4.1 Pricing Analysis for Cars - Week 4 - DSC680 - Binay Jena (contd. from Week3)

STEP 1 - Get Libraries & Read CSV

#1.1 : - import the required python libraries

In [590…
```
import warnings
warnings.filterwarnings('ignore')

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

#1.2 : - read the datatset using pandas - understand the startcure/ layout of the data

In [591…
```
cars = pd.read_csv('/Users/bjena/Documents/GitHub/dsc680/CarPrice_Assignment.cs
cars.head()
```

Out[591]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd
3	4	2	audi 100 ls	gas	std	four	sedan	fwd
4	5	2	audi 100ls	gas	std	four	sedan	4wd

5 rows × 26 columns

In [592…
```
cars.shape
```

Out[592]: (205, 26)

#1.3 : - get basic statistics of the dataset & columns

In [593…
```
cars.describe()
```

Out[593]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweig
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.5658
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.6802
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.0000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.0000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.0000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.0000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.0000

#1.4 : - data structure details (schema)

In [594…

```
cars.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
```

STEP 2 - Prep & Cleanse

#2.1 : - make CarName column more relevant - remove variant details, use manufacturer and model

In [595…

```
CompanyName = cars['CarName'].apply(lambda x : x.split(' ')[0])
```

```
cars.insert(3,"CompanyName",CompanyName)
cars.drop(['CarName'],axis=1,inplace=True)
cars.head()
```

Out[595]:

	car_ID	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewhe
0	1	3	alfa-romero	gas	std	two	convertible	rw
1	2	3	alfa-romero	gas	std	two	convertible	rw
2	3	1	alfa-romero	gas	std	two	hatchback	rw
3	4	2	audi	gas	std	four	sedan	fw
4	5	2	audi	gas	std	four	sedan	4w

5 rows × 26 columns

#2.2 : - cars/ automakers in scope

In [596…

```
cars.CompanyName.unique()
```

Out[596]:

```
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)
```

#2.3 : - fix spelling errors in CompanyName

In [597…

```
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[597]:

```
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 [598…

```
#2.4 : Checking for duplicates & NULLs
#2.4.1
cars.loc[cars.duplicated()]
```

Out[598]:

car_ID	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel

0 rows × 26 columns

In [599…

```
#2.4.2 duplicate check2
sum(cars.duplicated(subset = 'car_ID')) == 0
```

Out[599]:
```
True
```

In [600…
```
#2.4.3 Null value check
cars.isnull().sum()*100/cars.shape[0]
```

Out[600]:
```
car_ID              0.0
symboling           0.0
CompanyName         0.0
fueltype            0.0
aspiration          0.0
doornumber          0.0
carbody             0.0
drivewheel          0.0
enginelocation      0.0
wheelbase           0.0
carlength           0.0
carwidth            0.0
carheight           0.0
curbweight          0.0
enginetype          0.0
cylindernumber      0.0
enginesize          0.0
fuelsystem          0.0
boreratio           0.0
stroke              0.0
compressionratio    0.0
horsepower          0.0
peakrpm             0.0
citympg             0.0
highwaympg          0.0
price               0.0
dtype: float64
```

In [601…
```
# 2.4.1 & 2.4.2 No duplicate values
# 2.4.3 There are no NULL values in the dataset, hence dataset seems clean
```

In [602…
```
cars.columns
```

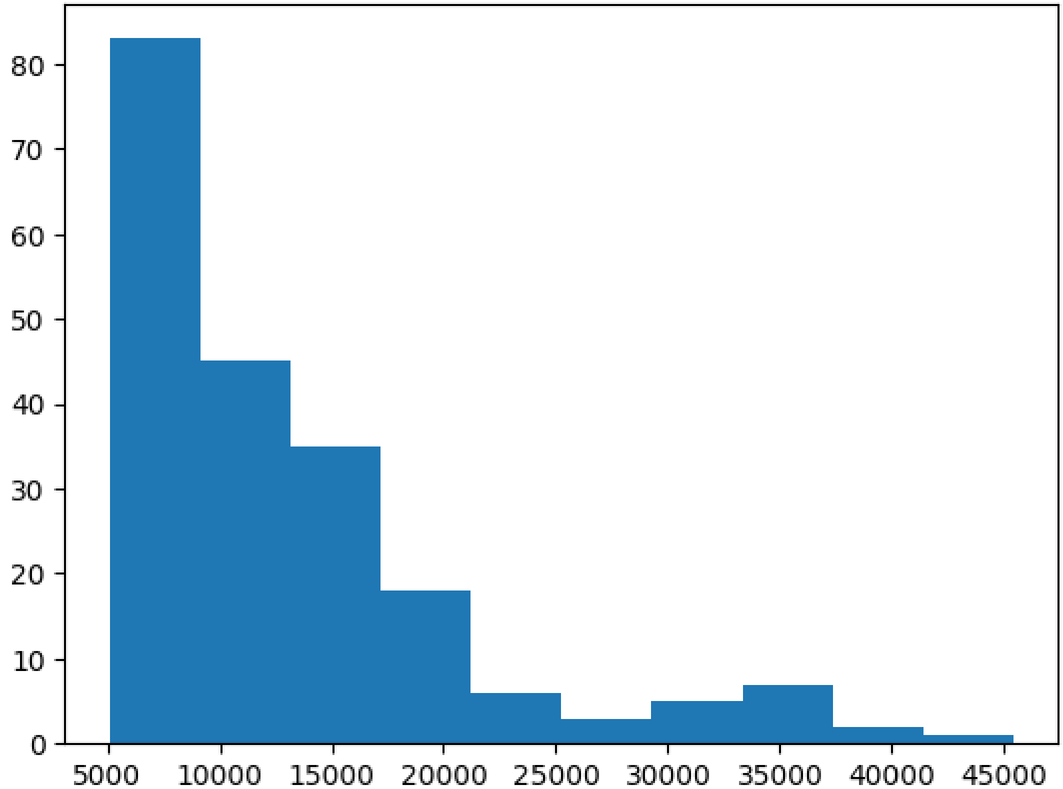
Out[602]:
```
Index(['car_ID', 'symboling', 'CompanyName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

STEP3: Data Visualization / Exploratory Analysis

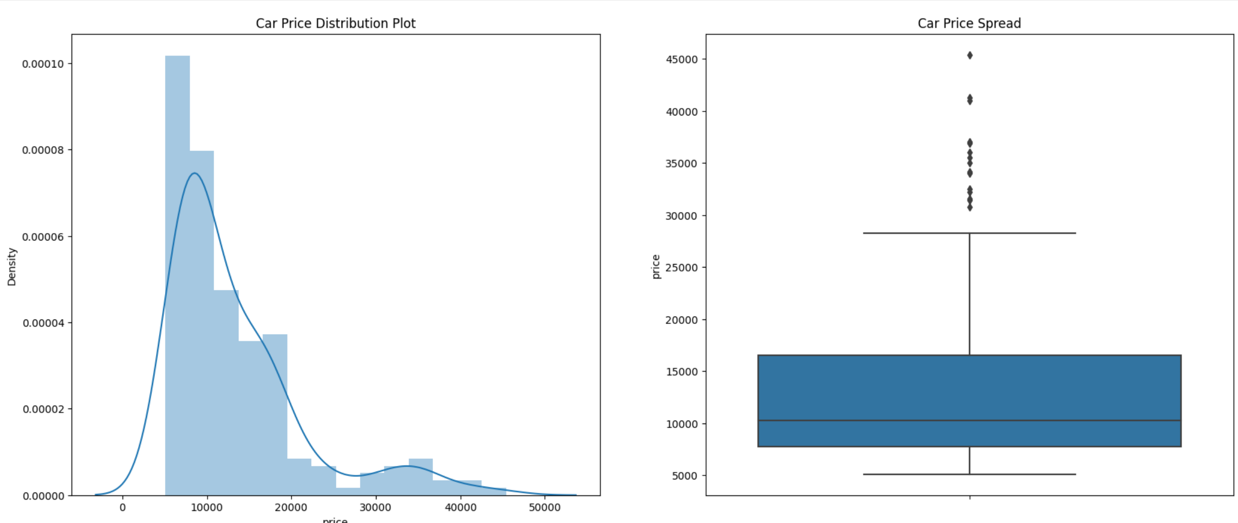
#3.1: price spread - price distribution plot

In [603…
```
#price distribution of different car names
plt.hist(cars.price)
```

Out[603]:
```
(array([83., 45., 35., 18.,  6.,  3.,  5.,  7.,  2.,  1.]),
 array([ 5118. ,  9146.2, 13174.4, 17202.6, 21230.8, 25259. , 29287.2,
        33315.4, 37343.6, 41371.8, 45400. ]),
 <BarContainer object of 10 artists>)
```



In [604…
```
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()
```



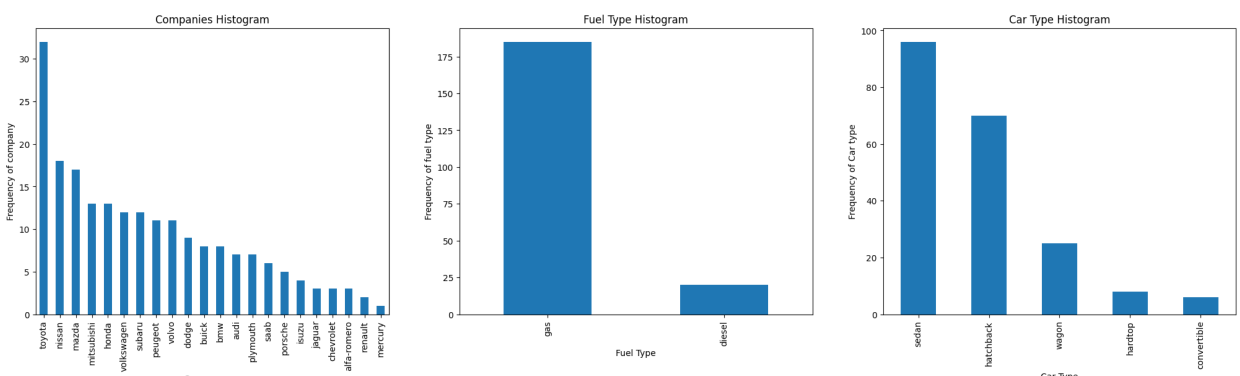
In [605…
```
print(cars.price.describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1]))
```

```
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
```

#3.2 : visualize categorical data

In [606…

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

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

plt.show()
```



lets symbol the categories

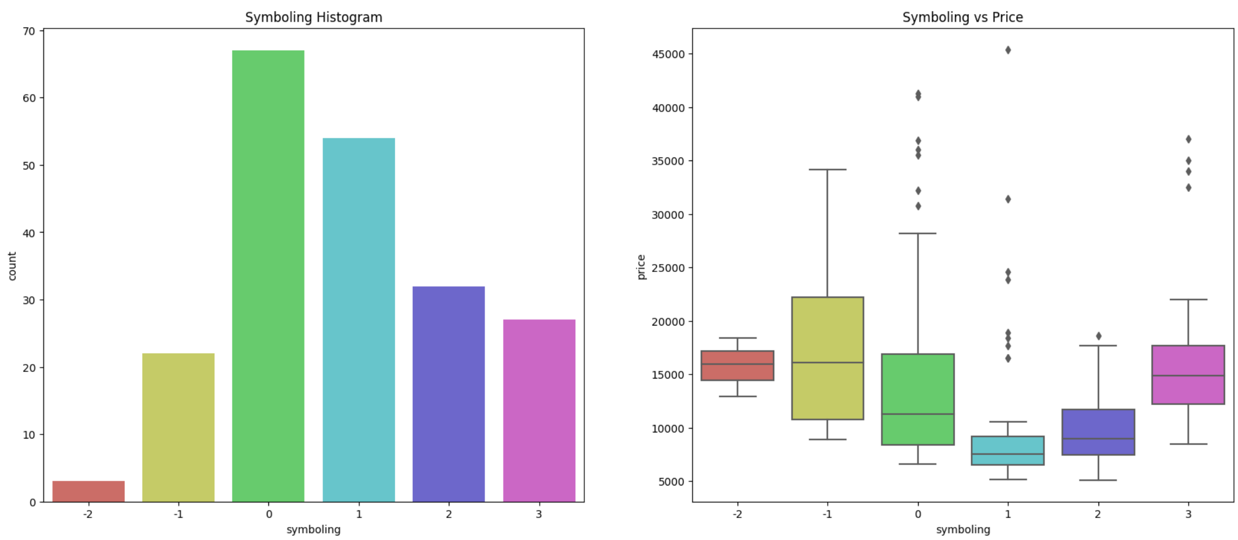
symboling : Its assigned insurance risk rating A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.(Categorical)

In [607…

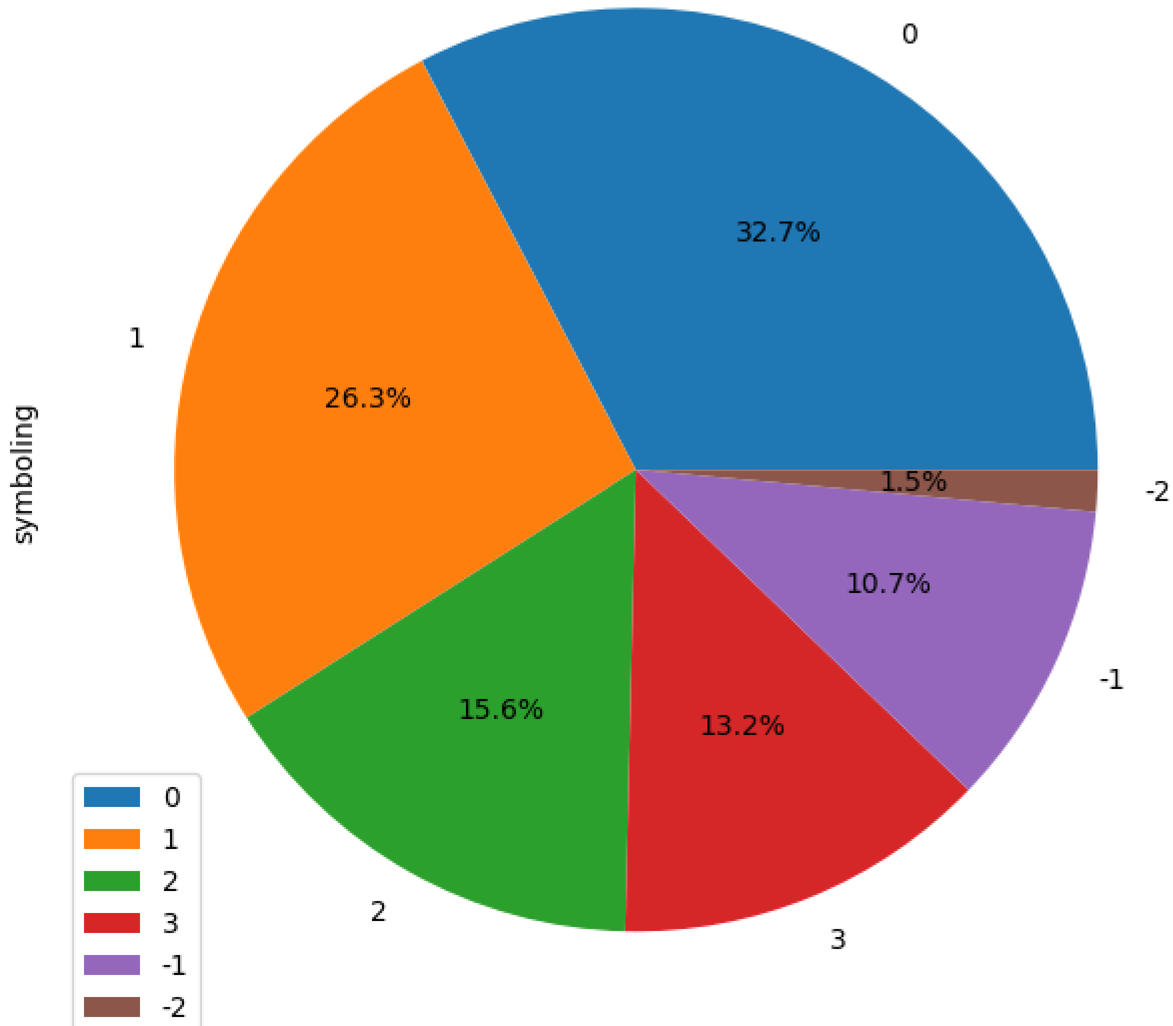
```
cars.symboling
```

Out[607]:
```
0      3
1      3
2      1
3      2
4      2
      ..
200   -1
201   -1
202   -1
203   -1
204   -1
Name: symboling, Length: 205, dtype: int64
```

In [608…
```
# count of automobile in each category and percent share of each category.
plt.figure(figsize=(20,8))

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

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

plt.show()
```



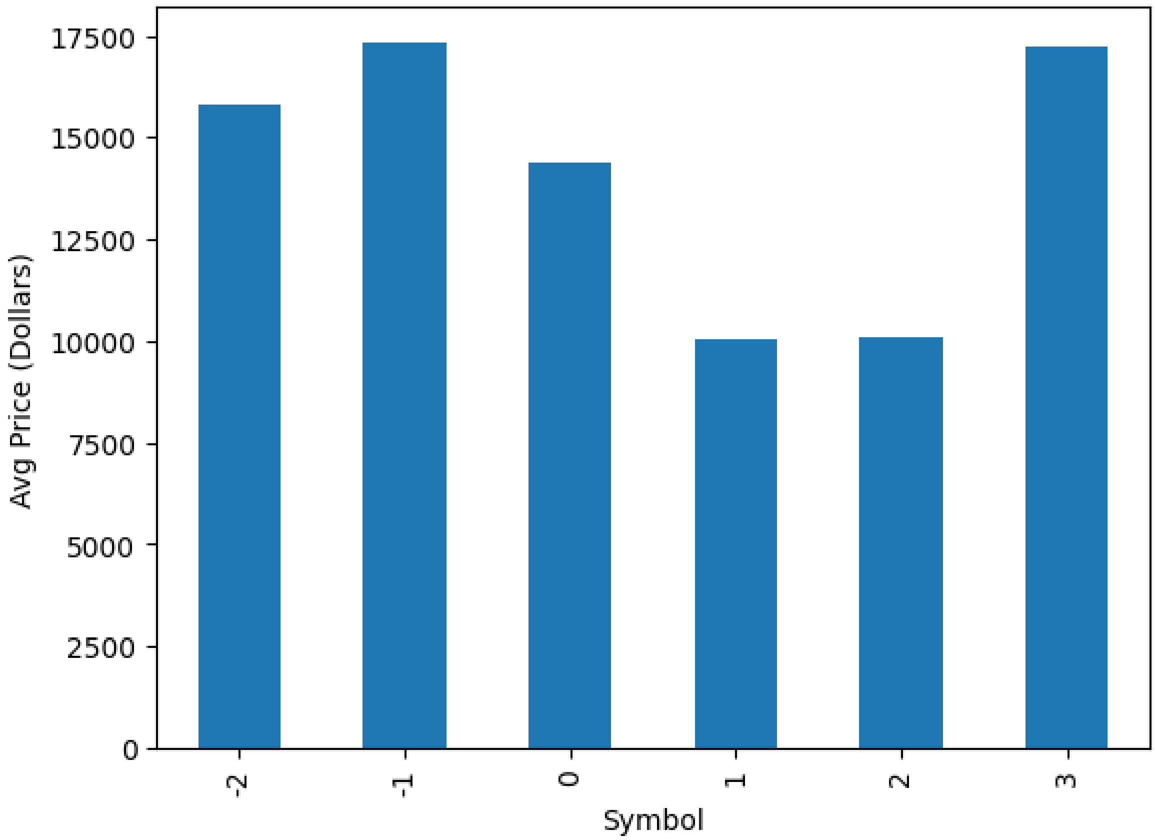
In [609…
```
car_sym = pd.DataFrame(cars['symboling'].value_counts())
car_sym.plot.pie(subplots=True,labels = car_sym.index.values, autopct='%1.1f%%'
# Unsquish the pie.
plt.gca().set_aspect('equal')
plt.show()
plt.tight_layout()
```




```
<Figure size 640x480 with 0 Axes>
```

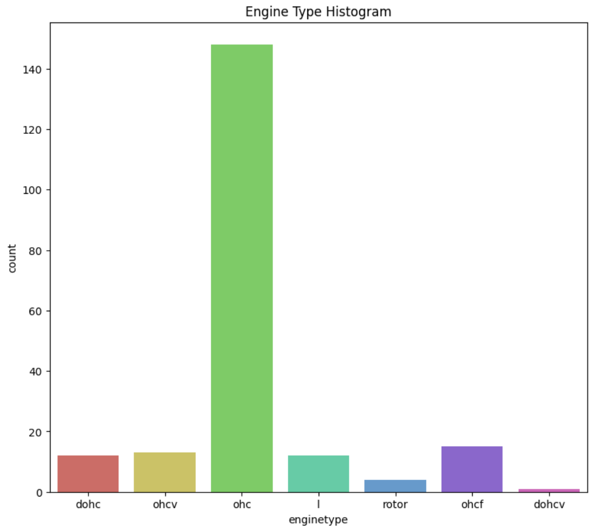
In [610…

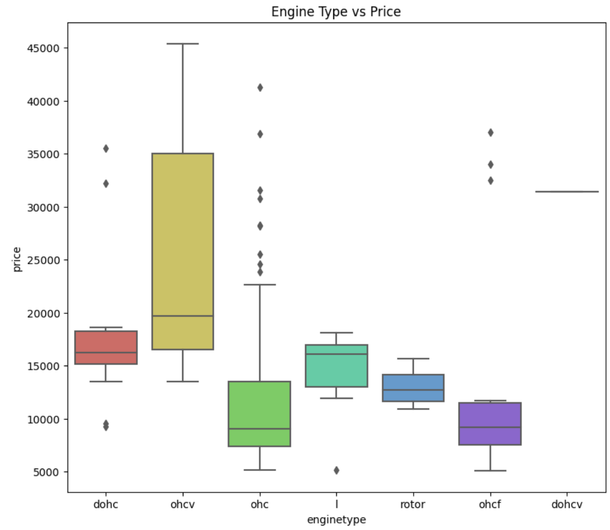
```
# Let's see average price of cars in each symbol category.
plt1 = cars[['symboling','price']].groupby("symboling").mean().plot(kind='bar',
plt1.set_xlabel("Symbol")
plt1.set_ylabel("Avg Price (Dollars)")
#xticks(rotation = 0)
plt.show()
```

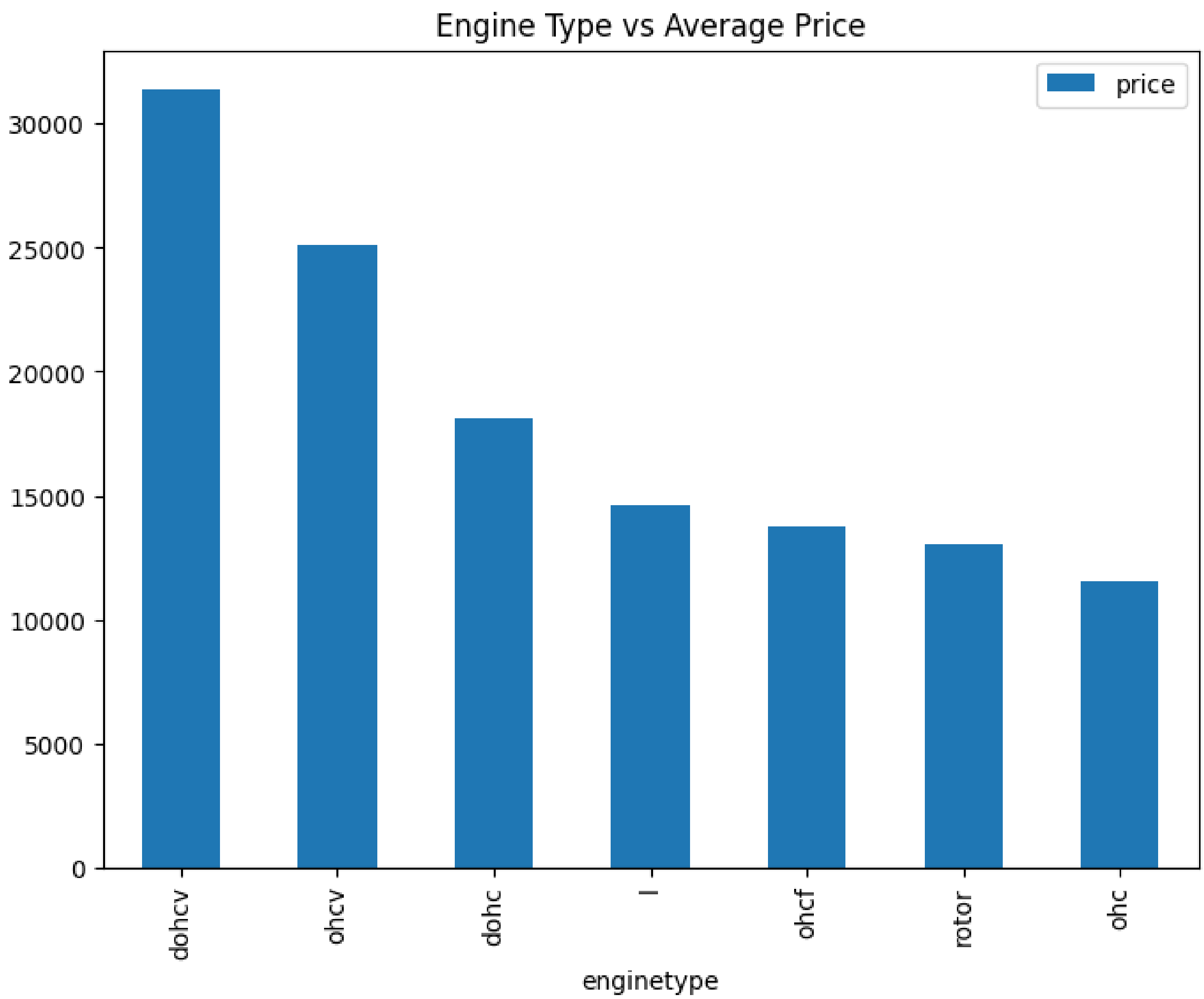


In [611…

```
#engine type vs price
plt.figure(figsize=(20,8))

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

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

plt.show()

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





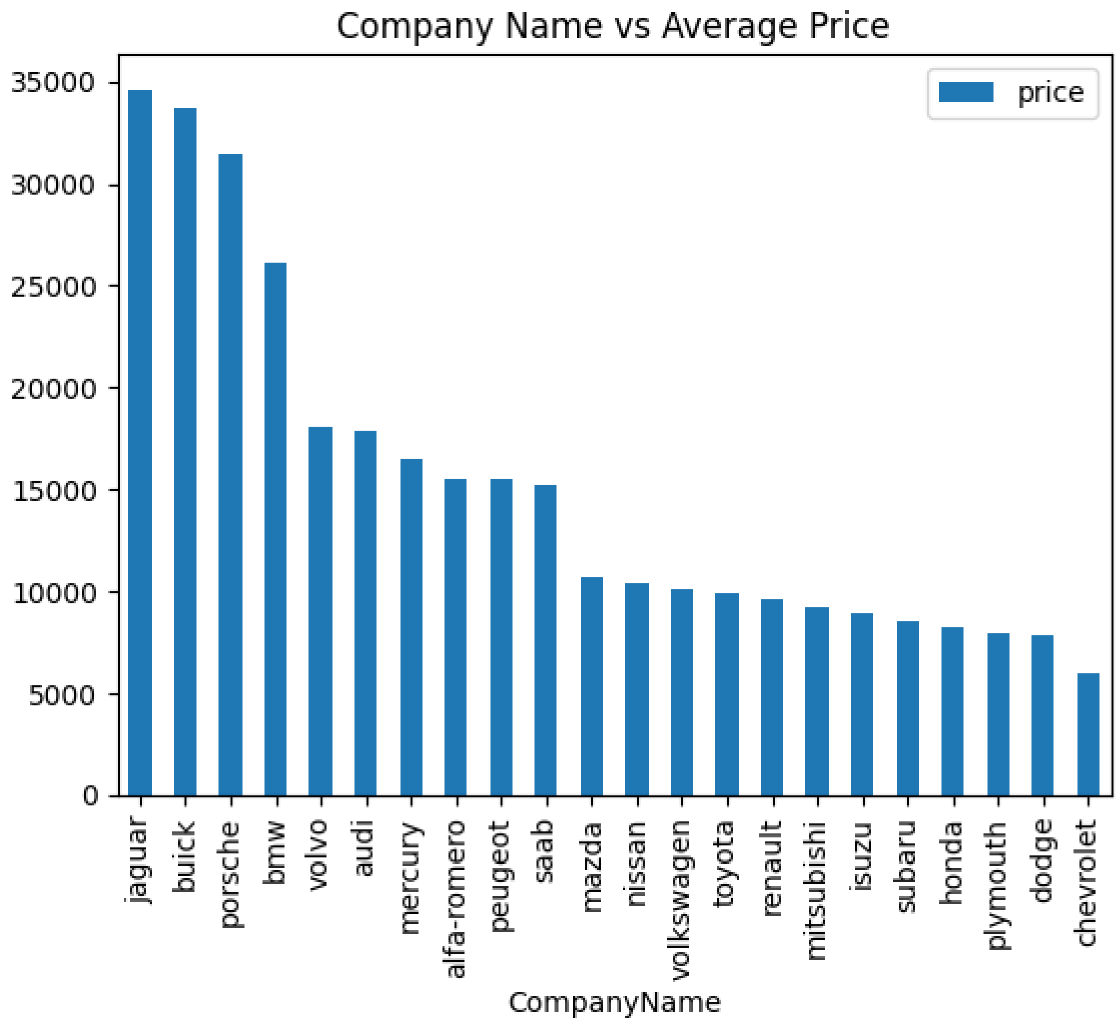


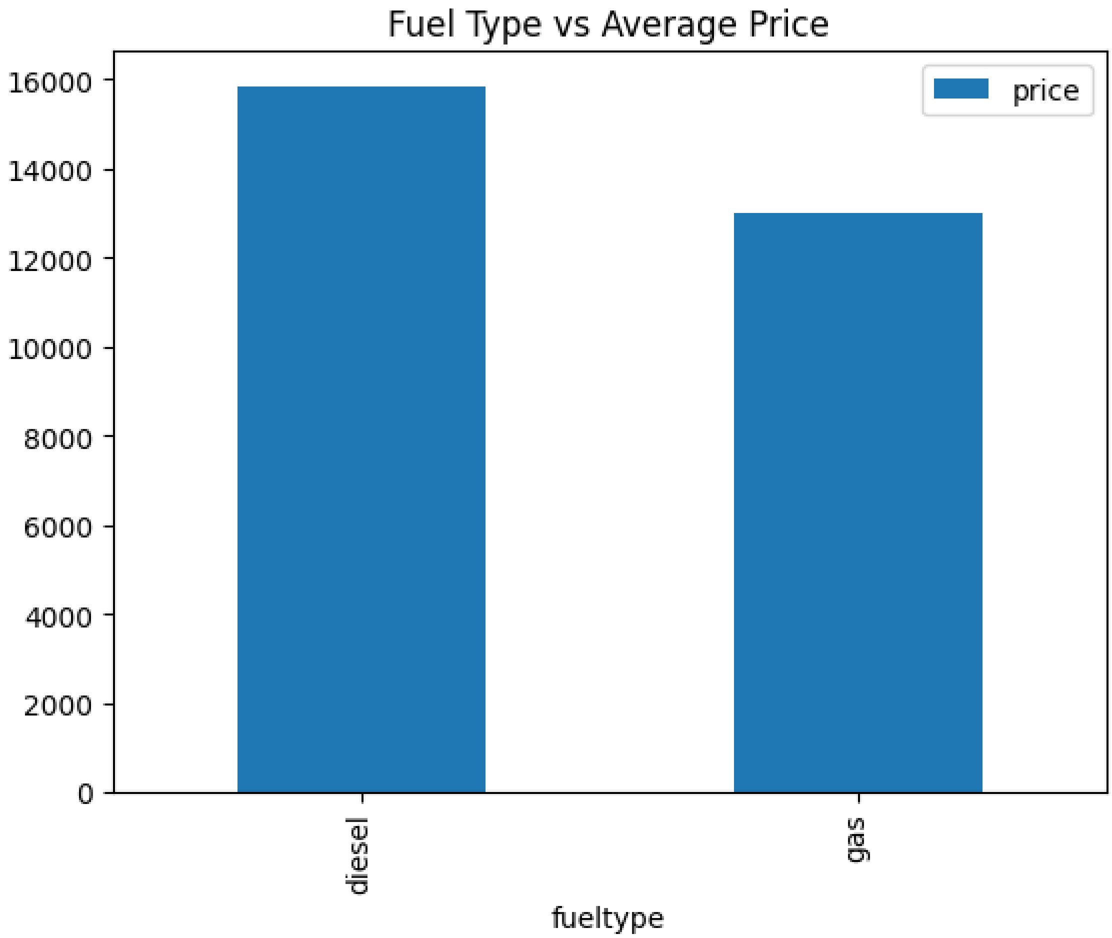
In [612…

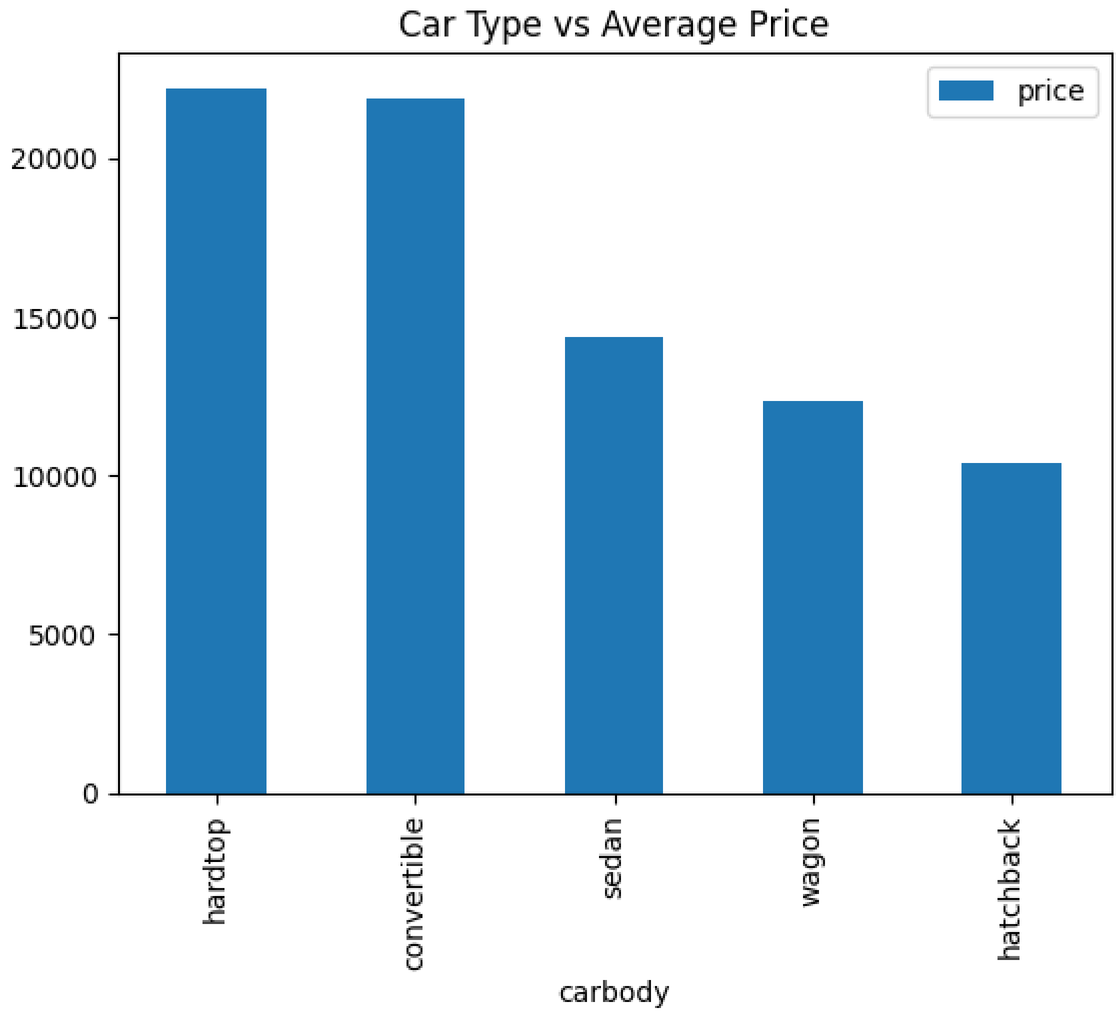
```
#features to price
plt.figure(figsize=(25, 6))

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

df = pd.DataFrame(cars.groupby(['fueltype'])['price'].mean().sort_values(ascenc
df.plot.bar()
```

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

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

```
<Figure size 2500x600 with 0 Axes>
```

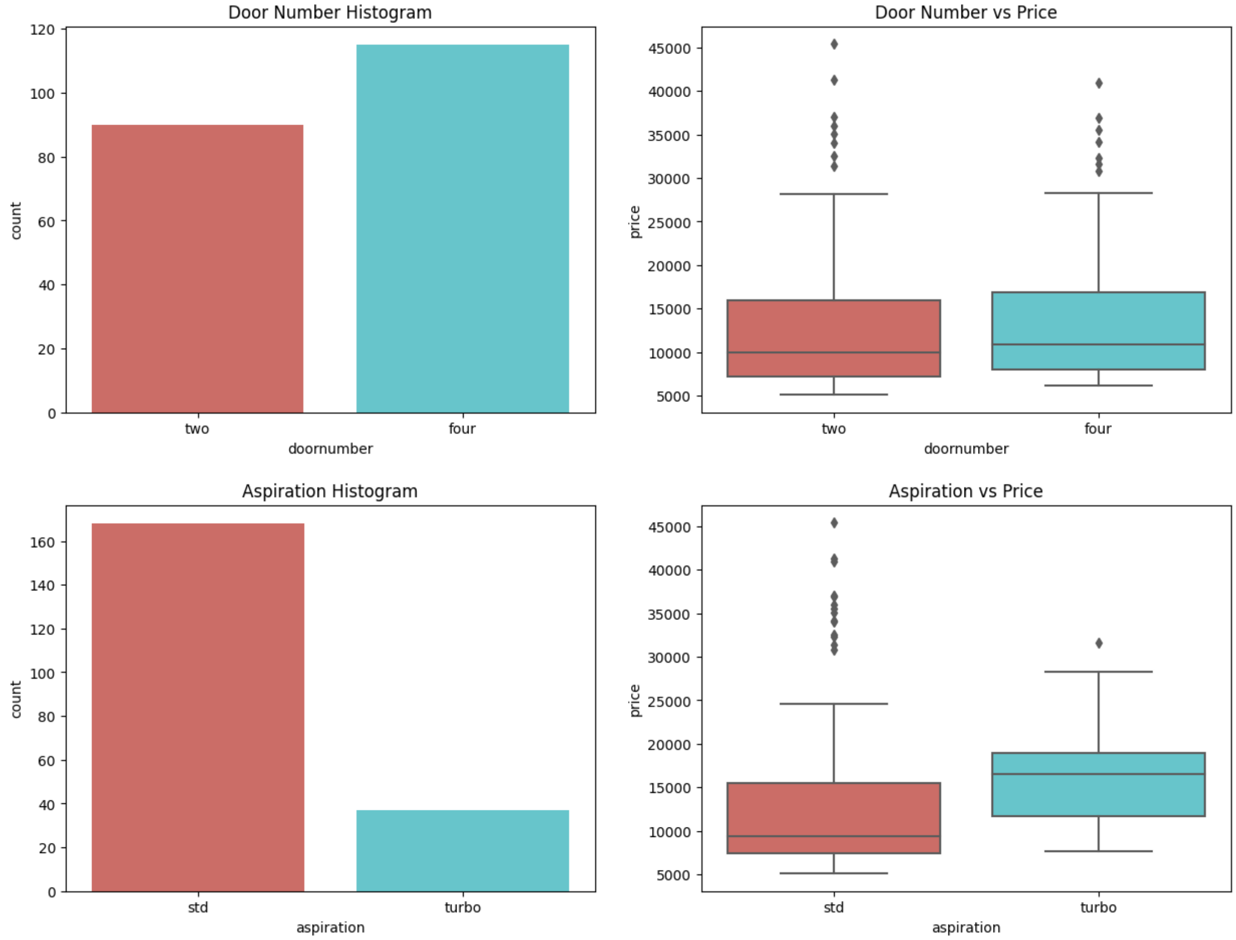



Fuel Type vs Average Price
price
16000
14000
12000
10000
8000
6000
4000
2000
0
diesel
gas
fueltype



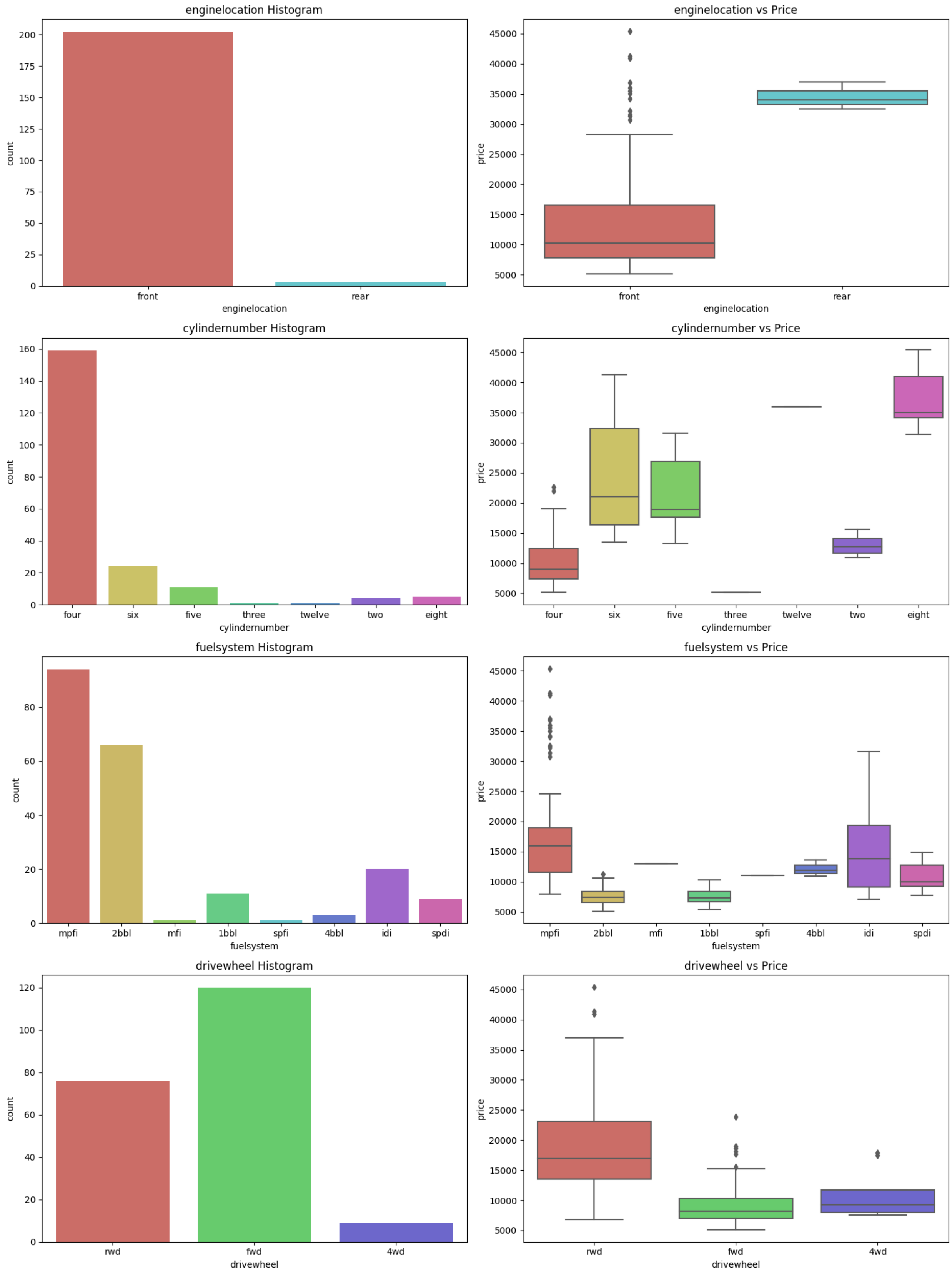
In [613…

```
#features to price
plt.figure(figsize=(15,5))

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

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

plt.show()

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

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

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

plt.show()
```



In [614…

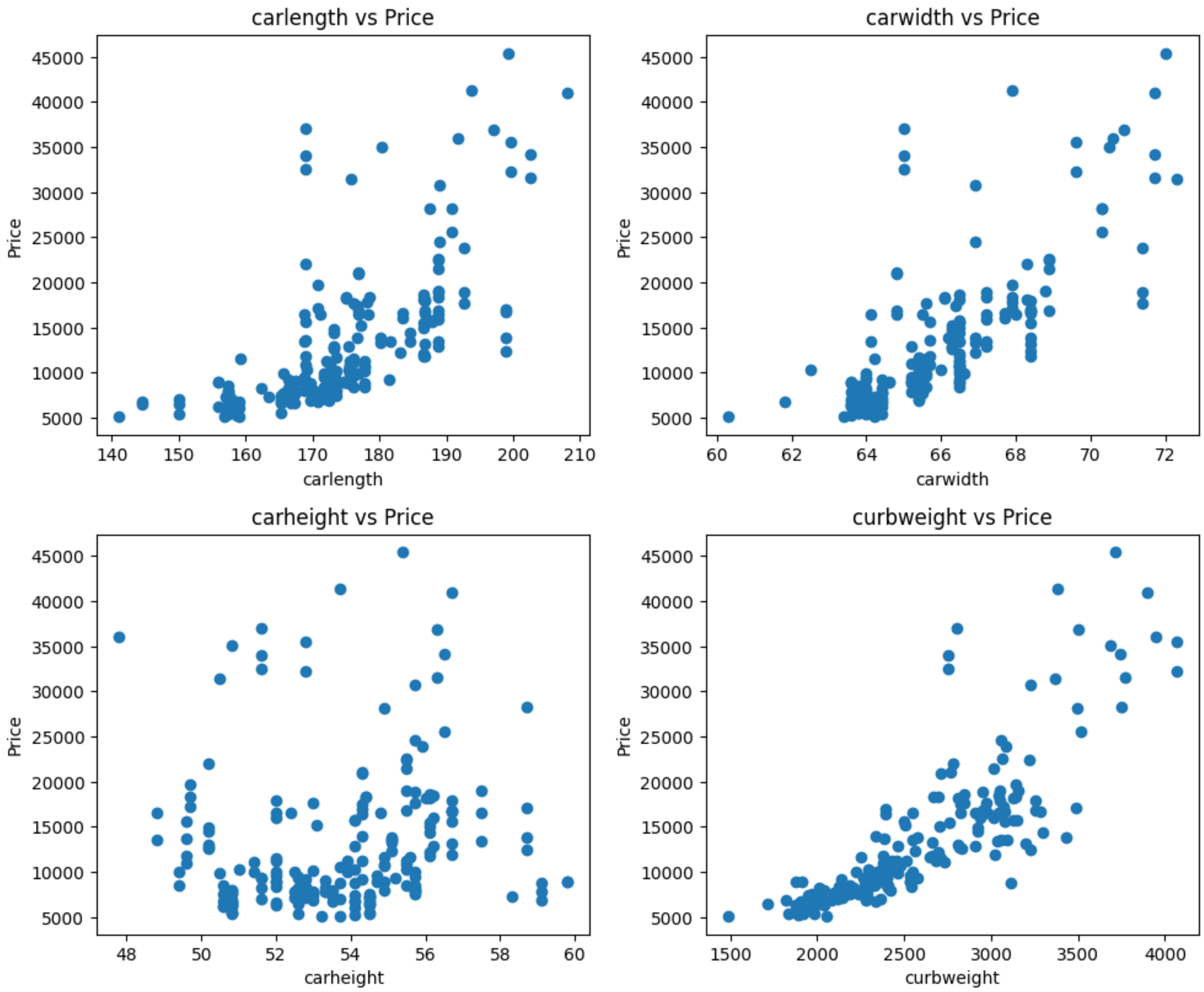
```
#features to price
def plot_count(xyz,fig):
    plt.subplot(4,2,fig)
    plt.title(xyz+' Histogram')
    sns.countplot(x=cars[xyz],palette=("hls"))
    plt.subplot(4,2,(fig+1))
    plt.title(xyz+' vs Price')
    sns.boxplot(x=cars[xyz], y=cars.price, palette=("hls"))

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

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

plt.tight_layout()
```



In [615…

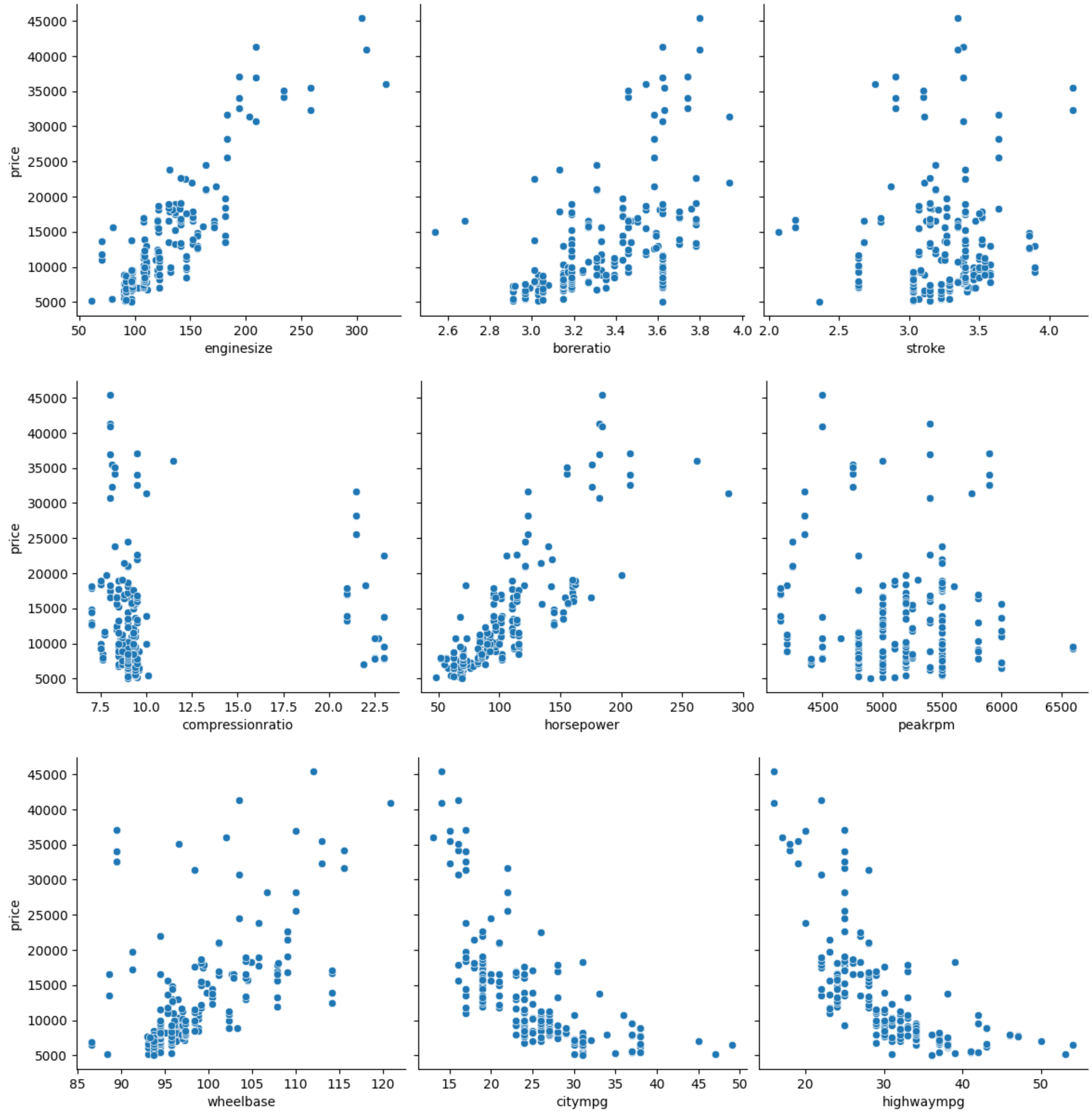
```
#features to price - correlation scatterplots
def scatter(x,fig):
    plt.subplot(5,2,fig)
    plt.scatter(cars[x],cars['price'])
    plt.title(x+' vs Price')
    plt.ylabel('Price')
    plt.xlabel(x)
```

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

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

plt.tight_layout()
```



In [616…

```
#features to price - correlation scatterplots
def pp(x,y,z):
    sns.pairplot(cars, x_vars=[x,y,z], y_vars='price',size=4, aspect=1, kind='s
    plt.show()

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



In [617…
```
np.corrcoef(cars['carlength'], cars['carwidth'])[0, 1]
```

Out[617]: 0.841118268481845

STEP 4: Feature Engineering

In [618…
```
#Fuel economy
cars['fueleconomy'] = (0.55 * cars['citympg']) + (0.45 * cars['highwaympg'])
```

In [619…
```
#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[619]:

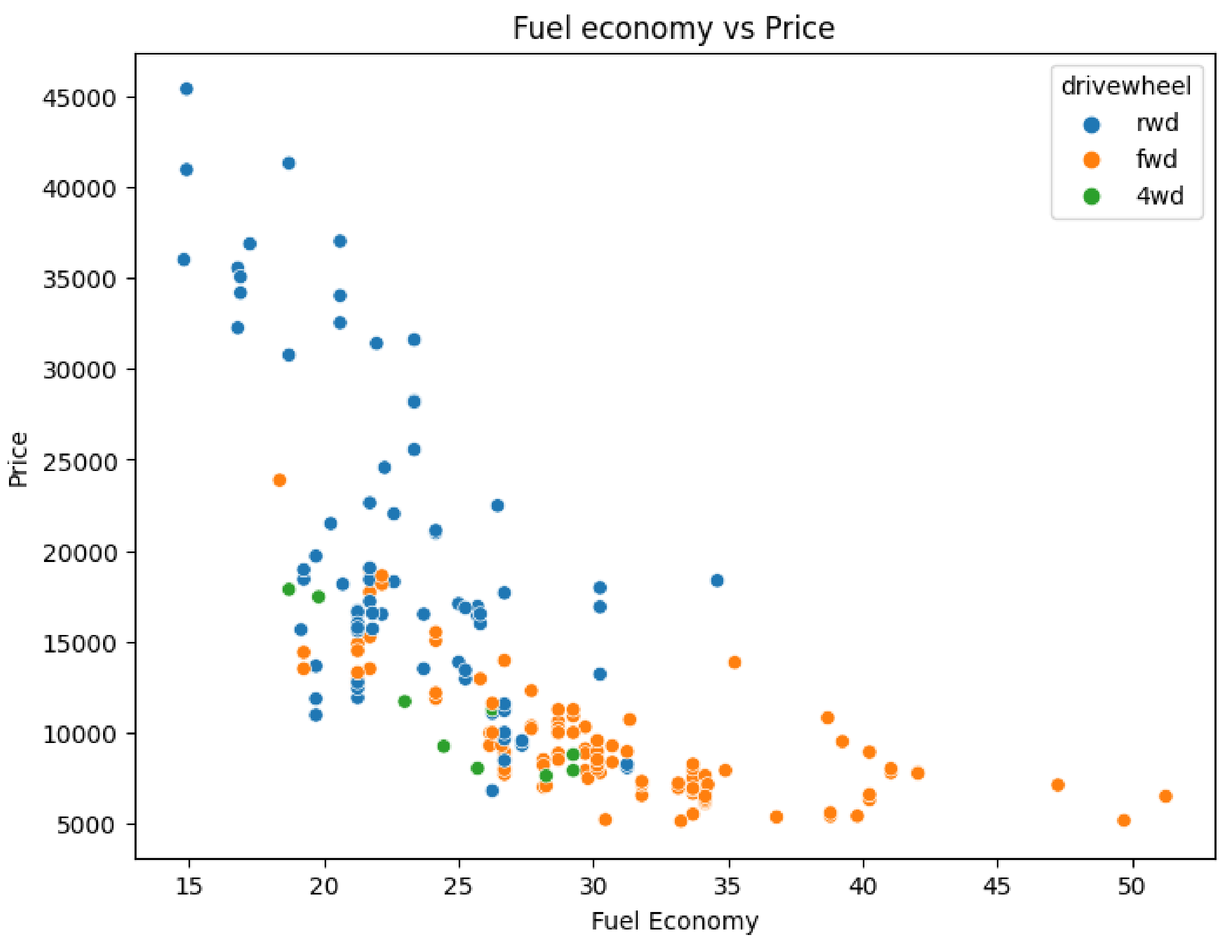
	car_ID	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewhe
0	1	3	alfa-romero	gas	std	two	convertible	rw
1	2	3	alfa-romero	gas	std	two	convertible	rw
2	3	1	alfa-romero	gas	std	two	hatchback	rw
3	4	2	audi	gas	std	four	sedan	fw
4	5	2	audi	gas	std	four	sedan	4w

5 rows × 28 columns

STEP 5 : Variable Analysis

In [620…

```
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()
```



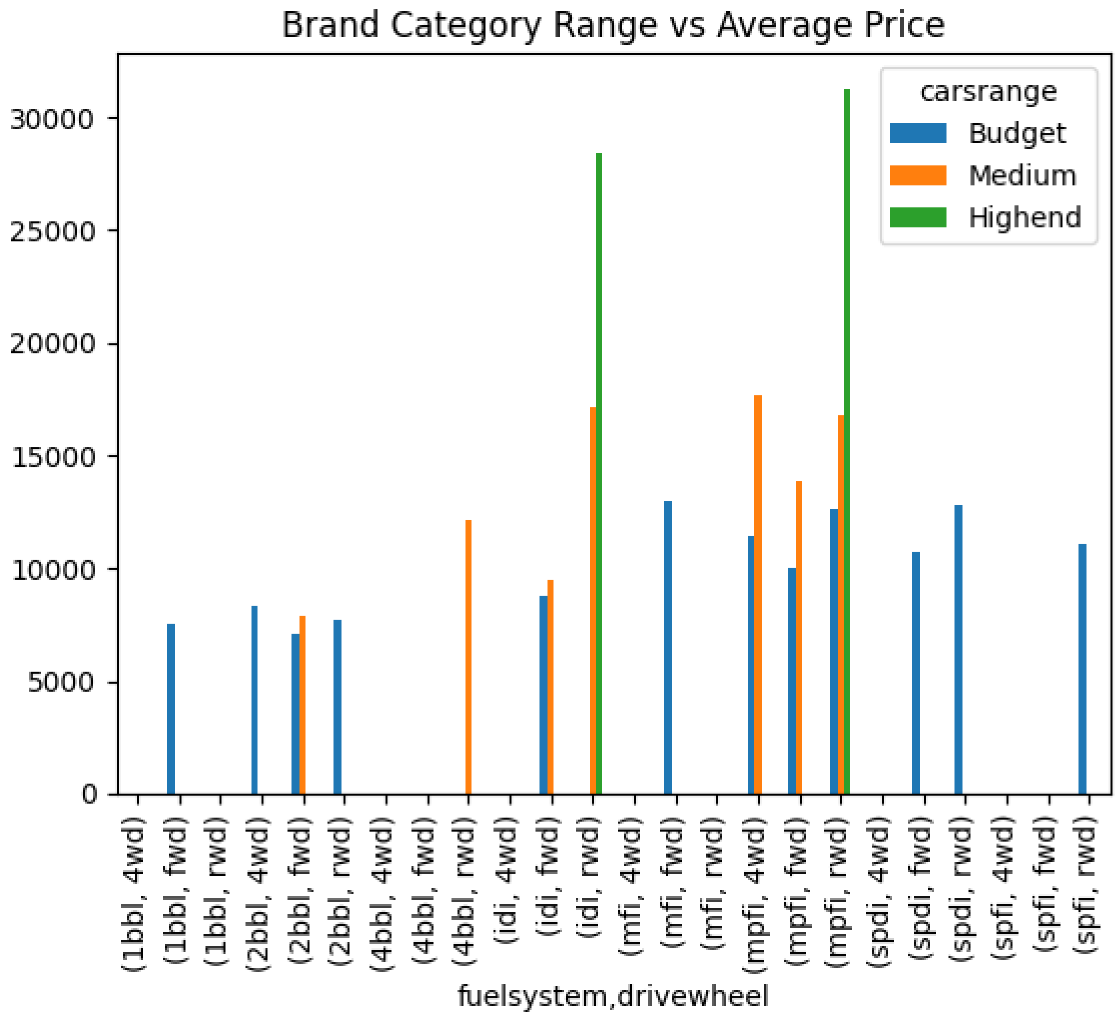
<Figure size 640x480 with 0 Axes>

In [621…

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

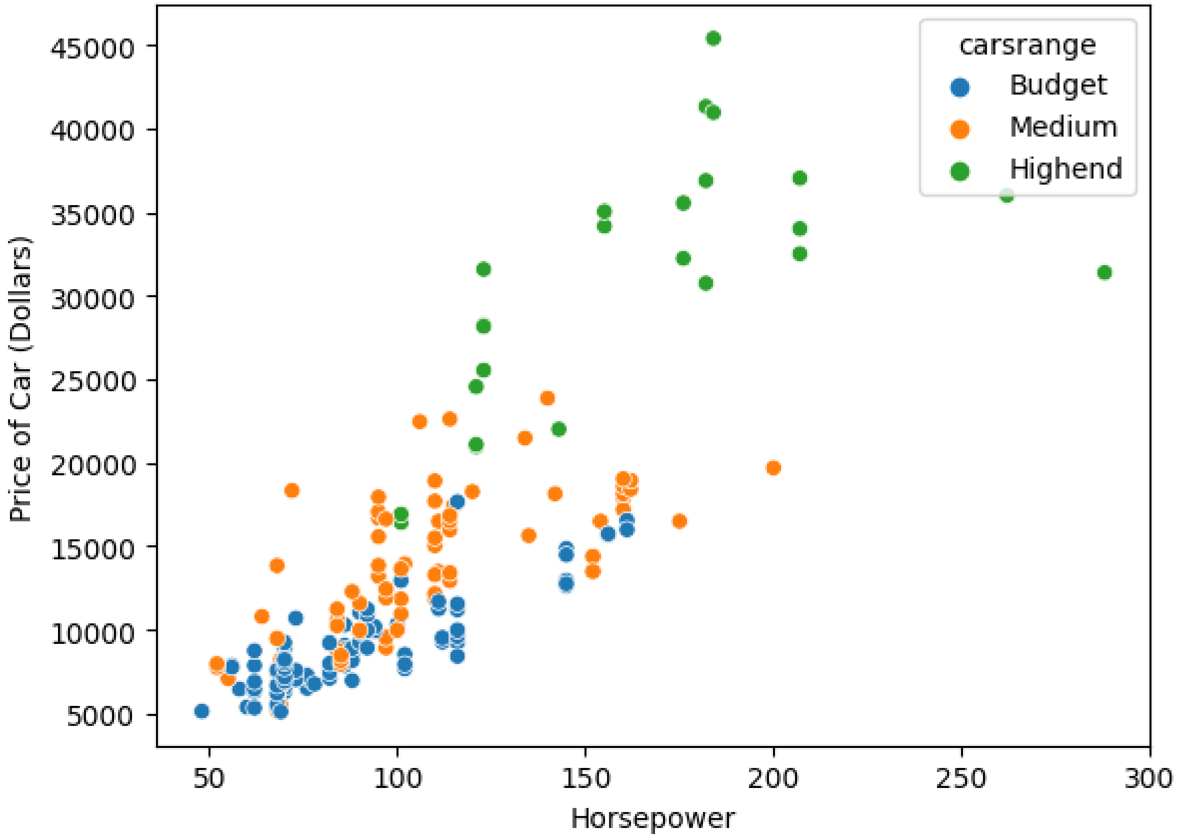
```
df = pd.DataFrame(cars.groupby(['fuelsystem','drivewheel','carsrange'])['price'
df.plot.bar()
plt.title('Brand Category Range vs Average Price')
plt.show()
```

<Figure size 2500x600 with 0 Axes>



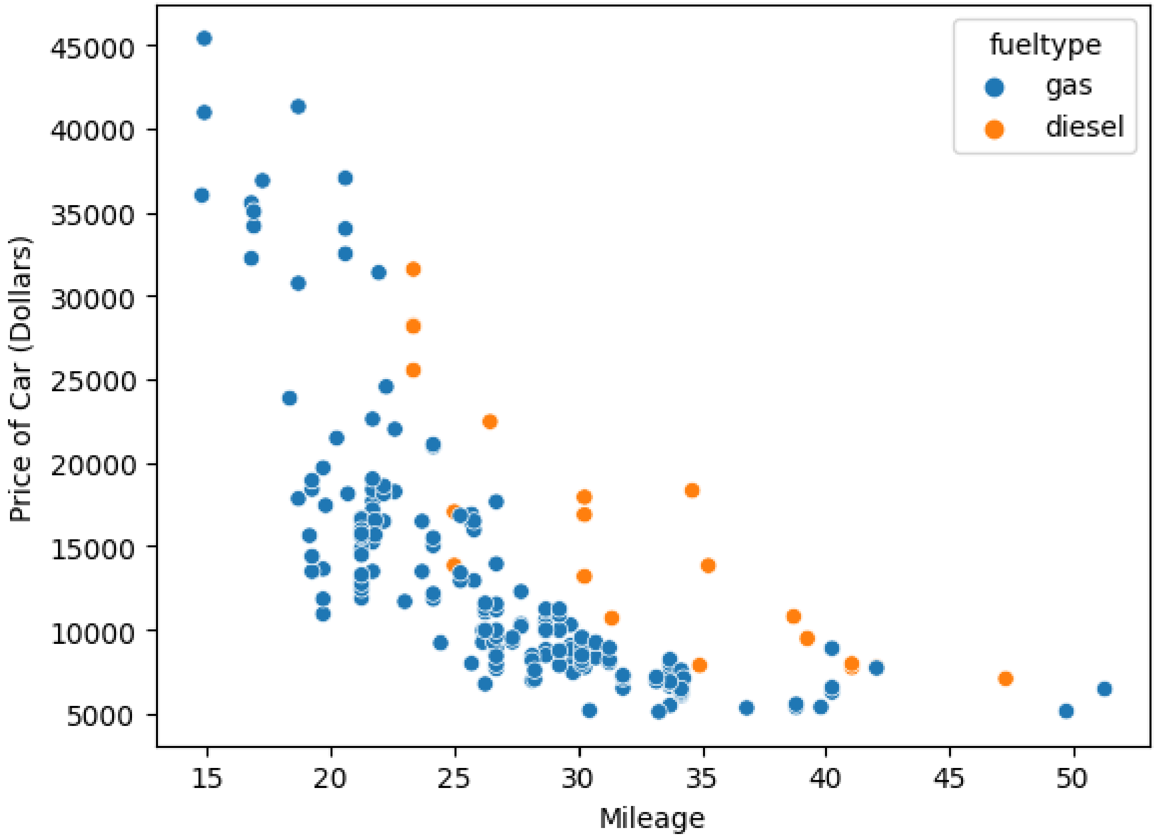
In [622…

```
# HorsePower vs Price per Brand Catgeory
plt1 = sns.scatterplot(x = 'horsepower', y = 'price', hue = 'carsrange', data =
plt1.set_xlabel('Horsepower')
plt1.set_ylabel('Price of Car (Dollars)')
plt.show()
```



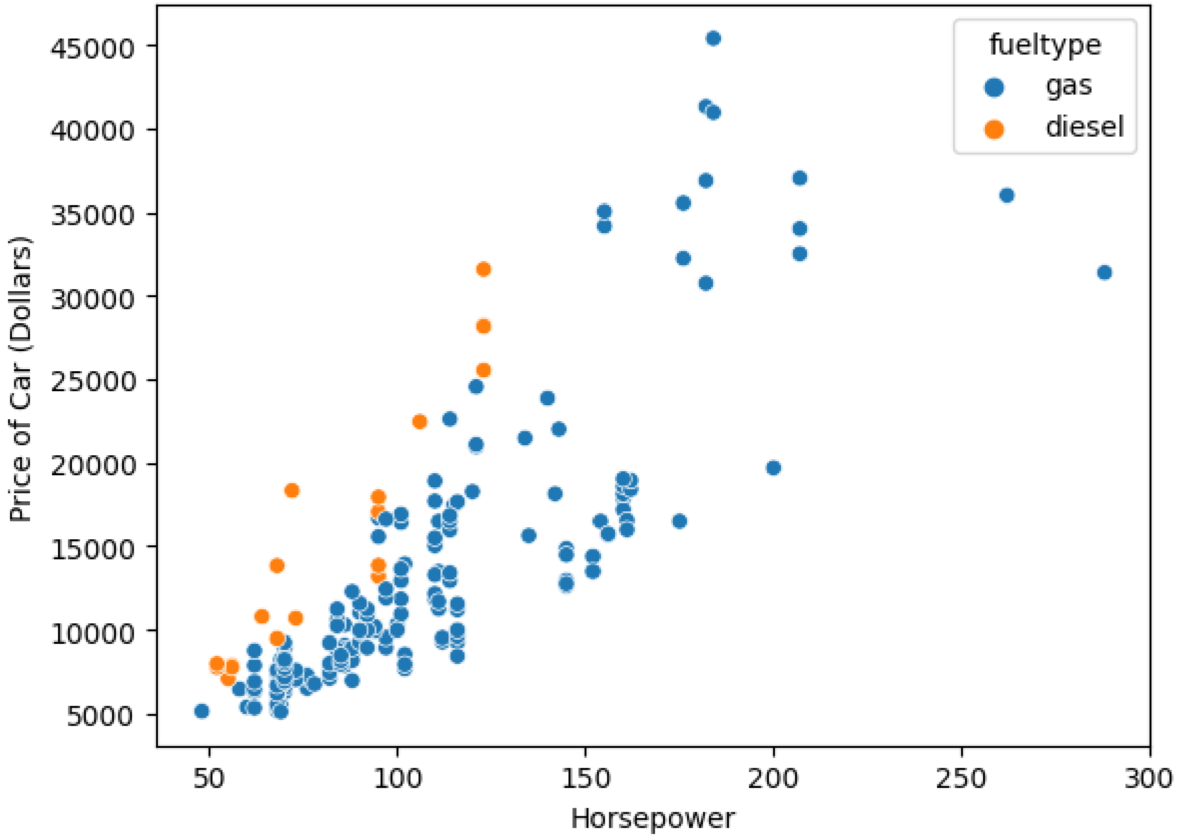
In [623…

```
# Mileage vs Fuel Type
plt1 = sns.scatterplot(x = 'fueleconomy', y = 'price', hue = 'fueltype', data =
plt1.set_xlabel('Mileage')
plt1.set_ylabel('Price of Car (Dollars)')
plt.show()
```



In [624…

```
# Horesepower vs Fuel Type
plt1 = sns.scatterplot(x = 'horsepower', y = 'price', hue = 'fueltype', data =
plt1.set_xlabel('Horsepower')
plt1.set_ylabel('Price of Car (Dollars)')
plt.show()
```



List of significant variables based on visual analysis. We can filter out variables, based on univariate and bivariate analysis, that dont impact price much. The most important drivers for price are: - 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

Logistic Regression

In [625…

```
cars_lr = cars[['price', 'fueltype', 'aspiration','carbody', 'drivewheel','whee
                  'curbweight', 'enginetype', 'cylindernumber', 'enginesize', '
                   'fueleconomy', 'carlength','carwidth', 'carsrange']]
cars_lr.head()
```

Out[625]:

	price	fueltype	aspiration	carbody	drivewheel	wheelbase	curbweight	enginetype	cy
0	13495	gas	std	convertible	rwd	88.6	2548	dohc	
1	16500	gas	std	convertible	rwd	88.6	2548	dohc	
2	16500	gas	std	hatchback	rwd	94.5	2823	ohcv	
3	13950	gas	std	sedan	fwd	99.8	2337	ohc	
4	17450	gas	std	sedan	4wd	99.4	2824	ohc	

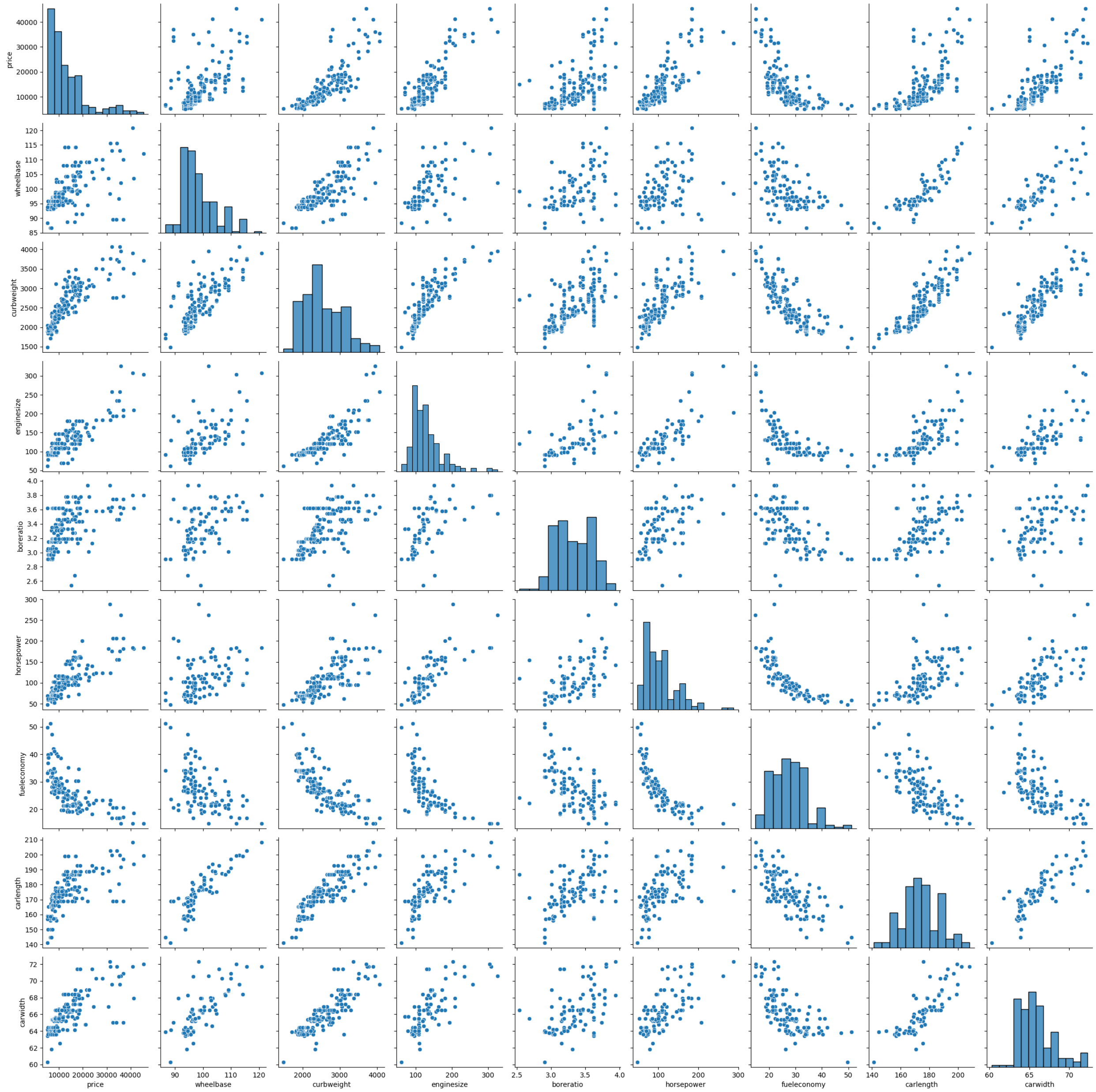
In [626…

```
cars_lr.shape
```

Out[626]: (205, 16)

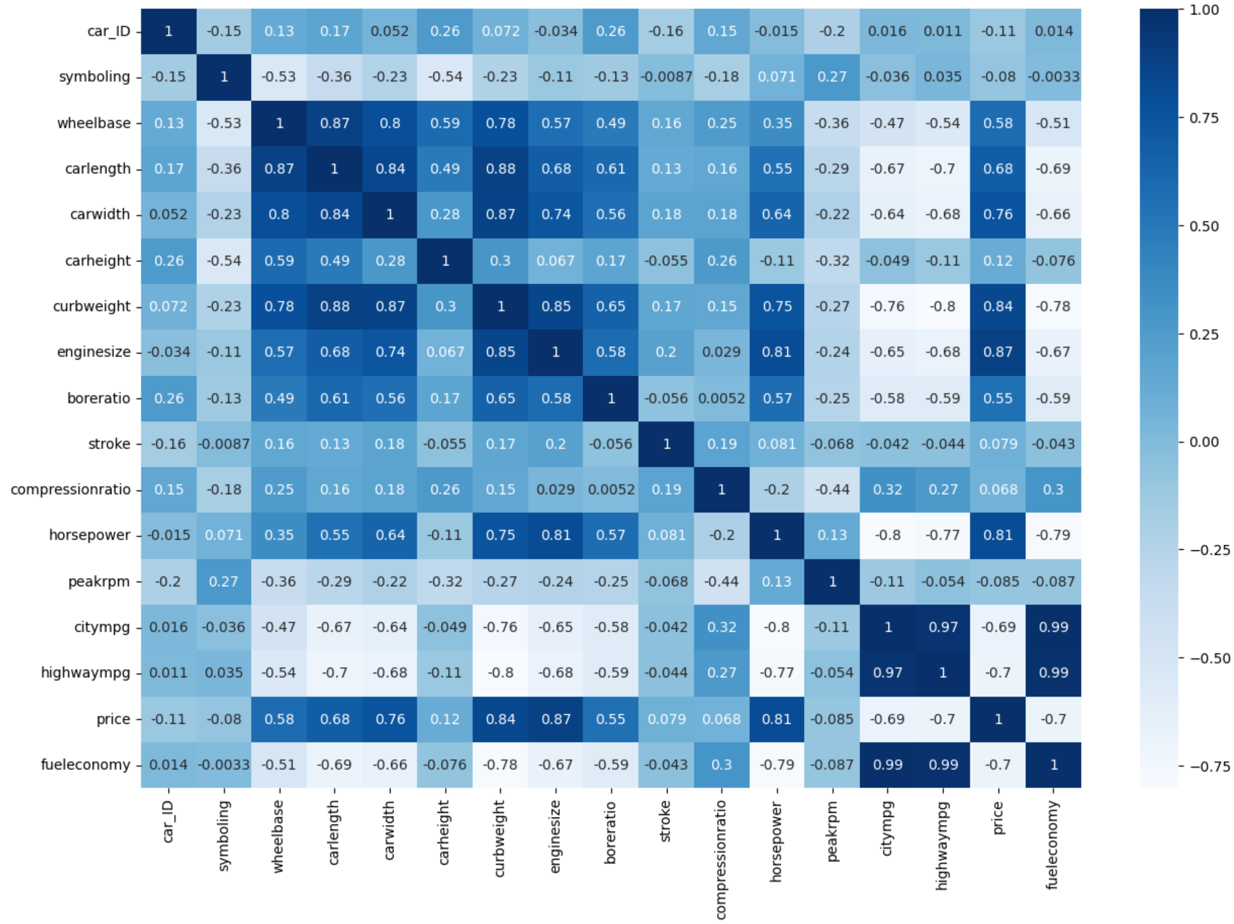
In [627…

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



In [628…
```
plt.figure(figsize=(15,10))
sns.heatmap(cars.corr() , cmap="Blues", annot=True)
```

Out[628]: <AxesSubplot: >



In [629…

```
car_data = pd.get_dummies(cars, drop_first =True)
car_data.head()
```

Out[629]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	bor
0	1	3	88.6	168.8	64.1	48.8	2548	130	
1	2	3	88.6	168.8	64.1	48.8	2548	130	
2	3	1	94.5	171.2	65.5	52.4	2823	152	
3	4	2	99.8	176.6	66.2	54.3	2337	109	
4	5	2	99.4	176.6	66.4	54.3	2824	136	

5 rows × 69 columns

In [630…

```
car_data.corr()
```

Out[630]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curb
car_ID	1.000000	-0.151621	0.129729	0.170636	0.052387	0.255960	0.(
symboling	-0.151621	1.000000	-0.531954	-0.357612	-0.232919	-0.541038	-0.:
wheelbase	0.129729	-0.531954	1.000000	0.874587	0.795144	0.589435	0.7
carlength	0.170636	-0.357612	0.874587	1.000000	0.841118	0.491029	0.8
carwidth	0.052387	-0.232919	0.795144	0.841118	1.000000	0.279210	0.8
...	...	...	...	...	...	...	
fuelsystem_mpfi	0.186275	0.012532	0.348891	0.511374	0.461896	0.108685	0.5
fuelsystem_spdi	-0.037015	0.181939	-0.117359	-0.079790	-0.046399	-0.278615	-0.0
fuelsystem_spfi	-0.066254	0.065707	-0.032129	-0.008245	-0.023158	-0.066778	0.0
carsrange_Medium	0.094711	0.025968	0.195713	0.286389	0.219547	0.342107	0
carsrange_Highend	-0.255618	-0.049092	0.333254	0.373687	0.453143	0.024294	0.!

69 rows × 69 columns

STEP 6: Feature Scaling

In [ ]:

In [631…

```
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
```

In [632…

```
from sklearn.preprocessing import MinMaxScaler

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

In [633…

```
df_train.head()
```

Out[633]:

	price	fueltype	aspiration	carbody	drivewheel	wheelbase	curbweight	enginety
122	0.068818	gas	std	sedan	fwd	0.244828	0.272692	o
125	0.466890	gas	std	hatchback	rwd	0.272414	0.500388	o
166	0.122110	gas	std	hatchback	rwd	0.272414	0.314973	do
1	0.314446	gas	std	convertible	rwd	0.068966	0.411171	do
199	0.382131	gas	turbo	wagon	rwd	0.610345	0.647401	o

In [634…

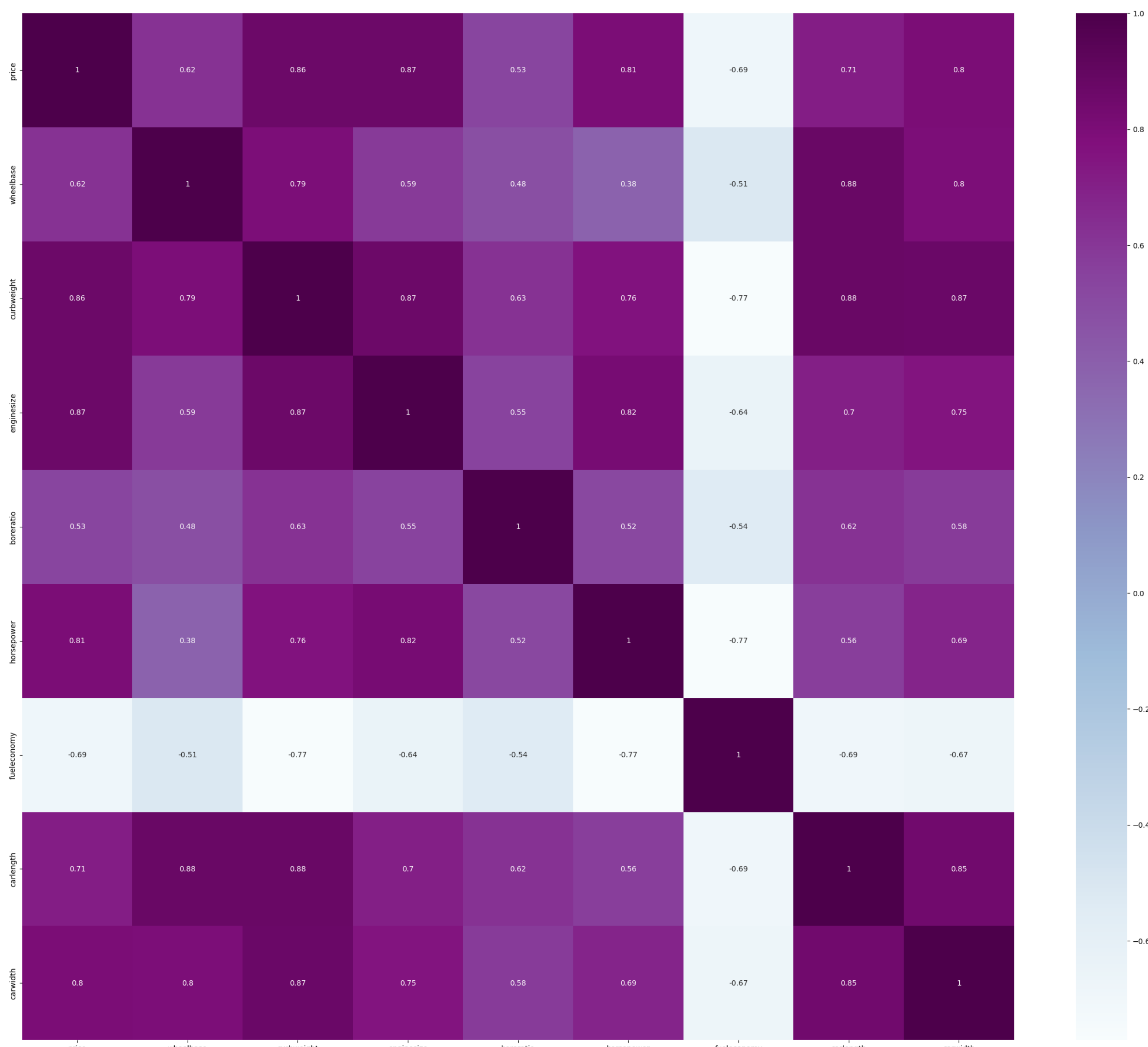
```
df_train.describe()
```

Out[634]:

	price	wheelbase	curbweight	enginesize	boreratio	horsepower	fueleconom
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.00000
mean	0.219309	0.411141	0.407878	0.241351	0.497946	0.227302	0.35826
std	0.215682	0.205581	0.211269	0.154619	0.207140	0.165511	0.18598
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.067298	0.272414	0.245539	0.135849	0.305556	0.091667	0.19890
50%	0.140343	0.341379	0.355702	0.184906	0.500000	0.191667	0.34430
75%	0.313479	0.503448	0.559542	0.301887	0.682540	0.283333	0.51234
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

In [635…

```
# Correlation using heatmap
plt.figure(figsize = (30, 25))
sns.heatmap(df_train.corr(), annot = True, cmap="BuPu")
plt.show()
```



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

In [636…

```
#Dividing data into X and y variables
```

```
y_train = df_train.pop('price')
X_train = df_train
```

STEP 7 : Model Development

Data Preparation

In [637…
```
# Categorical Variables are converted into Numerical Variables with the help of
cyl_no = pd.get_dummies(cars['cylindernumber'], drop_first = True)
```

In [638…
```
cars = pd.concat([cars, cyl_no], axis = 1)
```

In [639…
```
brand_cat = pd.get_dummies(cars['carsrange'], drop_first = True)
```

In [640…
```
cars = pd.concat([cars, brand_cat], axis = 1)
```

In [641…
```
eng_typ = pd.get_dummies(cars['enginetype'], drop_first = True)
```

In [642…
```
cars = pd.concat([cars, eng_typ], axis = 1)
```

In [643…
```
drwh = pd.get_dummies(cars['drivewheel'], drop_first = True)
```

In [644…
```
cars = pd.concat([cars, drwh], axis = 1)
```

In [645…
```
carb = pd.get_dummies(cars['carbody'], drop_first = True)
```

In [646…
```
cars = pd.concat([cars, carb], axis = 1)
```

In [647…
```
asp = pd.get_dummies(cars['aspiration'], drop_first = True)
```

In [648…
```
cars = pd.concat([cars, asp], axis = 1)
```

In [649…
```
fuelt = pd.get_dummies(cars['fueltype'], drop_first = True)
```

In [650…
```
cars = pd.concat([cars, fuelt], axis = 1)
```

In [651…
```
cars.drop(['fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginetype', 'cy
```

splitting data into training and test sets

In [652…
```
from sklearn.model_selection import train_test_split

# We specify this so that the train and test data set always have the same rows
np.random.seed(0)
df_train, df_test = train_test_split(cars, train_size = 0.7, test_size = 0.3, r
```

re-scaling the features

In [653…
```
# min-max scaling
from sklearn.preprocessing import MinMaxScaler
```

In [654…
```
scaler = MinMaxScaler()
```

In [655…
```
# Apply scaler() to all the columns except the 'dummy' variables
```

```
num_vars = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize','b

df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

In [656…
```
df_train.drop(columns=['CompanyName','doornumber','enginelocation','fuelsystem'
```

In [657…
```
df_train.head()
```

Out[657]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize
122	123	1	0.244828	0.426016	0.291667	50.8	0.272692	0.139623
125	126	3	0.272414	0.452033	0.666667	50.2	0.500388	0.339623
166	167	1	0.272414	0.448780	0.308333	52.6	0.314973	0.139623
1	2	3	0.068966	0.450407	0.316667	48.8	0.411171	0.260377
199	200	-1	0.610345	0.775610	0.575000	57.5	0.647401	0.260377

5 rows × 39 columns

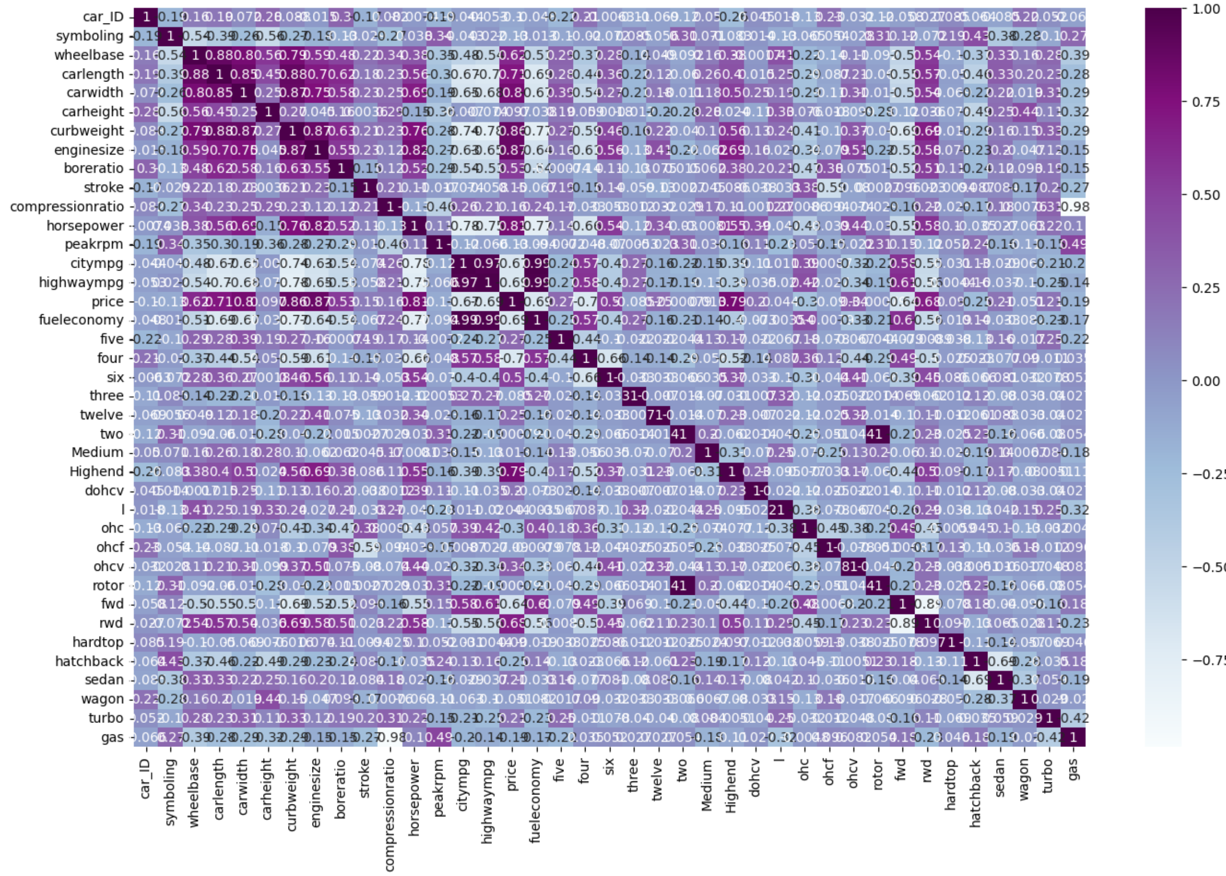
In [658…
```
df_train.describe()
```

Out[658]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000
mean	98.524476	0.797203	0.411141	0.525476	0.461655	53.551748	0.407878
std	58.977655	1.195999	0.205581	0.204848	0.184517	2.433766	0.211269
min	1.000000	-2.000000	0.000000	0.000000	0.000000	47.800000	0.000000
25%	48.500000	0.000000	0.272414	0.399187	0.304167	51.800000	0.245539
50%	97.000000	1.000000	0.341379	0.502439	0.425000	53.700000	0.355702
75%	147.500000	1.000000	0.503448	0.669919	0.550000	55.350000	0.559542
max	205.000000	3.000000	1.000000	1.000000	1.000000	59.100000	1.000000

8 rows × 39 columns

In [659…
```
# correlation coefficients to see which variables are highly correlated

plt.figure(figsize = (16, 10))
sns.heatmap(df_train.corr(), annot = True, cmap="BuPu")
plt.show()
```



In [660…
```
# Dividing into X and Y sets for the model building
y_train = df_train.pop('price')
X_train = df_train
```

In [661…
```
# RFE Recursive Feature Elimination
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

In [662…
```
# Running RFE with the output number of the variable equal to 10
lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, step = 10)             # running RFE
rfe = rfe.fit(X_train, y_train)
```

In [663…
```
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

Out[663]:
```
[('car_ID', False, 3),
 ('symboling', False, 3),
 ('wheelbase', True, 1),
 ('carlength', True, 1),
 ('carwidth', True, 1),
 ('carheight', False, 3),
 ('curbweight', True, 1),
 ('enginesize', True, 1),
 ('boreratio', True, 1),
 ('stroke', True, 1),
 ('compressionratio', False, 3),
 ('horsepower', True, 1),
 ('peakrpm', False, 3),
 ('citympg', False, 3),
 ('highwaympg', False, 3),
 ('fueleconomy', False, 3),
 ('five', False, 2),
 ('four', True, 1),
 ('six', False, 3),
 ('three', True, 1),
 ('twelve', True, 1),
 ('two', True, 1),
 ('Medium', False, 2),
 ('Highend', True, 1),
 ('dohcv', True, 1),
 ('l', False, 2),
 ('ohc', True, 1),
 ('ohcf', True, 1),
 ('ohcv', True, 1),
 ('rotor', True, 1),
 ('fwd', False, 2),
 ('rwd', False, 3),
 ('hardtop', False, 2),
 ('hatchback', True, 1),
 ('sedan', False, 2),
 ('wagon', False, 2),
 ('turbo', False, 2),
 ('gas', False, 2)]
```

In [664…
```
col = X_train.columns[rfe.support_]
col
```

Out[664]:
```
Index(['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize',
       'boreratio', 'stroke', 'horsepower', 'four', 'three', 'twelve', 'two',
       'Highend', 'dohcv', 'ohc', 'ohcf', 'ohcv', 'rotor', 'hatchback'],
      dtype='object')
```

In [665…
```
# Building model using statsmodel, for the detailed statistics
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train[col]
```

In [666…
```
# Adding a constant variable
import statsmodels.api as sm
X_train_rfe = sm.add_constant(X_train_rfe)
```

In [667…
```
lm = sm.OLS(y_train,X_train_rfe).fit()   # Running the linear model
```

In [668…
```
#summary of linear model
print(lm.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.940
Model:                            OLS   Adj. R-squared:                  0.932
Method:                 Least Squares   F-statistic:                     108.3
Date:                Fri, 30 Sep 2022   Prob (F-statistic):           8.15e-67
Time:                        15:03:38   Log-Likelihood:                 218.37
No. Observations:                 143   AIC:                            -398.7
Df Residuals:                     124   BIC:                            -342.4
Df Model:                          18                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2302      0.082      2.818      0.006       0.069       0.392
wheelbase      0.0071      0.062      0.114      0.910      -0.116       0.130
carlength     -0.1372      0.081     -1.696      0.092      -0.297       0.023
carwidth       0.3523      0.070      5.012      0.000       0.213       0.491
curbweight     0.2317      0.086      2.686      0.008       0.061       0.402
enginesize     0.7560      0.173      4.370      0.000       0.414       1.098
boreratio     -0.1790      0.062     -2.879      0.005      -0.302      -0.056
stroke        -0.1429      0.030     -4.821      0.000      -0.202      -0.084
horsepower     0.3833      0.076      5.043      0.000       0.233       0.534
four           0.0484      0.030      1.591      0.114      -0.012       0.109
three          0.2765      0.080      3.435      0.001       0.117       0.436
twelve        -0.4538      0.099     -4.605      0.000      -0.649      -0.259
two            0.1369      0.034      4.072      0.000       0.070       0.203
Highend        0.1680      0.029      5.893      0.000       0.112       0.224
dohcv         -0.2434      0.077     -3.146      0.002      -0.396      -0.090
ohc            0.0643      0.021      2.993      0.003       0.022       0.107
ohcf          -0.0012      0.031     -0.037      0.970      -0.063       0.060
ohcv          -0.0869      0.031     -2.812      0.006      -0.148      -0.026
rotor          0.1369      0.034      4.072      0.000       0.070       0.203
hatchback     -0.0404      0.013     -3.104      0.002      -0.066      -0.015
==============================================================================
Omnibus:                       39.991   Durbin-Watson:                   2.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               92.151
Skew:                           1.159   Prob(JB):                     9.76e-21
Kurtosis:                       6.176   Cond. No.                     2.73e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 2.69e-32. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [669…
```
# Calculate the VIFs variance inflation factor for the new model
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_rfe
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)
vif
```

Out[669]:

	Features	VIF
18	rotor	inf
12	two	inf
0	const	299.78
5	enginesize	31.91
4	curbweight	14.81
2	carlength	12.24
9	four	7.98
3	carwidth	7.50
6	boreratio	7.40
1	wheelbase	7.29
8	horsepower	7.06
15	ohc	4.47
7	stroke	3.83
13	Highend	3.82
16	ohcf	3.32
11	twelve	3.03
17	ohcv	2.53
10	three	2.02
14	dohcv	1.87
19	hatchback	1.72

In [670…

```
# Dropping curbweight as p-value is high.
X_train_new1 = X_train_rfe.drop(["curbweight"], axis = 1)


# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new1)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model

#again summary of linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.937
Model:                            OLS   Adj. R-squared:                  0.928
Method:                 Least Squares   F-statistic:                     108.9
Date:                Fri, 30 Sep 2022   Prob (F-statistic):           2.48e-66
Time:                        15:03:38   Log-Likelihood:                 214.33
No. Observations:                 143   AIC:                            -392.7
Df Residuals:                     125   BIC:                            -339.3
Df Model:                          17
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2348      0.084      2.806      0.006       0.069       0.400
wheelbase      0.0451      0.062      0.727      0.469      -0.078       0.168
carlength     -0.0803      0.080     -1.004      0.317      -0.239       0.078
carwidth       0.3891      0.071      5.510      0.000       0.249       0.529
enginesize     0.8710      0.172      5.071      0.000       0.531       1.211
boreratio     -0.1731      0.064     -2.718      0.007      -0.299      -0.047
stroke        -0.1449      0.030     -4.770      0.000      -0.205      -0.085
horsepower     0.4309      0.076      5.690      0.000       0.281       0.581
four           0.0506      0.031      1.623      0.107      -0.011       0.112
three          0.2748      0.082      3.333      0.001       0.112       0.438
twelve        -0.4908      0.100     -4.909      0.000      -0.689      -0.293
two            0.1424      0.034      4.144      0.000       0.074       0.210
Highend        0.1708      0.029      5.850      0.000       0.113       0.229
dohcv         -0.2748      0.078     -3.508      0.001      -0.430      -0.120
ohc            0.0526      0.022      2.442      0.016       0.010       0.095
ohcf          -0.0151      0.031     -0.483      0.630      -0.077       0.047
ohcv          -0.0907      0.032     -2.869      0.005      -0.153      -0.028
rotor          0.1424      0.034      4.144      0.000       0.074       0.210
hatchback     -0.0409      0.013     -3.068      0.003      -0.067      -0.015
==============================================================================
Omnibus:                       31.386   Durbin-Watson:                   2.081
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               60.463
Skew:                           0.980   Prob(JB):                     7.43e-14
Kurtosis:                       5.511   Cond. No.                     9.42e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 2.23e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

So mileage is insignificant now

In [671…

```python
# Dropping ohcf as p value is high.
X_train_new2 = X_train_new1.drop(["ohcf"], axis = 1)


# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new2)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model
```

```
#again summary of linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.937
Model:                            OLS   Adj. R-squared:                  0.929
Method:                 Least Squares   F-statistic:                     116.4
Date:                Fri, 30 Sep 2022   Prob (F-statistic):           2.54e-67
Time:                        15:03:38   Log-Likelihood:                 214.19
No. Observations:                 143   AIC:                            -394.4
Df Residuals:                     126   BIC:                            -344.0
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2181      0.076      2.872      0.005       0.068       0.368
wheelbase      0.0494      0.061      0.806      0.421      -0.072       0.171
carlength     -0.0717      0.078     -0.923      0.358      -0.226       0.082
carwidth       0.3855      0.070      5.506      0.000       0.247       0.524
enginesize     0.8929      0.165      5.409      0.000       0.566       1.220
boreratio     -0.1888      0.054     -3.466      0.001      -0.297      -0.081
stroke        -0.1438      0.030     -4.762      0.000      -0.204      -0.084
horsepower     0.4368      0.075      5.862      0.000       0.289       0.584
four           0.0565      0.029      1.982      0.050    8.07e-05       0.113
three          0.2902      0.076      3.831      0.000       0.140       0.440
twelve        -0.5010      0.097     -5.145      0.000      -0.694      -0.308
two            0.1498      0.031      4.878      0.000       0.089       0.211
Highend        0.1690      0.029      5.853      0.000       0.112       0.226
dohcv         -0.2652      0.076     -3.512      0.001      -0.415      -0.116
ohc            0.0577      0.019      3.073      0.003       0.021       0.095
ohcv          -0.0876      0.031     -2.839      0.005      -0.149      -0.027
rotor          0.1498      0.031      4.878      0.000       0.089       0.211
hatchback     -0.0403      0.013     -3.043      0.003      -0.066      -0.014
==============================================================================
Omnibus:                       30.812   Durbin-Watson:                   2.066
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               57.102
Skew:                           0.981   Prob(JB):                     3.98e-13
Kurtosis:                       5.395   Cond. No.                     9.11e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 2.39e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [672…

```
# Calculate the VIFs for the new model
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_new2
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)
vif
```

Out[672]:

	Features	VIF
11	two	inf
16	rotor	inf
0	const	248.11
4	enginesize	27.85
2	carlength	10.84
3	carwidth	7.13
1	wheelbase	6.78
8	four	6.72
7	horsepower	6.50
5	boreratio	5.44
6	stroke	3.81
12	Highend	3.76
14	ohc	3.27
10	twelve	2.83
15	ohcv	2.42
9	three	1.72
13	dohcv	1.70
17	hatchback	1.70

In [673…

```
# making predictions
```

In [674…

```
num_vars = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize','b

df_test[num_vars] = scaler.transform(df_test[num_vars])
```

In [675…

```
y_test = df_test.pop('price')
X_test = df_test
```

In [676…

```
# Now let's use our model to make predictions.

# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test[['carwidth', 'horsepower', 'wheelbase', 'enginesize']]

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

In [678…

```
# Making predictions
y_pred = lm.predict(X_test_new)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [678], line 2
      1 # Making predictions
----> 2 y_pred = lm.predict(X_test_new)

File /opt/homebrew/lib/python3.10/site-packages/statsmodels/base/model.py:1159
, in Results.predict(self, exog, transform, *args, **kwargs)
   1156         exog = exog[:, None]
   1157     exog = np.atleast_2d(exog)  # needed in count model shape[1]
-> 1159 predict_results = self.model.predict(self.params, exog, *args,
   1160                                      **kwargs)
   1162 if exog_index is not None and not hasattr(predict_results,
   1163                                           'predicted_values'):
   1164     if predict_results.ndim == 1:

File /opt/homebrew/lib/python3.10/site-packages/statsmodels/regression/linear_
model.py:381, in RegressionModel.predict(self, params, exog)
    378 if exog is None:
    379     exog = self.exog
--> 381 return np.dot(exog, params)

File <__array_function__ internals>:180, in dot(*args, **kwargs)

ValueError: shapes (62,5) and (18,) not aligned: 5 (dim 1) != 18 (dim 0)
```

In [ ]:

In [ ]: