In [1]:
```python
#import library
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:
```python
df= pd.read_csv("auto.csv")
df
```

Out[2]:

	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	110	5500	19	25	15250
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
199	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	23	28	16845
200	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	160	5300	19	25	19045
201	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	134	5500	18	23	21485
202	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	106	4800	26	27	22470
203	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	19	25	22625

204 rows × 26 columns

In [3]:
```python
# Let's give header to the datasets
headers= ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",
         "peak-rpm","city-mpg","highway-mpg","price"]
```

In [4]:
```python
df.columns=headers
```

In [5]:
```python
df.head(5)
```

Out[5]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compre
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

Identifying and handling missing data

In [6]:
```python
# Converts '?' to NaN
df.replace('?',np.nan, inplace=True)
df.head(5)
```

Out[6]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compre
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	
4	2	NaN	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

In [7]:
```python
# Checking the null value in each columns
df.isnull().sum()
```

Out[7]:
```
symboling             0
normalized-losses    40
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

Handling this missing value based on their characteristics.

In [8]:
```python
#Replacing "NaN" with mean value in "normalized-losses" column
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

In [9]:
```python
#replace the missing 'num-of-doors' values by the most frequent
df['num-of-doors'].value_counts()
```

Out[9]:
```
four    114
two      88
Name: num-of-doors, dtype: int64
```

In [10]:
```python
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

In [11]:
```python
#Replacing "NaN" with mean value in "bore" column
avg_bore=df['bore'].astype('float').mean(axis=0)
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

In [12]:
```python
#Replacing "NaN" with mean value in "stroke" column
avg_stroke=df['stroke'].astype('float').mean(axis=0)
df["stroke"].replace(np.nan, avg_stroke, inplace=True)
```

In [13]:
```python
#Replacing "NaN" with the mean value in the "horsepower" column
avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
```

In [14]:
```python
#Replacing "NaN" with the mean value in the "peak-rpm" column
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

In [15]:
```python
#Finally, let's drop all rows that do not have price data:
df.dropna(subset=["price"], axis=0, inplace=True)

# reset index, because we droped two rows
df.reset_index(drop=True, inplace=True)
```

In [16]:
```python
df.head()
```

Out[16]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compre
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	
4	2	122.0	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

In [17]:
```python
#Let's list the data types for each column
df.dtypes
```

Out[17]:
```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

In [18]:
```python
#Convert data types to proper format
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
df[["horsepower"]]=df[["horsepower"]].astype("int")
```

In [19]:
```python
df.dtypes
```

Out[19]:
```
symboling              int64
normalized-losses      int32
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower             int32
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

Let's standardized and normalized the data

In [20]:
```python
# transform mpg to L/100km by mathematical operation (235 divided by mpg)
df["highway-mpg"] = 235/df["highway-mpg"]

# rename column name from "highway-mpg" to "highway-L/100km"
df.rename(columns={"highway-mpg":'highway-L/100km'}, inplace=True)
```

In [21]:
```python
# transform mpg to L/100km by mathematical operation (235 divided by mpg)
df["city-mpg"] = 235/df["city-mpg"]

# rename column name from "city-mpg" to "city-L/100km"
df.rename(columns={"city-mpg":'city-L/100km'}, inplace=True)
```

In [22]:
```python
df.head()
```

Out[22]:

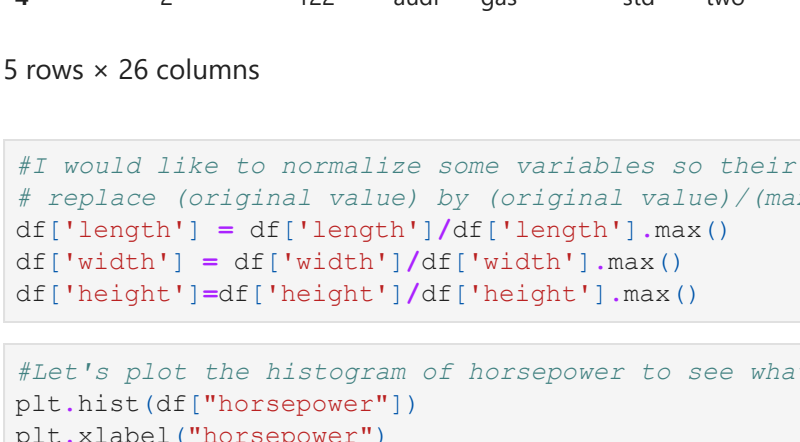
	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compre
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	
4	2	122	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	

5 rows × 26 columns

In [23]:
```python
#I would like to normalize some variables so their value ranges from 0 to 1
# replace (original value) by (original value)/(maximum value)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
df['height']=df['height']/df['height'].max()
```

In [24]:
```python
#Let's plot the histogram of horsepower to see what the distribution of horsepower looks like.
plt.hist(df["horsepower"])
plt.xlabel("horsepower")
plt.ylabel("count")
plt.title("horsepower")
```

Out[24]:
```
Text(0.5, 1.0, 'horsepower')
```



In [25]:
```python
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
group_names = ['Low', 'Medium', 'High']
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names, include_lowest=True )
df[['horsepower','horsepower-binned']].head(10)
```

Out[25]:

	horsepower	horsepower-binned
0	111	Low
1	154	Medium
2	102	Low
3	115	Low
4	110	Low
5	110	Low
6	110	Low
7	140	Medium
8	101	Low
9	101	Low

In [26]:
```python
df["horsepower-binned"].value_counts()
```

Out[26]:
```
Low       152
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

In [27]:
```python
plt.bar(group_names, df["horsepower-binned"].value_counts())
plt.xlabel("horsepower")
plt.ylabel("count")
plt.title("horsepower bins")
```

Out[27]:
```
Text(0.5, 1.0, 'horsepower bins')
```

In [28]:
```python
#We use indicator variables so we can use categorical variables for regression analysis later.
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

Out[28]:

	diesel	gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

In [29]:
```python
dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'}, inplace=True)
dummy_variable_1.head()
```

Out[29]:

	fuel-type-diesel	fuel-type-gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

In [30]:
```python
# merge data frame "df" and "dummy_variable_1"
df = pd.concat([df, dummy_variable_1], axis=1)

# drop original column "fuel-type" from "df"
df.drop("fuel-type", axis = 1, inplace=True)
```

In [31]:
```python
df.head()
```

Out[31]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	stroke	compression-ratio	horsepower
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	2.68	9.0	111
1	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	3.47	9.0	154
2	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	3.40	10.0	102
3	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	3.40	8.0	115
4	2	122	audi	std	two	sedan	fwd	front	99.8	0.851994	...	3.40	8.5	110

5 rows × 28 columns

In [32]:
```python
dummy_variable_2 = pd.get_dummies(df['aspiration'])

# change column names for clarity
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo': 'aspiration-turbo'}, inplace=True)

# show first 5 instances of data frame "dummy_variable_1"
dummy_variable_2.head()
```

Out[32]:

	aspiration-std	aspiration-turbo
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

In [33]:
```python
# merge the new dataframe to the original datafram
df = pd.concat([df, dummy_variable_2], axis=1)

# drop original column "aspiration" from "df"
df.drop('aspiration', axis = 1, inplace=True)
```

In [34]:
```python
#save the new dataset
df.to_csv('clean_df.csv')
```

Let's do EDA(Exploratory Data Analysis)

In [35]:
```python
df.corr()
```

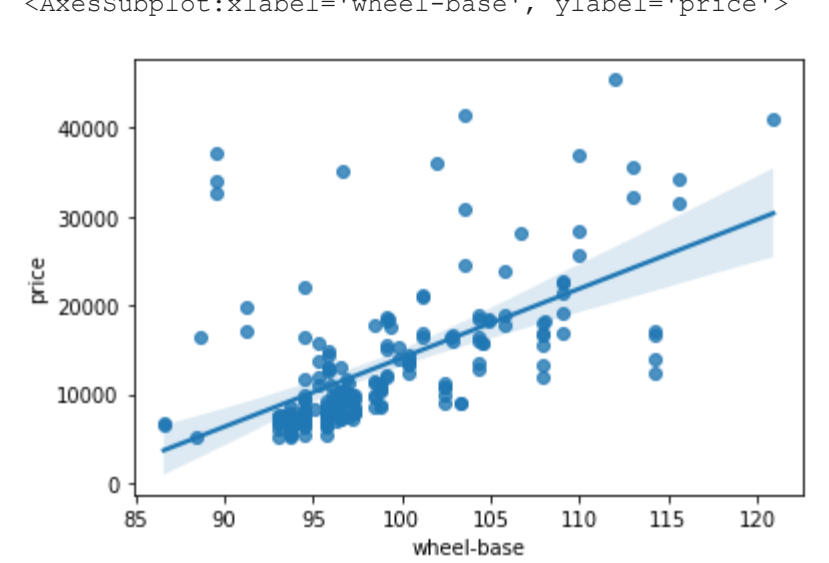
Out[35]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio
symboling	1.000000	0.469772	-0.529145	-0.364511	-0.237262	-0.542261	-0.234743	-0.112069	-0.145790	0.008072	-0.181073
normalized-losses	0.469772	1.000000	-0.057068	0.019433	0.086961	-0.377664	0.099404	0.112362	-0.029929	0.055673	-0.114738
wheel-base	-0.529145	-0.057068	1.000000	0.879005	0.814593	0.583789	0.787584	0.576779	0.501576	0.144733	0.249689
length	-0.364511	0.019433	0.879005	1.000000	0.857271	0.492955	0.881058	0.685531	0.610847	0.120929	0.159203
width	-0.237262	0.086961	0.814593	0.857271	1.000000	0.300995	0.867720	0.731100	0.548485	0.182864	0.189008
height	-0.542261	-0.377664	0.583789	0.492955	0.300995	1.000000	0.310660	0.076255	0.187917	-0.081103	0.259526
curb-weight	-0.234743	0.099404	0.787584	0.881058	0.867720	0.310660	1.000000	0.849090	0.644552	0.168669	0.156444
engine-size	-0.112069	0.112362	0.576779	0.685531	0.731100	0.076255	0.849090	1.000000	0.572878	0.208133	0.029005
bore	-0.145790	-0.029929	0.501576	0.610847	0.548485	0.187917	0.644552	0.572878	1.000000	-0.051087	0.002034
stroke	0.008072	0.055673	0.144733	0.120929	0.182864	-0.081103	0.168669	0.208133	-0.051087	1.000000	0.186780
compression-ratio	-0.181073	-0.114738	0.249689	0.159203	0.189008	0.259526	0.156444	0.029005	0.002034	0.186780	1.000000
horsepower	0.074617	0.217322	0.375610	0.580583	0.617115	-0.085829	0.758076	0.822689	0.566807	0.100881	-0.214260
peak-rpm	0.284033	0.239580	-0.365045	-0.286688	-0.247337	-0.315820	-0.279422	-0.256681	-0.267061	-0.066021	-0.436303
city-L/100km	0.062423	0.238712	0.484047	0.659174	0.677111	0.008923	0.785868	0.745337	0.553954	0.041310	-0.298898
highway-L/100km	-0.033159	0.181247	0.584953	0.708466	0.739845	0.088903	0.837217	0.783593	0.558675	0.051032	-0.222957
price	-0.083327	0.133999	0.589147	0.691044	0.752795	0.137284	0.834420	0.872337	0.543433	0.083298	0.071176
fuel-type-diesel	-0.195381	-0.101574	0.306706	0.210616	0.243448	0.281198	0.221082	0.070925	0.055395	0.240046	0.985228
fuel-type-gas	0.195381	0.101574	-0.306706	-0.210616	-0.243448	-0.281198	-0.221082	-0.070925	-0.055395	-0.240046	-0.985228
aspiration-std	0.050995	0.006915	-0.254907	-0.229294	-0.304459	-0.086531	-0.322097	-0.110278	-0.229338	-0.215805	-0.307074
aspiration-turbo	-0.050995	-0.006915	0.254907	0.229294	0.304459	0.086531	0.322097	0.110278	0.229338	0.215805	0.307074

In [36]:
```python
import seaborn as sns
```

In [37]:
```python
# Examine the correlation between 'wheel-base' and 'price'
sns.regplot(x="wheel-base", y="price", data=df)
```

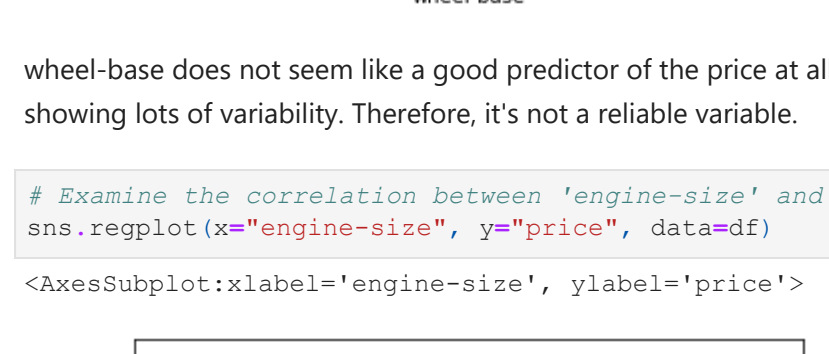
Out[37]:
```
<AxesSubplot:xlabel='wheel-base', ylabel='price'>
```



wheel-base does not seem like a good predictor of the price at all. Also, the data points are very scattered and far from the fitted line, showing lots of variability. Therefore, it's not a reliable variable.

In [38]:
```python
# Examine the correlation between 'engine-size' and 'price'
sns.regplot(x="engine-size", y="price", data=df)
```

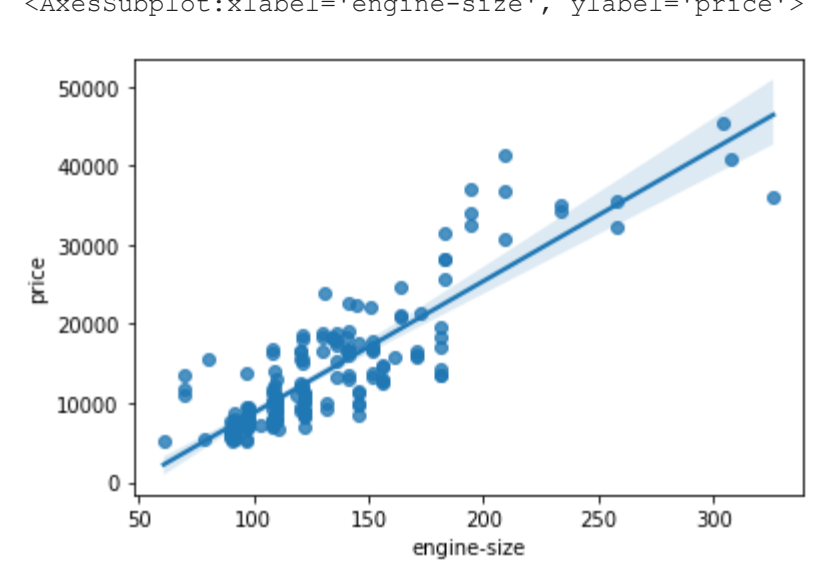
Out[38]:
```
<AxesSubplot:xlabel='engine-size', ylabel='price'>
```



As the engine-size goes up, the price goes up: this indicates a positive direct correlation between these two variables. Engine size seems like a pretty good predictor of price since the regression line is almost a perfect diagonal line.

In [39]:
```python
# Examine the correlation between 'length' and 'price'
sns.regplot(x="length", y="price", data=df)
```

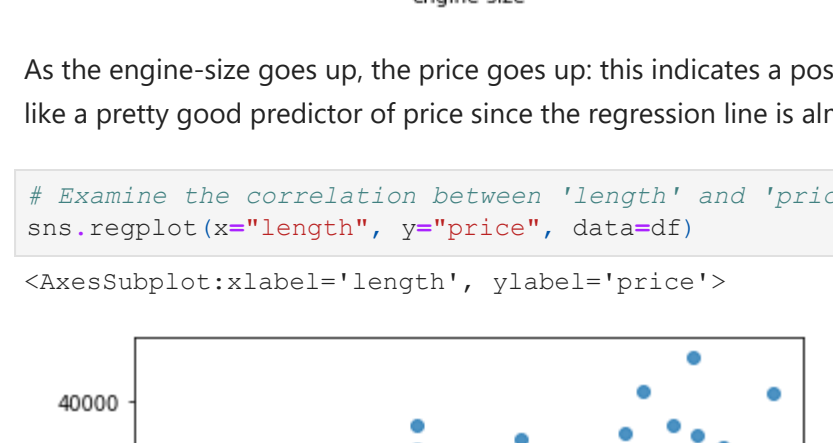
Out[39]:
```
<AxesSubplot:xlabel='length', ylabel='price'>
```



The data points are very scattered and far from the fitted line, showing lots of variability. Therefore, it's not a reliable variable.

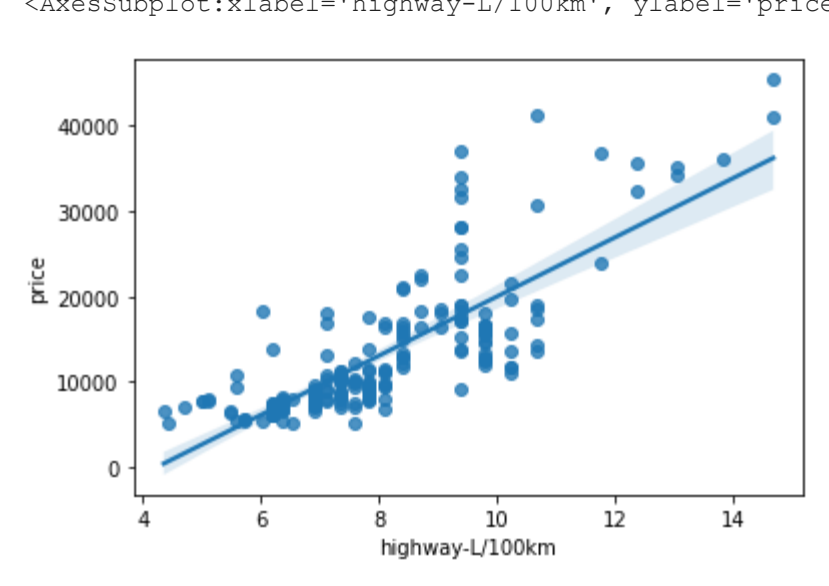
In [40]:
```python
# Examine the correlation between 'curb-weight' and 'price'
sns.regplot(x="curb-weight", y="price", data=df)
```

Out[40]:
```
<AxesSubplot:xlabel='curb-weight', ylabel='price'>
```



In [41]:
```python
# Examine the correlation between 'highway-L/100km' and 'price'
sns.regplot(x="highway-L/100km", y="price", data=df)
```

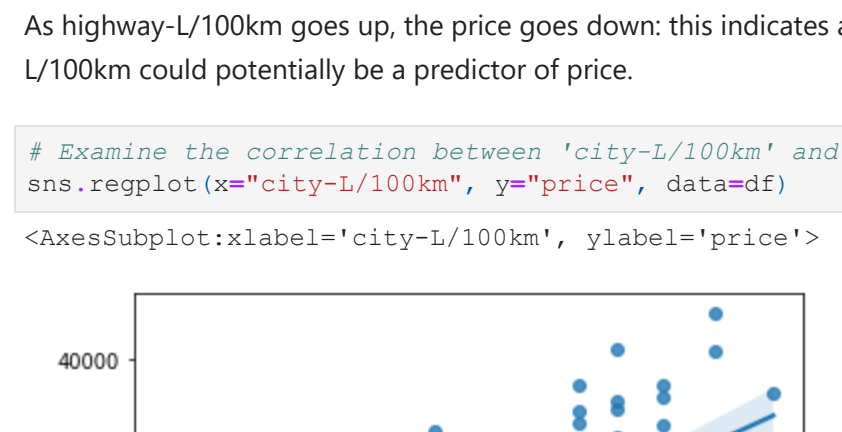
Out[41]:
```
<AxesSubplot:xlabel='highway-L/100km', ylabel='price'>
```



As highway-L/100km goes up, the price goes down: this indicates an inverse/negative relationship between these two variables. Highway-L/100km could potentially be a predictor of price.

In [42]:
```python
# Examine the correlation between 'city-L/100km' and 'price'
sns.regplot(x="city-L/100km", y="price", data=df)
```

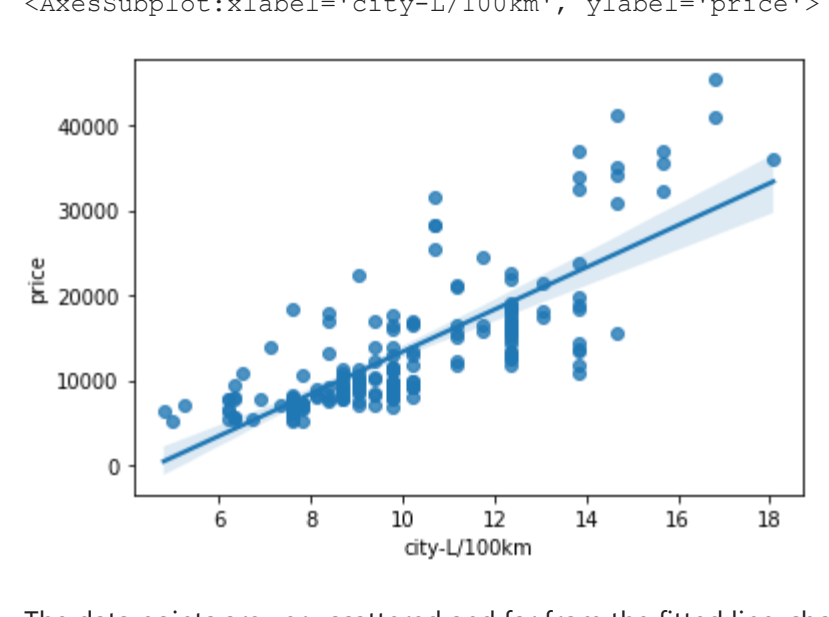
Out[42]:
```
<AxesSubplot:xlabel='city-L/100km', ylabel='price'>
```



The data points are very scattered and far from the fitted line, showing lots of variability. Therefore, it's not a reliable variable.

In [43]:
```python
# Examine the correlation between 'peak-rpm' and 'price'
sns.regplot(x="peak-rpm", y="price", data=df)
```

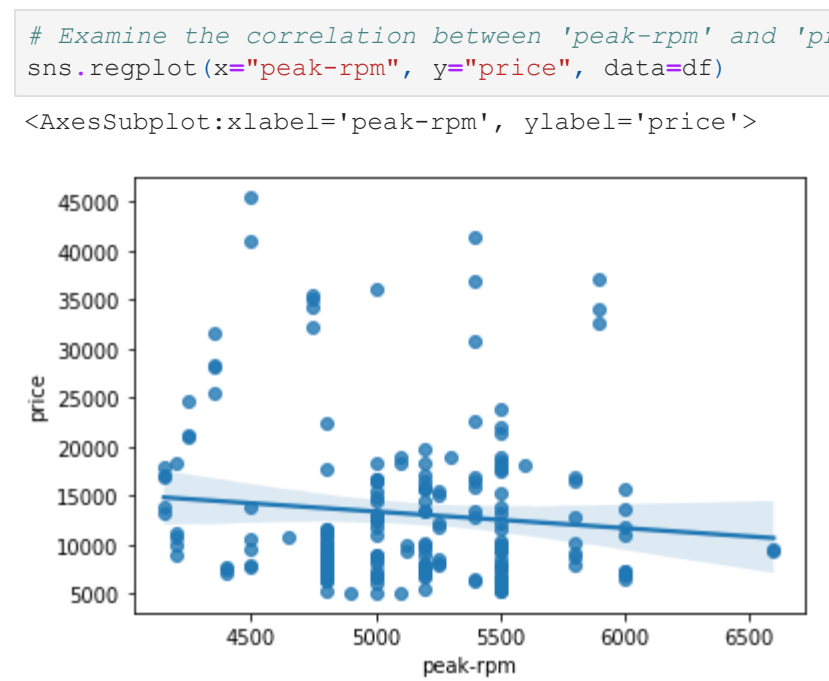
Out[43]:
```
<AxesSubplot:xlabel='peak-rpm', ylabel='price'>
```



Peak rpm does not seem like a good predictor of the price at all since the regression line is close to horizontal. Also, the data points are very scattered and far from the fitted line, showing lots of variability. Therefore, it's not a reliable variable.

In [44]:
```python
# Examine the correlation between 'stroke' and 'price'
sns.regplot(x="stroke", y="price", data=df)
```

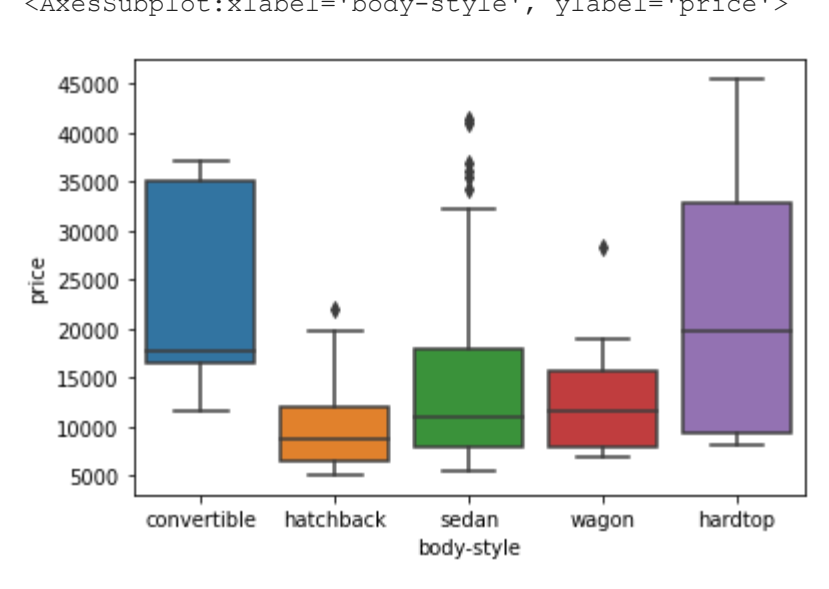
Out[44]:
```
<AxesSubplot:xlabel='stroke', ylabel='price'>
```



There is a weak correlation between the variable 'stroke' and 'price.' as such regression will not work well.

In [45]:
```python
#Let's look at the relationship between "body-style" and "price":
sns.boxplot(x="body-style", y="price", data=df)
```

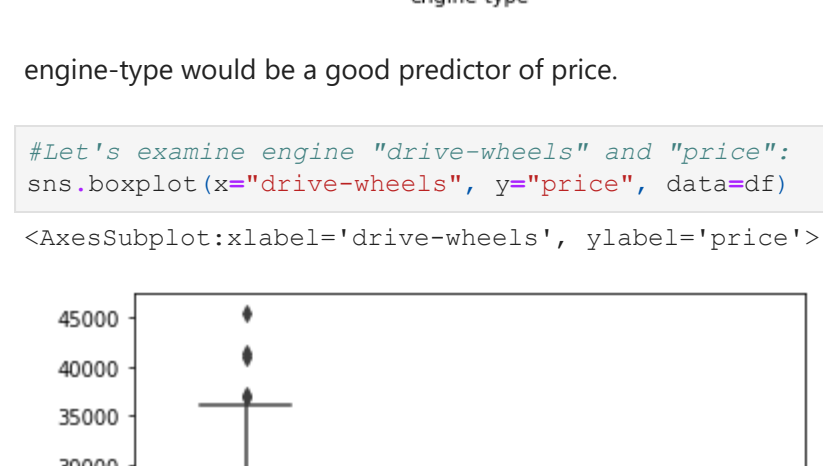
Out[45]:
```
<AxesSubplot:xlabel='body-style', ylabel='price'>
```



We see that the distributions of price between the different body-style categories have a significant overlap, so body-style would not be a good predictor of price.

In [46]:
```python
#Let's examine engine "engine-location" and "price":
sns.boxplot(x="engine-location", y="price", data=df)
```

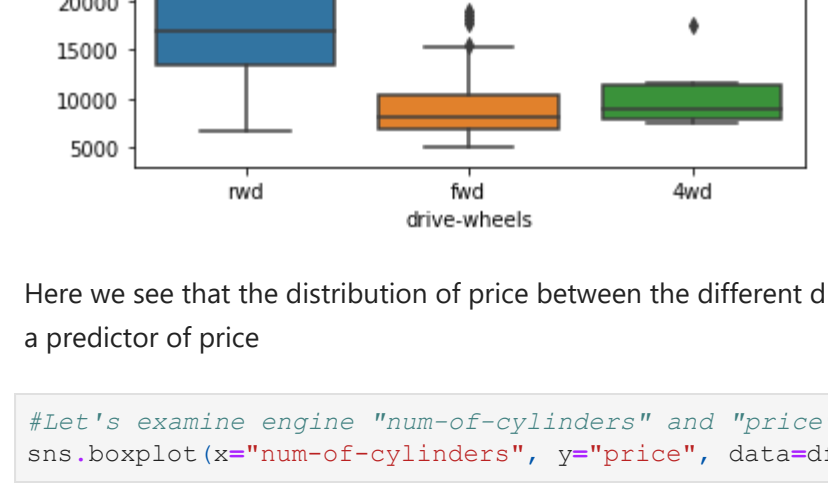
Out[46]:
```
<AxesSubplot:xlabel='engine-location', ylabel='price'>
```



Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.

In [47]:
```python
#Let's examine engine "engine-type" and "price":
sns.boxplot(x="engine-type", y="price", data=df)
```

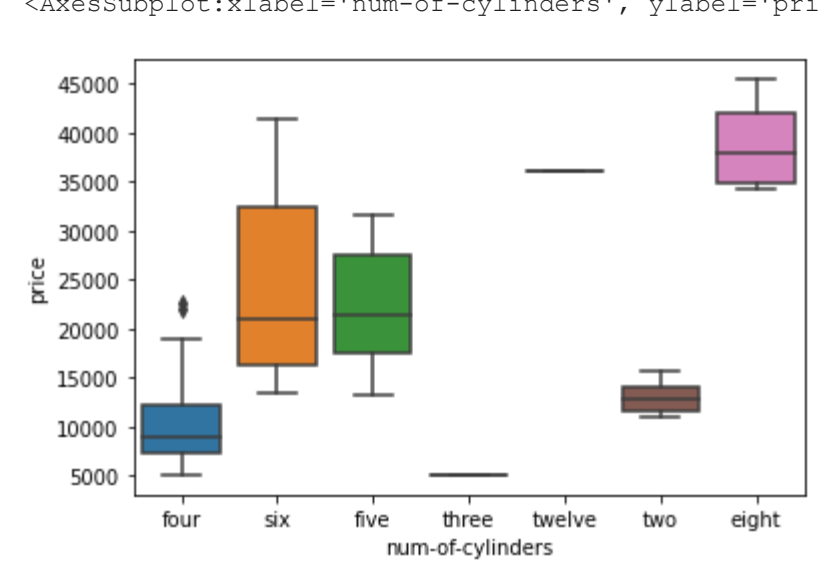
Out[47]:
```
<AxesSubplot:xlabel='engine-type', ylabel='price'>
```



engine-type would be a good predictor of price.

In [48]:
```python
#Let's examine engine "drive-wheels" and "price":
sns.boxplot(x="drive-wheels", y="price", data=df)
```

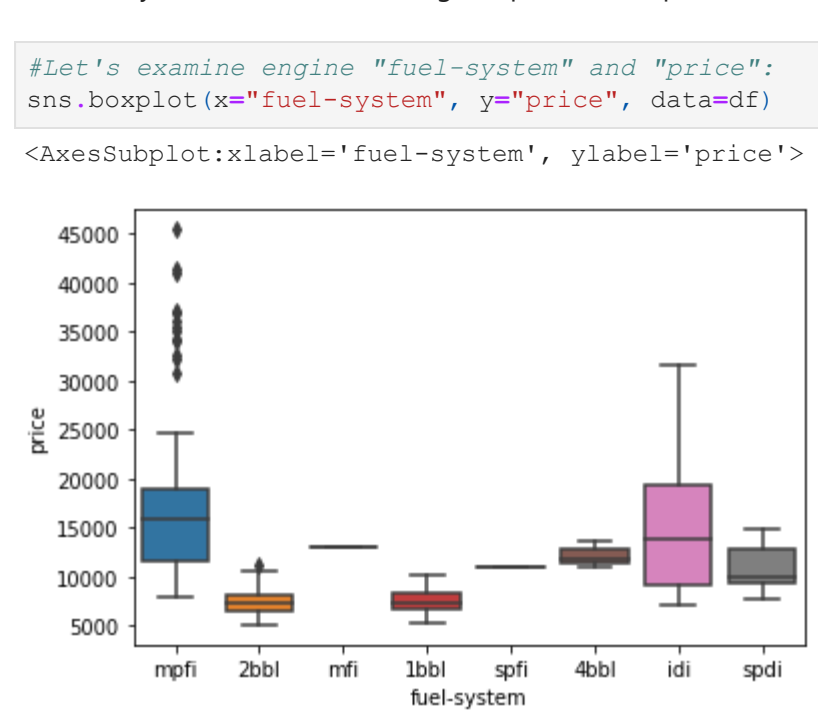
Out[48]:
```
<AxesSubplot:xlabel='drive-wheels', ylabel='price'>
```



Here we see that the distribution of price between the different drive-wheels categories differs. As such, drive-wheels could potentially be a predictor of price

In [49]:
```python
#Let's examine engine "num-of-cylinders" and "price":
sns.boxplot(x="num-of-cylinders", y="price", data=df)
```

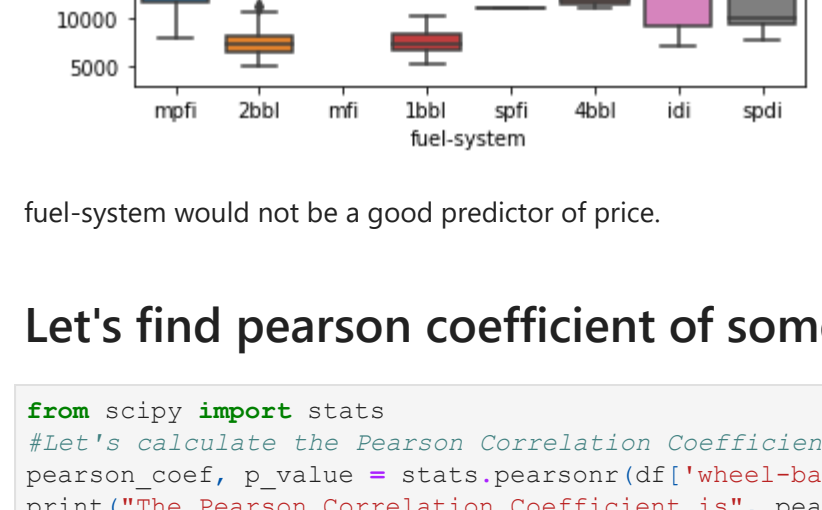
Out[49]:
```
<AxesSubplot:xlabel='num-of-cylinders', ylabel='price'>
```



num-of-cylinders would not be a good predictor of price.

In [50]:
```python
#Let's examine engine "fuel-system" and "price":
sns.boxplot(x="fuel-system", y="price", data=df)
```

Out[50]:
```
<AxesSubplot:xlabel='fuel-system', ylabel='price'>
```



fuel-system would not be a good predictor of price.

Let's find pearson coefficient of some variables.

In [51]:
```python
from scipy import stats
#Let's calculate the Pearson Correlation Coefficient and P-value of 'wheel-base' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['wheel-base'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_value)
```

```
The Pearson Correlation Coefficient is 0.5891470005448702  with a P-value of P = 4.4570195020504053e-20
```

Since the p-value is < 0.001, the correlation between wheel-base and price is statistically significant, although the linear relationship isn't extremely strong (~0.585).

In [52]:
```python
#Let's calculate the Pearson Correlation Coefficient and P-value of 'horsepower' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
```

```
The Pearson Correlation Coefficient is 0.8096565575365612  with a P-value of P =  1.0468839625927006e-47
```

Since the p-value is < 0.001, the correlation between horsepower and price is statistically significant, and the linear relationship is quite strong (~0.809, close to 1).

In [53]:
```python
##Let's calculate the Pearson Correlation Coefficient and P-value of 'length' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['length'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
```
```
The Pearson Correlation Coefficient is 0.6910440897821903  with a P-value of P =  9.960963222348891e-30
```
Since the p-value is <0.001, the correlation between length and price is statistically significant, and the linear relationship is moderately strong (~0.691).

In [54]:
```python
###Let's calculate the Pearson Correlation Coefficient and P-value of 'width' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['width'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_value )
```
```
The Pearson Correlation Coefficient is 0.7527948631832609  with a P-value of P = 8.25671414830892e-38
```
Since the p-value is < 0.001, the correlation between width and price is statistically significant, and the linear relationship is quite strong (~0.751).

In [55]:
```python
###Let's calculate the Pearson Correlation Coefficient and P-value of 'curb-weight' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['curb-weight'], df['price'])
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
```
```
The Pearson Correlation Coefficient is 0.8344204348498461  with a P-value of P =  3.9699775360220333e-53
```
Since the p-value is < 0.001, the correlation between curb-weight and price is statistically significant, and the linear relationship is quite strong (~0.834).

In [56]:
```python
###Let's calculate the Pearson Correlation Coefficient and P-value of 'engine-size' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['engine-size'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_value)
```
```
The Pearson Correlation Coefficient is 0.8723367498521141  with a P-value of P = 1.8977171466564027e-63
```
Since the p-value is < 0.001, the correlation between engine-size and price is statistically significant, and the linear relationship is quite strong (~0.872).

In [57]:
```python
###Let's calculate the Pearson Correlation Coefficient and P-value of 'bore' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['bore'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_value)
```
```
The Pearson Correlation Coefficient is 0.543432593555568  with a P-value of P = 9.207487524195135e-17
```
Since the p-value is < 0.001, the correlation between bore and price is statistically significant, but the linear relationship is only moderate (~0.521).

In [58]:
```python
###Let's calculate the Pearson Correlation Coefficient and P-value of 'city-L/100km' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['city-L/100km'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_value)
```
```
The Pearson Correlation Coefficient is 0.7902908680181893  with a P-value of P = 5.414138235107717e-44
```
Since the p-value is < 0.001, the correlation between city-L/100km and price is statistically significant, and the coefficient of about -0.687 shows that the relationship is negative and moderately strong.

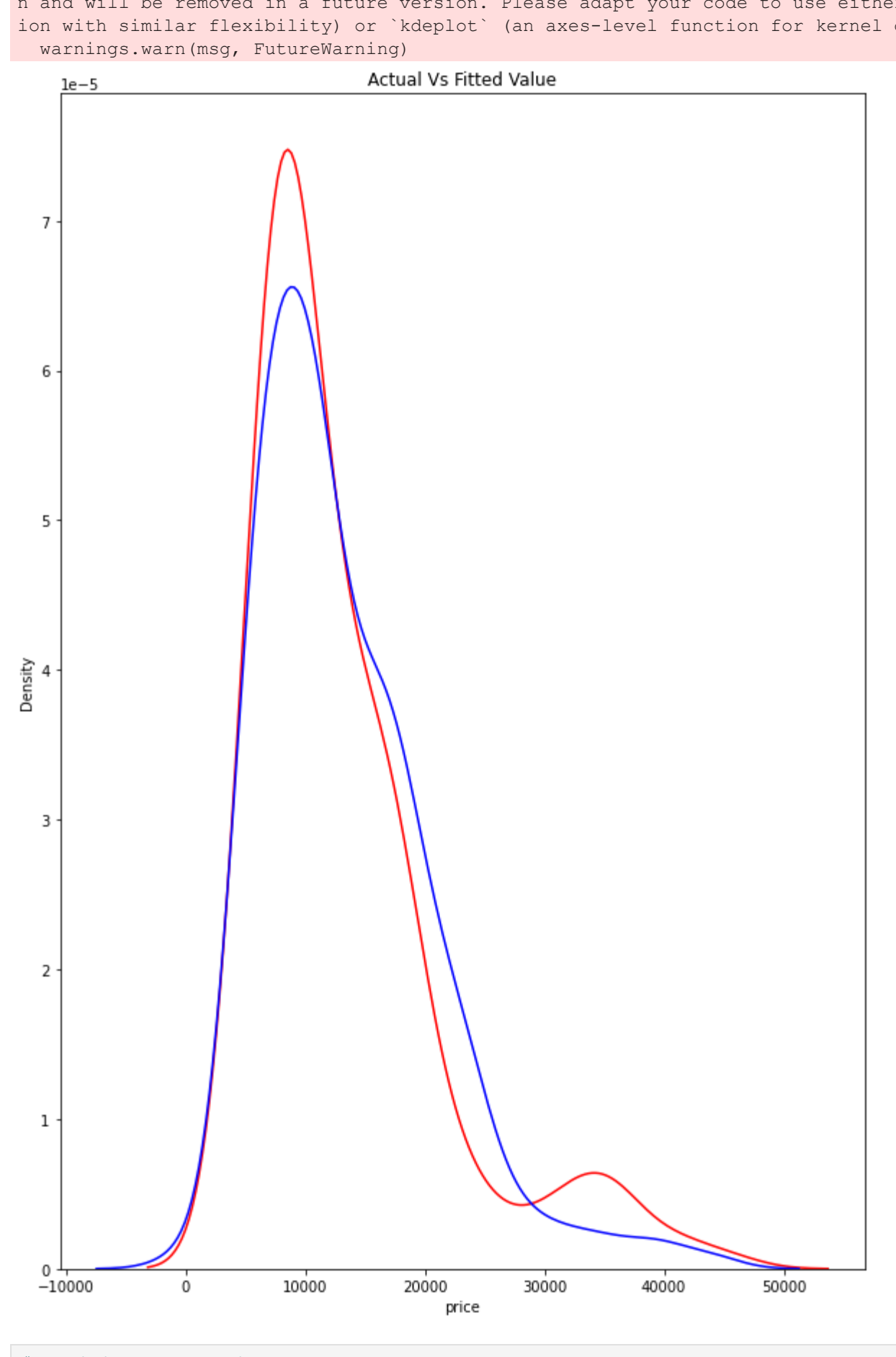
In [59]:
```python
###Let's calculate the Pearson Correlation Coefficient and P-value of 'highway-L/100km' and 'price'.
pearson_coef, p_value = stats.pearsonr(df['highway-L/100km'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P =", p_value)
```
```
The Pearson Correlation Coefficient is 0.8013130144661037  with a P-value of P = 4.6790420242570596e-46
```
Since the p-value is < 0.001, the correlation between highway-L/100km and price is statistically significant, and the coefficient of about -0.705 shows that the relationship is negative and moderately strong.

In [60]:
```python
#Now here we use multiple linear regression to predict
from sklearn.linear_model import LinearRegression
```
In [61]:
```python
Z=df[['curb-weight','engine-size','highway-L/100km','horsepower']]
```
In [62]:
```python
lm= LinearRegression()
lm
```
Out[62]:
```
▾ LinearRegression
LinearRegression()
```
In [63]:
```python
df.head(10)
```
Out[63]:

	symboling	normalized-losses	make	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	...	horsepower	peak-rpm	city-L/100km	highw L/100
0	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148	0.890278	...	111	5000.0	11.190476	8.703
1	1	122	alfa-romero	two	hatchback	rwd	front	94.5	0.822681	0.909722	...	154	5000.0	12.368421	9.038
2	2	164	audi	four	sedan	fwd	front	99.8	0.848630	0.919444	...	102	5500.0	9.791667	7.833
3	2	164	audi	four	sedan	4wd	front	99.4	0.848630	0.922222	...	115	5500.0	13.055556	10.681
4	2	122	audi	two	sedan	fwd	front	99.8	0.851994	0.920833	...	110	5500.0	12.368421	9.400
5	1	158	audi	four	sedan	fwd	front	105.8	0.925997	0.991667	...	110	5500.0	12.368421	9.400
6	1	122	audi	four	wagon	fwd	front	105.8	0.925997	0.991667	...	110	5500.0	12.368421	9.400
7	1	158	audi	four	sedan	fwd	front	105.8	0.925997	0.991667	...	140	5500.0	13.823529	11.750
8	2	192	bmw	two	sedan	rwd	front	101.2	0.849592	0.900000	...	101	5800.0	10.217391	8.103
9	0	192	bmw	four	sedan	rwd	front	101.2	0.849592	0.900000	...	101	5800.0	10.217391	8.103

10 rows × 29 columns

In [ ]:

In [64]:
```python
lm.fit(Z,df['price'])
```
Out[64]:
```
▾ LinearRegression
LinearRegression()
```
In [65]:
```python
Yhat=lm.predict(Z)
```
In [66]:
```python
plt.figure(figsize=(10,15))
ax1=sns.distplot(df['price'],hist=False,color='r',label='Actual Value')
sns.distplot(Yhat,hist=False,color='b',label='Fitted Value',ax= ax1)
plt.title('Actual Vs Fitted Value')
plt.show()
```
```
C:\Python310\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Python310\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```


In [80]:
```python
# training and Testing the datasets
from sklearn.model_selection import train_test_split
```
In [82]:
```python
x=Z
y=df['price']
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.3,random_state=0)
```
In [83]:
```python
clf= LinearRegression()
clf
```
Out[83]:
```
▾ LinearRegression
LinearRegression()
```
In [84]:
```python
clf.fit(x_train,y_train)
```
Out[84]:
```
▾ LinearRegression
LinearRegression()
```
In [85]:
```python
clf.predict(x_test)
```
Out[85]:
```
array([ 4531.06824483, 11355.3012862 , 16504.22223609, 23514.08914549,
        6298.24300816,  6986.07005584, 16484.70137173,  6052.58685631,
       19783.56280635,  7011.00383782, 15333.15369369, 13119.72897035,
        8941.77537256,  7373.02908166, 18856.16584123, 19418.55542794,
        6645.27007832, 13724.37750731,  9628.54848945,  5877.64873715,
        9699.10502501,  5656.59067451, 41918.14960728,  8001.6631518 ,
       11436.32898369, 20825.98978772,  6310.58542394, 10231.91612978,
       20825.98978772, 35199.75120891,  -774.80042587, 11271.76343985,
       11695.78211693, 16495.27919286,  5548.28525705, 16111.47228509,
       13514.55643891, 15098.23666485, 10663.89318152, 20379.78222967,
        9244.27230033, 24399.06220376, 12395.00419375, 17760.32080719,
       10530.89880949, 23349.95686175, 10810.01933169,  8602.20461409,
        7178.41371075, 10997.83170732, 17239.07115882, 21007.01188595,
       10367.68270345, 22687.14728767,  8322.18846812,  5638.79630616,
        3444.56425705,  9340.29281619, 10012.42754584, 11374.61690475])
```
In [86]:
```python
clf.score(x_test,y_test)
```
Out[86]:
```
0.7343178154693659
```
In [88]:
```python
clf.coef_
```
Out[88]:
```
array([   4.1141386 ,   88.57017602, 1070.31023174,   -4.47152162])
```
In [89]:
```python
clf.intercept_
```
Out[89]:
```
-16830.50153506208
```
In [92]:
```python
# Now after finding coefficient and intercept values and predict for curb-weight=2500, engine-size=140,highway=
clf.predict([[2500,140,10,125]])
```
```
C:\Python310\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```
Out[92]:
```
array([15998.83171355])
```
That's show the price of the car

In [ ]: