In [30]:
```python
import pandas as pd
url = ("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-S
df = pd.read_csv(url,header = 0)
```

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

Out[31]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	horsepower	peak-rpm
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154.0	5000.0
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102.0	5500.0
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115.0	5500.0

5 rows × 29 columns

In [32]:
```python
df.columns
```

Out[32]:
```
Index(['symboling', 'normalized-losses', 'make', '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',
       'city-L/100km', 'horsepower-binned', 'diesel', 'gas'],
      dtype='object')
```

In [33]:
```python
cols = list(df.columns)

# move the outcome column to the end
cols.append(cols.pop(cols.index('price')))

# reorder the columns in the dataframe
df = df[cols]

# save the updated dataframe back to the CSV file
df.to_csv('url', index=False)
```

In [ ]:

In [29]:
```python
df.describe()
```

Out[29]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compressi ra
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.000000	201.0000
mean	0.840796	122.00000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.256904	10.1642
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.296727	41.546834	0.268072	0.319256	4.0049
min	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.070000	7.0000
25%	0.000000	101.00000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.110000	8.6000
50%	1.000000	122.00000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.290000	9.0000
75%	2.000000	137.00000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.410000	9.4000
max	3.000000	256.00000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.0000

In [23]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```python
import pandas as pd


# Select the object features to convert
object_features = ['feature1', 'feature2', 'feature3']

# Convert the selected object features to float
df[object_features] = df[object_features].astype(float)

# Save the updated dataframe to a new CSV file
df.to_csv('updated_data.csv', index=False)
```

In [4]:
```python
df.dtypes
```

Out[4]:
```
symboling              int64
normalized-losses      int64
make                  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           float64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
city-L/100km         float64
horsepower-binned     object
diesel                 int64
gas                    int64
dtype: object
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 29 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          201 non-null    int64
 1   normalized-losses  201 non-null    int64
 2   make               201 non-null    object
 3   aspiration         201 non-null    object
 4   num-of-doors       201 non-null    object
 5   body-style         201 non-null    object
 6   drive-wheels       201 non-null    object
 7   engine-location    201 non-null    object
 8   wheel-base         201 non-null    float64
 9   length             201 non-null    float64
 10  width              201 non-null    float64
 11  height             201 non-null    float64
 12  curb-weight        201 non-null    int64
 13  engine-type        201 non-null    object
 14  num-of-cylinders   201 non-null    object
 15  engine-size        201 non-null    int64
 16  fuel-system        201 non-null    object
 17  bore               201 non-null    float64
 18  stroke             197 non-null    float64
 19  compression-ratio  201 non-null    float64
 20  horsepower         201 non-null    float64
 21  peak-rpm           201 non-null    float64
 22  city-mpg           201 non-null    int64
 23  highway-mpg        201 non-null    int64
 24  price              201 non-null    float64
 25  city-L/100km       201 non-null    float64
 26  horsepower-binned  200 non-null    object
 27  diesel             201 non-null    int64
 28  gas                201 non-null    int64
dtypes: float64(11), int64(8), object(10)
memory usage: 45.7+ KB
```

In [ ]:

In [ ]:

In [6]:
```python
df.corr()
```

Out[6]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	h
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.008245	-0.182196	
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029862	0.055563	-0.114713	
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.158502	0.250313	
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608971	0.124139	0.159733	
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544885	0.188829	0.189867	
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180449	-0.062704	0.259737	
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072	0.644060	0.167562	0.156433	
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572609	0.209523	0.028889	
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.000000	-0.055390	0.001263	
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209523	-0.055390	1.000000	0.187923	
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187923	1.000000	
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098462	-0.214514	
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.065713	-0.435780	
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027	-0.034696	0.331425	
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309	-0.035201	0.268465	
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	0.071107	
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.745059	0.554610	0.037300	-0.299372	
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	0.241303	0.985231	
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241303	-0.985231	

In [7]:

```
#Find the correlation between the following columns: bore, stroke, compression-ratio, and horsepower

df[['bore', 'stroke', 'compression-ratio', 'horsepower']].corr()
```

Out[7]:

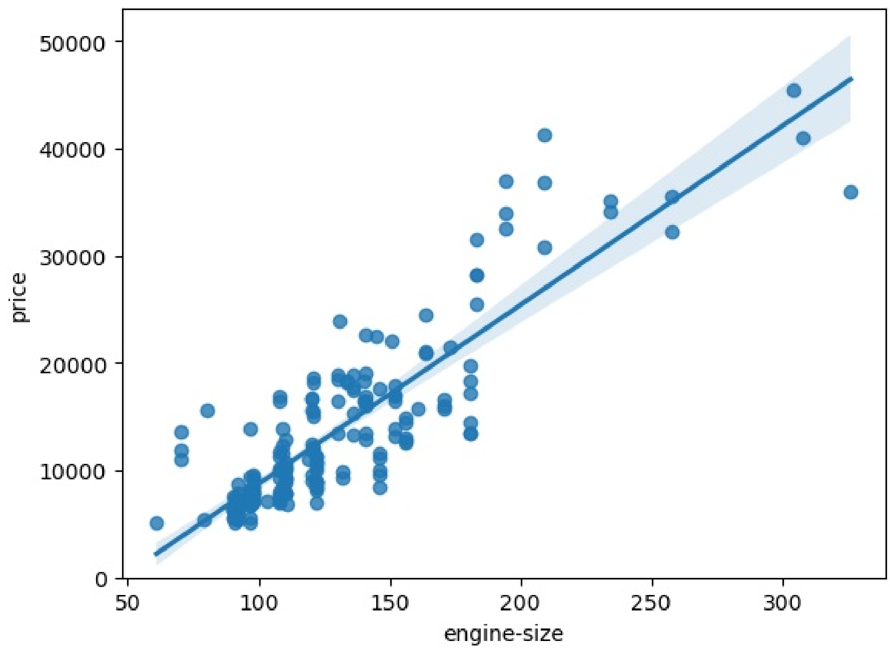
	bore	stroke	compression-ratio	horsepower
bore	1.000000	-0.055390	0.001263	0.566936
stroke	-0.055390	1.000000	0.187923	0.098462
compression-ratio	0.001263	0.187923	1.000000	-0.214514
horsepower	0.566936	0.098462	-0.214514	1.000000

In [8]:

```
# Engine size as potential predictor variable of price

sns.regplot(x="engine-size", y="price", data=df)
plt.ylim(0,)
```

Out[8]:

```
(0.0, 53042.53513934235)
```



In [9]:

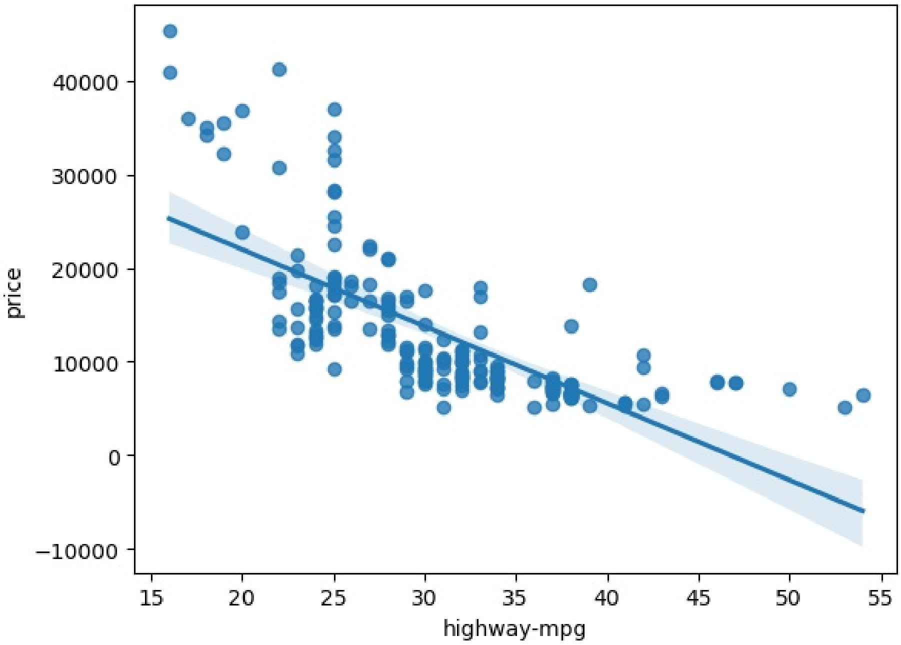
```
df[["engine-size", "price"]].corr()
```

Out[9]:

	engine-size	price
engine-size	1.000000	0.872335
price	0.872335	1.000000

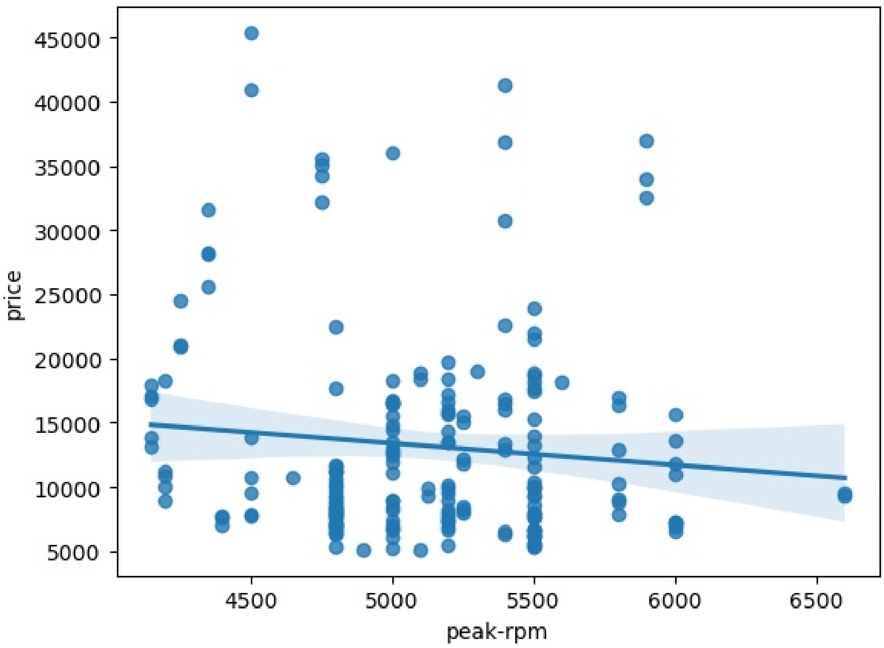
In [10]:
```
sns.regplot(x="highway-mpg", y="price", data=df)
```

Out[10]: <AxesSubplot:xlabel='highway-mpg', ylabel='price'>



In [11]:
```
sns.regplot(x="peak-rpm", y="price", data=df)
```

Out[11]: <AxesSubplot:xlabel='peak-rpm', ylabel='price'>



In [12]:
```
df[['peak-rpm','price']].corr()
```

Out[12]:

	peak-rpm	price
peak-rpm	1.000000	-0.101616
price	-0.101616	1.000000

In [13]:
```
df[['stroke','price']].corr()
```

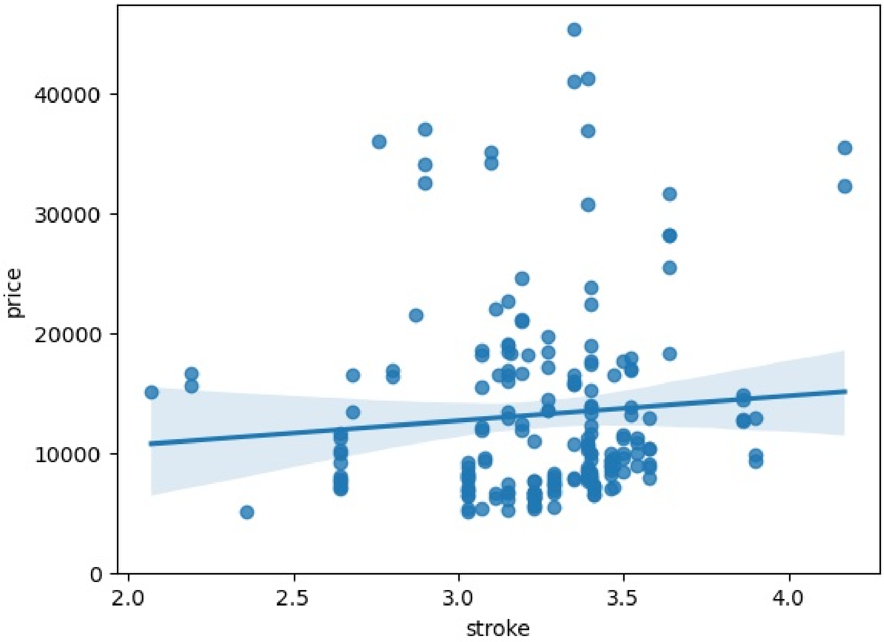
Out[13]:

	stroke	price
stroke	1.00000	0.08231
price	0.08231	1.00000

In [14]:
```
sns.regplot(x="stroke", y="price", data=df)
```

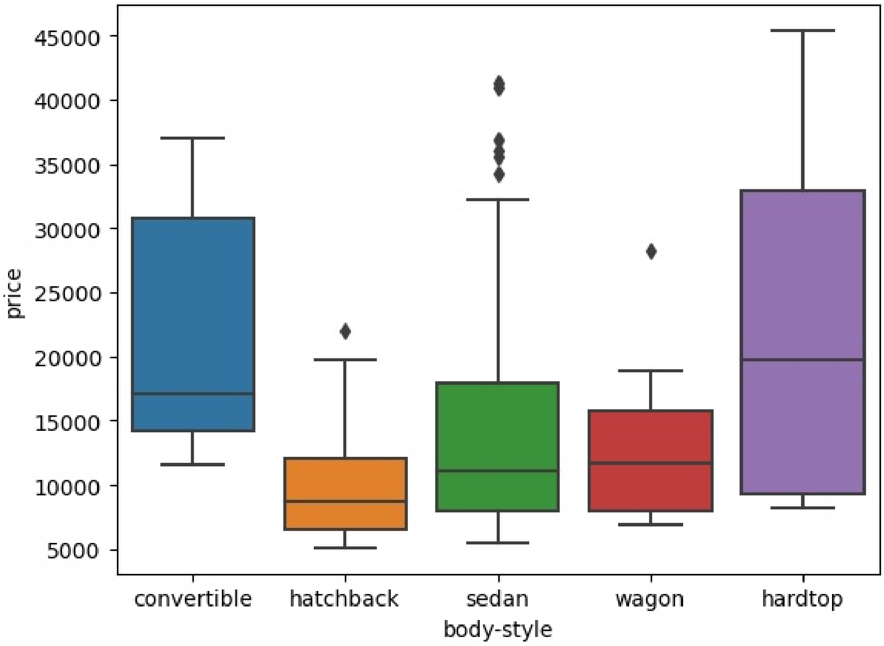
```
plt.ylim(0,)
```

Out[14]: (0.0, 47414.1)



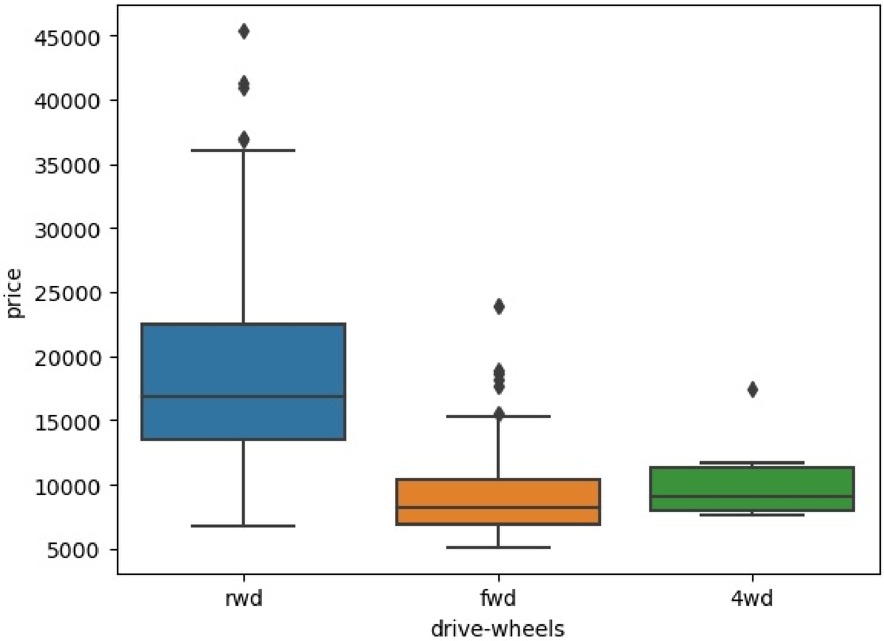
In [15]:
```
sns.boxplot(x="body-style", y="price", data=df)
```

Out[15]: <AxesSubplot:xlabel='body-style', ylabel='price'>



In [16]:
```
sns.boxplot(x="drive-wheels", y="price", data=df)
```

Out[16]: <AxesSubplot:xlabel='drive-wheels', ylabel='price'>



In [17]:
```
df.describe()
```

Out[17]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compressi ra
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.000000	201.000(
mean	0.840796	122.00000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.256904	10.164:
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.296727	41.546834	0.268072	0.319256	4.004!
min	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000(
25%	0.000000	101.00000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.110000	8.600(
50%	1.000000	122.00000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000(
75%	2.000000	137.00000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.410000	9.400(
max	3.000000	256.00000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000(

In [18]:
```
df.describe(include=['object'])
```

Out[18]:

	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system	horsepower-binned
count	201	201	201	201	201	201	201	201	201	200
unique	22	2	2	5	3	2	6	7	8	3
top	toyota	std	four	sedan	fwd	front	ohc	four	mpfi	Low
freq	32	165	115	94	118	198	145	157	92	115

In [19]:
```
df['drive-wheels'].value_counts()
```

Out[19]:
```
fwd     118
rwd      75
4wd       8
Name: drive-wheels, dtype: int64
```

In [20]:
```
df['drive-wheels'].value_counts().to_frame()
```

Out[20]:

	drive-wheels
fwd	118
rwd	75
4wd	8

In [21]:
```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm
```

Out[21]:
```
LinearRegression()
```

In [22]:
```
X = df[["length"]]
Y = df["price"]

lm.fit(X,Y)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
Yhat = lm.predict(X)
Yhat[0:10]
```

Out[23]:
```
array([10801.45044105, 10801.45044105, 11870.44409178, 14275.67980594,
       14275.67980594, 14587.46962074, 21446.8455463 , 21446.8455463 ,
       21446.8455463 , 14364.76261017])
```

In [24]:
```
lm.intercept_
```

Out[24]:
```
-64384.436327421616
```

In [25]:
```
lm.coef_
```

Out[25]:
```
array([92690.65779928])
```

In [75]:
```
Price = -64384.436327421616+92690.65779928,df[['length']]
```

In [27]:
```
lm1 = LinearRegression()
lm1

LinearRegression()
```

Out[27]:
```
LinearRegression()
```

In [28]:
```
lm1.fit(df[["engine-size"]], df[["price"]])
lm1
```

Out[28]:
```
LinearRegression()
```

In [29]:
```
lm1.coef_
```

Out[29]:
```
array([[166.86001569]])
```

In [30]:
```
lm1.intercept_
```

Out[30]:
```
array([-7963.33890628])
```

In [36]:
```
Price = 166.86 -7963,df[["engine-size"]]
```

In [37]:
```
Z= df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

In [38]:
```
lm.fit(Z,df["price"])
```

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

In [39]:
```
lm.intercept_
```

Out[39]:
```
-15806.624626329198
```

In [40]:
```
lm.coef_
```

Out[40]:
```
array([53.49574423,  4.70770099, 81.53026382, 36.05748882])
```

In [41]:
```
#Price = -15806.624 + 53.49574423horsepower + 4.70770099curb-weight + 81.53026382engine-size + 36.05748882highw
```

In [42]:
```
lm2 = LinearRegression()
lm2.fit(df[['normalized-losses' , 'highway-mpg']],df['price'])
```

Out[42]:
```
LinearRegression()
```

In [43]:
```
lm2.coef_
```

Out[43]:
```
array([   1.49789586, -820.45434016])
```

In [44]:
```
lm2.intercept_
```

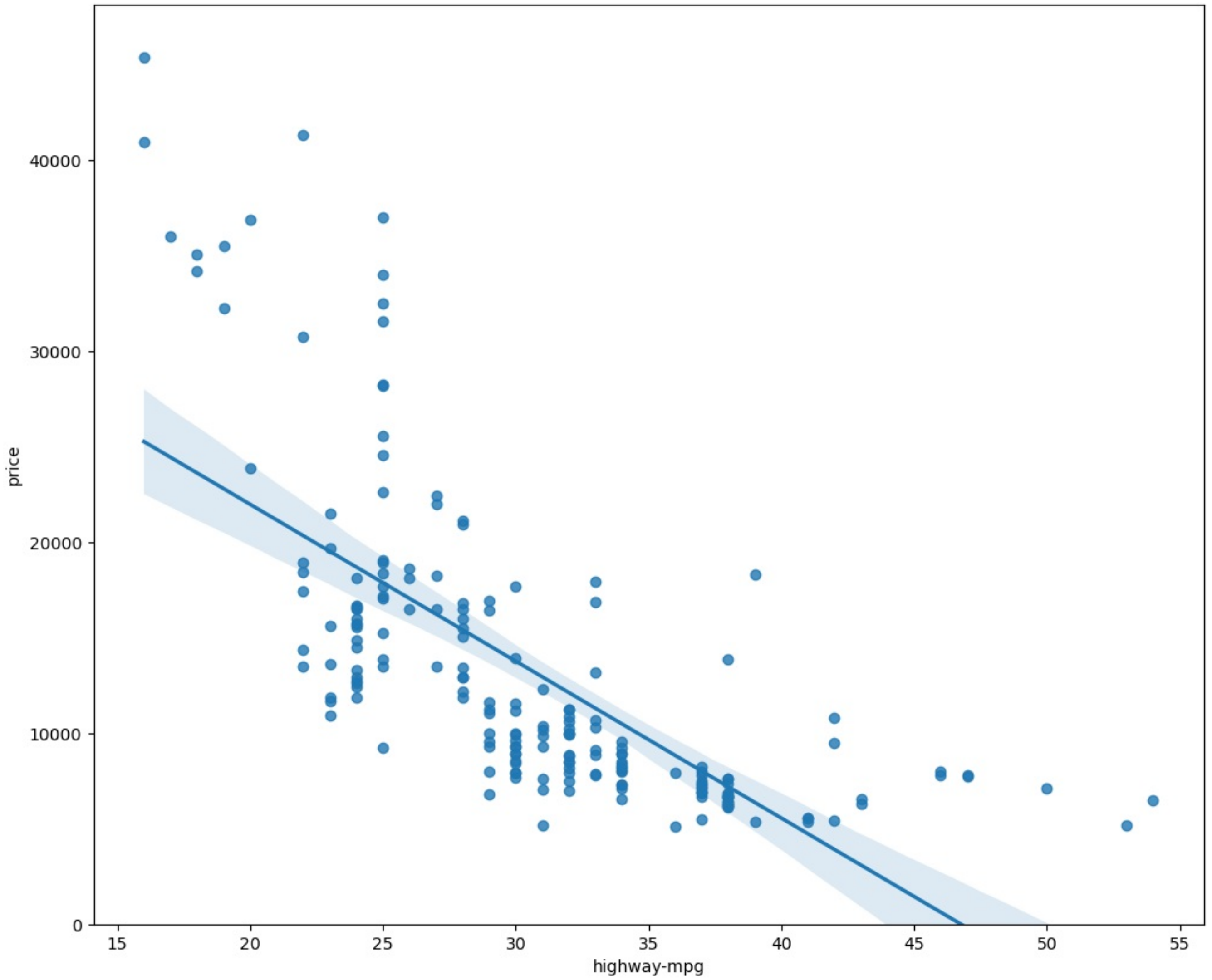
Out[44]:
```
38201.31327245728
```

In [45]:
```
Price = 38201.31327245728 +1.49789586normalized-losses-820.45434016highway-mpg
```

```
  File "C:\Users\hp\AppData\Local\Temp\ipykernel_11964\2134262372.py", line 1
    Price = 38201.31327245728 +1.49789586normalized-losses-820.45434016highway-mpg
                                        ^
SyntaxError: invalid syntax
```

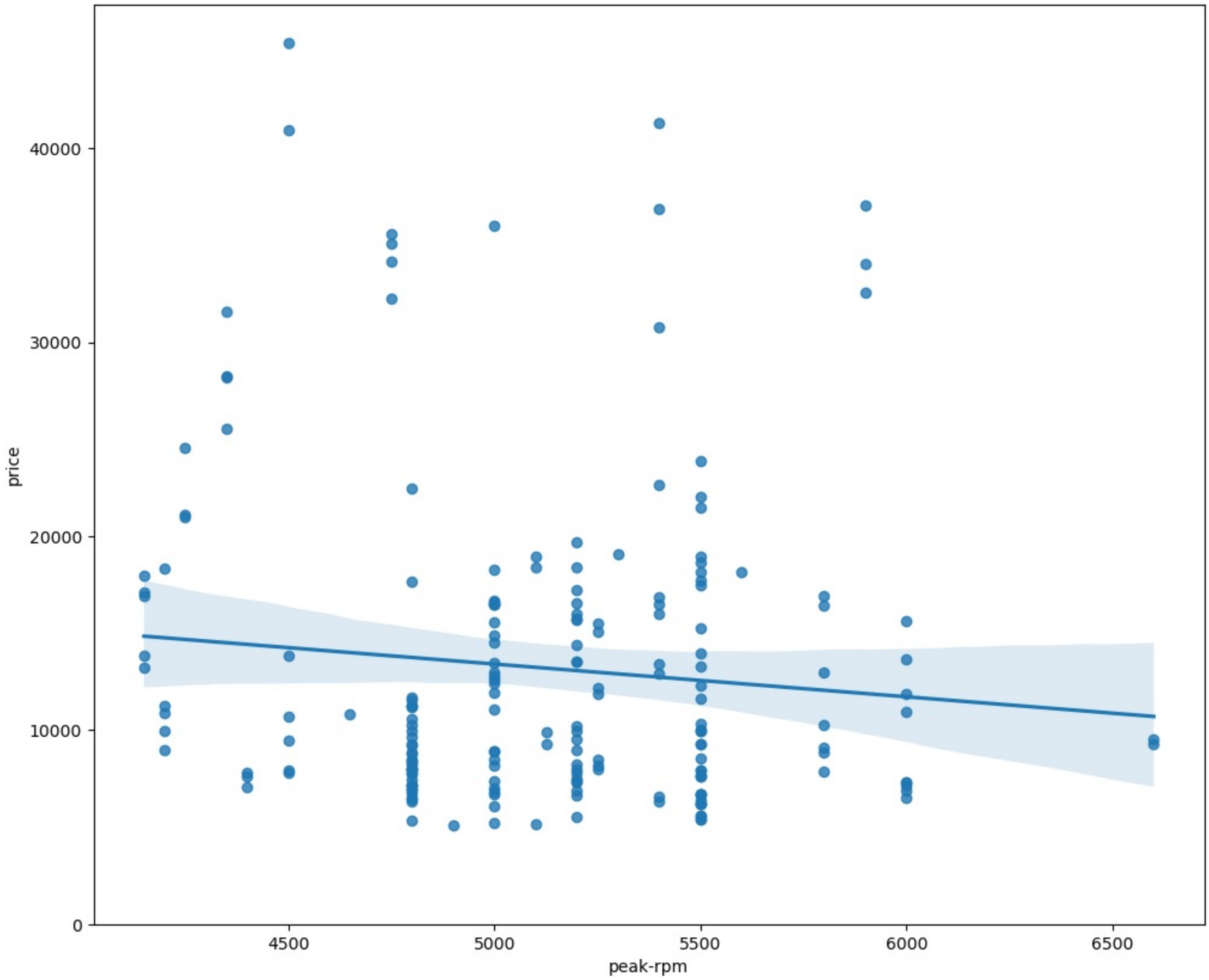
In [46]:
```
import seaborn as sns
width = 12
height = 10
plt.figure(figsize = (width,height))
sns.regplot(x = "highway-mpg",y ="price", data = df)
plt.ylim(0,)
```

Out[46]:
```
(0.0, 48179.76254365848)
```



In [47]:
```
plt.figure(figsize=(width, height))
sns.regplot(x="peak-rpm", y="price", data=df)
plt.ylim(0,)
```

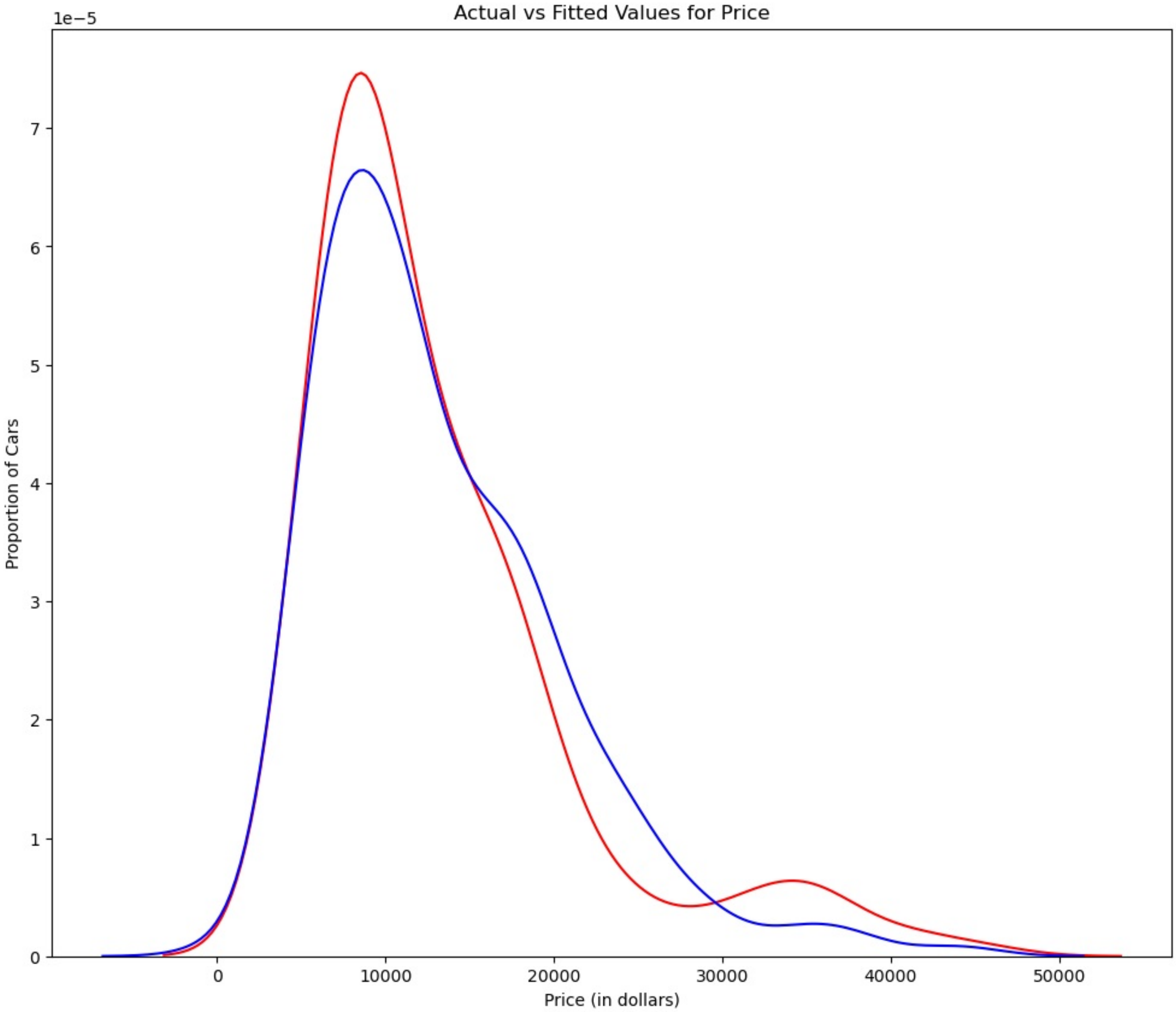
Out[47]:
```
(0.0, 47414.1)
```



In [48]:
```
Y_hat = lm.predict(Z)
```

In [49]:
```
plt.figure(figsize=(width, height))


ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")
sns.distplot(Y_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)


plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price (in dollars)')
plt.ylabel('Proportion of Cars')

plt.show()
plt.close()
```

```
C:\Users\hp\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecate
d function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\hp\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecate
d function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



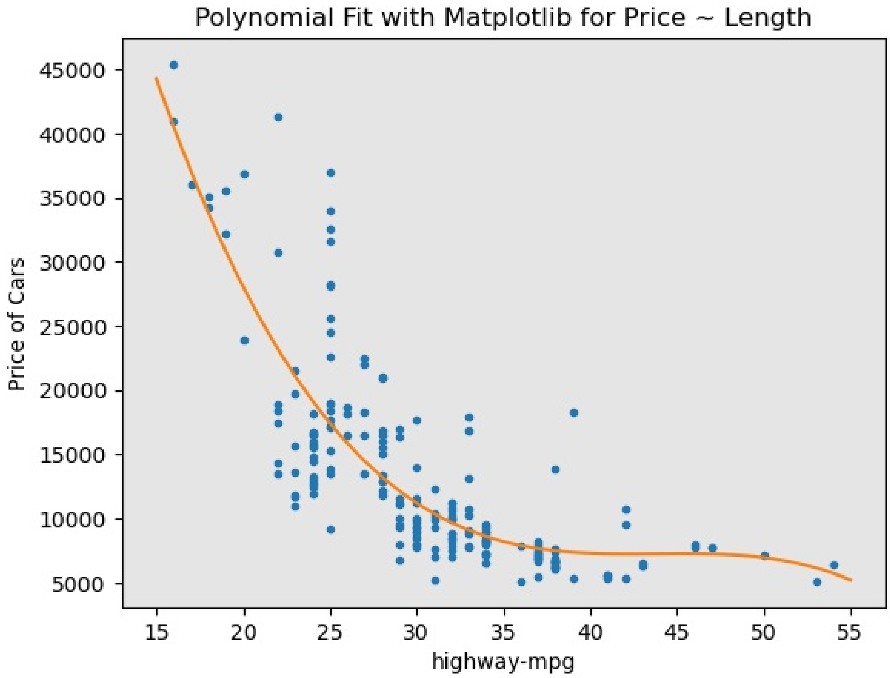
In [50]:
```
def PlotPolly(model,independent_variable,dependent_variabble,Name):
    x_new = np.linspace(15,55,100)
    y_new = model(x_new)

    plt.plot(independent_variable, dependent_variabble, '.', x_new, y_new, '-')
    plt.title ("Polynomial Fit with Matplotlib for Price ~ Length")
    ax = plt.gca()
    ax.set_facecolor((0.898, 0.898, 0.898))
    fig = plt.gcf()
    plt.xlabel(Name)
    plt.ylabel('Price of Cars')

    plt.show()
    plt.close()
```

In [51]:
```
x = df['highway-mpg']
y = df['price']
```

In [54]:
```
import numpy as np
f = np.polyfit(x, y, 3)
p = np.poly1d(f)
print(p)
```

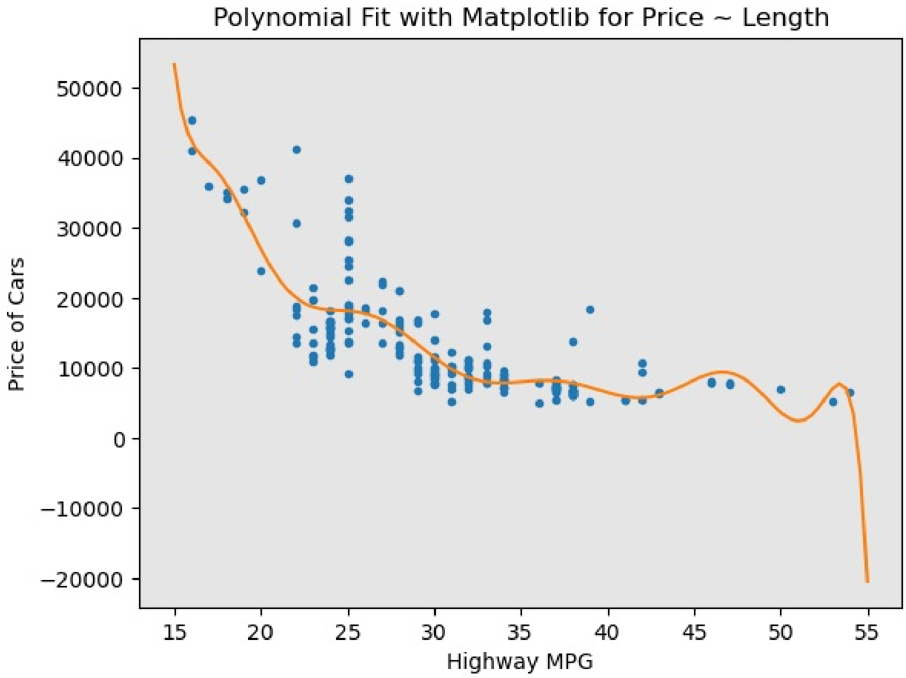
```
        3         2
-1.557 x + 204.8 x - 8965 x + 1.379e+05
```

In [55]:
```
PlotPolly(p, x, y, 'highway-mpg')
```



In [56]:
```
np.polyfit(x, y, 3)
```
Out[56]:
```
array([-1.55663829e+00,  2.04754306e+02, -8.96543312e+03,  1.37923594e+05])
```

In [57]:
```
from sklearn.preprocessing import PolynomialFeatures

f1 = np.polyfit(x, y, 11)
p1 = np.poly1d(f1)
print(p1)
PlotPolly(p1,x,y, 'Highway MPG')
```
```
             11             10              9           8         7
-1.243e-08 x  + 4.722e-06 x  - 0.0008028 x + 0.08056 x - 5.297 x
           6          5             4             3             2
 + 239.5 x - 7588 x + 1.684e+05 x - 2.565e+06 x + 2.551e+07 x - 1.491e+08 x + 3.879e+08
```



In [58]:
```
pr=PolynomialFeatures(degree=2)
pr
```
Out[58]:
```
PolynomialFeatures()
```

In [59]:
```
Z_pr=pr.fit_transform(Z)
```

In [60]:
```
Z.shape
```
Out[60]:
```
(201, 4)
```

In [61]:
```
Z_pr.shape
```

Out[61]: (201, 15)

In [62]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

In [63]:
```python
Input=[('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include_bias=False)), ('model',LinearRegre
```

In [64]:
```python
pipe=Pipeline(Input)
pipe
```

Out[64]:
```
Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures(include_bias=False)),
                ('model', LinearRegression())])
```

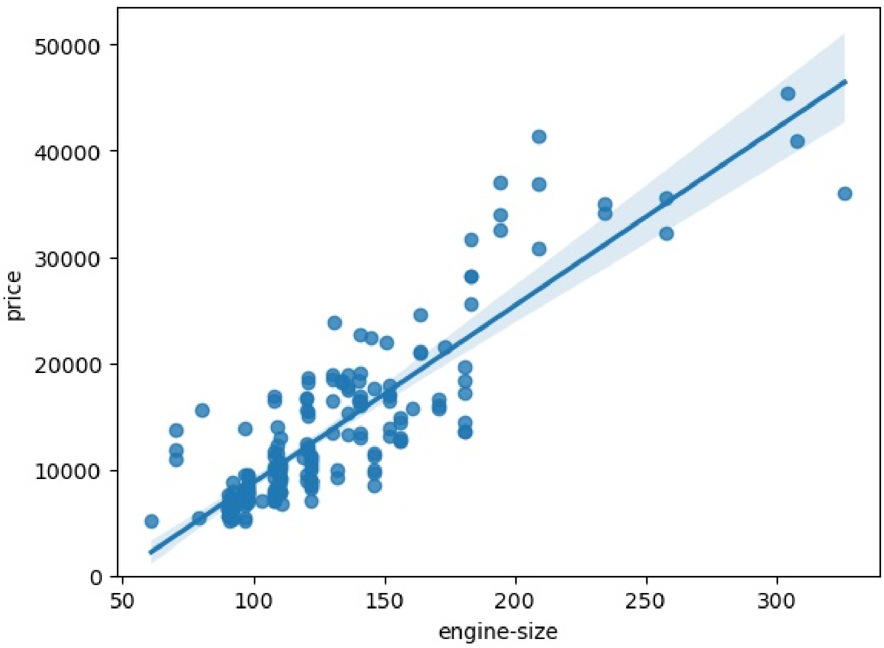
In [66]:
```python
df[['bore','stroke','compression-ratio','horsepower']].corr()
```

Out[66]:

	bore	stroke	compression-ratio	horsepower
bore	1.000000	-0.055390	0.001263	0.566936
stroke	-0.055390	1.000000	0.187923	0.098462
compression-ratio	0.001263	0.187923	1.000000	-0.214514
horsepower	0.566936	0.098462	-0.214514	1.000000

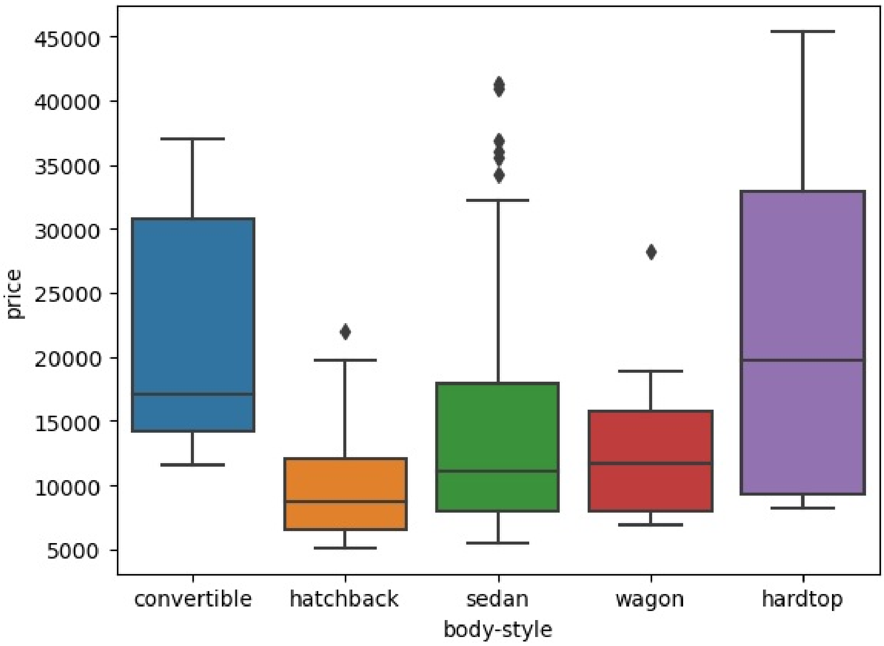
In [67]:
```python
sns.regplot(x="engine-size", y="price", data=df)
plt.ylim(0,)
```

Out[67]: (0.0, 53509.36151967784)



In [68]:
```python
sns.boxplot(x="body-style", y="price", data=df)
```

Out[68]: <AxesSubplot:xlabel='body-style', ylabel='price'>



In [69]:

```
df['drive-wheels'].value_counts().to_frame()
```

Out[69]:

	drive-wheels
fwd	118
rwd	75
4wd	8

In [70]:

```
drive_wheels_counts = df['drive-wheels'].value_counts().to_frame()
drive_wheels_counts.rename(columns={'drive-wheels': 'value_counts'}, inplace=True)
drive_wheels_counts
```

Out[70]:

	value_counts
fwd	118
rwd	75
4wd	8

In [72]:

```
engine_loc_counts = df['engine-location'].value_counts().to_frame()
engine_loc_counts.rename(columns={'engine-location': 'value_counts'}, inplace=True)
engine_loc_counts.index.name = 'engine-location'
engine_loc_counts.head()
```

Out[72]:

	value_counts
engine-location	
front	198
rear	3

In [73]:
```
df_group_one = df[['drive-wheels','body-style','price']]
```

In [74]:
```
# grouping results
df_group_one = df_group_one.groupby(['drive-wheels'],as_index=False).mean()
df_group_one
```

Out[74]:

	drive-wheels	price
0	4wd	10241.000000
1	fwd	9244.779661
2	rwd	19757.613333

In [ ]:
```
Use the "groupby" function to find the average "price" of each car based on "body-style".
```

In [76]:
```
df_group_one = df[['drive-wheels','body-style','price']]
```

In [77]:
```
df_group_one = df_group_one.groupby(['drive-wheels'],as_index=False).mean()
df_group_one
```

Out[77]:

	drive-wheels	price
0	4wd	10241.000000
1	fwd	9244.779661
2	rwd	19757.613333

In [78]:
```
df_gptest = df[['drive-wheels','body-style','price']]
grouped_test1 = df_gptest.groupby(['drive-wheels','body-style'],as_index=False).mean()
grouped_test1
```

Out[78]:

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

In [79]:
```
grouped_pivot = grouped_test1.pivot(index='drive-wheels',columns='body-style')
grouped_pivot
```

Out[79]:

					price
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	NaN	NaN	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

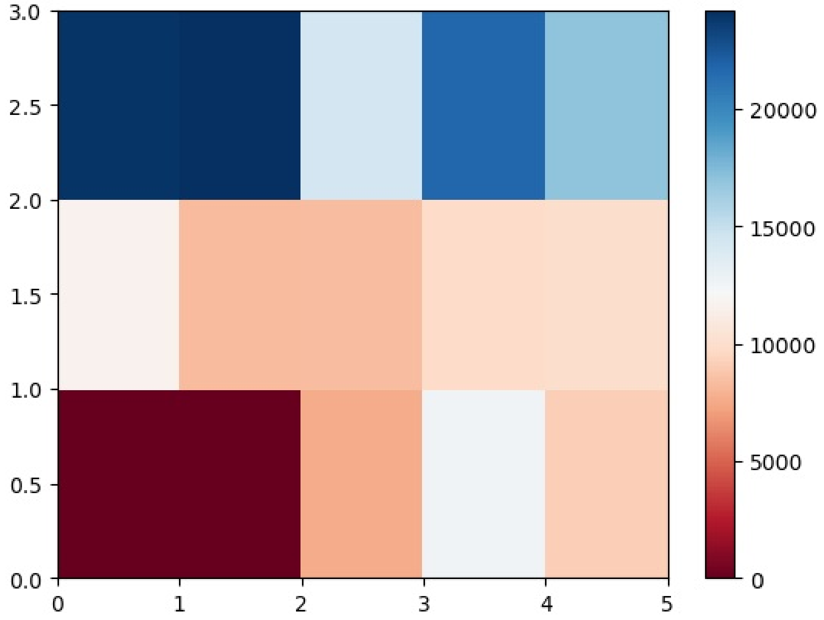
In [81]:
```
grouped_pivot = grouped_pivot.fillna(0) #fill missing values with 0
grouped_pivot
```

Out[81]:

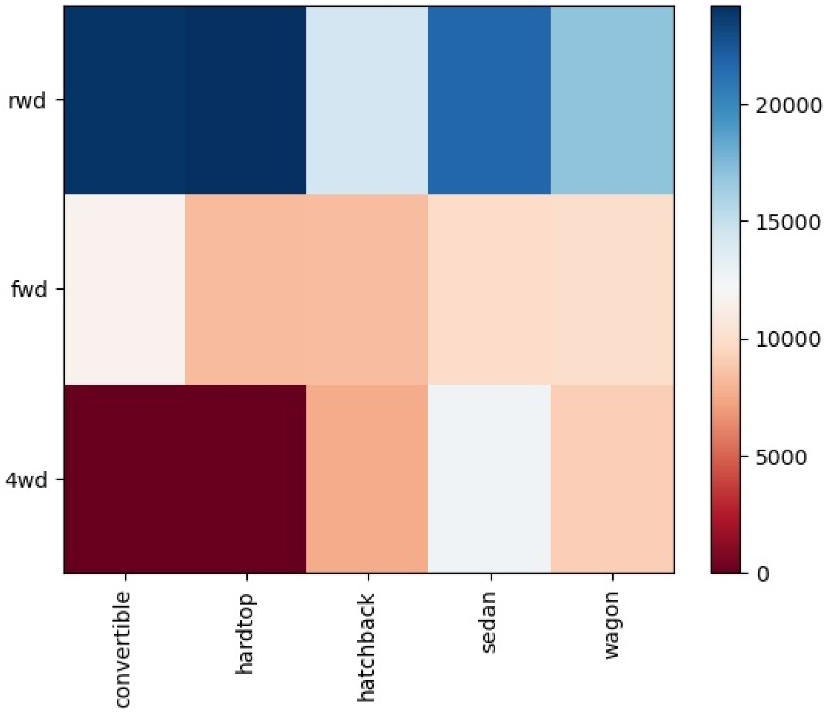
					price
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	0.0	0.000000	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

In [82]:
```
plt.pcolor(grouped_pivot, cmap='RdBu')
```

```
plt.colorbar()
plt.show()
```



In [83]:
```
fig, ax = plt.subplots()
im = ax.pcolor(grouped_pivot, cmap='RdBu')

#label names
row_labels = grouped_pivot.columns.levels[1]
col_labels = grouped_pivot.index

#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)

#insert labels
ax.set_xticklabels(row_labels, minor=False)
ax.set_yticklabels(col_labels, minor=False)

#rotate label if too long
plt.xticks(rotation=90)

fig.colorbar(im)
plt.show()
```



In [84]:
```
from scipy import stats
```

In [85]:
```
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 0.5846418222655081  with a P-value of P = 8.076488270732989e-20
```

In [86]:
```
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 0.809574567003656  with a P-value of P =  6.369057428259557e-48
```

In [87]:
```
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 0.7512653440522674  with a P-value of P = 9.200335510481516e-38
```

In [88]:
```
grouped_test2=df_gptest[['drive-wheels', 'price']].groupby(['drive-wheels'])
grouped_test2.head(2)
```

Out[88]:

	drive-wheels	price
0	rwd	13495.0
1	rwd	16500.0
3	fwd	13950.0
4	4wd	17450.0
5	fwd	15250.0
136	4wd	7603.0

In [91]:
```
df_gptest
```

Out[91]:

	drive-wheels	body-style	price
0	rwd	convertible	13495.0
1	rwd	convertible	16500.0
2	rwd	hatchback	16500.0
3	fwd	sedan	13950.0
4	4wd	sedan	17450.0
...	...	...	...
196	rwd	sedan	16845.0
197	rwd	sedan	19045.0
198	rwd	sedan	21485.0
199	rwd	sedan	22470.0
200	rwd	sedan	22625.0

201 rows × 3 columns

In [92]:
```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'], grouped_test2.get_group('rwd')['price'],

print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 67.95406500780399 , P = 3.3945443577151245e-23
```

In [93]:
```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'], grouped_test2.get_group('rwd')['price'])

print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 8.580681368924756 , P = 0.004411492211225333
```

In [94]:
```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'], grouped_test2.get_group('fwd')['price'])

print("ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 0.665465750252303 , P = 0.41620116697845666
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js