In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

```
/kaggle/input/car-price-prediction/CarPrice_Assignment.csv
/kaggle/input/car-price-prediction/Data Dictionary - carprices.xlsx
```

Load Data

In [2]:

```
df= pd.read_csv('/kaggle/input/car-price-prediction/CarPrice_Assignment.csv')
df.head()
```

Out[2]:

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

5 rows × 26 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
```

```
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [4]:

```
df.isna().sum()
```

Out[4]:

```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

In [5]:

```
df.describe()
```

Out[5]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	str
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	str
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170

In [6]:

```
df.nunique()
```

Out[6]:

```
car_ID              205
symboling             6
CarName             147
fueltype              2
aspiration            2
doornumber            2
carbody               5
drivewheel            3
enginelocation        2
wheelbase            53
carlength            75
carwidth             44
carheight            49
curbweight          171
enginetype            7
cylindernumber        7
enginesize           44
fuelsystem            8
boreratio            38
stroke               37
compressionratio     32
horsepower           59
peakrpm              23
citympg              29
highwaympg           30
price               189
dtype: int64
```

In [7]:

```
df['Company']= df['CarName'].apply(lambda x: x.split(" ")[0])
df['Company']= df['Company'].apply(lambda x: x.lower())
```

In [8]:

```
df.head()
```

Out[8]:

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

5 rows × 27 columns

In [9]:

```
df['Company'].unique()
```

Out[9]:

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

In [10]:

```
df['Company'].replace({'maxda': 'mazda', 'porcshce':'porsche', 'toyouta':'toyota', 'voks
wagen':'volkswagen','vw':'volkswagen'}, inplace=True)
```

In [11]:

```
df['Company'].unique()
```

Out[11]:

```
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 [12]:

```
df.drop(['car_ID','CarName'], axis=1, inplace=True)
```

In [13]:

```
cat_cols= [col for col in df.columns if df[col].dtype=='object']
num_cols= [col for col in df.columns if df[col].dtype!='object']
```

In [14]:

```
cat_cols
```

Out[14]:

```
['fueltype',
 'aspiration',
 'doornumber',
 'carbody',
 'drivewheel',
 'enginelocation',
 'enginetype',
 'cylindernumber',
 'fuelsystem',
 'Company']
```

In [15]:

```
num_cols
num_cols.remove('price')
num_cols
```

Out[15]:

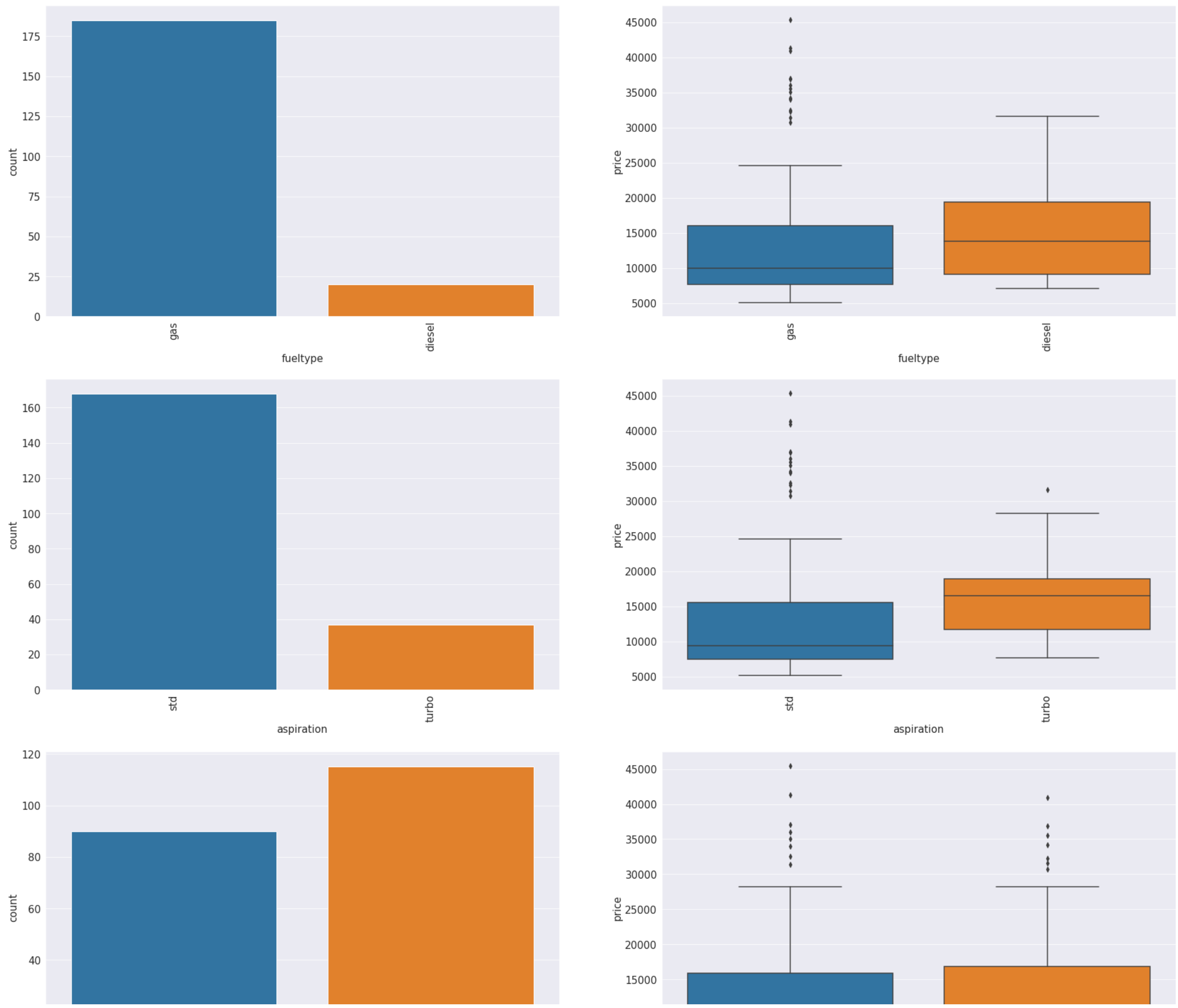
```
['symboling',
 'wheelbase',
 'carlength',
 'carwidth',
 'carheight',
 'curbweight',
 'enginesize',
 'boreratio',
```

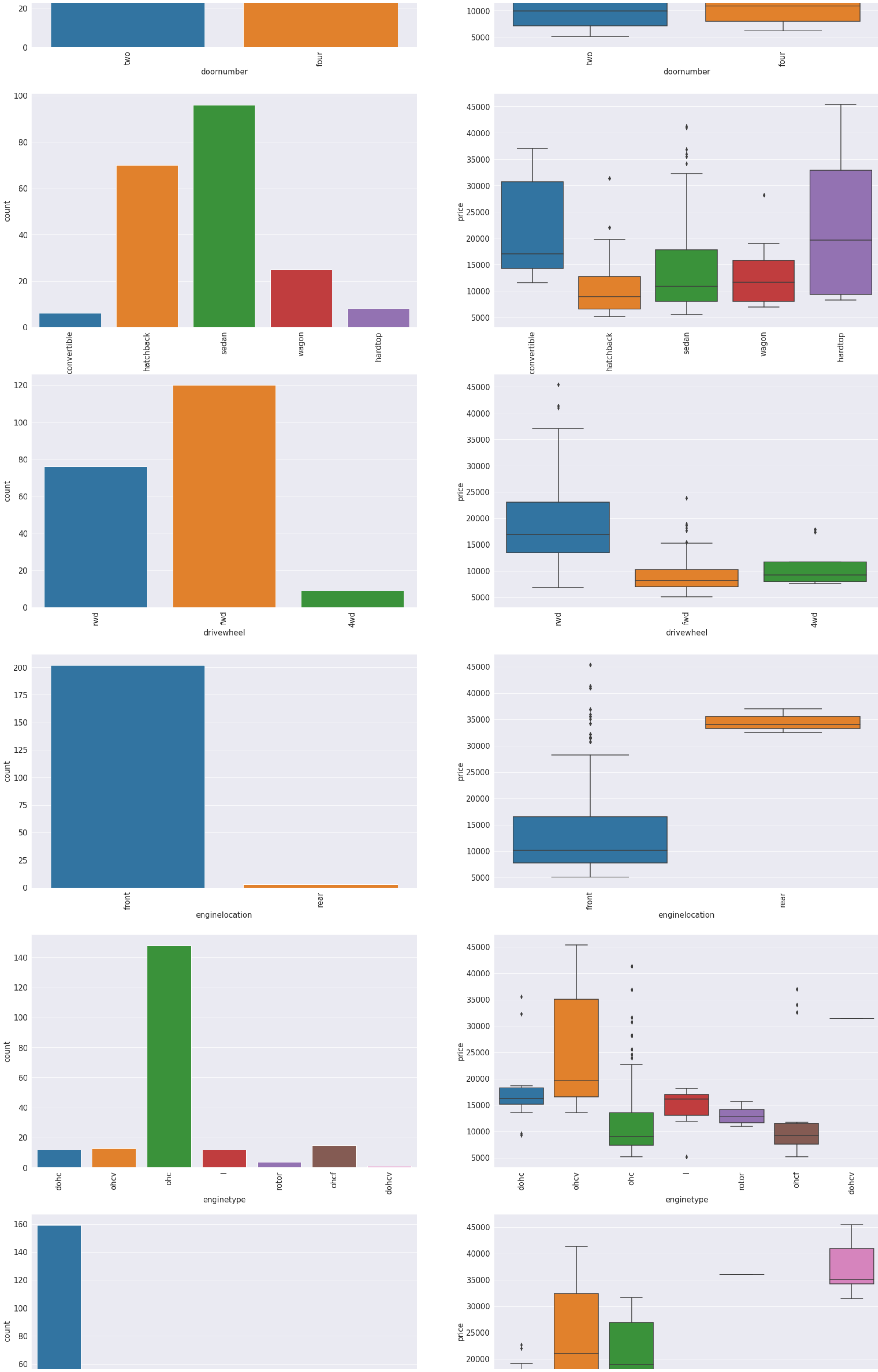
```
 'stroke',
 'compressionratio',
 'horsepower',
 'peakrpm',
 'citympg',
 'highwaympg']
```

Exploratory Data Analysis

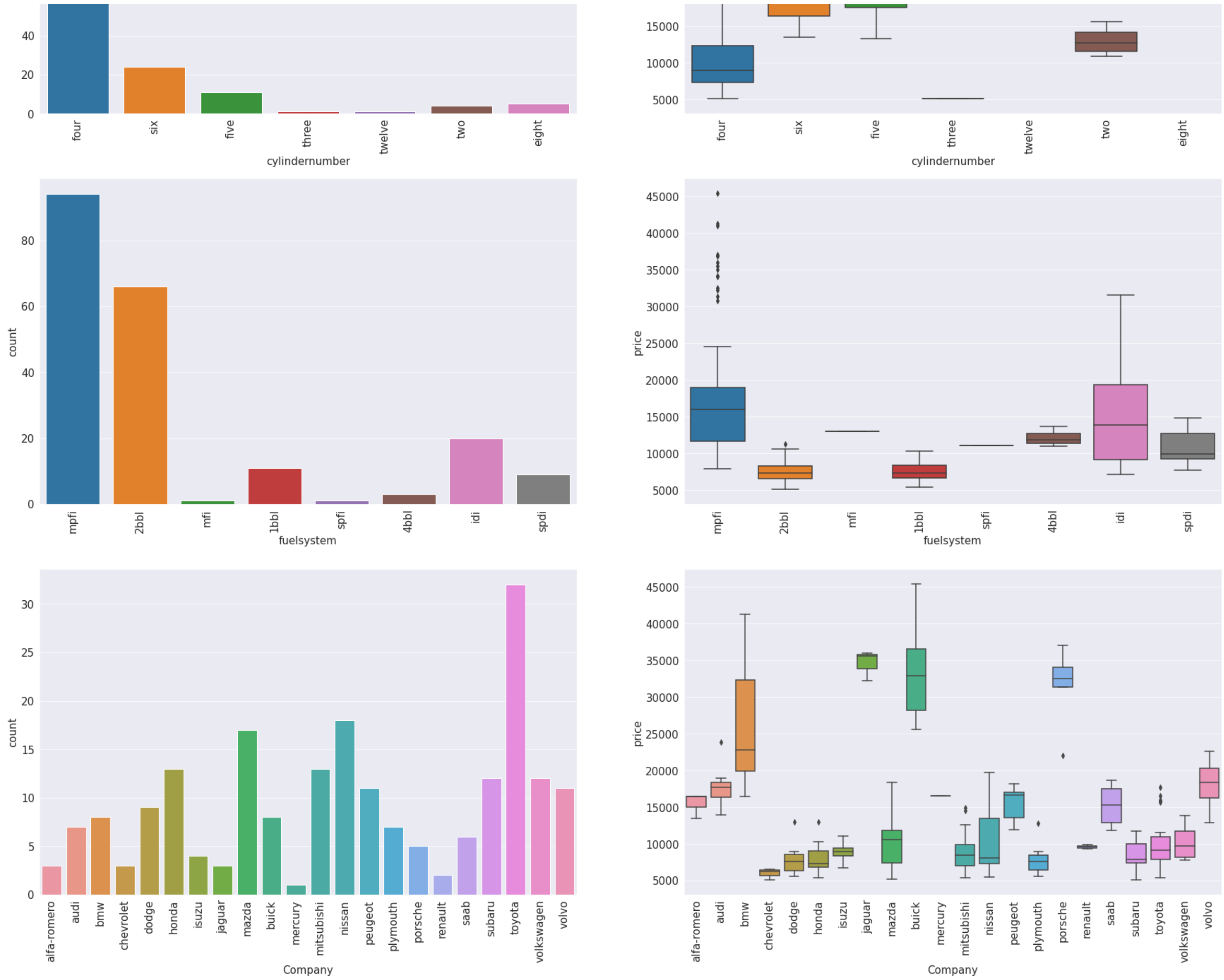
In [16]:

```
i=1
plt.figure(figsize=(30,100))
for col in cat_cols:
    plt.subplot(10,2,i)
    sns.countplot(df[col])
    plt.xticks(rotation=90, fontsize=15)
    plt.yticks(fontsize=15)
    plt.xlabel(col, fontsize=15)
    plt.ylabel('count',fontsize=15)

    i+=1
    plt.subplot(10,2,i)
    sns.boxplot(x=df[col], y=df['price'])
    plt.xticks(rotation=90, fontsize=15)
    plt.yticks(fontsize=15)
    plt.xlabel(col, fontsize=15)
    plt.ylabel('price',fontsize=15)
    i+=1
plt.show()
```





two
four
doornumber
count
price
convertible
hatchback
sedan
wagon
hardtop
rwd
fwd
4wd
drivewheel
front
rear
enginelocation
dohc
ohcv
ohc
l
rotor
ohcf
dohcv
enginetype



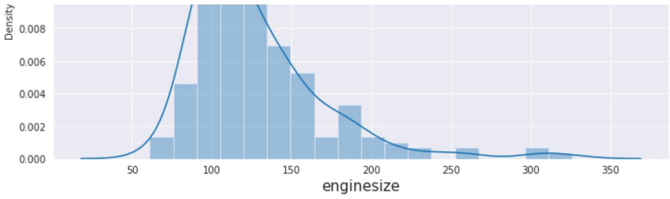
Observations

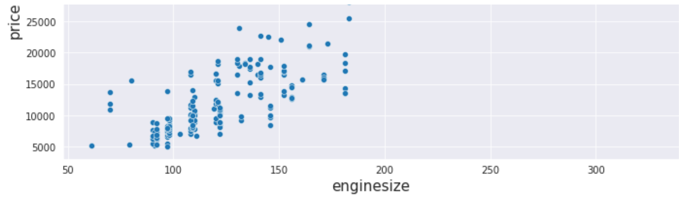
1. **Diesel cars tend to be slightly higher-priced than gas cars**
2. **The number of gas cars is significantly higher than diesel cars**
3. **Turbo cars are higher priced than std cars**
4. **Sedan and hatchbacks account for more than 75% of total cars**
5. **Real wheel drive cars are higher priced than other drive cars**
6. **The median price of cars with engine at back is over 3 times the median price of cars with engine in front**
7. **Majority of the cars have ohc engines but, ohcv engines seem to be higher priced**
8. **General trend is that the price of car increase with increase in number of cylinders**
9. **Toyota is the most liked car**
10. **Porsche, Jaguar, BMW, Buick are high range cars**

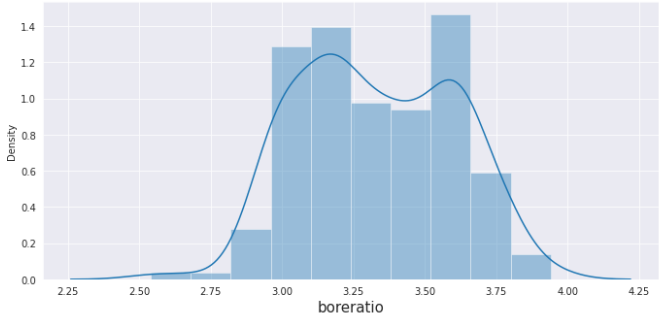
In [17]:

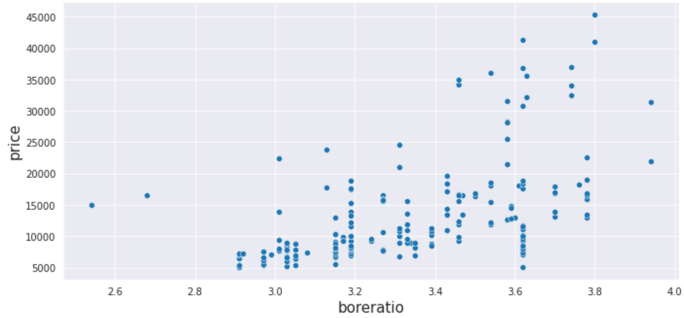
```
i=1
plt.figure(figsize=(25,100))
for col in num_cols:
    plt.subplot(16,2,i)
    sns.distplot(df[col])
    plt.xlabel(col,fontsize=15)
    plt.xticks(fontsize=10)
    i+=1

    plt.subplot(16,2,i)
    sns.scatterplot(x=df[col], y=df['price'])
    plt.xlabel(col,fontsize=15)
    plt.xticks(fontsize=10)
    plt.ylabel('price', fontsize=15)
```

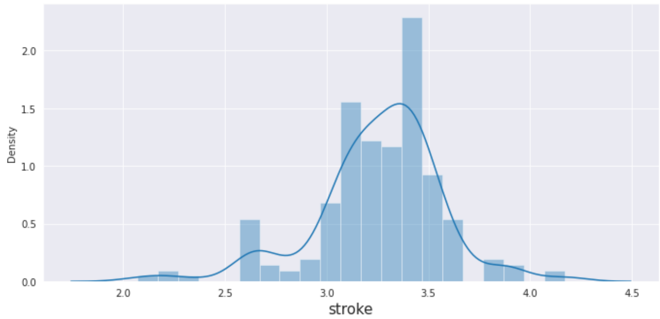
```
    plt.yticks(fontsize=10)
    i+=1
```

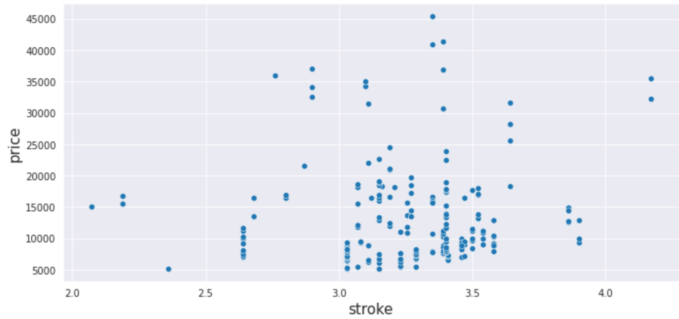

Density
enginesize

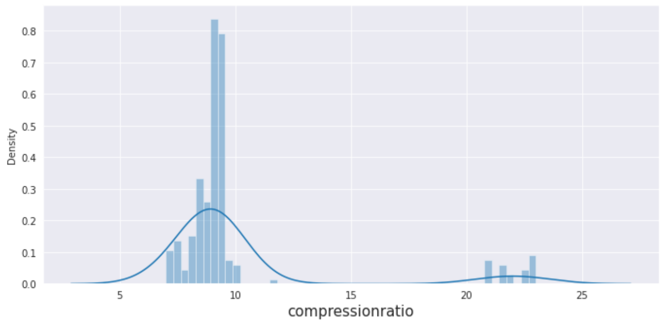

price
enginesize

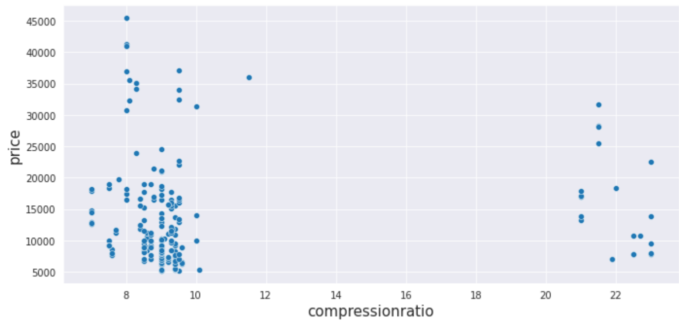

Density
boreratio

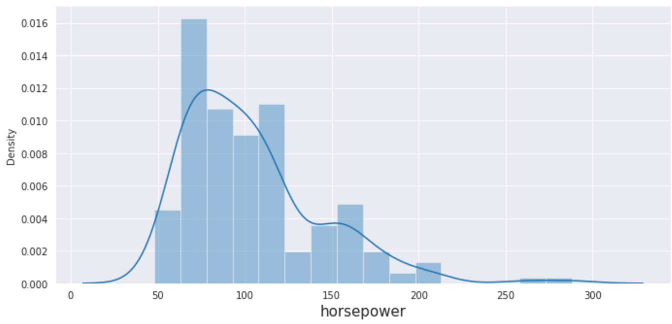

price
boreratio

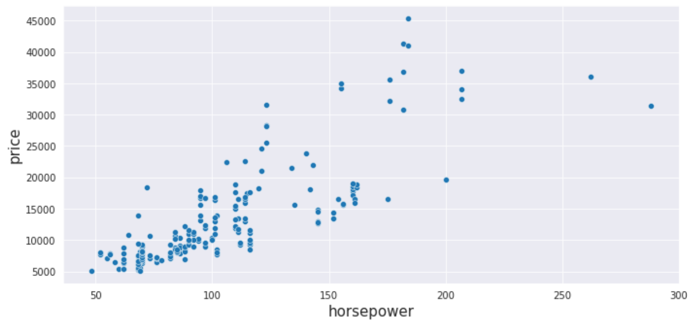

Density
stroke

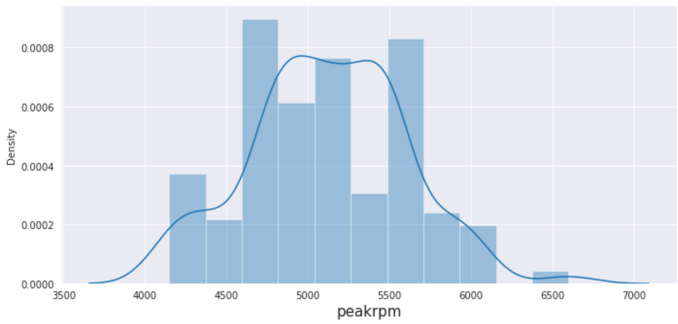

price
stroke

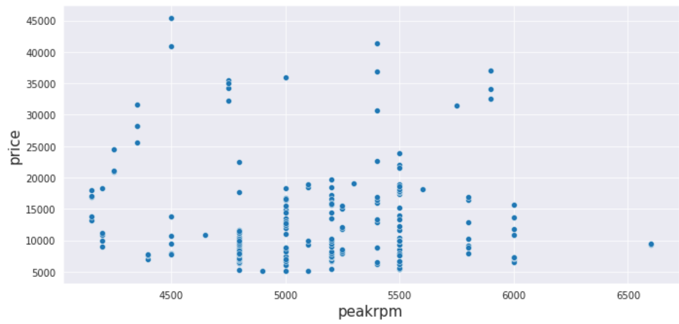

Density
compressionratio

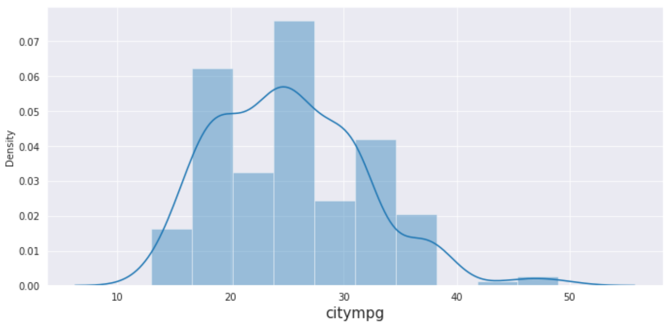

price
compressionratio

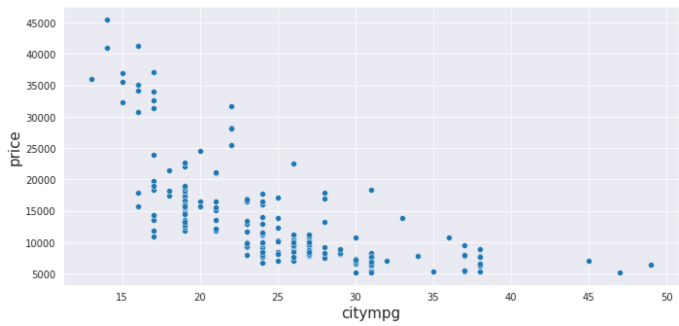

Density
horsepower

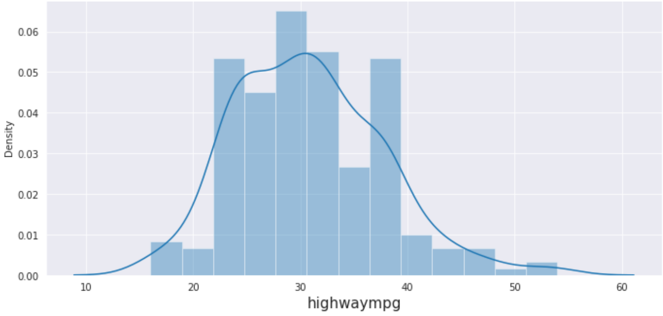

price
horsepower

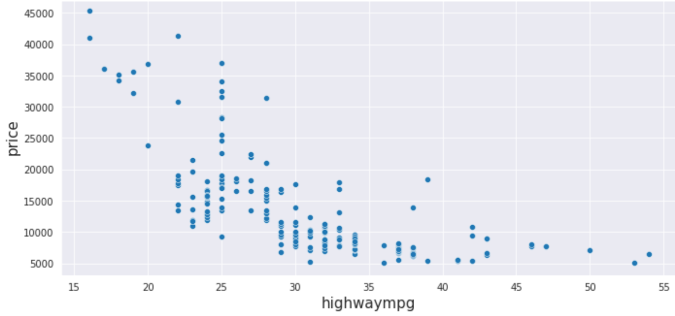

Density
peakrpm


price
peakrpm


Density
citympg


price
citympg





Observations

1. Positive co-relation between price and car-length,car width, curb weight, engine size, horsepower
2. Negative co-relation between price and city mileage, highway mileage
3. There is no relation of symboling with price, hence we drop the column

In [18]:

```
df.drop('symboling', axis=1, inplace=True)
num_cols.remove('symboling')
```

In [19]:

```
df.head()
```

Out[19]:

	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	carheight	...	fu
0	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	...	
1	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	...	
2	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	...	
3	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	...	
4	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	...	

5 rows × 24 columns

In [20]:

```
num_cols
```

Out[20]:

```
['wheelbase',
 'carlength',
 'carwidth',
 'carheight',
 'curbweight',
 'enginesize',
 'boreratio',
 'stroke',
 'compressionratio',
 'horsepower',
 'peakrpm',
 'citympg',
 'highwaympg']
```

Preprocessing

In [21]:

```
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()

df[cat_cols]= df[cat_cols].apply(lambda x: le.fit_transform(x))
```

In [22]:

```
df[cat_cols].head()
```

Out[22]:

	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	enginetype	cylindernumber	fuelsystem	Company
0	1	0	1	0	2	0	0	2	5	0
1	1	0	1	0	2	0	0	2	5	0
2	1	0	1	2	2	0	5	3	5	0
3	1	0	0	3	1	0	3	2	5	1
4	1	0	0	3	0	0	3	1	5	1

In [23]:

```
df.head()
```

Out[23]:

	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	carheight	...	fuels
0	1	0	1	0	2	0	88.6	168.8	64.1	48.8	...	
1	1	0	1	0	2	0	88.6	168.8	64.1	48.8	...	
2	1	0	1	2	2	0	94.5	171.2	65.5	52.4	...	
3	1	0	0	3	1	0	99.8	176.6	66.2	54.3	...	
4	1	0	0	3	0	0	99.4	176.6	66.4	54.3	...	

5 rows × 24 columns

In [24]:

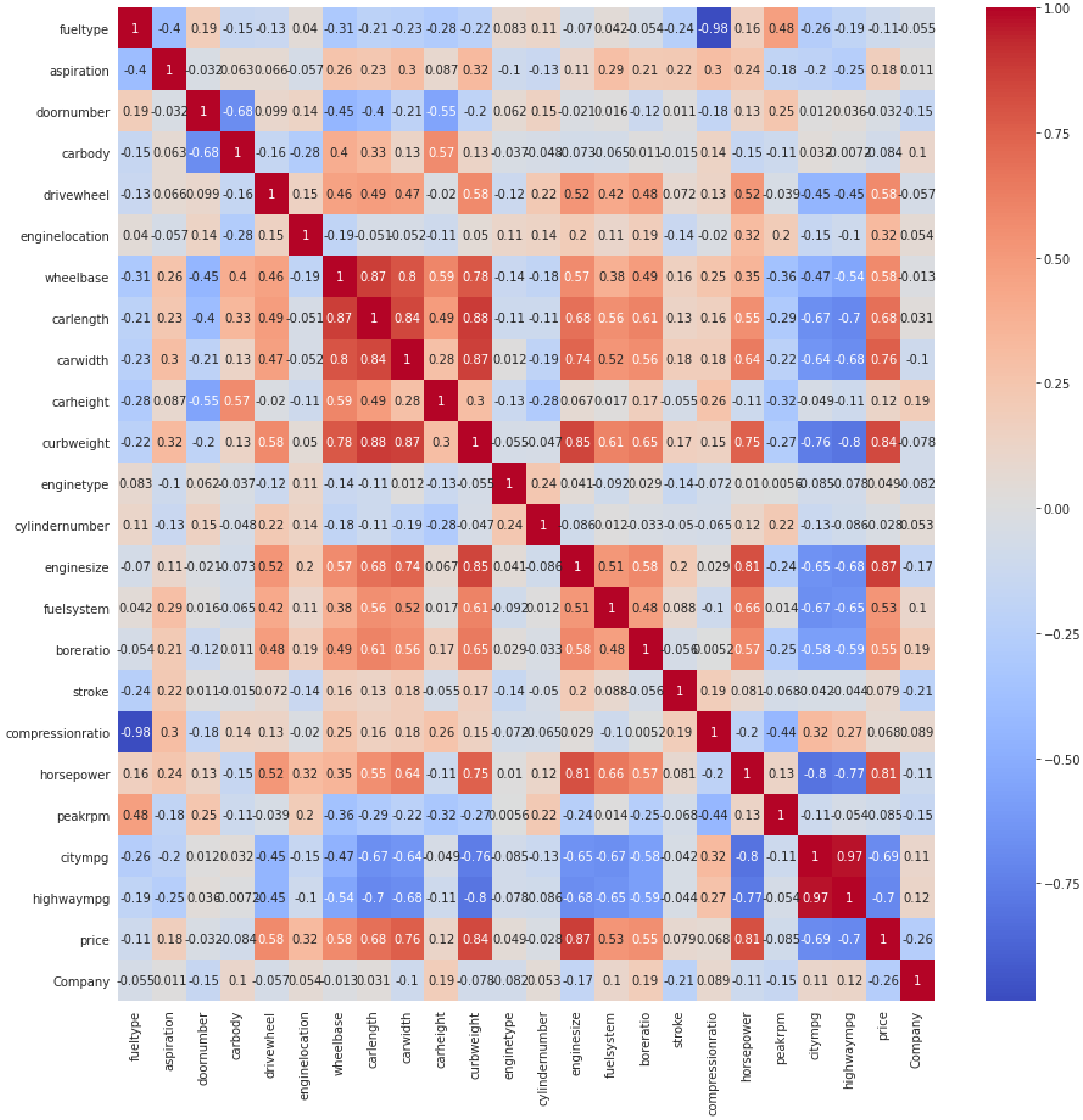
```
df.info()
```

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

```
 20  citympg           205 non-null    int64
 21  highwaympg        205 non-null    int64
 22  price             205 non-null    float64
 23  Company           205 non-null    int64
dtypes: float64(8), int64(16)
memory usage: 38.6 KB
```

In [25]:

```
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



In [26]:

```
from sklearn.preprocessing import StandardScaler

ss=StandardScaler()
df[num_cols]= ss.fit_transform(df[num_cols])
```

In [27]:

```
df.head()
```

```
df.head()
```

Out[27]:

	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	carheight	...	fuels
0	1	0	1	0	2	0	-1.690772	-0.426521	-0.844782	-2.020417	...	
1	1	0	1	0	2	0	-1.690772	-0.426521	-0.844782	-2.020417	...	
2	1	0	1	2	2	0	-0.708596	-0.231513	-0.190566	-0.543527	...	
3	1	0	0	3	1	0	0.173698	0.207256	0.136542	0.235942	...	
4	1	0	0	3	0	0	0.107110	0.207256	0.230001	0.235942	...	

5 rows × 24 columns

Training our Model

In [28]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

In [29]:

```
X= df.drop('price', axis=1)
y=df['price']

X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2, random_state= 42)
```

In [30]:

```
lr= LinearRegression()
lr.fit(X_train,y_train)
y_pred= lr.predict(X_test)

rmse= (mean_squared_error(y_test,y_pred))**(1/2)
r2= r2_score(y_test,y_pred)

print(rmse)
print(r2)
```

```
3483.207163635308
0.8463122094668135
```

In [31]:

```
dt= DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=100, min_samples_le
af= 5, random_state=42)
dt.fit(X_train,y_train)
y_pred= dt.predict(X_test)

rmse_train= (mean_squared_error(dt.predict(X_train),y_train))**(1/2)
rmse= (mean_squared_error(y_test,y_pred))**(1/2)
r2= r2_score(y_test,y_pred)
print(rmse_train)
print(rmse)
print(r2)
```

```
1546.3852781211913
```

```
2708.767381032721
0.9070553964342399
```

In [32]:

```
rf= RandomForestRegressor(max_depth=10, criterion='mse', min_samples_leaf=2, random_stat
e=42, verbose=1)
rf.fit(X_train,y_train)
y_pred= rf.predict(X_test)
rmse_train= (mean_squared_error(rf.predict(X_train),y_train))**(1/2)
rmse= (mean_squared_error(y_test,y_pred))**(1/2)
r2= r2_score(y_test,y_pred)
print(rmse_train)
print(rmse)
print(r2)
```

```
1182.8418416395994
1953.5505961315666
0.9516573910332784
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [33]:

```
xgb= XGBRegressor()
xgb.fit(X_train, y_train)
y_pred= xgb.predict(X_test)
rmse_train= (mean_squared_error(xgb.predict(X_train),y_train))**(1/2)
rmse= (mean_squared_error(y_test,y_pred))**(1/2)
r2= r2_score(y_test,y_pred)
print(rmse_train)
print(rmse)
print(r2)
```

```
283.4366075958827
2472.360172889496
0.9225708957261225
```

Conclusion

All our models are overfitting the data. Tried many different combinations with GridSearchCV but there is still overfitting. This is because the amount of data is very less. However, with the given data, RandomForestRegressor is the best fit

In [ ]:

```

```