

# Exploratory\_data\_analysis\_cars-20231003-1696291200.jupyterlite

January 10, 2024

## 1 Exploratory Data Analysis

Estimated time needed: **30** minutes

### 1.1 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

Import Data from Module

Analyzing Individual Feature Patterns using Visualization

Descriptive Statistical Analysis

Basics of Grouping

Correlation and Causation

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

### 1.2 Import Data from Module 2

Setup

Import libraries:

```
[1]: #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
```

```
[2]: import pandas as pd  
import numpy as np  
import seaborn as sns
```

Download the updated dataset by running the cell below.

The functions below will download the dataset into your browser and store it in dataframe **df**:

This dataset was hosted on IBM Cloud object. Click [HERE](#) for free storage.

```
[3]: '''from pyodide.http import pyfetch

async def download(url, filename):
    response = await pyfetch(url)
    if response.status == 200:
        with open(filename, "wb") as f:
            f.write(await response.bytes())'''
```

```
[3]: 'from pyodide.http import pyfetch\n\nasync def download(url, filename):\n    response = await pyfetch(url)\n    if response.status == 200:\n        with\nopen(filename, "wb") as f:\n            f.write(await response.bytes())'
```

```
[4]: #file_path= "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/\n      ↳IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/\n      ↳automobileEDA.csv"
```

```
[5]: file_name="cleaned_used_cars.csv"
```

```
[6]: df = pd.read_csv('cleaned_used_cars.csv')
```

```
[7]: #filepath='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/\n      ↳IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/\n      ↳automobileEDA.csv'\n      #df = pd.read_csv(filepath, header=None)
```

View the first 5 values of the updated dataframe using `dataframe.head()`

```
[8]: df.head()
```

```
[8]: Unnamed: 0  symboling  normalized-losses  make  num-of-doors  \
0          0          3          122  alfa-romero          two
1          1          3          122  alfa-romero          two
2          2          1          122  alfa-romero          two
3          3          2          164    audi          four
4          4          2          164    audi          four

      body-style  drive-wheels  engine-location  wheel-base  length  ...  \
0  convertible          rwd          front          88.6  0.811148  ...
1  convertible          rwd          front          88.6  0.811148  ...
2   hatchback          rwd          front          94.5  0.822681  ...
3        sedan          fwd          front          99.8  0.848630  ...
4        sedan          4wd          front          99.4  0.848630  ...

      peak-rpm  city-mpg  highway-mpg  price  city-L/100km  horsepower-binned  \
0    5000.0        21      8.703704  13495.0      11.190476                Low
```

1	5000.0	21	8.703704	16500.0	11.190476	Low
2	5000.0	19	9.038462	16500.0	12.368421	Medium
3	5500.0	24	7.833333	13950.0	9.791667	Low
4	5500.0	18	10.681818	17450.0	13.055556	Low

	fuel-type-diesel	fuel-type-gas	aspiration-std	aspiration-turbo
0	False	True	True	False
1	False	True	True	False
2	False	True	True	False
3	False	True	True	False
4	False	True	True	False

[5 rows x 31 columns]

### 1.3 Analyzing Individual Feature Patterns Using Visualization

To install Seaborn we use pip, the Python package manager.

Import visualization packages “Matplotlib” and “Seaborn”. Don’t forget about “%matplotlib inline” to plot in a Jupyter notebook.

```
[9]: 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 you are dealing with. This will help us find the right visualization method for that variable.

```
[10]: # list the data types for each column
print(df.dtypes)
```

```
Unnamed: 0          int64
symboling          int64
normalized-losses  int64
make              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          int64
peak-rpm            float64
city-mpg            int64
highway-mpg         float64
price               float64
city-L/100km        float64
horsepower-binned   object
fuel-type-diesel    bool
fuel-type-gas       bool
aspiration-std      bool
aspiration-turbo    bool
dtype: object

```

```
[11]: df['peak-rpm'].dtypes
```

```
[11]: dtype('float64')
```

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

```
[12]: numeric_df = df.select_dtypes(include=['float64', 'int64'])
numeric_df.corr()
```

```
[12]:
```

	Unnamed: 0	symboling	normalized-losses	wheel-base	\
Unnamed: 0	1.000000	-0.162764	-0.241092	0.125517	
symboling	-0.162764	1.000000	0.466264	-0.535987	
normalized-losses	-0.241092	0.466264	1.000000	-0.056661	
wheel-base	0.125517	-0.535987	-0.056661	1.000000	
length	0.161848	-0.365404	0.019424	0.876024	
width	0.043976	-0.242423	0.086802	0.814507	
height	0.252015	-0.550160	-0.373737	0.590742	
curb-weight	0.064820	-0.233118	0.099404	0.782097	
engine-size	-0.047764	-0.110581	0.112360	0.572027	
bore	0.244734	-0.140019	-0.029862	0.493244	
stroke	-0.162490	-0.008153	0.055045	0.158018	
compression-ratio	0.144301	-0.182196	-0.114713	0.250313	
horsepower	-0.022505	0.075810	0.217300	0.371178	
peak-rpm	-0.195662	0.279740	0.239543	-0.360305	
city-mpg	0.027956	-0.035527	-0.225016	-0.470606	
highway-mpg	-0.078346	-0.029807	0.181189	0.577576	
price	-0.118214	-0.082391	0.133999	0.584642	
city-L/100km	-0.099157	0.066171	0.238567	0.476153	
	length	width	height	curb-weight	engine-size \
Unnamed: 0	0.161848	0.043976	0.252015	0.064820	-0.047764

symboling	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581
normalized-losses	0.019424	0.086802	-0.373737	0.099404	0.112360
wheel-base	0.876024	0.814507	0.590742	0.782097	0.572027
length	1.000000	0.857170	0.492063	0.880665	0.685025
width	0.857170	1.000000	0.306002	0.866201	0.729436
height	0.492063	0.306002	1.000000	0.307581	0.074694
curb-weight	0.880665	0.866201	0.307581	1.000000	0.849072
engine-size	0.685025	0.729436	0.074694	0.849072	1.000000
bore	0.608971	0.544885	0.180449	0.644060	0.572609
stroke	0.123952	0.188822	-0.060663	0.167438	0.205928
compression-ratio	0.159733	0.189867	0.259737	0.156433	0.028889
horsepower	0.579795	0.615056	-0.087001	0.757981	0.822668
peak-rpm	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733
city-mpg	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546
highway-mpg	0.707108	0.736728	0.084301	0.836921	0.783465
price	0.690628	0.751265	0.135486	0.834415	0.872335
city-L/100km	0.657373	0.673363	0.003811	0.785353	0.745059

	bore	stroke	compression-ratio	horsepower	\
Unnamed: 0	0.244734	-0.162490	0.144301	-0.022505	
symboling	-0.140019	-0.008153	-0.182196	0.075810	
normalized-losses	-0.029862	0.055045	-0.114713	0.217300	
wheel-base	0.493244	0.158018	0.250313	0.371178	
length	0.608971	0.123952	0.159733	0.579795	
width	0.544885	0.188822	0.189867	0.615056	
height	0.180449	-0.060663	0.259737	-0.087001	
curb-weight	0.644060	0.167438	0.156433	0.757981	
engine-size	0.572609	0.205928	0.028889	0.822668	
bore	1.000000	-0.055390	0.001263	0.566903	
stroke	-0.055390	1.000000	0.187871	0.098128	
compression-ratio	0.001263	0.187871	1.000000	-0.214489	
horsepower	0.566903	0.098128	-0.214489	1.000000	
peak-rpm	-0.267392	-0.063561	-0.435780	0.107884	
city-mpg	-0.582027	-0.033956	0.331425	-0.822192	
highway-mpg	0.559112	0.047089	-0.223361	0.840627	
price	0.543155	0.082269	0.071107	0.809607	
city-L/100km	0.554610	0.036133	-0.299372	0.889482	

	peak-rpm	city-mpg	highway-mpg	price	city-L/100km
Unnamed: 0	-0.195662	0.027956	-0.078346	-0.118214	-0.099157
symboling	0.279740	-0.035527	-0.029807	-0.082391	0.066171
normalized-losses	0.239543	-0.225016	0.181189	0.133999	0.238567
wheel-base	-0.360305	-0.470606	0.577576	0.584642	0.476153
length	-0.285970	-0.665192	0.707108	0.690628	0.657373
width	-0.245800	-0.633531	0.736728	0.751265	0.673363
height	-0.309974	-0.049800	0.084301	0.135486	0.003811
curb-weight	-0.279361	-0.749543	0.836921	0.834415	0.785353

engine-size	-0.256733	-0.650546	0.783465	0.872335	0.745059
bore	-0.267392	-0.582027	0.559112	0.543155	0.554610
stroke	-0.063561	-0.033956	0.047089	0.082269	0.036133
compression-ratio	-0.435780	0.331425	-0.223361	0.071107	-0.299372
horsepower	0.107884	-0.822192	0.840627	0.809607	0.889482
peak-rpm	1.000000	-0.115413	0.017694	-0.101616	0.115830
city-mpg	-0.115413	1.000000	-0.909024	-0.686571	-0.949713
highway-mpg	0.017694	-0.909024	1.000000	0.801118	0.958306
price	-0.101616	-0.686571	0.801118	1.000000	0.789898
city-L/100km	0.115830	-0.949713	0.958306	0.789898	1.000000

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

```
[13]: # Write your code below and press Shift+Enter to execute
df[['bore', 'stroke', 'compression-ratio', 'horsepower']].corr()
```

```
[13]:          bore    stroke  compression-ratio  horsepower
bore          1.000000 -0.055390          0.001263    0.566903
stroke        -0.055390  1.000000          0.187871    0.098128
compression-ratio  0.001263  0.187871          1.000000   -0.214489
horsepower       0.566903  0.098128         -0.214489    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.

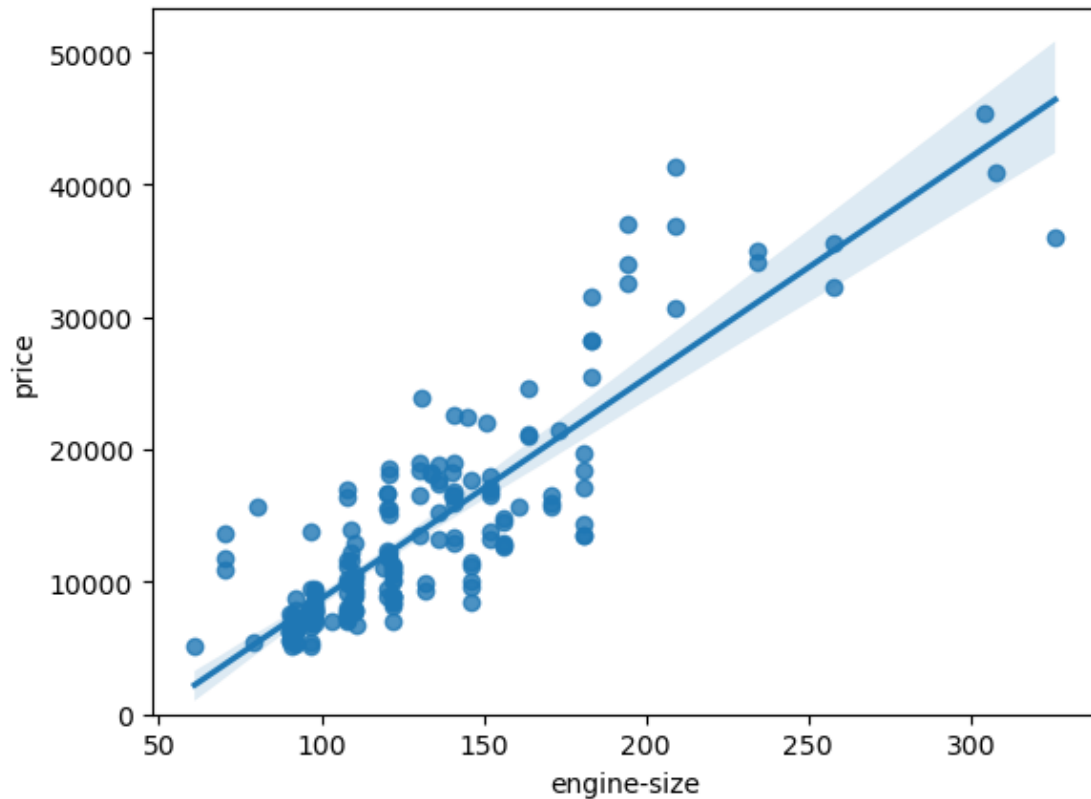
Let’s see several examples of different linear relationships:

Positive Linear Relationship

Let’s find the scatterplot of “engine-size” and “price”.

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

```
[14]: (0.0, 53327.553366673535)
```



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.

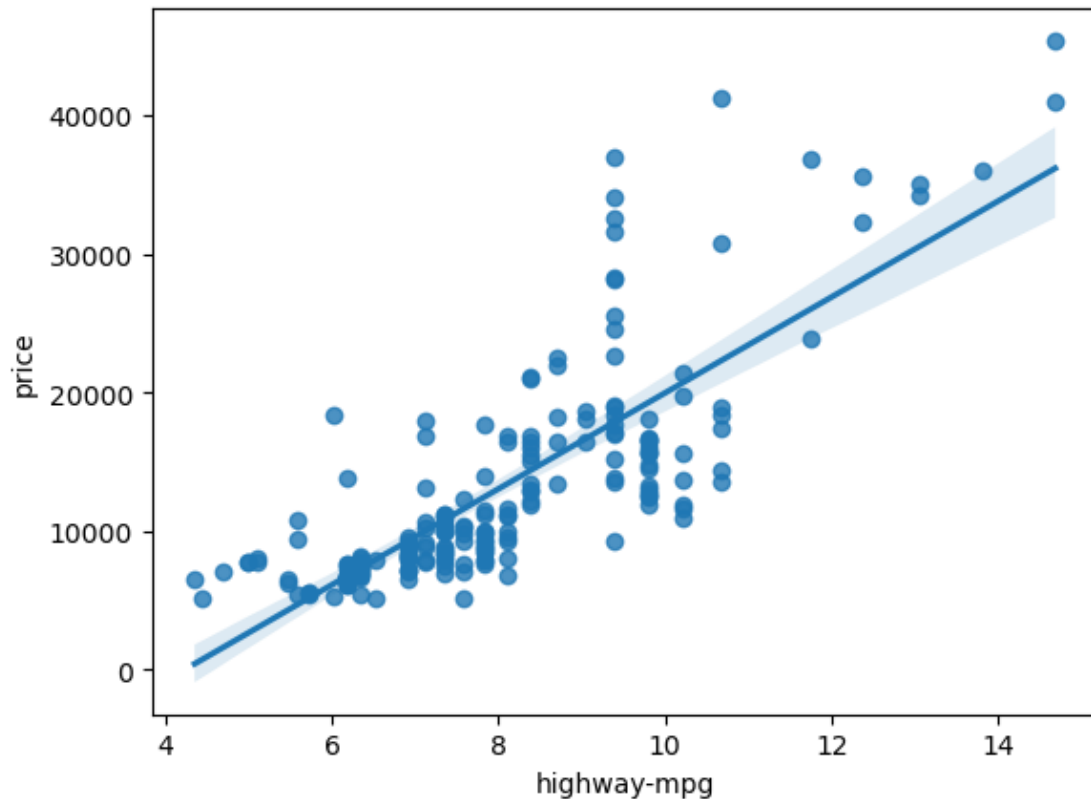
```
[15]: df[["engine-size", "price"]].corr()
```

```
[15]:      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”.

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

```
[16]: <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.

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

```
[17]:      highway-mpg    price
highway-mpg    1.000000  0.801118
price          0.801118  1.000000
```

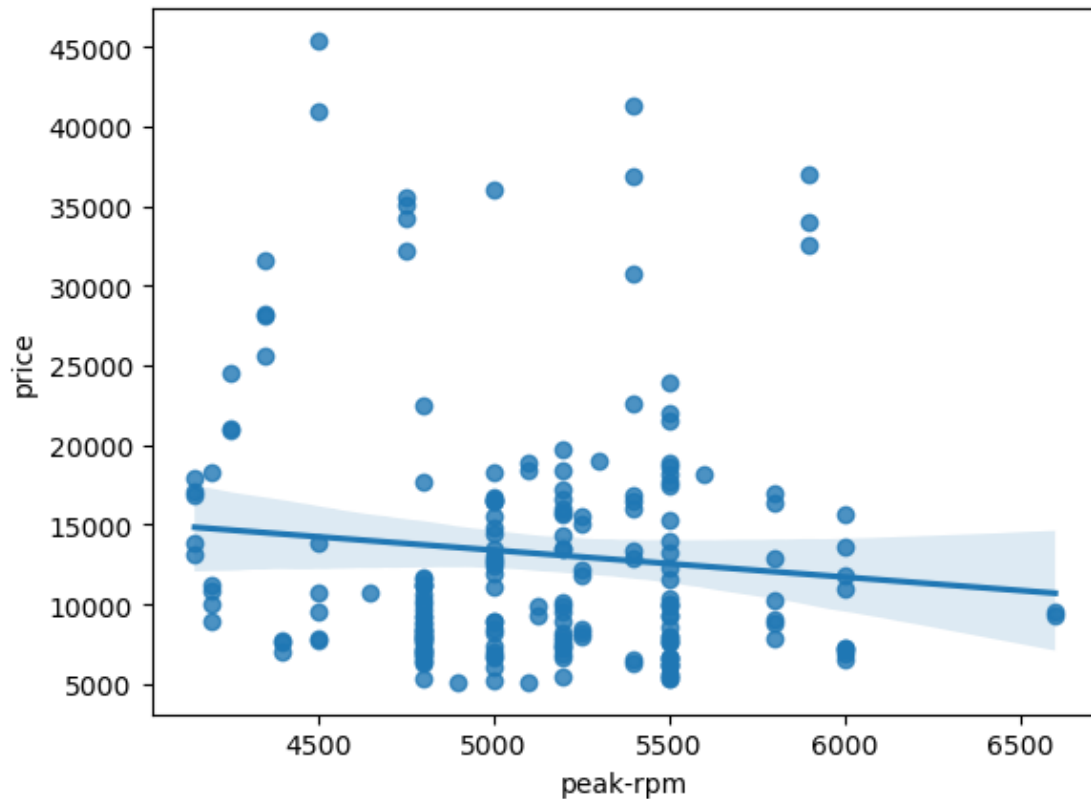
Weak Linear Relationship

Let’s see if “peak-rpm” is a predictor variable of “price”.

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

```
[18]: <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.

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

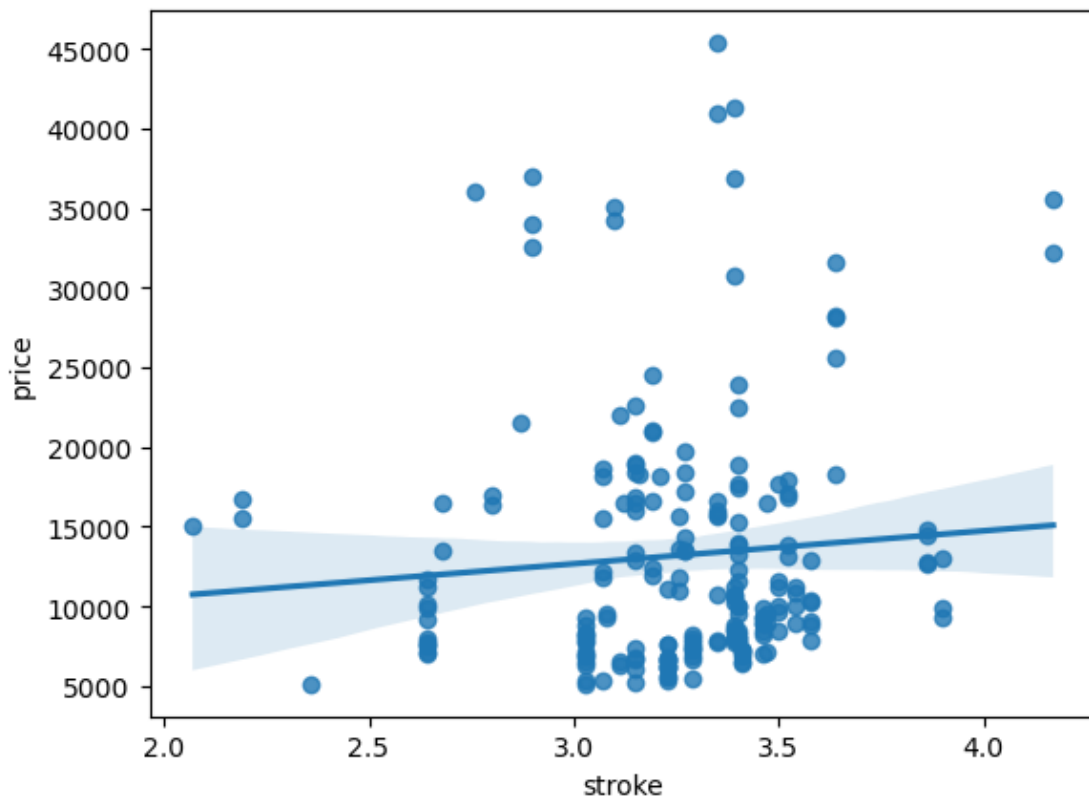
```
[19]:      peak-rpm    price
peak-rpm  1.000000 -0.101616
price    -0.101616  1.000000
```

```
[20]: # Write your code below and press Shift+Enter to execute
df[["stroke", "price"]].corr()
```

```
[20]:      stroke    price
stroke  1.000000  0.082269
price   0.082269  1.000000
```

```
[21]: # Write your code below and press Shift+Enter to execute
sns.regplot(x="stroke", y="price", data=df)
```

```
[21]: <Axes: xlabel='stroke', ylabel='price'>
```



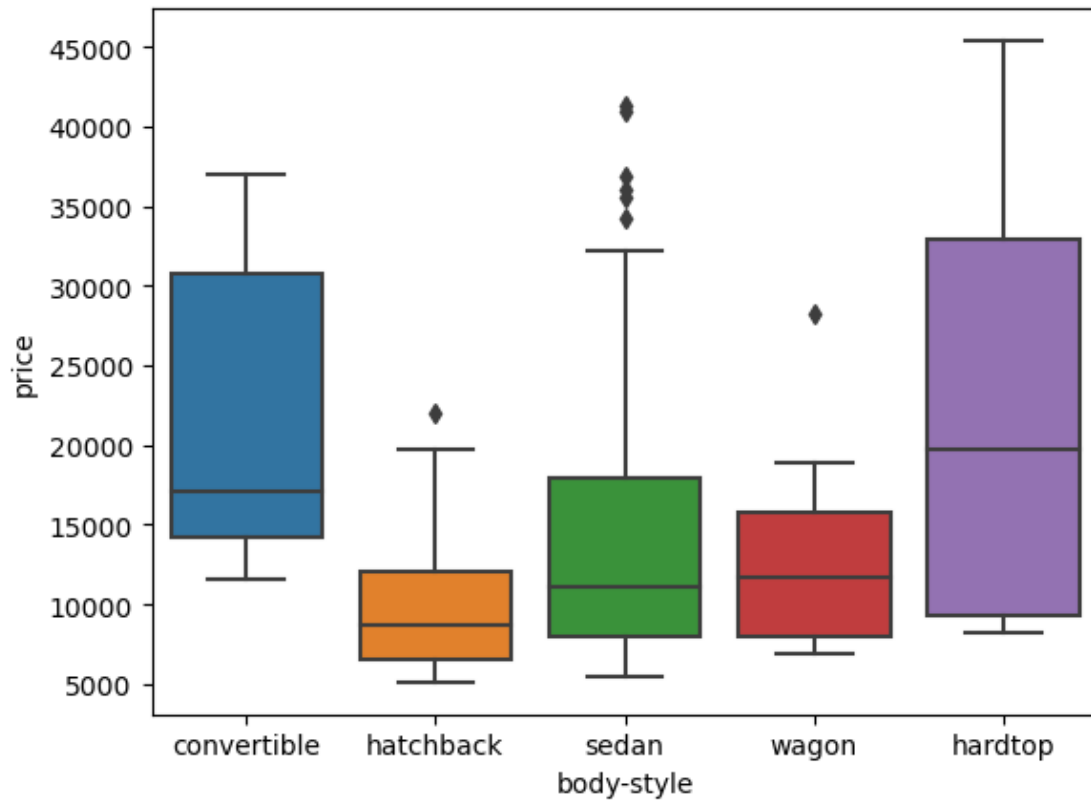
### Categorical Variables

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”.

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

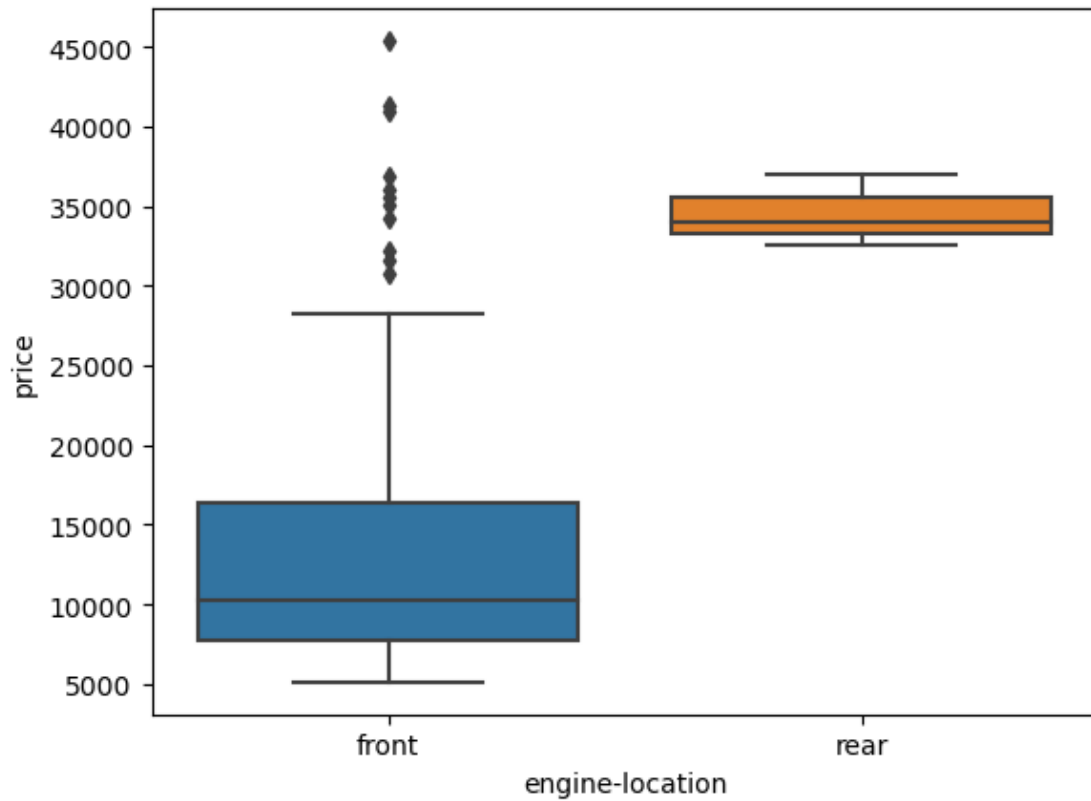
```
[22]: <Axes: xlabel='body-style', ylabel='price'>
```



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":

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

```
[23]: <Axes: xlabel='engine-location', ylabel='price'>
```

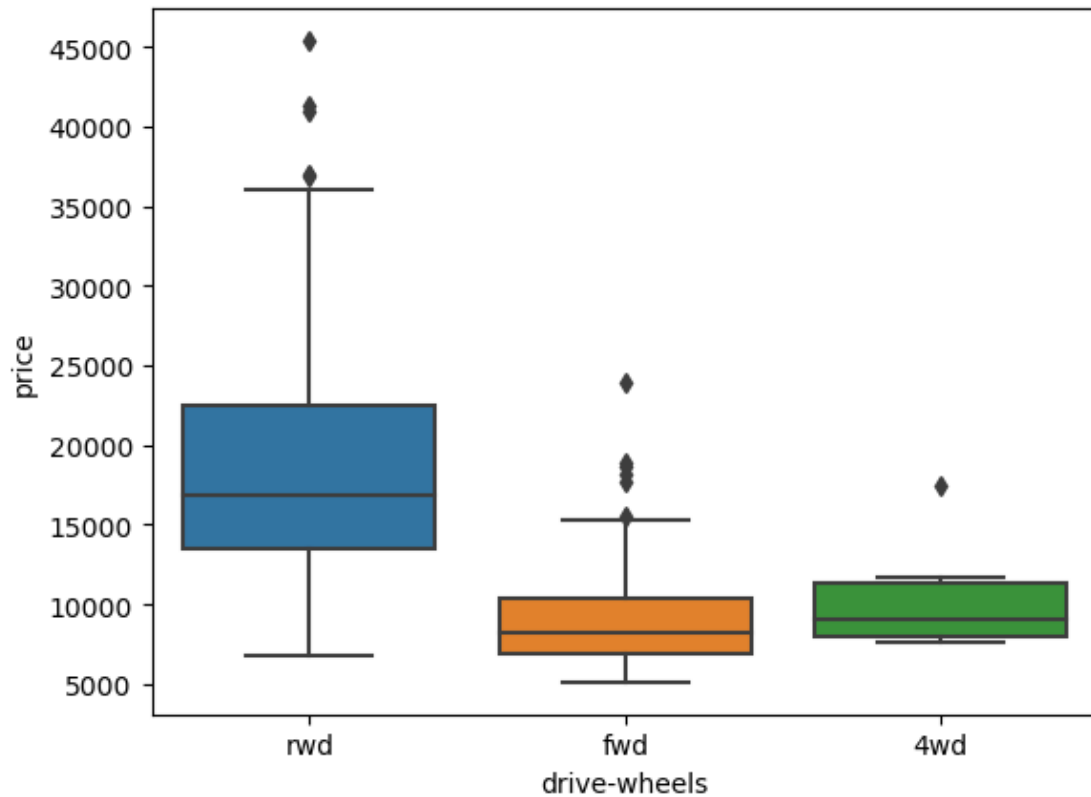


Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.

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

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

```
[24]: <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.

## 1.4 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:

```
[25]: df.describe()
```

```
[25]:
```

	Unnamed: 0	symboling	normalized-losses	wheel-base	length	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	100.000000	0.840796	122.000000	98.797015	0.837102	
std	58.167861	1.254802	31.99625	6.066366	0.059213	
min	0.000000	-2.000000	65.000000	86.600000	0.678039	
25%	50.000000	0.000000	101.000000	94.500000	0.801538	
50%	100.000000	1.000000	122.000000	97.000000	0.832292	
75%	150.000000	2.000000	137.000000	102.400000	0.881788	
max	200.000000	3.000000	256.000000	120.900000	1.000000	

	width	height	curb-weight	engine-size	bore	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	0.915126	0.899108	2555.666667	126.875622	3.330692	
std	0.029187	0.040933	517.296727	41.546834	0.268072	
min	0.837500	0.799331	1488.000000	61.000000	2.540000	
25%	0.890278	0.869565	2169.000000	98.000000	3.150000	
50%	0.909722	0.904682	2414.000000	120.000000	3.310000	
75%	0.925000	0.928094	2926.000000	141.000000	3.580000	
max	1.000000	1.000000	4066.000000	326.000000	3.940000	

	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	3.256874	10.164279	103.402985	5117.665368	25.179104	
std	0.316048	4.004965	37.365650	478.113805	6.423220	
min	2.070000	7.000000	48.000000	4150.000000	13.000000	
25%	3.110000	8.600000	70.000000	4800.000000	19.000000	
50%	3.290000	9.000000	95.000000	5125.369458	24.000000	
75%	3.410000	9.400000	116.000000	5500.000000	30.000000	
max	4.170000	23.000000	262.000000	6600.000000	49.000000	

	highway-mpg	price	city-L/100km
count	201.000000	201.000000	201.000000
mean	8.044957	13207.129353	9.944145
std	1.840739	7947.066342	2.534599
min	4.351852	5118.000000	4.795918
25%	6.911765	7775.000000	7.833333
50%	7.833333	10295.000000	9.791667
75%	9.400000	16500.000000	12.368421
max	14.687500	45400.000000	18.076923

The default setting of “describe” skips variables of type object. We can apply the method “describe” on the variables of type ‘object’ as follows:

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

```
[26]:      make num-of-doors body-style drive-wheels engine-location \
count      201          201          201          201          201
unique      22           2           5           3           2
top    toyota         four        sedan         fwd         front
freq       32          115          94          118          198

      engine-type num-of-cylinders fuel-system horsepower-binned
count      201          201          201          201
unique       6           7           8           3
top         ohc         four        mpfi         Low
freq      145          157          92          153
```

### 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']].

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

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

We can convert the series to a dataframe as follows:

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

```
[28]:      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’.

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

```
[29]:      count
drive-wheels
fwd          118
rwd           75
```

4wd	8
-----	---

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

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

```
[30]:          count
drive-wheels
fwd          118
rwd           75
4wd           8
```

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

```
[31]: # 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)
```

```
[31]:          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.

## 1.5 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.

For example, let's group by the variable “drive-wheels”. We see that there are 3 different categories of drive wheels.

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

```
[32]: 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”.

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



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

```
[34]: # Assuming you want to analyze 'price' column based on 'drive-wheels'
df_group_one = df.groupby(['drive-wheels'], as_index=False)['price'].mean()
df_group_one
```

```
[34]:  drive-wheels      price
0         4wd  10241.000000
1         fwd   9244.779661
2         rwd  19757.613333
```

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'.

```
[35]: # 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
```

```
[35]:  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. 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:

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

```
[36]:
```

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

body-style	wagon
drive-wheels	
4wd	9095.750000
fwd	9997.333333
rwd	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.

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

```
[37]:
```

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

body-style	wagon
drive-wheels	
4wd	9095.750000
fwd	9997.333333
rwd	16994.222222

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

```
[38]:
```

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

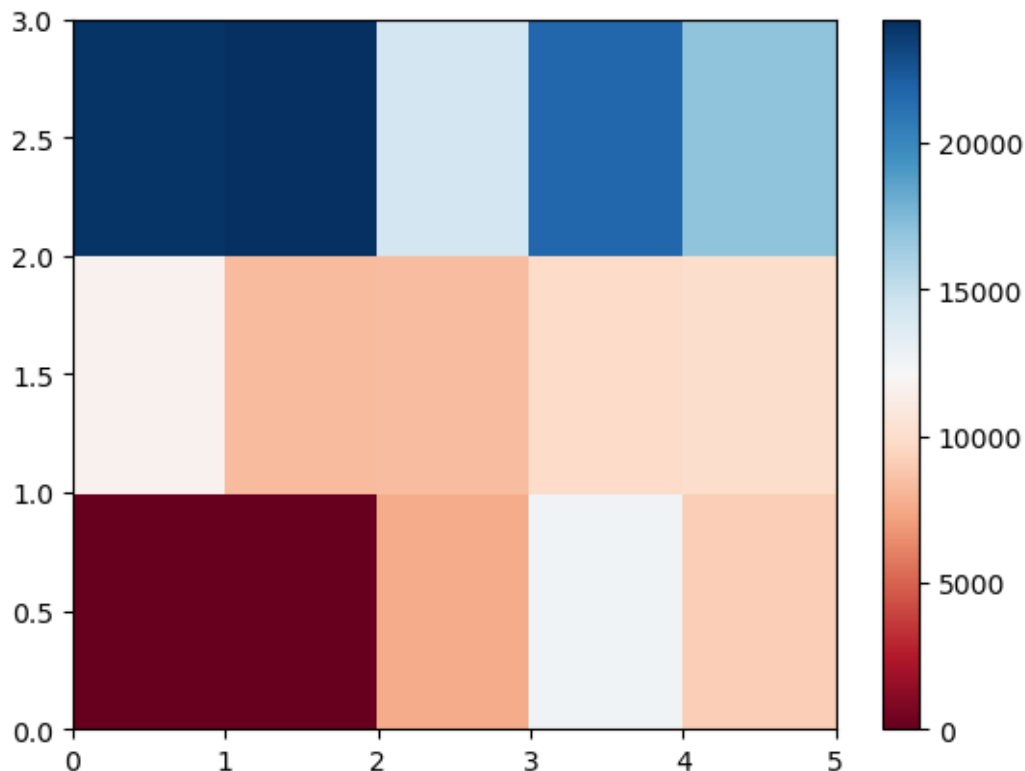
If you did not import “pyplot”, let's do it again.

```
[39]: 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.

```
[40]: #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:

```
[41]: 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

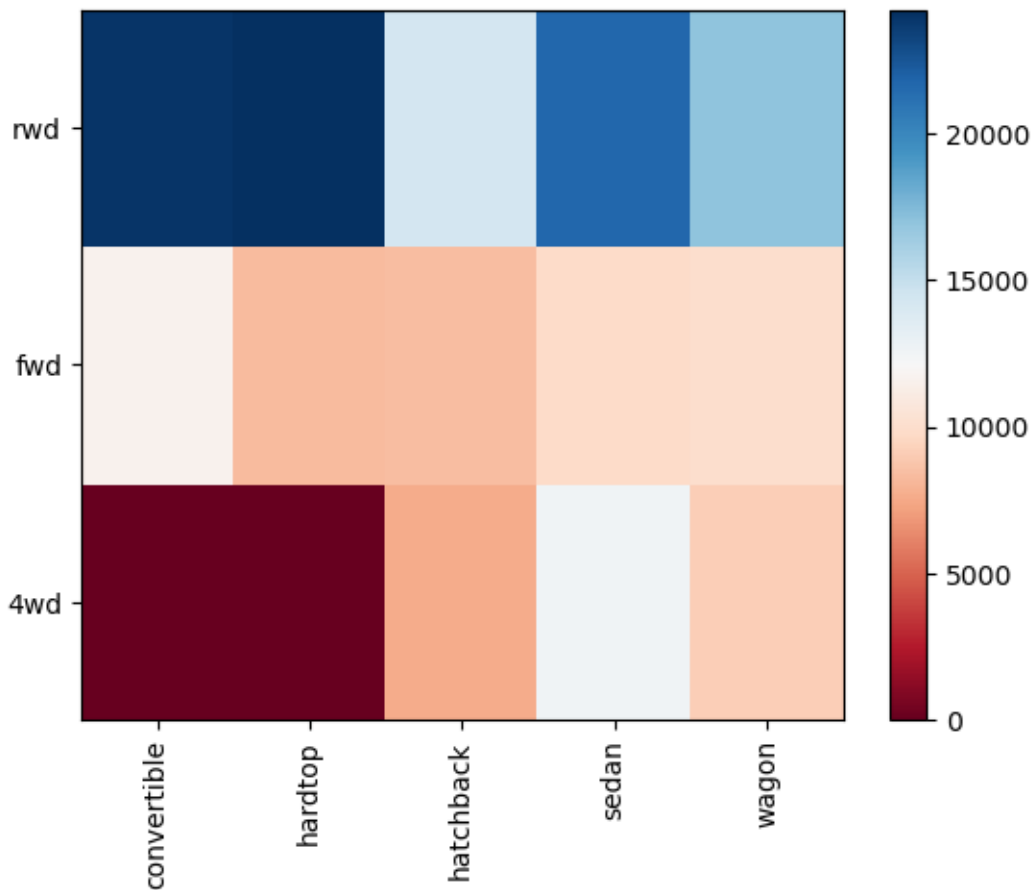
#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)

#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()

```



Visualization is very important in data science, and Python visualization packages provide great freedom. We will go more in-depth in a separate Python visualizations course.

The main question we want to answer in this module is, “What are the main characteristics which have the most impact on the car price?”.

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?

## 1.6 Correlation and Causation

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

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

It is important to know the difference between these two. Correlation does not imply causation. Determining correlation is much simpler the determining causation as causation may require independent experimentation.

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.

```
[42]: numeric_df = df.select_dtypes(include=['float64', 'int64'])
      numeric_df.corr()
```

```
[42]:
```

	Unnamed: 0	symboling	normalized-losses	wheel-base	\
Unnamed: 0	1.000000	-0.162764	-0.241092	0.125517	
symboling	-0.162764	1.000000	0.466264	-0.535987	
normalized-losses	-0.241092	0.466264	1.000000	-0.056661	
wheel-base	0.125517	-0.535987	-0.056661	1.000000	
length	0.161848	-0.365404	0.019424	0.876024	
width	0.043976	-0.242423	0.086802	0.814507	
height	0.252015	-0.550160	-0.373737	0.590742	
curb-weight	0.064820	-0.233118	0.099404	0.782097	
engine-size	-0.047764	-0.110581	0.112360	0.572027	
bore	0.244734	-0.140019	-0.029862	0.493244	
stroke	-0.162490	-0.008153	0.055045	0.158018	
compression-ratio	0.144301	-0.182196	-0.114713	0.250313	
horsepower	-0.022505	0.075810	0.217300	0.371178	
peak-rpm	-0.195662	0.279740	0.239543	-0.360305	
city-mpg	0.027956	-0.035527	-0.225016	-0.470606	
highway-mpg	-0.078346	-0.029807	0.181189	0.577576	
price	-0.118214	-0.082391	0.133999	0.584642	
city-L/100km	-0.099157	0.066171	0.238567	0.476153	

	length	width	height	curb-weight	engine-size \
Unnamed: 0	0.161848	0.043976	0.252015	0.064820	-0.047764
symboling	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581
normalized-losses	0.019424	0.086802	-0.373737	0.099404	0.112360
wheel-base	0.876024	0.814507	0.590742	0.782097	0.572027
length	1.000000	0.857170	0.492063	0.880665	0.685025
width	0.857170	1.000000	0.306002	0.866201	0.729436
height	0.492063	0.306002	1.000000	0.307581	0.074694
curb-weight	0.880665	0.866201	0.307581	1.000000	0.849072
engine-size	0.685025	0.729436	0.074694	0.849072	1.000000
bore	0.608971	0.544885	0.180449	0.644060	0.572609
stroke	0.123952	0.188822	-0.060663	0.167438	0.205928
compression-ratio	0.159733	0.189867	0.259737	0.156433	0.028889
horsepower	0.579795	0.615056	-0.087001	0.757981	0.822668
peak-rpm	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733
city-mpg	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546
highway-mpg	0.707108	0.736728	0.084301	0.836921	0.783465
price	0.690628	0.751265	0.135486	0.834415	0.872335
city-L/100km	0.657373	0.673363	0.003811	0.785353	0.745059

	bore	stroke	compression-ratio	horsepower \
Unnamed: 0	0.244734	-0.162490	0.144301	-0.022505
symboling	-0.140019	-0.008153	-0.182196	0.075810
normalized-losses	-0.029862	0.055045	-0.114713	0.217300
wheel-base	0.493244	0.158018	0.250313	0.371178
length	0.608971	0.123952	0.159733	0.579795
width	0.544885	0.188822	0.189867	0.615056
height	0.180449	-0.060663	0.259737	-0.087001
curb-weight	0.644060	0.167438	0.156433	0.757981
engine-size	0.572609	0.205928	0.028889	0.822668
bore	1.000000	-0.055390	0.001263	0.566903
stroke	-0.055390	1.000000	0.187871	0.098128
compression-ratio	0.001263	0.187871	1.000000	-0.214489
horsepower	0.566903	0.098128	-0.214489	1.000000
peak-rpm	-0.267392	-0.063561	-0.435780	0.107884
city-mpg	-0.582027	-0.033956	0.331425	-0.822192
highway-mpg	0.559112	0.047089	-0.223361	0.840627
price	0.543155	0.082269	0.071107	0.809607
city-L/100km	0.554610	0.036133	-0.299372	0.889482

	peak-rpm	city-mpg	highway-mpg	price	city-L/100km
Unnamed: 0	-0.195662	0.027956	-0.078346	-0.118214	-0.099157
symboling	0.279740	-0.035527	-0.029807	-0.082391	0.066171
normalized-losses	0.239543	-0.225016	0.181189	0.133999	0.238567
wheel-base	-0.360305	-0.470606	0.577576	0.584642	0.476153
length	-0.285970	-0.665192	0.707108	0.690628	0.657373

width	-0.245800	-0.633531	0.736728	0.751265	0.673363
height	-0.309974	-0.049800	0.084301	0.135486	0.003811
curb-weight	-0.279361	-0.749543	0.836921	0.834415	0.785353
engine-size	-0.256733	-0.650546	0.783465	0.872335	0.745059
bore	-0.267392	-0.582027	0.559112	0.543155	0.554610
stroke	-0.063561	-0.033956	0.047089	0.082269	0.036133
compression-ratio	-0.435780	0.331425	-0.223361	0.071107	-0.299372
horsepower	0.107884	-0.822192	0.840627	0.809607	0.889482
peak-rpm	1.000000	-0.115413	0.017694	-0.101616	0.115830
city-mpg	-0.115413	1.000000	-0.909024	-0.686571	-0.949713
highway-mpg	0.017694	-0.909024	1.000000	0.801118	0.958306
price	-0.101616	-0.686571	0.801118	1.000000	0.789898
city-L/100km	0.115830	-0.949713	0.958306	0.789898	1.000000

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

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.

```
[43]: from scipy import stats
```

Wheel-Base vs. Price

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

```
[44]: 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 0.5846418222655083 with a P-value of P =
8.076488270732552e-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 ( $\sim 0.585$ ).

Horsepower vs. Price

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

```
[45]: 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 0.8096068016571052 with a P-value of P = 6.273536270651023e-48

Conclusion:

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

Length vs. Price

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

```
[46]: 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 0.6906283804483644 with a P-value of P = 8.016477466158383e-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 ( $\sim 0.691$ ).

Width vs. Price

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

```
[47]: 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 0.7512653440522665 with a P-value of P = 9.200335510484122e-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 ( $\sim 0.751$ ).

### 1.6.1 Curb-Weight vs. Price

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

```
[48]: 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 0.8344145257702849 with a P-value of P = 2.189577238893391e-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 ( $\sim 0.834$ ).

Engine-Size vs. Price

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

```
[49]: 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 = 9.265491622198793e-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 ( $\sim 0.872$ ).

Bore vs. Price

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

```
[50]: 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 0.5431553832626606 with a P-value of P = 8.049189483935034e-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 ( $\sim 0.521$ ).

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

City-mpg vs. Price

```
[51]: 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 -0.6865710067844681 with a P-value of P = 2.3211320655673725e-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

```
[52]: 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 0.8011176263981971 with a P-value of P = 3.046784581041982e-46

**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.

```
[ ]:
```