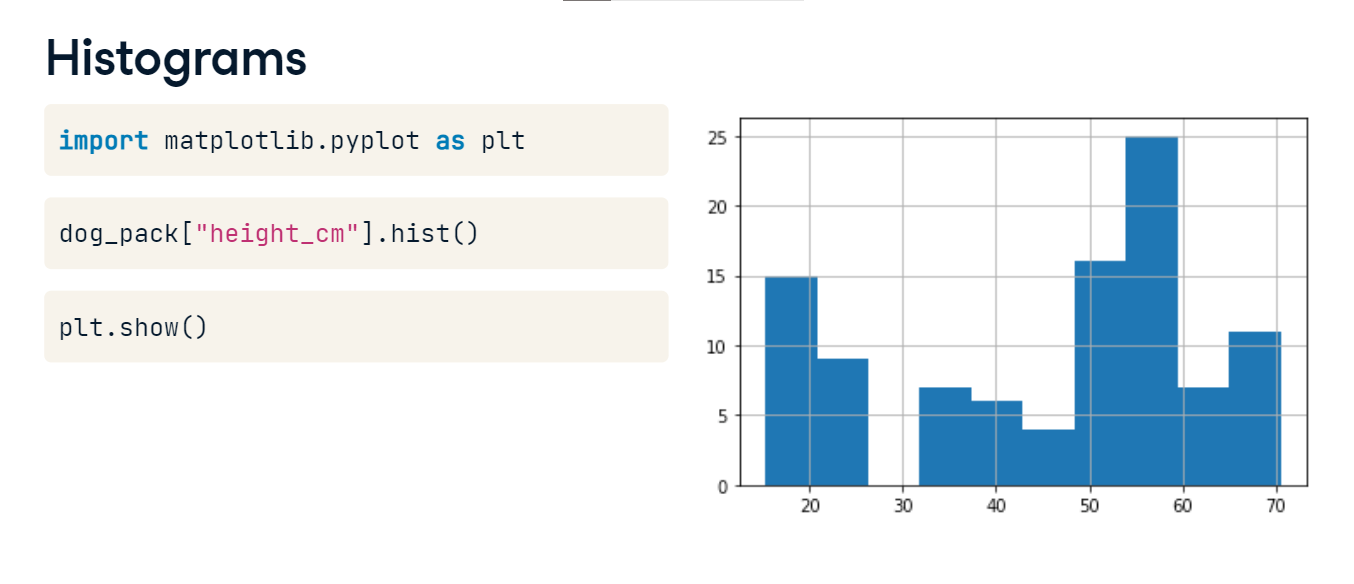
**Visualizing your data**

Plots are a powerful way to share the insights you've gained from your data. In this lesson, we'll use a bigger dataset of dogs, called dog\_pack, to make visualization easier.

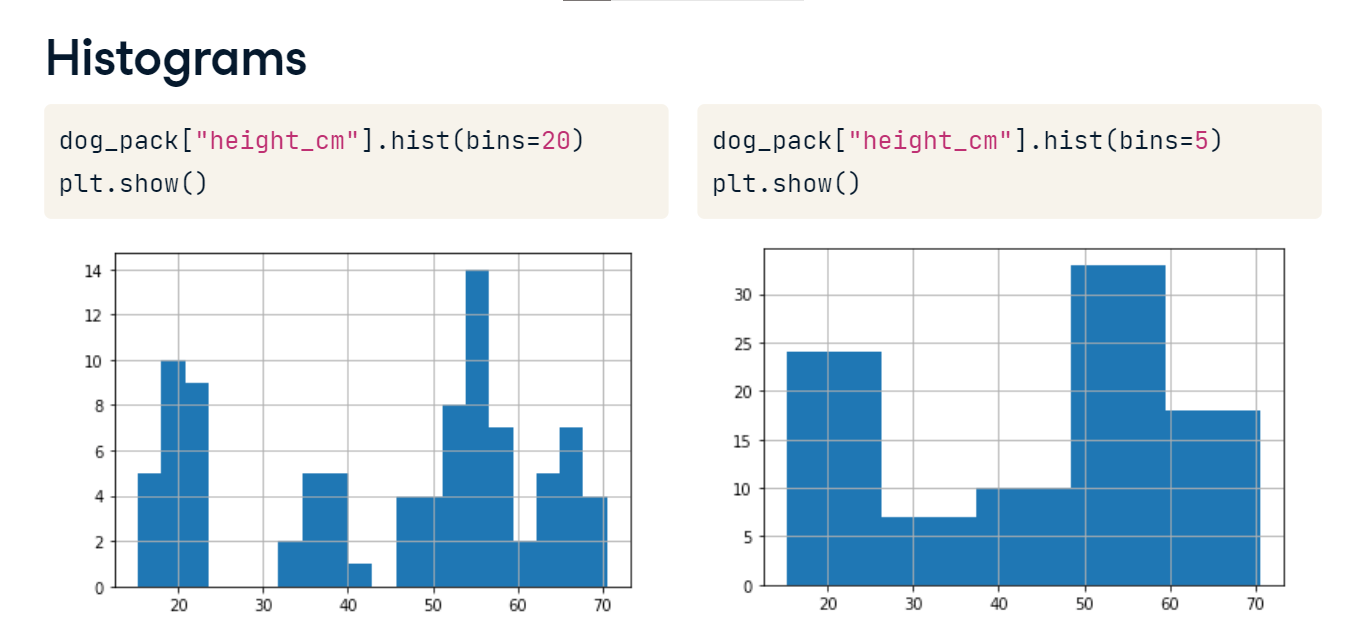
**Histograms**

Remember when we talked about matplotlib at the beginning of the course? We'll need to import matplotlib-dot-pyplot as plt in order to display our visualizations. Just like pd is the standard alias for pandas, plt is the standard alias for matplotlib-dot-pyplot. Let's create a histogram, which shows the distribution of a numeric variable. We can create a histogram of the height variable by selecting the column and calling dot-hist. In order to show the plot, we need to call plt-dot-show. The x-axis represents the heights of the dogs, and the y-axis represents the number of dogs in each height range. By grouping observations into ranges, the histogram allows us to see that there are a lot of dogs around 50 to 60 centimeters tall.



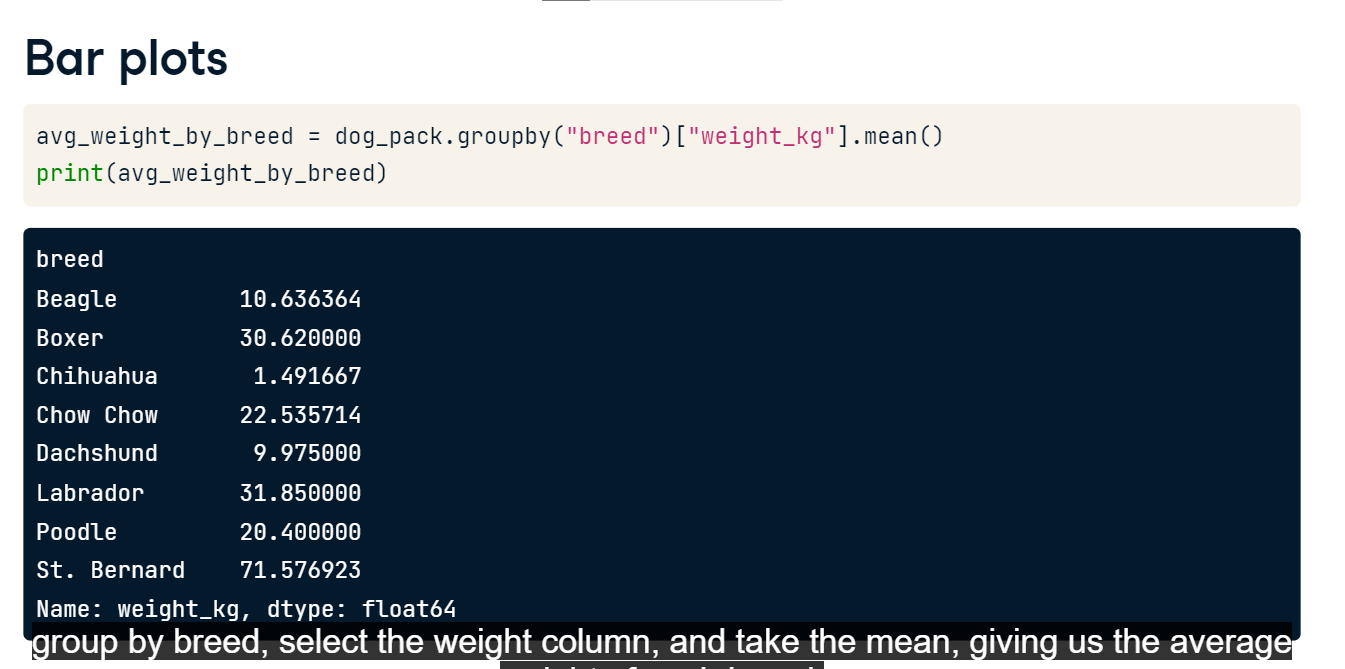
**Histograms**

We can adjust the number of bars, or bins, using the "bins" argument. Increasing or decreasing this can give us a better idea of what the distribution looks like.



**Bar plots**

Bar plots can reveal relationships between a categorical variable and a numeric variable, like breed and weight. To compute the average weight of each breed, we group by breed, select the weight column, and take the mean, giving us the average weight of each breed.



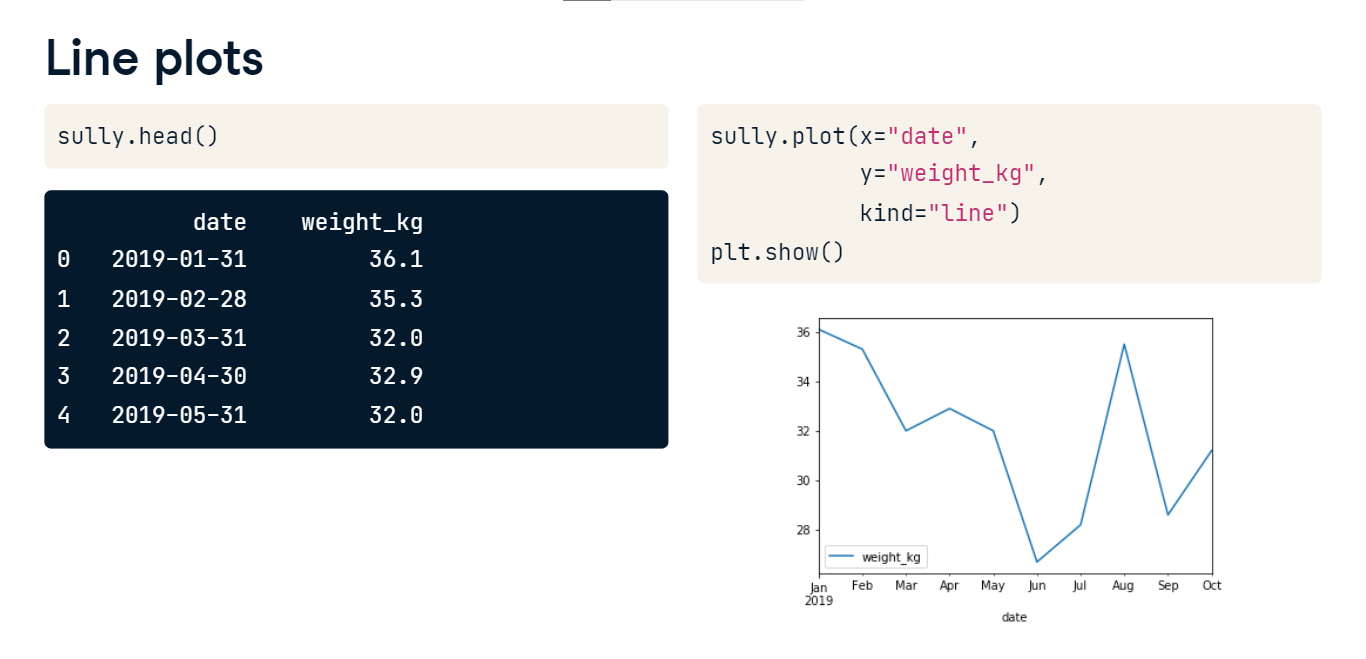
**Bar plots**

Now we can create a bar plot from the mean weights using the plot method, setting "kind" equal to "bar." Finally, we call plt-dot-show. To add a title to our plot, we can use the title argument of the plot method. It looks like Saint Bernards are the heaviest breed on average! Woof!



**Line plots**

Line plots are great for visualizing changes in numeric variables over time. Lucky for us, a Labrador named Sully has been weighed by his owner every month - let's see how his weight has changed over the year. We can use the plot method again, but this time, we pass in three arguments: date as x, weight as y, and "kind" equals "line." Sully's weight has fluctuated quite a bit over the year!



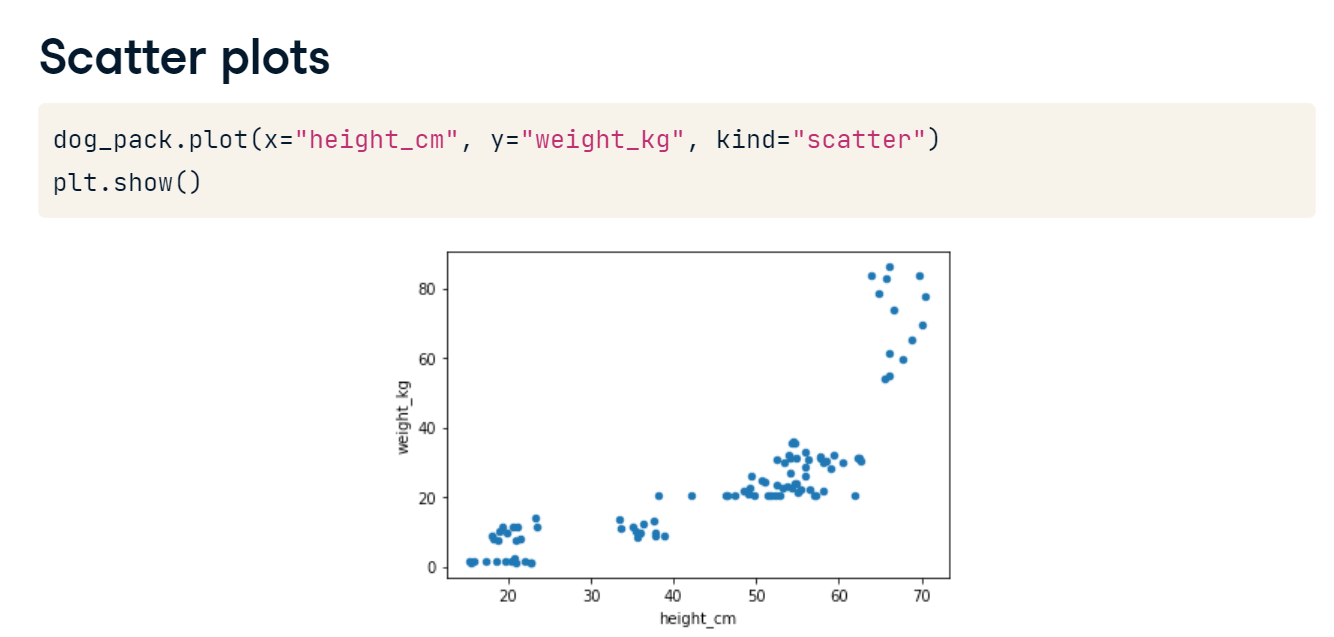
**Rotating axis labels**

We may want to rotate the x-axis labels to make the text easier to read. This can be done by passing an angle in degrees with the "rot" argument. Here, we rotate the labels by 45 degrees.



**Scatter plots**

Scatter plots are great for visualizing relationships between two numeric variables. To plot each dog's height versus their weight, we call the plot method with x equal to height\_cm, y equal to weight\_kg, and "kind" equal to "scatter." From our plot, it looks like taller dogs tend to weigh more.

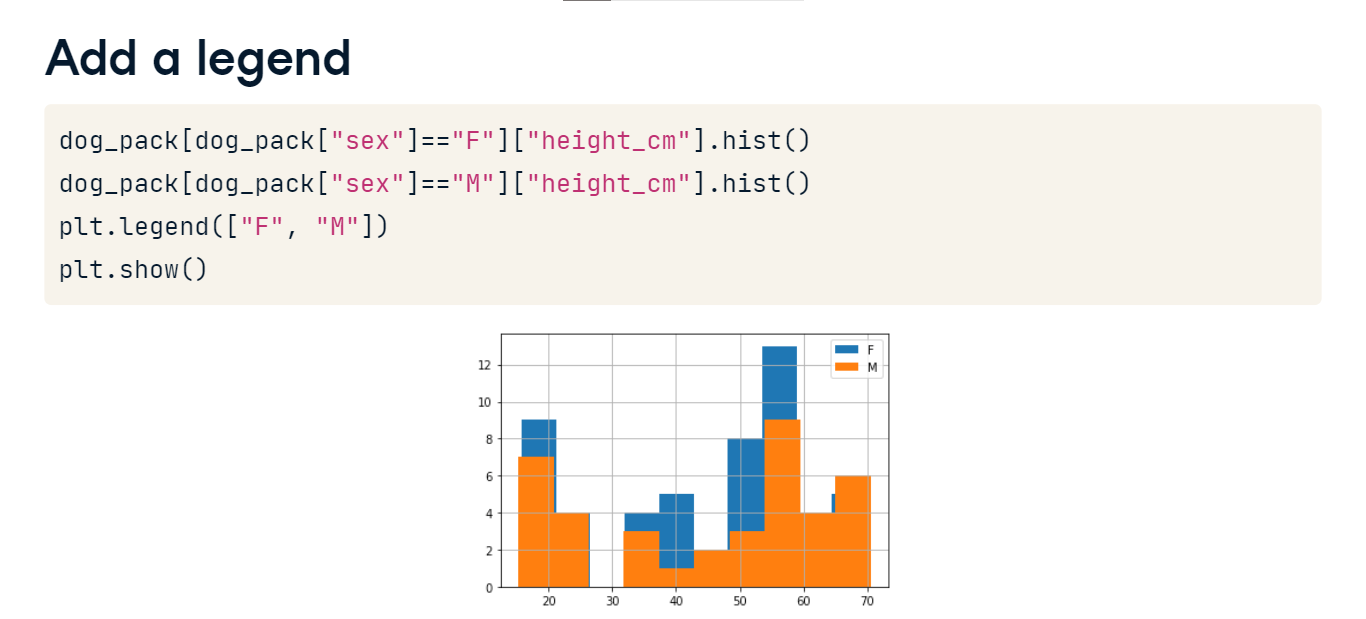


**Layering plots**

Plots can also be layered on top of one another. For example, we can create a histogram of female dogs' heights, and put a histogram of male dogs' heights on top, then call show. However, we can't tell which color represents which sex.

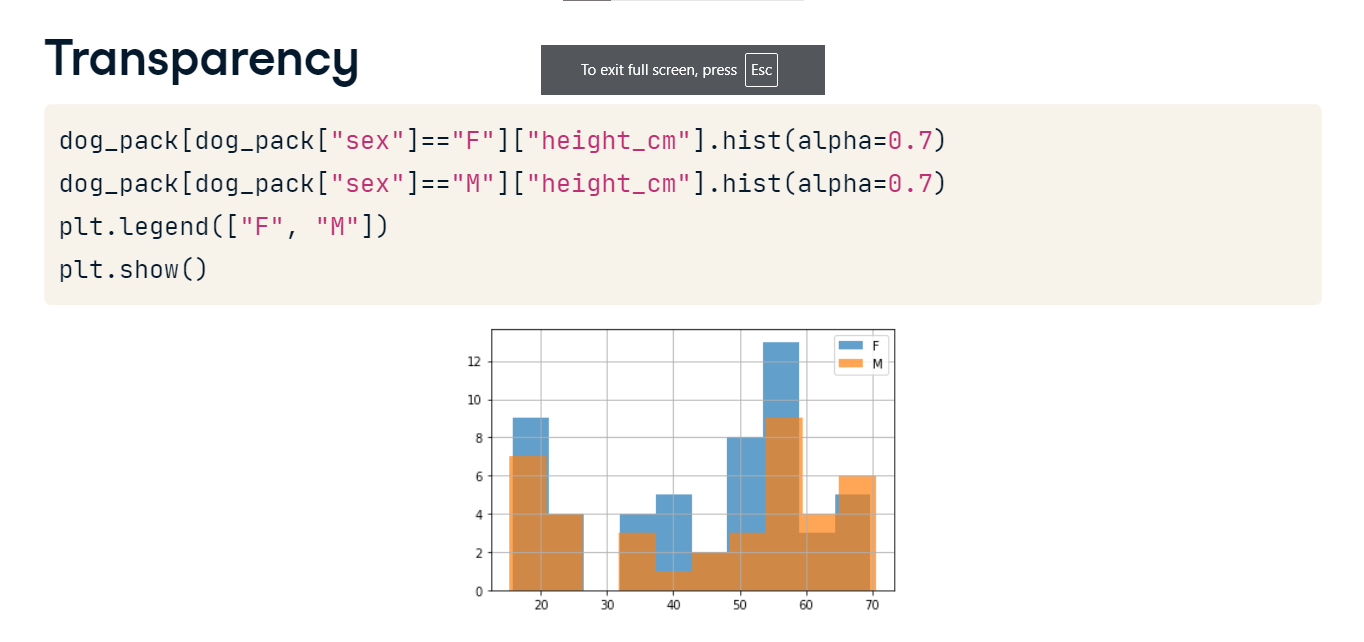
**Add a legend**

We can use plt-dot-legend, passing in a list of labels, and then call show. Now we know which color is which, but we can't see what's going on behind the orange histogram.

****

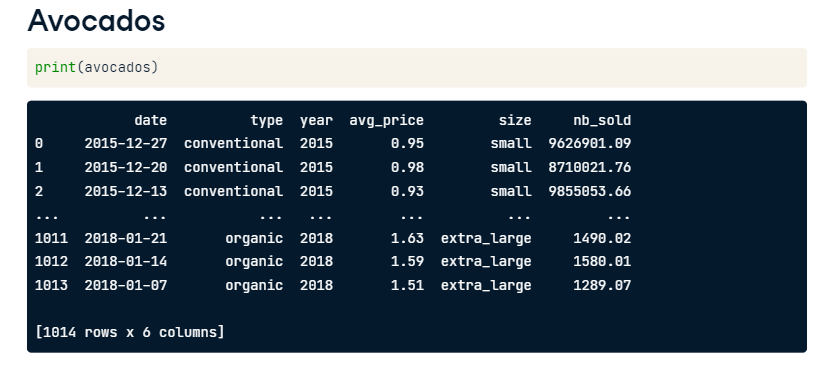
**Transparency**

Let's fix this problem by making the histograms translucent. We can use hist's alpha argument, which takes a number. 0 means completely transparent that is, invisible, and 1 means completely opaque.



**Avocados**

In this chapter, you'll be working with a dataset that contains weekly US avocado sales data, broken down by avocado size, and whether or not the avocados were organic.



**Missing values**

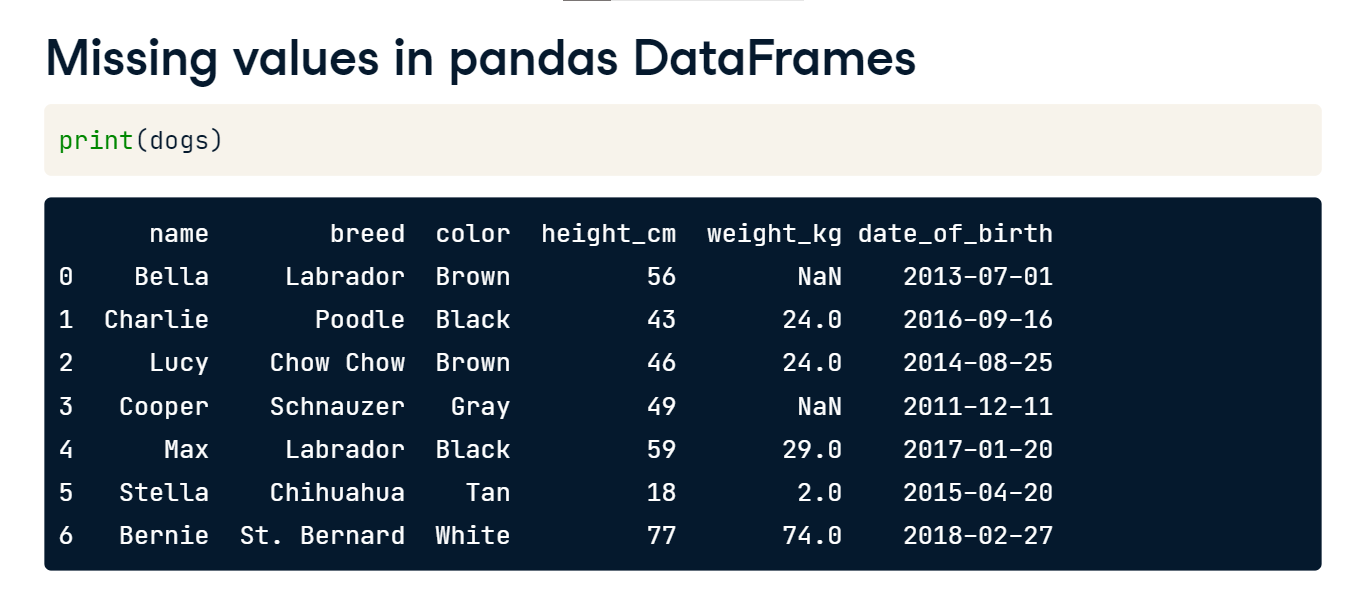
You could be given a DataFrame that has missing values, so it's important to know how to handle them.

**What's a missing value?**

Most data is not perfect - there's always a possibility that there are some pieces missing from your dataset. For example, maybe on the day that Bella and Cooper's owner weighed them,the scale was broken. Now we have two missing values in our dataset.

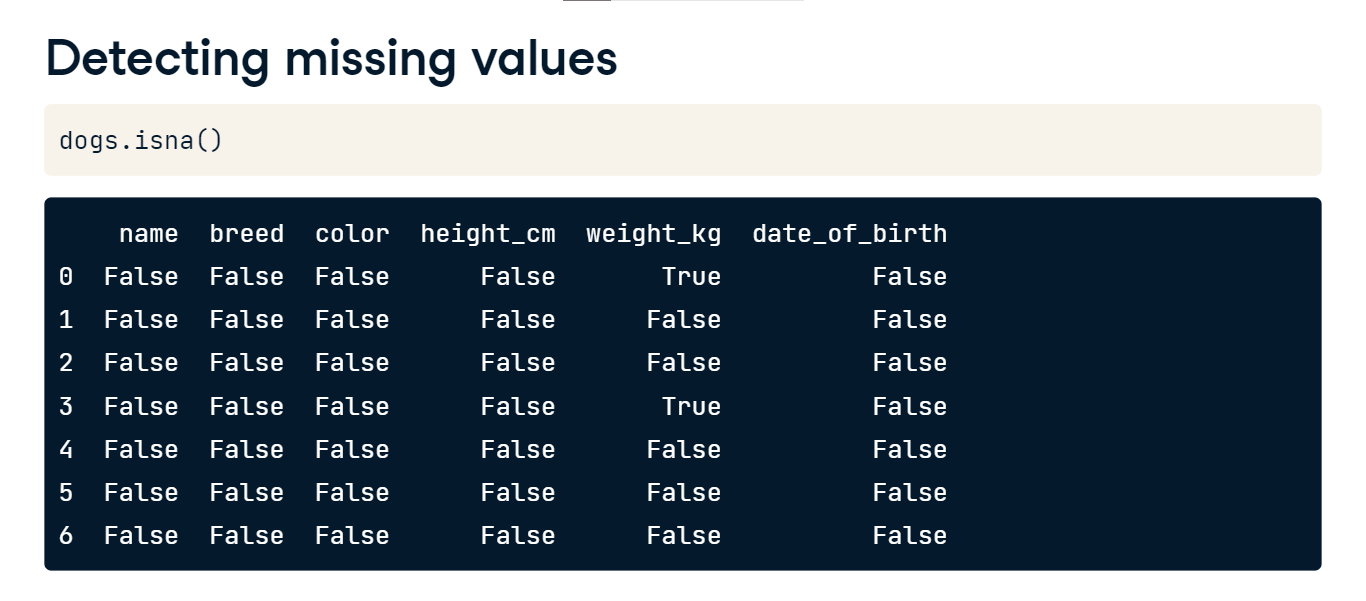
**Missing values in pandas DataFrames**

In a pandas DataFrame, missing values are indicated with N-a-N, which stands for "not a number."



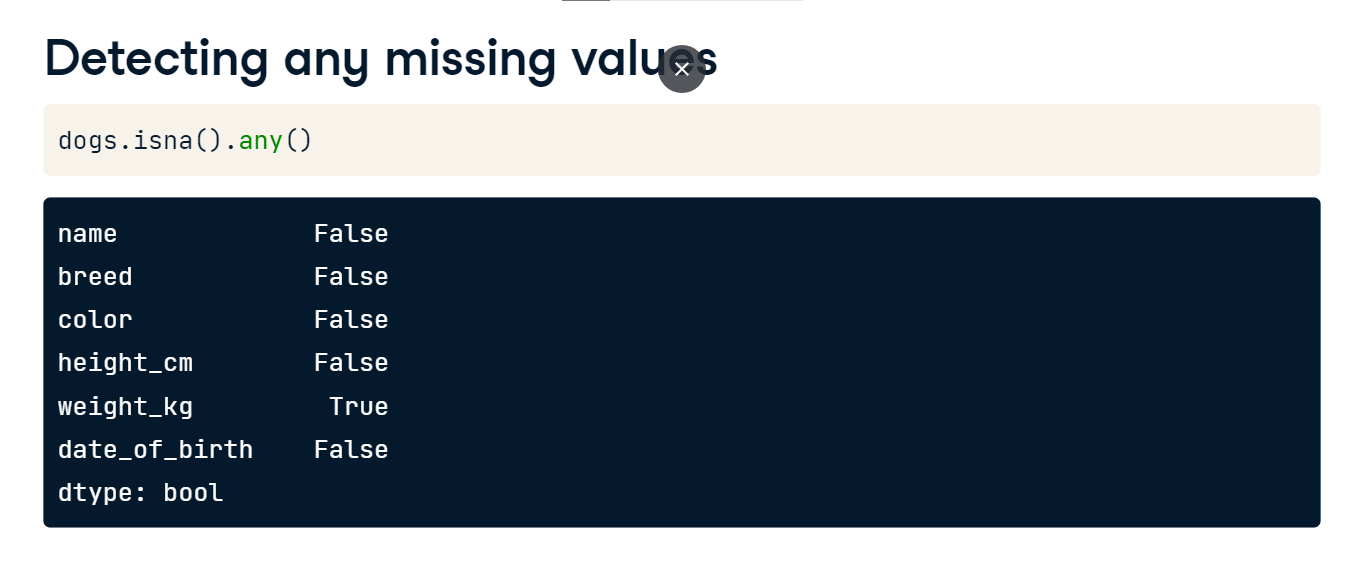
**Detecting missing values**

When you first get a DataFrame, it's a good idea to get a sense of whether it contains any missing values, and if so, how many. That's where the isna method comes in. When we call isna on a DataFrame, we get a Boolean for every single value indicating whether the value is missing or not, but this isn't very helpful when you're working with a lot of data.



**Detecting any missing values**

If we chain dot-isna with dot-any, we get one value for each variable that tells us if there are any missing values in that column. Here, we see that there's at least one missing value in the weight column, but not in any of the others.



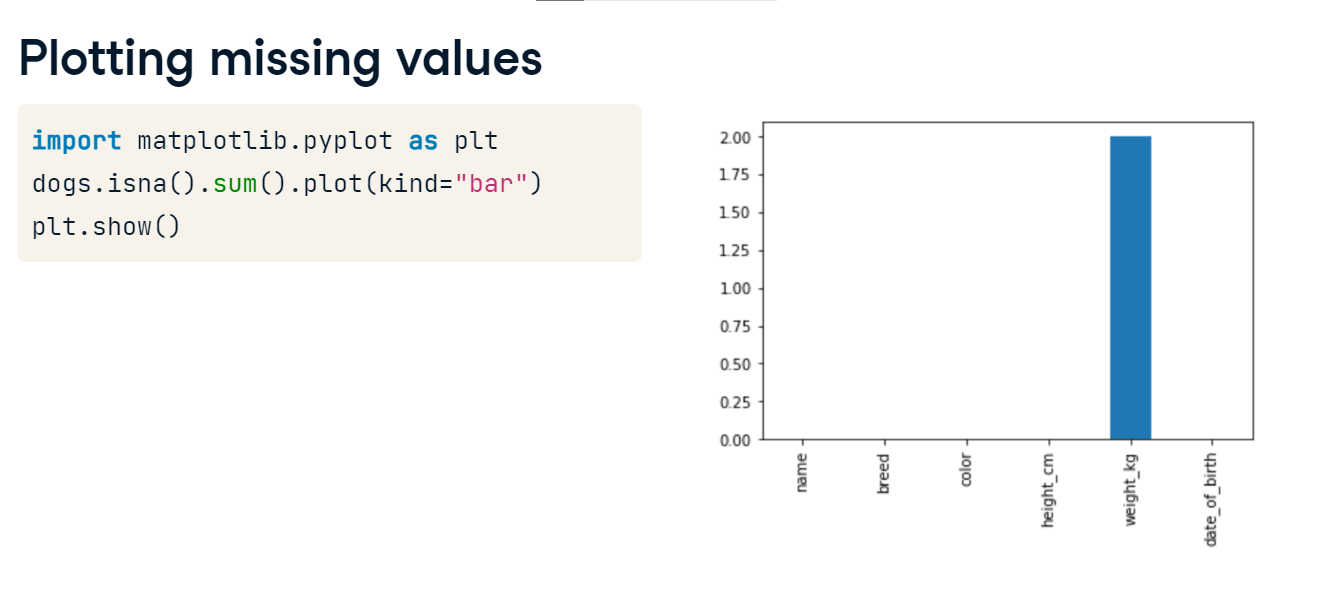
**Counting missing values**

Since taking the sum of Booleans is the same thing as counting the number of Trues, we can combine sum with isna to count the number of NaNs in each column.



**Plotting missing values**

We can use those counts to visualize the missing values in the dataset using a bar plot. Plots like this are more interesting when you have missing data across different variables, while here, only weights are missing. Now that we know there are missing values in the dataset, what can we do about them?



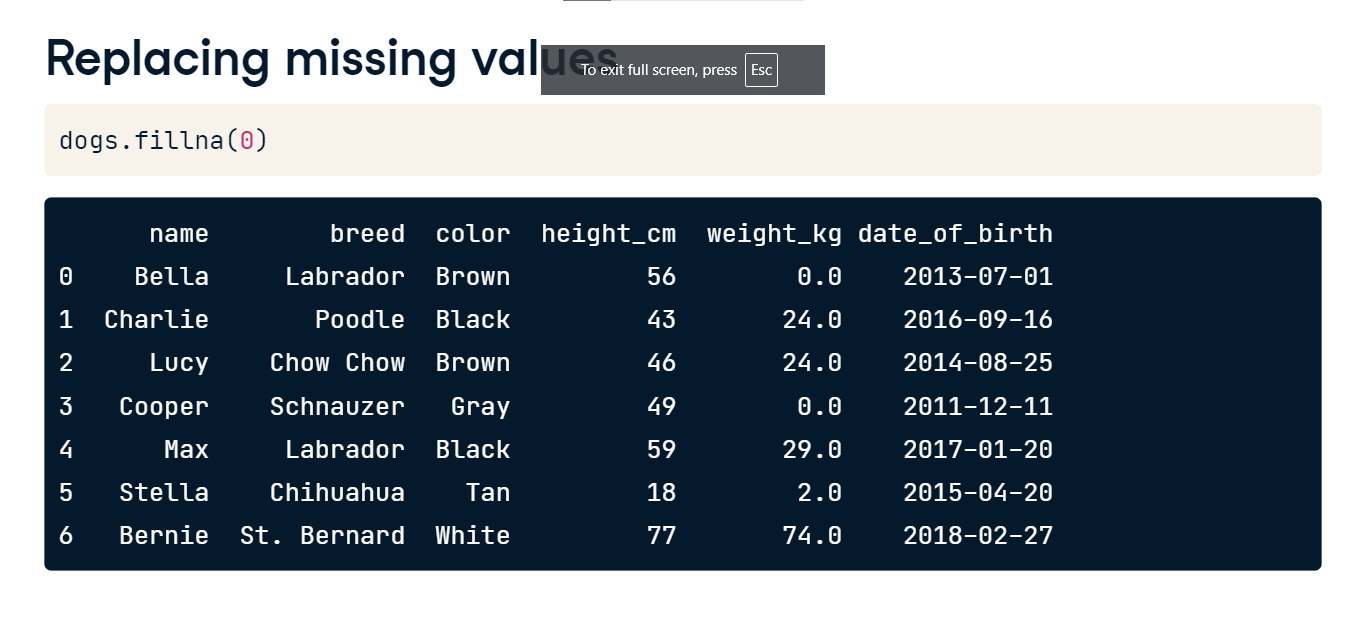
**Removing missing values**

One option is to remove the rows in the DataFrame that contain missing values. This can be done using the dropna method. However, this may not be ideal if you have a lot of missing data, since that means losing a lot of observations.



**Replacing missing values**

Another option is to replace missing values with another value. The fillna method takes in a value, and all NaNs will be replaced with this value. There are also many sophisticated techniques for replacing missing values, which you can learn more about in our course about missing data.

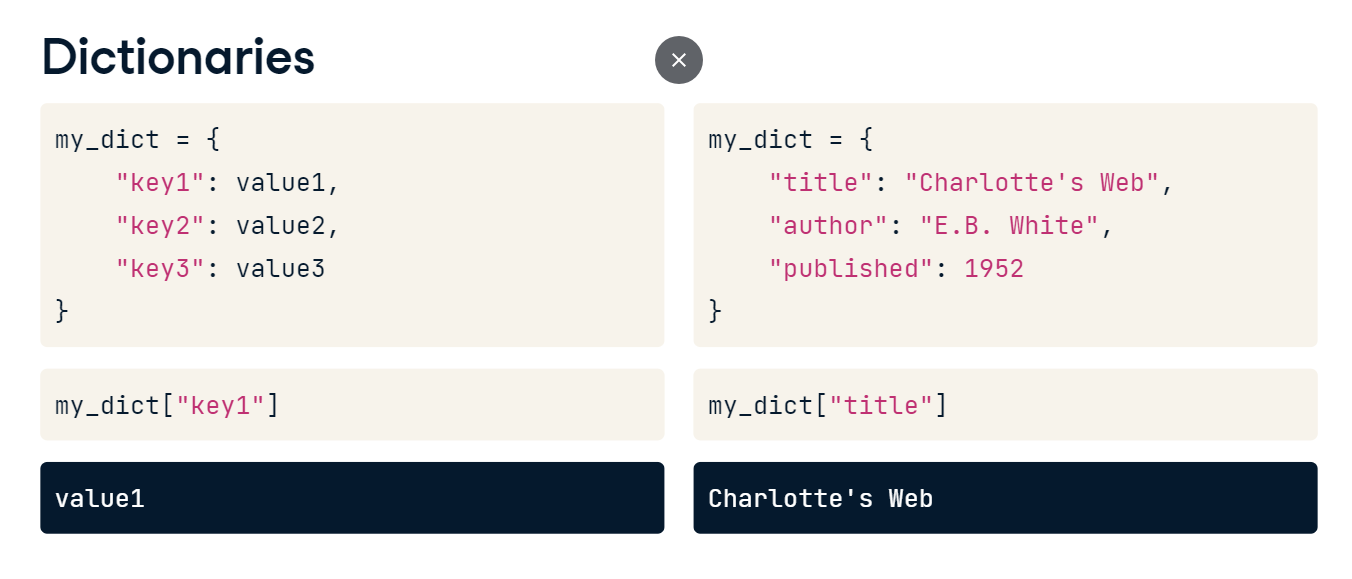


**Creating DataFrames**

Now that you've learned a lot about how to work with pandas DataFrames, how do you get data into a DataFrame in the first place?

**Dictionaries**

Before creating your own DataFrames, let's talk about dictionaries. A dictionary is a way of storing data in Python. It holds a set of key-value pairs. You can create a dictionary like this, using curly braces. Inside, each key-value pair is written as "key colon value." Let's create a dictionary that holds information about a book. "Title" is a key in the dictionary, and "Charlotte's Web" is its corresponding value, and so on. You can access values of a dictionary via their keys in square brackets. For example, we can access the value of "title" like this.

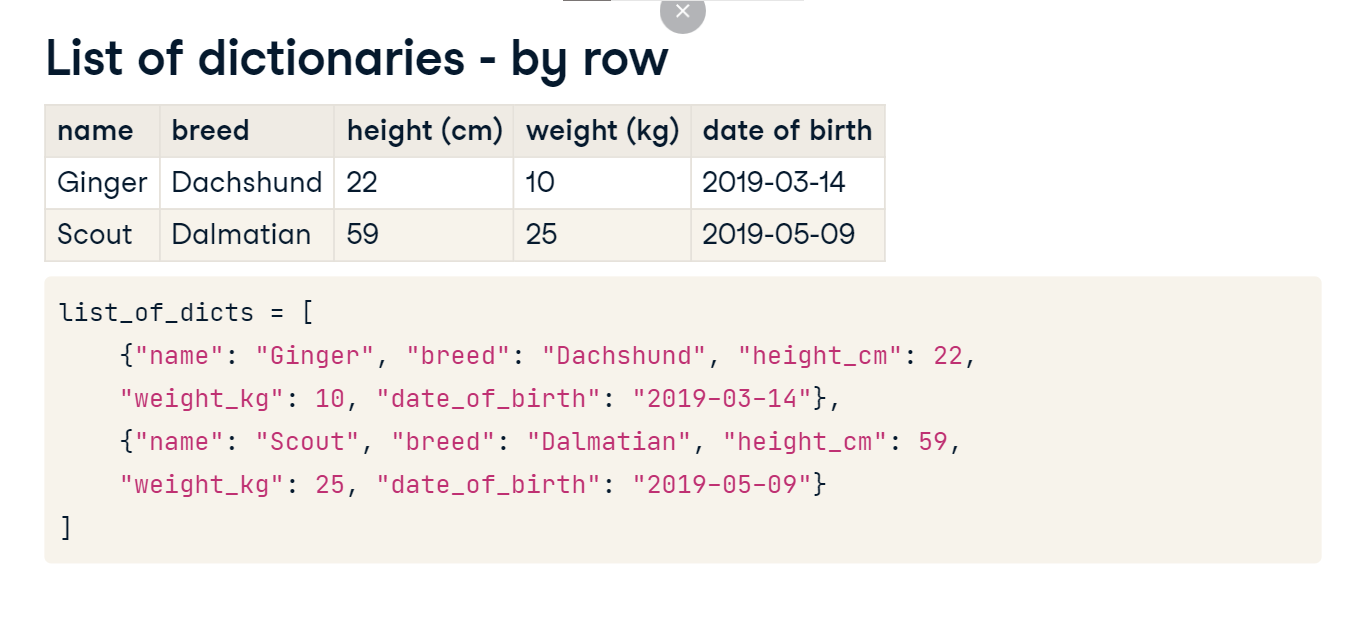


**Creating DataFrames**

There are many ways to create DataFrames from scratch, but we'll discuss two ways: from a list of dictionaries and from a dictionary of lists. In the first method, the DataFrame is built up row by row, while in the second method, the DataFrame is built up column by column.

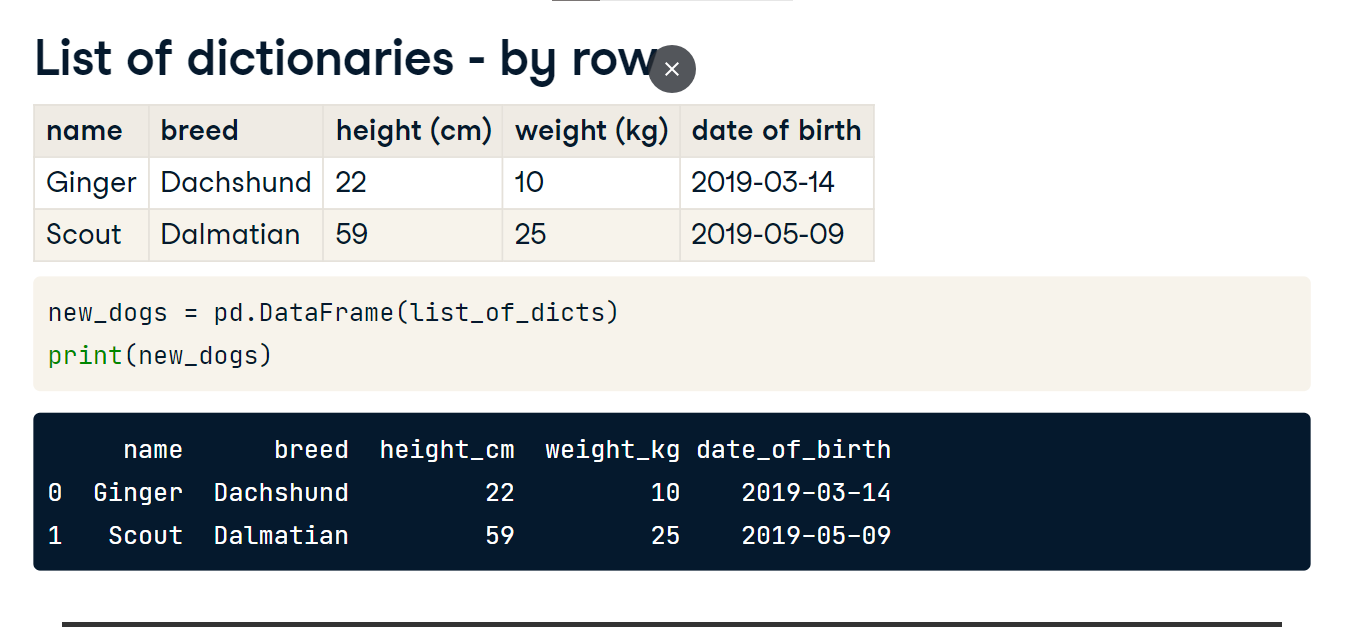
**List of dictionaries - by row**

We have some new dog data to put into a DataFrame. Let's start with the first method to do this, creating a list of dictionaries. First, we'll create a new list using square brackets to hold our dictionaries. Then, we'll go through the first row of our data and put it in a dictionary. Each key, on the left of each colon, will become a column name. Each value is one dog's data for that column. Here, the first key is "name," which is the first column name, and its corresponding value is "Ginger," the name of the first dog. The second key is the second column name, "breed," and its value is "Dachshund," which is the first dog's breed. Then we have the dog's height and weight. For the next row, we create another dictionary that follows the same format.



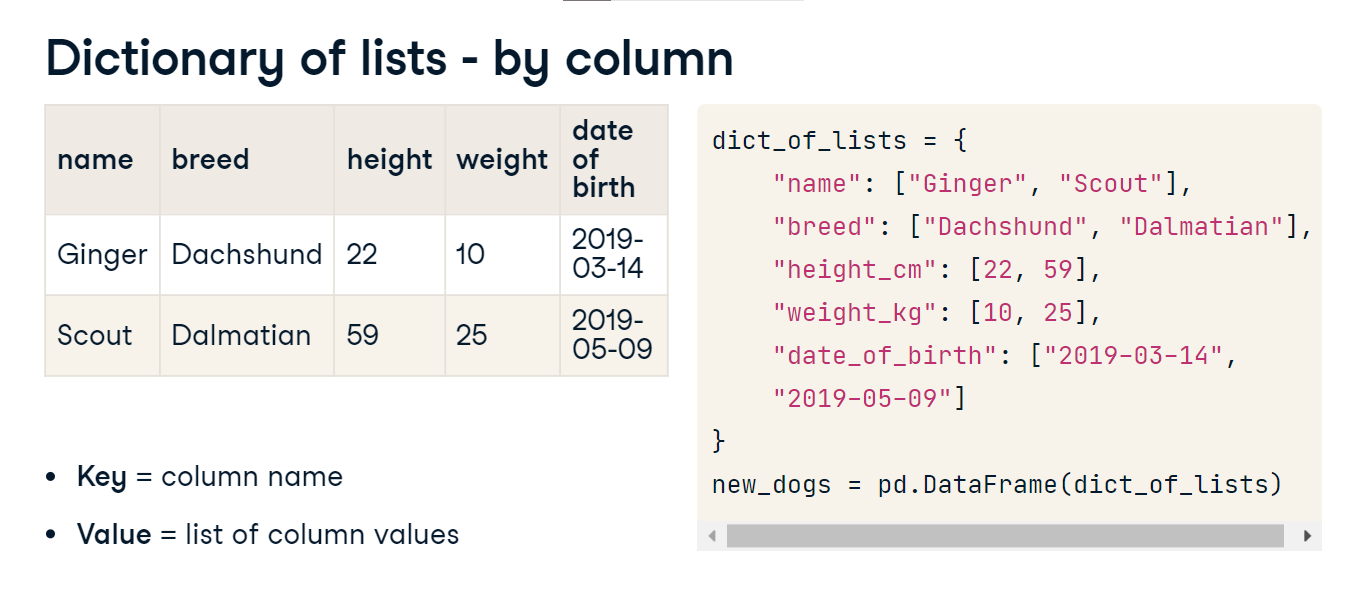
**List of dictionaries - by row**

Now that we have our list of dictionaries, we can pass it into pd-dot-DataFrame to convert it into DataFrame form.



**Dictionary of lists - by column**

Now let's talk about the dictionary of lists method. When using this method, we need to go through the data column by column. Remember that keys are to the left of a colon, and values are to the right. Each key will be a column name, and each value will be a list of the values in the column. First, we'll create a dictionary using curly braces. Let's start with the first column, which is called "name," so the first key is "name." The value is a list containing each name, from top to bottom. In this case, it's "Ginger" and "Scout." Next, we have the "breed" column, so we add "breed" as a key, and its corresponding value is a list containing "Dachshund" and "Dalmatian." Then we have height\_cm, which is 22 and 59, and weight\_kg, which is 10 and 25. Now that we have our dictionary of lists set up, we can pass it into pd-dot-DataFrame to convert it into a pandas DataFrame.



**Dictionary of lists - by column**

If we print the new DataFrame, we can see that it's exactly what we wanted.

**Reading and writing CSVs**

You now know how to create your own DataFrames, but typing out your data entry-by-entry isn't usually the most efficient way to get your data into a DataFrame. In this video, you'll learn how to pull data from CSV files.

**What's a CSV file?**

CSV, or comma-separated values, is a common data storage file type. It's designed to store tabular data, just like a pandas DataFrame. It's a text file, where each row of data has its own line, and each value is separated by a comma. Almost every database, programming language, and piece of data analysis software can read and write CSV files. That makes it a good storage format if you need to share your data with other people who may be using different tools than you.

**Example CSV file**

Remember the dogs from the last video? Their data is stored in a CSV file called new\_dogs-dot-csv, which looks like this.

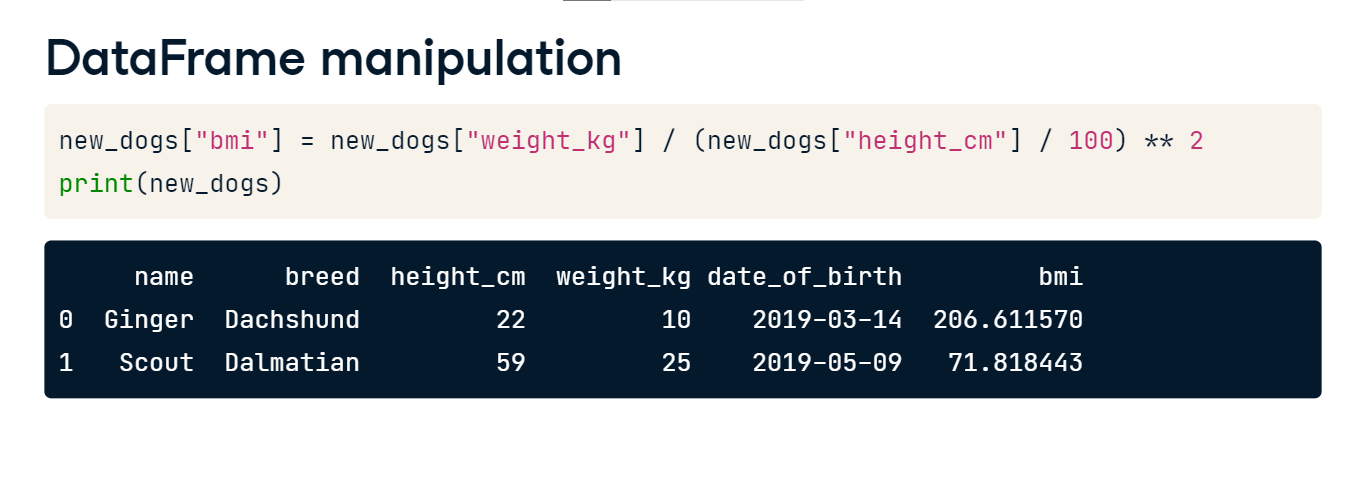
**CSV to DataFrame**

We can put this data in a DataFrame using the handy pandas function, read-underscore-csv, and pass it the file path of the CSV.

**DataFrame manipulation**

01:09 - 01:20

Now that the data is in DataFrame form, we can manipulate it using some of the functions from earlier in the course. Here, we'll add a body mass index column.



**DataFrame to CSV**

Now that we've changed the data let's create an updated CSV file to share with the dogs' owners. To convert a DataFrame to a CSV, we can use new\_dogs dot to-underscore-csv, and pass in a new file path. If we take a look at the new file, it contains the BMI column.

