns.boxplot(x=car_data[x], y=car_data.price, palette=("magma"))

plt.figure(figsize=(15,20))
```



```

plot_count('enginelocation', 1)
plot_count('cylindernumber', 3)
plot_count('fuelsystem', 5)
plot_count('drivewheel', 7)

```

```
plt.tight_layout()
```

```

#from the plot we can say that
#front engine located cars are in high number
#rear engine located cars are high priced
#cylinder no 4 has high count
#cylinder no 8 is high priced
#Most high ranged cars seeme to prefer rwd drivewheel.

```

E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn()
```

E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

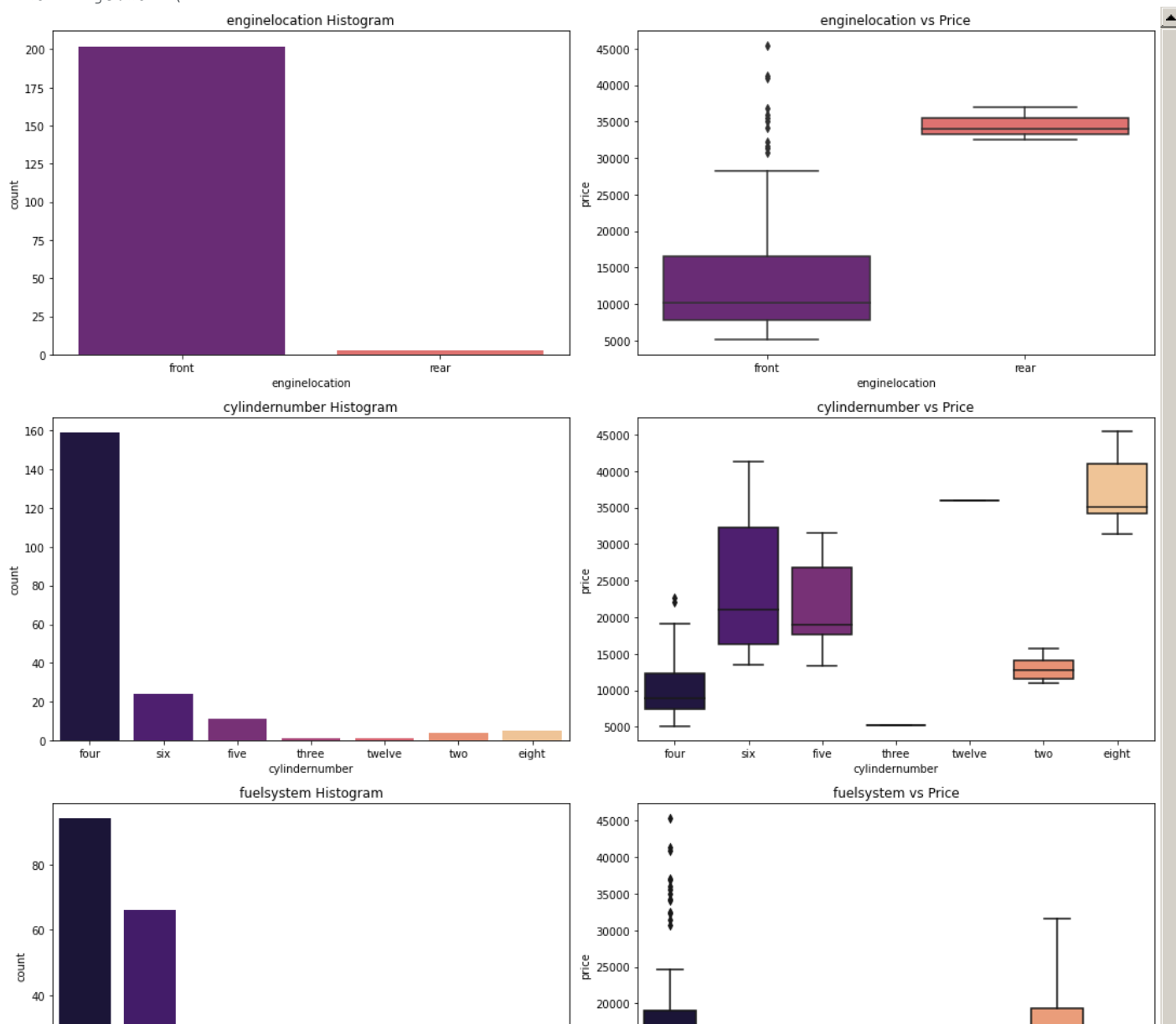
```
warnings.warn()
```

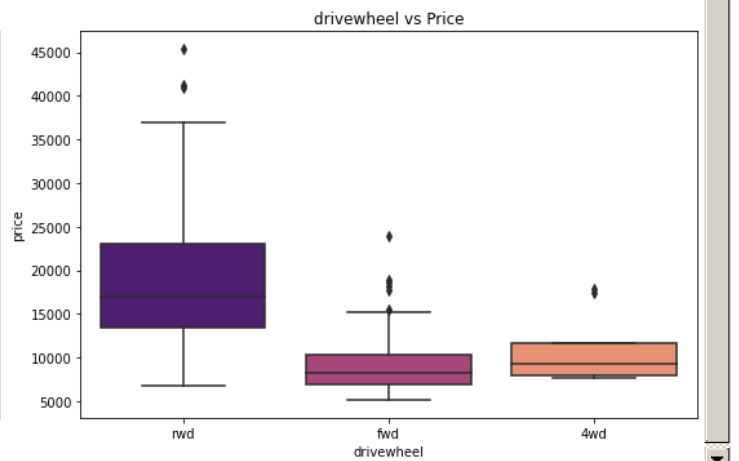
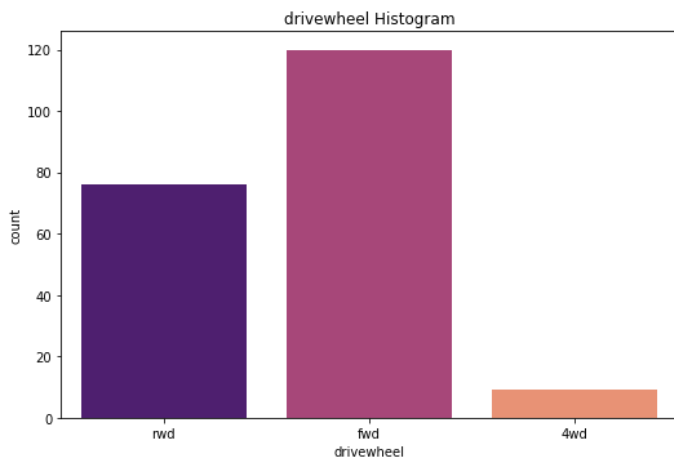
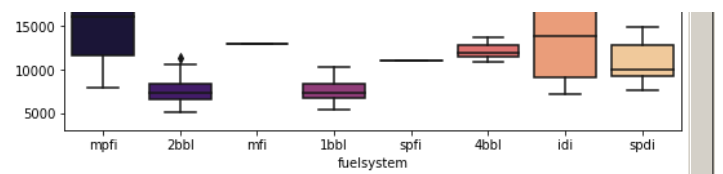
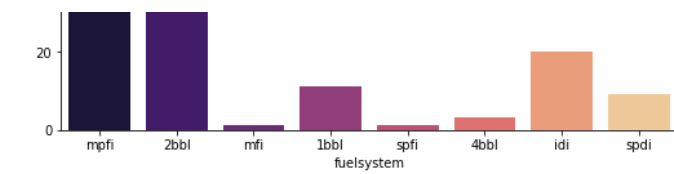
E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn()
```

E:\Anaconda\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn()
```





In [37]:

```
#visualizing numerical data
```

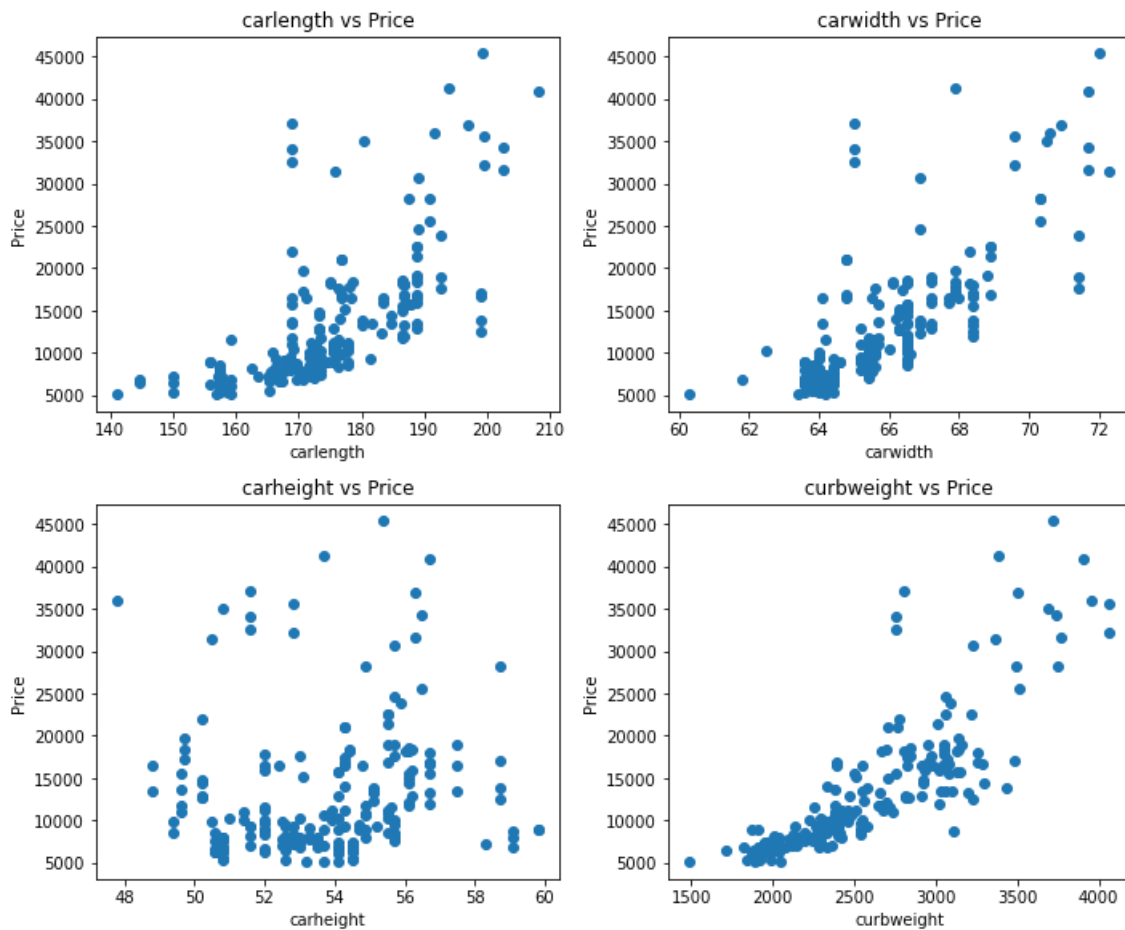
```
def scatter(x,fig):
    plt.subplot(5,2,fig)
    plt.scatter(car_data[x],car_data['price'])
    plt.title(x+' vs Price')
    plt.ylabel('Price')
    plt.xlabel(x)
```

```
plt.figure(figsize=(10,20))
```

```
scatter('carlength', 1)
scatter('carwidth', 2)
scatter('carheight', 3)
scatter('curbweight', 4)
```

```
plt.tight_layout()
```

```
#carwidth, carlength and curbweight seems to have a poitive correlation with price.
#carheight doesn't show any significant trend with price.
```



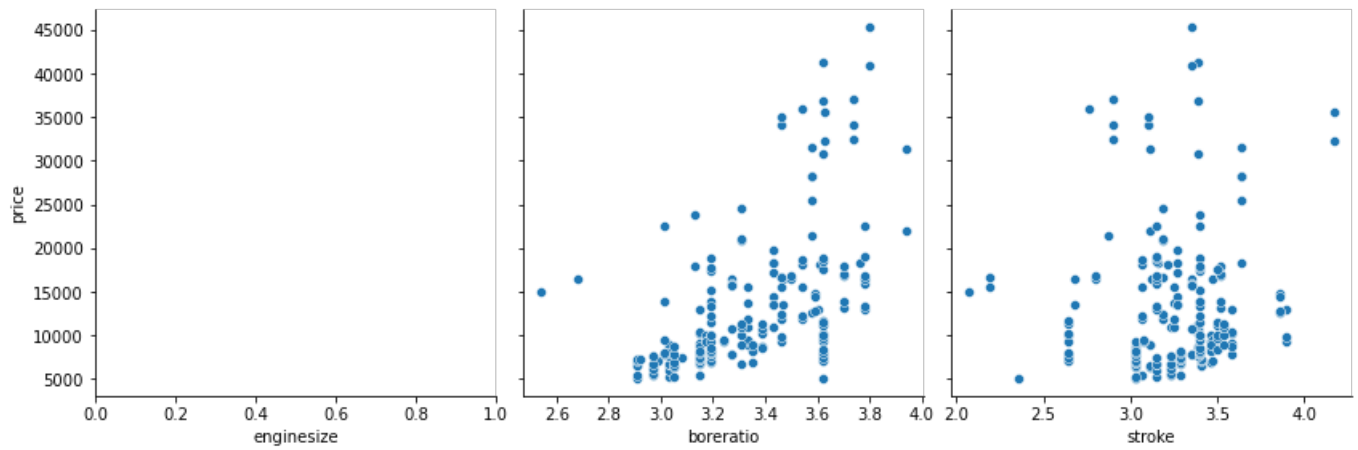
In [39]:

```
def pp(x,y,z):
    sns.pairplot(car_data, x_vars=[x,y,z], y_vars='price',size=4, aspect=1, kind='scatter')
    plt.show()

pp('enginesize', 'boreratio', 'stroke')
pp('compressionratio', 'horsepower', 'peakrpm')
#
```

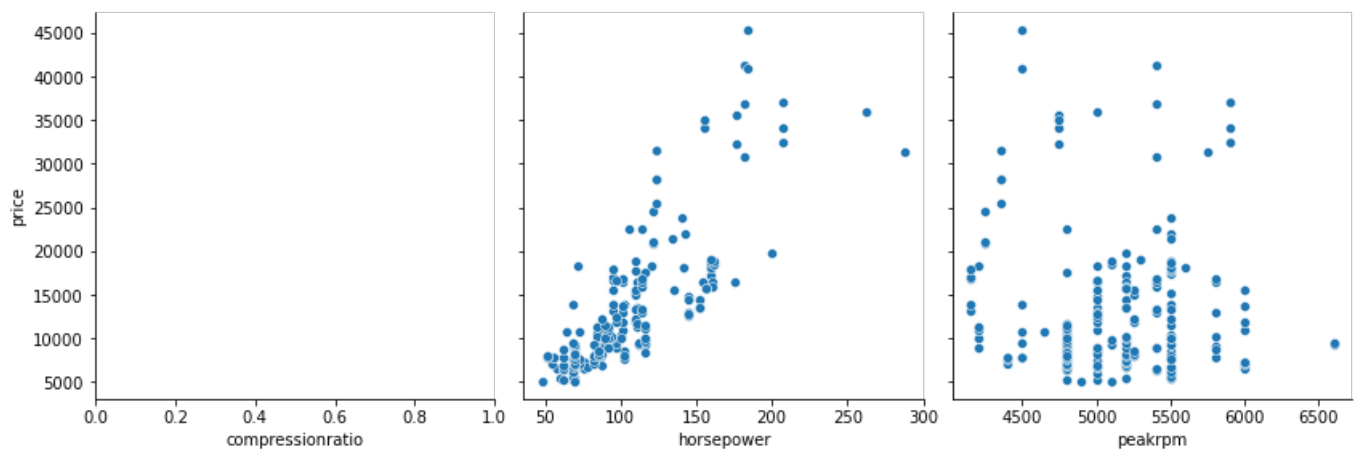
E:\Anaconda\lib\site-packages\seaborn\axisgrid.py:1912: UserWarning: The `size` parameter has been renamed to `height`; please update your code.

```
warnings.warn(msg, UserWarning)
```



E:\Anaconda\lib\site-packages\seaborn\axisgrid.py:1912: UserWarning: The `size` parameter has been renamed to `height`; please update your code.

```
warnings.warn(msg, UserWarning)
```



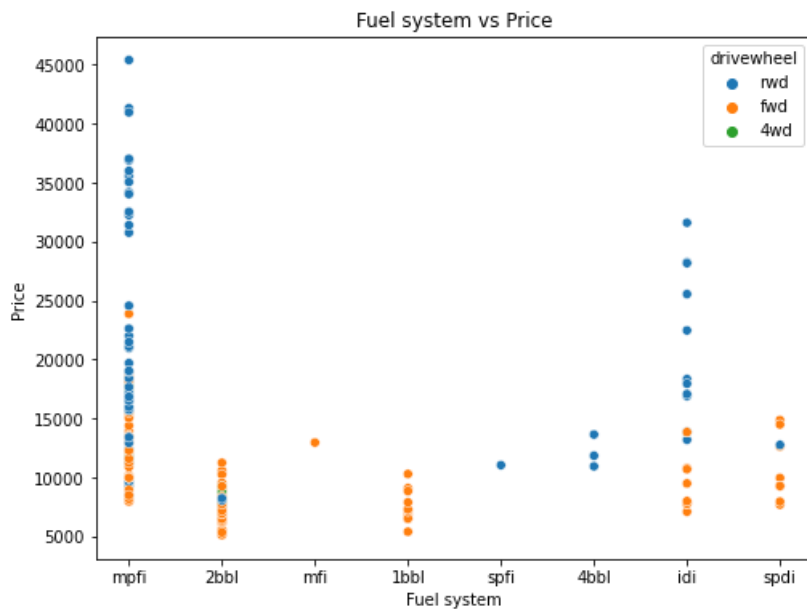
In [44]:

```
plt.figure(figsize=(8,6))

plt.title('Fuel system vs Price')
sns.scatterplot(x=car_data['fuelsystem'],y=car_data['price'],hue=car_data['drivewheel'])
plt.xlabel('Fuel system')
plt.ylabel('Price')

plt.show()
plt.tight_layout()

#from the plot we get that
#it has less impact on price.
```



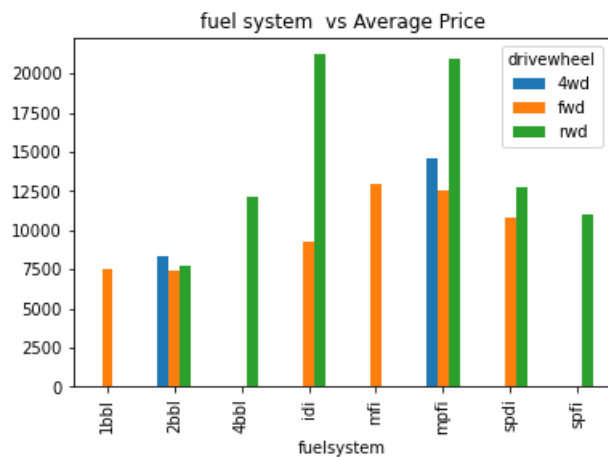
<Figure size 432x288 with 0 Axes>

In [47]:

```
plt.figure(figsize=(25, 6))

df = pd.DataFrame(car_data.groupby(['fuelsystem', 'drivewheel'])['price'].mean().unstack(fill_value=0))
df.plot.bar()
plt.title('fuel system vs Average Price')
plt.show()
#idi and mpfi are highly priced.
```

<Figure size 1800x432 with 0 Axes>



In [52]:

```
#significant variables
#EngineType
#fueltype
# CarBody
# Aspiration
# Cylinder Number
# Drivewheel
# Curbweight
# Car Length
# Car width
# Engine Size
# Boreratio
# Horse Power
# Wheel base
# Fuel Economy
```

In [54]:

```
car_data_lr = car_data[['price', 'fueltype', 'aspiration', 'carbody', 'drivewheel', 'wheelbase',
                        'curbweight', 'enginetype', 'cylindernumber', 'enginesize', 'boreratio', 'horsepower',
                        'carlength', 'carwidth']]
car_data_lr.head()
```

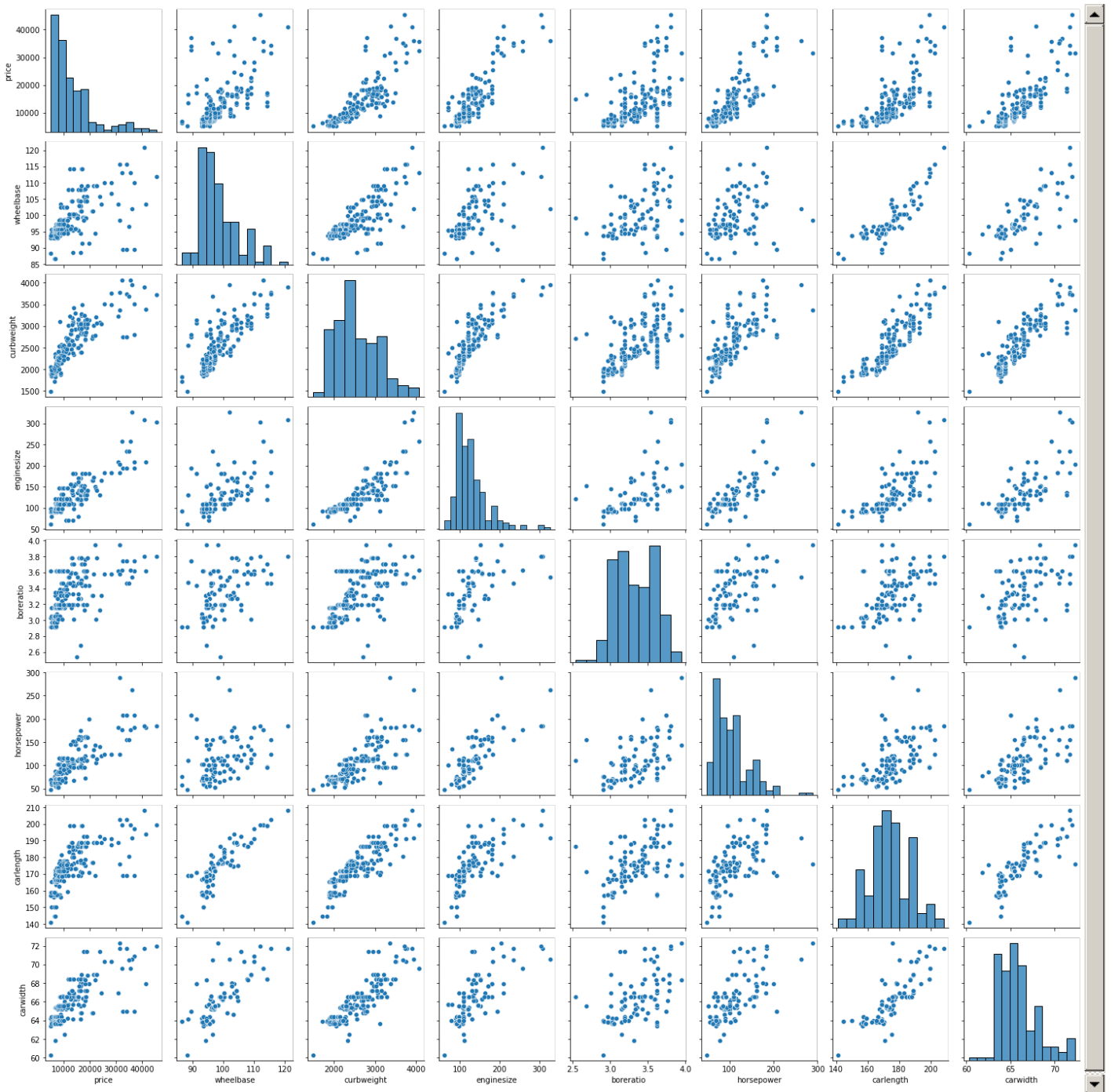
Out[54]:

	price	fueltype	aspiration	carbody	drivewheel	wheelbase	curbweight	enginetype	cylindernumber	enginesize	boreratio	horsepower	c
0	13495.0	gas	std	convertible	rwd	88.6	2548	dohc	four	130	3.47	111	
1	16500.0	gas	std	convertible	rwd	88.6	2548	dohc	four	130	3.47	111	
2	16500.0	gas	std	hatchback	rwd	94.5	2823	ohcv	six	152	2.68	154	
3	13950.0	gas	std	sedan	fwd	99.8	2337	ohc	four	109	3.19	102	
4	17450.0	gas	std	sedan	4wd	99.4	2824	ohc	five	136	3.19	115	



In [57]:

```
sns.pairplot(car_data_lr)
plt.show()
```



In [60]:

```
#train and test split

from sklearn.model_selection import train_test_split

np.random.seed(0)
df_train, df_test = train_test_split(car_data_lr, train_size = 0.7, test_size = 0.3, random_state = 100)
```

In [61]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
num_vars = ['wheelbase', 'curbweight', 'enginesize', 'boreratio', 'horsepower', 'carlength', 'carwidth', 'price']
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

<ipython-input-61-1bbb05ce7c2d>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
df\_train[num\_vars] = scaler.fit\_transform(df\_train[num\_vars])  
E:\Anaconda\lib\site-packages\pandas\core\indexing.py:1736: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
df\_train.loc[:, num\_vars] = scaler.fit\_transform(df\_train.loc[:, num\_vars])

In [62]:

```
df_train.head()
```

Out[62]:

	price	fueltype	aspiration	carbody	drivewheel	wheelbase	curbweight	enginetype	cylindernumber	enginesize	boreratio	horsepower
122	0.068818	gas	std	sedan	fwd	0.244828	0.272692	ohc	four	0.139623	0.230159	0.083333
125	0.466890	gas	std	hatchback	rwd	0.272414	0.500388	ohc	four	0.339623	1.000000	0.395833
166	0.122110	gas	std	hatchback	rwd	0.272414	0.314973	dohc	four	0.139623	0.444444	0.266667
1	0.314446	gas	std	convertible	rwd	0.068966	0.411171	dohc	four	0.260377	0.626984	0.262500
199	0.382131	gas	turbo	wagon	rwd	0.610345	0.647401	ohc	four	0.260377	0.746032	0.475000

df\_train.describe()

In [63]:

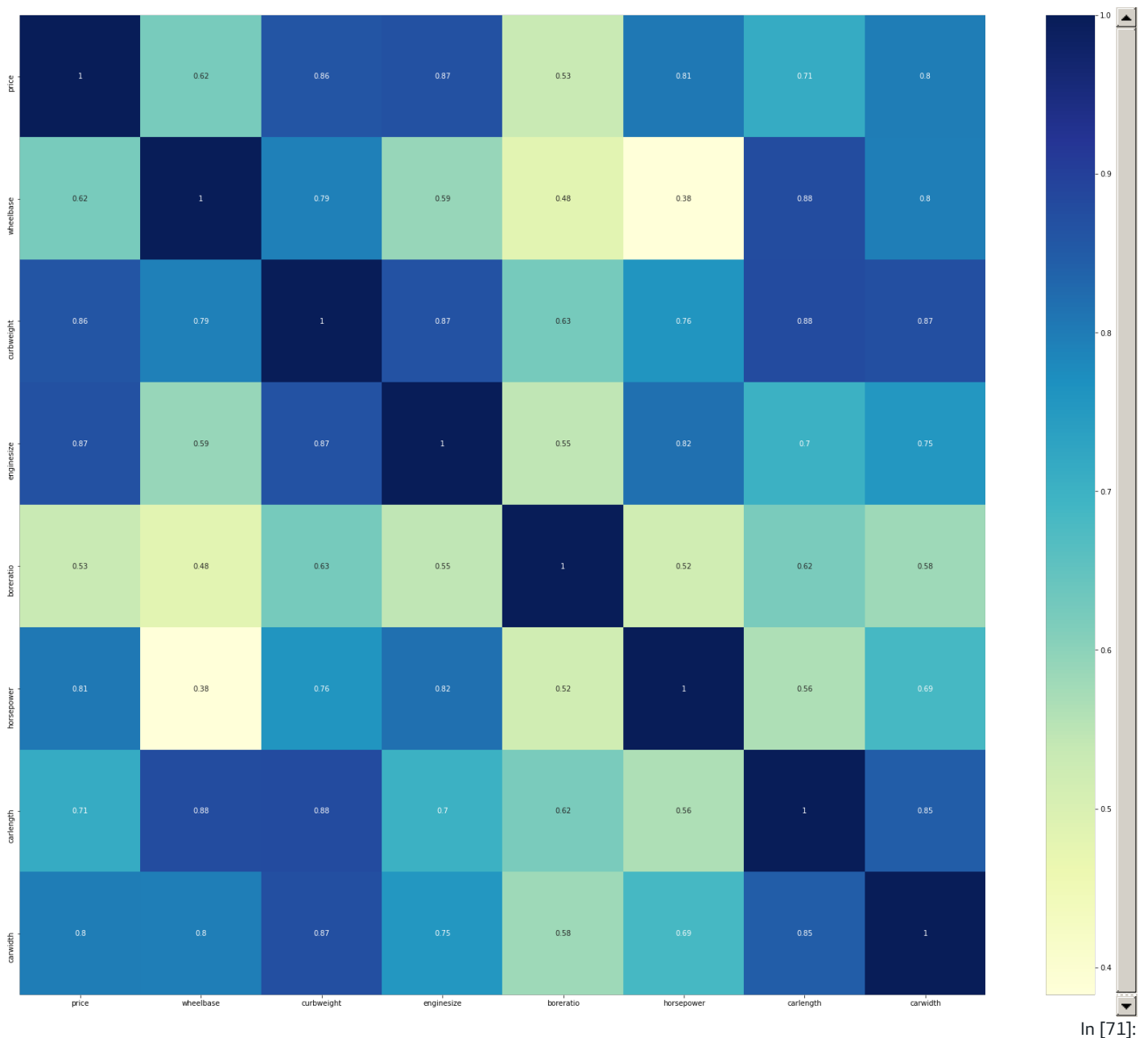
```
df_train.describe()
```

Out[63]:

	price	wheelbase	curbweight	enginesize	boreratio	horsepower	carlength	carwidth
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000
mean	0.219310	0.411141	0.407878	0.241351	0.497946	0.227302	0.525476	0.461655
std	0.215682	0.205581	0.211269	0.154619	0.207140	0.165511	0.204848	0.184517
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.067298	0.272414	0.245539	0.135849	0.305556	0.091667	0.399187	0.304167
50%	0.140343	0.341379	0.355702	0.184906	0.500000	0.191667	0.502439	0.425000
75%	0.313479	0.503448	0.559542	0.301887	0.682540	0.283333	0.669919	0.550000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [64]:

```
#Correlation using heatmap
plt.figure(figsize = (30, 25))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
#from plot
#Highly correlated variables to price are - curbweight, enginesize, horsepower, carwidth and highend.
```



*#still analysing and finding more significant variables*

*#Engine Size*

*#Curb Weight*

*#Horsepower*

*#Car Width*

*#Car Length*

*#City mpg (Negative correlation)*

```
car_data.drop(['wheelbase','carheight','boreratio','stroke','compressionratio','peakrpm'],axis=1,inplace=True)
car_data.head()
```

Out[71]:

	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	carlength	carwidth	curbweight	enginyer
0	3	alfa-romero	gas	std	two	convertible	rwd	front	168.8	64.1	2548	do
1	3	alfa-romero	gas	std	two	convertible	rwd	front	168.8	64.1	2548	do
2	1	alfa-romero	gas	std	two	hatchback	rwd	front	171.2	65.5	2823	oh
3	2	audi	gas	std	four	sedan	fwd	front	176.6	66.2	2337	o
4	2	audi	gas	std	four	sedan	4wd	front	176.6	66.4	2824	o

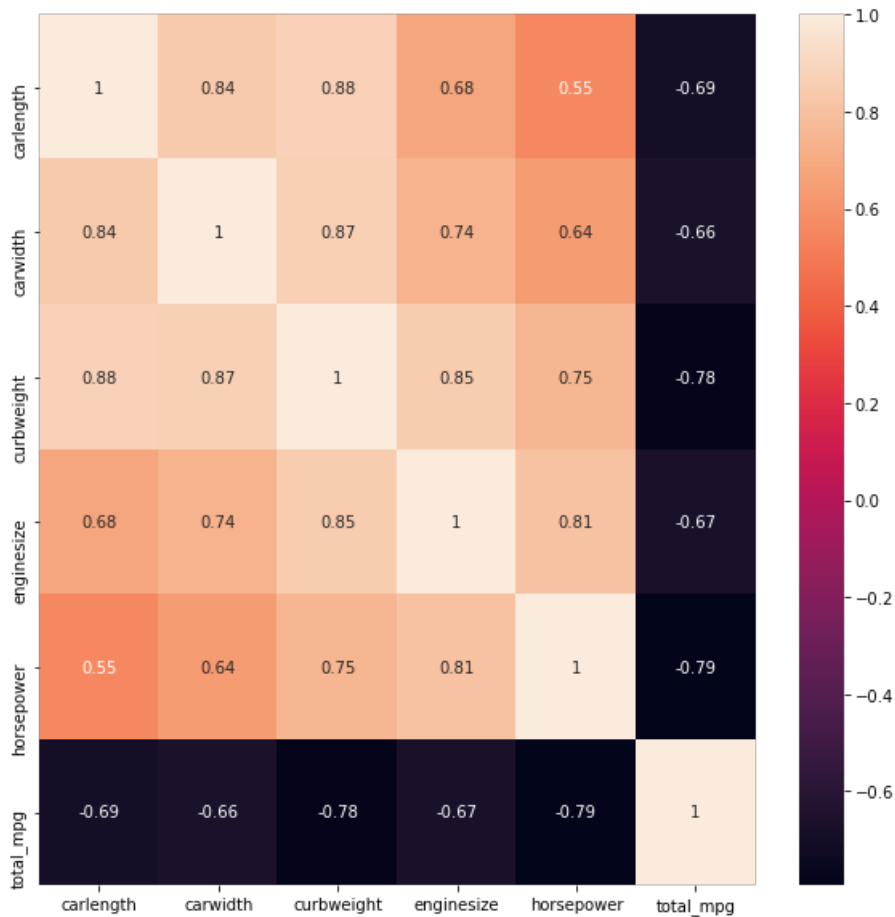
In [73]:

```
plt.figure(figsize=[10,10])
sns.heatmap(car_data.drop('price',axis=1).corr(),annot=True)
```



Out[73]:

&lt;AxesSubplot:&gt;



In [74]:

```
car_data=pd.get_dummies(car_data)
```

In [75]:

```
predictors=car_data['horsepower']
target=car_data['price']
```

In [76]:

```
import statsmodels.api as sm
predictors= sm.add_constant(predictors)
lm_1 = sm.OLS(target,predictors).fit()
print(lm_1.summary())
```

#### OLS Regression Results

```
=====
Dep. Variable:          price    R-squared:                0.653
Model:                  OLS      Adj. R-squared:           0.651
Method:                 Least Squares    F-statistic:           382.2
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):    1.48e-48
Time:                  19:28:05    Log-Likelihood:       -2024.0
No. Observations:      205    AIC:                   4052.
Df Residuals:          203    BIC:                   4059.
Df Model:              1
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	-3721.7615	929.849	-4.003	0.000	-5555.163	-1888.360
horsepower	163.2631	8.351	19.549	0.000	146.796	179.730

```
=====
Omnibus:                 47.741    Durbin-Watson:           0.792
Prob(Omnibus):           0.000    Jarque-Bera (JB):        91.702
Skew:                    1.141    Prob(JB):                 1.22e-20
Kurtosis:                 5.352    Cond. No.                 314.
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [77]:

```
car_data.drop(['horsepower','curbweight','total_mpg'],axis=1,inplace=True)
car_data.shape
```

Out[77]:

```
(205, 70)
```

In [78]:

```
cols_to_drop=car_data.corr()[ (car_data.corr()['price']<=0.5) & (car_data.corr()['price']>=-0.5)]
cols_to_drop=cols_to_drop.reset_index()['index']
cols_to_drop=list(cols_to_drop)
car_data.drop(cols_to_drop,axis=1,inplace=True)
```

In [80]:

```
car_data.head()
```

Out[80]:

	carlength	carwidth	enginesize	price	CompanyName_buick	drivewheel_fwd	drivewheel_rwd	cylindernumber_four	fuelsystem_2bbl	fuelsys
0	168.8	64.1	130	13495.0	0	0	1	1	0	
1	168.8	64.1	130	16500.0	0	0	1	1	0	
2	171.2	65.5	152	16500.0	0	0	1	0	0	
3	176.6	66.2	109	13950.0	0	1	0	1	0	
4	176.6	66.4	136	17450.0	0	0	0	0	0	

In [81]:

```
car_data.shape
```

Out[81]:

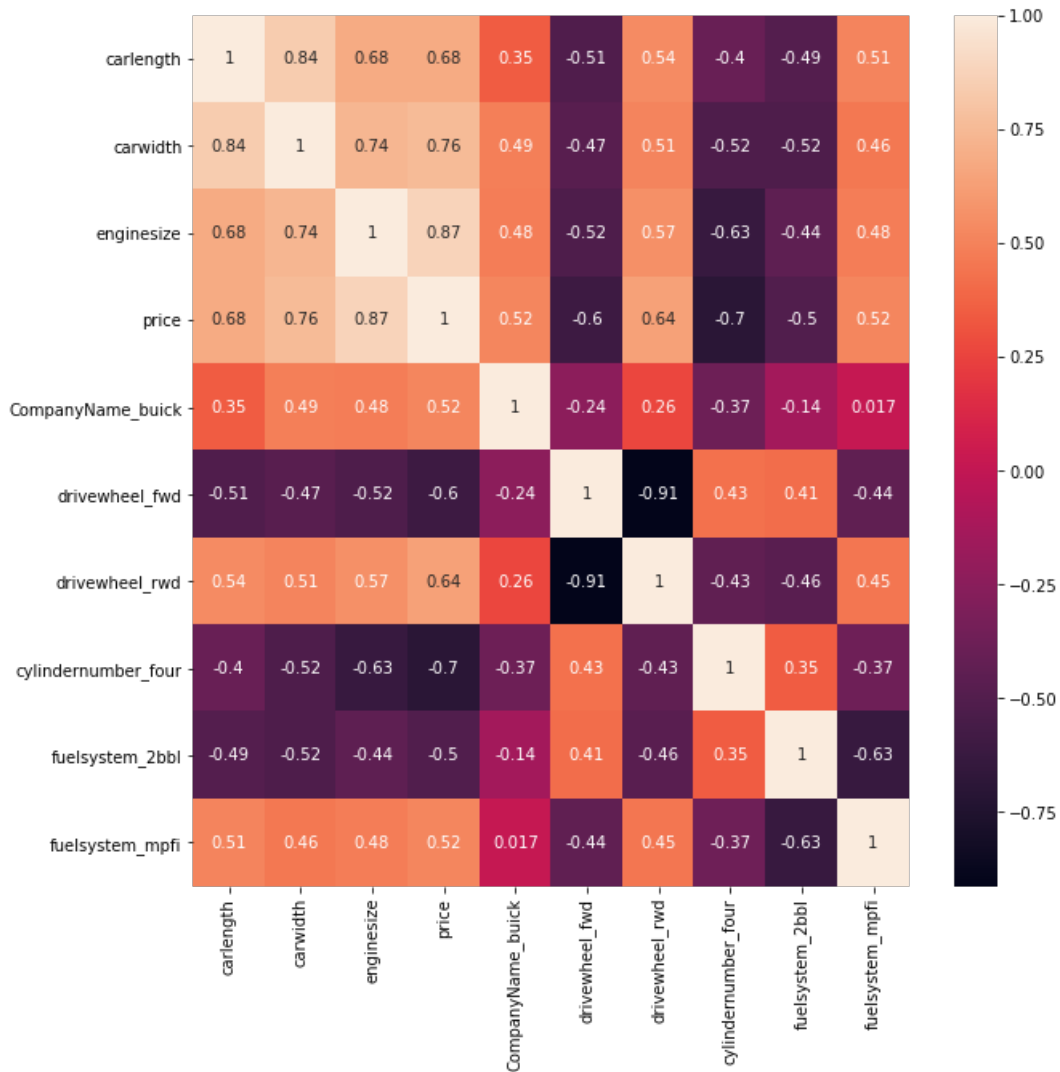
```
(205, 10)
```

In [82]:

```
plt.figure(figsize=[10,10])
sns.heatmap(car_data.corr(),annot=True)
```

Out[82]:

&lt;AxesSubplot:&gt;



In [84]:

```
#drop all columns having correlation to price around 0.5
car_data.drop(['carlength','carwidth','fuelsystem_2bbl', 'fuelsystem_mpf'],axis=1,inplace=True)
```

In [85]:

```
predictors=car_data.drop('price',axis=1)
target=car_data.price
predictors1=predictors['enginesize']
import statsmodels.api as sm
predictors1= sm.add_constant(predictors1)
lm_1 = sm.OLS(target,predictors1).fit()
print(lm_1.summary())
```

```

OLS Regression Results
=====
Dep. Variable:          price    R-squared:                0.764
Model:                  OLS      Adj. R-squared:           0.763
Method:                 Least Squares    F-statistic:             657.6
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):      1.35e-65
Time:                  19:35:37    Log-Likelihood:         -1984.4
No. Observations:      205    AIC:                    3973.
Df Residuals:          203    BIC:                    3979.
Df Model:               1
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	-8005.4455	873.221	-9.168	0.000	-9727.191	-6283.700
enginesize	167.6984	6.539	25.645	0.000	154.805	180.592

```

=====
Omnibus:                23.788    Durbin-Watson:           0.768
Prob(Omnibus):          0.000    Jarque-Bera (JB):        33.092
Skew:                   0.717    Prob(JB):                6.52e-08
Kurtosis:               4.348    Cond. No.:               429.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [86]:

```

predictors2=predictors[['enginesize','drivewheel_fwd']]
import statsmodels.api as sm
predictors2= sm.add_constant(predictors2)
lm_2 = sm.OLS(target,predictors2).fit()
print(lm_2.summary())

```

```

OLS Regression Results
=====
Dep. Variable:          price    R-squared:                0.794
Model:                  OLS      Adj. R-squared:           0.792
Method:                 Least Squares    F-statistic:             390.3
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):      4.11e-70
Time:                  19:36:43    Log-Likelihood:         -1970.3
No. Observations:      205    AIC:                    3947.
Df Residuals:          202    BIC:                    3957.
Df Model:               2
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	-3510.5293	1160.633	-3.025	0.003	-5799.040	-1222.019
enginesize	147.4621	7.157	20.604	0.000	133.350	161.574
drivewheel_fwd	-3291.5804	603.483	-5.454	0.000	-4481.515	-2101.646

```

=====
Omnibus:                21.512    Durbin-Watson:           0.819
Prob(Omnibus):          0.000    Jarque-Bera (JB):        35.873
Skew:                   0.580    Prob(JB):                1.62e-08
Kurtosis:               4.689    Cond. No.:               655.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [87]:

```

predictors3=predictors[['enginesize','drivewheel_rwd']]
import statsmodels.api as sm
predictors3= sm.add_constant(predictors3)
lm_3 = sm.OLS(target,predictors3).fit()
print(lm_3.summary())

```

```

OLS Regression Results
=====
Dep. Variable:          price    R-squared:                0.795
Model:                  OLS      Adj. R-squared:           0.793
Method:                 Least Squares    F-statistic:              391.4
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):       3.26e-70
Time:                  19:37:07    Log-Likelihood:          -1970.1
No. Observations:      205    AIC:                     3946.
Df Residuals:          202    BIC:                     3956.
Df Model:               2
Covariance Type:       nonrobust
=====
               coef    std err          t      P>|t|      [0.025      0.975]
-----
const          -6378.7190     868.209     -7.347     0.000    -8090.634    -4666.804
enginesize       144.6322       7.412     19.512     0.000     130.017     159.248
drivewheel_rwd  3508.0574     637.511      5.503     0.000     2251.028     4765.087
=====
Omnibus:                 19.717    Durbin-Watson:           0.781
Prob(Omnibus):           0.000    Jarque-Bera (JB):        33.357
Skew:                   0.528    Prob(JB):                5.71e-08
Kurtosis:                4.670    Cond. No.:               481.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [88]:

```

#adding both models
predictors4=predictors[['enginesize','drivewheel_fwd','drivewheel_rwd']]
import statsmodels.api as sm
predictors4= sm.add_constant(predictors4)
lm_4 = sm.OLS(target,predictors4).fit()
print(lm_4.summary())

```

```

OLS Regression Results
=====
Dep. Variable:          price    R-squared:                0.797
Model:                  OLS      Adj. R-squared:           0.794
Method:                 Least Squares    F-statistic:              262.5
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):       3.07e-69
Time:                  19:37:37    Log-Likelihood:          -1969.2
No. Observations:      205    AIC:                     3946.
Df Residuals:          201    BIC:                     3960.
Df Model:               3
Covariance Type:       nonrobust
=====
               coef    std err          t      P>|t|      [0.025      0.975]
-----
const          -4829.7218    1458.548     -3.311     0.001    -7705.740    -1953.703
enginesize       144.5557       7.399     19.537     0.000     129.966     159.145
drivewheel_fwd -1656.2169    1254.380     -1.320     0.188    -4129.650     817.216
drivewheel_rwd  1971.1119    1326.626      1.486     0.139     -644.777     4587.001
=====
Omnibus:                 20.686    Durbin-Watson:           0.794
Prob(Omnibus):           0.000    Jarque-Bera (JB):        36.360
Skew:                   0.539    Prob(JB):                1.27e-08
Kurtosis:                4.760    Cond. No.:              1.13e+03
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.13e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [89]:

```

predictors5=predictors[['enginesize','drivewheel_rwd','cylindernumber_four']]
import statsmodels.api as sm
predictors5= sm.add_constant(predictors5)
lm_5 = sm.OLS(target,predictors5).fit()
print(lm_5.summary())

```

```

OLS Regression Results
=====
Dep. Variable:          price    R-squared:                0.823
Model:                  OLS      Adj. R-squared:            0.820
Method:                 Least Squares    F-statistic:              311.8
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):       2.55e-75
Time:                  19:38:42    Log-Likelihood:          -1954.9
No. Observations:      205    AIC:                     3918.
Df Residuals:          201    BIC:                     3931.
Df Model:               3
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	21.9214	1389.248	0.016	0.987	-2717.449	2761.292
enginesize	120.8814	8.074	14.971	0.000	104.960	136.803
drivewheel_rwd	3098.2795	597.869	5.182	0.000	1919.379	4277.180
cylindernumber_four	-4170.3627	736.215	-5.665	0.000	-5622.059	-2718.666

```

=====
Omnibus:                12.155    Durbin-Watson:           0.948
Prob(Omnibus):          0.002    Jarque-Bera (JB):        25.608
Skew:                   0.202    Prob(JB):                2.75e-06
Kurtosis:               4.684    Cond. No.:               861.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [98]:

```

predictors6=predictors[['enginesize','drivewheel_fwd','cylindernumber_four']]
import statsmodels.api as sm
predictors6= sm.add_constant(predictors6)
lm_6 = sm.OLS(target,predictors6).fit()
print(lm_6.summary())

```

```

OLS Regression Results
=====
Dep. Variable:          price    R-squared:                0.821
Model:                  OLS      Adj. R-squared:            0.819
Method:                 Least Squares    F-statistic:              308.0
Date:                  Tue, 23 Feb 2021    Prob (F-statistic):       6.99e-75
Time:                  19:44:14    Log-Likelihood:          -1955.9
No. Observations:      205    AIC:                     3920.
Df Residuals:          201    BIC:                     3933.
Df Model:               3
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	2314.0832	1515.502	1.527	0.128	-674.239	5302.406
enginesize	124.4012	7.893	15.761	0.000	108.838	139.965
drivewheel_fwd	-2827.0579	570.267	-4.957	0.000	-3951.530	-1702.585
cylindernumber_four	-4087.0246	742.670	-5.503	0.000	-5551.448	-2622.601

```

=====
Omnibus:                12.945    Durbin-Watson:           0.985
Prob(Omnibus):          0.002    Jarque-Bera (JB):        25.263
Skew:                   0.273    Prob(JB):                3.27e-06
Kurtosis:               4.631    Cond. No.:               913.
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [100]:

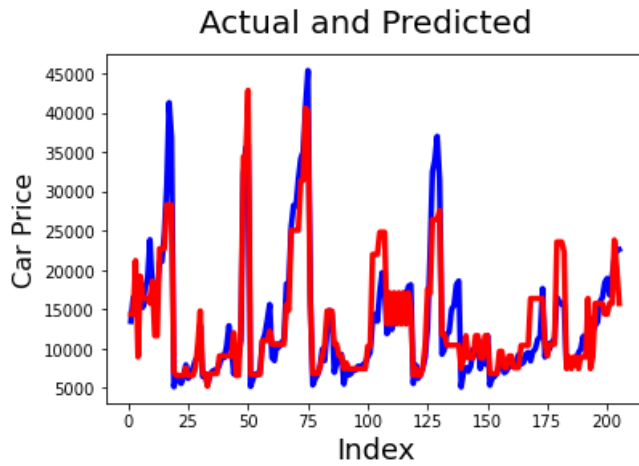
```

pred=lm_6.predict(predictors6)
# Actual vs Predicted
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target, color="blue", linewidth=3.5, linestyle="-") #Plotting Actual
plt.plot(c,pred, color="red", linewidth=3.5, linestyle="-") #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20) # Plot heading
plt.xlabel('Index', fontsize=18) # X-label
plt.ylabel('Car Price', fontsize=16)
(0, 0.5, 'Car Price')

```

Out[100]:

```
(0, 0.5, 'Car Price')
```



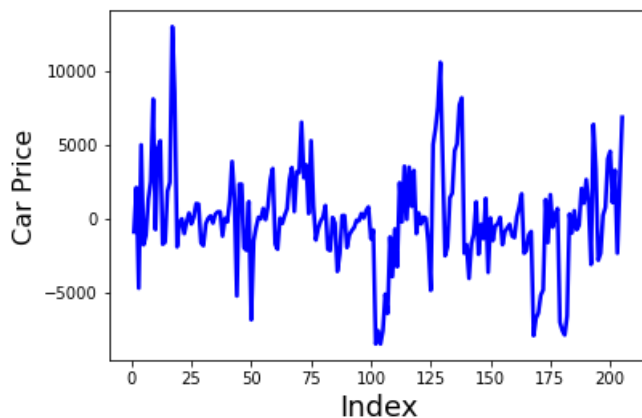
In [101]:

```
# Error terms
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target-pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)          # Plot heading
plt.xlabel('Index', fontsize=18)                  # X-label
plt.ylabel('Car Price', fontsize=16)
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(target, pred)
r_squared = r2_score(target, pred)
print('Mean Squared Error :',mse)
print('r_square_value :',r_squared)
```

Mean Squared Error : 11347528.099728283

r\_square\_value : 0.8213281339239666

Error Terms

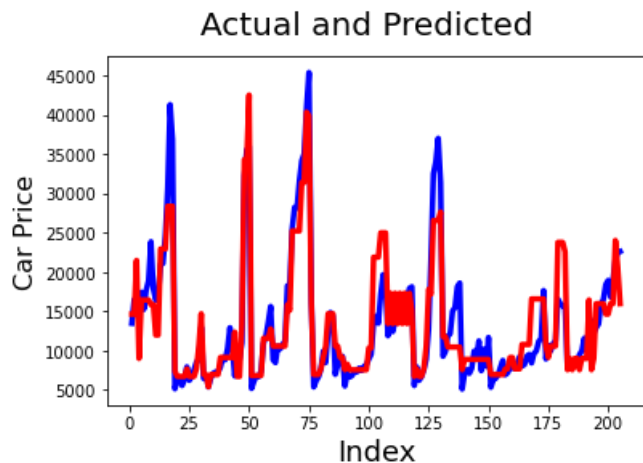


In [103]:

```
pred=lm_5.predict(predictors5)
# Actual vs Predicted
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target, color="blue", linewidth=3.5, linestyle="-")      #Plotting Actual
plt.plot(c,pred, color="red", linewidth=3.5, linestyle="-")        #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20)                  # Plot heading
plt.xlabel('Index', fontsize=18)                                    # X-label
plt.ylabel('Car Price', fontsize=16)
(0, 0.5, 'Car Price')
```

Out[103]:

(0, 0.5, 'Car Price')



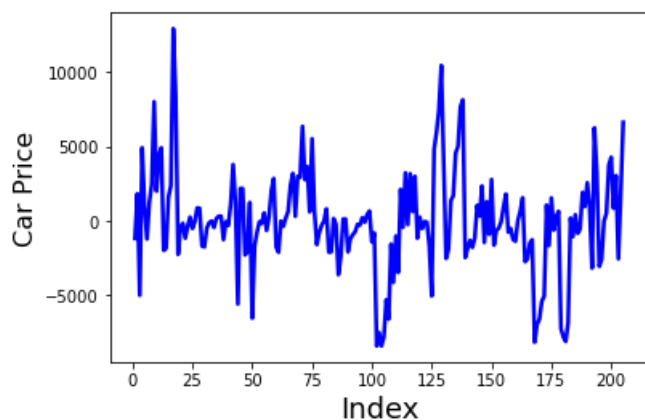
In [104]:

```
# Error terms
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target-pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)          # Plot heading
plt.xlabel('Index', fontsize=18)                  # X-label
plt.ylabel('Car Price', fontsize=16)
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(target, pred)
r_squared = r2_score(target, pred)
print('Mean Squared Error :',mse)
print('r_square_value :',r_squared)
```

Mean Squared Error : 11234025.847691916

r\_square\_value : 0.8231152772557074

Error Terms



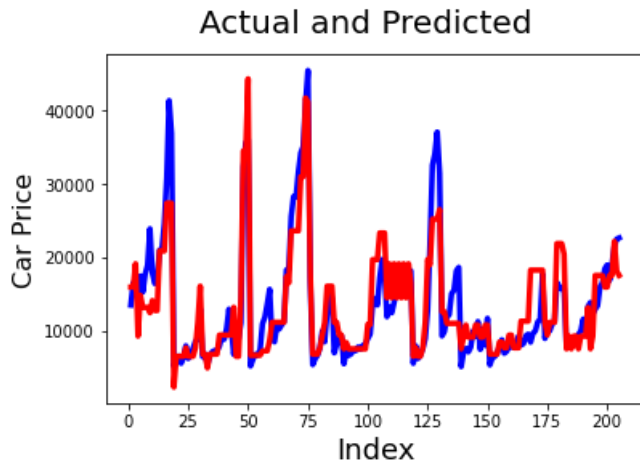
In [106]:

```
pred=lm_4.predict(predictors4)
# Actual vs Predicted
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target, color="blue", linewidth=3.5, linestyle="-")      #Plotting Actual
plt.plot(c,pred, color="red", linewidth=3.5, linestyle="-")        #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20)                  # Plot heading
plt.xlabel('Index', fontsize=18)                                    # X-label
plt.ylabel('Car Price', fontsize=16)
(0, 0.5, 'Car Price')
```



Out[106]:

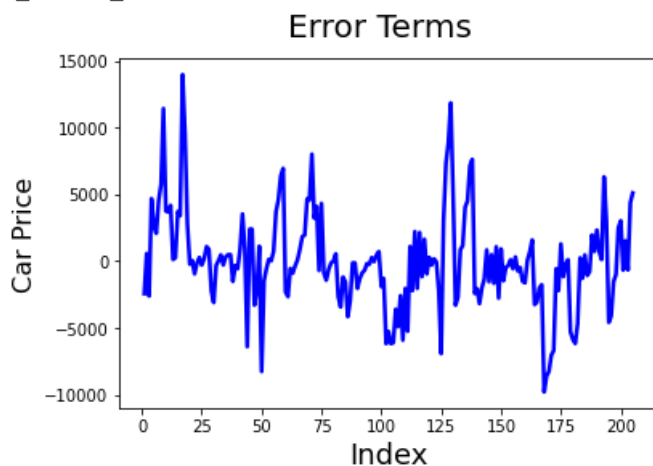
```
(0, 0.5, 'Car Price')
```



In [107]:

```
# Error terms
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target-pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)          # Plot heading
plt.xlabel('Index', fontsize=18)                  # X-label
plt.ylabel('Car Price', fontsize=16)
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(target, pred)
r_squared = r2_score(target, pred)
print('Mean Squared Error :',mse)
print('r_square_value :',r_squared)
```

```
Mean Squared Error : 12915408.499455474
r_square_value : 0.7966411611893495
```

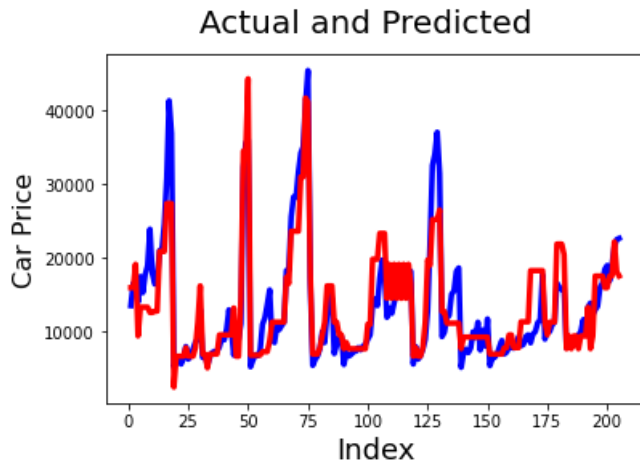


In [108]:

```
pred=lm_3.predict(predictors3)
# Actual vs Predicted
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target, color="blue", linewidth=3.5, linestyle="-")      #Plotting Actual
plt.plot(c,pred, color="red", linewidth=3.5, linestyle="-")        #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20)                  # Plot heading
plt.xlabel('Index', fontsize=18)                                    # X-label
plt.ylabel('Car Price', fontsize=16)
(0, 0.5, 'Car Price')
```

Out[108]:

```
(0, 0.5, 'Car Price')
```

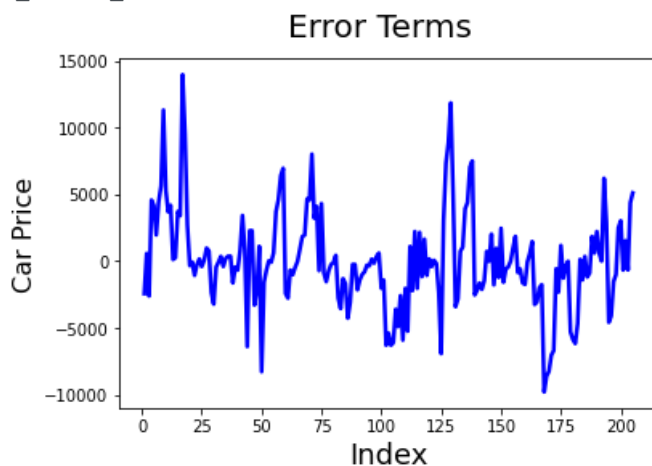


In [109]:

```
# Error terms
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target-pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)          # Plot heading
plt.xlabel('Index', fontsize=18)                  # X-label
plt.ylabel('Car Price', fontsize=16)
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(target, pred)
r_squared = r2_score(target, pred)
print('Mean Squared Error :',mse)
print('r_square_value :',r_squared)
```

Mean Squared Error : 13027426.534669885

r\_square\_value : 0.7948773875101807



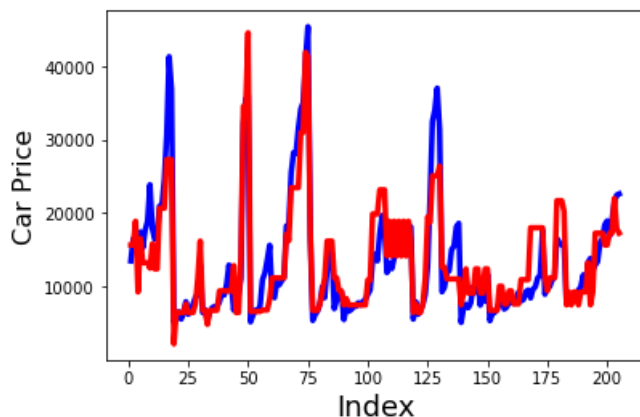
In [110]:

```
pred=lm_2.predict(predictors2)
# Actual vs Predicted
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target, color="blue", linewidth=3.5, linestyle="-")      #Plotting Actual
plt.plot(c,pred, color="red", linewidth=3.5, linestyle="-")         #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20)                  # Plot heading
plt.xlabel('Index', fontsize=18)                                    # X-label
plt.ylabel('Car Price', fontsize=16)
(0, 0.5, 'Car Price')
# Error terms
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target-pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)          # Plot heading
plt.xlabel('Index', fontsize=18)                  # X-label
plt.ylabel('Car Price', fontsize=16)
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(target, pred)
r_squared = r2_score(target, pred)
```

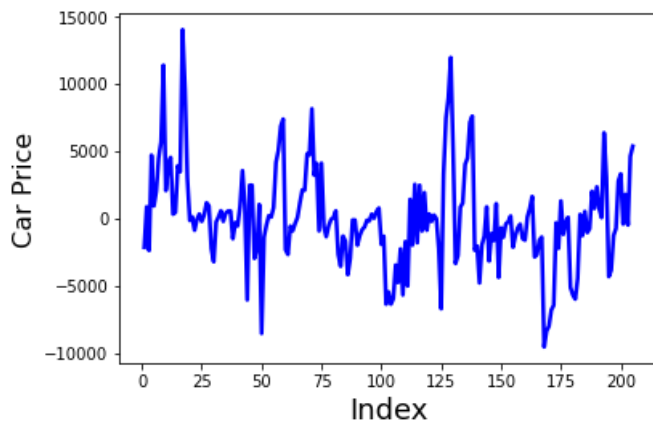
```
print('Mean_Squared_Error :',mse)
print('r_square_value :',r_squared)
Mean_Squared_Error : 13057261.284937426
r_square_value : 0.7944076261262594
```

```
Mean_Squared_Error : 13057261.284937426
r_square_value : 0.7944076261262594
```

### Actual and Predicted



### Error Terms

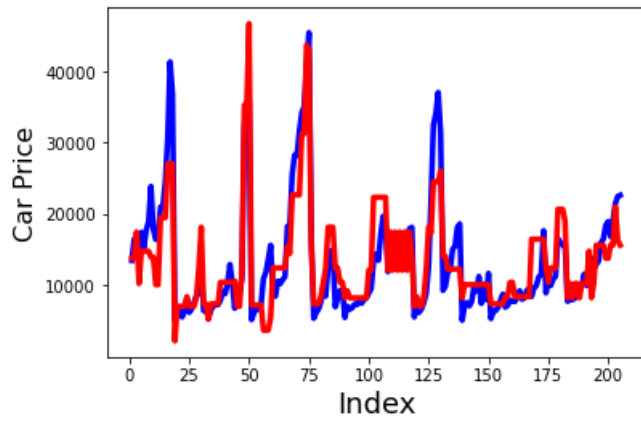


In [111]:

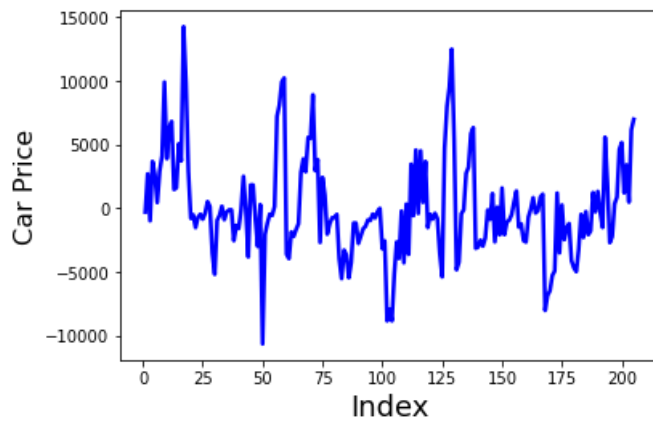
```
pred=lm_1.predict(predictors1)
# Actual vs Predicted
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target, color="blue", linewidth=3.5, linestyle="-") #Plotting Actual
plt.plot(c,pred, color="red", linewidth=3.5, linestyle="-") #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20) # Plot heading
plt.xlabel('Index', fontsize=18) # X-label
plt.ylabel('Car Price', fontsize=16)
(0, 0.5, 'Car Price')
# Error terms
c = [i for i in range(1,206,1)]
fig = plt.figure()
plt.plot(c,target-pred, color="blue", linewidth=2.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20) # Plot heading
plt.xlabel('Index', fontsize=18) # X-label
plt.ylabel('Car Price', fontsize=16)
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(target, pred)
r_squared = r2_score(target, pred)
print('Mean_Squared_Error :',mse)
print('r_square_value :',r_squared)
```

Mean\_Squared\_Error : 14980261.40555132  
r\_square\_value : 0.7641291357806176

Actual and Predicted



Error Terms



In []:

\\*from the above plots,engine size,wheeldrive,no of cylinders 4 plays a significant role in price predict