In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import warnings

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 200)
plt.style.use('ggplot')
```

In [2]:
```
df = pd.read_csv("data/CarPrice_Assignment.csv")
df.head()
```

Out[2]:

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

- Nominal = [CarName, carbody, drivewheel, enginelocation, enginetype, fuelsystem]
- ordinal = [Insurance, fueltype, doornumber]

EDA - Exploratory data analysis

Understanding data

In [3]:
```
df.columns
```

Out[3]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

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

Out[4]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	bore
count	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	3.32
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.27
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.54
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.15
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.31
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.58
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.94

Symboling : means the rate in insurance companies. It is always in the range [-3,3] **such that :** {{-3, "high risk"}, {-2, "moderately high-risk"}, {-1, "somewhat high-risk"}, {0, "average risk level"}, {1, "somewhat low-risk"}, {2, "moderately low-risk"}, {3, "low risk"}}

- **To-do** : Change the Symboling column name to Insurance

Wheelbase: The distance between the centers of the front and rear wheels of the car. It is an important dimension that affects stability, ride comfort, and other performance characteristics of the vehicle.

Curb weight : The weight of the car when it's ready for use, including all standard equipment and a full tank of fuel. It's a critical parameter for performance and fuel efficiency. (Total_Weight)

- **To-Do :** Change the curb weight column name to Total_weight

Bore ratio refers to the ratio of the diameter of the engine's cylinders to the length of the stroke. It's a design parameter that influences engine performance and efficiency.

Stroke : The distance traveled by the piston inside the engine cylinder from top to bottom during each engine cycle. It's another design parameter that affects engine characteristics.

peakrpm : The engine speed at which it produces its maximum power.

- **To-Do :** Change the peakrpm to Maxrpm or MaxEngineSpeed*

citympg: Represents the estimated fuel efficiency of the car when driven in city or urban conditions.

highwaympg: Represents the estimated fuel efficiency of the car when driven on highways.

In [5]:

```
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  
 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 [6]:

```python
df.shape
```

Out[6]:

```
(205, 26)
```

Data preperation - Preprocessing

In [7]:

```python
#pd.set_option('display.max_columns', 100)
df.head()
```

Out[7]:

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

In [8]:

```python
df = df.rename(columns=
          {"symboling":"Insurance_level",
          "curbweight":"Total_weight",
          "peakrpm":"Maxrpm"}
)
df.head()
```

Out[8]:

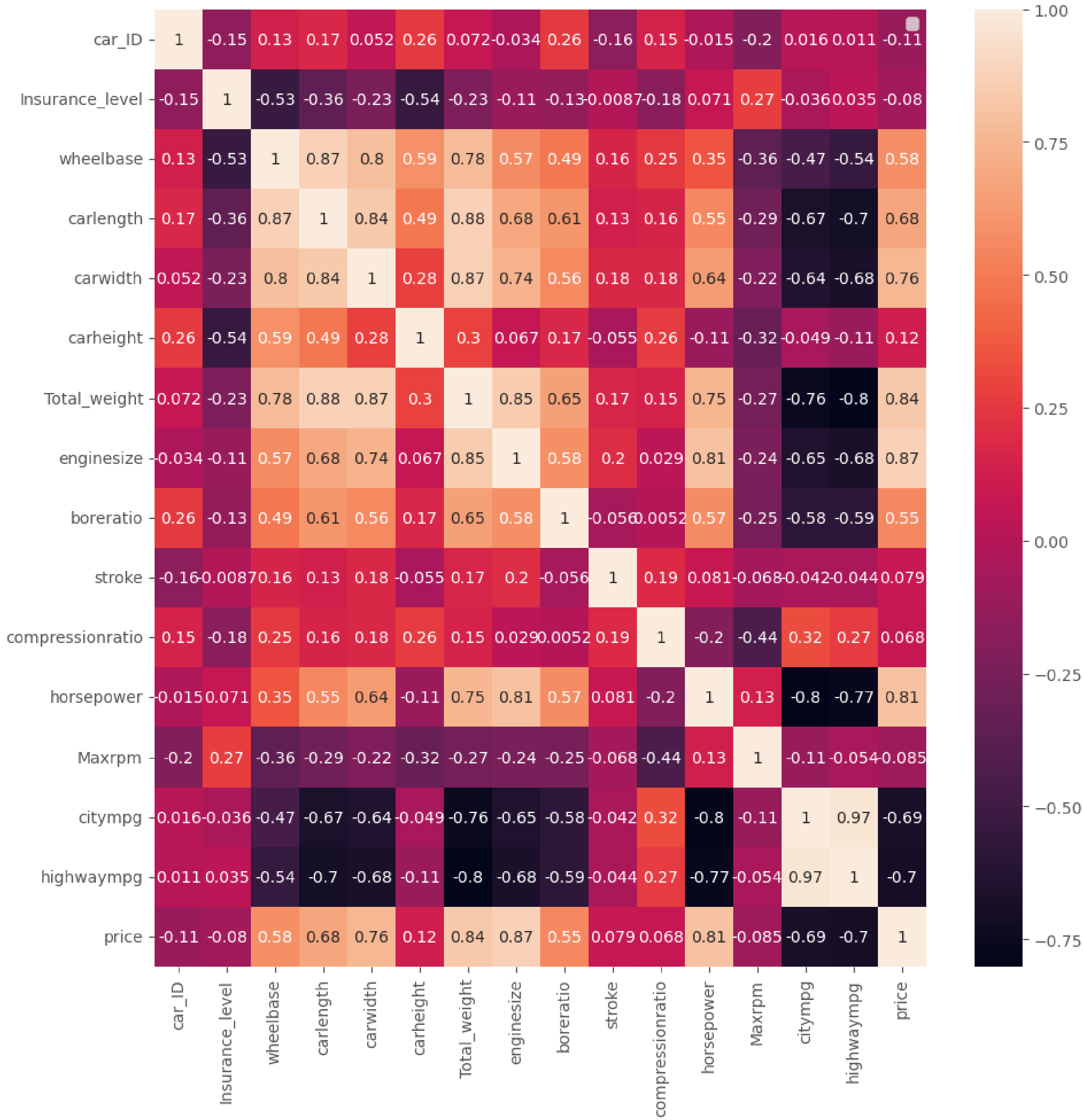
	car_ID	Insurance_level	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front
3	4	2	audi 100 ls	gas	std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

Calculate the correlation between features

In [9]:

```
cm = df.corr()
fig = plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with
an underscore are ignored when legend() is called with no argument.
```



Drop irrelevant columns

In [10]:
```
df = df[[#'car_ID',
    'Insurance_level', #'CarName',
        'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', #'enginelocation',
        'wheelbase',
       'carlength', 'carwidth', 'carheight', 'Total_weight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', #'stroke',
       #'compressionratio',
    'horsepower', 'Maxrpm', 'citympg', 'highwaympg',
       'price']]
```

In [11]:
```
df.head()
```

Out[11]:

	Insurance_level	fueltype	aspiration	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	ca
0	3	gas	std	two	convertible	rwd	88.6	168.8	64.1	
1	3	gas	std	two	convertible	rwd	88.6	168.8	64.1	
2	1	gas	std	two	hatchback	rwd	94.5	171.2	65.5	
3	2	gas	std	four	sedan	fwd	99.8	176.6	66.2	
4	2	gas	std	four	sedan	4wd	99.4	176.6	66.4	

Handling duplicates

In [12]:
```
#df[df["CarName"].duplicated()]
```

In [13]:
```
#df.loc[df["CarName"]=="audi 100ls"]
```

In [14]:
```
#df = df.loc[~df.duplicated(subset=["CarName", "carbody", "cylindernumber"])]
df.shape
```

Out[14]:
```
(205, 21)
```

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

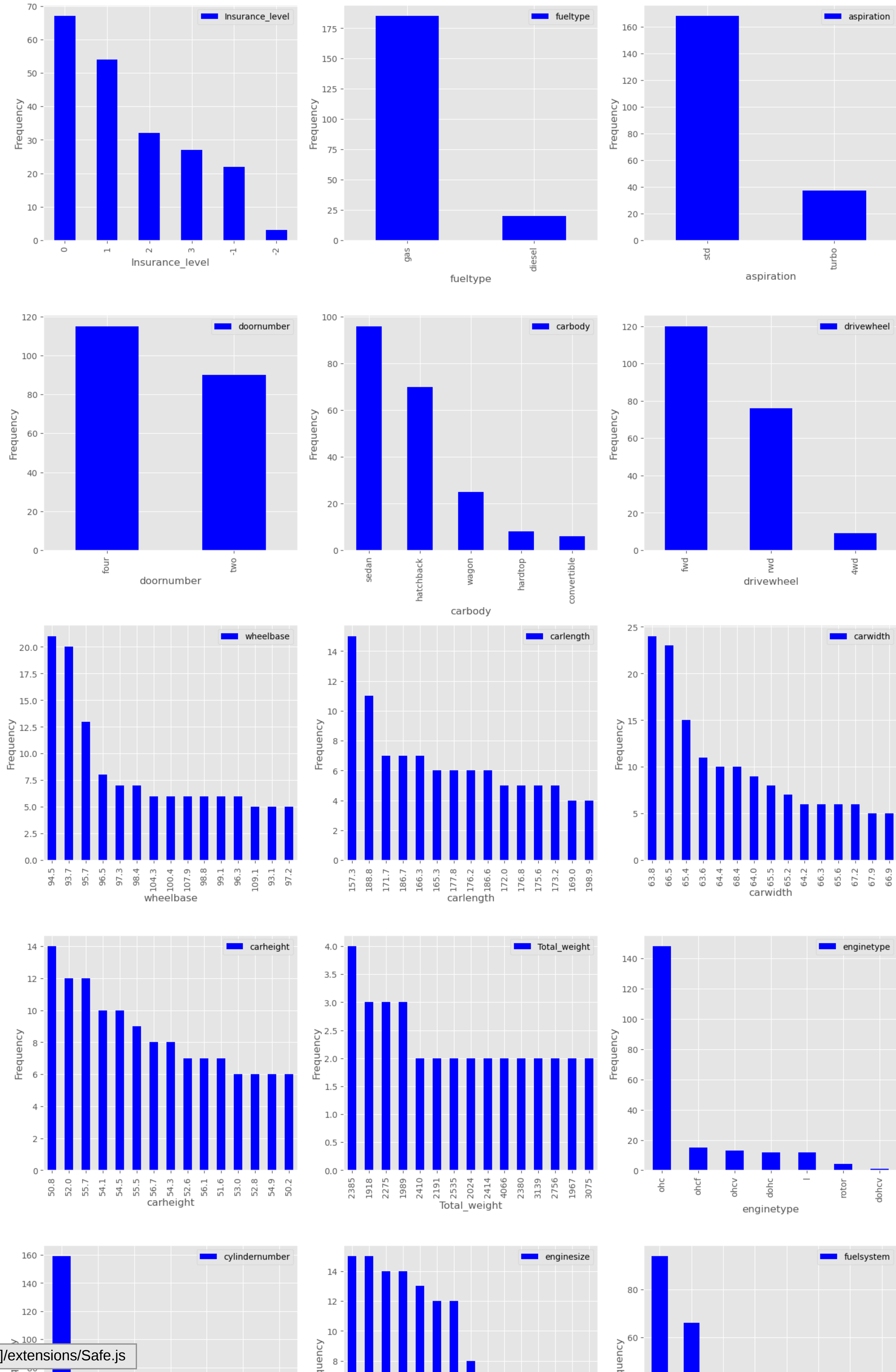
Out[15]:

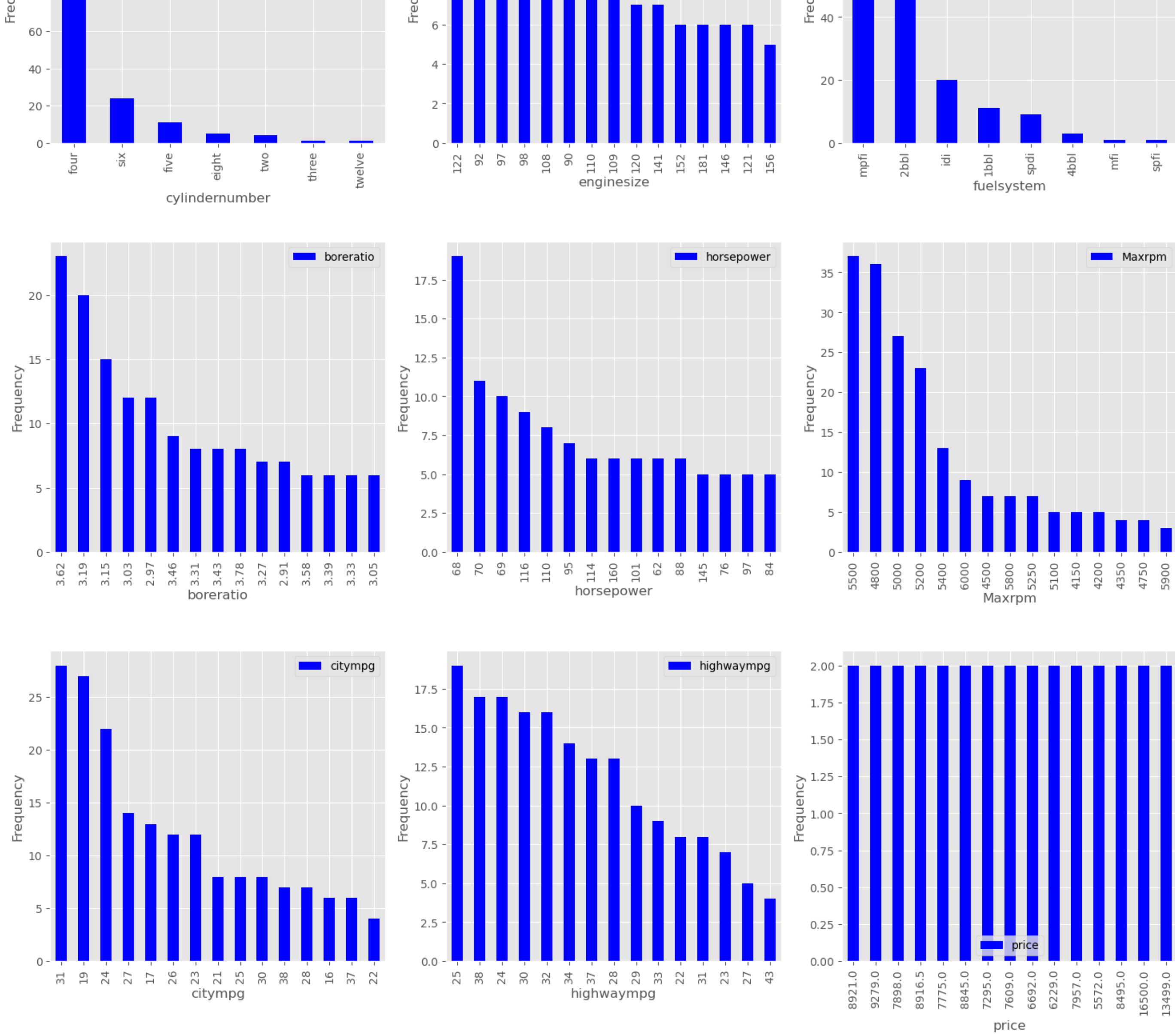
	Insurance_level	fueltype	aspiration	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	ca
0	3	gas	std	two	convertible	rwd	88.6	168.8	64.1	
1	3	gas	std	two	convertible	rwd	88.6	168.8	64.1	
2	1	gas	std	two	hatchback	rwd	94.5	171.2	65.5	
3	2	gas	std	four	sedan	fwd	99.8	176.6	66.2	
4	2	gas	std	four	sedan	4wd	99.4	176.6	66.4	

Understand Features - Visualization

distribution of the data

In [16]:
```
fig = plt.figure(figsize=(15,40))
for i in range(len(df.columns)):
    plt.subplot(8, 3, i+1)
    df[df.columns[i]].value_counts().head(15).plot(kind='bar', color="blue")
    plt.xlabel(df.columns[i])
    plt.ylabel("Frequency")
    plt.legend()
    plt.tight_layout()
plt.show()
```


Insurance_level
Frequency
0
1
2
3
-1
-2
fueltype
gas
diesel
aspiration
std
turbo
doornumber
four
two
carbody
sedan
hatchback
wagon
hardtop
convertible
drivewheel
fwd
rwd
4wd
wheelbase
carlength
carwidth
carheight
Total_weight
enginetype
ohc
ohcf
ohcv
dohc
l
rotor
dohcv
cylindernumber
enginesize
fuelsystem

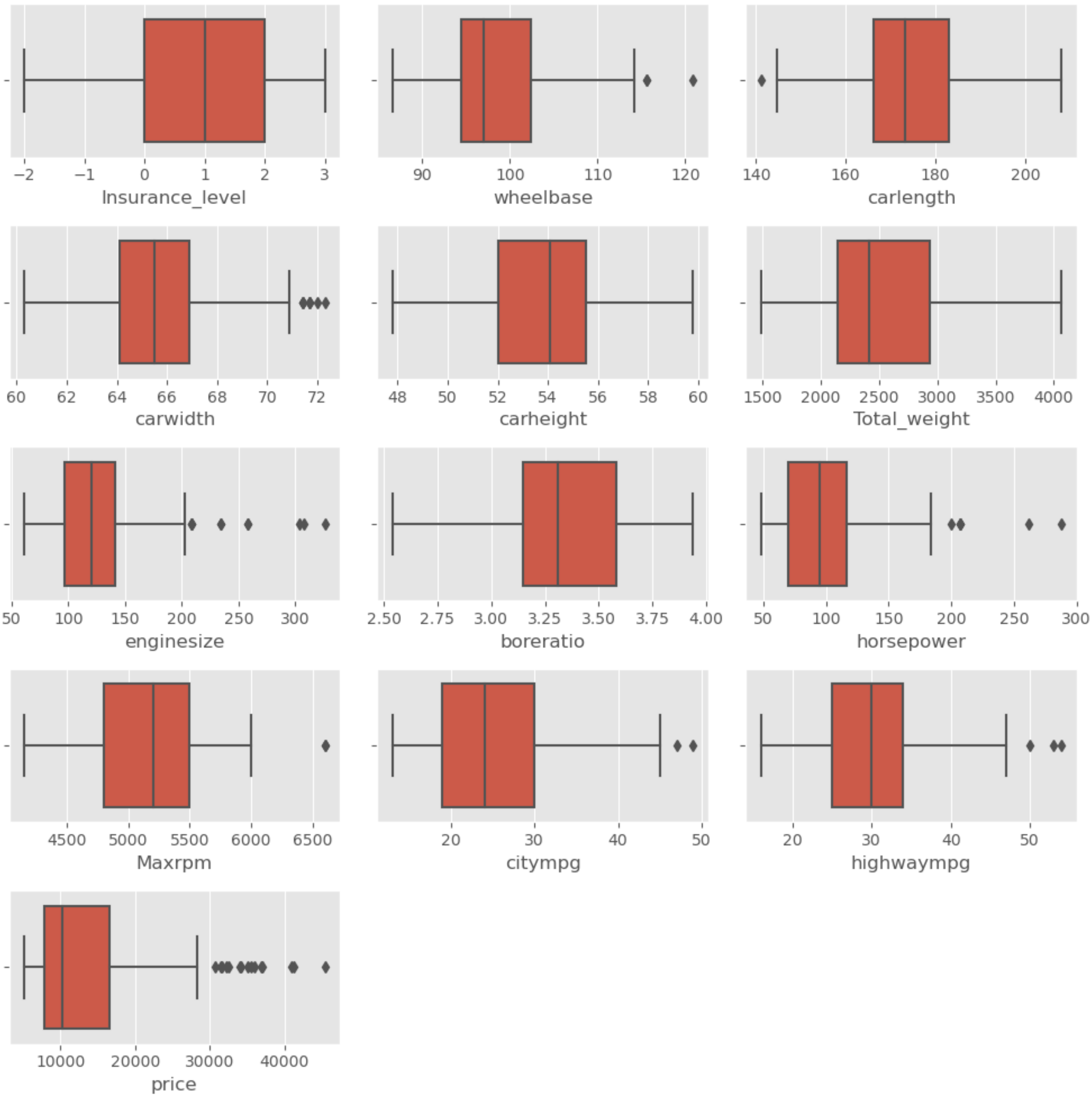


Box Plot for numerical data

In [17]:
```
#sns.boxplot(df["wheelbase"])
numerical_features = []

for col in df.columns:
    if df[col].dtype == np.int64 or df[col].dtype == np.float64:
        numerical_features.append(col)
```

In [18]:
```
fig = plt.figure(figsize=(10,10))
for i in range(len(numerical_features)):
    plt.subplot(5,3,i+1)
    sns.boxplot(df[numerical_features[i]])
    plt.tight_layout()
```



handling missing values

In [19]:
```
missing = df.isnull().sum()
missing
```

Out[19]:
```
Insurance_level    0
fueltype           0
aspiration         0
doornumber         0
carbody            0
drivewheel         0
wheelbase          0
carlength          0
carwidth           0
carheight          0
Total_weight       0
enginetype         0
cylindernumber     0
enginesize         0
fuelsystem         0
boreratio          0
horsepower         0
Maxrpm             0
citympg            0
highwaympg         0
price              0
dtype: int64
```

handling outliers

In [20]:
```
df.shape
```

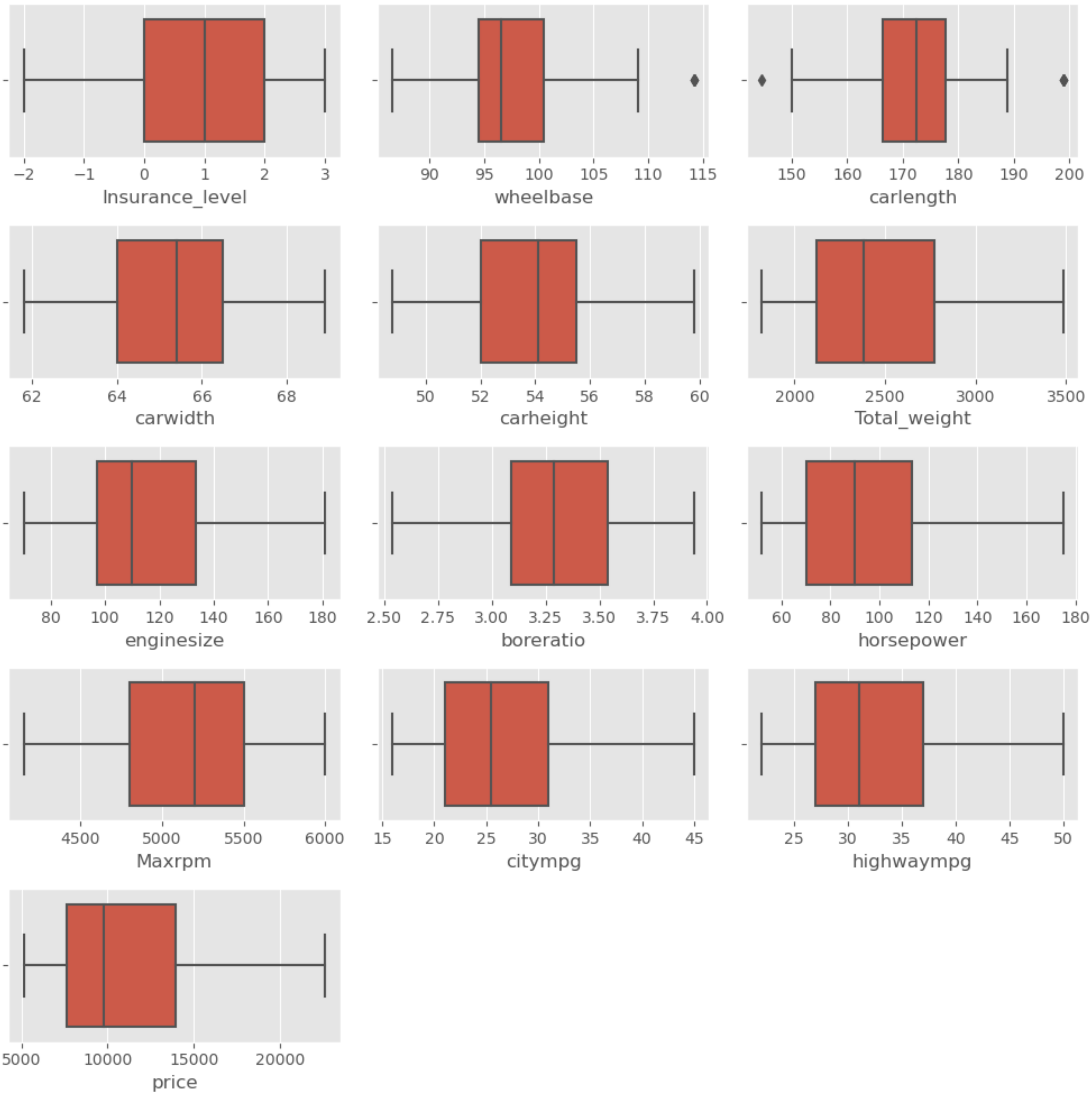
Out[20]:
```
(205, 21)
```

In [21]:
```
# remove the outliers is better idea as we don't have many outliers

for i in numerical_features:
    Q3 = np.percentile(df[i], 75)
    Q1= np.percentile(df[i], 25)
    IQR = Q3 - Q1
    Max_Val = Q3 + (1.5*IQR)
    Min_Val = Q1 - (1.5*IQR)
    df.drop(df[df[i] > Max_Val].index, inplace=True)
    df.drop(df[df[i] < Min_Val].index, inplace=True)
```

In [22]:
```
#box plot after removing the outliers

fig = plt.figure(figsize=(10,10))
for i in range(len(numerical_features)):
    plt.subplot(5,3,i+1)
    sns.boxplot(df[numerical_features[i]])
    plt.tight_layout()
```



In [23]:
```
df.shape
```

Out[23]:
```
(178, 21)
```

encoding non numerical data

In [24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 178 entries, 0 to 204
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Insurance_level  178 non-null    int64  
 1   fueltype         178 non-null    object 
 2   aspiration       178 non-null    object 
 3   doornumber       178 non-null    object 
 4   carbody          178 non-null    object 
 5   drivewheel       178 non-null    object 
 6   wheelbase        178 non-null    float64
 7   carlength        178 non-null    float64
 8   carwidth         178 non-null    float64
 9   carheight        178 non-null    float64
 10  Total_weight     178 non-null    int64  
 11  enginetype       178 non-null    object 
 12  cylindernumber   178 non-null    object 
 13  enginesize       178 non-null    int64  
 14  fuelsystem       178 non-null    object 
 15  boreratio        178 non-null    float64
 16  horsepower       178 non-null    int64  
 17  Maxrpm           178 non-null    int64  
 18  citympg          178 non-null    int64  
 19  highwaympg       178 non-null    int64  
 20  price            178 non-null    float64
dtypes: float64(6), int64(7), object(8)
memory usage: 30.6+ KB
```

In [25]:
```python
df.head()
```

Out[25]:

	Insurance_level	fueltype	aspiration	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	ca
0	3	gas	std	two	convertible	rwd	88.6	168.8	64.1	
1	3	gas	std	two	convertible	rwd	88.6	168.8	64.1	
2	1	gas	std	two	hatchback	rwd	94.5	171.2	65.5	
3	2	gas	std	four	sedan	fwd	99.8	176.6	66.2	
4	2	gas	std	four	sedan	4wd	99.4	176.6	66.4	

In [27]:
```python
# Nominal_Features = [CarName, carbody, drivewheel, enginelocation, enginetype, fuelsyst
# ordinal_Features = [doornumber, cylindernumber]

# Label encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df["doornumber"] = encoder.fit_transform(df["doornumber"])
df["cylindernumber"] = encoder.fit_transform(df["cylindernumber"])

# One Hot Encoding

dummies = [#"CarName",
    "carbody", "drivewheel", "enginetype", "fuelsystem", "aspiration", "fueltype"]
temp = pd.get_dummies(df[dummies], drop_first=True)
df  = pd.concat([df, temp], axis=1)
df.drop(dummies, axis = 1, inplace = True)
df.head()
```

Out[27]:

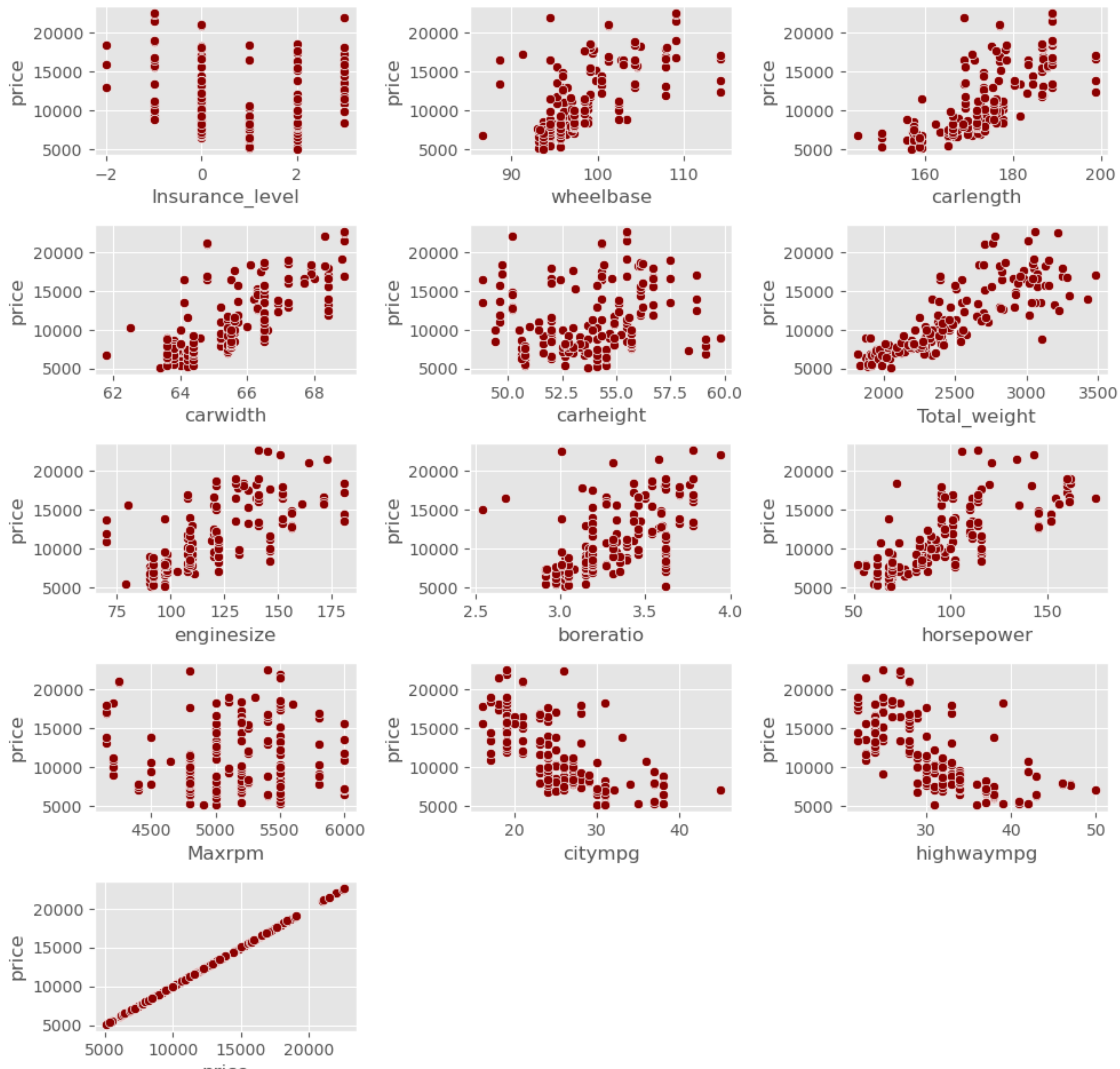
	Insurance_level	doornumber	wheelbase	carlength	carwidth	carheight	Total_weight	cylindernumber	engin
0	3	1	88.6	168.8	64.1	48.8	2548	1	
1	3	1	88.6	168.8	64.1	48.8	2548	1	
2	1	1	94.5	171.2	65.5	52.4	2823	2	
3	2	0	99.8	176.6	66.2	54.3	2337	1	
4	2	0	99.4	176.6	66.4	54.3	2824	0	

Relationships between Features

Scatter plot

In [28]:

```
fig = plt.figure(figsize=(10,15))
for i in range(len(numerical_features)):
    plt.subplot(8,3,i+1)
    sns.scatterplot(df[numerical_features[i]], df["price"], color="darkred")
    plt.tight_layout()
```



Correlation between features

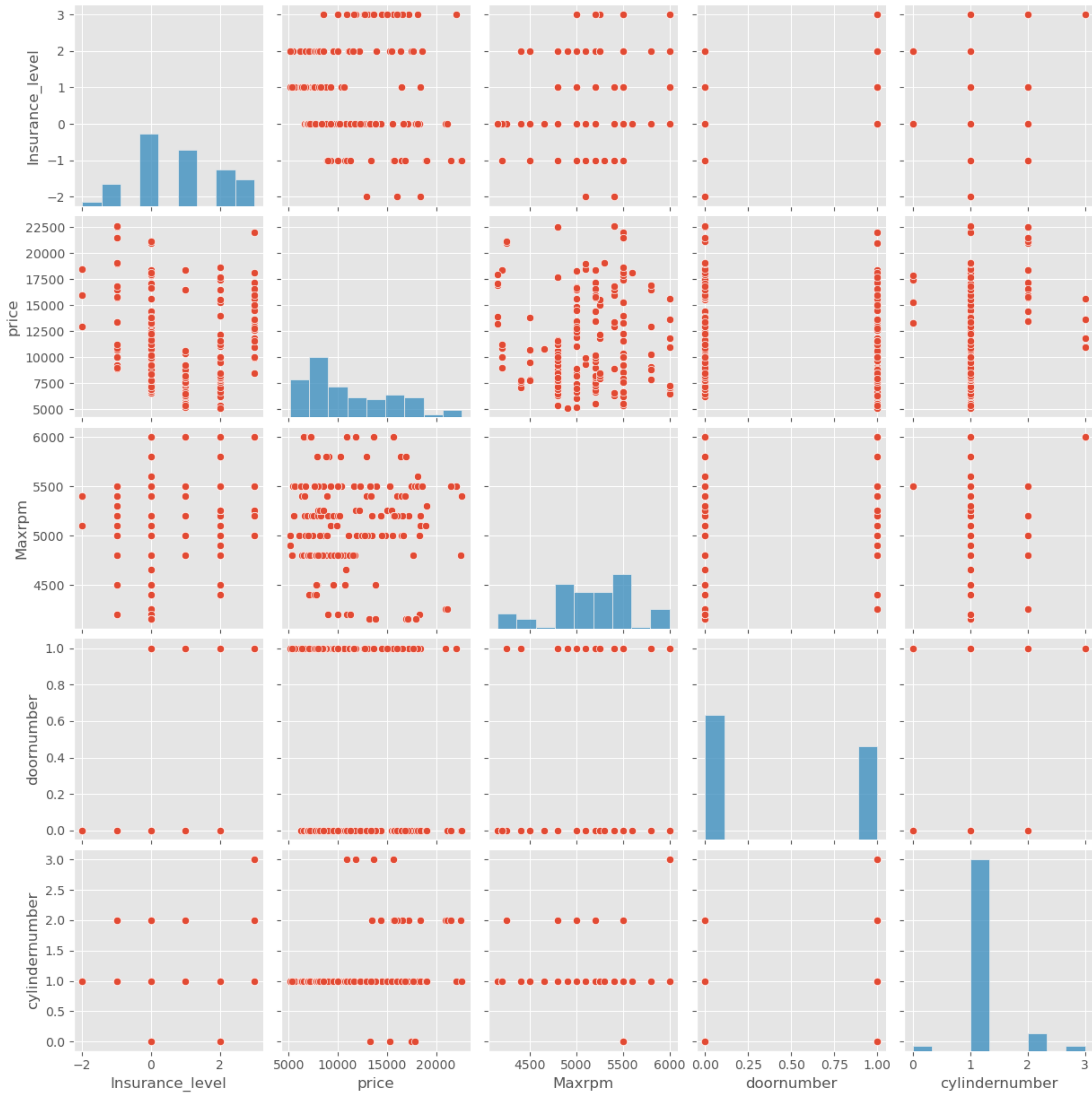
Finding correlation with significance for population

In [29]:
```
chosen_features = []
for i in df.columns:
    r, p = stats.pearsonr(df[i], df["price"])
    print(f"Feature : {i}, corr = {r}, significant : {p<0.05}")
    if p < 0.05 :
        chosen_features.append(i)
```

```
Feature : Insurance_level, corr = -0.0939057875591427, significant : False
Feature : doornumber, corr = -0.11299310200870943, significant : False
Feature : wheelbase, corr = 0.6417777202291751, significant : True
Feature : carlength, corr = 0.7189803519379578, significant : True
Feature : carwidth, corr = 0.7568765895308719, significant : True
Feature : carheight, corr = 0.19865692023563855, significant : True
Feature : Total_weight, corr = 0.834329375925847, significant : True
Feature : cylindernumber, corr = 0.2591185989258008, significant : True
Feature : enginesize, corr = 0.7090085061098484, significant : True
Feature : boreratio, corr = 0.5319481001137975, significant : True
Feature : horsepower, corr = 0.7563813087873641, significant : True
Feature : Maxrpm, corr = -0.05434753457368498, significant : False
Feature : citympg, corr = -0.6992034249177898, significant : True
Feature : highwaympg, corr = -0.6962594714589706, significant : True
Feature : price, corr = 0.9999999999999999, significant : True
Feature : carbody_hardtop, corr = -0.059043397349757054, significant : False
Feature : carbody_hatchback, corr = -0.1760424467423848, significant : True
Feature : carbody_sedan, corr = 0.12850725346032907, significant : False
Feature : carbody_wagon, corr = 0.02962367491532852, significant : False
Feature : drivewheel_fwd, corr = -0.6420114282940873, significant : True
Feature : drivewheel_rwd, corr = 0.6672404332657127, significant : True
Feature : enginetype_l, corr = 0.26302683208968536, significant : True
Feature : enginetype_ohc, corr = -0.32479415054754074, significant : True
Feature : enginetype_ohcf, corr = -0.15719135604766915, significant : True
Feature : enginetype_ohcv, corr = 0.2512534468302309, significant : True
Feature : enginetype_rotor, corr = 0.06866516043026794, significant : False
Feature : fuelsystem_2bbl, corr = -0.6239044424990882, significant : True
Feature : fuelsystem_4bbl, corr = 0.03275626981639924, significant : False
Feature : fuelsystem_idi, corr = 0.11896998537037169, significant : False
Feature : fuelsystem_mfi, corr = 0.03306529846454912, significant : False
Feature : fuelsystem_mpfi, corr = 0.619794441554144, significant : True
Feature : fuelsystem_spdi, corr = -0.003987197299105446, significant : False
Feature : fuelsystem_spfi, corr = -0.000296513166700464, significant : False
Feature : aspiration_turbo, corr = 0.352999605665082, significant : True
Feature : fueltype_gas, corr = -0.11896998537037169, significant : False
```

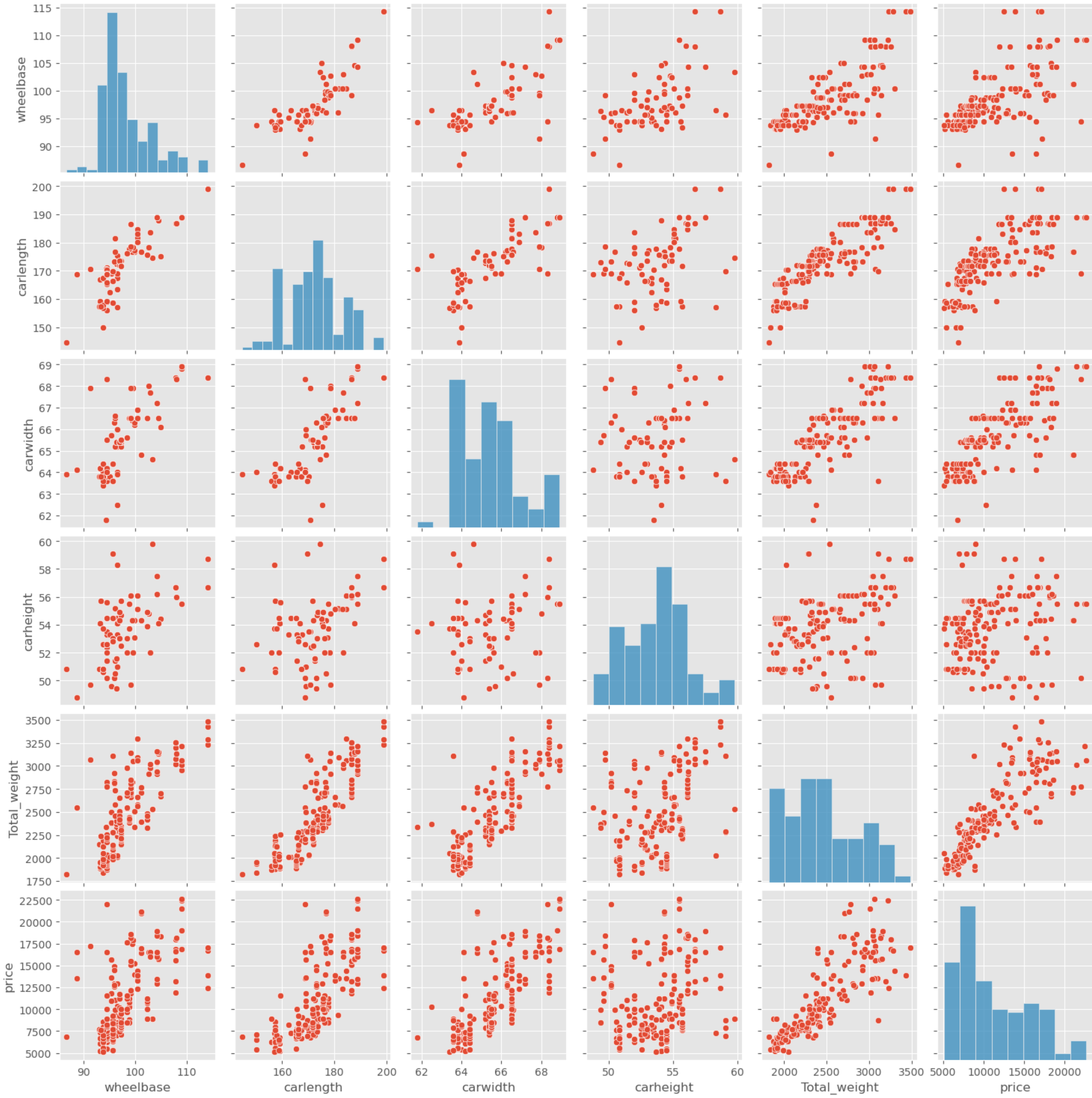
Pairplot to find relationships between variables

In [30]:
```
sns.pairplot(df, vars=["Insurance_level", "price", "Maxrpm", "doornumber", "cylindernumb
plt.show()
```



In [31]:
```
sns.pairplot(df, vars=["wheelbase", "carlength", "carwidth", "carheight", "Total_weight"
plt.show()
```

Loading [MathJax]/extensions/Safe.js



Choose the features

In [32]:
```
df = df[chosen_features]
df.head()
```

Out[32]:

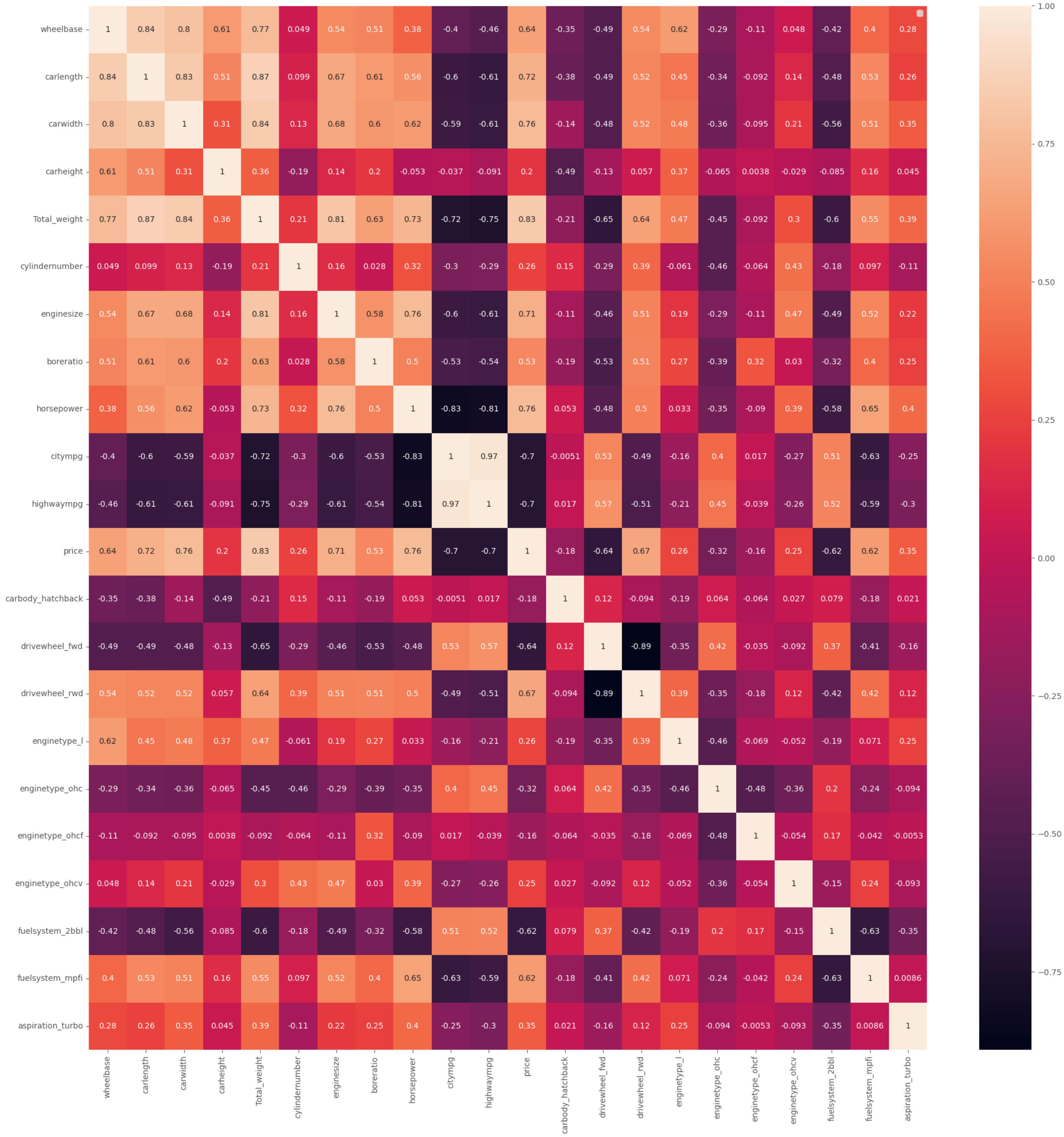
	wheelbase	carlength	carwidth	carheight	Total_weight	cylindernumber	enginesize	boreratio	horsepower
0	88.6	168.8	64.1	48.8	2548	1	130	3.47	111
1	88.6	168.8	64.1	48.8	2548	1	130	3.47	111
2	94.5	171.2	65.5	52.4	2823	2	152	2.68	154
3	99.8	176.6	66.2	54.3	2337	1	109	3.19	102
4	99.4	176.6	66.4	54.3	2824	0	136	3.19	115

In [33]:
```
df.shape
```

Out[33]:
```
(178, 22)
```

In [34]:
```python
cm = df.corr()
fig = plt.figure(figsize=(20,20))
sns.heatmap(cm, annot=True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with
an underscore are ignored when legend() is called with no argument.
```



Define input/output

In [35]:
```python
X = df.drop(["price"], 1)
y = df["price"]
```


nto train and test

In [36]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22
```

Scale the data

In [37]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Create the mdoel

Linear Regression model

In [38]:
```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures

model = LinearRegression()
model.fit(X_train, y_train)
```

Out[38]:
```
LinearRegression()
```

In [39]:
```
from sklearn.metrics import r2_score, mean_absolute_error

acc_train = r2_score(y_train, model.predict(X_train))
acc_train
```

Out[39]:
```
0.8335520137211543
```

In [40]:
```
acc_test = r2_score(y_test, model.predict(X_test))
acc_test
```

Out[40]:
```
0.7506005373119671
```

Lasso model

In [41]:
```
model = Lasso()
model.fit(X_train, y_train)
```

Out[41]:
```
Lasso()
```

In [42]:
```
acc_train = r2_score(y_train, model.predict(X_train))
acc_train
```

Out[42]:
```
0.8331918981785791
```

In [43]:
```
acc_test = r2_score(y_test, model.predict(X_test))
acc_test
```

Out[43]:
```
0.7520442571699866
```

Ridge model

In [44]:
```
model = Ridge()
model.fit(X_train, y_train)
```
Out[44]:
```
Ridge()
```
In [45]:
```
acc_train = r2_score(y_train, model.predict(X_train))
acc_train
```
Out[45]:
```
0.818013671064898
```
In [46]:
```
acc_test = r2_score(y_test, model.predict(X_test))
acc_test
```
Out[46]:
```
0.7699019407169656
```
Calculate the avarage metrics for differnt folds from the data

In [49]:
```
from sklearn.model_selection import KFold

# Define the number of folds (k)
k = 40

kf = KFold(n_splits=k, shuffle=True, random_state=22)
mae_scores = []
acc_train_Scores = []
acc_test_Scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = Ridge()
    model.fit(X_train, y_train)

    acc_train = r2_score(y_train, model.predict(X_train))
    acc_test = r2_score(y_test, model.predict(X_test))
    mae = mean_absolute_error(y_test, model.predict(X_test))
    print(f"Accuracy training = {acc_train}, Accuracy testing = {acc_test}, MAE = {mae}"
    acc_train_Scores.append(np.abs(acc_train))
    mae_scores.append(np.abs(mae))
    acc_test_Scores.append(np.abs(acc_test))

average_mae = sum(mae_scores) / len(mae_scores)
print(" ")
print(f"across {k} folds :-\n Average MAE: {average_mae}, Average training accuarcy: {np
```
Loading [MathJax]/extensions/Safe.js

```
Accuracy training = 0.8246830727806996, Accuracy testing = -0.6592368104050823, MAE = 10
92.1837840076987
Accuracy training = 0.8199418556134397, Accuracy testing = 0.835567992253909, MAE = 128
3.0198072286962
Accuracy training = 0.822710746580618, Accuracy testing = 0.25754905080691537, MAE = 115
4.990473047526
Accuracy training = 0.8259016020382088, Accuracy testing = 0.5154117871518231, MAE = 216
2.926062442332
Accuracy training = 0.8196549750207203, Accuracy testing = 0.7831249765189542, MAE = 155
0.3815205730484
Accuracy training = 0.817816043387178, Accuracy testing = 0.8622511415167642, MAE = 179
3.3255473565223
Accuracy training = 0.8318056431050102, Accuracy testing = 0.47682820222956834, MAE = 29
13.521237921102
Accuracy training = 0.8253339377975071, Accuracy testing = 0.3741069778030771, MAE = 168
6.5328096089565
Accuracy training = 0.8259254295561392, Accuracy testing = 0.4717522250273024, MAE = 195
3.603411961018
Accuracy training = 0.8218520902983052, Accuracy testing = 0.7608271278296563, MAE = 219
9.35512313262
Accuracy training = 0.8438036899311826, Accuracy testing = 0.4361521149560009, MAE = 415
5.205246060442
Accuracy training = 0.822310086788718, Accuracy testing = 0.7201557877513183, MAE = 163
3.574220270854
Accuracy training = 0.8218659837184015, Accuracy testing = 0.7894334180038283, MAE = 146
4.112189576545
Accuracy training = 0.8215037125870458, Accuracy testing = 0.7878963330837413, MAE = 75
2.7432730663422
Accuracy training = 0.8210512385412353, Accuracy testing = 0.3046671073725813, MAE = 109
0.3601142547893
Accuracy training = 0.8232287207523805, Accuracy testing = 0.48454564463032257, MAE = 18
60.0346243567924
Accuracy training = 0.8203774698667633, Accuracy testing = 0.43304574255600126, MAE = 91
9.3172924204002
Accuracy training = 0.821817803072378, Accuracy testing = 0.8357484880729735, MAE = 969.
296620437206
Accuracy training = 0.8220370521621599, Accuracy testing = 0.760465658343435, MAE = 127
2.8644389171368
Accuracy training = 0.8233228082102515, Accuracy testing = 0.25659994134954445, MAE = 17
85.1467741027536
Accuracy training = 0.8215961128414335, Accuracy testing = 0.7741051667137185, MAE = 120
8.7779229436292
Accuracy training = 0.8211748151384555, Accuracy testing = 0.8631454651139377, MAE = 104
8.9573575510021
Accuracy training = 0.8208771553529279, Accuracy testing = 0.903922129671255, MAE = 626.
9250056077326
Accuracy training = 0.8214256406311116, Accuracy testing = 0.5292749493309938, MAE = 258
4.257762404195
Accuracy training = 0.8253291289810638, Accuracy testing = 0.2460760453999623, MAE = 185
7.3068584202956
Accuracy training = 0.8218588995330907, Accuracy testing = 0.8359623544344815, MAE = 122
5.9875962909682
Accuracy training = 0.8197437303277968, Accuracy testing = 0.8728400545374185, MAE = 69
8.8464509094647
Accuracy training = 0.819018455968094, Accuracy testing = 0.9210768788541155, MAE = 128
1.650271502389
Accuracy training = 0.8191962247849361, Accuracy testing = 0.8650730356453604, MAE = 96
4.8434612141664
Accuracy training = 0.8240865734285314, Accuracy testing = 0.7250165803272401, MAE = 205
1.1865775523693
Accuracy training = 0.8212442325733427, Accuracy testing = 0.8332369981765564, MAE = 89
8.0269857154781
Accuracy training = 0.8197255515516135, Accuracy testing = 0.9142781408459413, MAE = 77
0.5109414347007
```

```
Accuracy training = 0.831265636455178, Accuracy testing = -6.571905742974321, MAE = 255
0.0489312341215
Accuracy training = 0.824280373229981, Accuracy testing = 0.6939590096503583, MAE = 214
3.7100688814644
Accuracy training = 0.8224687147496687, Accuracy testing = 0.32144196851175766, MAE = 11
09.0317877394864
Accuracy training = 0.8185777164514334, Accuracy testing = 0.9430427291191408, MAE = 76
5.2960996251877
Accuracy training = 0.8273139337463052, Accuracy testing = 0.5832269705955342, MAE = 249
3.913692179098
Accuracy training = 0.8206719275899699, Accuracy testing = 0.8720292560410182, MAE = 140
7.4924815053455
Accuracy training = 0.8225431529158324, Accuracy testing = 0.7984475359462904, MAE = 138
7.9188747279204
Accuracy training = 0.8199461281189927, Accuracy testing = 0.962391205914584, MAE = 299.
2284872239634

across 40 folds :-
 Average MAE: 1526.6603046351438, Average training accuarcy: 0.8229822016544525, Average
testing accuarcy: 0.8208954686366695
```

Try polynomial regression model

In [50]:
```
# reseting the values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [51]:
```
Poly_features = PolynomialFeatures(degree=2)

X_train_poly = Poly_features.fit_transform(X_train)
X_test_poly = Poly_features.transform(X_test)
```

In [52]:
```
new_model = Lasso()
new_model.fit(X_train_poly, y_train)
```

Out[52]:
```
Lasso()
```

In [53]:
```
acc_test = r2_score(y_test, new_model.predict(X_test_poly))
acc_test
```

Out[53]:
```
0.5798752136537053
```

In [54]:
```
acc_test = r2_score(y_train, new_model.predict(X_train_poly))
acc_test
```

Out[54]:
```
0.9688608837336373
```

Calculate the avarage metrics for differnt folds from the data

In [55]:
```
# Define the number of folds (k)
k = 10

kf = KFold(n_splits=k, shuffle=True, random_state=22)
mae_scores = []
acc_train_Scores = []
acc_test_Scores = []

Poly_features = PolynomialFeatures(degree=2)
```

```
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]


    X_train_poly = Poly_features.fit_transform(X_train)
    X_test_poly = Poly_features.transform(X_test)

    model = Lasso()
    model.fit(X_train_poly, y_train)

    acc_train = r2_score(y_train, model.predict(X_train_poly))
    acc_test = r2_score(y_test, model.predict(X_test_poly))
    mae = mean_absolute_error(y_test, model.predict(X_test_poly))
    print(f"Accuracy training = {acc_train}, Accuracy testing = {acc_test}, MAE = {mae}"
    acc_train_Scores.append(np.abs(acc_train))
    mae_scores.append(np.abs(mae))
    if np.abs(acc_test) <= 1:
        acc_test_Scores.append(np.abs(acc_test))

average_mae = sum(mae_scores) / len(mae_scores)
print(" ")
print(f"across {k} folds :-\n Average MAE: {average_mae}, Average training accuarcy: {np
```

```
Accuracy training = 0.9618381783326446, Accuracy testing = -0.022983089566298576, MAE =
2461.8277474267097
Accuracy training = 0.9607486055034671, Accuracy testing = 0.592072065080555, MAE = 215
6.763824480987
Accuracy training = 0.9614711433683769, Accuracy testing = 0.5801266722473952, MAE = 225
5.9475996791
Accuracy training = 0.9583534714551666, Accuracy testing = 0.5890632677551334, MAE = 145
9.7489222638603
Accuracy training = 0.956084538391031, Accuracy testing = 0.07773044205845903, MAE = 178
3.1850136927476
Accuracy training = 0.9594953572531848, Accuracy testing = 0.7667755519140387, MAE = 125
8.0179583988227
Accuracy training = 0.9569737388904127, Accuracy testing = 0.03616646130552792, MAE = 25
26.451100387102
Accuracy training = 0.9561259559934017, Accuracy testing = 0.8869794367798065, MAE = 116
9.1558080017105
Accuracy training = 0.9547643568165076, Accuracy testing = 0.8126072762860843, MAE = 137
2.4769546298432
Accuracy training = 0.9542873092398566, Accuracy testing = 0.9014210243229313, MAE = 96
9.4376001754931

across 10 folds :-
 Average MAE: 1741.3012529136377, Average training accuarcy: 0.9580142655244048, Average
testing accuarcy: 0.526592528731623
```

Try Decision Trees

In [56]:
```
# reseting the values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [57]:
```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random_state=22)
model.fit(X_train, y_train)
```

Out[57]:
```
DecisionTreeRegressor(random_state=22)
```

In [58]:
```
acc_train = r2_score(y_train, model.predict(X_train))
acc_test = r2_score(y_test, model.predict(X_test))

print(acc_train, acc_test)
```

```
0.9972253775501413 0.7378471189044432
```

Final Result - Conclusion

```
The Best Choice : Linear Regression model using L2 Regulization (Ridge
Model)
Using K-fold technique and using Ridge model applying it, we concluded
that the average results for 40 folds is :
Tarining Accuarcy : 82.0%
Testing Accuracy : 82.0%
Mean Absolute Error : 1526.0
```

Note : Lasso and Ridge models are linear regression models but with extra abilities for regulization which is a technique to avoid overfitting, and usually called L1 for Lasso and L2 for Ridge