INSTRUCTIONS

Every learner should submit his/her own homework solutions. However, you are allowed to discuss the homework with each other– but everyone must submit his/her own solution; you may not copy someone else's solution.

The homework consists of two parts:

1. Data from our lives
2. Data manipulation/Exploratory Data Analysis
3. Multipe regression Analysis

Follow the prompts in the attached jupyter notebook. Download the data and place it in your working directory, or modify the path to upload it to your notebook. Add markdown cells to your analysis to include your solutions, comments, answers. **Add as many cells as you need**, for easy readability comment when possible. Hopefully this homework will help you develop skills, make you understand the flow of an EDA, get you ready for individual work.

Submission: Send in both a ipynb and a pdf file of your work.

Good luck!

1. Data from our Lives

Describe a situation or problem from your job, everyday life, current events, etc., for which a regression model would be appropriate. List some (up to 5) predictors that you might use.

Canadian public transit receives complaints about something or the other daily. However, they receive excessive complaints about 10 or 5 days consecutively; it is a cycle, and authorities don't know why there is a pattern of receiving excessive complaints every few days. Later, they found a relationship between the pattern of receiving excessive complaints with extreme whether(**unexpected rain, rise in temperature fall in temperature, too much snow, and windy day**) as people not prepared for the extreme weather. Therefore, the weather reporting agency will increase its predicting efficacy during intense weather; somehow, the public transport will receive fewer complaints on extreme weather days as people are prepared.

The data

Title: 1985 Auto Imports Database

Relevant Information: -- Description This data set consists of three types of entities: (a) the specification of an auto in terms of various characteristics, (b) its assigned insurance risk rating, (c) its normalized losses in use as compared to other cars. The second rating corresponds to the degree to which the auto is more risky than its price indicates. Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

```
  The third factor is the relative average loss payment per
insured
  vehicle year.  This value is normalized for all autos within a
  particular size classification (two-door small, station
wagons,
  sports/speciality, etc...), and represents the average loss
per car
  per year.
```

-- Note: Several of the attributes in the database could be used as a "class" attribute.

1. Number of Instances: 205

2. Number of Attributes: 26 total -- 15 continuous -- 1 integer -- 10 nominal

3. Attribute Information:
   Attribute: Attribute Range:

   A. symboling: -3, -2, -1, 0, 1, 2, 3.
   B. normalized-losses: continuous from 65 to 256.
   C. make: alfa-romero, audi, bmw, chevrolet, dodge, honda,isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota,volkswagen, volvo
   D. fuel-type: diesel, gas.
   E. aspiration: std, turbo.
   F. num-of-doors: four, two.
   G. body-style: hardtop, wagon, sedan, hatchback, convertible.
   H. drive-wheels: 4wd, fwd, rwd.
   I. engine-location: front, rear.
   J. wheel-base: continuous from 86.6 120.9.
   K. length: continuous from 141.1 to 208.1.
   L. width: continuous from 60.3 to 72.3.
   M. height: continuous from 47.8 to 59.8.
   N. curb-weight: continuous from 1488 to 4066.
   O. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
   P. num-of-cylinders: eight, five, four, six, three, twelve, two.

Q. engine-size: continuous from 61 to 326.
R. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
S. bore: continuous from 2.54 to 3.94.
T. stroke: continuous from 2.07 to 4.17.
U. compression-ratio: continuous from 7 to 23.
V. horsepower: continuous from 48 to 288.
W. peak-rpm: continuous from 4150 to 6600.
X. city-mpg: continuous from 13 to 49.
Y. highway-mpg: continuous from 16 to 54.
Z. price: continuous from 5118 to 45400.

4. Missing Attribute Values: (denoted by "?")

In [1]:

```
from scipy import stats
from sklearn.linear_model import LinearRegression
from statsmodels.compat import lzip
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm

%matplotlib inline
```

In [2]:

```
#Read in data
df =pd.read_csv('auto_imports1.csv') #imports the dataset

df.head() #snapshot of the dataset
```

Out[2]:

	fuel_type	body	wheel_base	length	width	heights	curb_weight	engine_type	cylin
0	gas	convertible	88.6	168.8	64.1	48.8	2548	dohc	
1	gas	convertible	88.6	168.8	64.1	48.8	2548	dohc	
2	gas	hatchback	94.5	171.2	65.5	52.4	2823	ohcv	
3	gas	sedan	99.8	176.6	66.2	54.3	2337	ohc	
4	gas	sedan	99.4	176.6	66.4	54.3	2824	ohc	

2. Data

2.1 Munging

Check what types of variables do you have in your data? Do you see anything that doesn't make sense? *Hint: horse power is an object ?!*

In [3]:
```
##your code here
df.info() #summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   fuel_type    201 non-null    object 
 1   body         201 non-null    object 
 2   wheel_base   201 non-null    float64
 3   length       201 non-null    float64
 4   width        201 non-null    float64
 5   heights      201 non-null    float64
 6   curb_weight  201 non-null    int64  
 7   engine_type  201 non-null    object 
 8   cylinders    201 non-null    object 
 9   engine_size  201 non-null    int64  
 10  bore         201 non-null    object 
 11  stroke       201 non-null    object 
 12  compression  201 non-null    float64
 13  horse_power  201 non-null    object 
 14  peak_rpm     201 non-null    object 
 15  city_mpg     201 non-null    int64  
 16  highway_mpg  201 non-null    int64  
 17  price        201 non-null    int64  
dtypes: float64(5), int64(5), object(8)
memory usage: 28.4+ KB
```

There are 8 object variables, 5 float64, 5 int64. However, there are some varibles dosen't make sense like bore, stroke, horse power, peak rpm.

Replace '?' with None

Change the variables: bore, stroke, horse_power, peak_rpm to float64

In [4]:
```
df['bore'].unique() #checks " ? " before converting variable from object to f
```

Out[4]:
```
array(['3.47', '2.68', '3.19', '3.13', '3.5', '3.31', '3.62', '2.91',
       '3.03', '2.97', '3.34', '3.6', '2.92', '3.15', '3.43', '3.63',
       '3.54', '3.08', '?', '3.39', '3.76', '3.58', '3.46', '3.8', '3.78',
       '3.17', '3.35', '3.59', '2.99', '3.33', '3.7', '3.61', '3.94',
       '3.74', '2.54', '3.05', '3.27', '3.24', '3.01'], dtype=object)
```

In [5]:
```
df['stroke'].unique() #checks " ? " before converting variable from object to
```

Out[5]:
```
array(['2.68', '3.47', '3.4', '2.8', '3.19', '3.39', '3.03', '3.11',
       '3.23', '3.46', '3.9', '3.41', '3.07', '3.58', '4.17', '2.76',
       '3.15', '?', '3.16', '3.64', '3.1', '3.35', '3.12', '3.86', '3.29',
       '3.27', '3.52', '2.19', '3.21', '2.9', '2.07', '2.36', '2.64',
       '3.08', '3.5', '3.54', '2.87'], dtype=object)
```

In [6]:
```
df['horse_power'].unique() #checks " ? " before converting variable from obje
```

Out[6]:
```
array(['111', '154', '102', '115', '110', '140', '101', '121', '182',
       '48', '70', '68', '88', '145', '58', '76', '60', '86', '100', '78',
       '90', '176', '262', '135', '84', '64', '120', '72', '123', '155',
       '184', '175', '116', '69', '55', '97', '152', '160', '200', '95',
       '142', '143', '207', '?', '73', '82', '94', '62', '56', '112',
       '92', '161', '156', '52', '85', '114', '162', '134', '106'],
      dtype=object)
```

In [7]:
```
df['peak_rpm'].unique() #checks " ? " before converting variable from object
```

Out[7]:
```
array(['5000', '5500', '5800', '4250', '5400', '5100', '4800', '6000',
       '4750', '4650', '4200', '4350', '4500', '5200', '4150', '5600',
       '5900', '?', '5250', '4900', '4400', '6600', '5300'], dtype=object)
```

In [8]:
```
df = df.replace(['?'],'NaN') #replaces '?' with 'NaN'
```

In [9]:
```
## Your code here
float_values = {'bore':float,'stroke':float,'horse_power':float,'peak_rpm':fl
```

In [10]:
```
df = df.astype(float_values) #typecasting to float64
```

In [11]:
```
df.info() #summary of dataset after converting some variables from object to
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   fuel_type    201 non-null    object 
 1   body         201 non-null    object 
 2   wheel_base   201 non-null    float64
 3   length       201 non-null    float64
 4   width        201 non-null    float64
 5   heights      201 non-null    float64
 6   curb_weight  201 non-null    int64  
 7   engine_type  201 non-null    object 
 8   cylinders    201 non-null    object 
 9   engine_size  201 non-null    int64  
 10  bore         197 non-null    float64
 11  stroke       197 non-null    float64
 12  compression  201 non-null    float64
 13  horse_power  199 non-null    float64
 14  peak_rpm     199 non-null    float64
 15  city_mpg     201 non-null    int64  
 16  highway_mpg  201 non-null    int64  
 17  price        201 non-null    int64  
dtypes: float64(9), int64(5), object(4)
memory usage: 28.4+ KB
```

In [12]:

```
df.shape #shape of the data set
```

Out[12]:

```
(201, 18)
```

Drop body,engine_type,cylinders columns and name the new dataframe df2

In [13]:

```
## Your code here
df2 = df #creates new dataset
```

In [14]:

```
df2.head() #snapshot of new dataset df2
```

Out[14]:

	fuel_type	body	wheel_base	length	width	heights	curb_weight	engine_type	cylin
0	gas	convertible	88.6	168.8	64.1	48.8	2548	dohc	
1	gas	convertible	88.6	168.8	64.1	48.8	2548	dohc	
2	gas	hatchback	94.5	171.2	65.5	52.4	2823	ohcv	
3	gas	sedan	99.8	176.6	66.2	54.3	2337	ohc	
4	gas	sedan	99.4	176.6	66.4	54.3	2824	ohc	

In [15]:

```
df2 = df2.drop(['body','engine_type','cylinders'], axis=1) #drops 3 variables
```

In [16]:
```
df2.head() #snapshot of df2 after dropping the vaiables
```

Out[16]:

	fuel_type	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	con
0	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	
1	gas	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	
2	gas	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	
3	gas	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	
4	gas	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	

Drop all nan values

In [17]:
```
df2.isnull().sum() #checks null values
```

Out[17]:
```
fuel_type       0
wheel_base      0
length          0
width           0
heights         0
curb_weight     0
engine_size     0
bore            4
stroke          4
comprassion     0
horse_power     2
peak_rpm        2
city_mpg        0
highway_mpg     0
price           0
dtype: int64
```

In [18]:
```
df2 = df2.dropna() #drops null values
```

Get dummy variables for fuel_type within df2 drop first level

In [19]:
```
df2 = pd.get_dummies(df2, columns = ['fuel_type'], drop_first = True) #create
```

In [20]:
```
df2.info() #summary of dataset after dummy variables
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195 entries, 0 to 200
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   wheel_base      195 non-null    float64
 1   length          195 non-null    float64
 2   width           195 non-null    float64
 3   heights         195 non-null    float64
 4   curb_weight     195 non-null    int64
 5   engine_size     195 non-null    int64
 6   bore            195 non-null    float64
 7   stroke          195 non-null    float64
 8   comprassion     195 non-null    float64
 9   horse_power     195 non-null    float64
 10  peak_rpm        195 non-null    float64
 11  city_mpg        195 non-null    int64
 12  highway_mpg     195 non-null    int64
 13  price           195 non-null    int64
 14  fuel_type_gas   195 non-null    uint8
dtypes: float64(9), int64(5), uint8(1)
memory usage: 23.0 KB
```

2.2 EDA on df2

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Follow the lecture notes for ideas of how to perform EDA on your dataset. For help, here are the steps we talked about:

Suggested Steps in EDA:

```
Provide descriptions of your sample and features
Check for missing data
Identify the shape of your data
Identify significant correlations
Spot/deal with outliers in the dataset
```

These steps are a guidline. Try different things and share your insights about the dataset (**df2**).

Don't forget to add "markdown" cells to include your findings or to explain what you are doing

In [21]:
```
## Your EDA should start here
df2.shape #old shape
```

Out[21]: (195, 15)

In [22]:

```
df2.info() #summary
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195 entries, 0 to 200
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   wheel_base      195 non-null    float64
 1   length          195 non-null    float64
 2   width           195 non-null    float64
 3   heights         195 non-null    float64
 4   curb_weight     195 non-null    int64  
 5   engine_size     195 non-null    int64  
 6   bore            195 non-null    float64
 7   stroke          195 non-null    float64
 8   comprassion     195 non-null    float64
 9   horse_power     195 non-null    float64
 10  peak_rpm        195 non-null    float64
 11  city_mpg        195 non-null    int64  
 12  highway_mpg     195 non-null    int64  
 13  price           195 non-null    int64  
 14  fuel_type_gas   195 non-null    uint8  
dtypes: float64(9), int64(5), uint8(1)
memory usage: 23.0 KB
```

In [23]:

```
df2 = df2.drop_duplicates() #drop duplicate values
```

In [24]:

```
df2.shape # new shape after deleting duplicate values
```

Out[24]: (192, 15)

Now Let's see the descriptive statistics at glance.

In [25]:

```
df2.describe().T #summary statistics
```

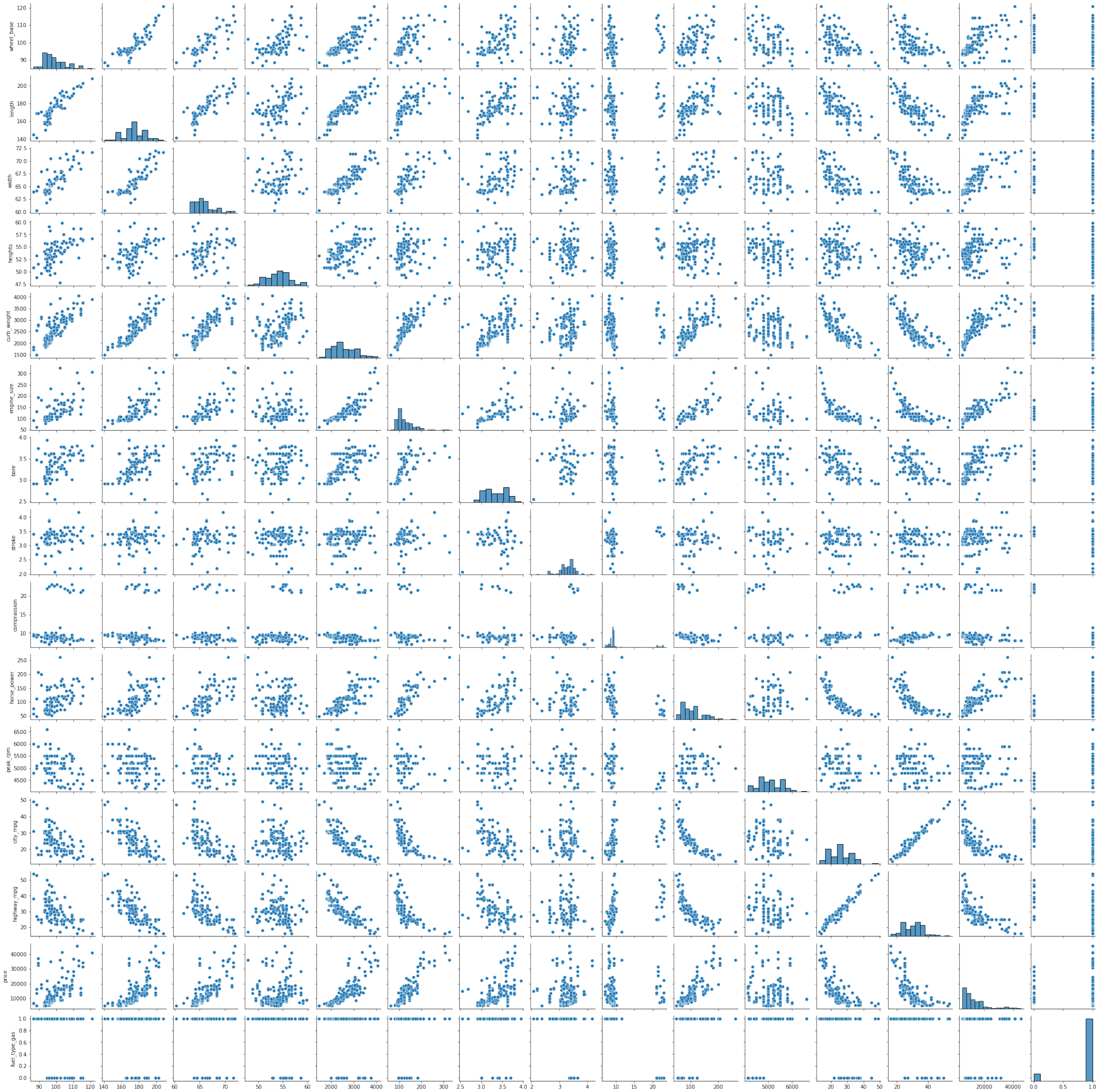
Out[25]:

	count	mean	std	min	25%	50%	75%	
wheel_base	192.0	98.964062	6.153849	86.60	94.5000	97.20	102.40	
length	192.0	174.443229	12.451618	141.10	166.6750	173.20	184.60	
width	192.0	65.910417	2.138110	60.30	64.1000	65.50	66.90	
heights	192.0	53.906250	2.387722	47.80	52.0000	54.10	55.70	
curb_weight	192.0	2565.140625	526.002275	1488.00	2163.0000	2422.50	2952.50	4
engine_size	192.0	128.385417	41.588704	61.00	98.0000	120.00	146.00	
bore	192.0	3.333646	0.271616	2.54	3.1500	3.31	3.59	
stroke	192.0	3.248594	0.316038	2.07	3.1075	3.29	3.41	
comprassion	192.0	10.226667	4.084397	7.00	8.5750	9.00	9.40	
horse_power	192.0	103.395833	38.069096	48.00	70.0000	95.00	116.00	
peak_rpm	192.0	5093.229167	469.215665	4150.00	4800.0000	5050.00	5500.00	6
city_mpg	192.0	25.364583	6.435536	13.00	19.0000	25.00	30.00	
highway_mpg	192.0	30.812500	6.862620	16.00	25.0000	30.00	34.50	
price	192.0	13332.802083	8088.861657	5118.00	7765.7500	10320.00	16525.75	45
fuel_type_gas	192.0	0.895833	0.306275	0.00	1.0000	1.00	1.00	

In [26]:

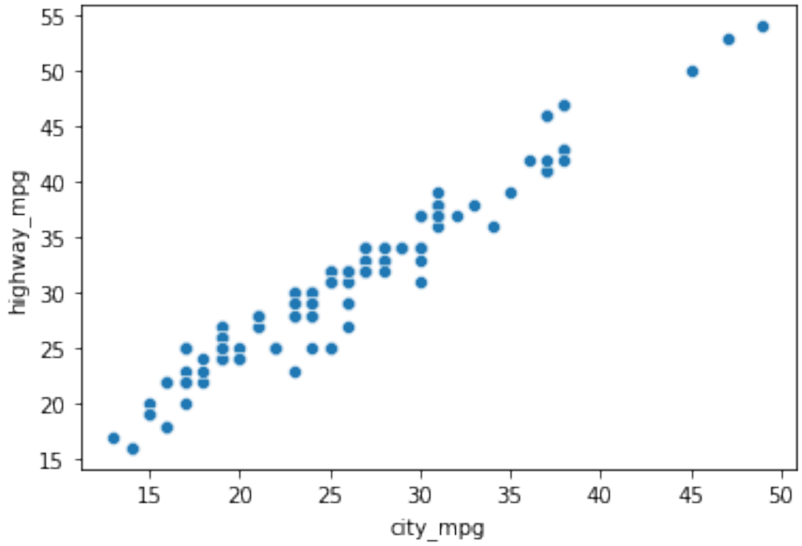
```
sns.pairplot(data=df2, height=2)
```

Out[26]: <seaborn.axisgrid.PairGrid at 0x7f7cc0f3b490>

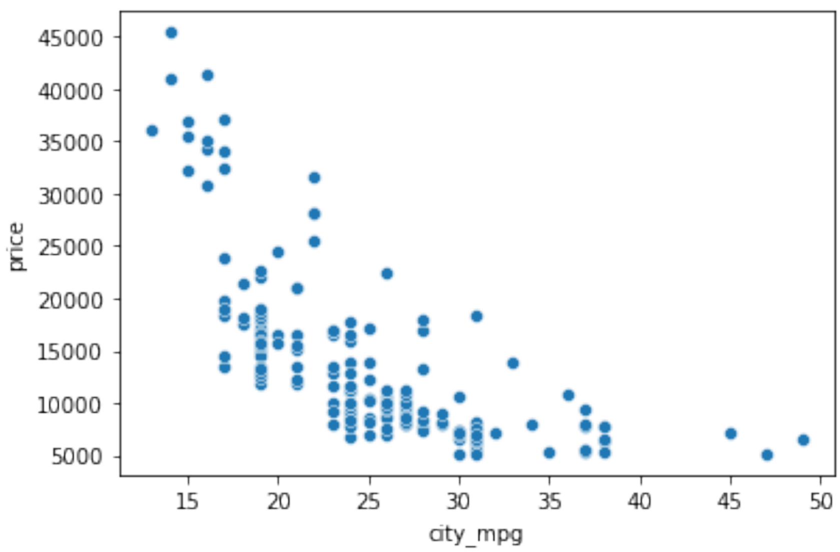


In [27]:

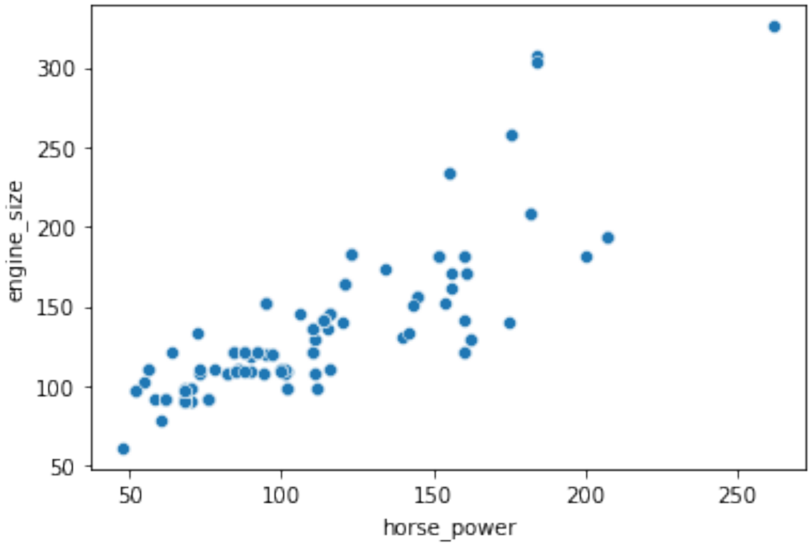
```
#Relationship between city_mpg and highway_mpg
sns.scatterplot(x='city_mpg', y='highway_mpg', data=df2)
plt.show()
```



In [28]:
```
#Relationship between city_mpg and price
sns.scatterplot(x='city_mpg', y='price', data=df2)
plt.show()
```



In [29]:
```
#Relationship between horse power and engine size
sns.scatterplot(x='horse_power', y='engine_size', data=df2)
plt.show()
```



By comparing all the variable we get some insight into the data set.

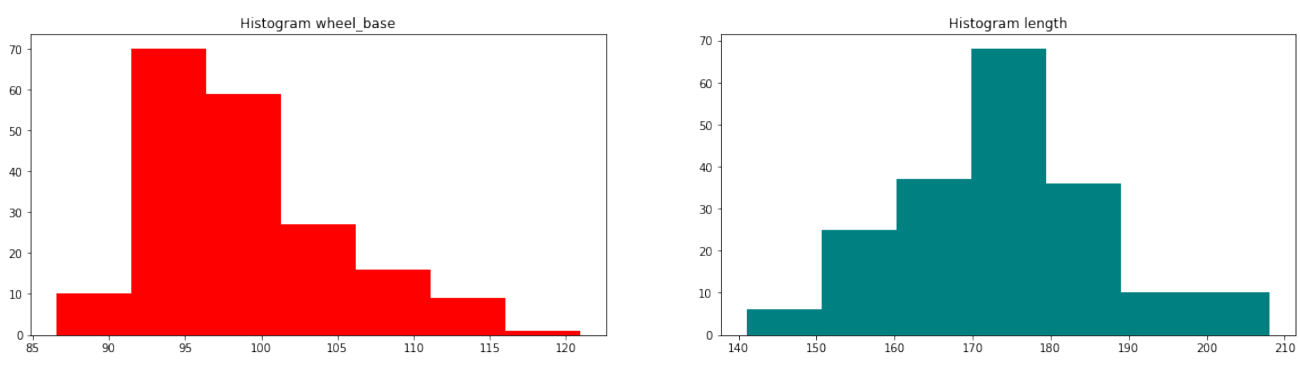
1. There are outliers.
2. I see some multicolinearity.
3. There are some negative relationships

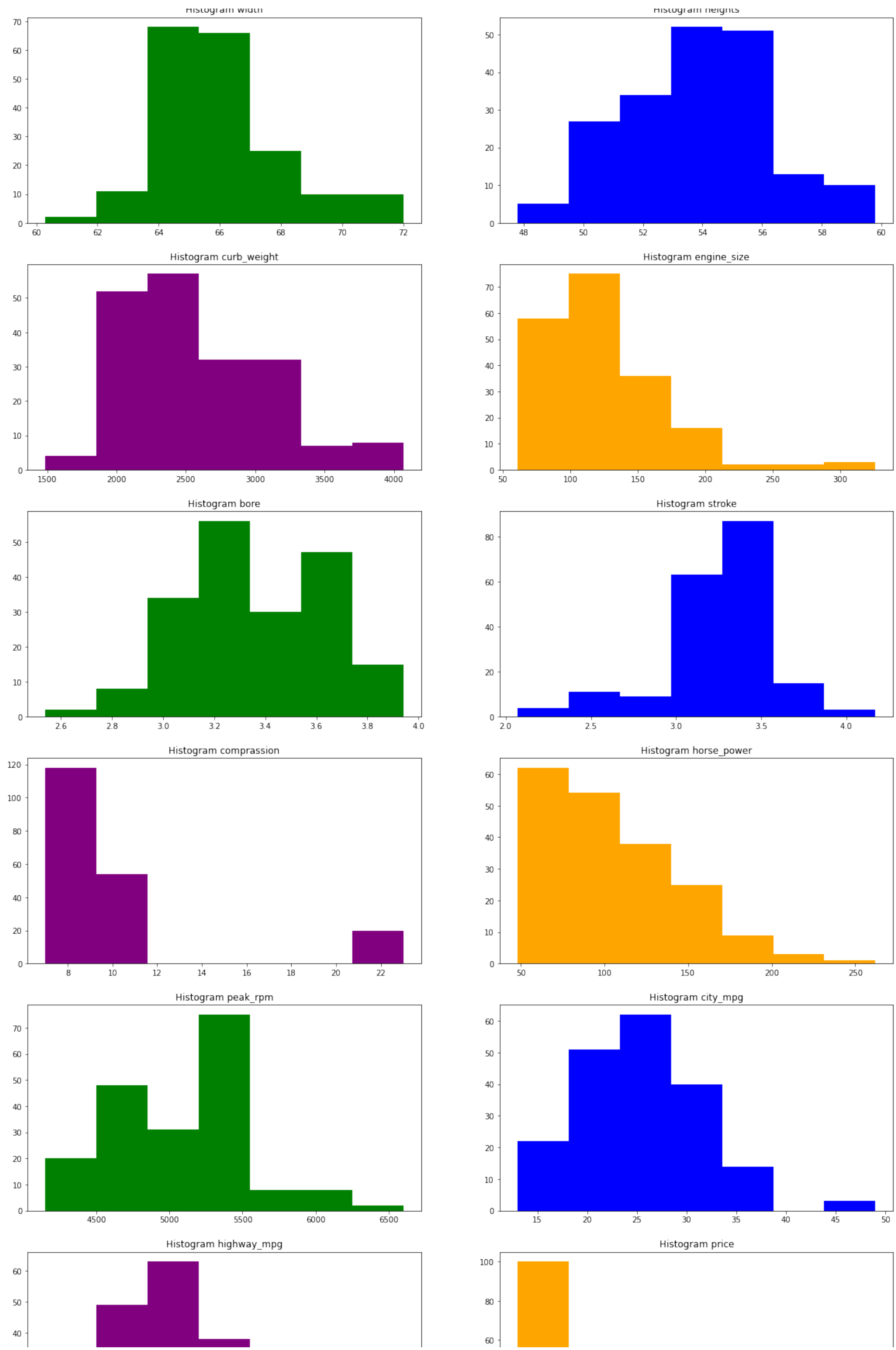
Histograms

Let's get a seak peek of df2 dataset by histograms

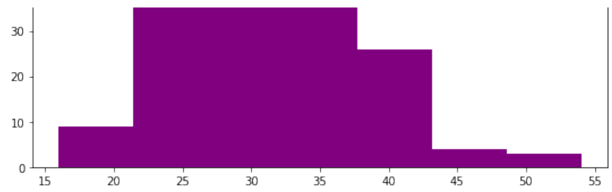
In [30]:

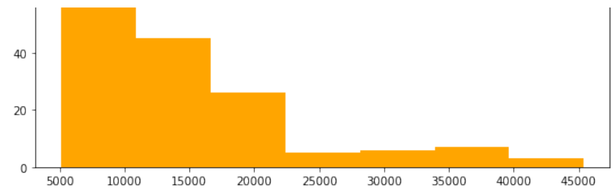
```
#histograms
fig, axes = plt.subplots(7, 2, figsize=(20,40))

axes[0,0].set_title("Histogram wheel_base")
axes[0,0].hist(df2.wheel_base, bins=7, color='red')

axes[0,1].set_title("Histogram length")
axes[0,1].hist(df2['length'], bins=7,color='teal');

axes[1,0].set_title("Histogram width")
axes[1,0].hist(df2['width'], bins=7, color='green');

axes[1,1].set_title("Histogram heights")
axes[1,1].hist(df2['heights'], bins=7, color='blue');

axes[2,0].set_title("Histogram curb_weight")
axes[2,0].hist(df2['curb_weight'], bins=7,color='purple');

axes[2,1].set_title("Histogram engine_size")
axes[2,1].hist(df2['engine_size'], bins=7, color='orange');

axes[3,0].set_title("Histogram bore")
axes[3,0].hist(df2['bore'], bins=7, color='green');

axes[3,1].set_title("Histogram stroke")
axes[3,1].hist(df2['stroke'], bins=7, color='blue');

axes[4,0].set_title("Histogram compression")
axes[4,0].hist(df2['compression'], bins=7,color='purple');

axes[4,1].set_title("Histogram horse_power")
axes[4,1].hist(df2['horse_power'], bins=7, color='orange');

axes[5,0].set_title("Histogram peak_rpm")
axes[5,0].hist(df2['peak_rpm'], bins=7, color='green');

axes[5,1].set_title("Histogram city_mpg")
axes[5,1].hist(df2['city_mpg'], bins=7, color='blue');

axes[6,0].set_title("Histogram highway_mpg")
axes[6,0].hist(df2['highway_mpg'], bins=7,color='purple');

axes[6,1].set_title("Histogram price")
axes[6,1].hist(df2['price'], bins=7, color='orange');
```





Histogram width
0 10 20 30 40 50 60 70
60 62 64 66 68 70 72
Histogram heights
0 10 20 30 40 50
48 50 52 54 56 58 60
Histogram curb_weight
0 10 20 30 40 50
1500 2000 2500 3000 3500 4000
Histogram engine_size
0 10 20 30 40 50 60 70
50 100 150 200 250 300
Histogram bore
0 10 20 30 40 50
2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0
Histogram stroke
0 20 40 60 80
2.0 2.5 3.0 3.5 4.0
Histogram comprassion
0 20 40 60 80 100 120
8 10 12 14 16 18 20 22
Histogram horse_power
0 10 20 30 40 50 60
50 100 150 200 250
Histogram peak_rpm
0 10 20 30 40 50 60 70
4500 5000 5500 6000 6500
Histogram city_mpg
0 10 20 30 40 50 60
15 20 25 30 35 40 45 50
Histogram highway_mpg
40 50 60
Histogram price
60 80 100





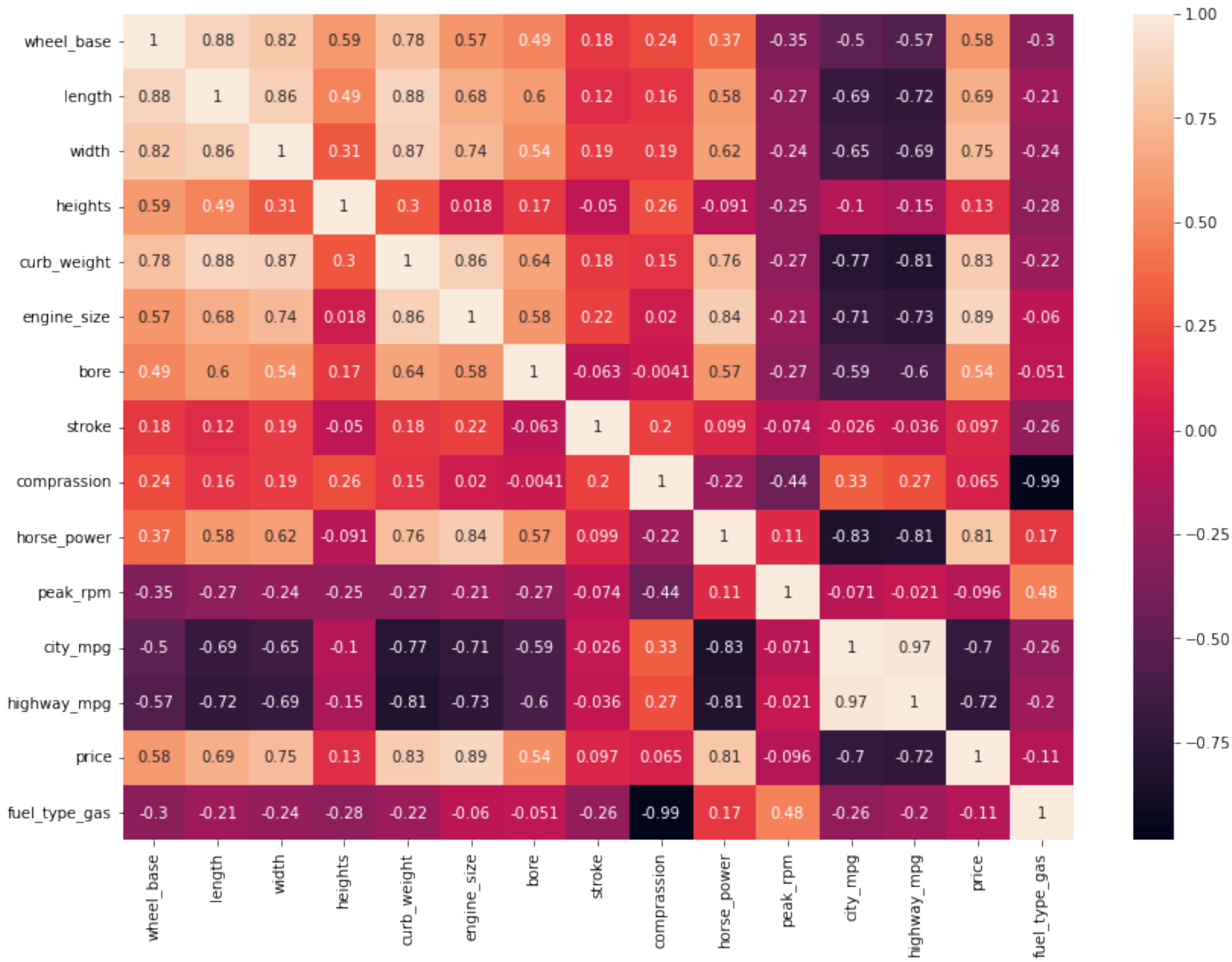
By seeing histograms

1. Some variaviable are right skewed.
2. Some have binomial distribution.
3. Some have normal distribution.

Let's see pearson correlation.

In [31]:

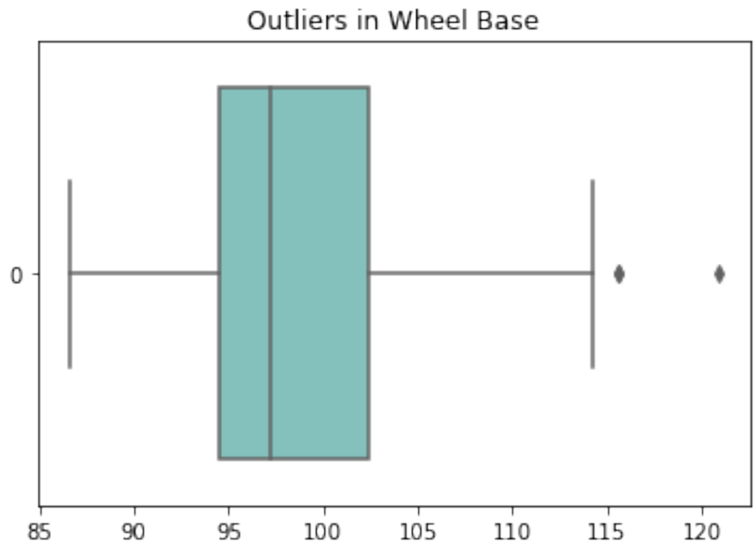
```
fig, ax = plt.subplots(figsize=(14,10))
sns.heatmap(df2.corr(method ='pearson'), annot=True)
plt.show()
```



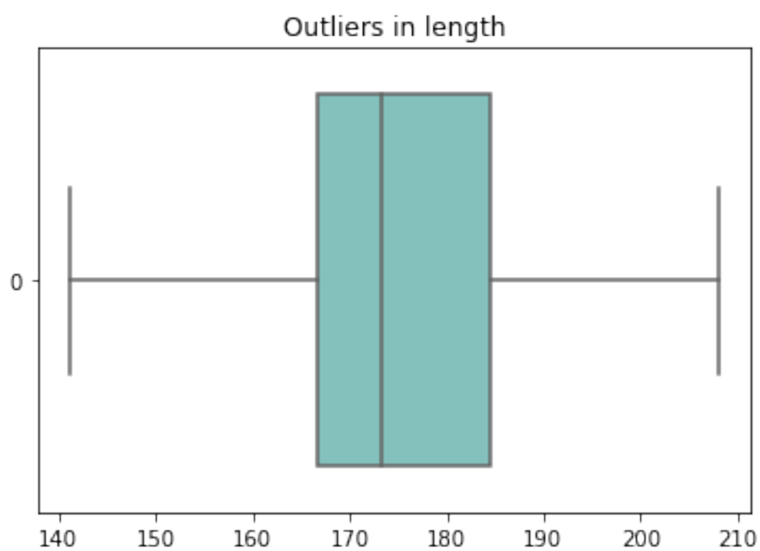
1. **The thumb rule is if coffient of correlation is more 75% that is strong cocorrelation.**
2. **price has sgnificant correlation with other variables.**

Spot/deal with outliers in the dataset.

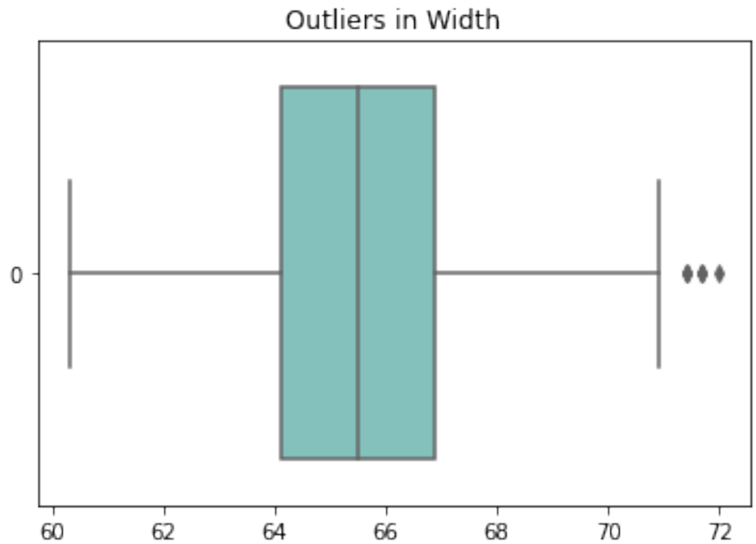
In [32]:
```
sns.boxplot(data=df2.wheel_base, orient='h', palette="GnBu")
plt.title('Outliers in Wheel Base')
plt.show()
```



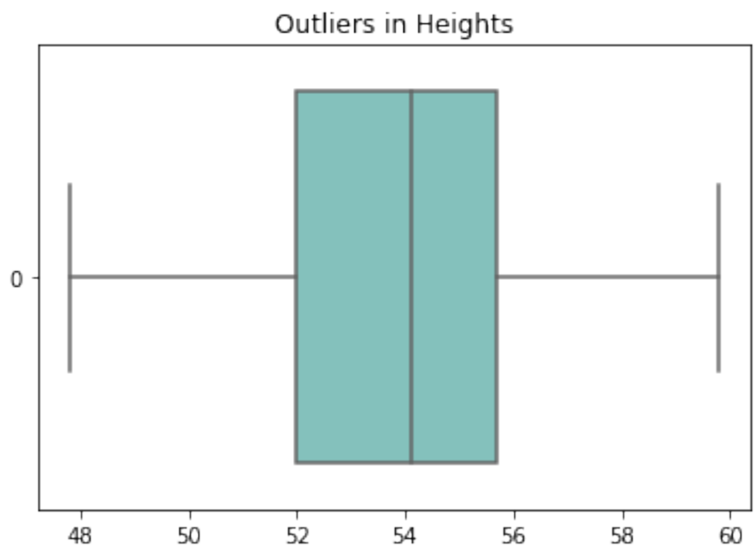
In [33]:
```
sns.boxplot(data=df2.length, orient='h', palette="GnBu")
plt.title('Outliers in length')
plt.show()
```



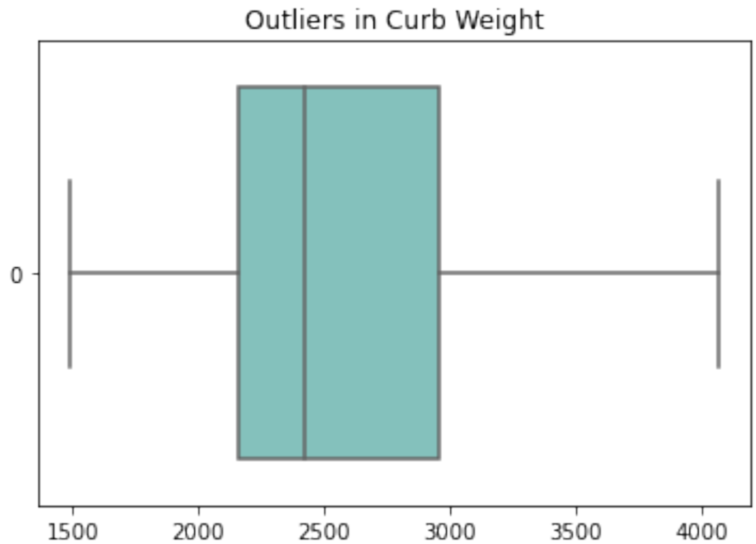
In [34]:
```
sns.boxplot(data=df2.width, orient='h', palette="GnBu")
plt.title('Outliers in Width')
plt.show()
```



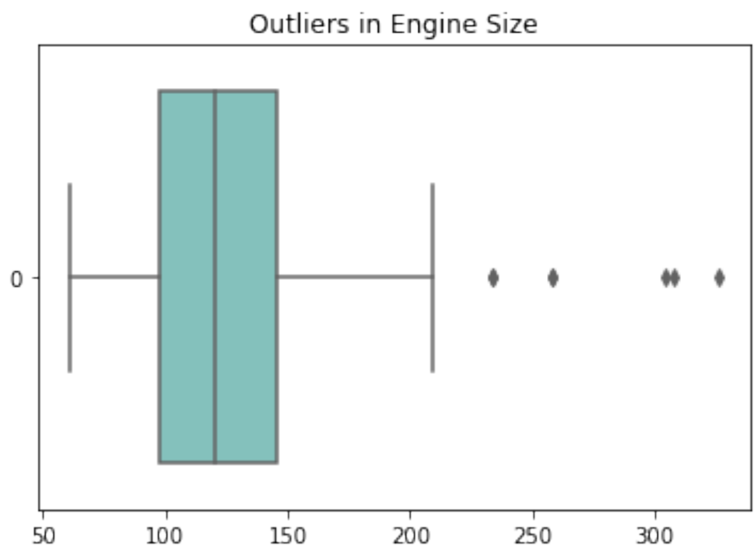
In [35]:
```
sns.boxplot(data=df2.heights, orient='h', palette="GnBu")
plt.title('Outliers in Heights')
plt.show()
```



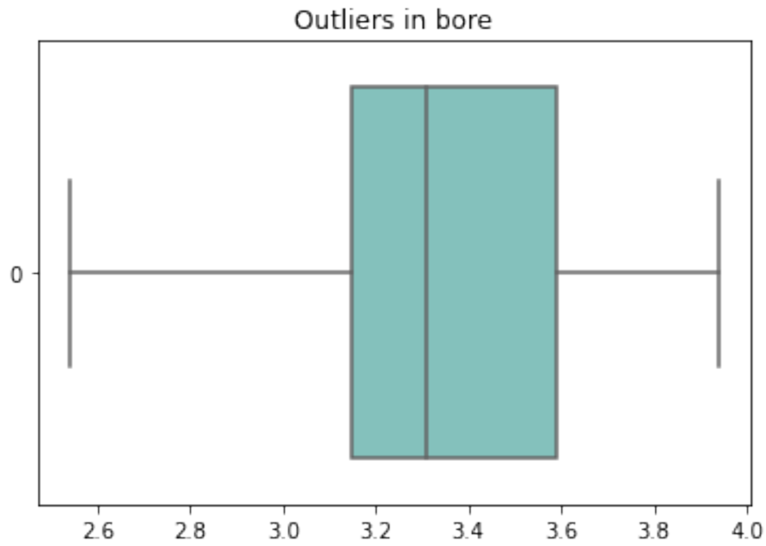
In [36]:
```
sns.boxplot(data=df2.curb_weight, orient='h', palette="GnBu")
plt.title('Outliers in Curb Weight')
plt.show()
```



In [37]:
```
sns.boxplot(data=df2.engine_size, orient='h', palette="GnBu")
plt.title('Outliers in Engine Size')
plt.show()
```

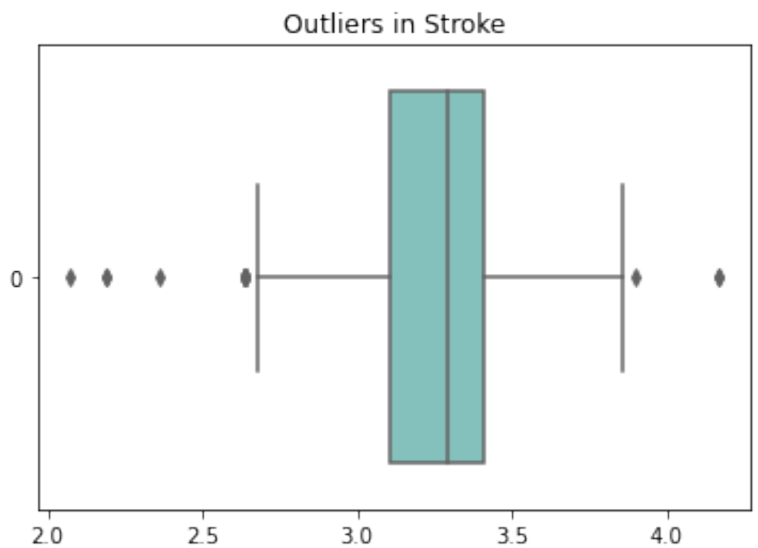


In [38]:
```
sns.boxplot(data=df2.bore, orient='h', palette="GnBu")
plt.title('Outliers in bore')
plt.show()
```



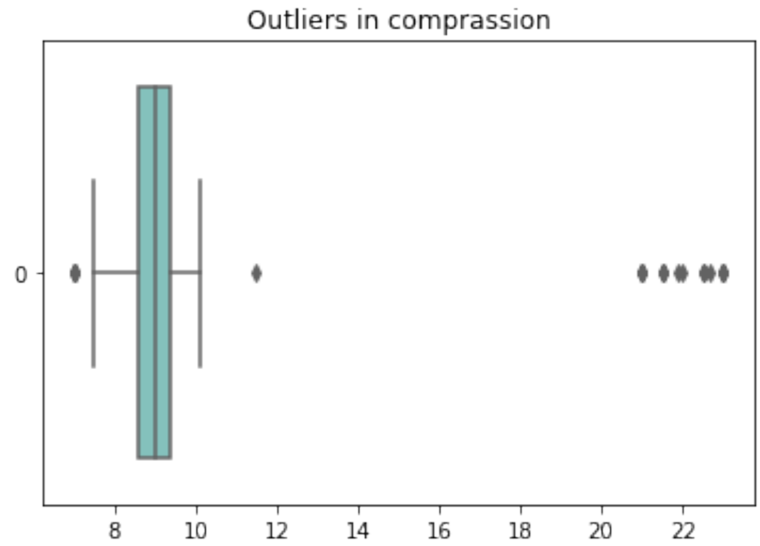
In [39]:

```
sns.boxplot(data=df2.stroke, orient='h', palette="GnBu")
plt.title('Outliers in Stroke')
plt.show()
```



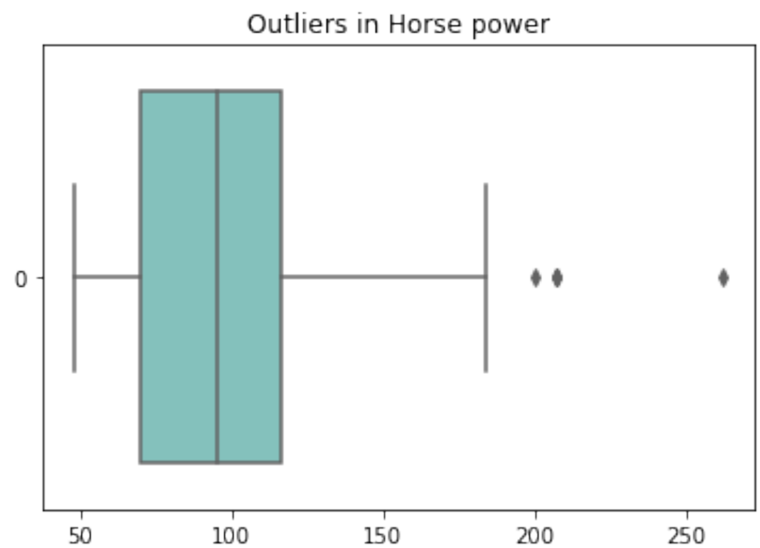
In [40]:

```
sns.boxplot(data=df2.compression, orient='h', palette="GnBu")
plt.title('Outliers in compression')
plt.show()
```



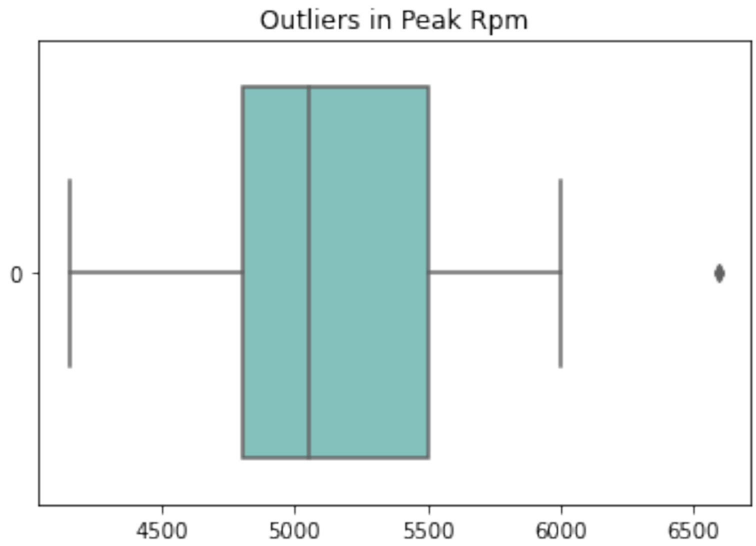
In [41]:

```
sns.boxplot(data=df2.horse_power, orient='h', palette="GnBu")
plt.title('Outliers in Horse power')
plt.show()
```



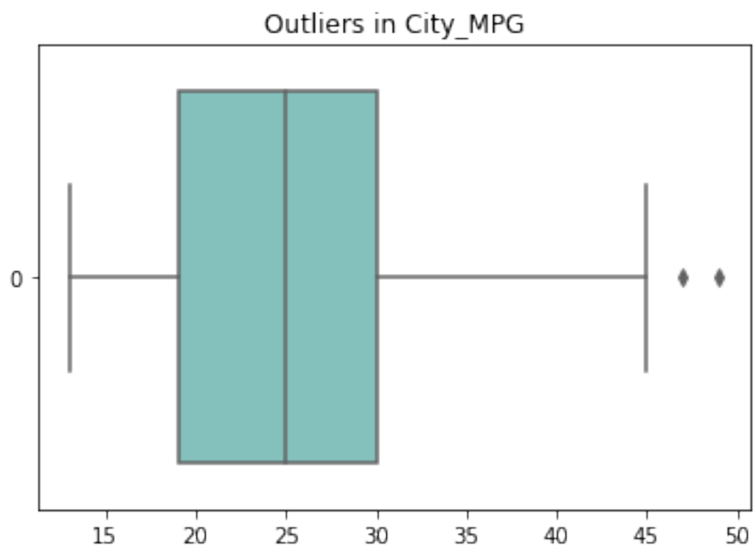
In [42]:

```
sns.boxplot(data=df2.peak_rpm, orient='h', palette="GnBu")
plt.title('Outliers in Peak Rpm')
plt.show()
```



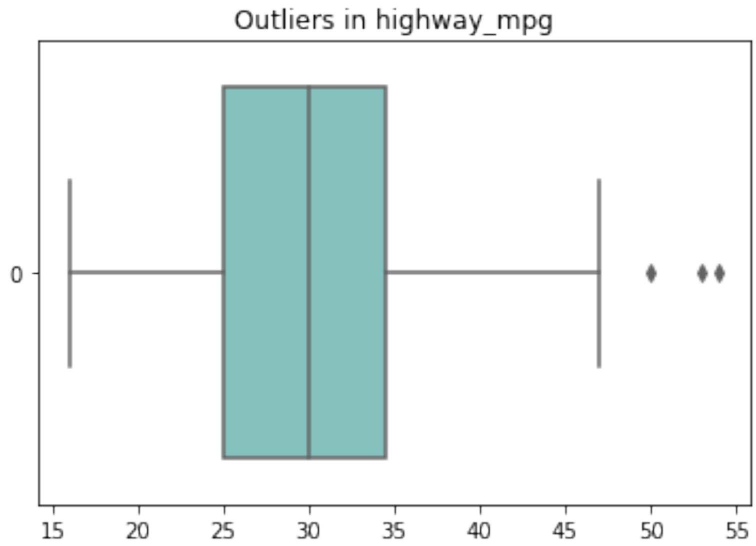
In [43]:

```
sns.boxplot(data=df2.city_mpg, orient='h', palette="GnBu")
plt.title('Outliers in City_MPG')
plt.show()
```



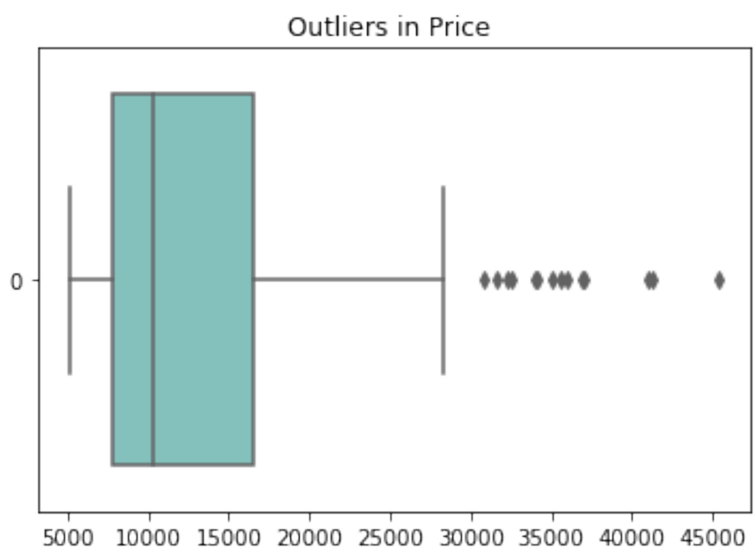
In [44]:

```
sns.boxplot(data=df2.highway_mpg, orient='h', palette="GnBu")
plt.title('Outliers in highway_mpg')
plt.show()
```



In [45]:

```
sns.boxplot(data=df2.price, orient='h', palette="GnBu")
plt.title('Outliers in Price ')
plt.show()
```



As there are more outliers in price. Now, let's deal with outliers in those variable

In [46]:

```
def outliers(df,X):
    Q1 = df[X].quantile(0.25) #25th percentile
    Q3 = df[X].quantile(0.75) #75th percentile
    IQR = Q3-Q1 #IQR

    old_shape = df.shape
    # print('The old shape is {}'.format(old_shape))

    lower_limit =  (Q1 - 1.5 * IQR) #lowernond
    upper_limit =  (Q3 + 1.5 * IQR) #upperbond

    find = df[(df[X]<lower_limit)|(df[X]>upper_limit)] #detects the outliers

    df.drop(find.index, inplace=True) #drops outliers by index

    new_shape = df.shape
    #print('The new shape is {}'.format(new_shape))

    sns.boxplot(x=X, data=df) #returns box without outliers
    plt.title('Boxplot without outliers')
```

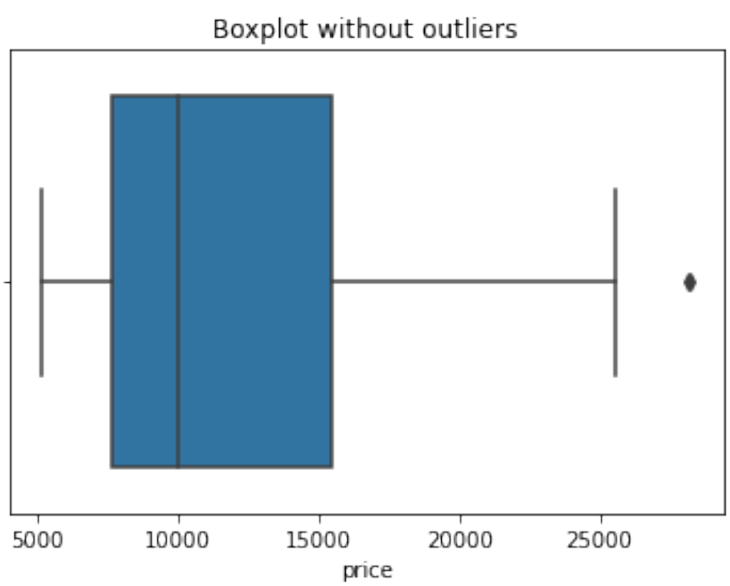
In [47]:

```
old_shape = df2.shape
print('The old shape is {}'.format(old_shape))
```

```
The old shape is (192, 15)
```

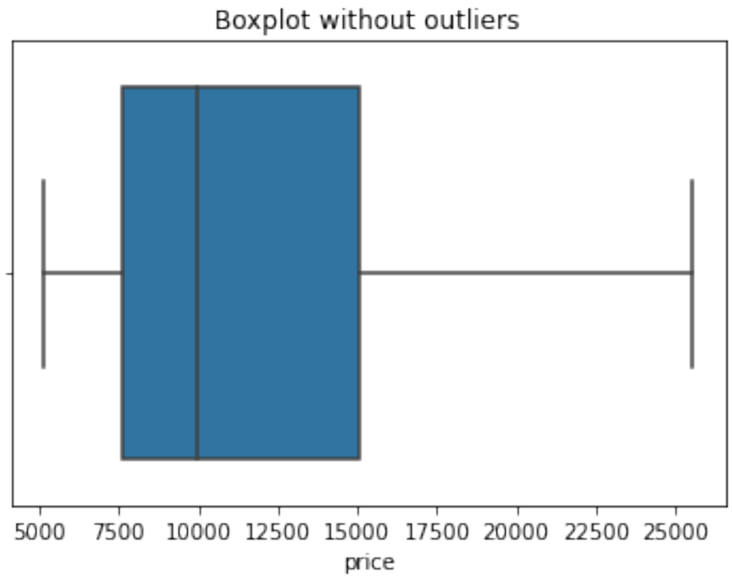
In [48]:

```
outliers(df2,'price')
```



In [49]:

```
outliers(df2,'price')
```



In [50]:
```
new_shape = df2.shape
print('The new shape is {}'.format(new_shape))
```

```
The new shape is (176, 15)
```

3. Multiple Regression Analysis ! Use the df2 dataset!

1. Create a model that uses all the variables and call it model1. The dependent variable is price, the independent variables are all the rest. Print out a summary of the model (coefficents, stanrard errors, confidence intervals and other metrics shown in class and answer the quesions based on your output.

I assume model1 as ***Ho*** .

In [51]:
```
traget = 'price' #dependent varible
```

In [52]:
```
x = df2.loc[:,df2.columns != traget] #Dataset without dependent variable
```

In [53]:
```
y = df2.loc[:,traget] #Dataset without independent variables
```

In [54]:
```
X = sm.add_constant(x) #adds constant
```

```
/Users/nithinreddynagapur/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/tsatools.py:142: FutureWarning: In a future version of pandas all argume
nts of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

In [55]:
```
model1 = sm.OLS(y,X) #OLS
```

In [56]:

```
results = model1.fit() #fitting OLS
```

In [57]:

```
results.summary() #gives summary of MLR
```

Out[57]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.802
Model:	OLS	**Adj. R-squared:**	0.784
Method:	Least Squares	**F-statistic:**	46.45
Date:	Wed, 23 Mar 2022	**Prob (F-statistic):**	3.81e-49
Time:	14:29:11	**Log-Likelihood:**	-1597.5
No. Observations:	176	**AIC:**	3225.
Df Residuals:	161	**BIC:**	3273.
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-4.43e+04	1.35e+04	-3.287	0.001	-7.09e+04	-1.77e+04
wheel_base	166.2156	79.980	2.078	0.039	8.270	324.161
length	-68.7157	43.153	-1.592	0.113	-153.935	16.503
width	546.0922	196.953	2.773	0.006	157.148	935.037
heights	-26.7776	109.114	-0.245	0.806	-242.256	188.701
curb_weight	3.5202	1.427	2.467	0.015	0.702	6.338
engine_size	11.5918	19.715	0.588	0.557	-27.341	50.524
bore	-1458.6368	901.550	-1.618	0.108	-3239.025	321.752
stroke	-1627.8471	711.318	-2.288	0.023	-3032.563	-223.131
comprassion	728.4750	364.473	1.999	0.047	8.710	1448.240
horse_power	53.4327	14.728	3.628	0.000	24.347	82.518
peak_rpm	0.0786	0.544	0.145	0.885	-0.995	1.153
city_mpg	-367.2598	134.836	-2.724	0.007	-633.535	-100.984
highway_mpg	227.9879	120.181	1.897	0.060	-9.347	465.323
fuel_type_gas	6892.8057	4829.865	1.427	0.155	-2645.251	1.64e+04

Omnibus:	30.741	**Durbin-Watson:**	0.716
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	46.240
Skew:	0.952	**Prob(JB):**	9.10e-11

Kurtosis:	4.636	**Cond. No.**	4.67e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.67e+05. This might indicate that there are strong multicollinearity or other numerical problems.

1. How do you interpret the intercept?
2. How many variables are statistically significant?
3. What is the variance of the model?
4. What is the coefficeint of determination and how do you interpret it?
5. What is the F-statistics used for? How do you interpret it for this model?

1. Intercept

In [58]:
```
lr = LinearRegression()
```

In [59]:
```
lr.fit(x,y)
```

Out[59]:
```
LinearRegression()
```

In [60]:
```
lr.intercept_
```

Out[60]:
```
-44297.847581810085
```

-44297.847581810085 is the intercept of model1

1. variables that are statistically significant with p-value less than 0.1.

Length, Width, Crub Weight, Storke, Horse Power, City/MPG are statistically significant with p-value less than 0.1 and has no zero in confidence interval.

1. Variance of Model

In [61]:
```
#variance of the model
results.mse_resid
```

Out[61]:
```
4899723.000271157
```

1. Cofficeint of determination

The coefficient of determination of model2 is 80%, 80% of the variation in y is explained by the x's.

1. F-statistics

Goodness of Fit test:

Ho : β0= β1 =...= βp=0 H1 : at least one of them ≠ 0

We reject Ho if F-statistic is large than P-value associated with it.

F-statistics = 46.45 P-value associated with it = 3.81e-49 Conclusion : We reject Null Hypothesis Ho

2. Drop all the variables that are not statistically significant at least at 90% confidence level. Run another regression model with price as the dependent variable and the rest of the variables as the independent variabls. Call it model2. Print a summary of the results and answer the questions bellow.

In [62]:
```
df2 = df2.drop(['wheel_base','length','heights','engine_size','bore','peak_rp
```

In [63]:
```
## your code goes here
traget2 = 'price' #dependent varible
```

In [64]:
```
a = df2.loc[:,df2.columns != traget2] #Dataset without dependent variable
```

In [65]:
```
b = df2.loc[:,traget2] #Dataset without independent variables
```

In [66]:
```
A = sm.add_constant(a) #adds constant
```

```
/Users/nithinreddynagapur/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/tsatools.py:142: FutureWarning: In a future version of pandas all argume
nts of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

In [67]:
```
model2 = sm.OLS(b,A) #OLS
```

In [68]:
```
results2 = model2.fit() #fitting OLS
```

In [69]:
```
results2.summary() #gives summary of MLR
```

Out[69]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.784
Model:	OLS	**Adj. R-squared:**	0.776
Method:	Least Squares	**F-statistic:**	102.1
Date:	Wed, 23 Mar 2022	**Prob (F-statistic):**	1.46e-53
Time:	14:29:12	**Log-Likelihood:**	-1605.1
No. Observations:	176	**AIC:**	3224.
Df Residuals:	169	**BIC:**	3246.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-3.837e+04	1.03e+04	-3.719	0.000	-5.87e+04	-1.8e+04
width	645.7369	166.740	3.873	0.000	316.575	974.899
curb_weight	2.9352	1.038	2.827	0.005	0.886	4.985
stroke	-1392.2620	594.605	-2.341	0.020	-2566.073	-218.451
comprassion	210.7149	68.946	3.056	0.003	74.608	346.821
horse_power	53.9260	11.033	4.888	0.000	32.146	75.706
city_mpg	-104.9062	67.902	-1.545	0.124	-238.951	29.138

Omnibus:	47.812	**Durbin-Watson:**	0.641
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	93.068
Skew:	1.282	**Prob(JB):**	6.17e-21
Kurtosis:	5.474	**Cond. No.**	1.53e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.53e+05. This might indicate that there are strong multicollinearity or other numerical problems.

1. How do you interpret the intercept?
2. How many variables are statistically significant?
3. What is the variance of the model?
4. What is the coefficeint of determination and how do you interpret it? What is the Adjusted R-squared and compare it to the model1's value.
5. What is the F-statistics used for? How do you interpret it for this model?

1. Intercept

In [70]:
```
lrm = LinearRegression()
```

In [71]:
```
lrm.fit(a,b)
```

Out[71]:
```
LinearRegression()
```

In [72]:
```
lrm.intercept_
```

Out[72]:
```
-38371.800848168634
```

-38371.800848168634 is intercept of model2

1. variables are statistically significant

intercept, width, curb_weight, stroke, comprassion, horse_power, variables are statistically significant.

3.variance

In [73]:
```
results2.mse_resid
```

Out[73]:
```
5086410.492710208
```

The coefficient of determination of model2 is 78.4%, 78.4% of the variation in y is explained by the x's. The adjusted R-squared compared to model1 77.6% is

Goodness of Fit test:

Ho : $\beta0= \beta1 =...= \beta p=0$ H1 : at least one of them $\neq 0$

We reject Ho if F-statistic is large than P-value associated with it.

F-statistics = 95.37 P-value associated with it = 1.24e-51 Conclusion : We reject Null Hypothesis Ho

3. Compare the two models with ANOVA. What are your null and alternative hypothesis? What is your conclusion?

In [74]:
```
##your code goes here
anova_lm(results2,results)
```

Out[74]:

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	169.0	8.596034e+08	0.0	NaN	NaN	NaN
1	161.0	7.888554e+08	8.0	7.074797e+07	1.804897	0.079641

H0: The full model is better

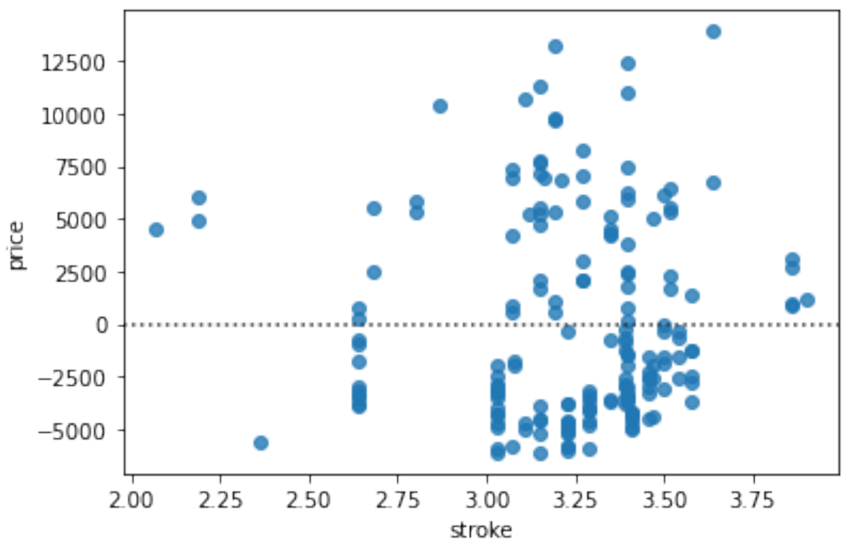
Ha: The reduced model is better

conclusion : Reject H0 and go with Ha

Residual Plots

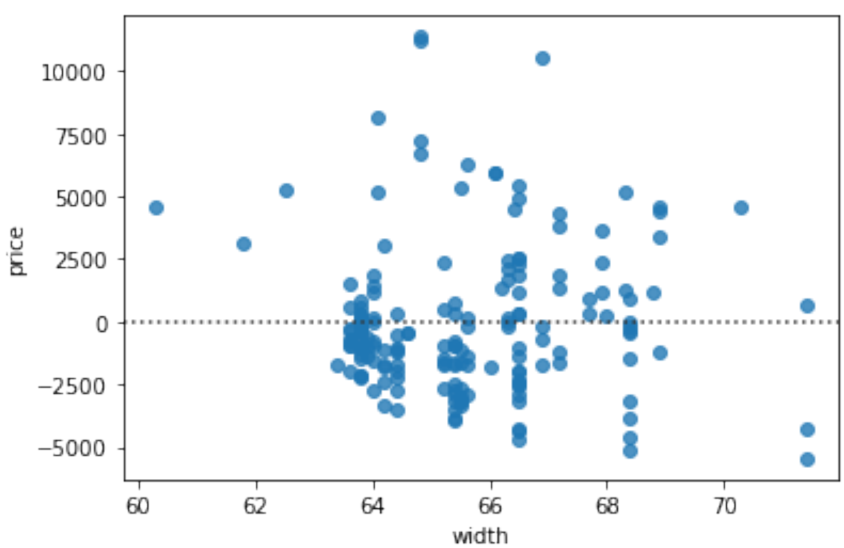
In [75]:

```
sns.residplot(y = 'price',x = 'stroke', data = df2) #Residual plot between pr
plt.show()
```



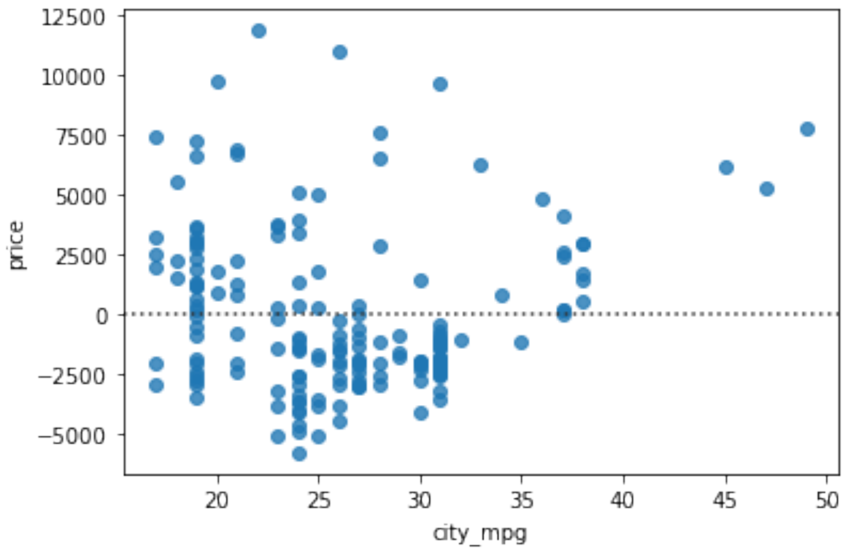
In [76]:

```
sns.residplot(y = 'price',x = 'width', data = df2) #Residual plot between pri
plt.show()
```



In [77]:

```
sns.residplot(y = 'price',x = 'city_mpg', data = df2) #Residual plot between 
plt.show()
```



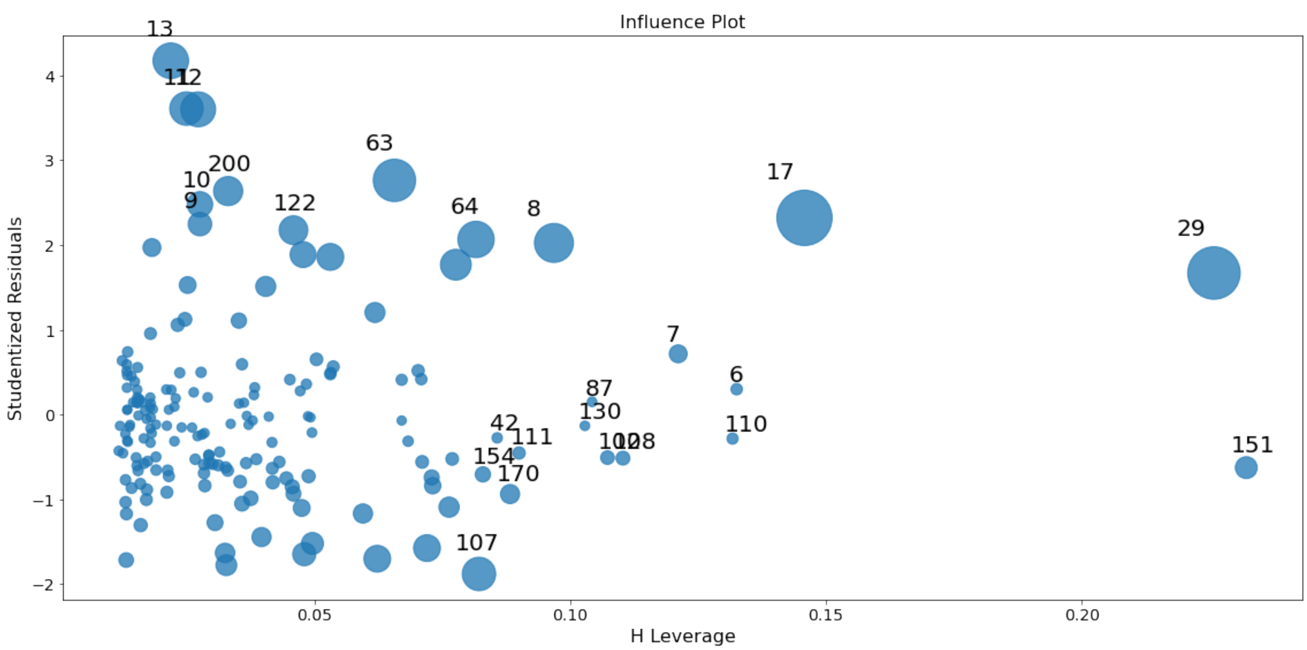
Influence Plot

In [78]:

```
plt.rc("figure", figsize=(16, 8))
plt.rc("font", size=14)
```

In [79]:

```
fig = sm.graphics.influence_plot(results2, criterion="cooks")
fig.tight_layout(pad=1.0)
```



4.Checking the assumptions:

-What are the assumptions?

-Do they hold?

To test the assumption that the errors are independent, one can use the **Durbin-Watson test**; this is the method statsmodels.stats.stattools.durbin_watson(). For this test, a value of 2, or close to it, is ideal. The statistical value ranges between 0-4 where a value closer to 0 is more evidence for positive serial correlation and a value closer to 4 is more evidence for negative serial correlation.

In [80]:
```
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from scipy import stats
from statsmodels.compat import lzip
import statsmodels
```

We have two assumptions for this dataset.

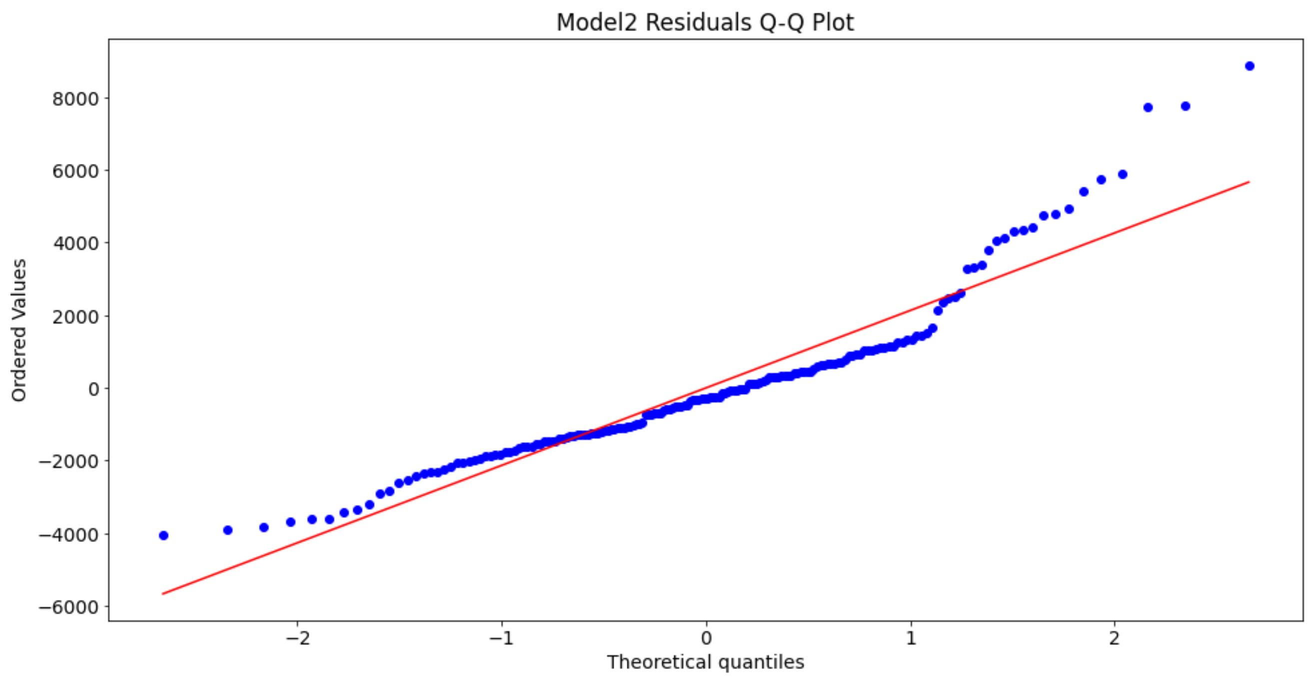
Assumption of normality: the null hypothesis is that the errors are normally distributed

In [81]:
```
name = ['Jarque-Bera', 'Chi^2 two-tail prob.', 'Skew', 'Kurtosis']
test = sms.jarque_bera(results2.resid)
lzip(name, test)
```

Out[81]:
```
[('Jarque-Bera', 93.06770024056209),
 ('Chi^2 two-tail prob.', 6.174555107770633e-21),
 ('Skew', 1.2817924416465811),
 ('Kurtosis', 5.4736779456822395)]
```

In [82]:
```
#Running plot & giving it a title
stats.probplot(results2.resid, dist="norm", plot= plt)
plt.title("Model2 Residuals Q-Q Plot")
```

Out[82]: Text(0.5, 1.0, 'Model2 Residuals Q-Q Plot')



Assumption of Homoscedasticity

The assumption of homoscedasticity is a vital assumption for linear regression. If this assumption is violated, then the standard errors will be biased. The standard errors are used to conduct significance tests, and calculate the confidence intervals.

This can be tested using a residual vs. fitted values plot, looking at a scatter plot (if a cone shape is present then heteroscedasticity is present), or by using a statistical test such as **Breusch-Pagan, Cook-Weisberg test, or White general test**. In this example, the Bruesch-Pagan test will be used.

Null hypothesis that we have constant variance (homoscedasticity)

In [83]:

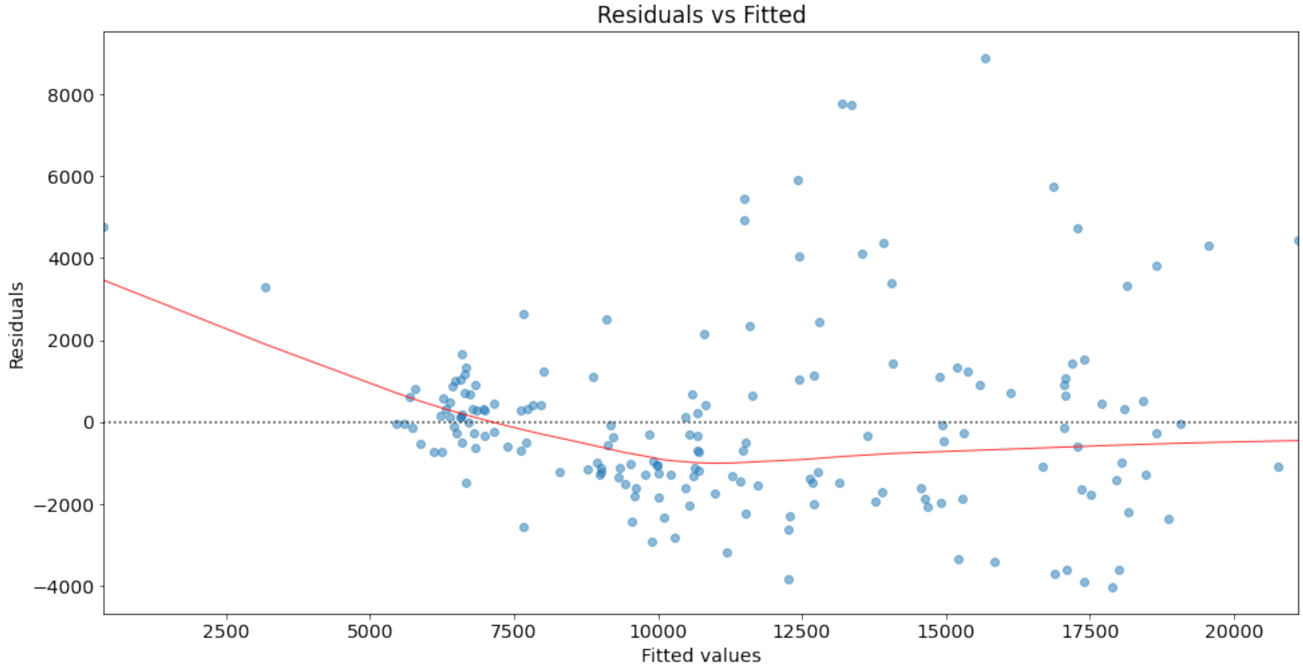
```
name = ['Lagrange multiplier statistic', 'p-value',
        'f-value', 'f p-value']
test = sms.het_breuschpagan(results2.resid, results2.model.exog)
lzip(name, test)
```

Out[83]:

```
[('Lagrange multiplier statistic', 16.87106746406179),
 ('p-value', 0.009768955035083864),
 ('f-value', 2.98626859363713),
 ('f p-value', 0.008477275380465537)]
```

In [84]:

```
# fitted values
model_fitted_y = results2.fittedvalues

#  Plot
plot = sns.residplot(x=model_fitted_y, y='price', data=df2, lowess=True,
                     scatter_kws={'alpha': 0.5},
                     line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

# Titel and labels
plot.set_title('Residuals vs Fitted')
plot.set_xlabel('Fitted values')
plot.set_ylabel('Residuals');
```



5. Is there Multicollinearity in your data?

Multicollinearity diagnosis Variance inflation factor (VIF) VIF < max(10, 1/1-R-squared)

Multicollinearity : The effect of individual varibles cannot clearly separated.

Note : *Tolerance must be lessthan 0.1 and VIR must be greaterthan 10 then there is Multicollinearity*

In [85]:

```
X_df=pd.DataFrame(x)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_df.values, i) for i in range
vif["Feature"] = X_df.columns
vif.round(2)
```

Out[85]:

	VIF Factor	Feature
0	2196.86	wheel_base
1	1947.17	length
2	3363.00	width
3	1114.91	heights
4	461.93	curb_weight
5	205.49	engine_size
6	289.40	bore
7	176.54	stroke
8	467.62	comprassion
9	79.70	horse_power
10	278.59	peak_rpm
11	470.00	city_mpg
12	541.10	highway_mpg
13	596.27	fuel_type_gas

In [86]:

```
X2_df=pd.DataFrame(a)
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X2_df.values, i) for i in rang
vif["Feature"] = X2_df.columns

vif.round()
```

Out[86]:

	VIF Factor	Feature
0	529.0	width
1	224.0	curb_weight
2	128.0	stroke
3	18.0	comprassion
4	43.0	horse_power
5	99.0	city_mpg

After seeing both models i can say that there is a multicolinearnity in our data.

Conclusion

When I began exercising what we had learned in CS746, Everything I had done in this dataset was inaccurate because of many outliers. However, I have a lot from the exercise. We have dropped many columns by doing OLS output; I have tried removing all the outliers. But, again, the dataset became too low, and I removed outliers only for the price variable. All assumptions and correlations in Multilinear regression are depended upon how to act with the dataset in EDA Part. Therefore, we should learn more about choosing a significant variable with a strong correlation dependent variable.

References

1. Sat Dataset
2. CS746 Lecture No 11.