



Data Analysis with Python

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Explore features or characteristics to predict price of car

Table of Contents

1. Import Data from Module
2. Analyzing Individual Feature Patterns using Visualization
3. Descriptive Statistical Analysis
4. Basics of Grouping
5. Correlation and Causation
6. ANOVA

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

1. Import Data from Module 2

Setup

Import libraries:

```
In [6]: 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. Click [HERE](#) for free storage.

```
In [7]: path='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101-Notes/data-files/taxii.csv'
df = pd.read_csv(path)
df.head()
```

```
Out[7]:
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.8111
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.8111
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.8226
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.8486
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.8486

5 rows × 29 columns

2. Analyzing Individual Feature Patterns Using Visualization

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

```
In [8]: %%capture
! pip install seaborn
```

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

```
In [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.

```
In [10]: # 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"?

```

In [11]: # Write your code below and press Shift+Enter to execute
print(df["peak-rpm"].dtypes)

```

float64

► [Click here for the solution](#)

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

```

In [12]: df.corr()

```

```

Out[12]:

```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engir s
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.1105
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.1123
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.5720
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.6850
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.7294
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.0746
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.8490

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.0000
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.5726
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.2095
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.0288
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.8226
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.2567
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.6505
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.6795
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.8723
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.7450
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.0707
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.0707

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']]`

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

```
Out[13]:
```

	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](#)

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.

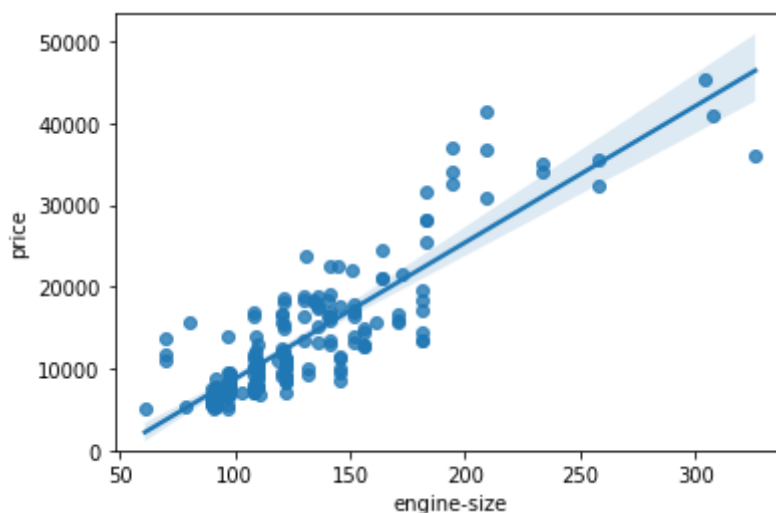
Let's see several examples of different linear relationships:

Positive Linear Relationship

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

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

```
Out[14]: (0.0, 53375.49323603402)
```



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

```
Out[15]:
```

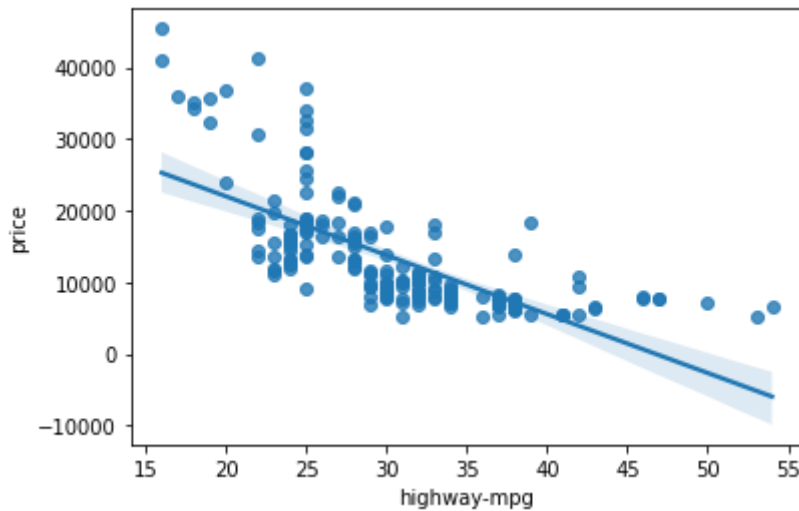
	engine-size	price
engine-size	1.000000	0.872335
price	0.872335	1.000000

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

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

<AxesSubplot:xlabel='highway-mpg', ylabel='price'>
```

Out[16]:



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

```
Out[17]:
```

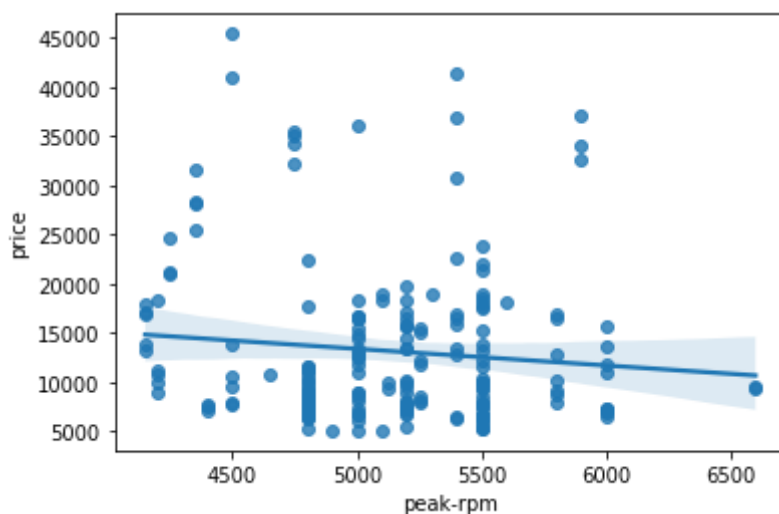
	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

Weak Linear Relationship

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

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

```
Out[18]: <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.

```
In [22]: df[['peak-rpm', 'price']].corr()
```

```
Out[22]:
```

	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"]].

```
In [27]: # Write your code below and press Shift+Enter to execute

#The correlation is 0.0823, the non-diagonal elements of the table.
df[["stroke", "price"]].corr()
```

```
Out[27]:
```

	stroke	price
stroke	1.00000	0.08231
price	0.08231	1.00000

► [Click here for the solution](#)

Question 3 b):

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

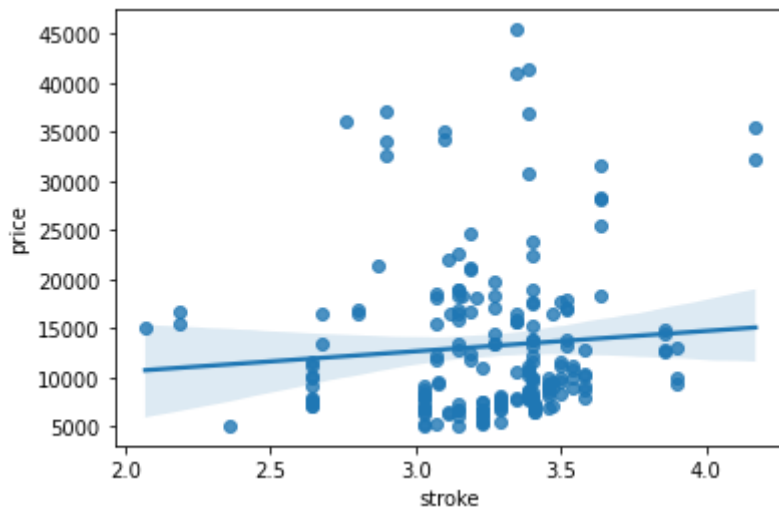
Verify your results using the function "regplot()".

```
In [28]: # Write your code below and press Shift+Enter to execute

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

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

```
Out[28]: <AxesSubplot:xlabel='stroke', ylabel='price'>
```



► [Click here for the solution](#)

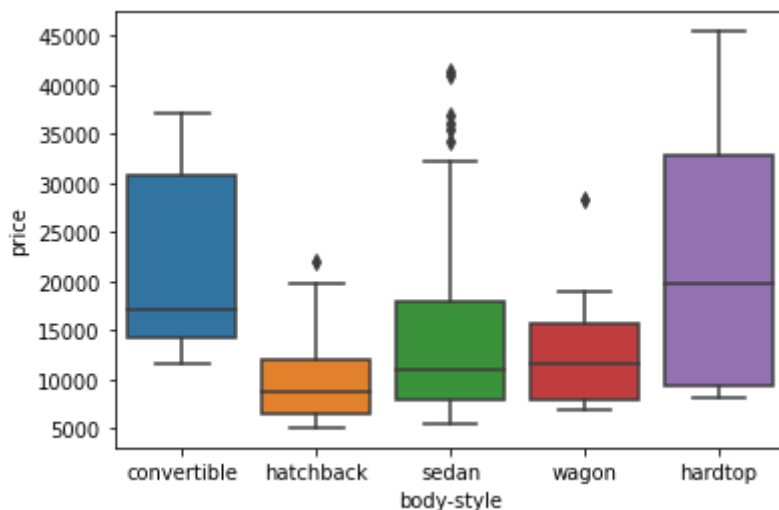
Categorical Variables

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

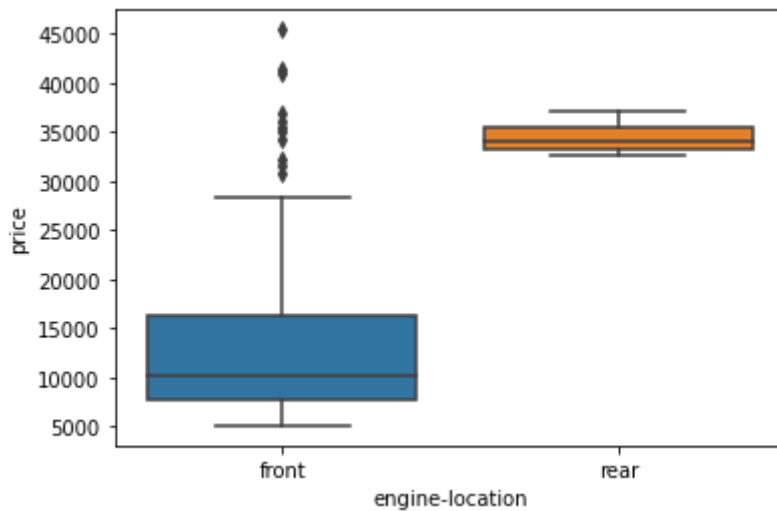
```
Out[29]: <AxesSubplot:xlabel='body-style', ylabel='price'>
```



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":

```
In [30]: sns.boxplot(x="engine-location", y="price", data=df)
```

```
Out[30]: <AxesSubplot:xlabel='engine-location', ylabel='price'>
```

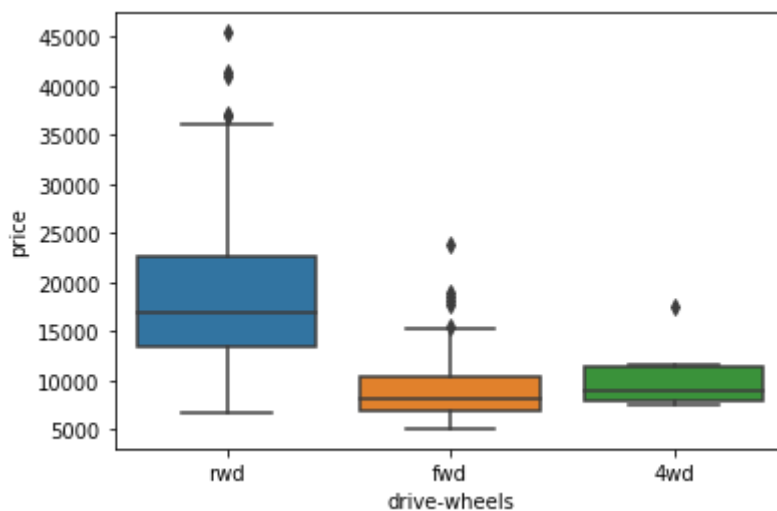



Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.

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

```
In [31]: # drive-wheels
sns.boxplot(x="drive-wheels", y="price", data=df)
```

```
Out[31]: <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:

In [32]:

```
df.describe()
```

Out[32]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	eng
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.00
mean	0.840796	122.00000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.87
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.296727	41.54
min	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.00
25%	0.000000	101.00000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.00
50%	1.000000	122.00000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.00
75%	2.000000	137.00000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.00
max	3.000000	256.00000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.00

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 [33]:

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

Out[33]:

	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system	horsepower
count	201	201	201	201	201	201	201	201	201	201
unique	22	2	2	5	3	2	6	7	8	11
top	toyota	std	four	sedan	fwd	front	ohc	four	mpfi	Low
freq	32	165	115	94	118	198	145	157	92	111

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 [34]:

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

Out[34]:

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

We can convert the series to a dataframe as follows:

```
In [35]: df['drive-wheels'].value_counts().to_frame()
```

```
Out[35]:
```

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 [36]: 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[36]:
```

value_counts	
fwd	118
rwd	75
4wd	8

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

```
In [37]: drive_wheels_counts.index.name = 'drive-wheels'
drive_wheels_counts
```

```
Out[37]:
```

value_counts	
drive-wheels	
fwd	118
rwd	75
4wd	8

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

```
In [38]: # 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[38]:
```

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.

```
In [39]: df['drive-wheels'].unique()

Out[39]: 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 [40]: df_group_one = df[['drive-wheels', 'body-style', 'price']]
```

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

```
In [41]: # grouping results
df_group_one = df_group_one.groupby(['drive-wheels'], as_index=False).mean()
df_group_one
```

```
Out[41]:
```

	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'.

```
In [42]: # 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
```

```
Out[42]:
```

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333

	drive-wheels	body-style	price
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:

```
In [43]: grouped_pivot = grouped_test1.pivot(index='drive-wheels', columns='body-style')
grouped_pivot
```

```
Out[43]:
```

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

```
Out[44]:
```

		price				
	body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels						
4wd		0.0	0.000000	7603.000000	12647.333333	9095.750000

						price
body-style	convertible	hardtop	hatchback	sedan	wagon	
drive-wheels						
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333	
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222	

Question 4:

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

```
In [45]: # Write your code below and press Shift+Enter to execute
df_gptest2 = df[['body-style', 'price']]
grouped_test_bodystyle = df_gptest2.groupby(['body-style'], as_index=False).mean()
grouped_test_bodystyle
```

```
Out[45]:
```

	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](#)

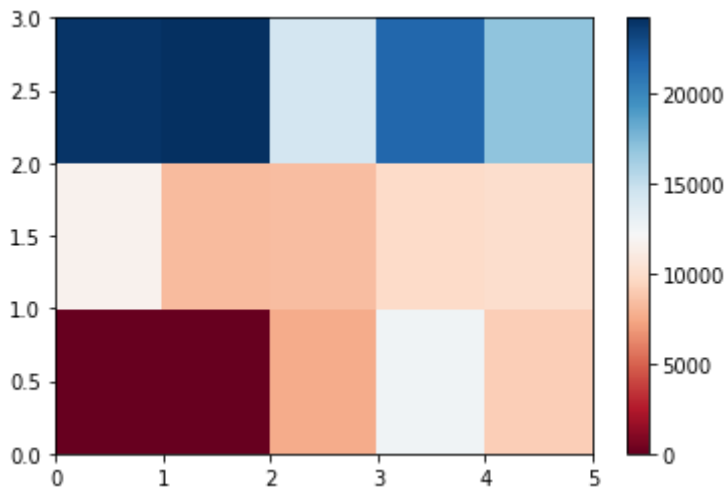
If you did not import "pyplot", let's do it again.

```
In [46]: 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 [47]: #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 [48]: fig, ax = plt.subplots()
im = ax.pcolor(grouped_pivot, cmap='RdBu')

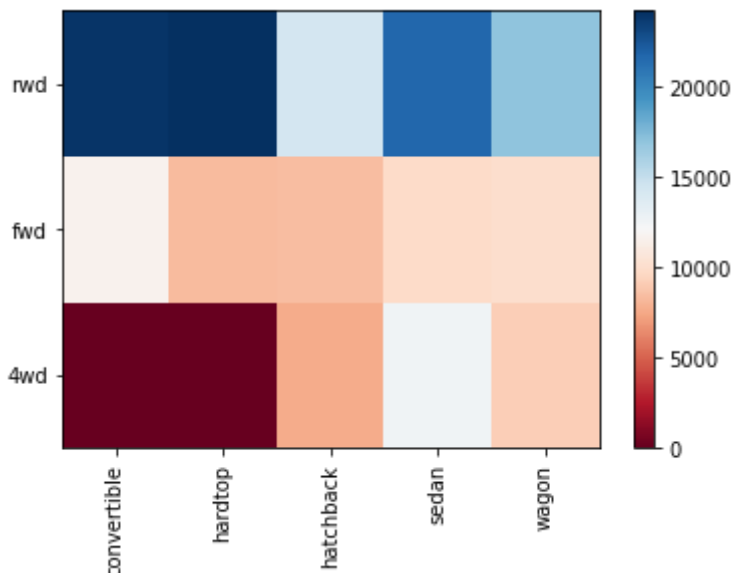
#Label names
row_labels = grouped_pivot.columns.levels[1]
col_labels = grouped_pivot.index

#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?

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.

In [49]:

```
df.corr()
```

Out[49]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572027
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209500
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028847

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.8226
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.2567
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.6505
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.6795
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.8723
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.7450
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.0707
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.0707

Sometimes we would like to know the significance 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.

```
In [50]: from scipy import stats
```

Wheel-Base vs. Price

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

```
In [51]: 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
```

```
The Pearson Correlation Coefficient is 0.584641822265508 with a P-value of P = 8.076488270733218e-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 [52]: pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P
```

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'.

```
In [53]: pearson_coef, p_value = stats.pearsonr(df['length'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P
```

The Pearson Correlation Coefficient is 0.6906283804483638 with a P-value of P = 8.016477466159556e-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 [54]: pearson_coef, p_value = stats.pearsonr(df['width'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P
```

The Pearson Correlation Coefficient is 0.7512653440522673 with a P-value of P = 9.200335510481646e-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 [55]: 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
```

The Pearson Correlation Coefficient is 0.8344145257702843 with a P-value of P = 2.189577238894065e-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 [56]: 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
```

The Pearson Correlation Coefficient is 0.8723351674455185 with a P-value of P = 9.265491622198389e-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 [57]: pearson_coef, p_value = stats.pearsonr(df['bore'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P
```

The Pearson Correlation Coefficient is 0.5431553832626602 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

```
In [58]: 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
```

The Pearson Correlation Coefficient is -0.6865710067844678 with a P-value of P = 2.321132065567641e-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 [59]: 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
```

The Pearson Correlation Coefficient is -0.704692265058953 with a P-value of P = 1.7495471144476358e-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.

```
In [60]: grouped_test2=df_gptest[['drive-wheels', 'price']].groupby(['drive-wheels'])
grouped_test2.head(2)
```

```
Out[60]:
```

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

```
In [61]: df_gptest
```

```
Out[61]:
```

	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

	drive-wheels	body-style	price
...
196	rwd	sedan	16845.0
197	rwd	sedan	19045.0
198	rwd	sedan	21485.0
199	rwd	sedan	22470.0
200	rwd	sedan	22625.0

201 rows × 3 columns

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

```
In [62]: grouped_test2.get_group('4wd')['price']
```

```
Out[62]: 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**.

```
In [63]: # ANOVA
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'], grouped_test2
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

```
In [64]: f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'], grouped_test2
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

```
In [65]: f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'], grouped_test2
print( "ANOVA results: F=", f_val, ", P =", p_val)
```

ANOVA results: F= 8.580681368924756 , P = 0.004411492211225333

4wd and fwd

```
In [66]: f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'], grouped_test2
print("ANOVA results: F=", f_val, ", P =", p_val)
```

ANOVA results: F= 0.665465750252303 , P = 0.41620116697845666

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.

Thank you for completing this lab!

Author

[Joseph Santarcangelo](#)

Other Contributors

[Mahdi Noorian PhD](#)

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

[Fiorella Wenver](#)[Yi Yao.](#)

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

© IBM Corporation 2020. All rights reserved.

PLUS

Heatmap with all columns

In [73]:

```
df_corr = df.corr()
df_corr
```

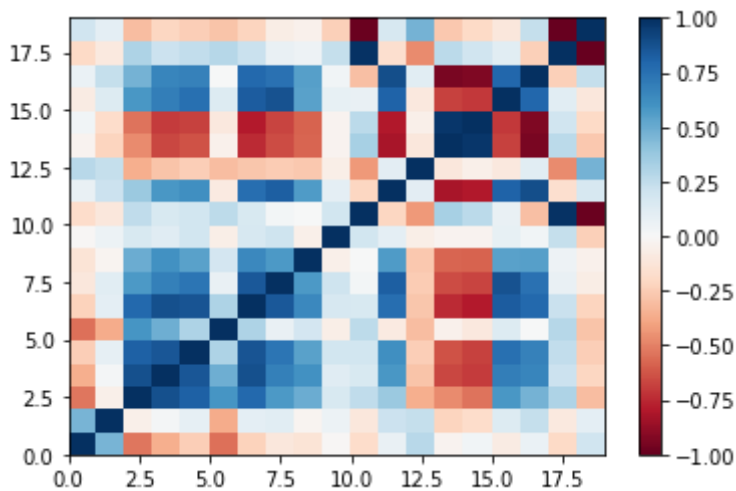
Out[73]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572603
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209501
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028848
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822604
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256738
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650505
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679513

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engir s
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.8723
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.7450
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.0707
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.0707

In [83]:

```
# Heatmap
plt.pcolor(df_corr, cmap='RdBu')
plt.colorbar()
plt.show()
```



In [84]:

```
# Heatmap with names
fig, ax = plt.subplots()
im = ax.pcolor(df_corr, cmap='RdBu')

#Label names
row_labels = df_corr.columns
col_labels = df_corr.index

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