Exploratory Data Analysis

Objectives

- Explore features or characteristics to predict price of car
- Analyze patterns and run descriptive statistical analysis
- Group data based on identified parameters and create pivot tables
- Identify the effect of independent attributes on price of cars

Table of Contents

What are the main characteristics that have the most impact on the car price?

1. Import Data

Import libraries:

In [ ]:
```
#install specific version of libraries used in lab
#! mamba install pandas==1.3.3
#! mamba install numpy=1.21.2
#! mamba install scipy=1.7.1-y
#!  mamba install seaborn=0.9.0-y
```

In [49]:
```
import pandas as pd
import numpy as np
```

Load the data and store it in dataframe `df` :

This dataset was hosted on IBM Cloud object.

In [50]:
```
path='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-
df = pd.read_csv(path)
df.head()
```

Out[50]:

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

5 rows × 29 columns

2. Analyzing Individual Feature Patterns Using Visualization

Import visualization packages "Matplotlib" and "Seaborn". We don't forget about "%matplotlib inline" to plot in a Jupyter notebook.

In [51]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

How to choose the right visualization method?

When visualizing individual variables, it is important to first understand what type of variable we are dealing with. This will help us find the right visualization method for that variable.

In [52]:
```
# list the data types for each column
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
```

What is the data type of the column "peak-rpm"?

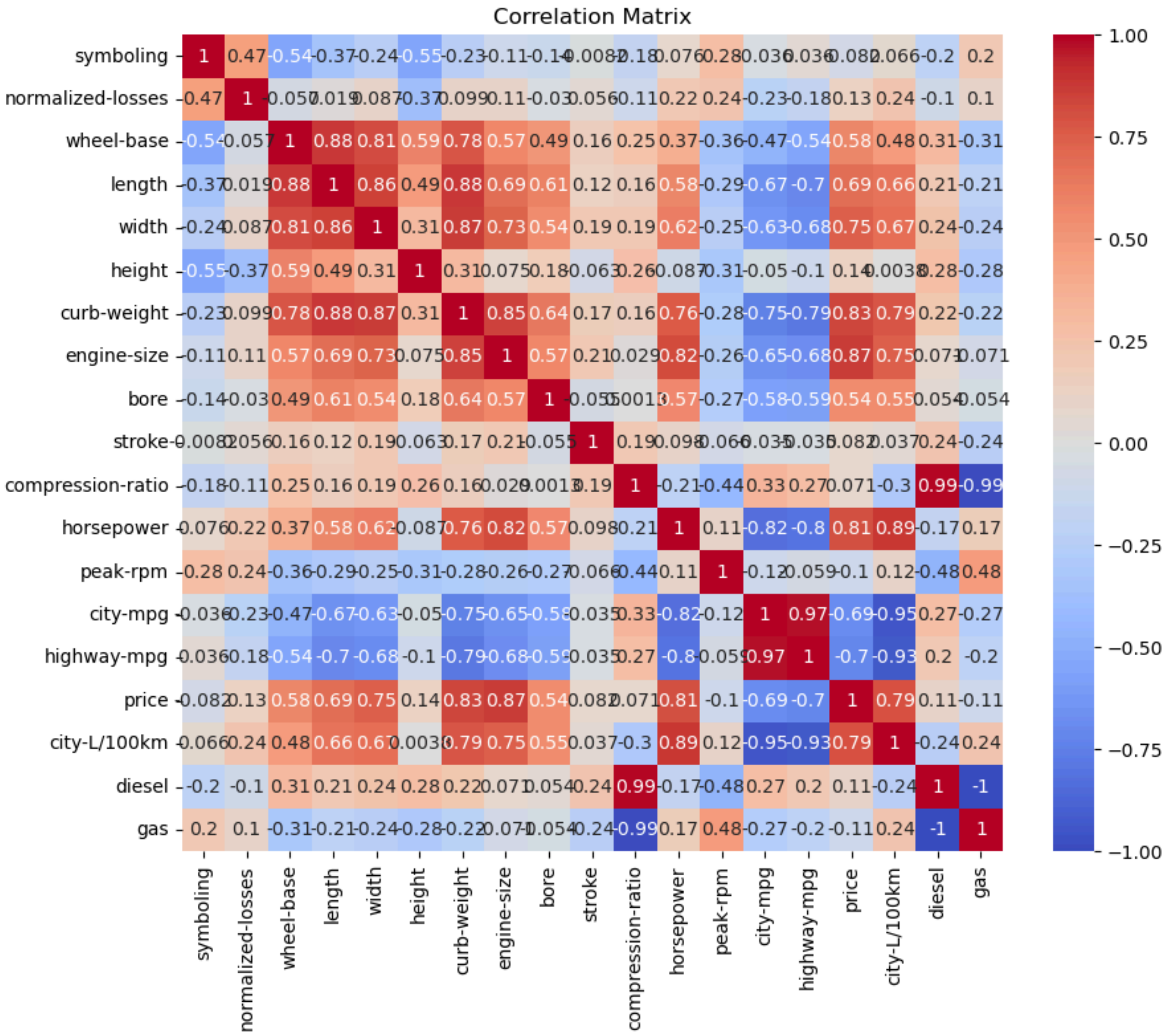
In [53]:
```
df['peak-rpm'].dtypes
```

Out[53]:
```
dtype('float64')
```

For example, we can calculate the correlation between variables of type "int64" or "float64" using the method "corr":

In [54]:
```
#Filter Numeric Columns: Use select_dtypes() to select only numeric columns
numeric_df = df.select_dtypes(include=['number'])
# correlation coef for numeric data
correlation_matrix = numeric_df.corr()
```

In [20]:
```
#Visualize the Correlation Matrix: Use libraries like matplotlib or seaborn to create a heatmap for
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



The diagonal elements are always one; we will study correlation more precisely Pearson correlation in-depth at the end of the notebook.

Question 2: Find the correlation between the following columns: bore, stroke, compression-ratio, and horsepower.

In [55]:
```
df[['bore','stroke','compression-ratio','horsepower']].corr()
```

Out[55]:

	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

Continuous Numerical Variables:

Continuous numerical variables are variables that may contain any value within some range. They can be of type "int64" or "float64". A great way to visualize these variables is by using scatterplots with fitted lines.

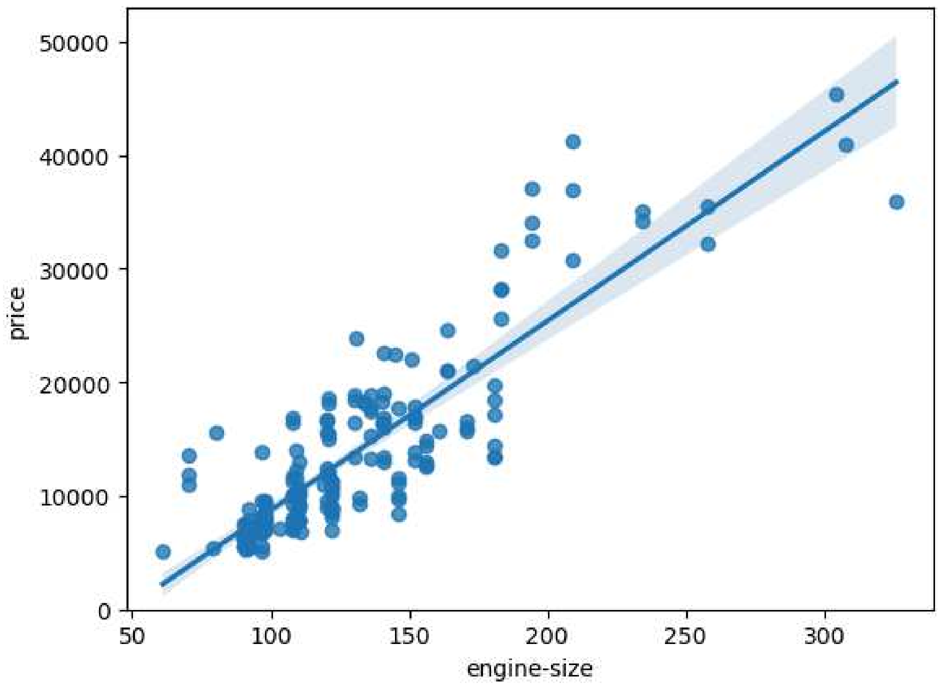
In order to start understanding the (linear) relationship between an individual variable and the price, we can use "regplot" which plots the scatterplot plus the fitted regression line for the data. This will be useful later on for visualizing the fit of the simple linear regression model as well.

Positive Linear Relationship

Let's find the scatterplot of "engine-size" and "price".

In [56]:
```
# Engine size as potential predictor variable of price
sns.regplot(x="engine-size", y="price", data=df)
plt.ylim(0,)
```

Out[56]: (0.0, 52989.8590520188)



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.

We can examine the correlation between 'engine-size' and 'price' and see that it's approximately 0.87.

In [57]:
```
df[["engine-size", "price"]].corr()
```

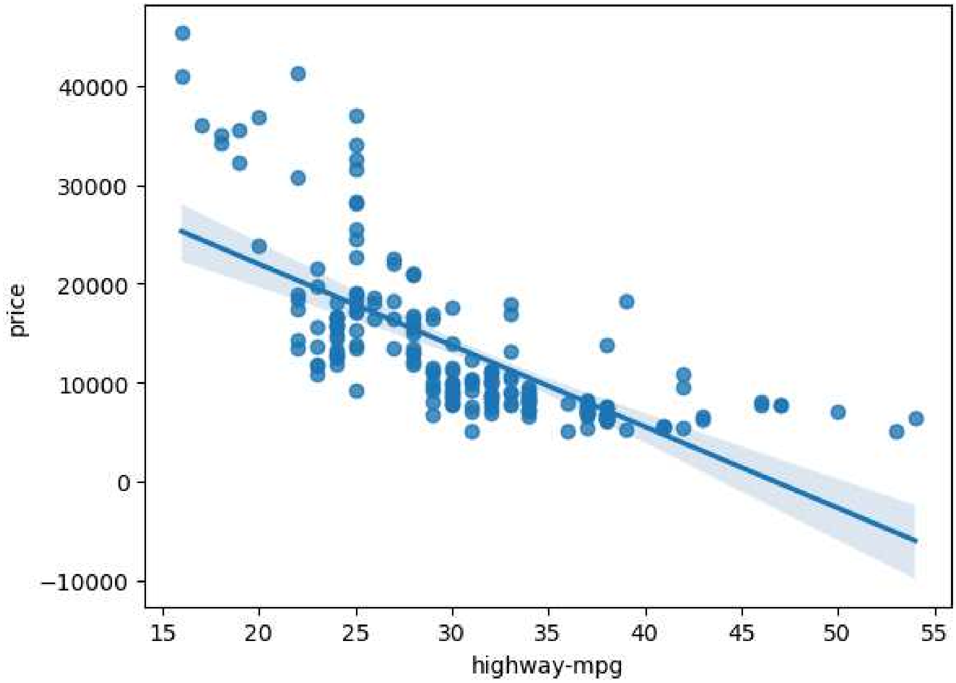
Out[57]:

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

Highway mpg is a potential predictor variable of price. Let's find the scatterplot of "highway-mpg" and "price".

In [58]:
```
sns.regplot(x="highway-mpg", y="price", data=df)
```

Out[58]: <Axes: xlabel='highway-mpg', ylabel='price'>



As 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.

We can examine the correlation between 'highway-mpg' and 'price' and see it's approximately -0.704.

In [59]:
```
df[['highway-mpg', 'price']].corr()
```

Out[59]:

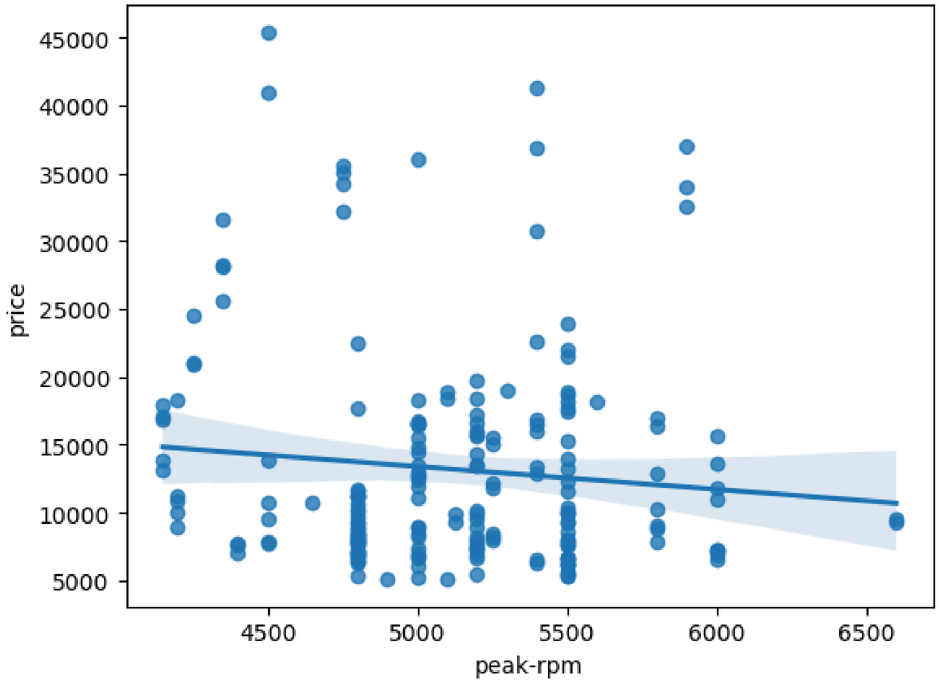
	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

Weak Linear Relationship

Let's see if "peak-rpm" is a predictor variable of "price".

In [60]:
```
sns.regplot(x="peak-rpm", y="price", data=df)
```

Out[60]: <Axes: 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.

We can examine the correlation between 'peak-rpm' and 'price' and see it's approximately -0.101616.

In [61]:
```
df[['peak-rpm','price']].corr()
```

Out[61]:

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

Find the correlation between x="stroke" and y="price".

In [29]:
```
df[["stroke","price"]].corr()
```

Out[29]:

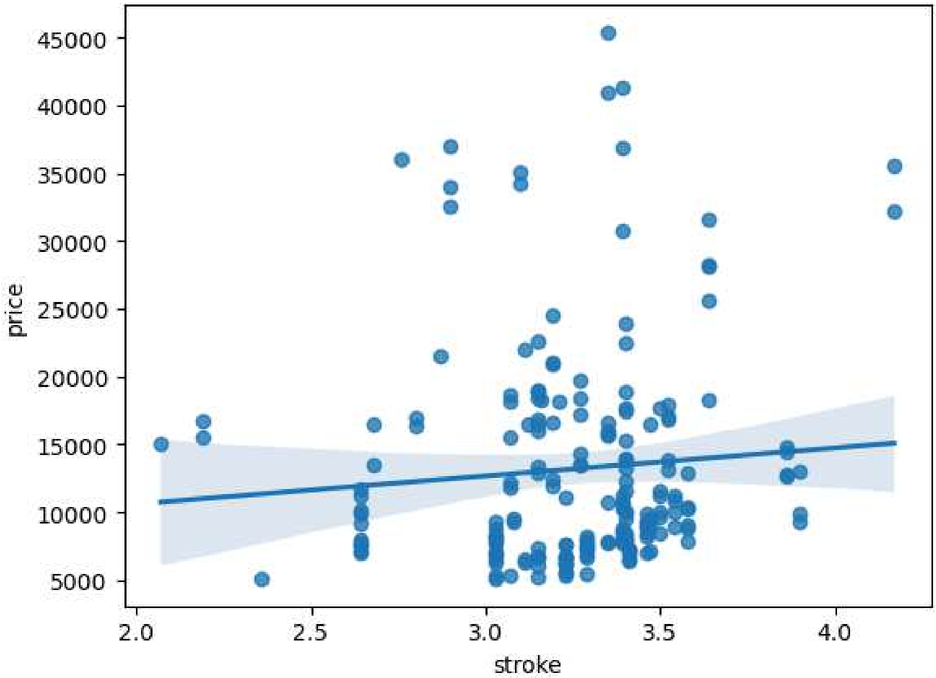
	stroke	price
stroke	1.00000	0.08231
price	0.08231	1.00000

```
<div class="alert alert-danger alertdanger" style="margin-top: 20px">

<p>Given the correlation results between "price" and "stroke", do you expect a linear
relationship?</p>
<p>Verify your results using the function "regplot()".</p>
</div>
```

In [62]:

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

sns.regplot(x="stroke", y="price", data=df)
```

Out[62]: <Axes: xlabel='stroke', ylabel='price'>



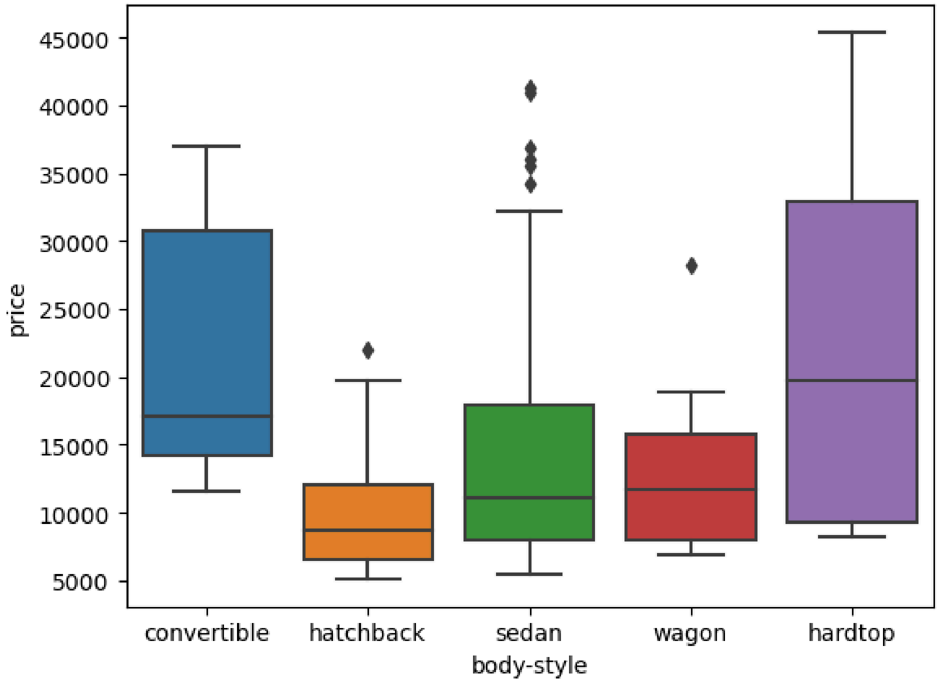
Categorical Variables

These are variables that describe a 'characteristic' of a data unit, and are selected from a small group of categories. The categorical variables can have the type "object" or "int64". A good way to visualize categorical variables is by using boxplots.

Let's look at the relationship between "body-style" and "price".

In [63]:
```
sns.boxplot(x="body-style", y="price", data=df)
```

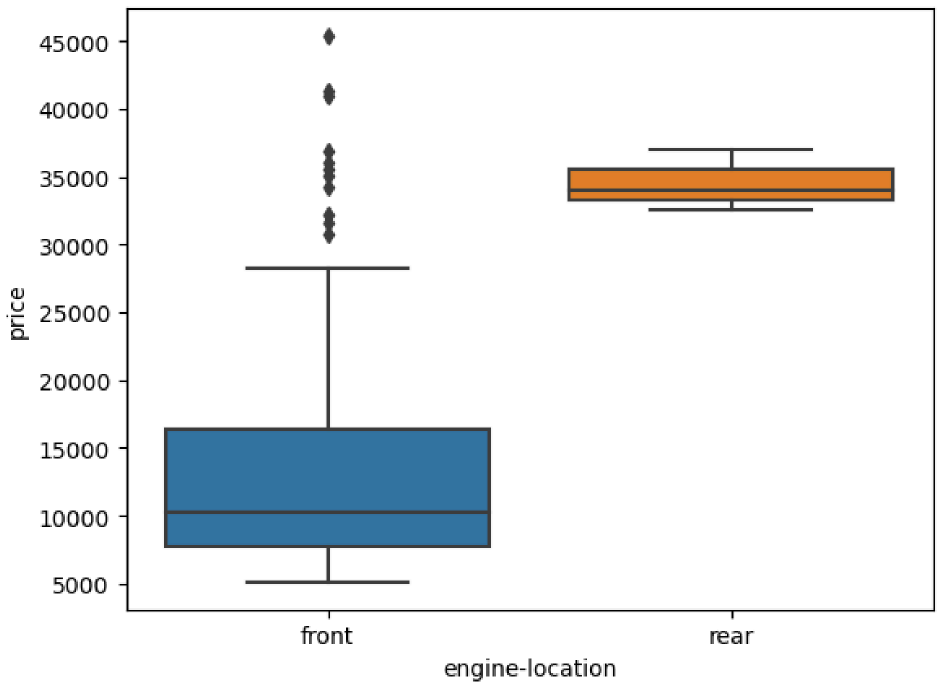
Out[63]: <Axes: xlabel='body-style', ylabel='price'>



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. Let's examine engine "engine-location" and "price":

In [32]:
```
sns.boxplot(x="engine-location", y="price", data=df)
```

Out[32]: <Axes: xlabel='engine-location', ylabel='price'>

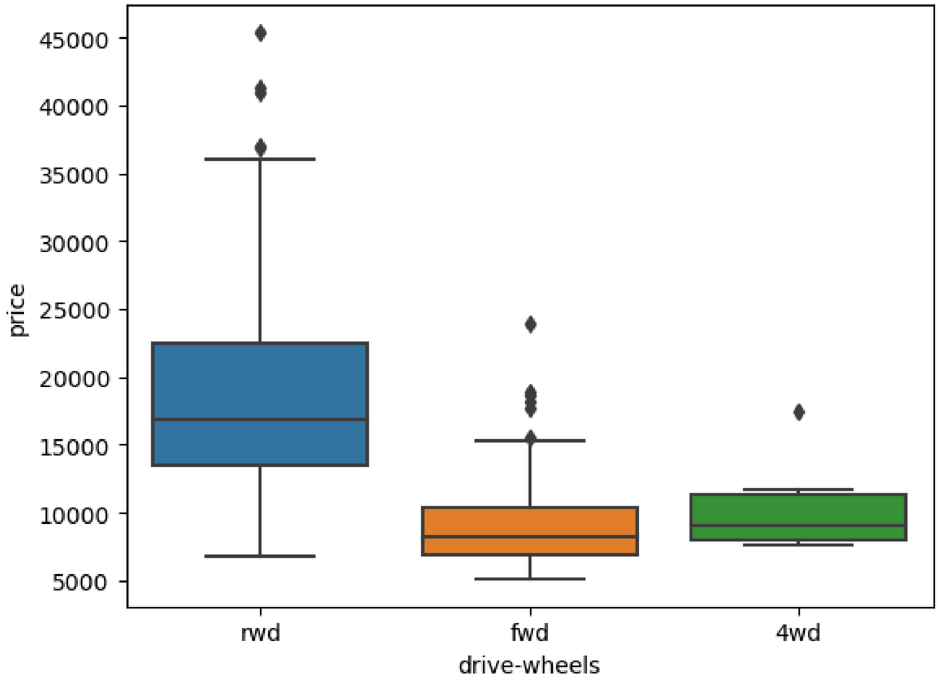


Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.

Let's examine "drive-wheels" and "price".

In [64]:
```
# drive-wheels
sns.boxplot(x="drive-wheels", y="price", data=df)
```

Out[64]: <Axes: 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.

3. Descriptive Statistical Analysis

Let's first take a look at the variables by utilizing a description method.

The **describe** function automatically computes basic statistics for all continuous variables. Any NaN values are automatically skipped in these statistics.

This will show:

- the count of that variable
- the mean
- the standard deviation (std)
- the minimum value
- the IQR (Interquartile Range: 25%, 50% and 75%)
- the maximum value

We can apply the method "describe" as follows:

In [65]:
```
df.describe()
```

Out[65]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.
mean	0.840796	122.00000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.296727	41.546834	0.268072	0.
min	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.
25%	0.000000	101.00000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.
50%	1.000000	122.00000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.
75%	2.000000	137.00000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.
max	3.000000	256.00000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.

The default setting of "describe" skips variables of type object. We can apply the method "describe" on the variables of type 'object' as follows:

In [66]:
```
df.describe(include=['object'])
```

Out[66]:

	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

Value Counts

Value counts is a good way of understanding how many units of each characteristic/variable we have. We can apply the "value_counts" method on the column "drive-wheels". Don't forget the method "value_counts" only works on pandas series, not pandas dataframes. As a result, we only include one bracket `df['drive-wheels']` , not two brackets `df[['drive-wheels']]` .

In [67]:
```
df['drive-wheels'].value_counts()
```

Out[67]:
```
drive-wheels
fwd    118
rwd     75
4wd      8
Name: count, dtype: int64
```

We can convert the series to a dataframe as follows:

In [68]:
```
df['drive-wheels'].value_counts().to_frame()
```

Out[68]:

	count
drive-wheels	
fwd	118
rwd	75
4wd	8

Let's repeat the above steps but save the results to the dataframe "drive_wheels_counts" and rename the column 'drive-wheels' to 'value_counts'.

In [69]:
```
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[69]:

	count
drive-wheels	
fwd	118
rwd	75
4wd	8

Now let's rename the index to 'drive-wheels':

In [70]:
```
drive_wheels_counts.index.name = 'drive-wheels'
drive_wheels_counts
```

Out[70]:

	count
drive-wheels	
fwd	118
rwd	75
4wd	8

We can repeat the above process for the variable 'engine-location'.

In [71]:
```
# engine-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[71]:

	count
engine-location	
front	198
rear	3

After examining the value counts of the engine location, we see that engine location would not be a good predictor variable for the price. This is because we only have three cars with a rear engine and 198 with an engine in the front, so this result is skewed. Thus, we are not able to draw any conclusions about the engine location.

4. Basics of Grouping

The "groupby" method groups data by different categories. The data is grouped based on one or several variables, and analysis is performed on the individual groups.

Let's group by the variable "drive-wheels". We see that there are 3 different categories of drive wheels.

In [72]:
```
df['drive-wheels'].unique()
```

Out[72]:
```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

If we want to know, on average, which type of drive wheel is most valuable, we can group "drive-wheels" and then average them.

We can select the columns 'drive-wheels', 'body-style' and 'price', then assign it to the variable "df_group_one".

In [74]:
```
df_group_one = df[['drive-wheels','price']]
```

We can then calculate the average price for each of the different categories of data.

In [81]:
```
# grouping results
df_group_one = df_group_one.groupby(['drive-wheels'],as_index=False).mean()
df_group_one = df_group_one.sort_values(by='price', ascending=False)
df_group_one
```

Out[81]:

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

From our 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.

You can also group by multiple variables. For example, let's group by both 'drive-wheels' and 'body-style'. This groups the dataframe by the unique combination of 'drive-wheels' and 'body-style'. We can store the results in the variable 'grouped_test1'.

In [87]:
```
# grouping results
df_gptest = df[['drive-wheels','body-style','price']]
grouped_test1 = df_gptest.groupby(['drive-wheels','body-style'],as_index=False).mean()
grouped_test1 = grouped_test1.sort_values(by='price', ascending=False)
grouped_test1
```

Out[87]:

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

This grouped data is much easier to visualize when it is made into a pivot table. A pivot table is like an Excel spreadsheet, with one variable along the column and another along the row. We can convert the dataframe to a pivot table using the method "pivot" to create a pivot table from the groups.

In this case, we will leave the drive-wheels variable as the rows of the table, and pivot body-style to become the columns of the table:

In [85]:
```
grouped_pivot = grouped_test1.pivot(index='drive-wheels',columns='body-style')
grouped_pivot
```

Out[85]:

	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

Often, we won't have data for some of the pivot cells. We can fill these missing cells with the value 0, but any other value could potentially be used as well. It should be mentioned that missing data is quite a complex subject and is an entire course on its own.

In [86]:
```
grouped_pivot = grouped_pivot.fillna(0) #fill missing values with 0
grouped_pivot
```

Out[86]:

	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

Use the "groupby" function to find the average "price" of each car based on "body-style".

In [90]:
```
df_group_bodystyle = df[['body-style','price']]
df_group_bodystyle = df_group_bodystyle.groupby(['body-style'],as_index=False).mean()
df_group_bodystyle = df_group_bodystyle.sort_values(by='price', ascending=False)
df_group_bodystyle
```

Out[90]:

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

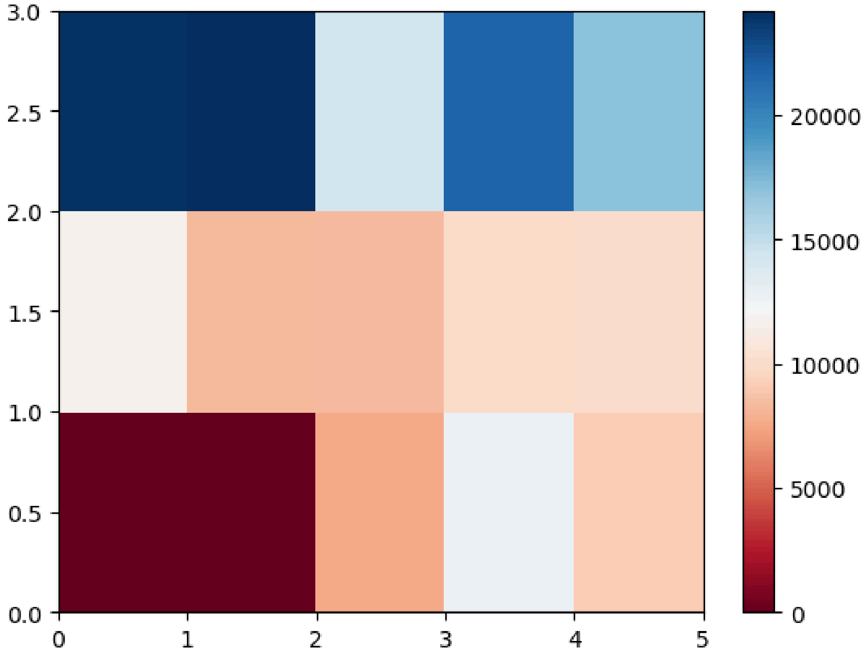
If you did not import "pyplot", let's do it again.

In [91]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Variables: Drive Wheels and Body Style vs. Price

Let's use a heat map to visualize the relationship between Body Style vs Price.

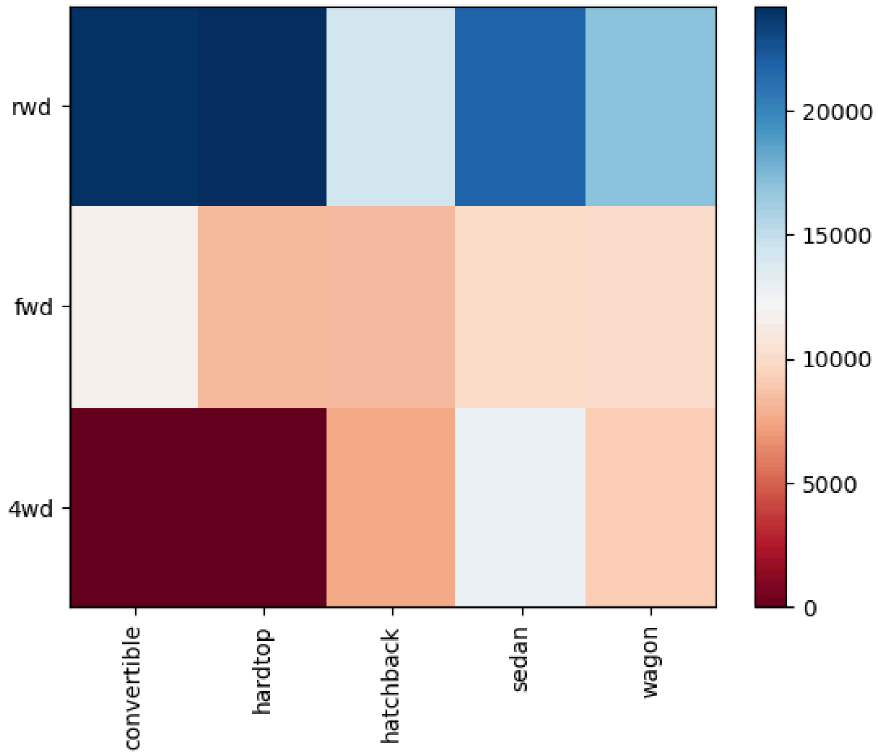
In [92]:
```
#use the grouped results
plt.pcolor(grouped_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```



The heatmap plots the target variable (price) proportional to colour with respect to the variables 'drive-wheel' and 'body-style' on the vertical and horizontal axis, respectively. This allows us to visualize how the price is related to 'drive-wheel' and 'body-style'.

The default labels convey no useful information to us. Let's change that:

In [93]:
```
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()
```




```
<p>Visualization is very important in data science, and Python visualization packages provide great freedom.</p>

<p>The main question we want to answer in this lab is, "What are the main characteristics which have the most impact on the car price?".</p>
```

To get a better measure of the important characteristics, we look at the correlation of these variables with the car price. In other words: how is the car price dependent on this variable?

5. Correlation and Causation

Correlation: a measure of the extent of interdependence between variables.

Causation: the relationship between cause and effect between two variables.

Pearson Correlation

The Pearson Correlation measures the linear dependence between two variables X and Y.

The resulting coefficient is a value between -1 and 1 inclusive, where:

- **1**: Perfect positive linear correlation.
- **0**: No linear correlation, the two variables most likely do not affect each other.
- **-1**: Perfect negative linear correlation.

Pearson Correlation is the default method of the function "corr". Like before, we can calculate the Pearson Correlation of the of the 'int64' or 'float64' variables.

In [96]:
```python
df_numeric = df.select_dtypes(include=['number'])  # Select only numeric columns
correlation_matrix = df_numeric.corr()  # Calculate the correlation matrix
correlation_matrix
```

Out[96]:

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

Sometimes we would like to know the significant of the correlation estimate.

Filter High Correlations

We can filter the correlation matrix to show only values above a certain threshold, like 0.8 (or another threshold our choice):

In [97]:
```
threshold = 0.8
high_corr = correlation_matrix[(correlation_matrix > threshold) & (correlation_matrix != 1.0)]
print(high_corr)
```

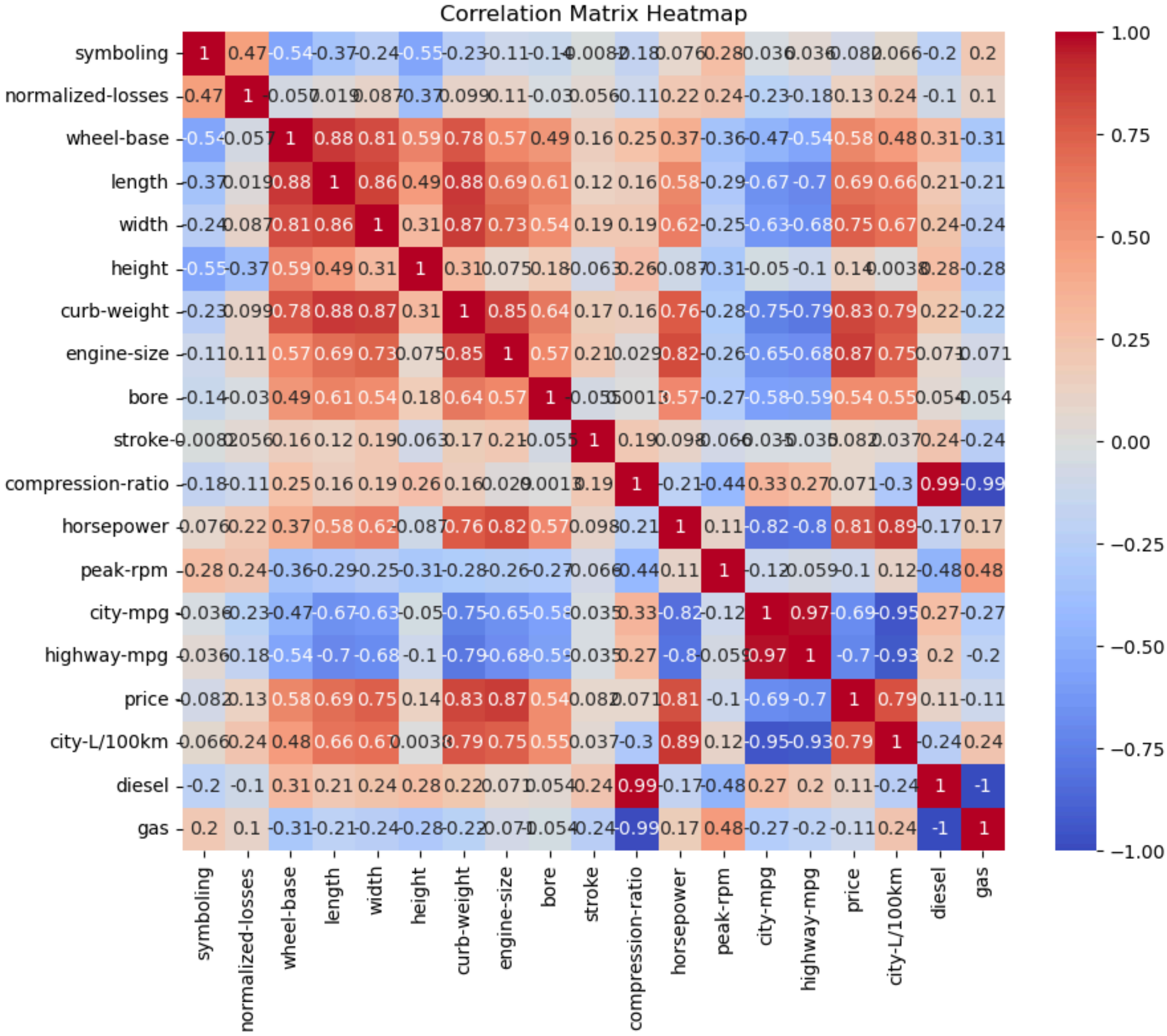
```
                   symboling  normalized-losses  wheel-base    length  \
symboling                NaN                NaN         NaN       NaN
normalized-losses        NaN                NaN         NaN       NaN
wheel-base               NaN                NaN         NaN  0.876024
length                   NaN                NaN    0.876024       NaN
width                    NaN                NaN    0.814507  0.857170
height                   NaN                NaN         NaN       NaN
curb-weight              NaN                NaN         NaN  0.880665
engine-size              NaN                NaN         NaN       NaN
bore                     NaN                NaN         NaN       NaN
stroke                   NaN                NaN         NaN       NaN
compression-ratio        NaN                NaN         NaN       NaN
horsepower               NaN                NaN         NaN       NaN
peak-rpm                 NaN                NaN         NaN       NaN
city-mpg                 NaN                NaN         NaN       NaN
highway-mpg              NaN                NaN         NaN       NaN
price                    NaN                NaN         NaN       NaN
city-L/100km             NaN                NaN         NaN       NaN
diesel                   NaN                NaN         NaN       NaN
gas                      NaN                NaN         NaN       NaN

                      width  height  curb-weight  engine-size  bore  stroke  \
symboling               NaN     NaN          NaN          NaN   NaN     NaN
normalized-losses       NaN     NaN          NaN          NaN   NaN     NaN
wheel-base         0.814507     NaN          NaN          NaN   NaN     NaN
length             0.857170     NaN     0.880665          NaN   NaN     NaN
width                   NaN     NaN     0.866201          NaN   NaN     NaN
height                  NaN     NaN          NaN          NaN   NaN     NaN
curb-weight        0.866201     NaN          NaN     0.849072   NaN     NaN
engine-size             NaN     NaN     0.849072          NaN   NaN     NaN
bore                    NaN     NaN          NaN          NaN   NaN     NaN
stroke                  NaN     NaN          NaN          NaN   NaN     NaN
compression-ratio       NaN     NaN          NaN          NaN   NaN     NaN
horsepower              NaN     NaN          NaN     0.822676   NaN     NaN
peak-rpm                NaN     NaN          NaN          NaN   NaN     NaN
city-mpg                NaN     NaN          NaN          NaN   NaN     NaN
highway-mpg             NaN     NaN          NaN          NaN   NaN     NaN
price                   NaN     NaN     0.834415     0.872335   NaN     NaN
city-L/100km            NaN     NaN          NaN          NaN   NaN     NaN
diesel                  NaN     NaN          NaN          NaN   NaN     NaN
gas                     NaN     NaN          NaN          NaN   NaN     NaN

                   compression-ratio  horsepower  peak-rpm  city-mpg  \
symboling                        NaN         NaN       NaN       NaN
normalized-losses                NaN         NaN       NaN       NaN
wheel-base                       NaN         NaN       NaN       NaN
length                           NaN         NaN       NaN       NaN
width                            NaN         NaN       NaN       NaN
height                           NaN         NaN       NaN       NaN
curb-weight                      NaN         NaN       NaN       NaN
engine-size                      NaN    0.822676       NaN       NaN
bore                             NaN         NaN       NaN       NaN
stroke                           NaN         NaN       NaN       NaN
compression-ratio                NaN         NaN       NaN       NaN
horsepower                       NaN         NaN       NaN       NaN
peak-rpm                         NaN         NaN       NaN       NaN
city-mpg                         NaN         NaN       NaN       NaN
highway-mpg                      NaN         NaN       NaN  0.972044
price                            NaN    0.809575       NaN       NaN
city-L/100km                     NaN    0.889488       NaN       NaN
diesel                      0.985231         NaN       NaN       NaN
gas                              NaN         NaN       NaN       NaN

                   highway-mpg     price  city-L/100km    diesel  gas
symboling                  NaN       NaN           NaN       NaN  NaN
normalized-losses          NaN       NaN           NaN       NaN  NaN
wheel-base                 NaN       NaN           NaN       NaN  NaN
length                     NaN       NaN           NaN       NaN  NaN
width                      NaN       NaN           NaN       NaN  NaN
height                     NaN       NaN           NaN       NaN  NaN
curb-weight                NaN  0.834415           NaN       NaN  NaN
engine-size                NaN  0.872335           NaN       NaN  NaN
bore                       NaN       NaN           NaN       NaN  NaN
stroke                     NaN       NaN           NaN       NaN  NaN
compression-ratio          NaN       NaN           NaN  0.985231  NaN
```

horsepower	NaN	0.809575	0.889488	NaN	NaN
peak-rpm	NaN	NaN	NaN	NaN	NaN
city-mpg	0.972044	NaN	NaN	NaN	NaN
highway-mpg	NaN	NaN	NaN	NaN	NaN
price	NaN	NaN	NaN	NaN	NaN
city-L/100km	NaN	NaN	NaN	NaN	NaN
diesel	NaN	NaN	NaN	NaN	NaN
gas	NaN	NaN	NaN	NaN	NaN

We use a Heatmap for Visualization

A heatmap makes it easy to visually identify high correlations:

In [98]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Get Correlation Pairs with Price

```
We can isolate the correlations between price and all other numeric columns:
Wou can also extract correlation pairs from the matrix for easier interpretation.
```

In [100]:
```python
correlation_with_price = correlation_matrix['price'].sort_values(ascending=False)
print(correlation_with_price)
```

```
price                1.000000
engine-size          0.872335
curb-weight          0.834415
horsepower           0.809575
city-L/100km         0.789898
width                0.751265
length               0.690628
wheel-base           0.584642
bore                 0.543155
height               0.135486
normalized-losses    0.133999
diesel               0.110326
stroke               0.082310
compression-ratio    0.071107
symboling           -0.082391
peak-rpm            -0.101616
gas                 -0.110326
city-mpg            -0.686571
highway-mpg         -0.704692
Name: price, dtype: float64
```

```
This shows us the correlation values of price with all other columns,
sorted from the strongest positive correlation to the strongest negative correlation.
```

```
If we want to focus only on variables with a high correlation (e.g., > 0.5), we can filter them:
```

In [101]:
```python
high_corr_with_price = correlation_with_price[correlation_with_price.abs() > 0.5]
print(high_corr_with_price)
```

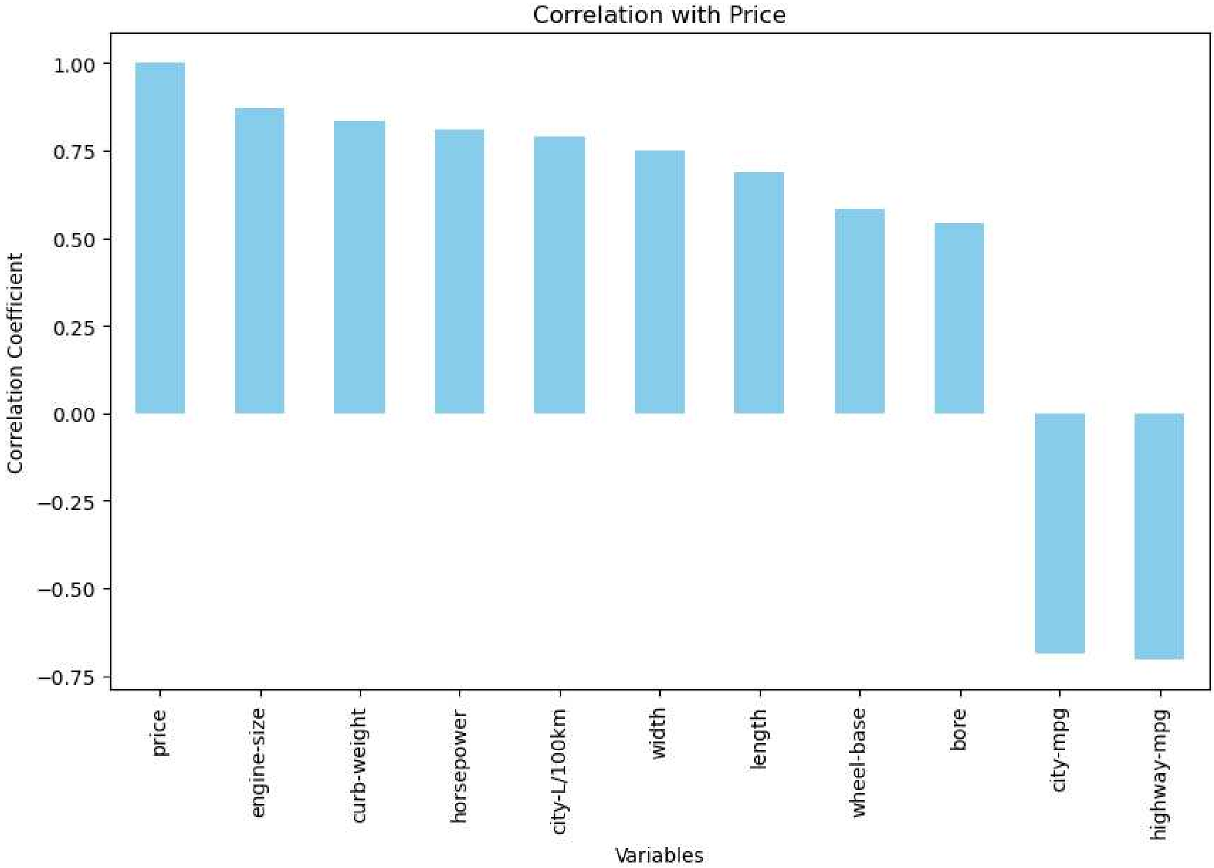
```
price           1.000000
engine-size     0.872335
curb-weight     0.834415
horsepower      0.809575
city-L/100km    0.789898
width           0.751265
length          0.690628
wheel-base      0.584642
bore            0.543155
city-mpg       -0.686571
highway-mpg    -0.704692
Name: price, dtype: float64
```

Visualize Correlation with price

A bar chart can make it easier to see the strength of correlations:

In [102]:

```
high_corr_with_price.plot(kind='bar', figsize=(10, 6), color='skyblue')
plt.title('Correlation with Price')
plt.ylabel('Correlation Coefficient')
plt.xlabel('Variables')
plt.show()
```



This has create a bar chart showing the variables most correlated with price.

By focusing on correlations with price, we:

- Directly assessed which variables are likely to influence or predict price.
- Avoid being distracted by relationships between other variables that may not impact your prediction goal.
- Identify the most important features to include in your predictive model.

P-value

What is this P-value? The P-value is the probability value that the correlation between these two variables is statistically significant. Normally, we choose a significance level of 0.05, which means that we are 95% confident that the correlation between the variables is significant.

By convention, when the

- p-value is $<$ 0.001: we say there is strong evidence that the correlation is significant.
- the p-value is $<$ 0.05: there is moderate evidence that the correlation is significant.
- the p-value is $<$ 0.1: there is weak evidence that the correlation is significant.
- the p-value is $>$ 0.1: there is no evidence that the correlation is significant.

We can obtain this information using "stats" module in the "scipy" library.

In [103]:

```
from scipy import stats
```

Wheel-Base vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'wheel-base' and 'price'.

In [104]:
```
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 0.584641822265508  with a P-value of P = 8.076488270732885e
-20
```

Conclusion:

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).

Horsepower vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'horsepower' and 'price'.

In [ ]:
```
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)
```

Conclusion:

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).

Length vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'length' and 'price'.

In [105]:
```
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 0.6906283804483638  with a P-value of P =  8.01647746615972
3e-30
```

Conclusion:

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).

Width vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'width' and 'price':

In [106]:
```
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 0.7512653440522673  with a P-value of P = 9.20033551048206e
-38
```

Conclusion:

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).

Curb-Weight vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'curb-weight' and 'price':

In [107]:
```
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 0.8344145257702843  with a P-value of P =  2.18957723889396
5e-53
```

Conclusion:

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).

Engine-Size vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'engine-size' and 'price':

In [108]:
```
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 0.8723351674455185  with a P-value of P = 9.265491622198793
e-64
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between engine-size and price is statistically significant, and the linear relationship is very strong (~0.872).

Bore vs. Price

Let's calculate the Pearson Correlation Coefficient and P-value of 'bore' and 'price':

In [109]:
```
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 0.5431553832626602  with a P-value of P =   8.0491894839353
15e-17
```

Conclusion:

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).

We can relate the process for each 'city-mpg' and 'highway-mpg':

City-mpg vs. Price

In [110]:
```
pearson_coef, p_value = stats.pearsonr(df['city-mpg'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
```

```
The Pearson Correlation Coefficient is -0.6865710067844678  with a P-value of P =  2.3211320655675
098e-29
```

Conclusion:

Since the p-value is $<$ 0.001, the correlation between city-mpg and price is statistically significant, and the coefficient of about -0.687 shows that the relationship is negative and moderately strong.

Highway-mpg vs. Price

In [111]:
```
pearson_coef, p_value = stats.pearsonr(df['highway-mpg'], df['price'])
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value )
```

```
The Pearson Correlation Coefficient is -0.704692265058953  with a P-value of P =  1.74954711444755
7e-31
```

Conclusion:

Since the p-value is < 0.001, the correlation between highway-mpg and price is statistically significant, and the coefficient of about -0.705 shows that the relationship is negative and moderately strong.

Conclusion: Important Variables

We now have a better idea of what our data looks like and which variables are important to take into account when predicting the car price. We have narrowed it down to the following variables:

Continuous numerical variables:

- Length
- Width
- Curb-weight
- Engine-size
- Horsepower
- City-mpg
- Highway-mpg
- Wheel-base
- Bore

Categorical variables:

- Drive-wheels

As we now move into building machine learning models to automate our analysis, feeding the model with variables that meaningfully affect our target variable will improve our model's prediction performance.