Car Price Prediction

Import Libraries

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

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

Read Data

In [2]:
```
cars = pd.read_csv('../input/CarPrice_Assignment.csv')
cars.head()
```

Out[2]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	64.1
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	64.1
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	66.2
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	66.4

In [3]:
```
cars.shape
```

Out[3]:
```
(205, 26)
```

In [4]:
```
cars.describe()
```

Out[4]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionr
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.00
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.14
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.97
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.00
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.60
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.00
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.40
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.00

In [5]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
car_ID              205 non-null int64
symboling           205 non-null int64
CarName             205 non-null object
fueltype            205 non-null object
aspiration          205 non-null object
doornumber          205 non-null object
carbody             205 non-null object
drivewheel          205 non-null object
enginelocation      205 non-null object
wheelbase           205 non-null float64
carlength           205 non-null float64
carwidth            205 non-null float64
carheight           205 non-null float64
curbweight          205 non-null int64
enginetype          205 non-null object
cylindernumber      205 non-null object
enginesize          205 non-null int64
fuelsystem          205 non-null object
boreratio           205 non-null float64
stroke              205 non-null float64
compressionratio    205 non-null float64
horsepower          205 non-null int64
peakrpm             205 non-null int64
citympg             205 non-null int64
highwaympg          205 non-null int64
price               205 non-null float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.7+ KB
```

Data Cleaning

In [6]:
```
#Splitting company name from CarName column
CompanyName = cars['CarName'].apply(lambda x : x.split(' ')[0])
cars.insert(3,"CompanyName",CompanyName)
cars.drop(['CarName'],axis=1,inplace=True)
cars.head()
```

Out[6]:

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

In [7]:
```
cars.CompanyName.unique()
```

Out[7]:
```
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)
```

Fixing invalid values

- There seems to be some spelling error in the CompanyName column.
  - `maxda` = `mazda`
  - `Nissan` = `nissan`
  - `porsche` = `porcshce`
  - `toyota` = `toyouta`
  - `vokswagen` = `volkswagen` = `vw`

In [8]:
```
cars.CompanyName = cars.CompanyName.str.lower()

def replace_name(a,b):
    cars.CompanyName.replace(a,b,inplace=True)

replace_name('maxda','mazda')
replace_name('porcshce','porsche')
replace_name('toyouta','toyota')
replace_name('vokswagen','volkswagen')
replace_name('vw','volkswagen')

cars.CompanyName.unique()
```

Out[8]:
```
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 [9]:
```
#Checking for duplicates
cars.loc[cars.duplicated()]
```

Out[9]:

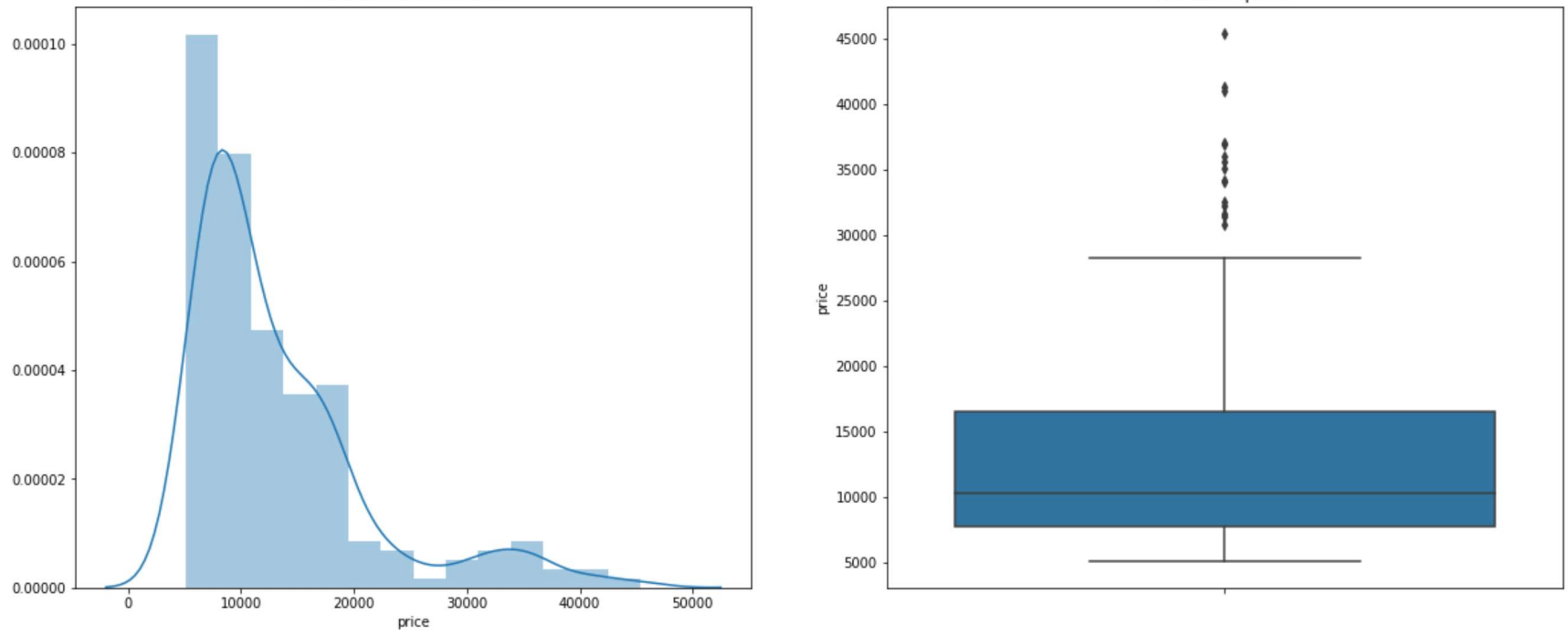
car_ID	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth

In [10]:
```
cars.columns
```

Out[10]:
```
Index(['car_ID', 'symboling', 'CompanyName', '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')
```

EDA

In [11]:
```
plt.figure(figsize=(20,8))

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

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

plt.show()
```



In [12]:
```
print(cars.price.describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1]))
```

```
count      205.000000
mean     13276.710571
std       7988.852332
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      16503.000000
85%      18500.000000
90%      22563.000000
100%     45400.000000
max      45400.000000
Name: price, dtype: float64
```

Inference :

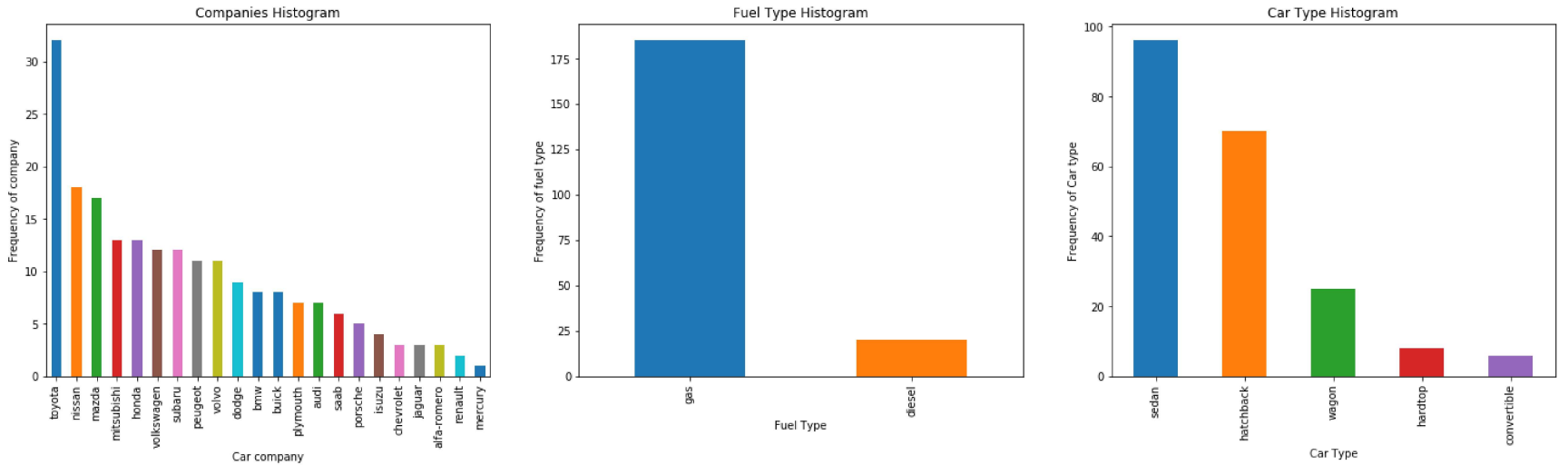
The plot appeared to be right-skewed, indicating that the majority of prices in the dataset are low (below 15,000). There is a notable disparity between the mean and the median of the price distribution. The data points are widely dispersed from the mean, suggesting a high variance in car prices. Specifically, 85% of the prices are below 18,500, while the remaining 15% fall between 18,500 and 45,400.

Visualising Categorical Data

- CompanyName
- Symboling
- fueltype
- enginetype
- carbody
- doornumber
- enginelocation
- fuelsystem
- cylindernumber
- aspiration
- drivewheel

In [13]:
```
plt.figure(figsize=(25, 6))

plt.subplot(1,3,1)
plt1 = cars.CompanyName.value_counts().plot('bar')
```

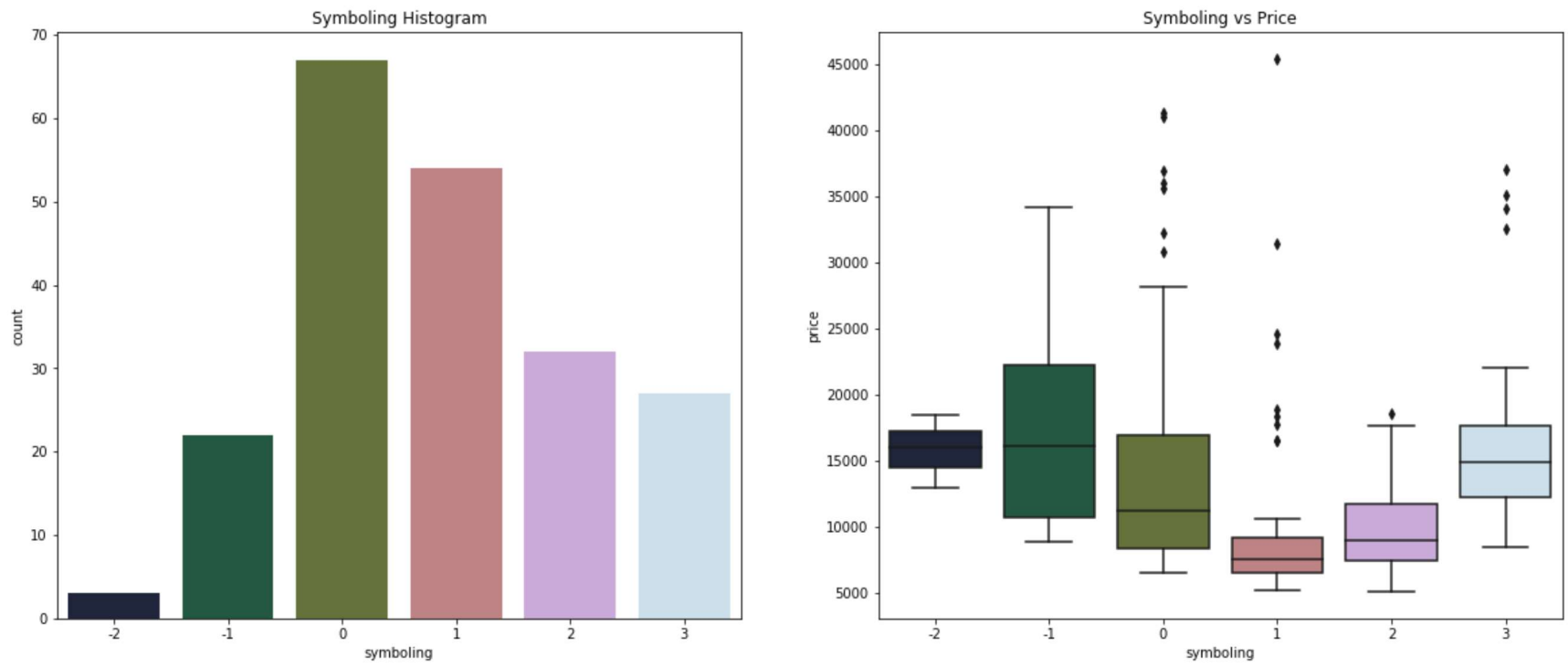
```
plt.title('Companies Histogram')
plt1.set(xlabel = 'Car company', ylabel='Frequency of company')

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

plt.subplot(1,3,3)
plt1 = cars.carbody.value_counts().plot('bar')
plt.title('Car Type Histogram')
plt1.set(xlabel = 'Car Type', ylabel='Frequency of Car type')

plt.show()
```



Inference :

It appears that Toyota is the favored car company. There are more gas-fueled cars than diesel. Sedans are the most preferred car type.

In [14]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
plt.title('Symboling Histogram')
sns.countplot(cars.symboling, palette=("cubehelix"))

plt.subplot(1,2,2)
plt.title('Symboling vs Price')
sns.boxplot(x=cars.symboling, y=cars.price, palette=("cubehelix"))

plt.show()
```

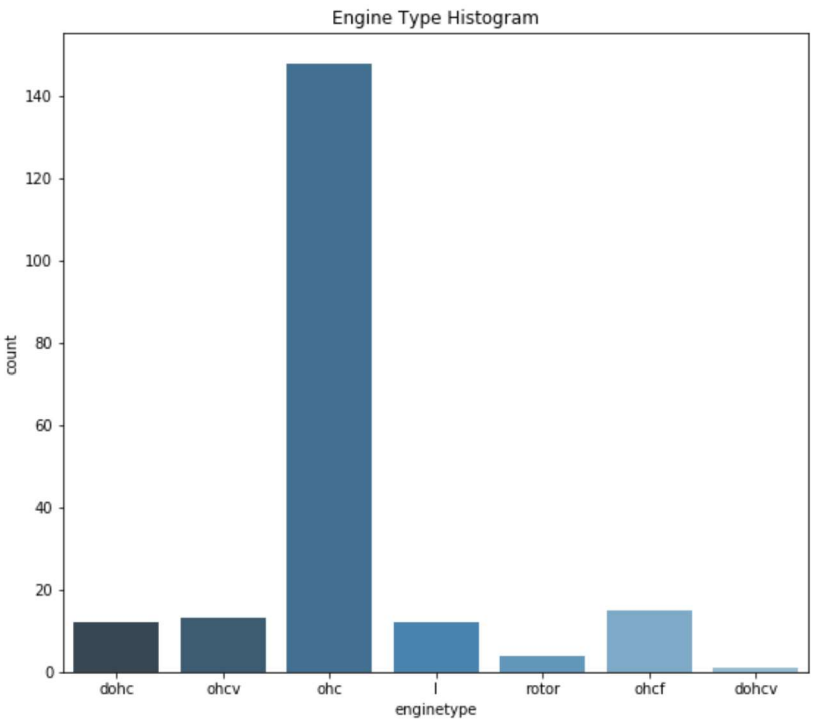


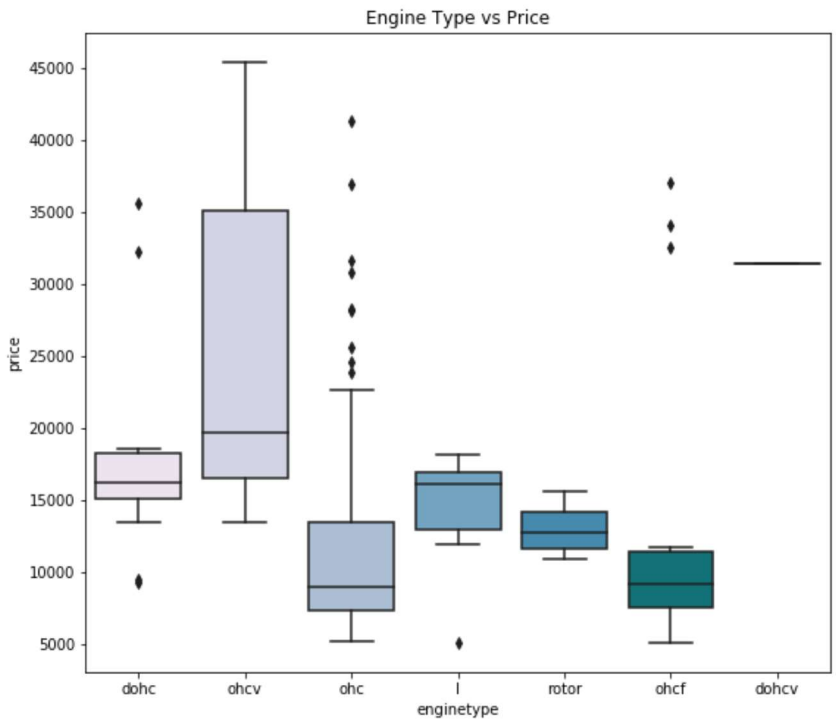
Inference :

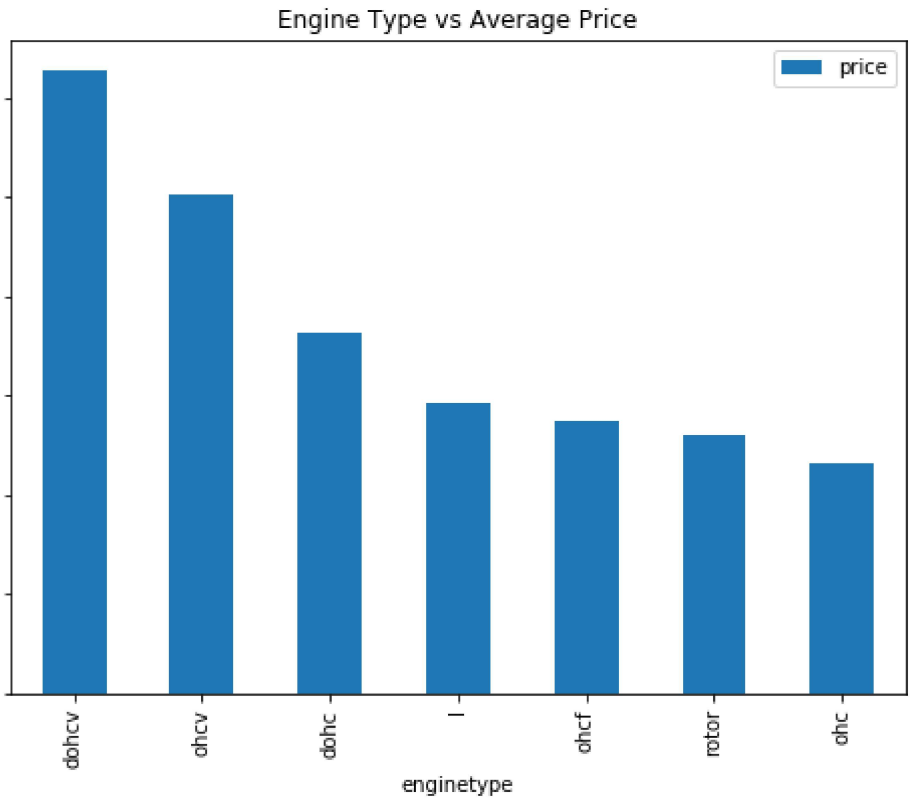
It appears that the symboling values of 0 and 1 have a high number of rows, indicating they are the most sold. Cars with a symboling of -1 seem to be higher priced, which aligns with the understanding that an insurance risk rating of -1 is relatively good. However, it's noticeable that cars with a symboling of 3 have a price range similar to those with a -2 value. There is a price dip at symboling 1.

In [15]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
plt.title('Engine Type Histogram')
sns.countplot(cars.enginetype, palette=("Blues_d"))

plt.subplot(1,2,2)
plt.title('Engine Type vs Price')
sns.boxplot(x=cars.enginetype, y=cars.price, palette=("PuBuGn"))

plt.show()

df = pd.DataFrame(cars.groupby(['enginetype'])['price'].mean().sort_values(ascending = False))
df.plot.bar(figsize=(8,6))
```

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





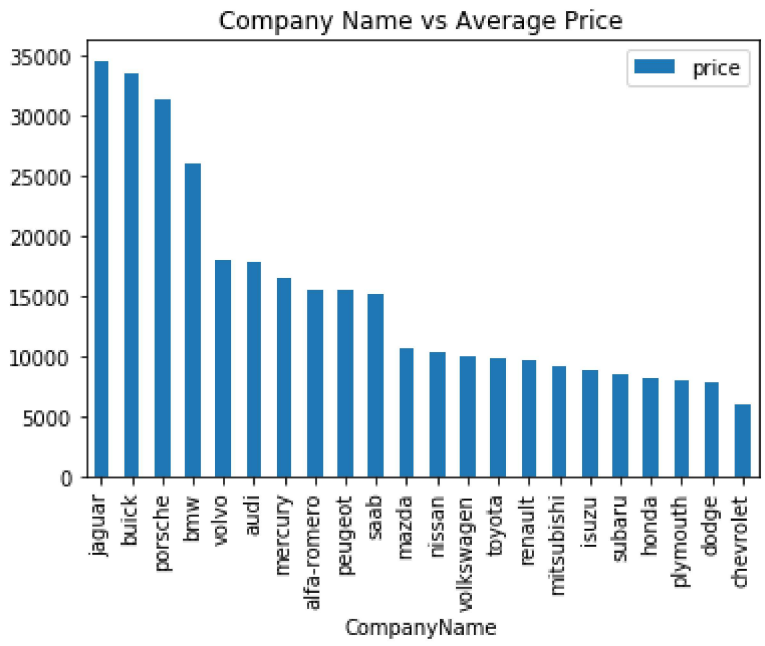


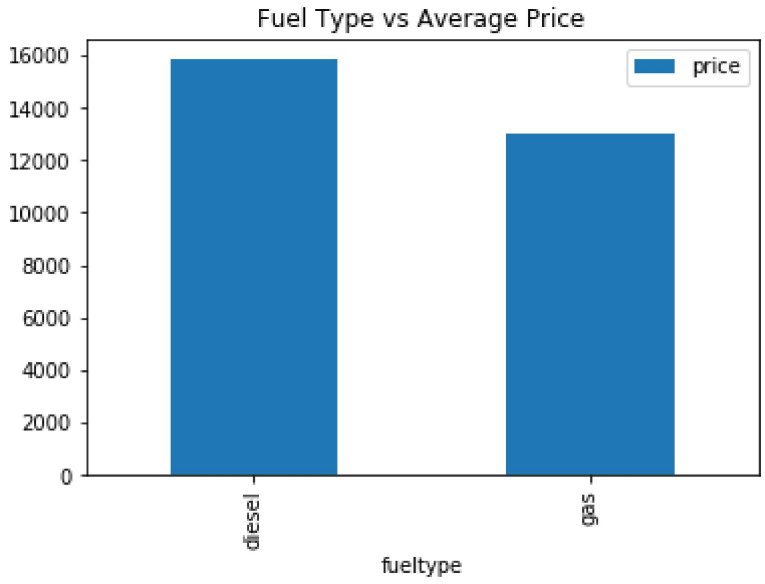
Inference :

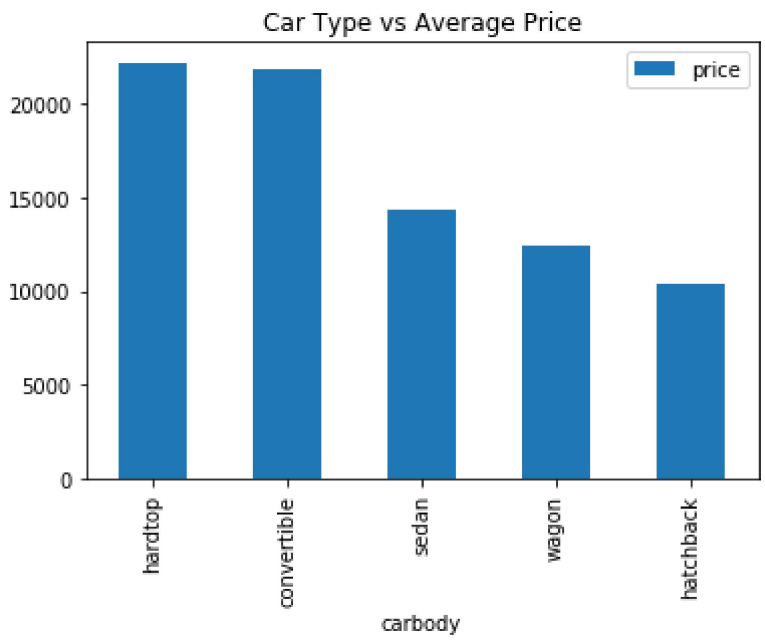
The 'ohc' engine type appears to be the most favored. 'ohcv' exhibits the highest price range, although 'dohcv' only has one row. 'ohc' and 'ohcf' types, on the other hand, show lower price ranges.

In [16]:
```
plt.figure(figsize=(25, 6))

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

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

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

```
<Figure size 1800x432 with 0 Axes>
```



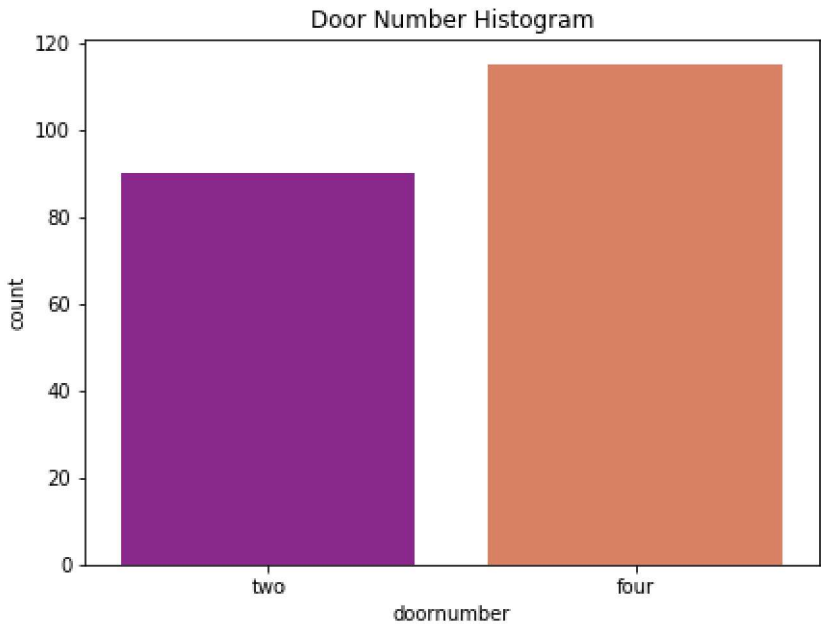


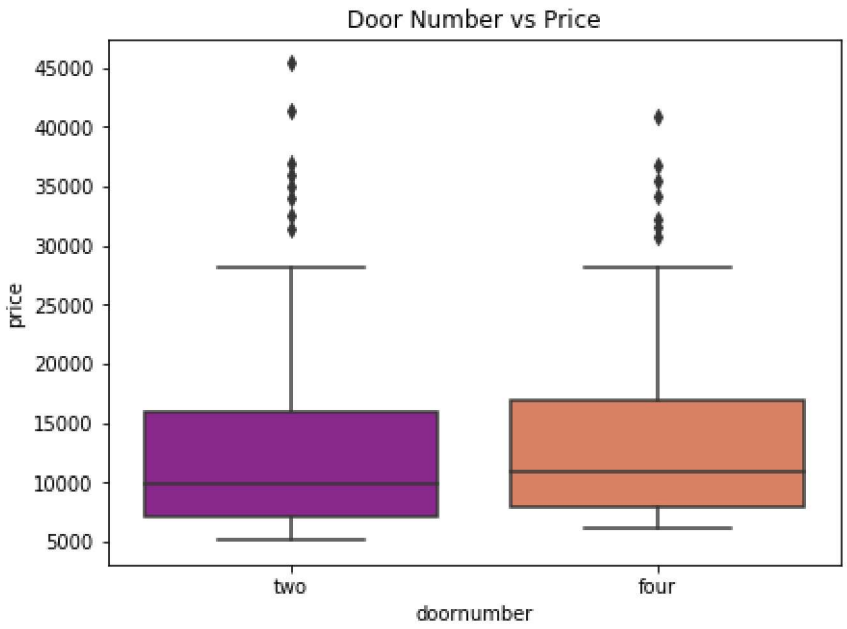


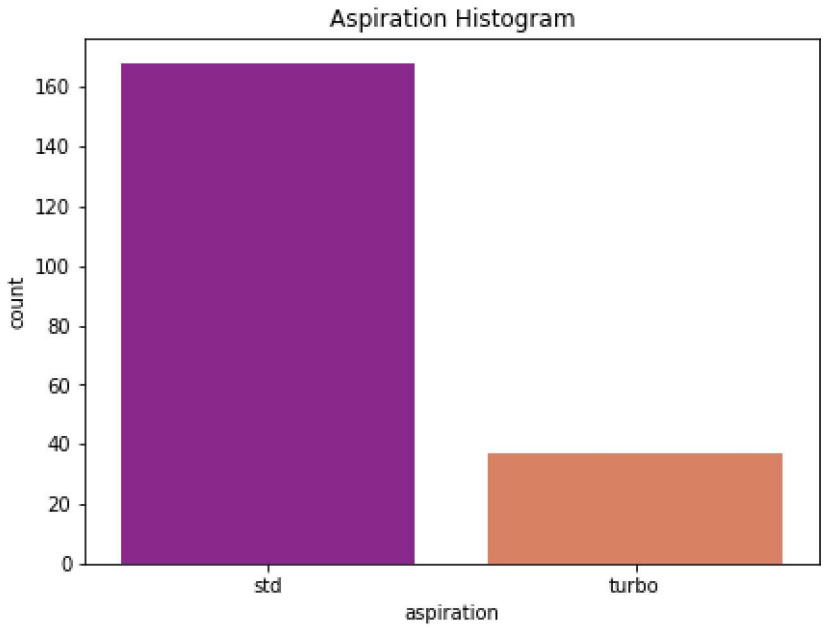
Inference :

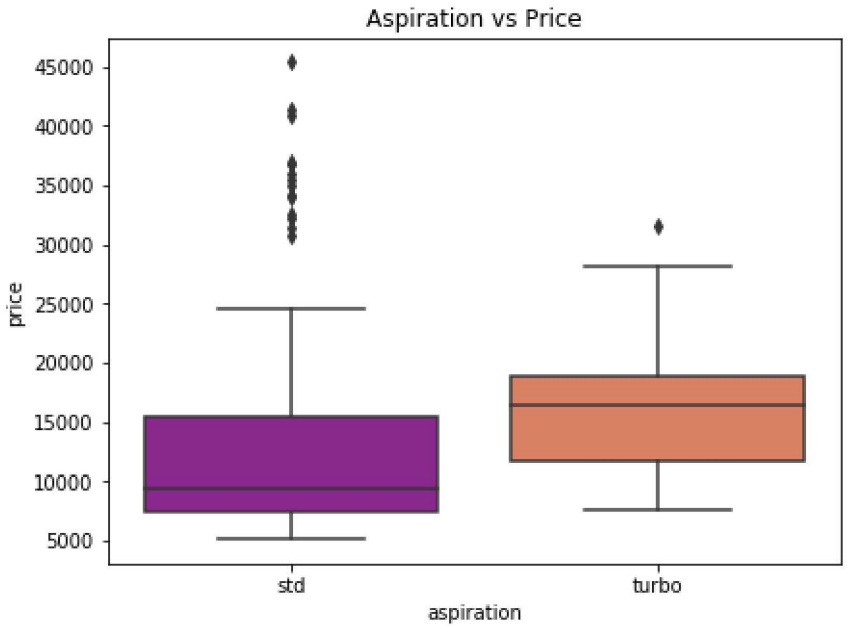
Jaguar and Buick appear to have the highest average prices among the car brands. Diesel cars have a higher average price compared to gas-fueled ones. Additionally, hardtops and convertibles show a higher average price.

In [17]:
```
plt.figure(figsize=(15,5))

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

plt.subplot(1,2,2)
plt.title('Door Number vs Price')
sns.boxplot(x=cars.doornumber, y=cars.price, palette=("plasma"))

plt.show()

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

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

plt.subplot(1,2,2)
plt.title('Aspiration vs Price')
sns.boxplot(x=cars.aspiration, y=cars.price, palette=("plasma"))

plt.show()
```





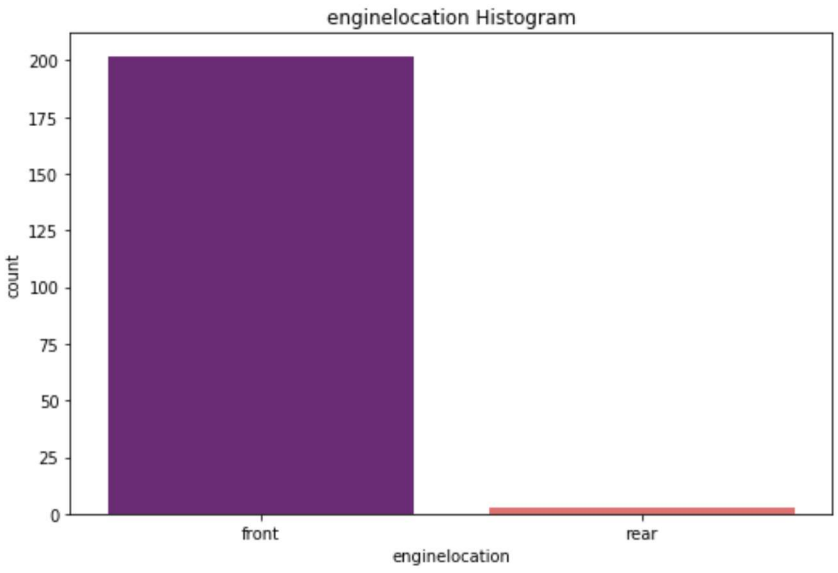


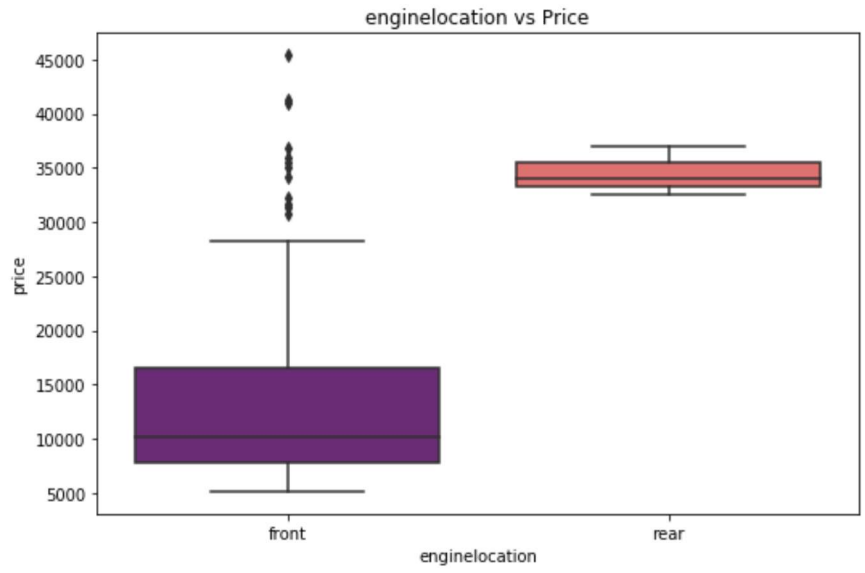


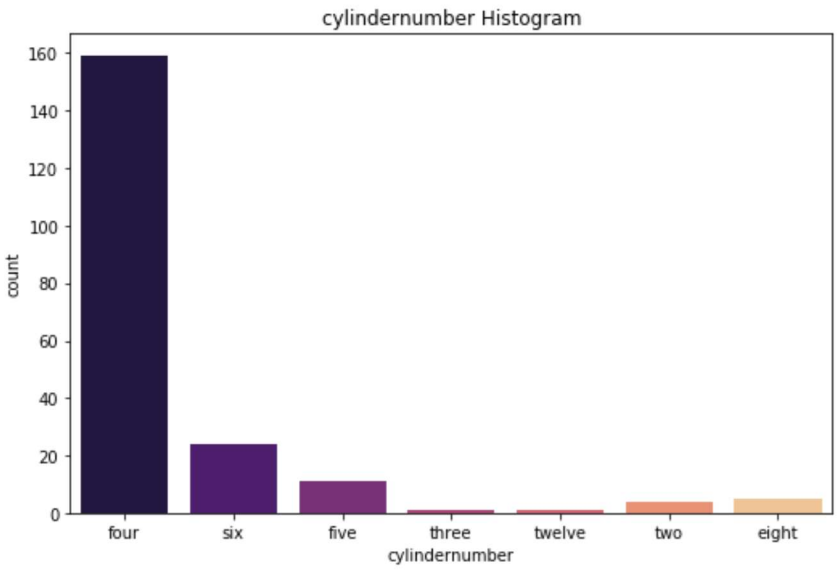
Inference :

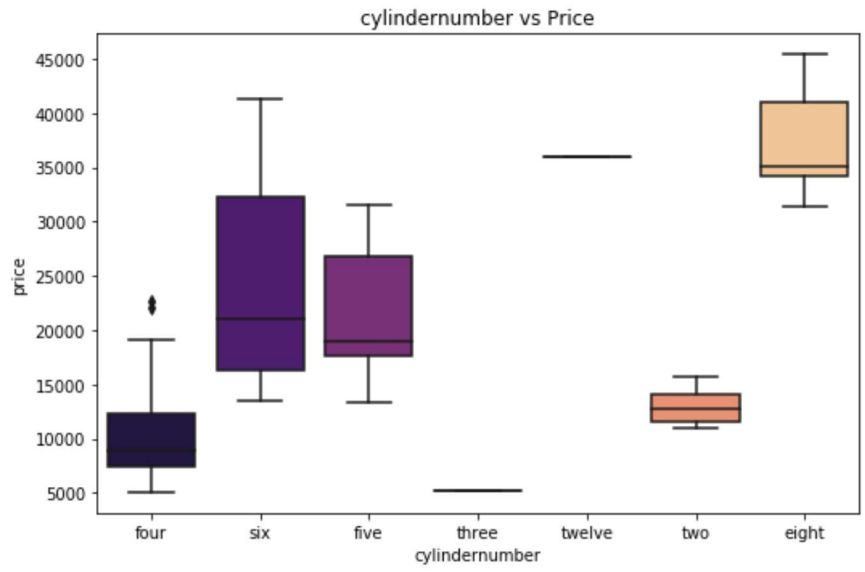
The 'doornumber' variable doesn't seem to significantly impact the price, as there's no noticeable difference between its categories.It appears that cars with turbo aspiration have a higher price range compared to those with standard aspiration, although there are some high values outside the whiskers.

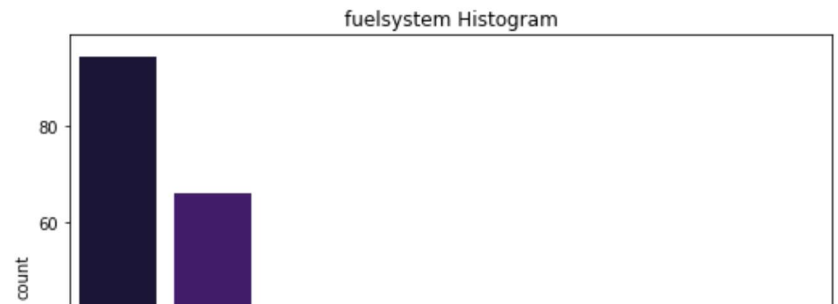
In [18]:
```
def plot_count(x,fig):
    plt.subplot(4,2,fig)
    plt.title(x+' Histogram')
    sns.countplot(cars[x],palette=("magma"))
    plt.subplot(4,2,(fig+1))
    plt.title(x+' vs Price')
    sns.boxplot(x=cars[x], y=cars.price, palette=("magma"))

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

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

plt.tight_layout()
```

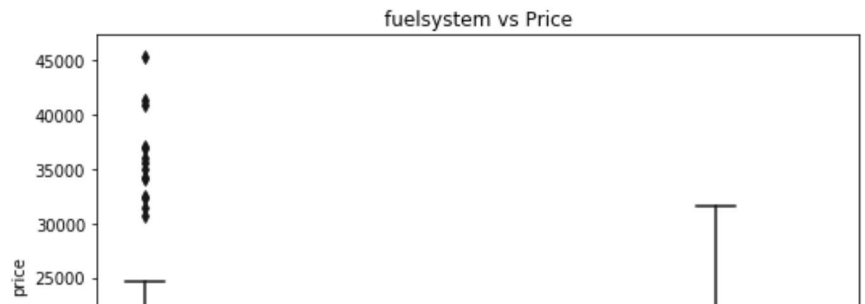


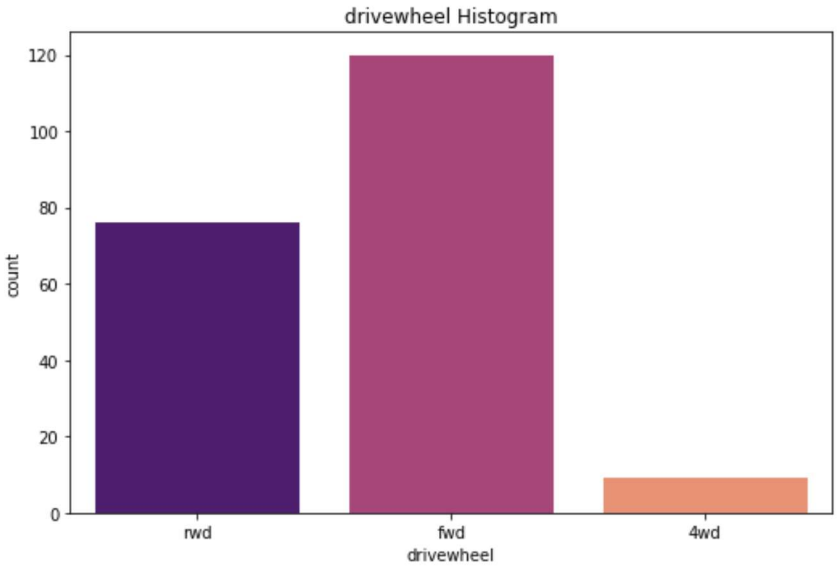


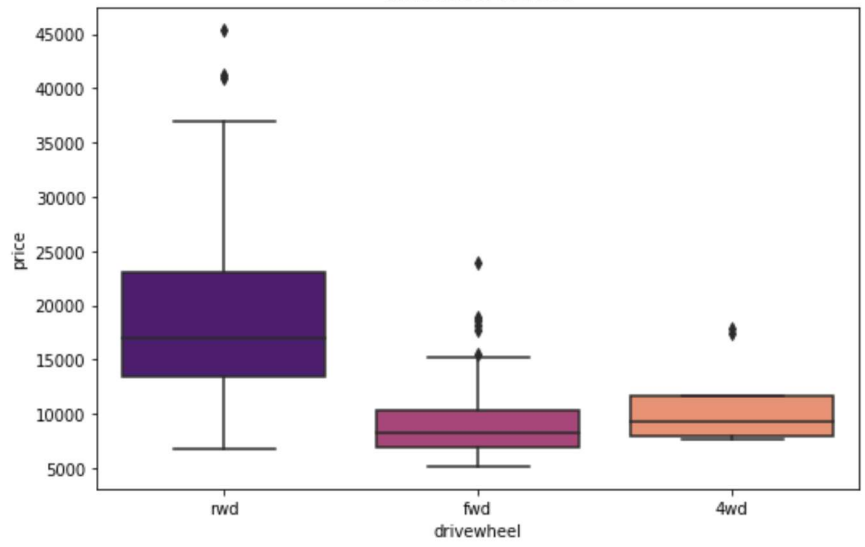












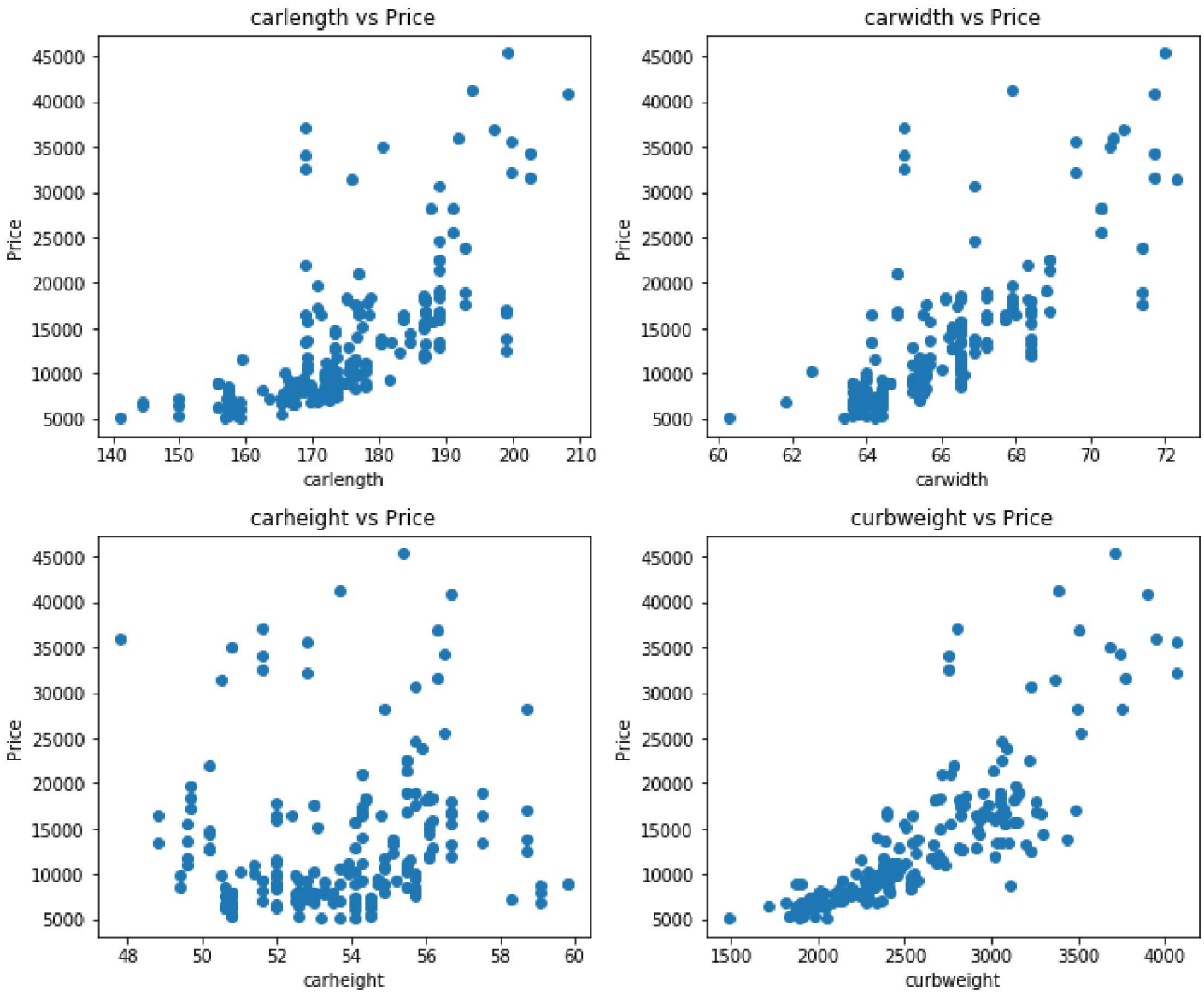
Inference :

There are very few data points for 'enginelocation' categories, making it difficult to draw inferences.The most common numbers of cylinders are four, six, and five, although eight cylinders have the highest price range.'MPFI' and '2bbl' are the most common types of fuel systems, with 'MPFI' and 'IDI' having the highest price range. However, there is insufficient data for other categories to draw meaningful inferences.There's a significant difference in the 'drivewheel' category. It seems that most high-priced cars prefer 'RWD' drivewheels.

Visualising numerical data

In [19]:
```
def scatter(x,fig):
    plt.subplot(5,2,fig)
    plt.scatter(cars[x],cars['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()
```

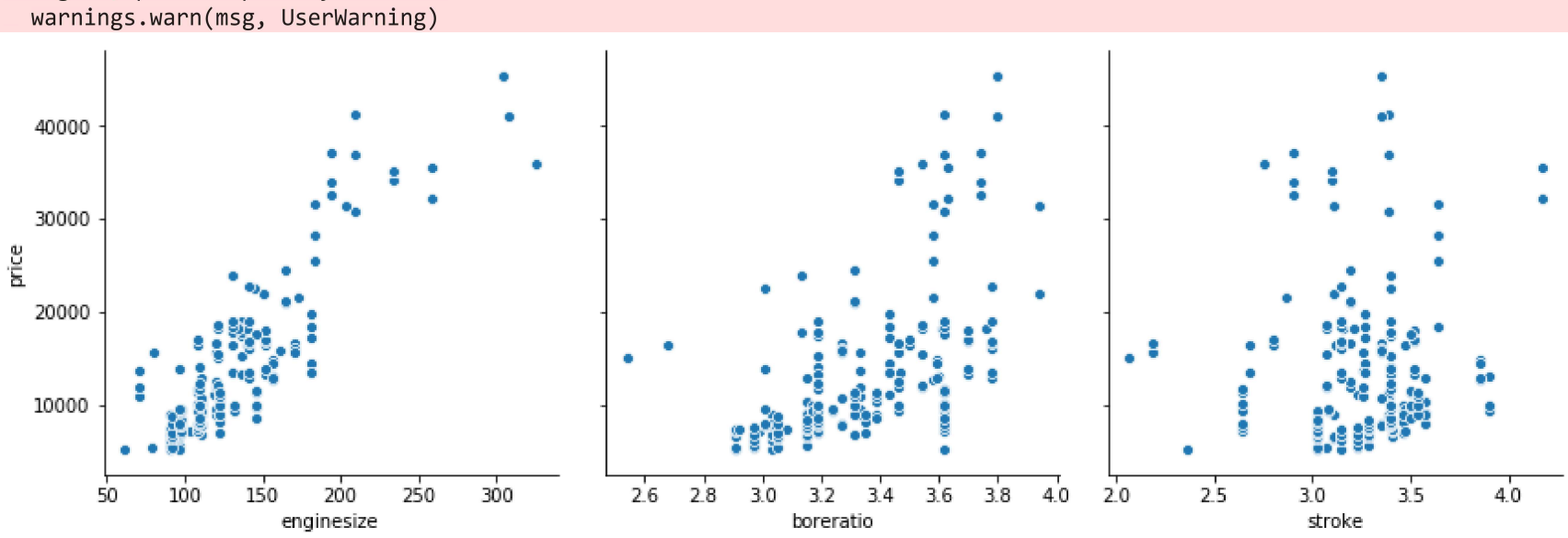


Inference :

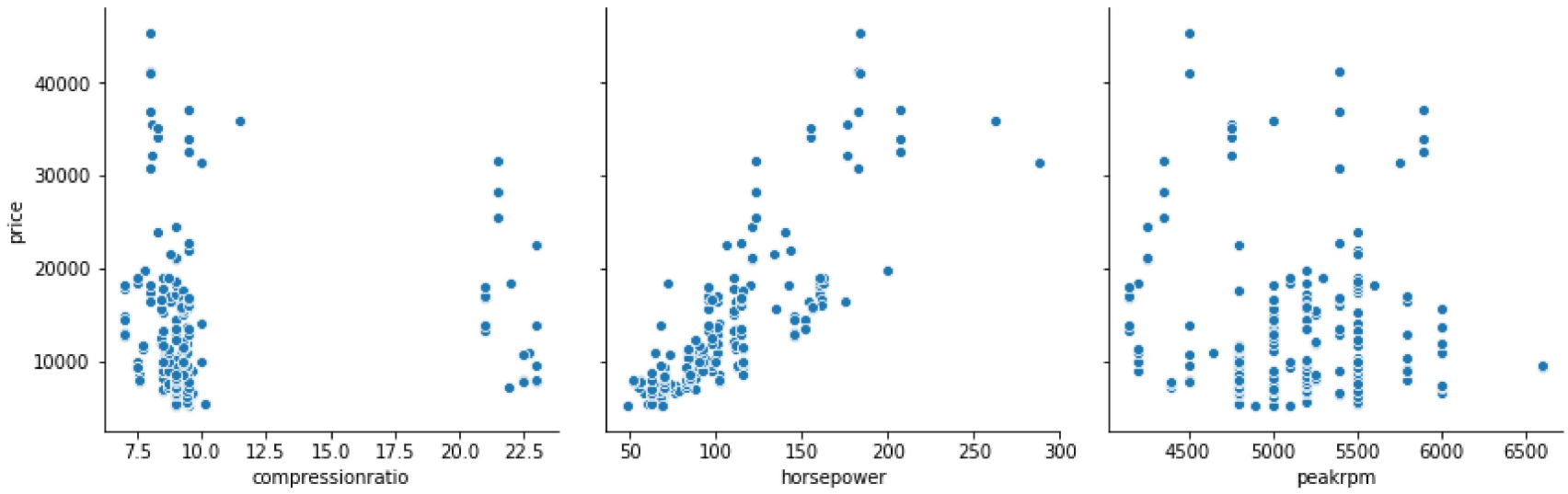
Car width, car length, and curb weight demonstrate a positive correlation with price. Conversely, car height doesn't exhibit any significant trend concerning price.

In [20]:
```
def pp(x,y,z):
    sns.pairplot(cars, x_vars=[x,y,z], y_vars='price',size=4, aspect=1, kind='scatter')
    plt.show()

pp('enginesize', 'boreratio', 'stroke')
pp('compressionratio', 'horsepower', 'peakrpm')
pp('wheelbase', 'citympg', 'highwaympg')
```

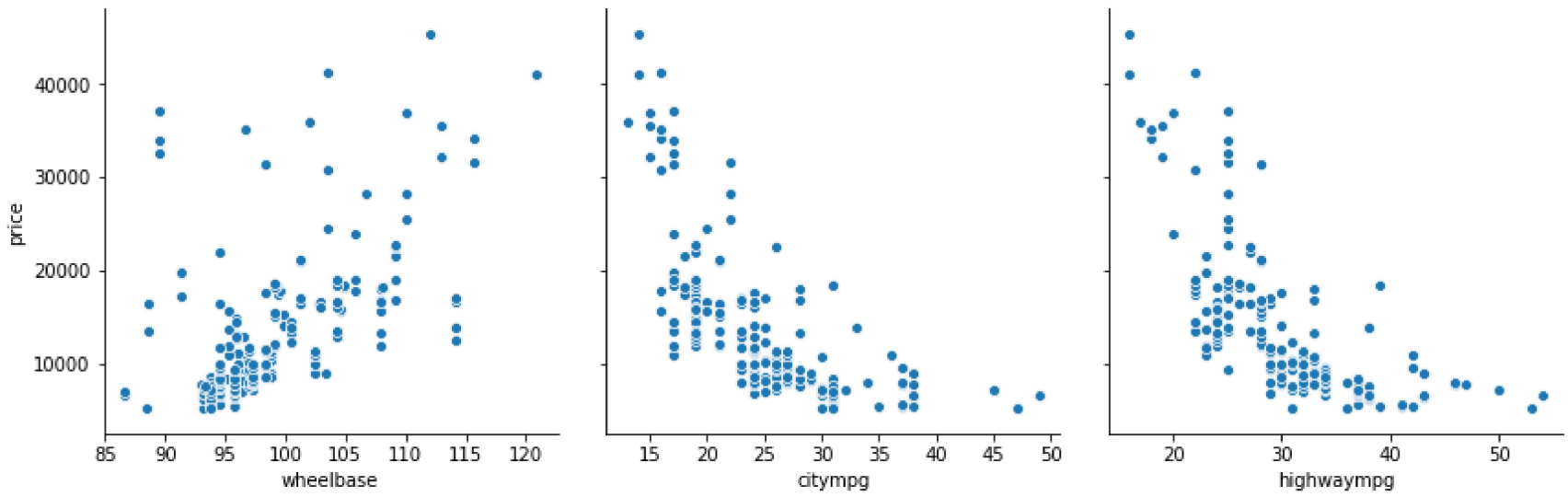
```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:2065: UserWarning: The `size` parameter has been renamed to
`height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```




```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:2065: UserWarning: The `size` parameter has been renamed to
`height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```




```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:2065: UserWarning: The `size` parameter has been renamed to
`height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```



Inference :

Engine size, bore ratio, horsepower, and wheelbase demonstrate a notable positive correlation with price. Conversely, city miles per gallon (citympg) and highway miles per gallon (highwaympg) exhibit a significant negative correlation with price.

In [21]:
```
np.corrcoef(cars['carlength'], cars['carwidth'])[0, 1]
```

Out[21]:
```
0.841118268481846
```

Deriving new features

In [22]:
```
#Fuel economy
cars['fueleconomy'] = (0.55 * cars['citympg']) + (0.45 * cars['highwaympg'])
```

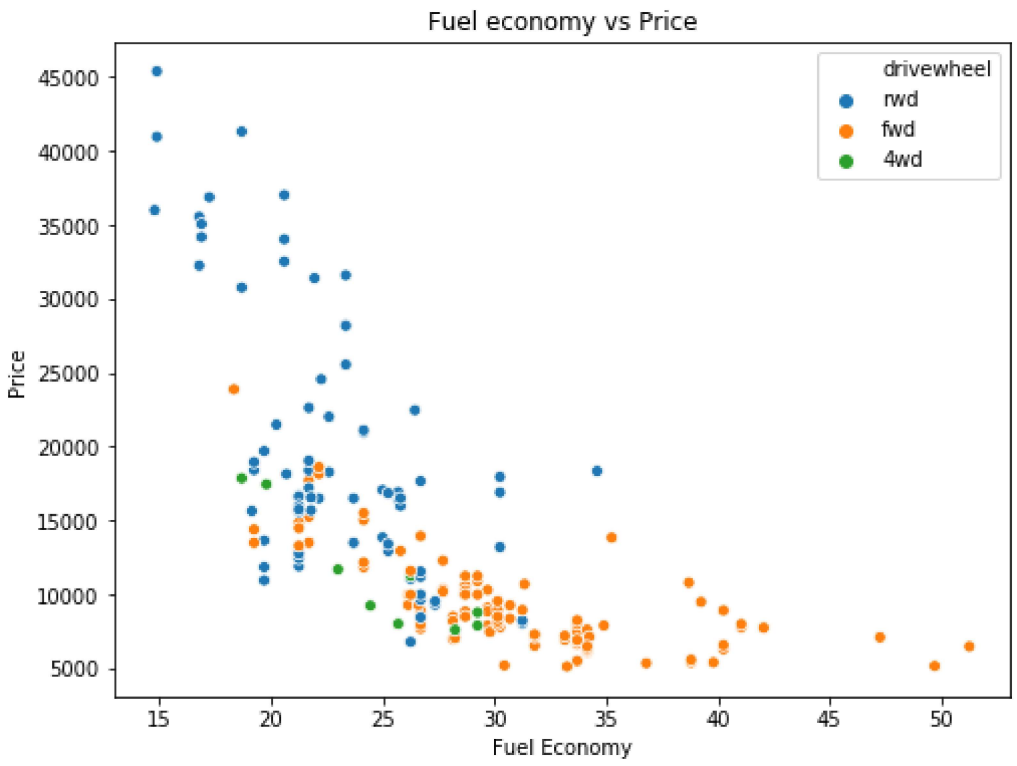
In [23]:
```
#Binning the Car Companies based on avg prices of each Company.
cars['price'] = cars['price'].astype('int')
temp = cars.copy()
table = temp.groupby(['CompanyName'])['price'].mean()
temp = temp.merge(table.reset_index(), how='left',on='CompanyName')
bins = [0,10000,20000,40000]
cars_bin=['Budget','Medium','Highend']
cars['carsrange'] = pd.cut(temp['price_y'],bins,right=False,labels=cars_bin)
cars.head()
```

Out[23]:

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

Bivariate Analysis

In [24]:
```
plt.figure(figsize=(8,6))

plt.title('Fuel economy vs Price')
sns.scatterplot(x=cars['fueleconomy'],y=cars['price'],hue=cars['drivewheel'])
plt.xlabel('Fuel Economy')
plt.ylabel('Price')

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



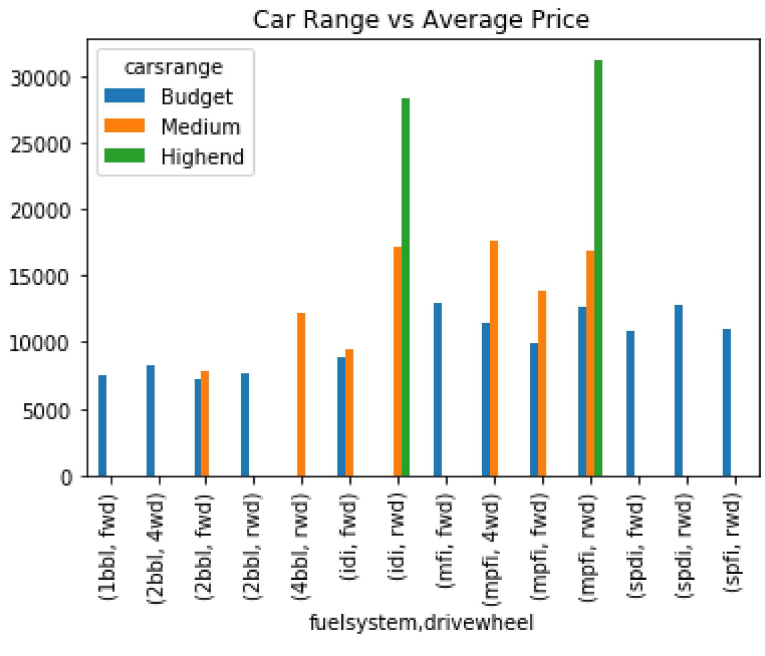

```
<Figure size 432x288 with 0 Axes>
```

Inference :

The fuel economy demonstrates an evident and significant negative correlation with price.

In [25]:
```
plt.figure(figsize=(25, 6))

df = pd.DataFrame(cars.groupby(['fuelsystem','drivewheel','carsrange'])['price'].mean().unstack(fill_value=0))
df.plot.bar()
plt.title('Car Range vs Average Price')
plt.show()
```

```
<Figure size 1800x432 with 0 Axes>
```



Inference :

Cars with higher price ranges tend to favor rear-wheel drive (RWD) drivewheels coupled with either an IDI or MPFI fuel system.

List of significant variables after Visual analysis :

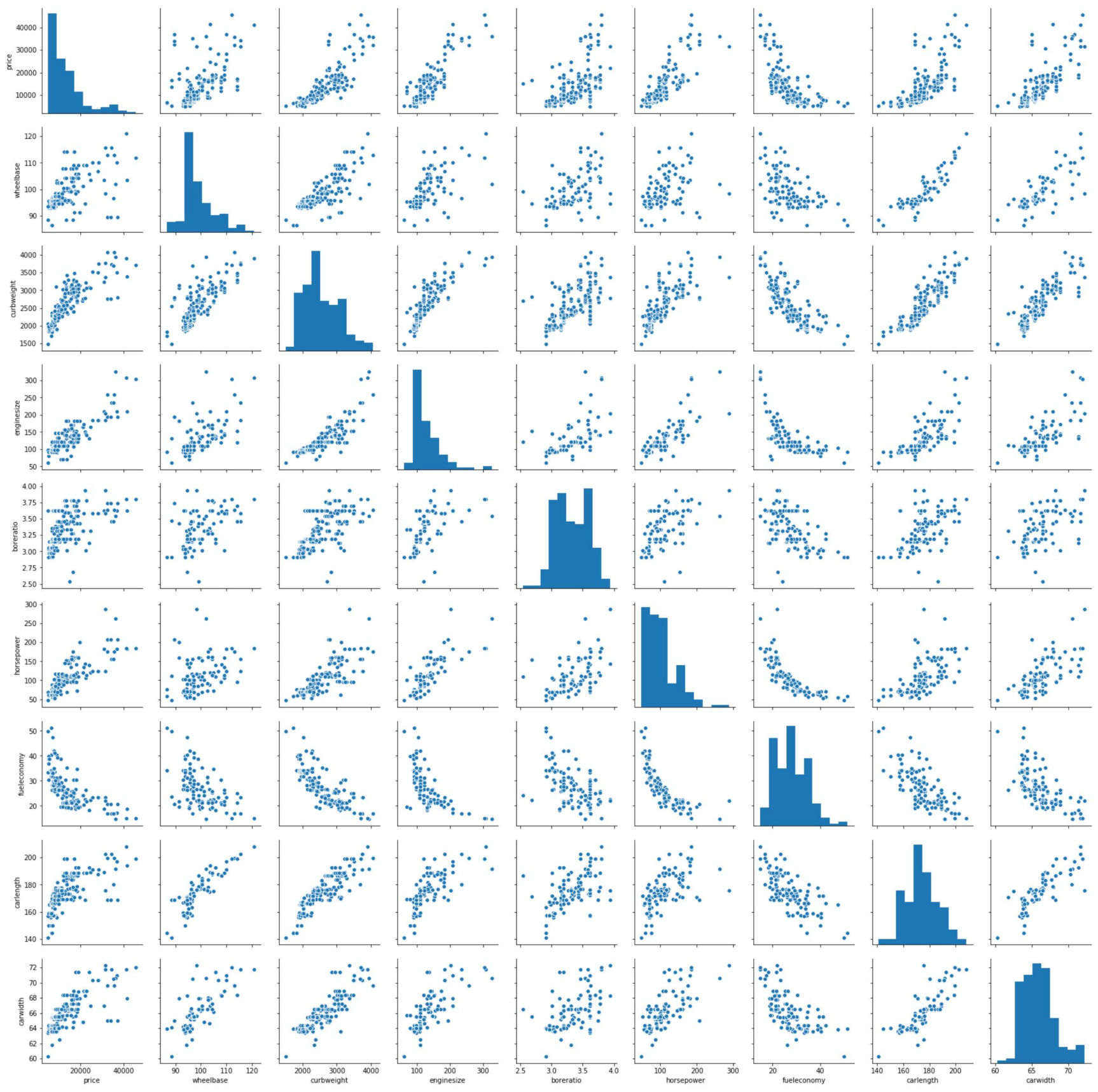
- Car Range
- Engine Type
- Fuel type
- Car Body
- Aspiration
- Cylinder Number
- Drivewheel
- Curbweight
- Car Length
- Car width
- Engine Size
- Boreratio
- Horse Power
- Wheel base
- Fuel Economy

In [26]:
```
cars_lr = cars[['price', 'fueltype', 'aspiration','carbody', 'drivewheel','wheelbase',
                'curbweight', 'enginetype', 'cylindernumber', 'enginesize', 'boreratio','horsepower',
                  'fueleconomy', 'carlength','carwidth', 'carsrange']]
cars_lr.head()
```

Out[26]:

	price	fueltype	aspiration	carbody	drivewheel	wheelbase	curbweight	enginetype	cylindernumber	enginesize	boreratio	horsepow
0	13495	gas	std	convertible	rwd	88.6	2548	dohc	four	130	3.47	11
1	16500	gas	std	convertible	rwd	88.6	2548	dohc	four	130	3.47	11
2	16500	gas	std	hatchback	rwd	94.5	2823	ohcv	six	152	2.68	15
3	13950	gas	std	sedan	fwd	99.8	2337	ohc	four	109	3.19	10
4	17450	gas	std	sedan	4wd	99.4	2824	ohc	five	136	3.19	11

In [27]:
```
sns.pairplot(cars_lr)
plt.show()
```



Dummy Variables

In [28]:
```
# Defining the map function
def dummies(x,df):
    temp = pd.get_dummies(df[x], drop_first = True)
    df = pd.concat([df, temp], axis = 1)
    df.drop([x], axis = 1, inplace = True)
    return df
# Applying the function to the cars_lr

cars_lr = dummies('fueltype',cars_lr)
cars_lr = dummies('aspiration',cars_lr)
cars_lr = dummies('carbody',cars_lr)
cars_lr = dummies('drivewheel',cars_lr)
cars_lr = dummies('enginetype',cars_lr)
cars_lr = dummies('cylindernumber',cars_lr)
cars_lr = dummies('carsrange',cars_lr)
```

In [29]:
```
cars_lr.head()
```

Out[29]:

	price	wheelbase	curbweight	enginesize	boreratio	horsepower	fueleconomy	carlength	carwidth	gas	turbo	hardtop	hatchback	se
0	13495	88.6	2548	130	3.47	111	23.70	168.8	64.1	1	0	0	0	
1	16500	88.6	2548	130	3.47	111	23.70	168.8	64.1	1	0	0	0	
2	16500	94.5	2823	152	2.68	154	22.15	171.2	65.5	1	0	0	1	
3	13950	99.8	2337	109	3.19	102	26.70	176.6	66.2	1	0	0	0	
4	17450	99.4	2824	136	3.19	115	19.80	176.6	66.4	1	0	0	0	

In [30]:
```
cars_lr.shape
```

Out[30]:
```
(205, 31)
```

Train-Test Split and Feature Scaling

In [31]:
```
from sklearn.model_selection import train_test_split

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

In [32]:
```
from sklearn.preprocessing import MinMaxScaler

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

```
/opt/conda/lib/python3.6/site-packages/sklearn/preprocessing/data.py:334: DataConversionWarning: Data with input dtype
int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [33]:
```
df_train.head()
```

Out[33]:

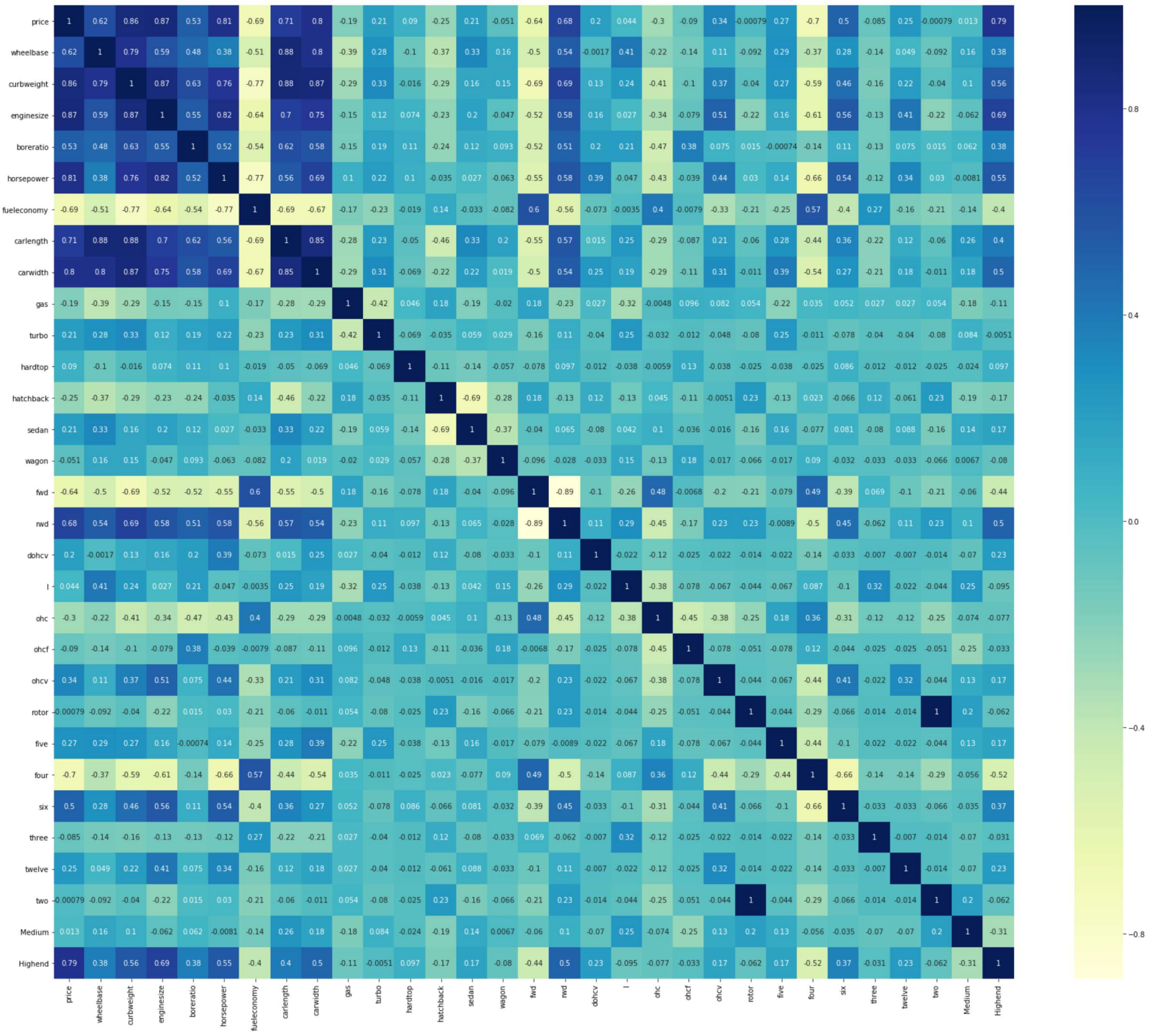
	price	wheelbase	curbweight	enginesize	boreratio	horsepower	fueleconomy	carlength	carwidth	gas	turbo	hardtop	hatchbacl
122	0.068818	0.244828	0.272692	0.139623	0.230159	0.083333	0.530864	0.426016	0.291667	1	0	0	0
125	0.466890	0.272414	0.500388	0.339623	1.000000	0.395833	0.213992	0.452033	0.666667	1	0	0	1
166	0.122110	0.272414	0.314973	0.139623	0.444444	0.266667	0.344307	0.448780	0.308333	1	0	0	1
1	0.314446	0.068966	0.411171	0.260377	0.626984	0.262500	0.244170	0.450407	0.316667	1	0	0	0
199	0.382131	0.610345	0.647401	0.260377	0.746032	0.475000	0.122085	0.775610	0.575000	1	1	0	0

In [34]:
```
df_train.describe()
```

Out[34]:

	price	wheelbase	curbweight	enginesize	boreratio	horsepower	fueleconomy	carlength	carwidth	gas	turbo
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000
mean	0.219309	0.411141	0.407878	0.241351	0.497946	0.227302	0.358265	0.525476	0.461655	0.909091	0.181818
std	0.215682	0.205581	0.211269	0.154619	0.207140	0.165511	0.185980	0.204848	0.184517	0.288490	0.387050
min	0.000000	0.000000	0.000000	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	0.198903	0.399187	0.304167	1.000000	0.000000
50%	0.140343	0.341379	0.355702	0.184906	0.500000	0.191667	0.344307	0.502439	0.425000	1.000000	0.000000
75%	0.313479	0.503448	0.559542	0.301887	0.682540	0.283333	0.512346	0.669919	0.550000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [35]:
```
#Correlation using heatmap
plt.figure(figsize = (30, 25))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



Highly correlated variables to price are - `curbweight` , `enginesize` , `horsepower` , `carwidth` and `highend` .

In [36]:
```
#Dividing data into X and y variables
y_train = df_train.pop('price')
X_train = df_train
```

Model Building

In [37]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [38]:
```
lm = LinearRegression()
lm.fit(X_train,y_train)
rfe = RFE(lm, 10)
rfe = rfe.fit(X_train, y_train)
```

In [39]:
```
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

Out[39]:
```
[('wheelbase', False, 3),
 ('curbweight', True, 1),
 ('enginesize', False, 13),
 ('boreratio', False, 10),
 ('horsepower', True, 1),
 ('fueleconomy', True, 1),
 ('carlength', False, 11),
 ('carwidth', True, 1),
 ('gas', False, 17),
 ('turbo', False, 18),
 ('hardtop', False, 2),
 ('hatchback', True, 1),
 ('sedan', True, 1),
 ('wagon', True, 1),
 ('fwd', False, 16),
 ('rwd', False, 15),
 ('dohcv', True, 1),
 ('l', False, 19),
 ('ohc', False, 7),
 ('ohcf', False, 8),
 ('ohcv', False, 9),
 ('rotor', False, 20),
 ('five', False, 6),
 ('four', False, 4),
 ('six', False, 5),
 ('three', False, 14),
 ('twelve', True, 1),
 ('two', False, 21),
 ('Medium', False, 12),
 ('Highend', True, 1)]
```

In [40]:
```
X_train.columns[rfe.support_]
```

Out[40]:
```
Index(['curbweight', 'horsepower', 'fueleconomy', 'carwidth', 'hatchback',
       'sedan', 'wagon', 'dohcv', 'twelve', 'Highend'],
      dtype='object')
```

Building model using statsmodel, for the detailed statistics

In [41]:
```
X_train_rfe = X_train[X_train.columns[rfe.support_]]
X_train_rfe.head()
```

Out[41]:

	curbweight	horsepower	fueleconomy	carwidth	hatchback	sedan	wagon	dohcv	twelve	Highend
122	0.272692	0.083333	0.530864	0.291667	0	1	0	0	0	0
125	0.500388	0.395833	0.213992	0.666667	1	0	0	0	0	1
166	0.314973	0.266667	0.344307	0.308333	1	0	0	0	0	0
1	0.411171	0.262500	0.244170	0.316667	0	0	0	0	0	0
199	0.647401	0.475000	0.122085	0.575000	0	0	1	0	0	0

In [42]:
```
def build_model(X,y):
    X = sm.add_constant(X) #Adding the constant
    lm = sm.OLS(y,X).fit() # fitting the model
    print(lm.summary()) # model summary
    return X

def checkVIF(X):
    vif = pd.DataFrame()
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return(vif)
```

MODEL 1

In [43]:
```
X_train_new = build_model(X_train_rfe,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.929
Model:                            OLS   Adj. R-squared:                  0.923
Method:                 Least Squares   F-statistic:                     172.1
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           1.29e-70
Time:                        06:45:45   Log-Likelihood:                 205.85
No. Observations:                 143   AIC:                            -389.7
Df Residuals:                     132   BIC:                            -357.1
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -0.0947      0.042     -2.243      0.027      -0.178      -0.011
curbweight      0.2657      0.069      3.870      0.000       0.130       0.402
horsepower      0.4499      0.074      6.099      0.000       0.304       0.596
fueleconomy     0.0933      0.052      1.792      0.075      -0.010       0.196
carwidth        0.2609      0.062      4.216      0.000       0.138       0.383
hatchback      -0.0929      0.025     -3.707      0.000      -0.143      -0.043
sedan          -0.0704      0.025     -2.833      0.005      -0.120      -0.021
wagon          -0.0997      0.028     -3.565      0.001      -0.155      -0.044
dohcv          -0.2676      0.079     -3.391      0.001      -0.424      -0.112
twelve         -0.1192      0.067     -1.769      0.079      -0.253       0.014
Highend         0.2586      0.020     12.929      0.000       0.219       0.298
==============================================================================
Omnibus:                       43.093   Durbin-Watson:                   1.867
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              130.648
Skew:                           1.128   Prob(JB):                     4.27e-29
Kurtosis:                       7.103   Cond. No.                         32.0
==============================================================================

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

p-vale of `twelve` seems to be higher than the significance value of 0.05, hence dropping it as it is insignificant in presence of other variables.

In [44]:
```
X_train_new = X_train_rfe.drop(["twelve"], axis = 1)
```

MODEL 2

In [45]:
```
X_train_new = build_model(X_train_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.927
Model:                            OLS   Adj. R-squared:                  0.922
Method:                 Least Squares   F-statistic:                     187.9
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           4.25e-71
Time:                        06:45:45   Log-Likelihood:                 204.17
No. Observations:                 143   AIC:                            -388.3
Df Residuals:                     133   BIC:                            -358.7
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -0.0764      0.041     -1.851      0.066      -0.158       0.005
curbweight      0.2756      0.069      3.995      0.000       0.139       0.412
horsepower      0.3997      0.069      5.824      0.000       0.264       0.535
fueleconomy     0.0736      0.051      1.435      0.154      -0.028       0.175
carwidth        0.2580      0.062      4.137      0.000       0.135       0.381
hatchback      -0.0951      0.025     -3.766      0.000      -0.145      -0.045
sedan          -0.0744      0.025     -2.983      0.003      -0.124      -0.025
wagon          -0.1050      0.028     -3.744      0.000      -0.160      -0.050
dohcv          -0.2319      0.077     -3.015      0.003      -0.384      -0.080
Highend         0.2565      0.020     12.743      0.000       0.217       0.296
==============================================================================
Omnibus:                       48.027   Durbin-Watson:                   1.880
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              159.802
Skew:                           1.231   Prob(JB):                     1.99e-35
Kurtosis:                       7.556   Cond. No.                         29.6
==============================================================================

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

In [46]:
```
X_train_new = X_train_new.drop(["fueleconomy"], axis = 1)
```

MODEL 3

In [47]:
```
X_train_new = build_model(X_train_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.926
Model:                            OLS   Adj. R-squared:                  0.922
Method:                 Least Squares   F-statistic:                     209.5
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           7.85e-72
Time:                        06:45:45   Log-Likelihood:                 203.07
No. Observations:                 143   AIC:                            -388.1
Df Residuals:                     134   BIC:                            -361.5
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0305      0.026     -1.165      0.246      -0.082       0.021
curbweight     0.2593      0.068      3.796      0.000       0.124       0.394
horsepower     0.3469      0.058      5.964      0.000       0.232       0.462
carwidth       0.2488      0.062      3.995      0.000       0.126       0.372
hatchback     -0.0922      0.025     -3.650      0.000      -0.142      -0.042
sedan         -0.0711      0.025     -2.850      0.005      -0.120      -0.022
wagon         -0.1047      0.028     -3.721      0.000      -0.160      -0.049
dohcv         -0.1968      0.073     -2.689      0.008      -0.342      -0.052
Highend        0.2610      0.020     13.083      0.000       0.222       0.301
==============================================================================
Omnibus:                       48.637   Durbin-Watson:                   1.909
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              161.444
Skew:                           1.250   Prob(JB):                     8.77e-36
Kurtosis:                       7.566   Cond. No.                         27.2
==============================================================================

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

In [48]:
```
#Calculating the Variance Inflation Factor
checkVIF(X_train_new)
```

Out[48]:

	Features	VIF
0	const	26.90
1	curbweight	8.10
5	sedan	6.07
4	hatchback	5.63
3	carwidth	5.14
2	horsepower	3.61
6	wagon	3.58
8	Highend	1.63
7	dohcv	1.46

dropping `curbweight` because of high VIF value. (shows that curbweight has high multicollinearity.)

In [49]:
```
X_train_new = X_train_new.drop(["curbweight"], axis = 1)
```

MODEL 4

In [50]:
```
X_train_new = build_model(X_train_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.918
Model:                            OLS   Adj. R-squared:                  0.914
Method:                 Least Squares   F-statistic:                     215.9
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           4.70e-70
Time:                        06:45:45   Log-Likelihood:                 195.77
No. Observations:                 143   AIC:                            -375.5
Df Residuals:                     135   BIC:                            -351.8
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0319      0.027     -1.161      0.248      -0.086       0.022
horsepower     0.4690      0.051      9.228      0.000       0.368       0.569
carwidth       0.4269      0.043      9.944      0.000       0.342       0.512
hatchback     -0.1044      0.026     -3.976      0.000      -0.156      -0.052
sedan         -0.0756      0.026     -2.896      0.004      -0.127      -0.024
wagon         -0.0865      0.029     -2.974      0.003      -0.144      -0.029
dohcv         -0.3106      0.070     -4.435      0.000      -0.449      -0.172
Highend        0.2772      0.020     13.559      0.000       0.237       0.318
==============================================================================
Omnibus:                       43.937   Durbin-Watson:                   2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              127.746
Skew:                           1.171   Prob(JB):                     1.82e-28
Kurtosis:                       6.995   Cond. No.                         18.0
==============================================================================

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

In [51]:
```
checkVIF(X_train_new)
```

Out[51]:

	Features	VIF
0	const	26.89
4	sedan	6.06
3	hatchback	5.54
5	wagon	3.47
1	horsepower	2.50
2	carwidth	2.22
7	Highend	1.56
6	dohcv	1.21

dropping `sedan` because of high VIF value.

In [52]:
```
X_train_new = X_train_new.drop(["sedan"], axis = 1)
```

MODEL 5

In [53]:
```
X_train_new = build_model(X_train_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.913
Model:                            OLS   Adj. R-squared:                  0.909
Method:                 Least Squares   F-statistic:                     237.6
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           1.68e-69
Time:                        06:45:45   Log-Likelihood:                 191.46
No. Observations:                 143   AIC:                            -368.9
Df Residuals:                     136   BIC:                            -348.2
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0934      0.018     -5.219      0.000      -0.129      -0.058
horsepower     0.5001      0.051      9.805      0.000       0.399       0.601
carwidth       0.3963      0.043      9.275      0.000       0.312       0.481
hatchback     -0.0373      0.013     -2.938      0.004      -0.062      -0.012
wagon         -0.0170      0.017     -1.008      0.315      -0.050       0.016
dohcv         -0.3203      0.072     -4.460      0.000      -0.462      -0.178
Highend        0.2808      0.021     13.402      0.000       0.239       0.322
==============================================================================
Omnibus:                       34.143   Durbin-Watson:                   2.024
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               72.788
Skew:                           1.018   Prob(JB):                     1.56e-16
Kurtosis:                       5.841   Cond. No.                         16.4
==============================================================================

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

In [54]:
```
checkVIF(X_train_new)
```

Out[54]:

	Features	VIF
0	const	10.82
1	horsepower	2.39
2	carwidth	2.09
6	Highend	1.55
3	hatchback	1.23
5	dohcv	1.21
4	wagon	1.11

dropping `wagon` because of high p-value.

In [55]:
```
X_train_new = X_train_new.drop(["wagon"], axis = 1)
```

MODEL 6

In [56]:
```
X_train_new = build_model(X_train_new,y_train)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.912
Model:                            OLS   Adj. R-squared:                  0.909
Method:                 Least Squares   F-statistic:                     284.8
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           1.57e-70
Time:                        06:45:45   Log-Likelihood:                 190.93
No. Observations:                 143   AIC:                            -369.9
Df Residuals:                     137   BIC:                            -352.1
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0970      0.018     -5.530      0.000      -0.132      -0.062
horsepower     0.5013      0.051      9.832      0.000       0.401       0.602
carwidth       0.3952      0.043      9.252      0.000       0.311       0.480
hatchback     -0.0336      0.012     -2.764      0.006      -0.058      -0.010
dohcv         -0.3231      0.072     -4.502      0.000      -0.465      -0.181
Highend        0.2833      0.021     13.615      0.000       0.242       0.324
==============================================================================
Omnibus:                       36.097   Durbin-Watson:                   2.028
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               78.717
Skew:                           1.067   Prob(JB):                     8.07e-18
Kurtosis:                       5.943   Cond. No.                         16.3
==============================================================================

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

In [57]:
```
checkVIF(X_train_new)
```

Out[57]:

	Features	VIF
0	const	10.39
1	horsepower	2.39
2	carwidth	2.08
5	Highend	1.53
4	dohcv	1.21
3	hatchback	1.13

MODEL 7

In [58]:
```
#Dropping dohcv to see the changes in model statistics
X_train_new = X_train_new.drop(["dohcv"], axis = 1)
X_train_new = build_model(X_train_new,y_train)
checkVIF(X_train_new)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.899
Model:                            OLS   Adj. R-squared:                  0.896
Method:                 Least Squares   F-statistic:                     308.0
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           1.04e-67
Time:                        06:45:45   Log-Likelihood:                 181.06
No. Observations:                 143   AIC:                            -352.1
Df Residuals:                     138   BIC:                            -337.3
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0824      0.018     -4.480      0.000      -0.119      -0.046
horsepower     0.4402      0.052      8.390      0.000       0.336       0.544
carwidth       0.3957      0.046      8.677      0.000       0.306       0.486
hatchback     -0.0414      0.013     -3.219      0.002      -0.067      -0.016
Highend        0.2794      0.022     12.591      0.000       0.236       0.323
==============================================================================
Omnibus:                       29.385   Durbin-Watson:                   1.955
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               98.010
Skew:                           0.692   Prob(JB):                     5.22e-22
Kurtosis:                       6.812   Cond. No.                         12.9
==============================================================================

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

Out[58]:

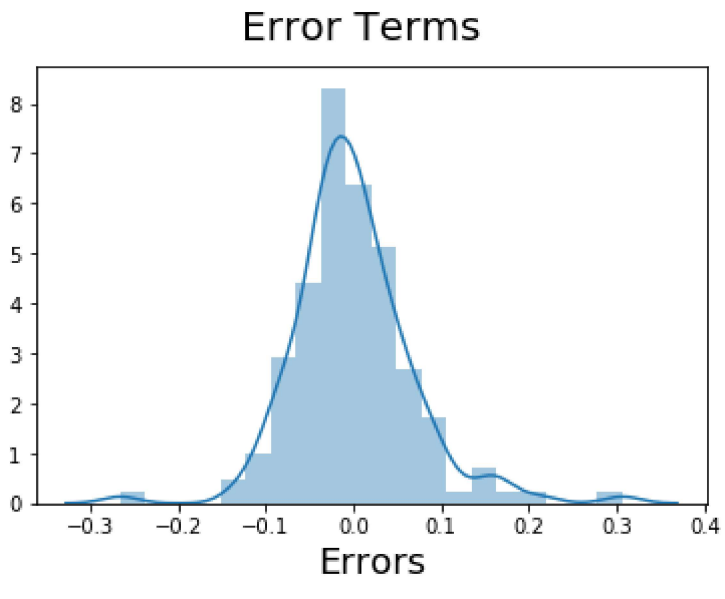
	Features	VIF
0	const	10.04
1	horsepower	2.22
2	carwidth	2.08
4	Highend	1.53
3	hatchback	1.10

Residual Analysis of Model

In [59]:
```
lm = sm.OLS(y_train,X_train_new).fit()
y_train_price = lm.predict(X_train_new)
```

In [60]:
```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)
```

Out[60]:
```
Text(0.5, 0, 'Errors')
```



Error terms seem to be approximately normally distributed, so the assumption on the linear modeling seems to be fulfilled.

Prediction and Evaluation

In [61]:
```
#Scaling the test set
num_vars = ['wheelbase', 'curbweight', 'enginesize', 'boreratio', 'horsepower','fueleconomy','carlength','carwidth','pr
df_test[num_vars] = scaler.fit_transform(df_test[num_vars])
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/preprocessing/data.py:334: DataConversionWarning: Data with input dtype
int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [62]:
```
#Dividing into X and y
y_test = df_test.pop('price')
X_test = df_test
```

In [63]:
```
# Now let's use our model to make predictions.
X_train_new = X_train_new.drop('const',axis=1)
# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test[X_train_new.columns]

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)
```

In [64]:
```
# Making predictions
y_pred = lm.predict(X_test_new)
```

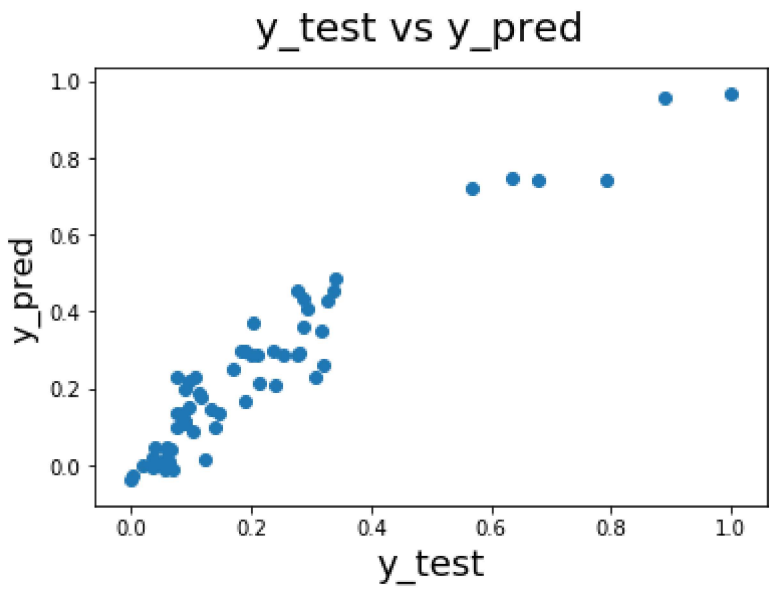
Evaluation of test via comparison of y_pred and y_test

In [65]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[65]:
```
0.8614595209022036
```

In [66]:
```
#EVALUATION OF THE MODEL
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)               # Plot heading
plt.xlabel('y_test', fontsize=18)                           # X-label
plt.ylabel('y_pred', fontsize=16)
```

Out[66]:
```
Text(0, 0.5, 'y_pred')
```



Evaluation of the model using Statistics

In [67]:
```
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.899
Model:                            OLS   Adj. R-squared:                  0.896
Method:                 Least Squares   F-statistic:                     308.0
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           1.04e-67
Time:                        06:45:46   Log-Likelihood:                 181.06
No. Observations:                 143   AIC:                            -352.1
Df Residuals:                     138   BIC:                            -337.3
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0824      0.018     -4.480      0.000      -0.119      -0.046
horsepower     0.4402      0.052      8.390      0.000       0.336       0.544
carwidth       0.3957      0.046      8.677      0.000       0.306       0.486
hatchback     -0.0414      0.013     -3.219      0.002      -0.067      -0.016
Highend        0.2794      0.022     12.591      0.000       0.236       0.323
==============================================================================
Omnibus:                       29.385   Durbin-Watson:                   1.955
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               98.010
Skew:                           0.692   Prob(JB):                     5.22e-22
Kurtosis:                       6.812   Cond. No.                         12.9
==============================================================================

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

Inference :

The R-squared and Adjusted R-squared values, measuring the extent of fit, stand at 0.899 and 0.896 respectively, explaining approximately 90% of the variance.

The F-statistics and Prob(F-statistics) are calculated as 308.0 and approximately 0.0 (1.04e-67), signifying a highly significant model fit. This strong fit, explaining 90% variance, is unlikely to have occurred by chance.

Regarding the p-values, all coefficients' p-values appear to be lower than the significance level of 0.05. This suggests that all predictors are statistically significant