Data Analysis with Python

1. Problem statement What are the main characteristics which have the most impact on the car price ?

In [1]:
```python
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
path = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/automobileEDA.csv'
df = pd.read_csv(path)
df.head()
```

Out[2]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km	horsepower-binned	diesel	gas
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21	27	13495.0	11.190476	Medium	0	1
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21	27	16500.0	11.190476	Medium	0	1
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154.0	5000.0	19	26	16500.0	12.368421	Medium	0	1
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102.0	5500.0	24	30	13950.0	9.791667	Medium	0	1
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115.0	5500.0	18	22	17450.0	13.055556	Medium	0	1

5 rows × 29 columns

2.Analyzing Individual Feature Patterns using Visualization

In [6]:
```python
print(df.dtypes)
```

```
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 [7]:
```python
df.corr()
```

```
C:\Users\VictorM\AppData\Local\Temp\ipykernel_24488\1134722465.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid column
s or specify the value of numeric_only to silence this warning.
  df.corr()
```

Out[7]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km	diesel	gas
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.008245	-0.182196	0.075819	0.279740	-0.035527	0.036233	-0.082391	0.066171	-0.196735	0.196735
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029862	0.055563	-0.114713	0.217299	0.239543	-0.225016	-0.181877	0.133999	0.238567	-0.101546	0.101546
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.158502	0.250313	0.371147	-0.360305	-0.470606	-0.543304	0.584642	0.476153	0.307237	-0.307237
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608971	0.124139	0.159733	0.579821	-0.285970	-0.665192	-0.698142	0.690628	0.657373	0.211187	-0.211187
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544885	0.188829	0.189867	0.615077	-0.245800	-0.633531	-0.680635	0.751265	0.673363	0.244356	-0.244356
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180449	-0.062704	0.259737	-0.087027	-0.309974	-0.049800	-0.104812	0.135486	0.003811	0.281578	-0.281578
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072	0.644060	0.167562	0.156433	0.757976	-0.279361	-0.749543	-0.794889	0.834415	0.785353	0.221046	-0.221046
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572609	0.209523	0.028889	0.822676	-0.256733	-0.650546	-0.679571	0.872335	0.745059	0.070779	-0.070779
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.000000	-0.055390	0.001263	0.566936	-0.267392	-0.582027	-0.591309	0.543155	0.554610	0.054458	-0.054458
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209523	-0.055390	1.000000	0.187923	0.098462	-0.065713	-0.034696	-0.035201	0.082310	0.037300	0.241303	-0.241303
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187923	1.000000	-0.214514	-0.435780	0.331425	0.268465	0.071107	-0.299372	0.985231	-0.985231
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098462	-0.214514	1.000000	0.107885	-0.822214	-0.804575	0.809575	0.889488	-0.169053	0.169053
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.065713	-0.435780	0.107885	1.000000	-0.115413	-0.058598	-0.101616	0.115830	-0.475812	0.475812
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027	-0.034696	0.331425	-0.822214	-0.115413	1.000000	0.972044	-0.686571	-0.949713	0.265676	-0.265676
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309	-0.035201	0.268465	-0.804575	-0.058598	0.972044	1.000000	-0.704692	-0.930028	0.198690	-0.198690
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	0.071107	0.809575	-0.101616	-0.686571	-0.704692	1.000000	0.789898	0.110326	-0.110326
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	0.889488	0.115830	-0.949713	-0.930028	0.789898	1.000000	-0.241282	0.241282
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	0.241303	0.985231	-0.169053	-0.475812	0.265676	0.198690	0.110326	-0.241282	1.000000	-1.000000
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241303	-0.985231	0.169053	0.475812	-0.265676	-0.198690	-0.110326	0.241282	-1.000000	1.000000

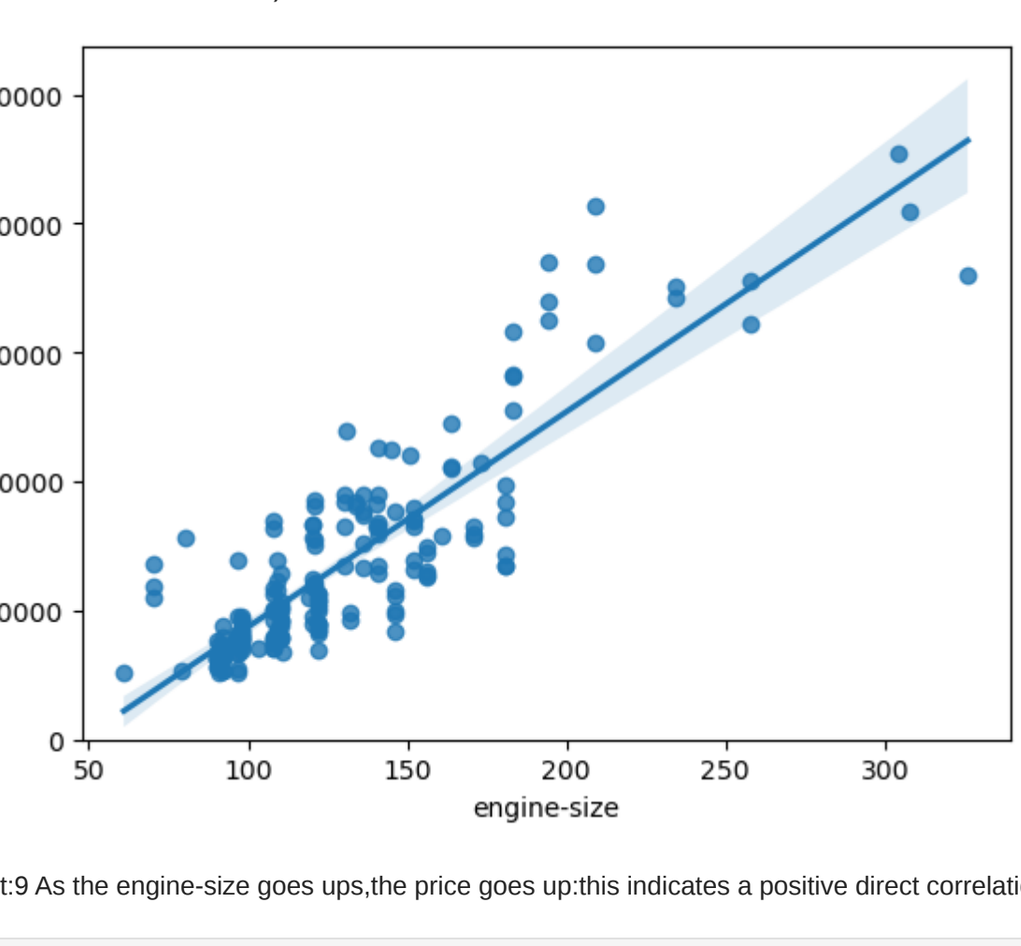
In [8]:
```python
# Correlation between  bore, stroke ,compression-ratio  and horsepower.
df[['bore','stroke','compression-ratio','horsepower']].corr()
```

Out[8]:

	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 [9]:
```python
#  Engine size as potential predictor  variable of price
sns.regplot(x="engine-size",y="price", data=df )
plt.ylim(0,)
```

Out[9]:
```
(0.0, 53697.61366949864)
```



output:9 As the engine-size goes ups,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.

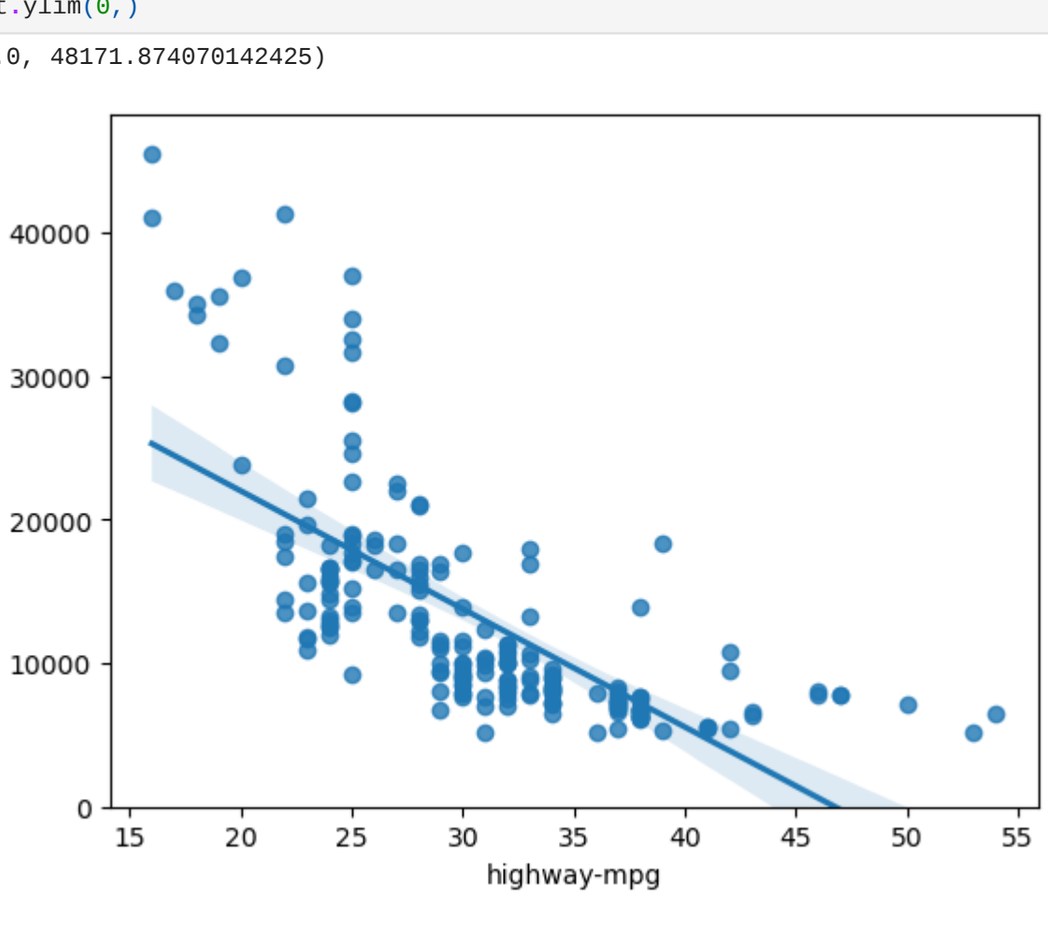
In [10]:
```python
# Calculating  the  correlation  between engine and price
df[["engine-size" , "price"]].corr()
```

Out[10]:

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

In [14]:
```python
# highway mpg is a potential predictor variable of price
sns.regplot(x="highway-mpg" , y="price" , data=df)
plt.ylim(0,)
```

Out[14]:
```
(0.0, 48171.874070142425)
```



Output 14: As the highway-mpg goes up ,the price goes down:this indicates an inverse/negative relationship between these two variables.Highway mpg could potentially be a predictor of price.

In [15]:
```python
# Correlation highway-mpg and price
df[["highway-mpg" , "price"]].corr()
```

Out[15]:

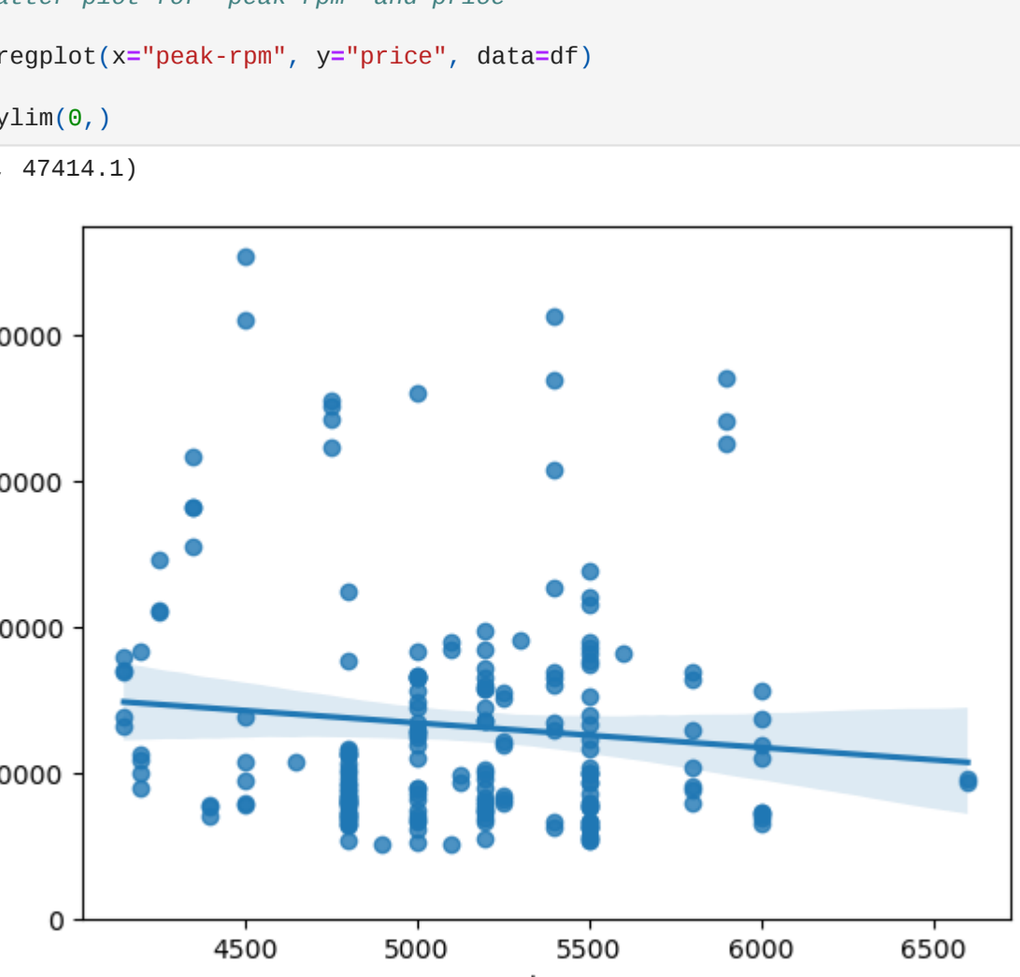
	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

In [19]:
```python
# Scatter plot for  peak-rpm  and price

sns.regplot(x="peak-rpm", y="price", data=df)

plt.ylim(0,)
```

Out[19]:
```
(0.0, 47414.1)
```



Output 19: 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.There its it is not a reliable variable.

In [20]:
```python
# Calculating the Correlation peak-rpm and price
df[['peak-rpm','price']].corr()
```

Out[20]:

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

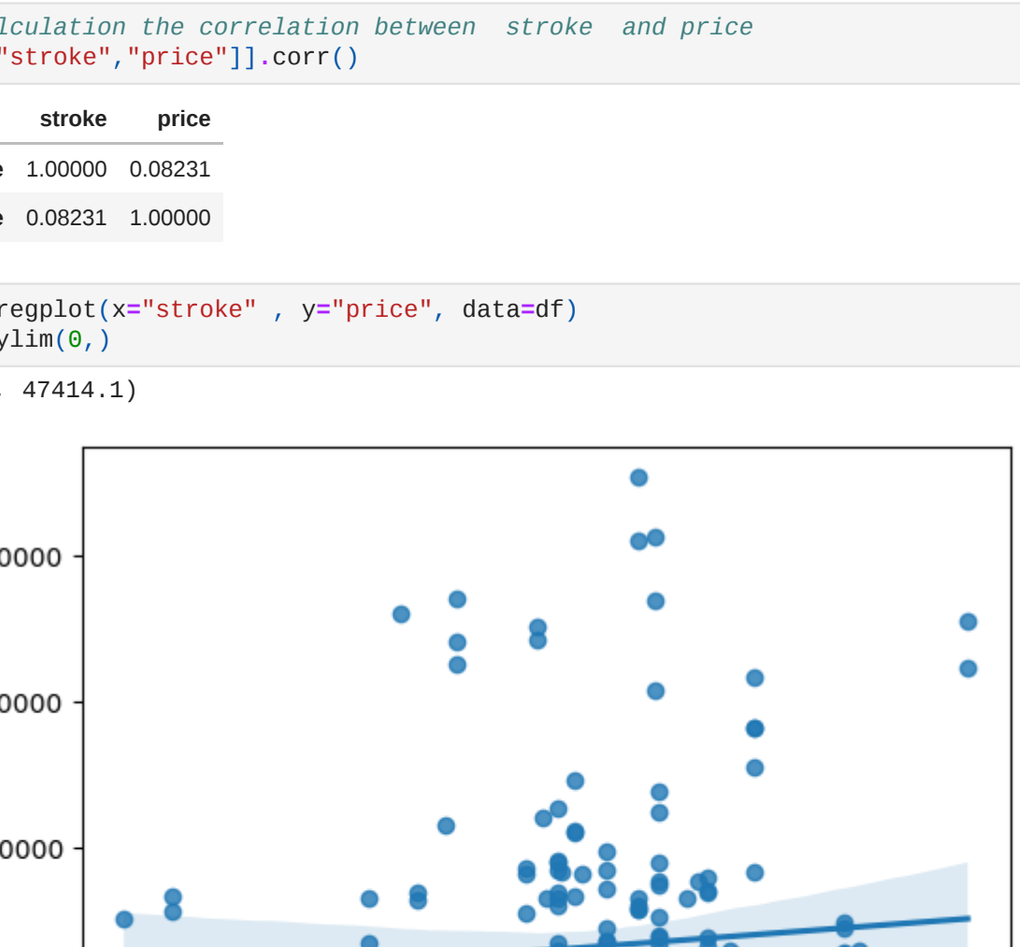
In [21]:
```python
# Calculation the correlation between  stroke  and price
df[["stroke","price"]].corr()
```

Out[21]:

	stroke	price
stroke	1.00000	0.08231
price	0.08231	1.00000

In [23]:
```python
sns.regplot(x="stroke" , y="price", data=df)
plt.ylim(0,)
```

Out[23]:
```
(0.0, 47414.1)
```



Output23: Stroke is not good predictor because the linear regression line it looks more horizontal.Also scatter plots are far from the line.

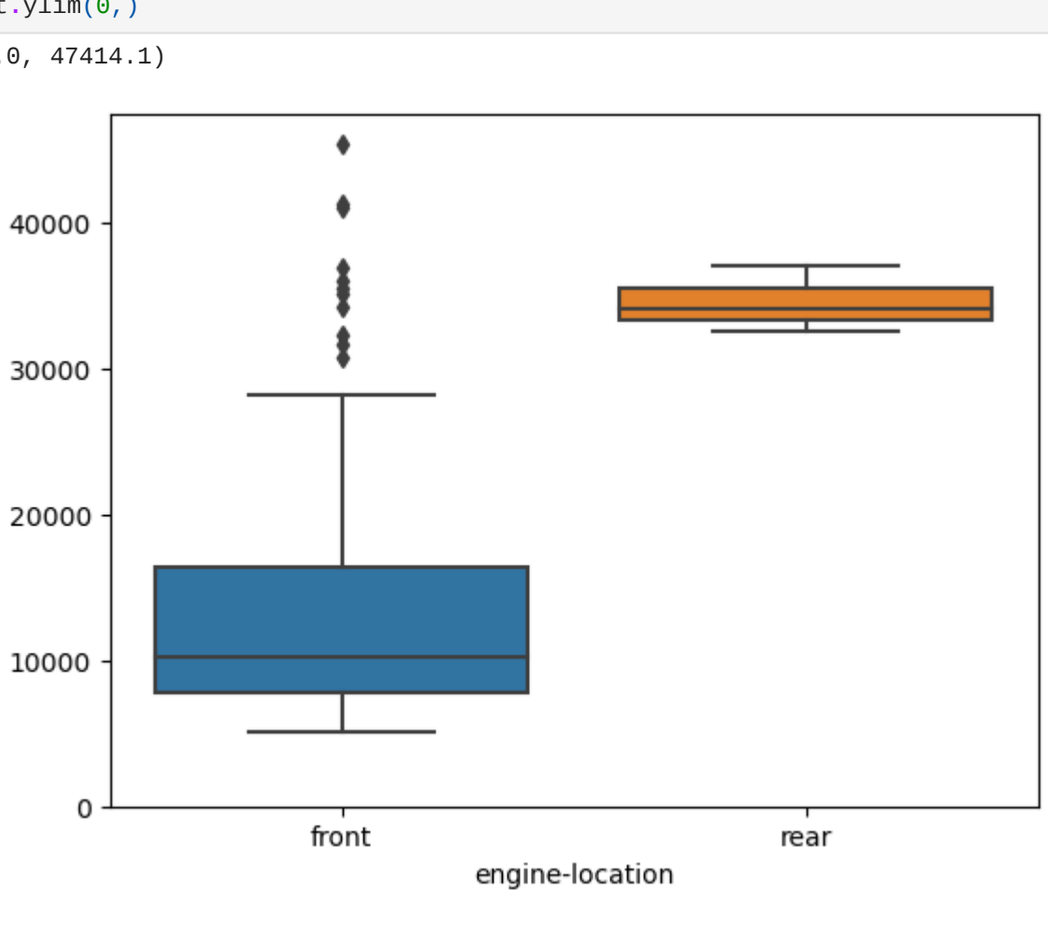
In [7]:
```python
# Relationship  between body-style and price
sns.boxplot(x="body-style", y="price", data=df)
plt.ylim(0,)
```

Out[7]:
```
(0.0, 47414.1)
```

Output:4: It shows body_style categories have a significant overlap, and so body-style would not be a good predictor of price .

In [6]:
```python
# Relationship between engine  location  and price
sns.boxplot(x="engine-location",y="price", data=df)
plt.ylim(0,)
```

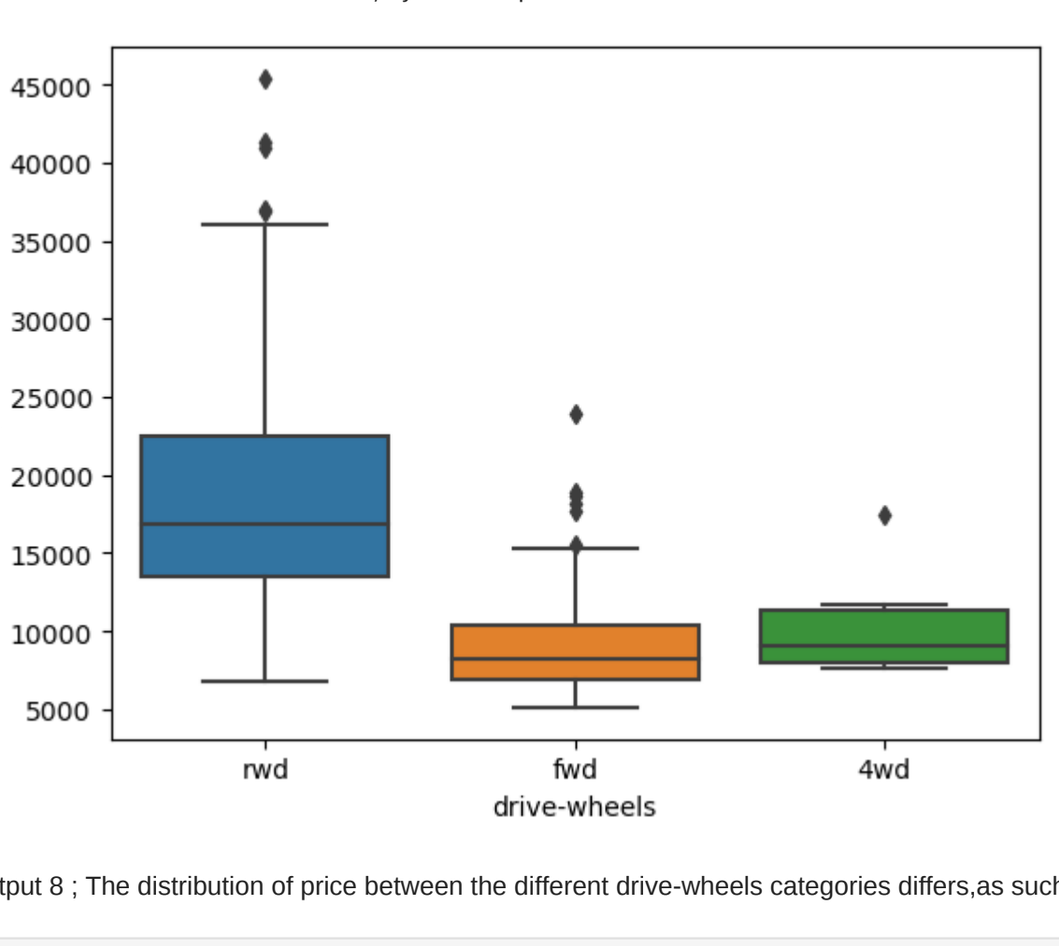
Out[6]:
```
(0.0, 47414.1)
```



Output 6: 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 [8]:
```python
#drive- wheels
sns.boxplot(x="drive-wheels" , y="price" , data=df)
```

Out[8]:
```
<Axes: xlabel='drive-wheels', ylabel='price'>
```



Output 8 ; The distribution of price between the different drive-wheels categories differs,as such drive-wheels could potentially be predictor of price.

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

Out[4]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km	diesel	gas
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000
mean	0.840796	122.00000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.256904	10.164279	103.405534	5117.665368	25.179104	30.686567	13207.129353	9.944145	0.099502	0.900498
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.296727	41.546834	0.268072	0.319256	4.004965	37.365700	478.113805	6.423220	6.815150	7947.066342	2.534599	0.300083	0.300083
min	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	4150.000000	13.000000	16.000000	5118.000000	4.795918	0.000000	0.000000
25%	0.000000	101.00000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.110000	8.600000	70.000000	4800.000000	19.000000	25.000000	7775.000000	7.833333	0.000000	1.000000
50%	1.000000	122.00000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5125.369458	24.000000	30.000000	10295.000000	9.791667	0.000000	1.000000
75%	2.000000	137.00000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.410000	9.400000	116.000000	5500.000000	30.000000	34.000000	16500.000000	12.368421	0.000000	1.000000
max	3.000000	256.00000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	262.000000	6600.000000	49.000000	54.000000	45400.000000	18.076923	1.000000	1.000000

In [5]:
```python
df.describe(include=['object'])
```

Out[5]:

	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 [6]:
```python
# calculating value counts  for drive wheels
df['drive-wheels'].value_counts()
```

Out[6]:
```
fwd    118
rwd     75
4wd      8
Name: drive-wheels, dtype: int64
```

In [7]:
```python
# converting  the series to a dataframe
df['drive-wheels'].value_counts().to_frame()
```

Out[7]:

	drive-wheels
fwd	118
rwd	75
4wd	8

In [8]:
```python
# Renaming drive_wheels to value_counts
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[8]:

	value_counts
fwd	118
rwd	75
4wd	8

In [9]:
```python
#Renaming -location as variable
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(10)
```

Out[9]:

	value_counts
engine-location	
front	198
rear	3

In [10]:
```python
# Grouping by variable wheels
df['drive-wheels'].unique()
```

Out[10]:
```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

In [11]:
```python
# creating  df_group_one
df_group_one = df[['drive-wheels', 'body-style' , 'price']]
```

In [12]:
```python
#grouping results
df_group_one = df_group_one.groupby(['drive-wheels'], as_index=False).mean()
df_group_one
```

```
C:\Users\VictorM\AppData\Local\Temp\ipykernel_30416\3483700661.py:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the function.
  df_group_one = df_group_one.groupby(['drive-wheels'], as_index=False).mean()
```

Out[12]:

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

Out 18: From the data ,it seems rear-wheel drive vehicles are,on average,the most expensive,while 4-wheel and front-wheel are approximately the same in price.

In [13]:
```python
#Grouping the results
df_gptest = df[['drive-wheels','body-style' , 'price']]
grouped_test1 =df_gptest.groupby(['drive-wheels' , 'body-style'], as_index=False).mean()
grouped_test1
```

Out[13]:

	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

This grouped data is much easier to visualize when it is made into a pivot table.We can conver the dataframe to apivot table using the method "pivot" to create a pivot table from the groups.

In [14]:
```python
# Creating  a pivot table.
grouped_pivot =grouped_test1.pivot(index='drive-wheels',columns = 'body-style')
grouped_pivot
```

Out[14]:

	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 [18]:
```python
grouped_pivot = grouped_pivot.fillna(0)
grouped_pivot
```

Out[18]:

	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 [20]:
```python
df_gptest2 = df[['body-style','price']]
grouped_test_bodystyle =df_gptest2.groupby(['body-style'],as_index=False).mean()
grouped_test_bodystyle
```

Out[20]:

	body-style	price
0	convertible	21890.500000
1	hardtop	22208.500000
2	hatchback	9957.441176
3	sedan	14459.755319
4	wagon	12371.960000

In [24]:
```python
import matplotlib.pyplot as plt
```

In [25]:
```python
# Relationship  between Body Style vs Price
plt.pcolor(grouped_pivot, cmap='Rdbu')
plt.colorbar()
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[25], line 2
      1 # Relationship  between Body Style vs Price
----> 2 plt.pcolor(grouped_pivot, cmap='Rdbu')
      3 plt.colorbar()
      4 plt.show()

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\pyplot.py:2732, in pcolor(shading, alpha, norm, cmap, vmin, vmax, data, *args, **kwargs)
   2728 @_copy_docstring_and_deprecators(Axes.pcolor)
   2729 def pcolor(
   2730         *args, shading=None, alpha=None, norm=None, cmap=None,
   2731         vmin=None, vmax=None, data=None, **kwargs):
-> 2732     __ret = gca().pcolor(
   2733         *args, shading=shading, alpha=alpha, norm=norm, cmap=cmap,
   2734         vmin=vmin, vmax=vmax,
   2735         **({"data": data} if data is not None else {}), **kwargs)
   2736     sci(__ret)
   2737     return __ret

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\__init__.py:1442, in _preprocess_data.<locals>.inner(ax, data, *args, **kwargs)
   1439 @functools.wraps(func)
   1440 def inner(ax, *args, data=None, **kwargs):
   1441     if data is None:
-> 1442         return func(ax, *map(sanitize_sequence, args), **kwargs)
   1444     bound = new_sig.bind(ax, *args, **kwargs)
   1445     auto_label = (bound.arguments.get(label_namer)
   1446                   or bound.kwargs.get(label_namer))

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\axes\_axes.py:5997, in Axes.pcolor(self, shading, alpha, norm, cmap, vmin, vmax, *args, **kwargs)
   5993     kwargs['antialiaseds'] = False
   5995 kwargs.setdefault('snap', False)
-> 5997 collection = mcoll.PolyCollection(
   5998     verts, array=C, cmap=cmap, norm=norm, alpha=alpha, **kwargs)
   5999 collection._scale_norm(norm, vmin, vmax)
   6001 x = X.compressed()

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\_api\deprecation.py:454, in make_keyword_only.<locals>.wrapper(*args, **kwargs)
    448 if len(args) > name_idx:
    449     warn_deprecated(
    450         since, message="Passing the %(name)s %(obj_type)s "
    451         "positionally is deprecated since Matplotlib %(since)s; the "
    452         "parameter will become keyword-only %(removal)s.",
    453         name=name, obj_type=f"parameter of {func.__name__}()")
--> 454 return func(*args, **kwargs)

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\collections.py:1170, in PolyCollection.__init__(self, verts, sizes, closed, **kwargs)
   1149 @_api.make_keyword_only("3.6", name="closed")
   1150 def __init__(self, verts, sizes=None, closed=True, **kwargs):
   1151     """
   1152     Parameters
   1153     ----------
   (...)
   1168         Forwarded to `.Collection`.
   1169     """
-> 1170     super().__init__(**kwargs)
   1171     self.set_sizes(sizes)
   1172     self.set_verts(verts, closed)

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\_api\deprecation.py:454, in make_keyword_only.<locals>.wrapper(*args, **kwargs)
    448 if len(args) > name_idx:
    449     warn_deprecated(
    450         since, message="Passing the %(name)s %(obj_type)s "
    451         "positionally is deprecated since Matplotlib %(since)s; the "
    452         "parameter will become keyword-only %(removal)s.",
    453         name=name, obj_type=f"parameter of {func.__name__}()")
--> 454 return func(*args, **kwargs)

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\collections.py:157, in Collection.__init__(self, edgecolors, facecolors, linewidths, linestyles, capstyle, joinstyle, antialiaseds, offsets, offset_transform, nor
m, cmap, pickradius, hatch, urls, zorder, **kwargs)
     97 """
     98 Parameters
     99 ----------
   (...)
    154     :doc:`/gallery/misc/zorder_demo` for all defaults and examples.
    155 """
    156 artist.Artist.__init__(self)
--> 157 cm.ScalarMappable.__init__(self, norm, cmap)
    158 # list of un-scaled dash patterns
    159 # this is needed scaling the dash pattern by linewidth
    160 self._us_linestyles = [(0, None)]

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\cm.py:400, in ScalarMappable.__init__(self, norm, cmap)
    398 self.set_norm(norm)  # The Normalize instance of this ScalarMappable.
    399 self.cmap = None  # So that the setter knows we're initializing.
--> 400 self.set_cmap(cmap)  # The Colormap instance of this ScalarMappable.
    401 #: The last colorbar associated with this ScalarMappable. May be None.
    402 self.colorbar = None

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\cm.py:585, in ScalarMappable.set_cmap(self, cmap)
    576 """
    577 Set the colormap for luminance data.
    578 
   (...)
    581 cmap : `.Colormap` or str or None
    582 """
    583 in_init = self.cmap is None
--> 585 self.cmap = _ensure_cmap(cmap)
    586 if not in_init:
    587     self.changed()

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\cm.py:723, in _ensure_cmap(cmap)
    720 cmap_name = cmap if cmap is not None else mpl.rcParams["image.cmap"]
    721 # use check_in_list to ensure type stability of the exception raised by
    722 # the internal usage of this (ValueError vs KeyError)
--> 723 _api.check_in_list(sorted(_colormaps), cmap=cmap_name)
    724 return mpl.colormaps[cmap_name]

File ~\AppData\Local\anaconda\lib\site-packages\matplotlib\_api\__init__.py:131, in check_in_list(_values, _print_supported_values, **kwargs)
    129 if _print_supported_values:
    130     msg += f"; supported values are {', '.join(map(repr, values))}"
--> 131 raise ValueError(msg)

ValueError: 'Rdbu' is not a valid value for cmap; supported values are 'Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu
_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_
r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_
r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'b
one', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'crest', 'crest_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare', 'flare_
r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gnuplot',
'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv', 'hsv_r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako', 'mako_r', 'nipy_spectral', 'nipy_spectral
_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'rocket', 'rocket_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'ta
b20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winte
r', 'winter_r'
```

In [26]:
```python
from scipy import stats
```

In [28]:
```python
# Calculating the Pearson Correlation Coefficient  and P-value of 'wheel-base'
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
```

Conclusion:p-value is <0.001 , the correlation between wheel -base and price is statistically significant ,although the linear relationship isn"t extremely strong (~0.586)

In [ ]: