

+

Name Last Modified

- clean_df.csv a day ago
- data-wrangling.ipynb a day ago
- exploratory-data-analysis.ipynb a minute ago
- review-introduction.ipynb a day ago

Launcher

exploratory-data-analysis.ipynb

Python

Simplifying AI and Machine-Learning with Watson Studio

- Get your free account and use the Lite plan forever
- No credit card and no autorenewals

[Click Here](#)

Support



Data Analysis with Python

Exploratory Data Analysis

Welcome!

In this section, we will explore several methods to see if certain characteristics or features can be used to predict car price.

Table of content

- Import Data from Module
- Analyzing Individual Feature Patterns using Visualization
- Descriptive Statistical Analysis
- Basics of Grouping
- Correlation and Causation
- ANOVA

Estimated Time Needed: 30 min

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

1. Import Data from Module 2

Setup

Import libraries

```
[1]: import pandas as pd
import numpy as np
```

load data and store in dataframe df:

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

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

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

5 rows x 29 columns

2. Analyzing Individual Feature Patterns using Visualization

To install seaborn we use the pip which is the python package manager.

```
[3]: %capture
! pip install seaborn
```

Support

Support

Support

Support

Support

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

```
[4]: 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.

```
[5]: # 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"?

Double-click [here](#) for the solution.

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

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

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.008245	-0.182196	0.075819	0.279740	-0.035527	0.036
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029862	0.055563	-0.114713	0.217299	0.239543	-0.225016	-0.181
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.158502	0.250313	0.371147	-0.360305	-0.470606	-0.543
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608971	0.124139	0.159733	0.579821	-0.285970	-0.665192	-0.698
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544885	0.188829	0.189867	0.615077	-0.245800	-0.633531	-0.680
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180449	-0.062704	0.259737	-0.087027	-0.309974	-0.049800	-0.104
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072	0.644060	0.167562	0.156433	0.757976	-0.279361	-0.749543	-0.794
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572609	0.209523	0.028889	0.822676	-0.256733	-0.650546	-0.679
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.000000	-0.055390	0.001263	0.566936	-0.267392	-0.582027	-0.551
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209523	-0.055390	1.000000	0.187923	0.098462	-0.065713	-0.034696	-0.001
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187923	1.000000	-0.214514	-0.435780	0.331425	0.268
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098462	-0.214514	1.000000	0.107885	-0.822214	-0.804
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.065713	-0.435780	0.107885	1.000000	-0.115413	-0.058
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027	-0.034696	0.331425	-0.822214	-0.115413	1.000000	0.972
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309	-0.035201	0.268465	-0.804575	-0.058598	0.972044	1.000
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	0.071107	0.809575	-0.101616	-0.686571	-0.704
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.745059	0.554610	0.037300	-0.299372	0.889488	0.115830	-0.949713	-0.930
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	0.241303	0.985231	-0.169053	-0.475812	0.265676	0.198
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241303	-0.985231	0.169053	0.475812	-0.265676	-0.198

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']]

```
[27]: # Write your code below and press Shift+Enter to execute
df_corr_1 = df[['bore','stroke','compression-ratio','horsepower']]
df_corr_1.corr()
```

	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

Double-click **here** for the solution.

<!-- The answer is below:

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

-->

Continuous numerical variables:

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

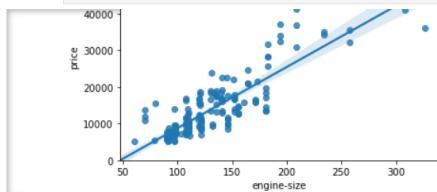
In order to start understanding the (linear) relationship between an individual variable and the price. We can do this by using "regplot", which plots the `scatterplot` plus the fitted regression line for the data.

Let's see several examples of different linear relationships:

Positive linear relationship

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

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



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 it's approximately 0.87

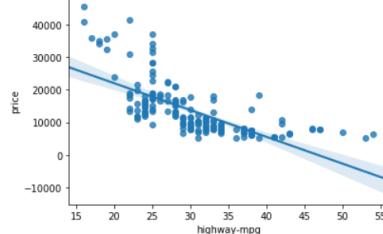
```
[31]: df[['engine-size', 'price']].corr()
```

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

Highway mpg is a potential predictor variable of price

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

```
[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9894b18eb8>
```



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

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

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

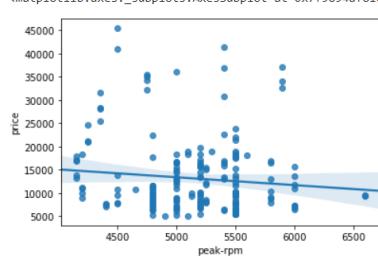
	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

Weak Linear Relationship

Let's see if "Peak-rpm" as a predictor variable of "price".

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

```
[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9894af8160>
```



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

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

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

Question 3 a):

Find the correlation between $x = \text{"stroke"}$, $y = \text{"price"}$.

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

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

	stroke	price
stroke	1.000000	0.08231
price	0.08231	1.000000

Double-click [here](#) for the solution.

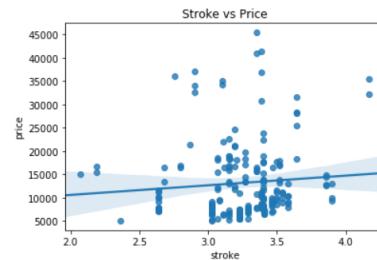
Question 3 b):

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

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

```
[41]: # Write your code below and press Shift+Enter to execute
sns.regplot(x = 'stroke', y = 'price', data=df)
plt.title ('Stroke vs Price')
```

```
[41]: Text(0.5, 1.0, 'Stroke vs Price')
```



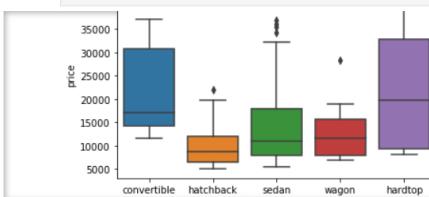
Double-click [here](#) for the solution.

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.

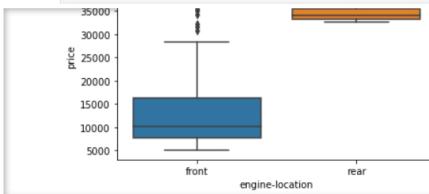
Let's look at the relationship between "body-style" and "price".

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



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

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



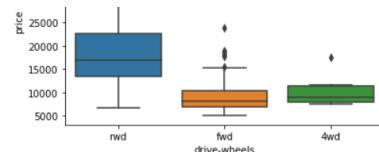
Here we see that the distribution of price between these two engine-location categories, front and rear, are distinct enough to take engine-location as a potential good predictor of price.

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

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

```
[44]: <matplotlib.axes._subplots.AxesSubplot at 0x7f989493a588>
```





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:

[45]:	<code>df.describe()</code>																																																																																																																															
[45]:	<table border="1"> <thead> <tr><th></th><th>symboling</th><th>normalized-losses</th><th>wheel-base</th><th>length</th><th>width</th><th>height</th><th>curb-weight</th><th>engine-size</th><th>bore</th><th>stroke</th><th>compression-ratio</th><th>horsepower</th><th>peak-rpm</th><th>city-mpg</th></tr> </thead> <tbody> <tr><td>count</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>197.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td><td>201.000000</td></tr> <tr><td>mean</td><td>0.840796</td><td>122.000000</td><td>98.797015</td><td>0.837102</td><td>0.915126</td><td>53.766667</td><td>2555.666667</td><td>126.875622</td><td>3.330692</td><td>3.256904</td><td>10.164279</td><td>103.405534</td><td>5117.665368</td></tr> <tr><td>std</td><td>1.254802</td><td>31.99625</td><td>6.066366</td><td>0.059213</td><td>0.029187</td><td>2.447822</td><td>0.678039</td><td>41.546834</td><td>0.268072</td><td>0.319256</td><td>4.004965</td><td>37.365700</td><td>478.113805</td></tr> <tr><td>min</td><td>-2.000000</td><td>65.000000</td><td>86.600000</td><td>0.678039</td><td>0.837500</td><td>47.800000</td><td>1488.000000</td><td>61.000000</td><td>2.540000</td><td>2.070000</td><td>7.000000</td><td>48.000000</td><td>13.000000</td></tr> <tr><td>25%</td><td>0.000000</td><td>101.000000</td><td>94.500000</td><td>0.801538</td><td>0.890278</td><td>52.000000</td><td>2169.000000</td><td>98.000000</td><td>3.150000</td><td>3.110000</td><td>8.600000</td><td>70.000000</td><td>4800.000000</td></tr> <tr><td>50%</td><td>1.000000</td><td>122.000000</td><td>97.000000</td><td>0.832292</td><td>0.909722</td><td>54.100000</td><td>2414.000000</td><td>120.000000</td><td>3.310000</td><td>3.290000</td><td>9.000000</td><td>95.000000</td><td>5125.369458</td></tr> <tr><td>75%</td><td>2.000000</td><td>137.000000</td><td>102.400000</td><td>0.881788</td><td>0.925000</td><td>55.500000</td><td>2926.000000</td><td>141.000000</td><td>3.580000</td><td>3.410000</td><td>9.400000</td><td>116.000000</td><td>5500.000000</td></tr> <tr><td>max</td><td>3.000000</td><td>256.000000</td><td>120.900000</td><td>1.000000</td><td>1.000000</td><td>59.800000</td><td>4066.000000</td><td>326.000000</td><td>3.940000</td><td>4.170000</td><td>23.000000</td><td>262.000000</td><td>6600.000000</td></tr> </tbody> </table>		symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.000000	201.000000	201.000000	201.000000	201.000000	mean	0.840796	122.000000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.256904	10.164279	103.405534	5117.665368	std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	0.678039	41.546834	0.268072	0.319256	4.004965	37.365700	478.113805	min	-2.000000	65.000000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	13.000000	25%	0.000000	101.000000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.110000	8.600000	70.000000	4800.000000	50%	1.000000	122.000000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5125.369458	75%	2.000000	137.000000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.410000	9.400000	116.000000	5500.000000	max	3.000000	256.000000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	262.000000	6600.000000
	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg																																																																																																																		
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.000000	201.000000	201.000000	201.000000	201.000000																																																																																																																			
mean	0.840796	122.000000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.256904	10.164279	103.405534	5117.665368																																																																																																																			
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	0.678039	41.546834	0.268072	0.319256	4.004965	37.365700	478.113805																																																																																																																			
min	-2.000000	65.000000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	13.000000																																																																																																																			
25%	0.000000	101.000000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.110000	8.600000	70.000000	4800.000000																																																																																																																			
50%	1.000000	122.000000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5125.369458																																																																																																																			
75%	2.000000	137.000000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.410000	9.400000	116.000000	5500.000000																																																																																																																			
max	3.000000	256.000000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	262.000000	6600.000000																																																																																																																			

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

[46]:	<code>df.describe(include=['object'])</code>																																																							
[46]:	<table border="1"> <thead> <tr><th></th><th>make</th><th>aspiration</th><th>num-of-doors</th><th>body-style</th><th>drive-wheels</th><th>engine-location</th><th>engine-type</th><th>num-of-cylinders</th><th>fuel-system</th><th>horsepower-binned</th></tr> </thead> <tbody> <tr><td>count</td><td>201</td><td>201</td><td>201</td><td>201</td><td>201</td><td>201</td><td>201</td><td>201</td><td>201</td><td>200</td></tr> <tr><td>unique</td><td>22</td><td>2</td><td>2</td><td>5</td><td>3</td><td>2</td><td>6</td><td>7</td><td>8</td><td>3</td></tr> <tr><td>top</td><td>toyota</td><td>std</td><td>four</td><td>sedan</td><td>fwd</td><td>front</td><td>ohc</td><td>four</td><td>mpfi</td><td>Low</td></tr> <tr><td>freq</td><td>32</td><td>165</td><td>115</td><td>94</td><td>118</td><td>198</td><td>145</td><td>157</td><td>92</td><td>115</td></tr> </tbody> </table>		make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system	horsepower-binned	count	201	201	201	201	201	201	201	201	201	200	unique	22	2	2	5	3	2	6	7	8	3	top	toyota	std	four	sedan	fwd	front	ohc	four	mpfi	Low	freq	32	165	115	94	118	198	145	157	92	115
	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system	horsepower-binned																																														
count	201	201	201	201	201	201	201	201	201	200																																														
unique	22	2	2	5	3	2	6	7	8	3																																														
top	toyota	std	four	sedan	fwd	front	ohc	four	mpfi	Low																																														
freq	32	165	115	94	118	198	145	157	92	115																																														

Value Counts

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

[47]:	<code>df['drive-wheels'].value_counts()</code>								
[47]:	<table border="1"> <tbody> <tr><td>fwd</td><td>118</td></tr> <tr><td>rwd</td><td>75</td></tr> <tr><td>4wd</td><td>8</td></tr> <tr><td>Name: drive-wheels, dtype: int64</td><td></td></tr> </tbody> </table>	fwd	118	rwd	75	4wd	8	Name: drive-wheels, dtype: int64	
fwd	118								
rwd	75								
4wd	8								
Name: drive-wheels, dtype: int64									
	We can convert the series to a Dataframe as follows :								
[48]:	<code>df['drive-wheels'].value_counts().to_frame()</code>								
[48]:	<table border="1"> <thead> <tr><th>drive-wheels</th><th>value_counts</th></tr> </thead> <tbody> <tr><td>fwd</td><td>118</td></tr> <tr><td>rwd</td><td>75</td></tr> <tr><td>4wd</td><td>8</td></tr> </tbody> </table>	drive-wheels	value_counts	fwd	118	rwd	75	4wd	8
drive-wheels	value_counts								
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'.

[49]:	<code>drive_wheels_counts = df['drive-wheels'].value_counts().to_frame() drive_wheels_counts.rename(columns={'drive-wheels': 'value_counts'}, inplace=True) drive_wheels_counts</code>								
[49]:	<table border="1"> <thead> <tr><th>value_counts</th><th>drive-wheels</th></tr> </thead> <tbody> <tr><td>fwd</td><td>118</td></tr> <tr><td>rwd</td><td>75</td></tr> <tr><td>4wd</td><td>8</td></tr> </tbody> </table>	value_counts	drive-wheels	fwd	118	rwd	75	4wd	8
value_counts	drive-wheels								
fwd	118								
rwd	75								
4wd	8								
	Now let's rename the index to 'drive-wheels':								
[50]:	<code>drive_wheels_counts.index.name = 'drive-wheels' drive_wheels_counts</code>								
[50]:	<table border="1"> <thead> <tr><th>value_counts</th><th>drive-wheels</th></tr> </thead> <tbody> <tr><td>fwd</td><td>118</td></tr> <tr><td>rwd</td><td>75</td></tr> <tr><td>4wd</td><td>8</td></tr> </tbody> </table>	value_counts	drive-wheels	fwd	118	rwd	75	4wd	8
value_counts	drive-wheels								
fwd	118								
rwd	75								
4wd	8								

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

```
[51]: # 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)
```

	value_counts
front	198
rear	3

Examining the value counts of the 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; 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.

```
[54]: df['drive-wheels'].unique()
[54]: 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".

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

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

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

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

```
[57]: # 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
```

	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-wheel variable as the rows of the table, and pivot body-style to become the columns of the table:

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

	body-style	convertible	hardtop	hatchback	sedan	wagon	price
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.

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

	body-style	convertible	hardtop	hatchback	sedan	wagon	price
drive-wheels							
4wd		0.0	0.000000	7603.000000	12647.333333	9095.750000	
fwd		11595.0	8249.000000	8396.387755	9811.800000	9997.333333	
rwd		23949.6	24202.714286	14337.777778	21711.833333	16994.222222	

Question 4:

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

```
[62]: # Write your code below and press Shift+Enter to execute
df_gptest = df[['drive-wheels','body-style','price']]
grouped_test2 = df_gptest.groupby(['body-style'],as_index=False).mean()
grouped_test2
```

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

Double-click **here** for the solution.

<!-- The answer is below:

```
# grouping results
df_gptest2 = df[['body-style','price']]
grouped_test2bodystyle = df_gptest2.groupby(['body-style'],as_index= False).mean()
grouped_test2bodystyle
```

-->

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

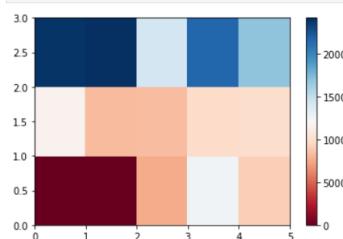
```
[63]: import matplotlib.pyplot as plt
%matplotlib inline
```

Variables: Drive Wheels and Body Style vs Price

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. [Learn more here.](#)

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

```
[66]: #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' in 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:

```
[74]: fig, ax = plt.subplots()
im = ax.pcolor(grouped_pivot, cmap='RdBu')

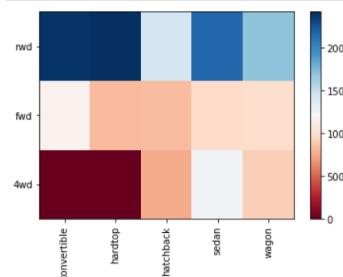
#label names
row_labels = grouped_pivot.columns.levels[1]
col_labels = grouped_pivot.index

#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 and that correlation does not imply causation. Determining correlation is much simpler than 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:** Total positive linear correlation.
- **0:** No linear correlation, the two variables most likely do not affect each other.
- **-1:** Total negative linear correlation.

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

[75]:	df.corr()														
	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg
symboling	1.00000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.008245	-0.182196	0.075819	0.279740	-0.035527	0.023
normalized-losses	0.466264	1.00000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029862	0.055563	-0.114713	0.217299	0.239543	-0.225016	-0.194
wheel-base	-0.535987	-0.056661	1.00000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.158502	0.250313	0.371147	-0.360305	-0.470606	-0.519
length	-0.365404	0.019424	0.876024	1.00000	0.857170	0.492063	0.880665	0.685025	0.608971	0.124139	0.159733	0.579821	-0.285970	-0.665192	-0.698
width	-0.242423	0.086802	0.814507	0.857170	1.00000	0.306002	0.666201	0.729436	0.544885	0.188829	0.189867	0.615077	-0.245800	-0.633531	-0.680
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.00000	0.307581	0.074694	0.180449	-0.062704	0.259737	-0.087027	-0.309974	-0.049800	-0.104
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.00000	0.849072	0.644060	0.167562	0.156433	0.757976	-0.279361	-0.749543	-0.794
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.00000	0.572609	0.209523	0.028889	0.822676	-0.256733	-0.650546	-0.679
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.00000	-0.055390	0.001263	0.566936	-0.267392	-0.582027	-0.591
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209523	-0.055390	1.00000	0.187923	0.098462	-0.065713	-0.034696	-0.035
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187923	1.00000	-0.214514	-0.435780	0.331425	0.268
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098462	-0.214514	1.00000	0.107885	-0.822214	-0.804
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.065713	-0.435780	0.107885	1.00000	-0.115413	-0.058
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027	-0.034696	0.331425	-0.822214	-0.115413	1.00000	0.972
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309	-0.035201	0.268465	-0.804575	-0.058598	0.972044	1.000
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	0.071107	0.809575	-0.101616	-0.686571	-0.71
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.745059	0.554610	0.037300	-0.299372	0.889488	0.115830	-0.949713	-0.91
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	0.241303	0.985231	-0.169053	-0.475812	0.265676	0.198
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241303	-0.985231	0.169053	0.475812	-0.265676	-0.198

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.

[76]: from scipy import stats

Wheel-base vs Price

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

```
[77]: person_coef, p_value = stats.pearsonr(df['wheel-base'], df['price'])
print("The Pearson Correlation Coefficient is", person_coef, "with a P-value of P =", p_value)
The Pearson Correlation Coefficient is 0.5846418222655081 with a P-value of P = 8.076488270732955e-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 (~0.58).

Horsepower vs Price

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

```
[78]: person_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
print("The Pearson Correlation Coefficient is", person_coef, "with a P-value of P =", p_value)
The Pearson Correlation Coefficient is 0.8095745670036559 with a P-value of P = 6.36905742825998e-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'.

```
[79]: person_coef, p_value = stats.pearsonr(df['length'], df['price'])
```

```
[80]: print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
```

Conclusion:

Since the p-value is < 0.001, the correlation between 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':

```
[80]: 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)
```

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':

```
[81]: 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)
```

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':

```
[82]: 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)
```

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':

```
[83]: 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)
```

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

```
[84]: 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)
```

Conclusion:

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

Highway-mpg vs Price

```
[85]: 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)
```

Conclusion:

Since the p-value is < 0.001, the correlation between highway-mpg and price is statistically significant, and the coefficient of ~ -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 is our calculated score value.

If our price variable is strongly correlated with the variable we are analyzing, 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.

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

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

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

```
[86]:   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
```

```
[87]: df_gptest
```

```
[87]:   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
      5        fwd  sedan  15250.0
      6        fwd  sedan  17710.0
      7        fwd  wagon  18920.0
      8        fwd  sedan  23875.0
      9        rwd  sedan  16430.0
      10       rwd  sedan  16925.0
      11       rwd  sedan  20970.0
      12       rwd  sedan  21105.0
      13       rwd  sedan  24565.0
      14       rwd  sedan  30760.0
      15       rwd  sedan  41315.0
      16       rwd  sedan  36880.0
      17        fwd  hatchback  5151.0
      18        fwd  hatchback  6295.0
      19        fwd  sedan  6575.0
      20        fwd  hatchback  5572.0
      21        fwd  hatchback  6377.0
      22        fwd  hatchback  7957.0
      23        fwd  hatchback  6229.0
      24        fwd  sedan  6692.0
      25        fwd  sedan  7609.0
      26        fwd  sedan  8558.0
      27        fwd  wagon  8921.0
      28        fwd  hatchback  12964.0
      29        fwd  hatchback  6479.0
      ...
      ...
      ...
      171       fwd  hatchback  9988.0
      172       fwd  sedan  10898.0
      173       fwd  hatchback  11248.0
      174       rwd  hatchback  16558.0
      175       rwd  hatchback  15998.0
      176       rwd  sedan  15690.0
      177       rwd  wagon  15750.0
      178       fwd  sedan  7775.0
      179       fwd  sedan  7975.0
      180       fwd  sedan  7995.0
      181       fwd  sedan  8195.0
      182       fwd  sedan  8495.0
      183       fwd  sedan  9495.0
      184       fwd  sedan  9995.0
      185       fwd  convertible  11595.0
      186       fwd  hatchback  9980.0
      187       fwd  sedan  13295.0
      188       fwd  sedan  13845.0
      189       fwd  wagon  12290.0
      190       rwd  sedan  12940.0
      191       rwd  wagon  13415.0
      192       rwd  sedan  15985.0
      193       rwd  wagon  16515.0
      194       rwd  sedan  18420.0
      195       rwd  wagon  18950.0
      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 × 3 columns

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

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

```
[88]: 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**.

```
[89]: # 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= 67.95406500788399 , 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?

Separately: fwd and rwd

```
[90]: 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

```
[91]: 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

```
[92]: 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 = 0.41620116697845666
```

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 notebook

```
<p><a href="https://cocl.us/corsera_da0101en_notebook_bottom"></a></p>
```

About the Authors:

This notebook was written by [Mahdi Noorian PhD](#), [Joseph Santarcangelo](#), Bahare Talayan, Eric Xiao, Steven Dong, Parizad, Hima Vsudevan and [Fiorella Wenver](#) and [Yi Yao](#).

[Joseph Santarcangelo](#) is a Data Scientist at IBM, and holds a PhD in Electrical Engineering. His research focused on using Machine Learning, Signal Processing, and Computer Vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Copyright © 2018 IBM Developer Skills Network. This notebook and its source code are released under the terms of the [MIT License](#).

