Exploratory-Data-Analysis.jupyterlite

June 1, 2022

1 Data Analysis with Python

Estimated time needed: **30** minutes

1.1 Objectives

After completing this lab you will be able to:

- Explore features or charecteristics to predict price of car

Table of Contents

Import Data from Module

Analyzing Individual Feature Patterns using Visualization

Descriptive Statistical Analysis

Basics of Grouping

Correlation and Causation

ANOVA

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

1. Import Data from Module 2

Setup

you are running the lab in your browser, so we will install the libraries using `piplite`

[1]:
```
#you are running the lab in your  browser, so we will install the libraries␣
↪using ``piplite``
import piplite
await piplite.install(['pandas'])
await piplite.install(['matplotlib'])
await piplite.install(['scipy'])
await piplite.install(['seaborn'])
```

Import libraries:

If you run the lab locally using Anaconda, you can load the correct library and versions by uncommenting the following:

[61]:
```
#If you run the lab locally using Anaconda, you can load the correct library␣
↪and versions by uncommenting the following:
#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
```

```
/lib/python3.9/site-packages/pandas/compat/__init__.py:124: UserWarning: Could
not import the lzma module. Your installed Python is incomplete. Attempting to
use lzma compression will result in a RuntimeError.
  warnings.warn(msg)
```

This function will download the dataset into your browser

[3]:
```
#This function will download the dataset into your browser

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())
```

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

This dataset was hosted on IBM Cloud object. Click HERE for free storage.

[4]:
```
path='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/
↪automobileEDA.csv'
```

you will need to download the dataset; if you are running locally, please comment out the following

#you will need to download the dataset; if you are running locally, please comment out the following await download(path, “auto.csv”) path=“auto.csv”

[5]:
```
await download(path, "auto.csv")
filename="auto.csv"
```

[6]:
```
df = pd.read_csv(filename)
df.head()
```

[6]:
```
   symboling  normalized-losses         make aspiration num-of-doors  \
0          3                122  alfa-romero        std          two
```

```
1          3                 122  alfa-romero        std          two
2          1                 122  alfa-romero        std          two
3          2                 164         audi        std         four
4          2                 164         audi        std         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  …

   compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
0                9.0       111.0    5000.0       21          27  13495.0
1                9.0       111.0    5000.0       21          27  16500.0
2                9.0       154.0    5000.0       19          26  16500.0
3               10.0       102.0    5500.0       24          30  13950.0
4                8.0       115.0    5500.0       18          22  17450.0

  city-L/100km  horsepower-binned  diesel  gas
0    11.190476             Medium       0    1
1    11.190476             Medium       0    1
2    12.368421             Medium       0    1
3     9.791667             Medium       0    1
4    13.055556             Medium       0    1

[5 rows x 29 columns]
```

2. 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.

[7]:
```
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.

[8]:
```
# 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
```

Question #1:

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

[9]:
```
# Write your code below and press Shift+Enter to execute
print(df['peak-rpm'].dtypes)
```

```
float64
```

Click here for the solution

```
float64
```

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

[10]:
```
df.corr()
```

[10]:
```
                   symboling  normalized-losses  wheel-base    length  \
symboling           1.000000           0.466264   -0.535987 -0.365404
normalized-losses   0.466264           1.000000   -0.056661  0.019424
```

wheel-base	-0.535987	-0.056661	1.000000	0.876024
length	-0.365404	0.019424	0.876024	1.000000
width	-0.242423	0.086802	0.814507	0.857170
height	-0.550160	-0.373737	0.590742	0.492063
curb-weight	-0.233118	0.099404	0.782097	0.880665
engine-size	-0.110581	0.112360	0.572027	0.685025
bore	-0.140019	-0.029862	0.493244	0.608971
stroke	-0.008245	0.055563	0.158502	0.124139
compression-ratio	-0.182196	-0.114713	0.250313	0.159733
horsepower	0.075819	0.217299	0.371147	0.579821
peak-rpm	0.279740	0.239543	-0.360305	-0.285970
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142
price	-0.082391	0.133999	0.584642	0.690628
city-L/100km	0.066171	0.238567	0.476153	0.657373
diesel	-0.196735	-0.101546	0.307237	0.211187
gas	0.196735	0.101546	-0.307237	-0.211187

	width	height	curb-weight	engine-size	bore \
symboling	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019
normalized-losses	0.086802	-0.373737	0.099404	0.112360	-0.029862
wheel-base	0.814507	0.590742	0.782097	0.572027	0.493244
length	0.857170	0.492063	0.880665	0.685025	0.608971
width	1.000000	0.306002	0.866201	0.729436	0.544885
height	0.306002	1.000000	0.307581	0.074694	0.180449
curb-weight	0.866201	0.307581	1.000000	0.849072	0.644060
engine-size	0.729436	0.074694	0.849072	1.000000	0.572609
bore	0.544885	0.180449	0.644060	0.572609	1.000000
stroke	0.188829	-0.062704	0.167562	0.209523	-0.055390
compression-ratio	0.189867	0.259737	0.156433	0.028889	0.001263
horsepower	0.615077	-0.087027	0.757976	0.822676	0.566936
peak-rpm	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392
city-mpg	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027
highway-mpg	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309
price	0.751265	0.135486	0.834415	0.872335	0.543155
city-L/100km	0.673363	0.003811	0.785353	0.745059	0.554610
diesel	0.244356	0.281578	0.221046	0.070779	0.054458
gas	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458

	stroke	compression-ratio	horsepower	peak-rpm \
symboling	-0.008245	-0.182196	0.075819	0.279740
normalized-losses	0.055563	-0.114713	0.217299	0.239543
wheel-base	0.158502	0.250313	0.371147	-0.360305
length	0.124139	0.159733	0.579821	-0.285970
width	0.188829	0.189867	0.615077	-0.245800
height	-0.062704	0.259737	-0.087027	-0.309974
curb-weight	0.167562	0.156433	0.757976	-0.279361

engine-size	0.209523	0.028889	0.822676	-0.256733
bore	-0.055390	0.001263	0.566936	-0.267392
stroke	1.000000	0.187923	0.098462	-0.065713
compression-ratio	0.187923	1.000000	-0.214514	-0.435780
horsepower	0.098462	-0.214514	1.000000	0.107885
peak-rpm	-0.065713	-0.435780	0.107885	1.000000
city-mpg	-0.034696	0.331425	-0.822214	-0.115413
highway-mpg	-0.035201	0.268465	-0.804575	-0.058598
price	0.082310	0.071107	0.809575	-0.101616
city-L/100km	0.037300	-0.299372	0.889488	0.115830
diesel	0.241303	0.985231	-0.169053	-0.475812
gas	-0.241303	-0.985231	0.169053	0.475812

	city-mpg	highway-mpg	price	city-L/100km	diesel \
symboling	-0.035527	0.036233	-0.082391	0.066171	-0.196735
normalized-losses	-0.225016	-0.181877	0.133999	0.238567	-0.101546
wheel-base	-0.470606	-0.543304	0.584642	0.476153	0.307237
length	-0.665192	-0.698142	0.690628	0.657373	0.211187
width	-0.633531	-0.680635	0.751265	0.673363	0.244356
height	-0.049800	-0.104812	0.135486	0.003811	0.281578
curb-weight	-0.749543	-0.794889	0.834415	0.785353	0.221046
engine-size	-0.650546	-0.679571	0.872335	0.745059	0.070779
bore	-0.582027	-0.591309	0.543155	0.554610	0.054458
stroke	-0.034696	-0.035201	0.082310	0.037300	0.241303
compression-ratio	0.331425	0.268465	0.071107	-0.299372	0.985231
horsepower	-0.822214	-0.804575	0.809575	0.889488	-0.169053
peak-rpm	-0.115413	-0.058598	-0.101616	0.115830	-0.475812
city-mpg	1.000000	0.972044	-0.686571	-0.949713	0.265676
highway-mpg	0.972044	1.000000	-0.704692	-0.930028	0.198690
price	-0.686571	-0.704692	1.000000	0.789898	0.110326
city-L/100km	-0.949713	-0.930028	0.789898	1.000000	-0.241282
diesel	0.265676	0.198690	0.110326	-0.241282	1.000000
gas	-0.265676	-0.198690	-0.110326	0.241282	-1.000000

	gas
symboling	0.196735
normalized-losses	0.101546
wheel-base	-0.307237
length	-0.211187
width	-0.244356
height	-0.281578
curb-weight	-0.221046
engine-size	-0.070779
bore	-0.054458
stroke	-0.241303
compression-ratio	-0.985231
horsepower	0.169053

```
peak-rpm            0.475812
city-mpg           -0.265676
highway-mpg        -0.198690
price              -0.110326
city-L/100km        0.241282
diesel             -1.000000
gas                 1.000000
```

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.

Hint: if you would like to select those columns, use the following syntax: df[['bore','stroke','compression-ratio','horsepower']]

[11]:
```
# Write your code below and press Shift+Enter to execute
df[['bore','stroke','compression-ratio','horsepower']].corr()
```

[11]:

	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

Click here for the solution

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

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.

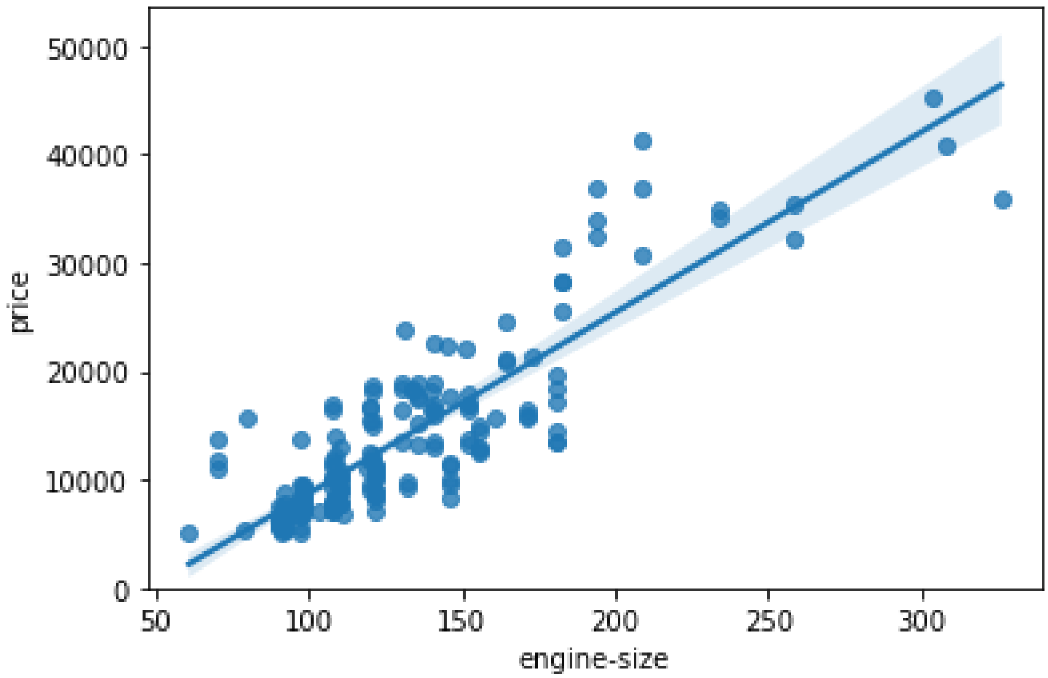
Let's see several examples of different linear relationships:

Positive Linear Relationship

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

[12]:
```
# Engine size as potential predictor variable of price
sns.regplot(x="engine-size", y="price", data=df)
plt.ylim(0,)
```

[12]: (0.0, 53587.760828598424)



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.

[13]:
```
df[["engine-size", "price"]].corr()
```

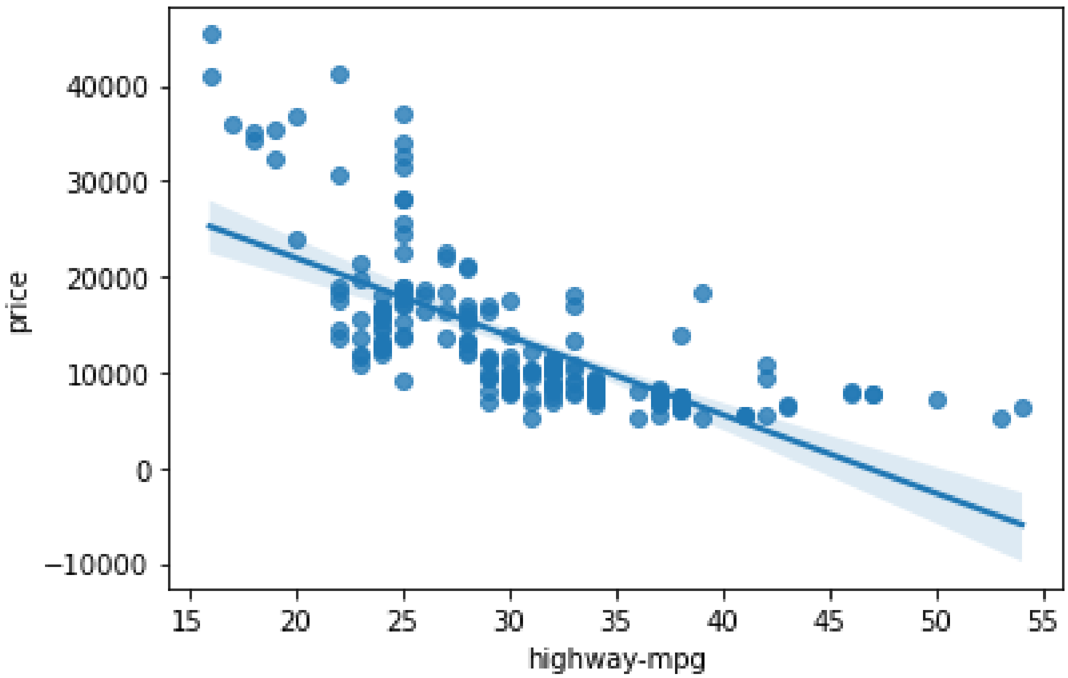
[13]:

	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".

[14]:
```
sns.regplot(x="highway-mpg", y="price", data=df)
```

[14]:
```
<AxesSubplot: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.

[15]:
```
df[['highway-mpg', 'price']].corr()
```

[15]:

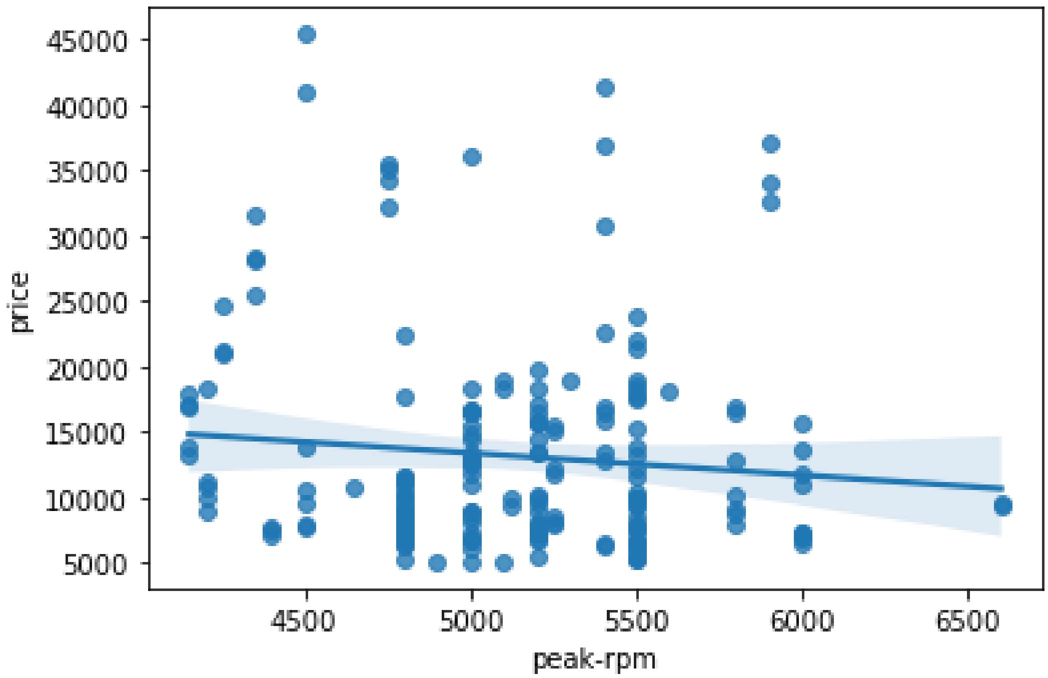
	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

Weak Linear Relationship

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

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

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



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

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

[17]:
```
df[['peak-rpm','price']].corr()
```

[17]:

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

Question 3 a):

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

Hint: if you would like to select those columns, use the following syntax: df[["stroke","price"]].

[18]:
```
# Write your code below and press Shift+Enter to execute
df[['stroke','price']].corr()
```

[18]:

	stroke	price
stroke	1.00000	0.08231
price	0.08231	1.00000

Click here for the solution

```
#The correlation is 0.0823, the non-diagonal elements of the table.

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

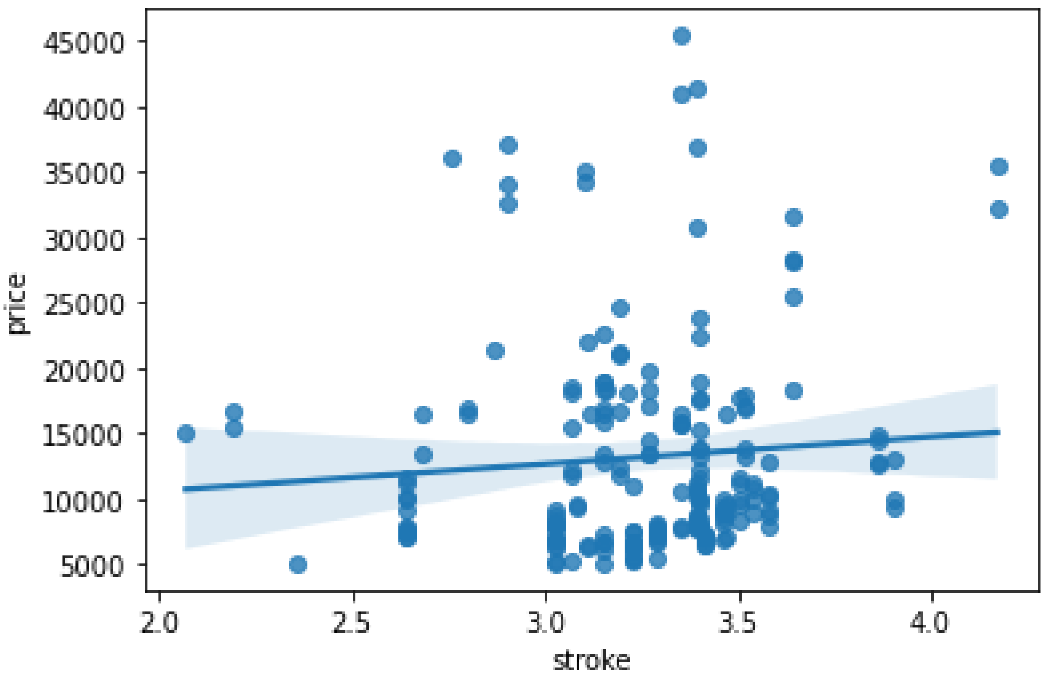
Question 3 b):

Given the correlation results between "price" and "stroke", do you expect a linear relationship?

Verify your results using the function "regplot()".

[20]:
```
# Write your code below and press Shift+Enter to execute
sns.regplot(x="stroke", y="price", data=df)
```

[20]:
```
<AxesSubplot:xlabel='stroke', ylabel='price'>
```



Click here for the solution

```
#There is a weak correlation between the variable 'stroke' and 'price.' as such regression wil

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

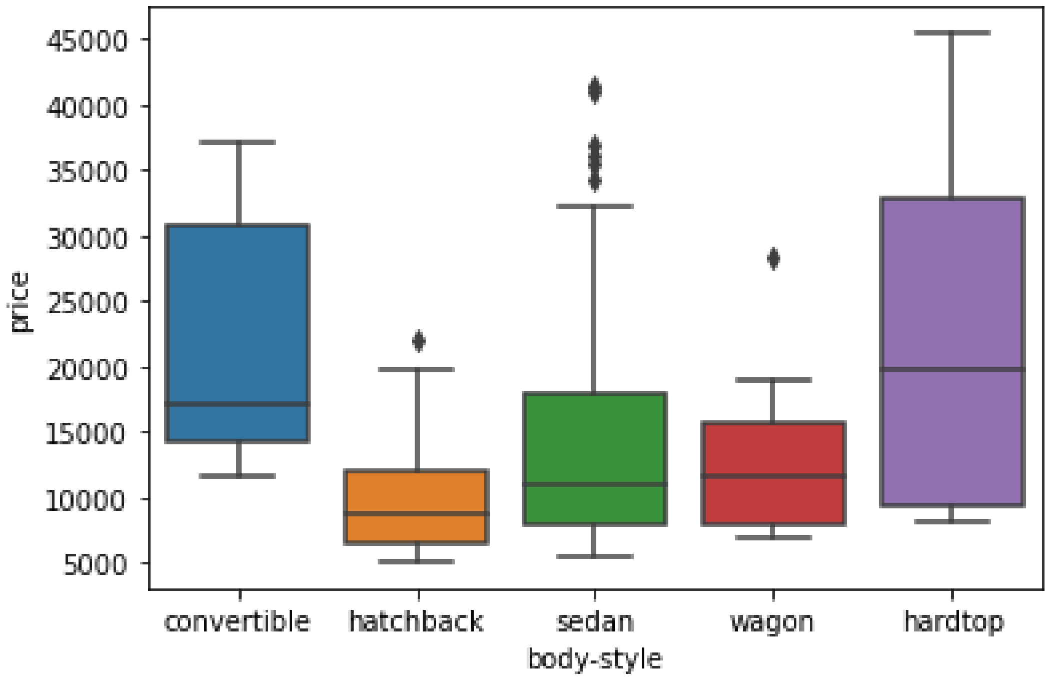
Categorical Variables

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".

[21]:
```
sns.boxplot(x="body-style", y="price", data=df)
```

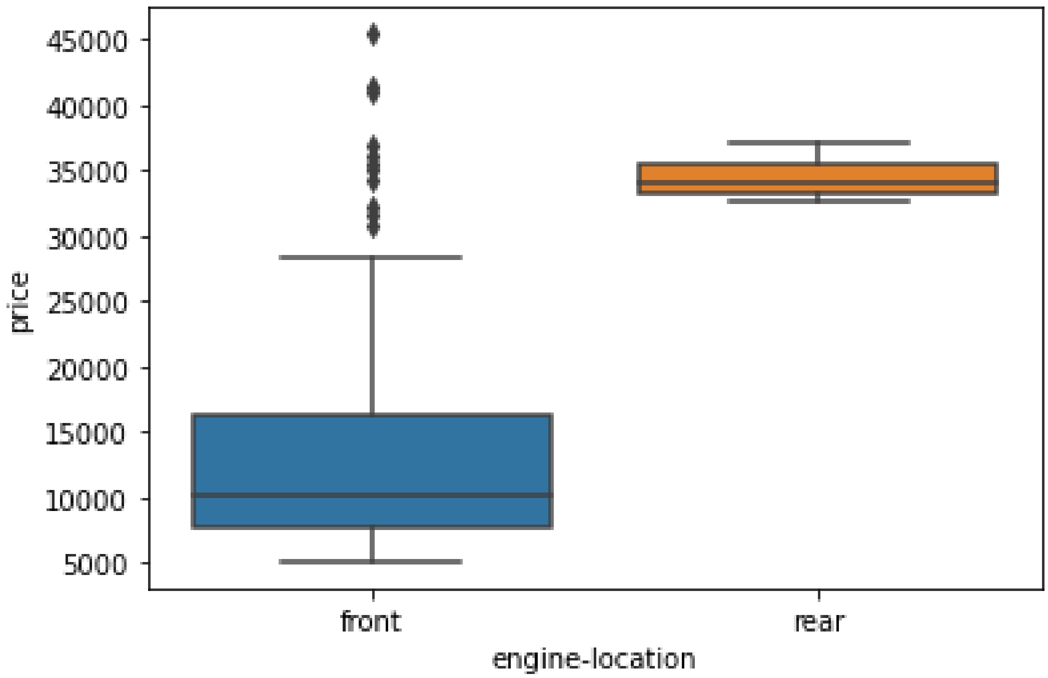
[21]: <AxesSubplot:xlabel='body-style', ylabel='price'>



We see that the distributions of price between the different body-style categories have a significant overlap, so body-style would not be a good predictor of price. Let's examine engine "engine-location" and "price":

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

[22]: <AxesSubplot:xlabel='engine-location', ylabel='price'>

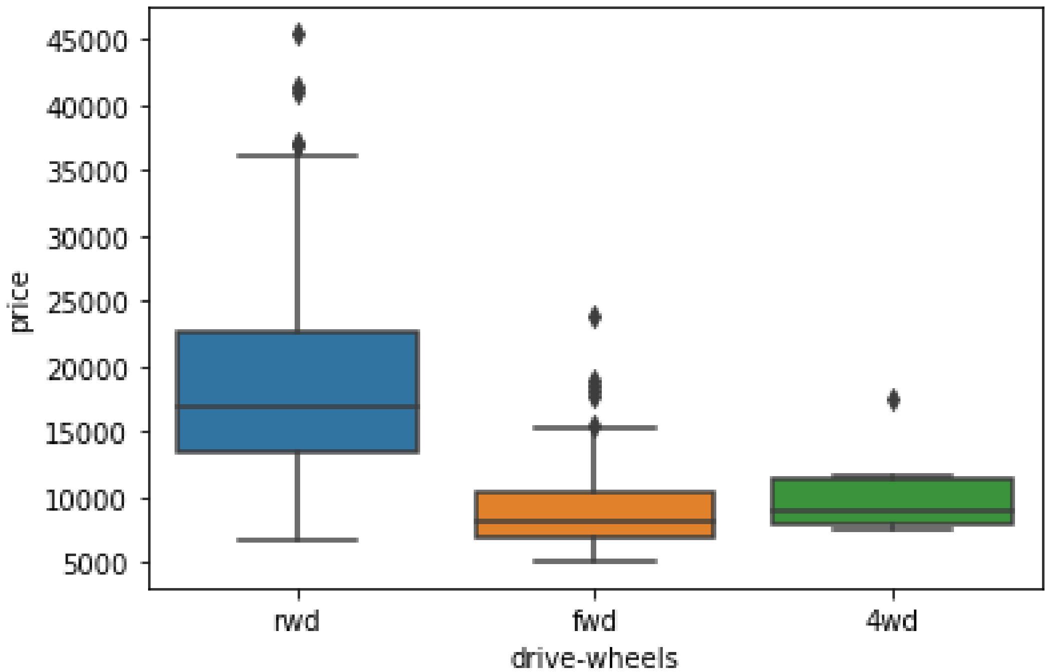


Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.

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

[23]:
```python
# drive-wheels
sns.boxplot(x="drive-wheels", y="price", data=df)
```

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



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

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:

[24]:
```
df.describe()
```

[24]:
```
        symboling  normalized-losses  wheel-base      length       width  \
count  201.000000          201.00000  201.000000  201.000000  201.000000
```

```
mean      0.840796          122.00000   98.797015    0.837102    0.915126
std       1.254802           31.99625    6.066366    0.059213    0.029187
min      -2.000000           65.00000   86.600000    0.678039    0.837500
25%       0.000000          101.00000   94.500000    0.801538    0.890278
50%       1.000000          122.00000   97.000000    0.832292    0.909722
75%       2.000000          137.00000  102.400000    0.881788    0.925000
max       3.000000          256.00000  120.900000    1.000000    1.000000

           height  curb-weight  engine-size        bore      stroke  \
count  201.000000   201.000000   201.000000  201.000000  197.000000
mean    53.766667  2555.666667   126.875622    3.330692    3.256904
std      2.447822   517.296727    41.546834    0.268072    0.319256
min     47.800000  1488.000000    61.000000    2.540000    2.070000
25%     52.000000  2169.000000    98.000000    3.150000    3.110000
50%     54.100000  2414.000000   120.000000    3.310000    3.290000
75%     55.500000  2926.000000   141.000000    3.580000    3.410000
max     59.800000  4066.000000   326.000000    3.940000    4.170000

       compression-ratio  horsepower      peak-rpm    city-mpg highway-mpg  \
count         201.000000  201.000000    201.000000  201.000000  201.000000
mean           10.164279  103.405534   5117.665368   25.179104   30.686567
std             4.004965   37.365700    478.113805    6.423220    6.815150
min             7.000000   48.000000   4150.000000   13.000000   16.000000
25%             8.600000   70.000000   4800.000000   19.000000   25.000000
50%             9.000000   95.000000   5125.369458   24.000000   30.000000
75%             9.400000  116.000000   5500.000000   30.000000   34.000000
max            23.000000  262.000000   6600.000000   49.000000   54.000000

              price  city-L/100km      diesel         gas
count    201.000000    201.000000  201.000000  201.000000
mean   13207.129353      9.944145    0.099502    0.900498
std     7947.066342      2.534599    0.300083    0.300083
min     5118.000000      4.795918    0.000000    0.000000
25%     7775.000000      7.833333    0.000000    1.000000
50%    10295.000000      9.791667    0.000000    1.000000
75%    16500.000000     12.368421    0.000000    1.000000
max    45400.000000     18.076923    1.000000    1.000000
```

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

[25]:
```python
df.describe(include=['object'])
```

[25]:
```
          make aspiration num-of-doors body-style drive-wheels  \
count      201        201          201        201          201
unique      22          2            2          5            3
top     toyota        std         four      sedan          fwd
```

```
freq        32       165          115         94           118

       engine-location engine-type num-of-cylinders fuel-system  \
count              201         201              201         201
unique               2           6                7           8
top              front         ohc             four        mpfi
freq               198         145              157          92

       horsepower-binned
count                200
unique                 3
top                  Low
freq                 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']].

[26]:
```python
df['drive-wheels'].value_counts()
```

[26]:
```
fwd    118
rwd     75
4wd      8
Name: drive-wheels, dtype: int64
```

We can convert the series to a dataframe as follows:

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

[27]:
```
     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'.

[28]:
```python
drive_wheels_counts = df['drive-wheels'].value_counts().to_frame()
drive_wheels_counts.rename(columns={'drive-wheels': 'value_counts'},␣
 ↪inplace=True)
drive_wheels_counts
```

[28]:
```
     value_counts
fwd           118
rwd            75
4wd             8
```

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

[29]:
```
drive_wheels_counts.index.name = 'drive-wheels'
drive_wheels_counts
```

[29]:
```
              value_counts
drive-wheels
fwd                    118
rwd                     75
4wd                      8
```

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

[30]:
```
# 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)
```

[30]:
```
                 value_counts
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.

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

[31]:
```
df['drive-wheels'].unique()
```

[31]:
```
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".

[32]:
```
df_group_one = df[['drive-wheels','body-style','price']]
```

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

[33]:
```
# grouping results
df_group_one = df_group_one.groupby(['drive-wheels'],as_index=False).mean()
df_group_one
```

[33]:

	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'.

[34]:
```
# 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
```

[34]:

	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:

[35]:
```
grouped_pivot = grouped_test1.pivot(index='drive-wheels',columns='body-style')
grouped_pivot
```

[35]:
```
                     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.

[36]:
```
grouped_pivot = grouped_pivot.fillna(0) #fill missing values with 0
grouped_pivot
```

[36]:
```
                     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
```

Question 4:

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

[37]:
```
# Write your code below and press Shift+Enter to execute
# grouping results
df_gptest2 = df[['body-style','price']]
grouped_test_bodystyle = df_gptest2.groupby(['body-style'],as_index= False).
 ↪mean()
grouped_test_bodystyle
```

[37]:
```
    body-style         price
0  convertible  21890.500000
```

```
1      hardtop  22208.500000
2    hatchback   9957.441176
3        sedan  14459.755319
4        wagon  12371.960000
```

Click here for the solution

```
# grouping results
df_gptest2 = df[['body-style','price']]
grouped_test_bodystyle = df_gptest2.groupby(['body-style'],as_index= False).mean()
grouped_test_bodystyle
```

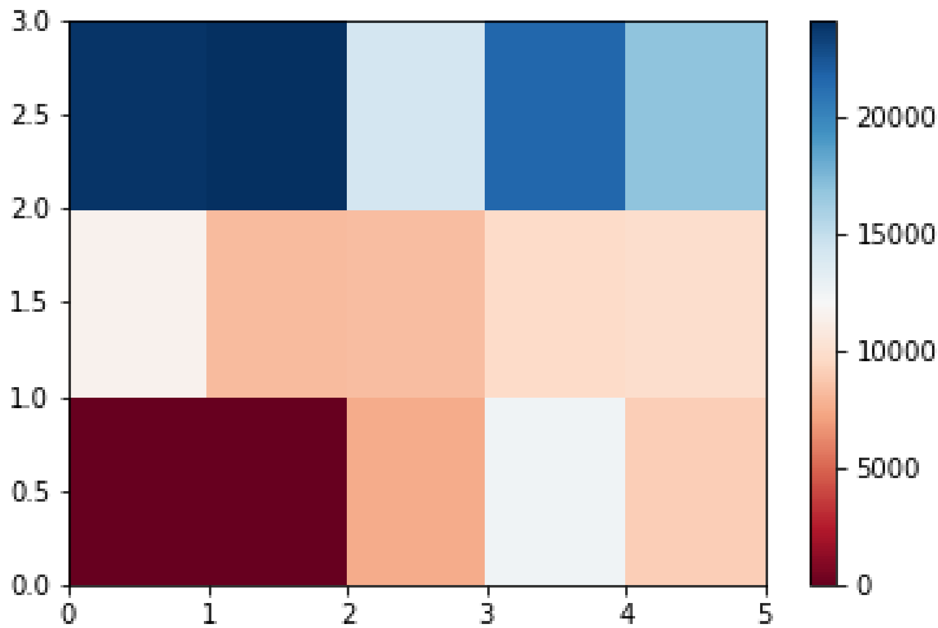
If you did not import "pyplot", let's do it again.

[38]:
```
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.

[39]:
```
#use the grouped results
plt.pcolor(grouped_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```

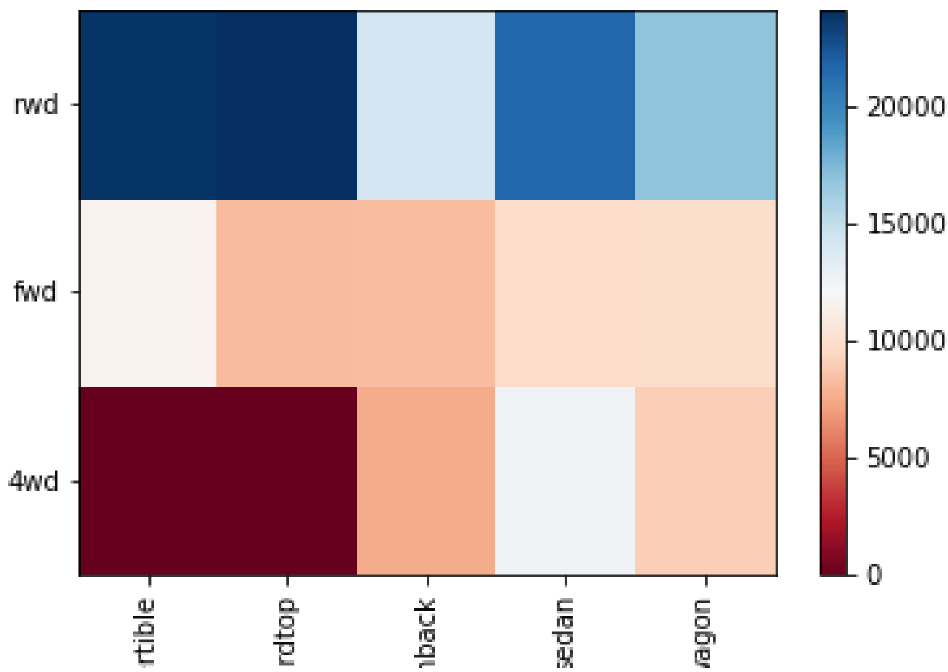


```
<Figure size 432x288 with 0 Axes>
```

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:

[40]:
```
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()
```



`<Figure size 432x288 with 0 Axes>`

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?

5. 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.

[41]:
```
df.corr()
```

[41]:

	symboling	normalized-losses	wheel-base	length
symboling	1.000000	0.466264	-0.535987	-0.365404
normalized-losses	0.466264	1.000000	-0.056661	0.019424
wheel-base	-0.535987	-0.056661	1.000000	0.876024
length	-0.365404	0.019424	0.876024	1.000000
width	-0.242423	0.086802	0.814507	0.857170
height	-0.550160	-0.373737	0.590742	0.492063
curb-weight	-0.233118	0.099404	0.782097	0.880665
engine-size	-0.110581	0.112360	0.572027	0.685025
bore	-0.140019	-0.029862	0.493244	0.608971
stroke	-0.008245	0.055563	0.158502	0.124139
compression-ratio	-0.182196	-0.114713	0.250313	0.159733
horsepower	0.075819	0.217299	0.371147	0.579821
peak-rpm	0.279740	0.239543	-0.360305	-0.285970
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142
price	-0.082391	0.133999	0.584642	0.690628
city-L/100km	0.066171	0.238567	0.476153	0.657373
diesel	-0.196735	-0.101546	0.307237	0.211187
gas	0.196735	0.101546	-0.307237	-0.211187

	width	height	curb-weight	engine-size	bore
symboling	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019
normalized-losses	0.086802	-0.373737	0.099404	0.112360	-0.029862
wheel-base	0.814507	0.590742	0.782097	0.572027	0.493244
length	0.857170	0.492063	0.880665	0.685025	0.608971
width	1.000000	0.306002	0.866201	0.729436	0.544885
height	0.306002	1.000000	0.307581	0.074694	0.180449
curb-weight	0.866201	0.307581	1.000000	0.849072	0.644060
engine-size	0.729436	0.074694	0.849072	1.000000	0.572609
bore	0.544885	0.180449	0.644060	0.572609	1.000000
stroke	0.188829	-0.062704	0.167562	0.209523	-0.055390
compression-ratio	0.189867	0.259737	0.156433	0.028889	0.001263
horsepower	0.615077	-0.087027	0.757976	0.822676	0.566936
peak-rpm	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392
city-mpg	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027
highway-mpg	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309
price	0.751265	0.135486	0.834415	0.872335	0.543155
city-L/100km	0.673363	0.003811	0.785353	0.745059	0.554610
diesel	0.244356	0.281578	0.221046	0.070779	0.054458
gas	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458

	stroke	compression-ratio	horsepower	peak-rpm	\
symboling	-0.008245	-0.182196	0.075819	0.279740	
normalized-losses	0.055563	-0.114713	0.217299	0.239543	
wheel-base	0.158502	0.250313	0.371147	-0.360305	
length	0.124139	0.159733	0.579821	-0.285970	
width	0.188829	0.189867	0.615077	-0.245800	
height	-0.062704	0.259737	-0.087027	-0.309974	
curb-weight	0.167562	0.156433	0.757976	-0.279361	
engine-size	0.209523	0.028889	0.822676	-0.256733	
bore	-0.055390	0.001263	0.566936	-0.267392	
stroke	1.000000	0.187923	0.098462	-0.065713	
compression-ratio	0.187923	1.000000	-0.214514	-0.435780	
horsepower	0.098462	-0.214514	1.000000	0.107885	
peak-rpm	-0.065713	-0.435780	0.107885	1.000000	
city-mpg	-0.034696	0.331425	-0.822214	-0.115413	
highway-mpg	-0.035201	0.268465	-0.804575	-0.058598	
price	0.082310	0.071107	0.809575	-0.101616	
city-L/100km	0.037300	-0.299372	0.889488	0.115830	
diesel	0.241303	0.985231	-0.169053	-0.475812	
gas	-0.241303	-0.985231	0.169053	0.475812	

	city-mpg	highway-mpg	price	city-L/100km	diesel	\
symboling	-0.035527	0.036233	-0.082391	0.066171	-0.196735	
normalized-losses	-0.225016	-0.181877	0.133999	0.238567	-0.101546	
wheel-base	-0.470606	-0.543304	0.584642	0.476153	0.307237	
length	-0.665192	-0.698142	0.690628	0.657373	0.211187	
width	-0.633531	-0.680635	0.751265	0.673363	0.244356	
height	-0.049800	-0.104812	0.135486	0.003811	0.281578	
curb-weight	-0.749543	-0.794889	0.834415	0.785353	0.221046	
engine-size	-0.650546	-0.679571	0.872335	0.745059	0.070779	
bore	-0.582027	-0.591309	0.543155	0.554610	0.054458	
stroke	-0.034696	-0.035201	0.082310	0.037300	0.241303	
compression-ratio	0.331425	0.268465	0.071107	-0.299372	0.985231	
horsepower	-0.822214	-0.804575	0.809575	0.889488	-0.169053	
peak-rpm	-0.115413	-0.058598	-0.101616	0.115830	-0.475812	
city-mpg	1.000000	0.972044	-0.686571	-0.949713	0.265676	
highway-mpg	0.972044	1.000000	-0.704692	-0.930028	0.198690	
price	-0.686571	-0.704692	1.000000	0.789898	0.110326	
city-L/100km	-0.949713	-0.930028	0.789898	1.000000	-0.241282	
diesel	0.265676	0.198690	0.110326	-0.241282	1.000000	
gas	-0.265676	-0.198690	-0.110326	0.241282	-1.000000	

	gas
symboling	0.196735
normalized-losses	0.101546
wheel-base	-0.307237

```
length              -0.211187
width               -0.244356
height              -0.281578
curb-weight         -0.221046
engine-size         -0.070779
bore                -0.054458
stroke              -0.241303
compression-ratio -0.985231
horsepower           0.169053
peak-rpm             0.475812
city-mpg            -0.265676
highway-mpg         -0.198690
price               -0.110326
city-L/100km         0.241282
diesel              -1.000000
gas                  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.

[42]:
```
from scipy import stats
```

Wheel-Base vs. Price

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

[43]:
```
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 0.5846418222655085  with a P-value of P =
8.076488270732243e-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'.

[44]:
```
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 0.8095745670036559  with a P-value of P =
6.369057428260101e-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 (~0.809, close to 1).

Length vs. Price

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

[45]:
```
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 0.6906283804483643  with a P-value of P =
8.01647746615853e-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':

[46]:
```
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 0.7512653440522666  with a P-value of P =
9.200335510483739e-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).

1.1.1 Curb-Weight vs. Price

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

[47]:
```
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 0.8344145257702845  with a P-value of P =
2.189577238893816e-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':

[107]:
```
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 0.8723351674455188  with a P-value of P =
9.265491622196808e-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':

[108]:
```
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 0.54315538326266  with a P-value of P =
8.049189483935489e-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

[109]:
```
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 -0.6865710067844684  with a P-value of P
=  2.3211320655672453e-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

[110]:
```
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 -0.7046922650589534  with a P-value of P
=  1.749547114447437e-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.

6. ANOVA

ANOVA: Analysis of Variance

The Analysis of Variance (ANOVA) is a statistical method used to test whether there are significant differences between the means of two or more groups. ANOVA returns two parameters:

F-test score: ANOVA assumes the means of all groups are the same, calculates how much the actual means deviate from the assumption, and reports it as the F-test score. A larger score means there is a larger difference between the means.

P-value: P-value tells how statistically significant our calculated score value is.

If our price variable is strongly correlated with the variable we are analyzing, we expect ANOVA to return a sizeable F-test score and a small p-value.

Drive Wheels

Since ANOVA analyzes the difference between different groups of the same variable, the groupby function will come in handy. Because the ANOVA algorithm averages the data automatically, we do not need to take the average before hand.

To see if different types of 'drive-wheels' impact 'price', we group the data.

[111]:
```
grouped_test2=df_gptest[['drive-wheels', 'price']].groupby(['drive-wheels'])
grouped_test2.head(2)
```

[111]:

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

[112]:
```
df_gptest
```

[112]:
```
    drive-wheels   body-style    price
0            rwd  convertible  13495.0
1            rwd  convertible  16500.0
2            rwd    hatchback  16500.0
3            fwd        sedan  13950.0
4            4wd        sedan  17450.0
..           …          …        …
196          rwd        sedan  16845.0
197          rwd        sedan  19045.0
198          rwd        sedan  21485.0
199          rwd        sedan  22470.0
200          rwd        sedan  22625.0

[201 rows x 3 columns]
```

We can obtain the values of the method group using the method "get_group".

[113]:
```
grouped_test2.get_group('4wd')['price']
```

[113]:
```
4      17450.0
136     7603.0
140     9233.0
141    11259.0
144     8013.0
145    11694.0
150     7898.0
151     8778.0
Name: price, dtype: float64
```

We can use the function 'f_oneway' in the module 'stats' to obtain the F-test score and P-value.

[114]:
```
# ANOVA
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'],␣
 ↪grouped_test2.get_group('rwd')['price'], grouped_test2.
 ↪get_group('4wd')['price'])

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

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

This is a great result with a large F-test score showing a strong correlation and a P-value of almost 0 implying almost certain statistical significance. But does this mean all three tested groups are all this highly correlated?

Let's examine them separately.

fwd and rwd

[115]:
```python
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'],␣
↪grouped_test2.get_group('rwd')['price'])

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

```
ANOVA results: F= 130.5533160959111 , P = 2.2355306355677845e-23
```

Let's examine the other groups.

4wd and rwd

[116]:
```python
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'],␣
↪grouped_test2.get_group('rwd')['price'])

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

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

4wd and fwd

[117]:
```python
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'],␣
↪grouped_test2.get_group('fwd')['price'])

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

```
ANOVA results: F= 0.665465750252303 , P = 0.41620116697845655
```

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.

1.1.2 Thank you for completing this lab!

1.2 Author

Joseph Santarcangelo

1.2.1 Other Contributors

Mahdi Noorian PhD

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver

Yi Yao.

1.3 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-30	2.1	Lakshmi	changed URL of csv
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

##

[ ]:

[ ]: