**Introducing DataFrames**

Hi, I'm Richie. I'll be your tour guide through the world of pandas.

**What's the point of pandas?**

pandas is a Python package for data manipulation. It can also be used for data visualization; we'll get to that in Chapter 4.

**Course outline**

We'll start by talking about DataFrames, which form the core of pandas. In chapter 2, we'll discuss aggregating data to gather insights. In chapter 3, you'll learn all about slicing and indexing to subset DataFrames. Finally, you'll visualize your data, deal with missing data, and read data into a DataFrame. Let's dive in.

**pandas is built on NumPy and Matplotlib**

pandas is built on top of two essential Python packages, NumPy and Matplotlib. Numpy provides multidimensional array objects for easy data manipulation that pandas uses to store data, and Matplotlib has powerful data visualization capabilities that pandas takes advantage of.

**pandas is popular**

pandas has millions of users, with PyPi recording about 14 million downloads in December 2019. This represents almost the entire Python data science community!

1. 1 https://pypistats.org/packages/pandas

**Rectangular data**

There are several ways to store data for analysis, but rectangular data, sometimes called "tabular data" is the most common form. In this example, with dogs, each observation, or each dog, is a row, and each variable, or each dog property, is a column. pandas is designed to work with rectangular data like this.

**pandas DataFrames**

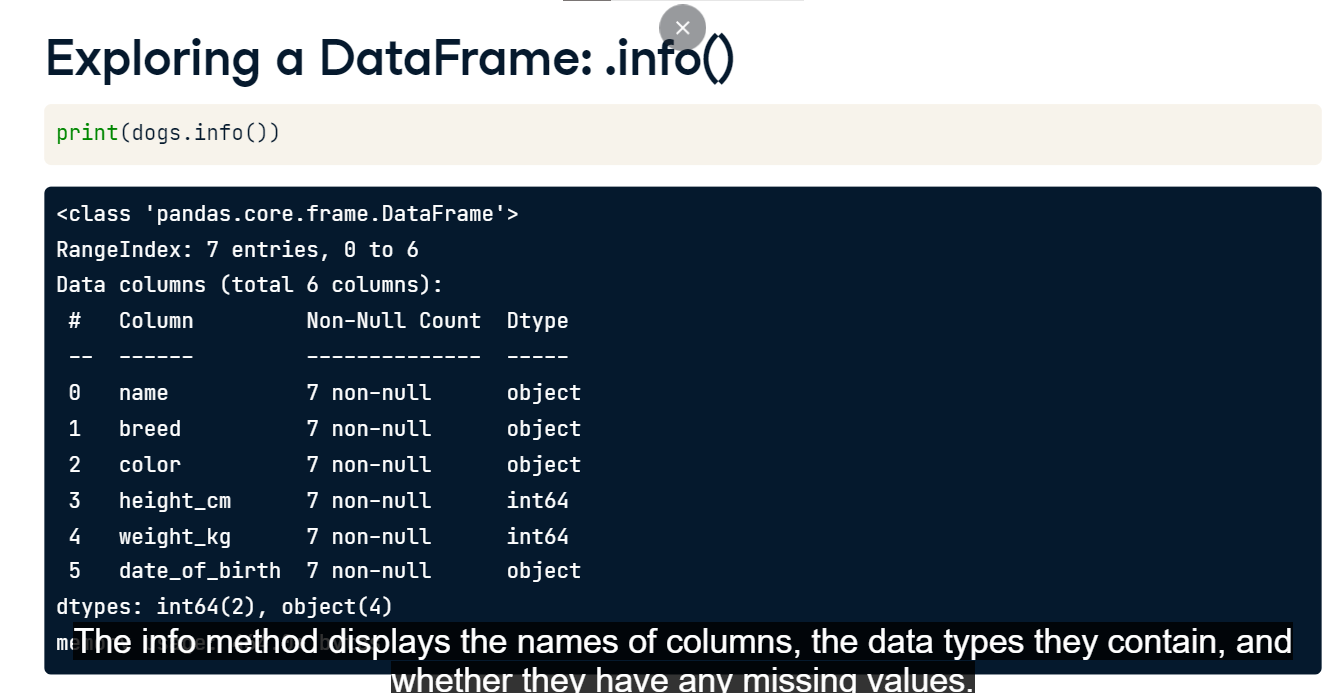
In pandas, rectangular data is represented as a DataFrame object. Every programming language used for data analysis has something similar to this. R also has DataFrames, while SQL has database tables. Every value within a column has the same data type, either text or numeric, but different columns can contain different data types.

**Exploring a DataFrame: .head()**

When you first receive a new dataset, you want to quickly explore it and get a sense of its contents. pandas has several methods for this. The first is head, which returns the first few rows of the DataFrame. We only had seven rows to begin with, so it's not super exciting, but this becomes very useful if you have many rows.

**Exploring a DataFrame: .info()**

The info method displays the names of columns, the data types they contain, and whether they have any missing values.

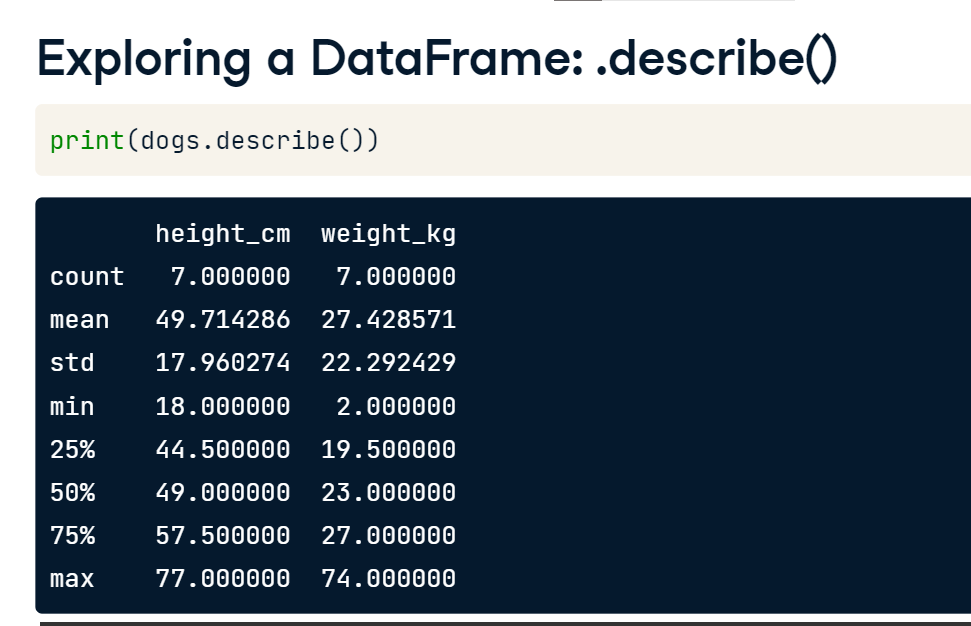


**Exploring a DataFrame: .shape**

A DataFrame's shape attribute contains a tuple that holds the number of rows followed by the number of columns. Since this is an attribute instead of a method, you write it without parentheses.

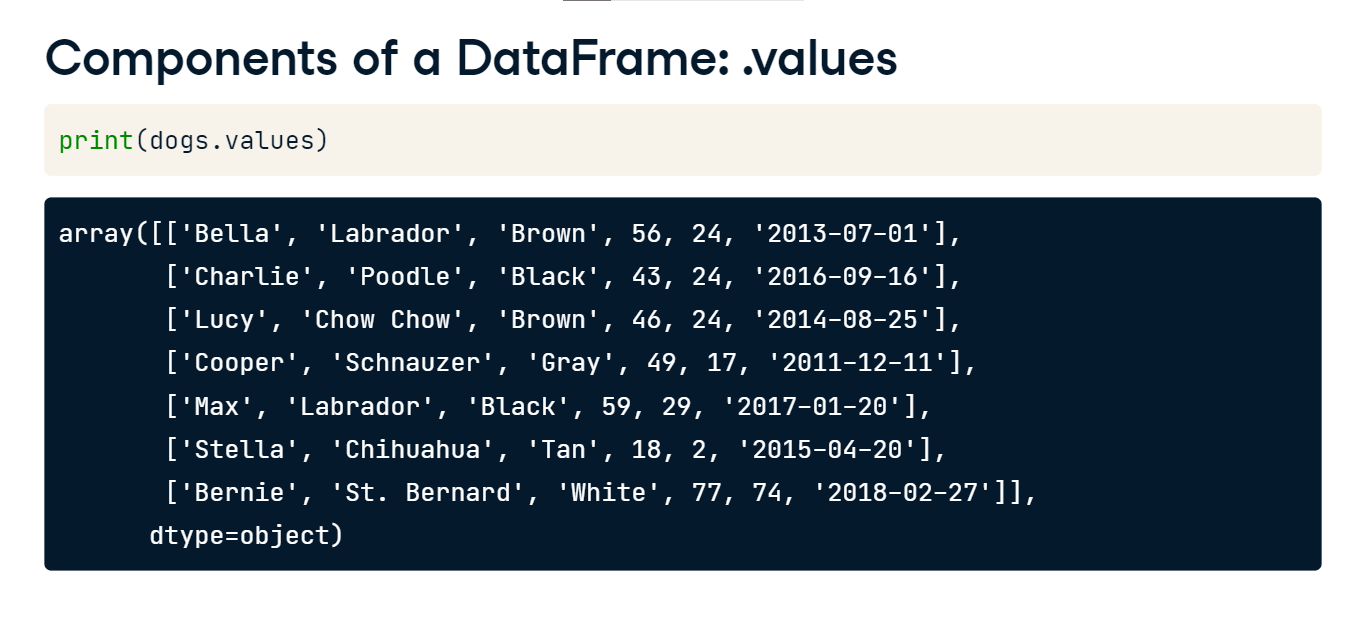
**Exploring a DataFrame: .describe()**

The describe method computes some summary statistics for numerical columns, like mean and median. "count" is the number of non-missing values in each column. describe is good for a quick overview of numeric variables, but if you want more control, you'll see how to perform more specific calculations later in the course.



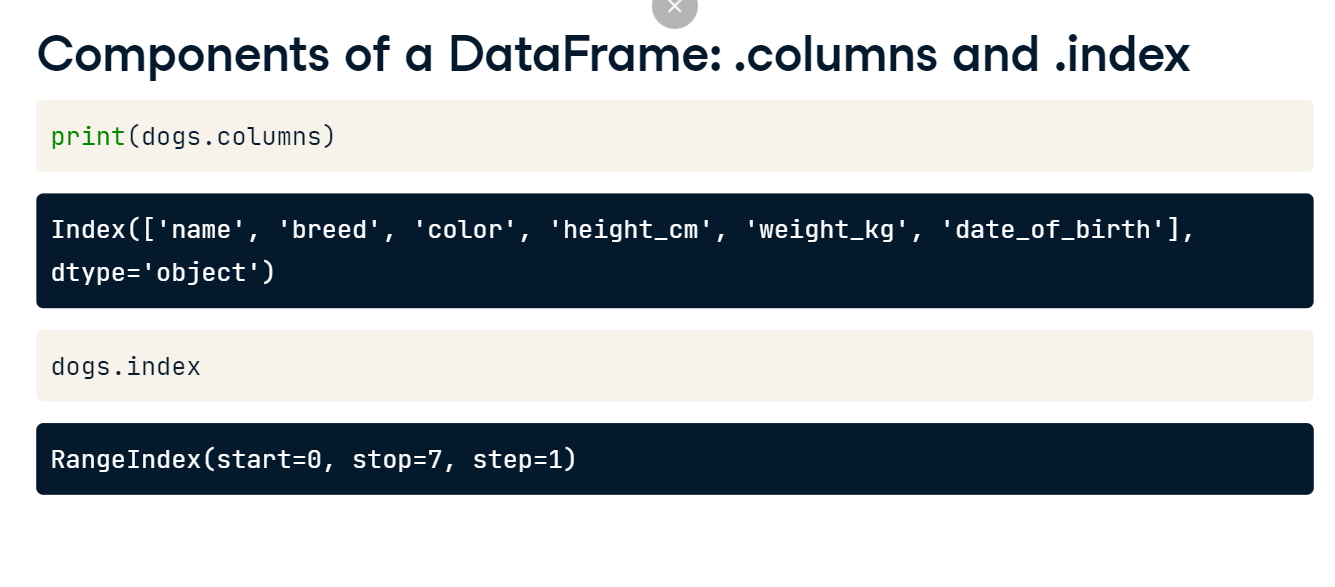
**Components of a DataFrame: .values**

DataFrames consist of three different components, accessible using attributes. The values attribute, as you might expect, contains the data values in a 2-dimensional NumPy array.



**Components of a DataFrame: .columns and .index**

The other two components of a DataFrame are labels for columns and rows. The columns attribute contains column names, and the index attribute contains row numbers or row names. Be careful, since row labels are stored in dot-index, not in dot-rows. Notice that these are Index objects, which we'll cover in Chapter 3. This allows for flexibility in labels. For example, the dogs data uses row numbers, but row names are also possible.



**pandas Philosophy**

Python has a semi-official philosophy on how to write good code called The Zen of Python. One suggestion is that given a programming problem, there should only be one obvious solution. As you go through this course, bear in mind that pandas deliberately doesn't follow this philosophy. Instead, there are often multiple ways to solve a problem, leaving you to choose the best. In this respect, pandas is like a Swiss Army Knife, giving you a variety of tools, making it incredibly powerful, but more difficult to learn. In this course, we aim for a more streamlined approach to pandas, only covering the most important ways of doing things.

1. 1 https://www.python.org/dev/peps/pep-0020/

**Sorting and subsetting**

In this video, we'll cover the two simplest and possibly most important ways to find interesting parts of your DataFrame.

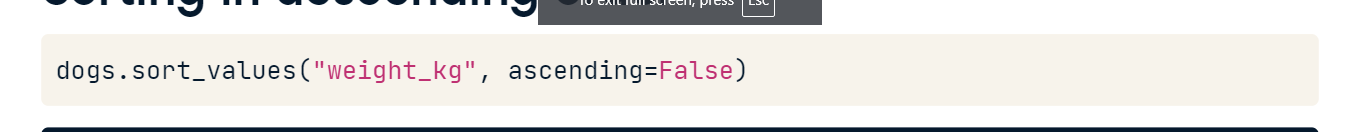
**Sorting**

The first thing you can do is change the order of the rows by sorting them so that the most interesting data is at the top of the DataFrame. You can sort rows using the sort\_values method, passing in a column name that you want to sort by. For example, when we apply sort\_values on the weight\_kg column of the dogs DataFrame, we get the lightest dog at the top, Stella the Chihuahua, and the heaviest dog at the bottom, Bernie the Saint Bernard.



**Sorting in descending order**

Setting the ascending argument to False will sort the data the other way around, from heaviest dog to lightest dog.

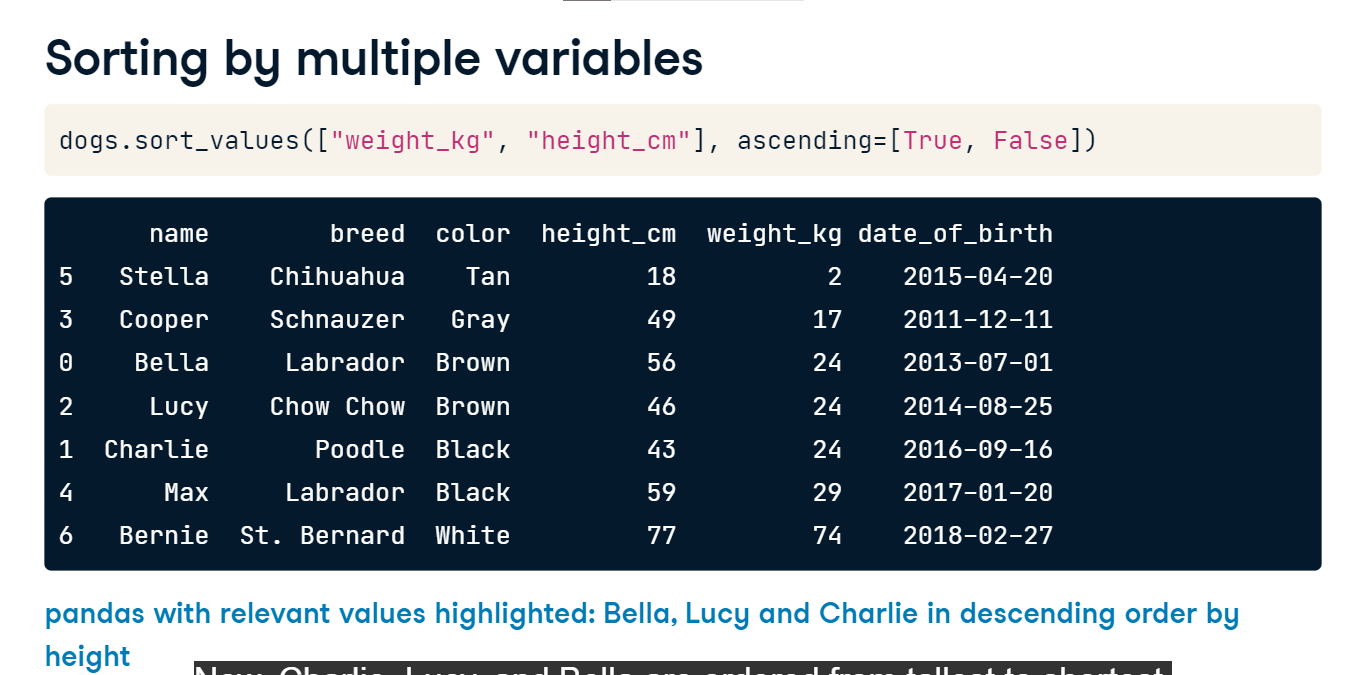


**Sorting by multiple variables**

We can sort by multiple variables by passing a list of column names to sort\_values. Here, we sort first by weight, then by height. Now, Charlie, Lucy, and Bella are ordered from shortest to tallest, even though they all weigh the same.

**Sorting by multiple variables**

To change the direction values are sorted in, pass a list to the ascending argument to specify which direction sorting should be done for each variable. Now, Charlie, Lucy, and Bella are ordered from tallest to shortest.



**Subsetting columns**

We may want to zoom in on just one column. We can do this using the name of the DataFrame, followed by square brackets with a column name inside. Here, we can look at just the name column.

**Subsetting multiple columns**

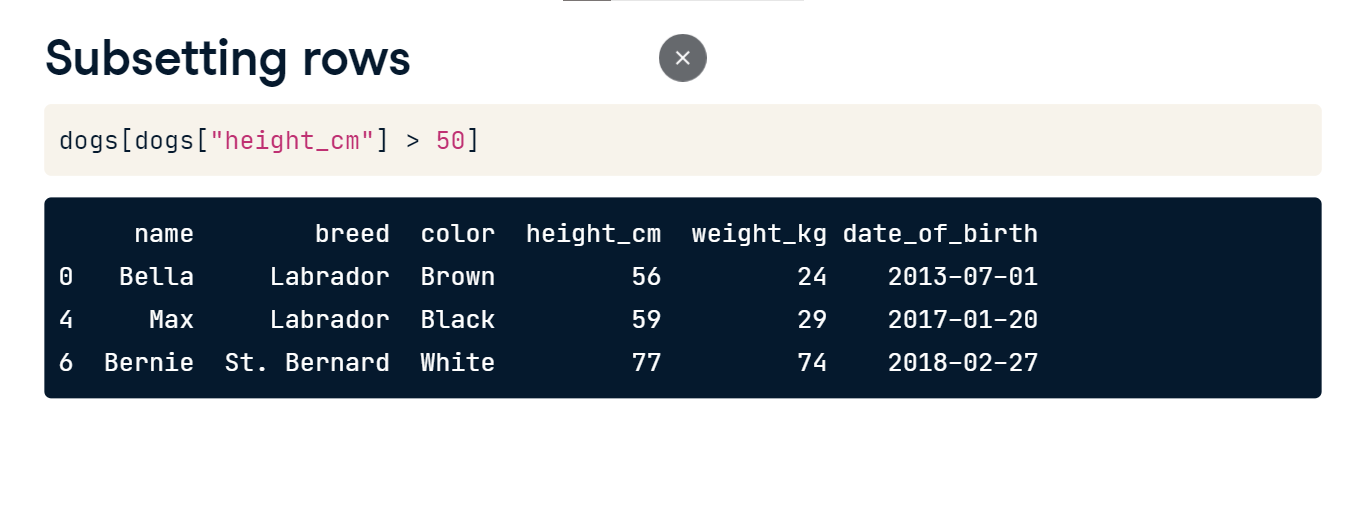
To select multiple columns, you need two pairs of square brackets. In this code, the inner and outer square brackets are performing different tasks. The outer square brackets are responsible for subsetting the DataFrame, and the inner square brackets are creating a list of column names to subset. This means you could provide a separate list of column names as a variable and then use that list to perform the same subsetting. Usually, it's easier to do in one line.



**Subsetting rows**

There are lots of different ways to subset rows. The most common way to do this is by creating a logical condition to filter against. For example, let's find all the dogs whose height is greater than 50 centimeters. Now we have a True or False value for every row.

We can use the logical condition inside of square brackets to subset the rows we're interested in to get all of the dogs taller than 50 centimeters.



**Subsetting based on text data**

We can also subset rows based on text data. Here, we use the double equal sign in the logical condition to filter the dogs that are Labradors.

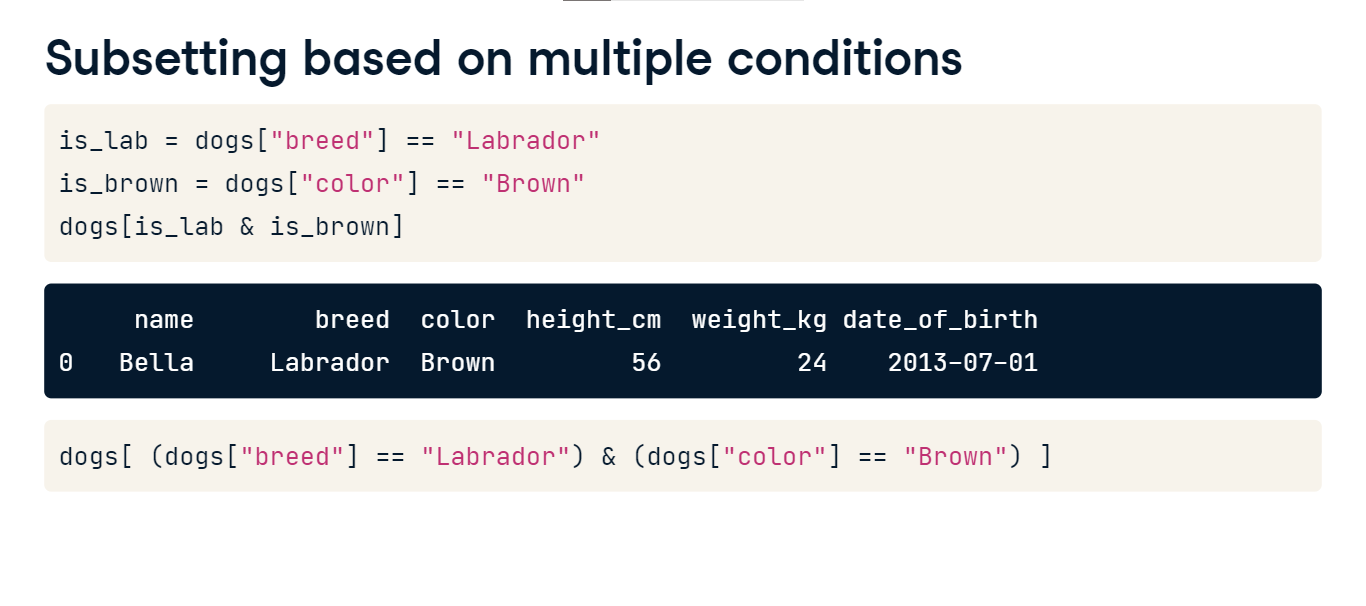
**Subsetting based on dates**

We can also subset based on dates. Here, we filter all the dogs born before 2015. Notice that the dates are in quotes and are written as year then month, then day. This is the international standard date format.



**Subsetting based on multiple conditions**

To subset the rows that meet multiple conditions, you can combine conditions using logical operators, such as the "and" operator seen here. This means that only rows that meet both of these conditions will be subsetted. You could also do this in one line of code, but you'll also need to add parentheses around each condition.



**Subsetting using .isin()**

If you want to filter on multiple values of a categorical variable, the easiest way is to use the isin method. This takes in a list of values to filter for. Here, we check if the color of a dog is black or brown, and use this condition to subset the data.



**New columns**

In the last lesson, you saw how to subset and sort a DataFrame to extract interesting bits. However, often when you first receive a DataFrame, the contents aren't exactly what you want. You may have to add new columns derived from existing columns.

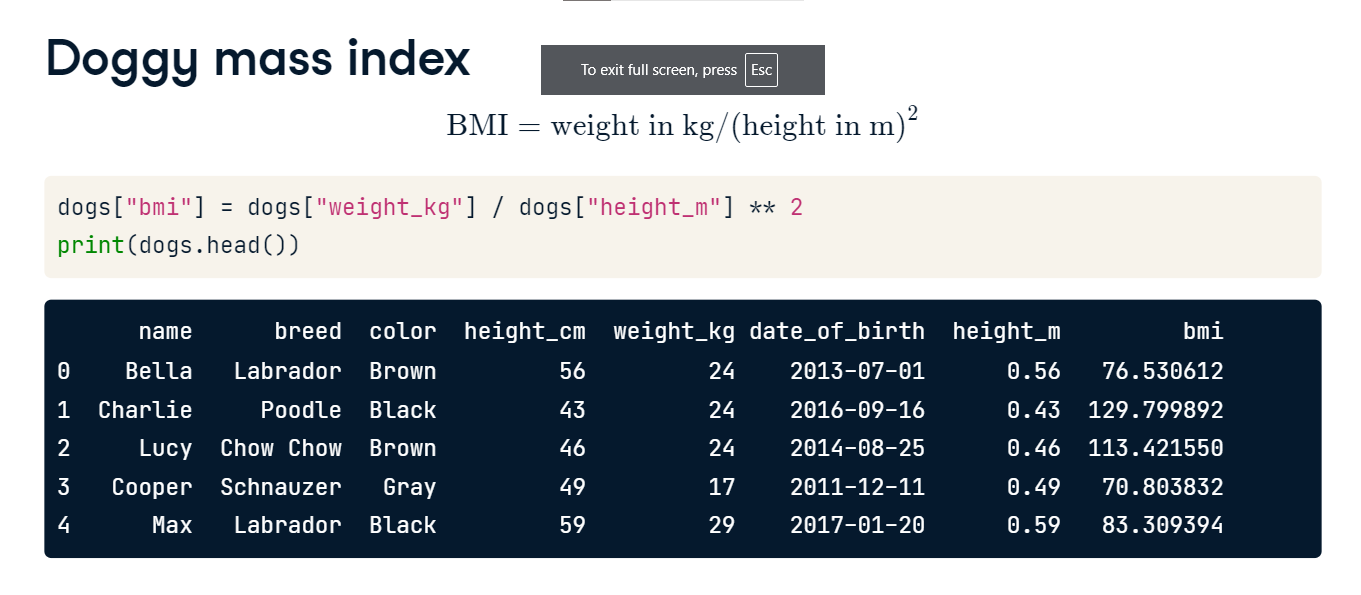
**Adding a new column**

Creating and adding new columns can go by many names, including mutating a DataFrame, transforming a DataFrame, and feature engineering. Let's say we want to add a new column to our DataFrame that has each dog's height in meters instead of centimeters. On the left-hand side of the equals, we use square brackets with the name of the new column we want to create. On the right-hand side, we have the calculation. Notice that both the existing column and the new column we just created are in the DataFrame.



**Doggy mass index**

Let's see what the results are if we calculate the body mass index, or BMI, of these dogs. BMI is usually calculated by taking a person's weight in kilograms and dividing it by their height in meters, squared. Instead of doing this with people, we'll try it out with dogs. Again, the new column is on the left-hand side of the equals, but this time, our calculation involves two columns.



**Multiple manipulations**

The real power of pandas comes in when you combine all the skills you've learned so far. Let's figure out the names of skinny, tall dogs. First, to define the skinny dogs, we take the subset of the dogs who have a BMI of under 100. Next, we sort the result in descending order of height to get the tallest skinny dogs at the top. Finally, we keep only the columns we're interested in. Here, you can see that Max is the tallest dog with a BMI of under 100.

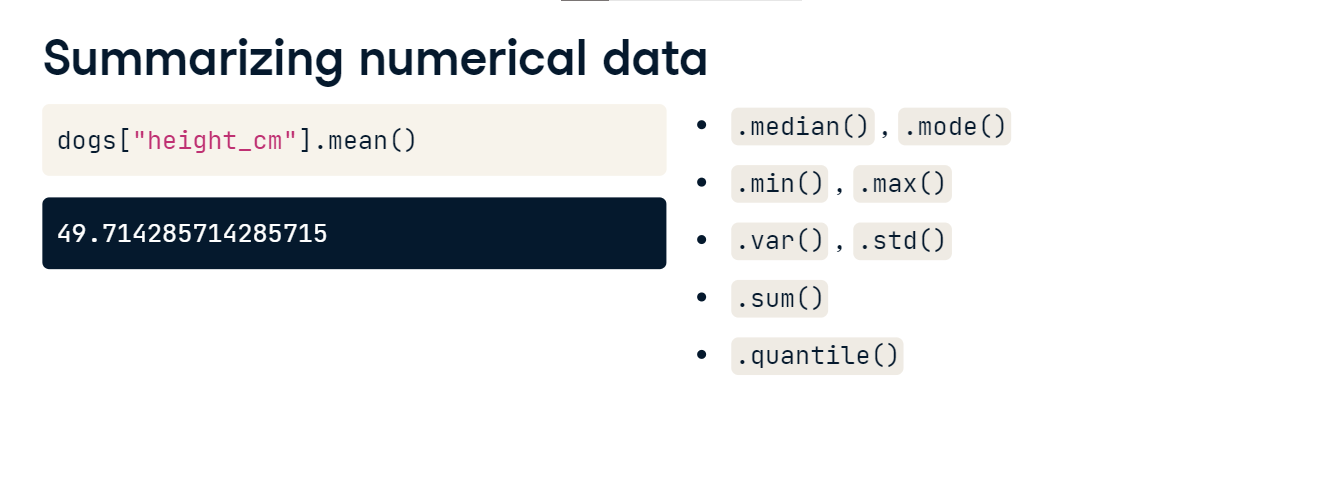


**Summary statistics**

Hi, I'm Maggie, and I'll be the other instructor for this course. In the first chapter, you learned about DataFrames, how to sort and subset them, and how to add new columns to them. In this chapter, we'll talk about aggregating data, starting with summary statistics. Summary statistics, as follows from their name, are numbers that summarize and tell you about your dataset.

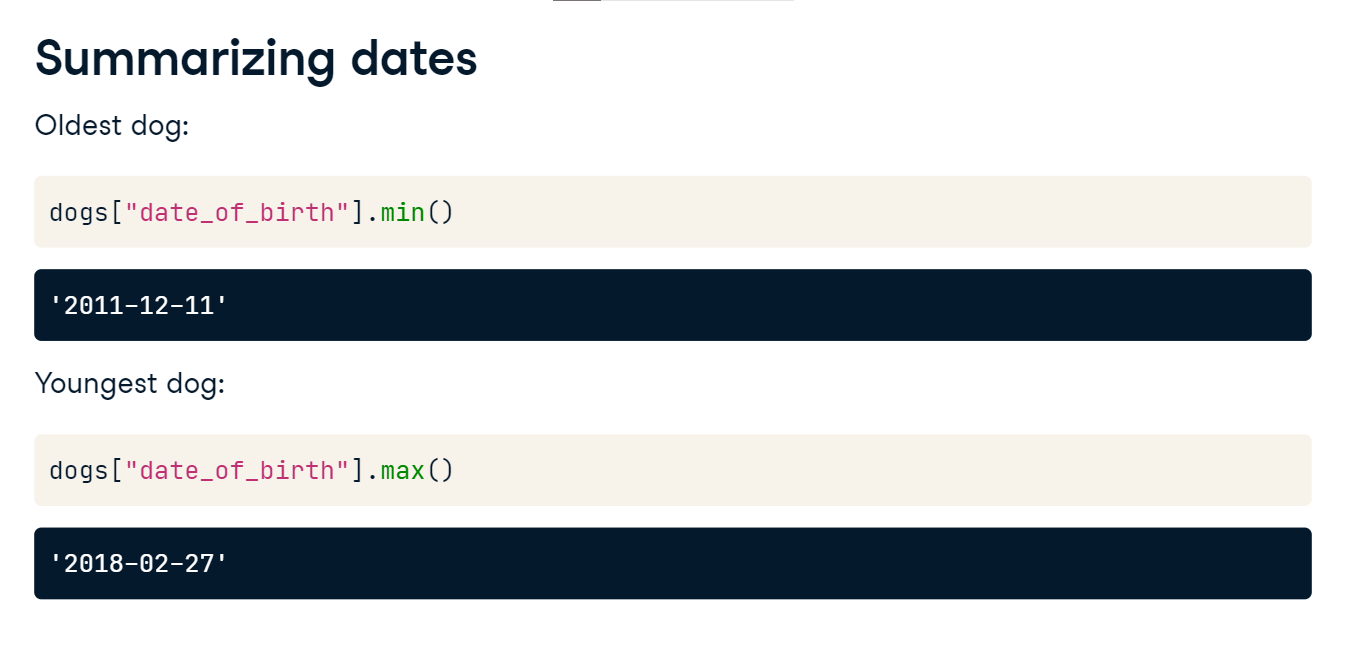
**Summarizing numerical data**

One of the most common summary statistics for numeric data is the mean, which is one way of telling you where the "center" of your data is. You can calculate the mean of a column by selecting the column with square brackets and calling dot-mean. There are lots of other summary statistics that you can compute on columns, like median and mode, minimum and maximum, and variance and standard deviation. You can also take sums and calculate quantiles.



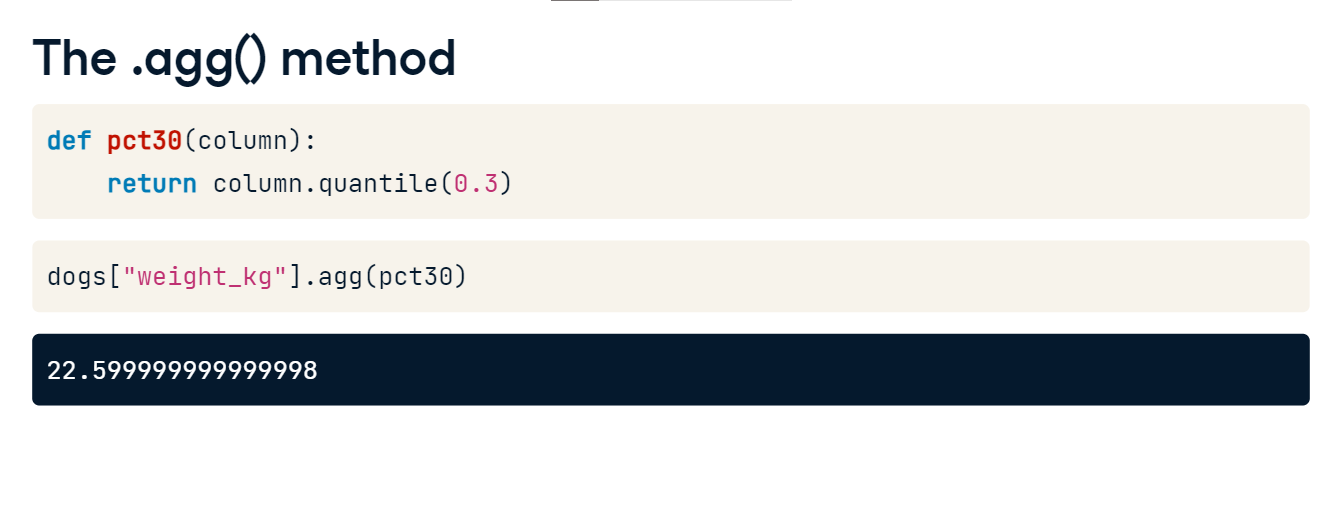
**Summarizing dates**

You can also get summary statistics for date columns. For example, we can find the oldest dog's date of birth by taking the minimum of the date of birth column. Similarly, we can take the maximum to see that the youngest dog was born in 2018.



**The .agg() method**

The aggregate, or agg, method allows you to compute custom summary statistics. Here, we create a function called pct30 that computes the thirtieth percentile of a DataFrame column. Don't worry if this code doesn't make sense to you -- just know that the function takes in a column and spits out the column's thirtieth percentile. Now we can subset the weight column and call dot-agg, passing in the name of our function, pct30. It gives us the thirtieth percentile of the dogs' weights.



**Summaries on multiple columns**

agg can also be used on more than one column. By selecting the weight and height columns before calling agg, we get the thirtieth percentile for both columns.



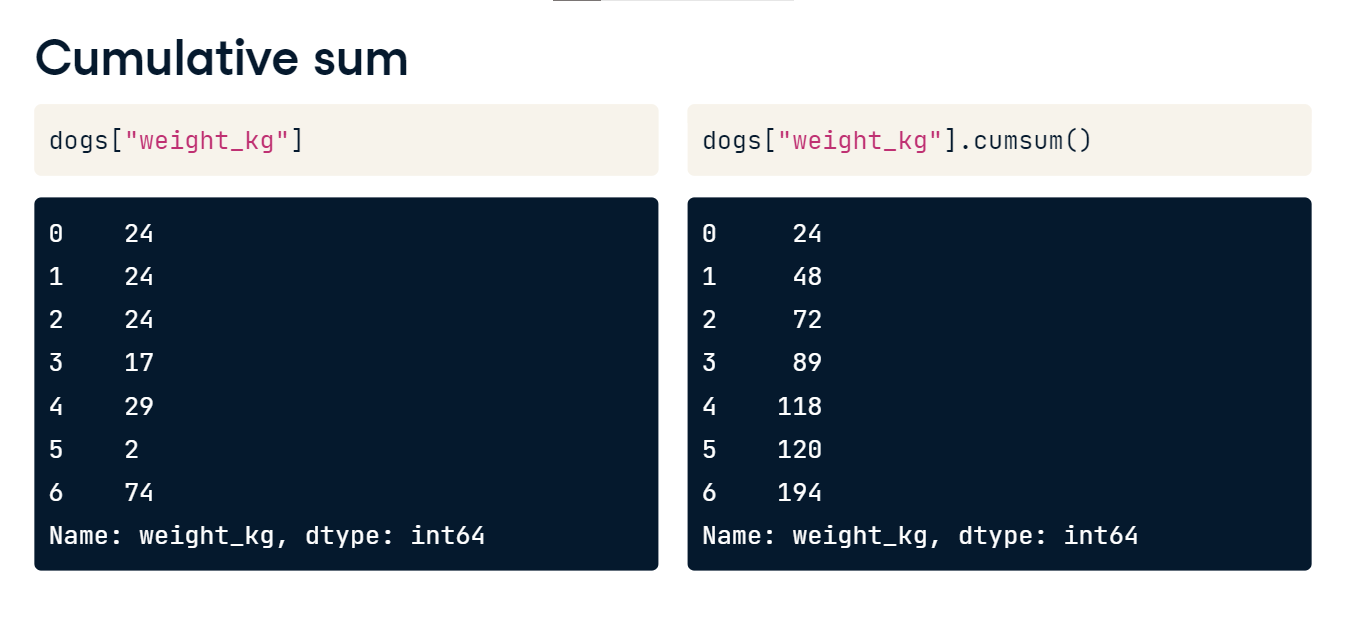
**Multiple summaries**

We can also use agg to get multiple summary statistics at once. Here's another function that computes the fortieth percentile called pct40. We can pass a list of functions into agg, in this case, pct30 and pct40, which will return the thirtieth and fortieth percentiles of the dogs' weights.



**Cumulative sum**

pandas also has methods for computing cumulative statistics, for example, the cumulative sum. Calling cumsum on a column returns not just one number, but a number for each row of the DataFrame. The first number returned, or the number in the zeroth index, is the first dog's weight. The next number is the sum of the first and second dogs' weights. The third number is the sum of the first, second, and third dogs' weights, and so on. The last number is the sum of all the dogs' weights.



**Cumulative statistics**

pandas also has methods for other cumulative statistics, such as the cumulative maximum, cumulative minimum, and the cumulative product. These all return an entire column of a DataFrame, rather than a single number.

**Walmart**

In this chapter, you'll be working with data on Walmart stores, which is a chain of department stores in the US. The dataset contains weekly sales in US dollars in various stores. Each store has an ID number and a specific store type. The sales are also separated by department ID. Along with weekly sales, there is information about whether it was a holiday week or not, the average temperature during the week in that location, the average fuel price in dollars per liter that week, and the national unemployment rate that week.

**Counting**

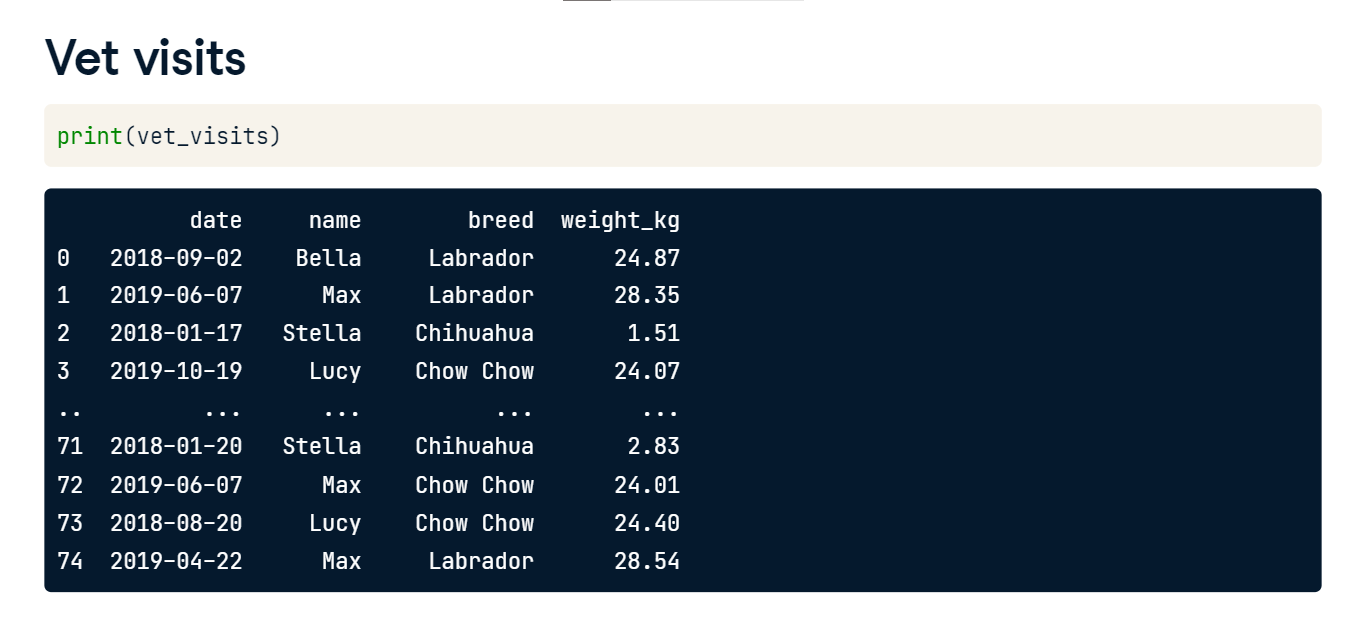
So far, in this chapter, you've learned how to summarize numeric variables. In this video, you'll learn how to summarize categorical data using counting.

**Avoiding double counting**

Counting dogs is no easy task when they're running around the park. It's hard to keep track of who you have and haven't counted!

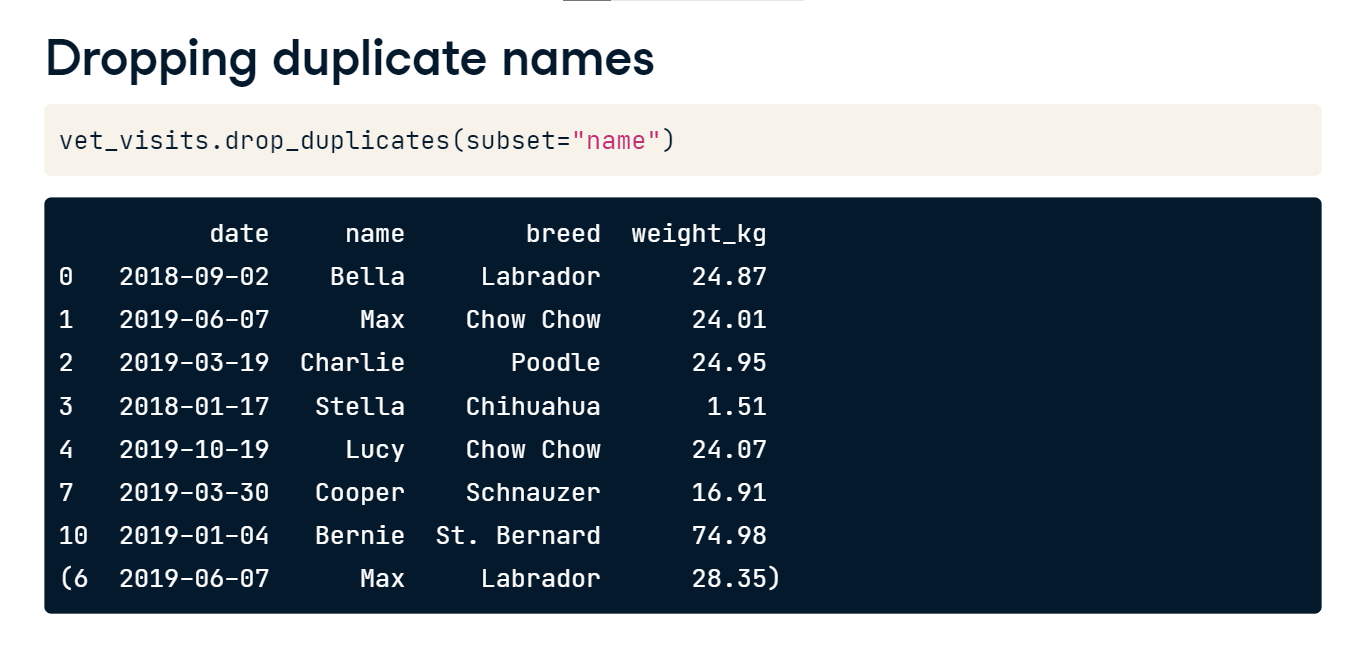
**Vet visits**

Here's a DataFrame that contains vet visits. The vet's office wants to know how many dogs of each breed have visited their office. However, some dogs have been to the vet more than once, like Max and Stella, so we can't just count the number of each breed in the breed column.



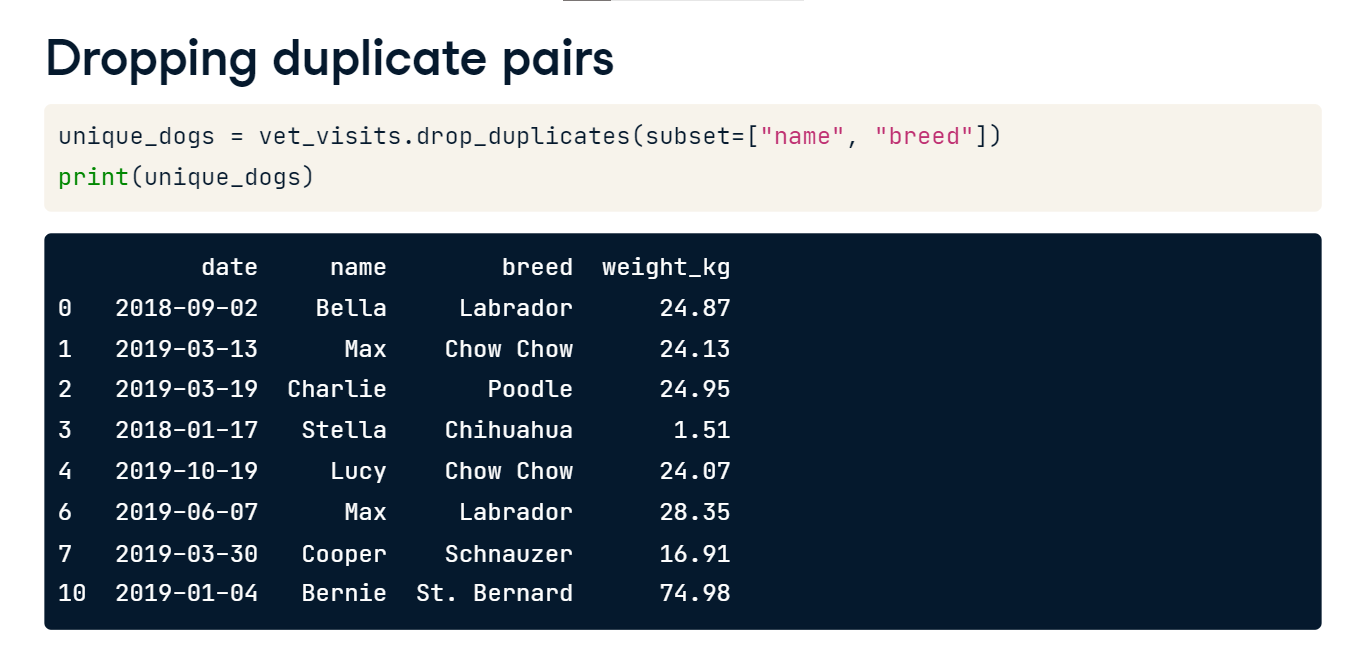
**Dropping duplicate names**

Let's try to fix this by removing rows that contain a dog name already listed earlier in the dataset, or in other words; we'll extract a dog with each name from the dataset once. We can do this using the drop\_duplicates method. It takes an argument, subset, which is the column we want to find our duplicates based on - in this case, we want all the unique names. Now we have a list of dogs where each one appears once. We have Max the Chow Chow, but where did Max the Labrador go? Because we have two different dogs with the same name, we'll need to consider more than just name when dropping duplicates.



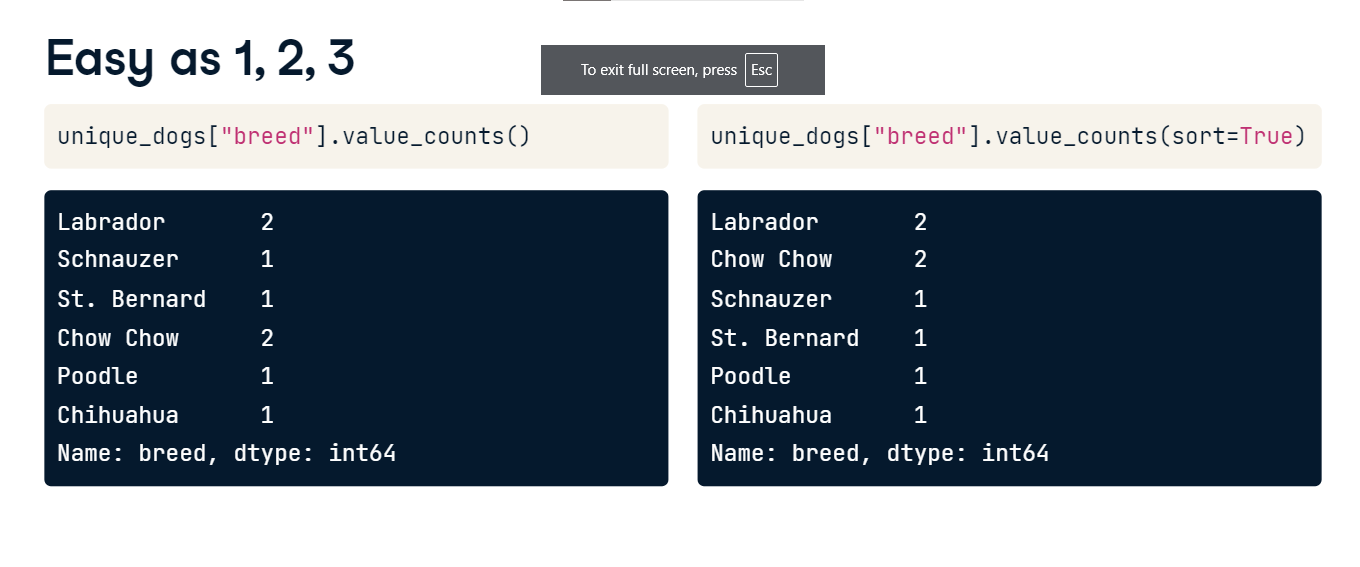
**Dropping duplicate pairs**

Since Max and Max are different breeds, we can drop the rows with pairs of name and breed listed earlier in the dataset. To base our duplicate dropping on multiple columns, we can pass a list of column names to the subset argument, in this case, name and breed. Now both Maxes have been included, and we can start counting.



**Easy as 1, 2, 3**

To count the dogs of each breed, we'll subset the breed column and use the value\_counts method. We can also use the sort argument to get the breeds with the biggest counts on top.



**Proportions**

The normalize argument can be used to turn the counts into proportions of the total. 25% of the dogs that go to this vet are Labradors.



**Grouped summary statistics**

So far, you've been calculating summary statistics for all rows of a dataset, but summary statistics can be useful to compare different groups.

**Summaries by group**

While computing summary statistics of entire columns may be useful, you can gain many insights from summaries of individual groups. For example, does one color of dog weigh more than another on average? Are female dogs taller than males? You can already answer these questions with what you've learned so far! We can subset the dogs into groups based on their color, and take the mean of each. But that's a lot of work, and the duplicated code means you can easily introduce copy and paste bugs.



**Grouped summaries**

That's where the groupby method comes in. We can group by the color variable, select the weight column, and take the mean. This will give us the mean weight for each dog color. This was just one line of code compared to the five we had to write before to get the same results.



**Multiple grouped summaries**

Just like with ungrouped summary statistics, we can use the agg method to get multiple statistics. Here, we pass a list of functions into agg after grouping by color. This gives us the minimum, maximum, and sum of the different colored dogs' weights.



**Grouping by multiple variables**

You can also group by multiple columns and calculate summary statistics. Here, we group by color and breed, select the weight column and take the mean. This gives us the mean weight of each breed of each color.



**Many groups, many summaries**

You can also group by multiple columns and aggregate by multiple columns.



**Pivot tables**

Pivot tables are another way of calculating grouped summary statistics. If you've ever used a spreadsheet, chances are you've used a pivot table. Let's see how to create pivot tables in pandas.

**Group by to pivot table**

In the last lesson, we grouped the dogs by color and calculated their mean weights. We can do the same thing using the pivot\_table method. The "values" argument is the column that you want to summarize, and the index column is the column that you want to group by. By default, pivot\_table takes the mean value for each group.



**Different statistics**

If we want a different summary statistic, we can use the aggfunc argument and pass it a function. Here, we take the median for each dog color using NumPy's median function.



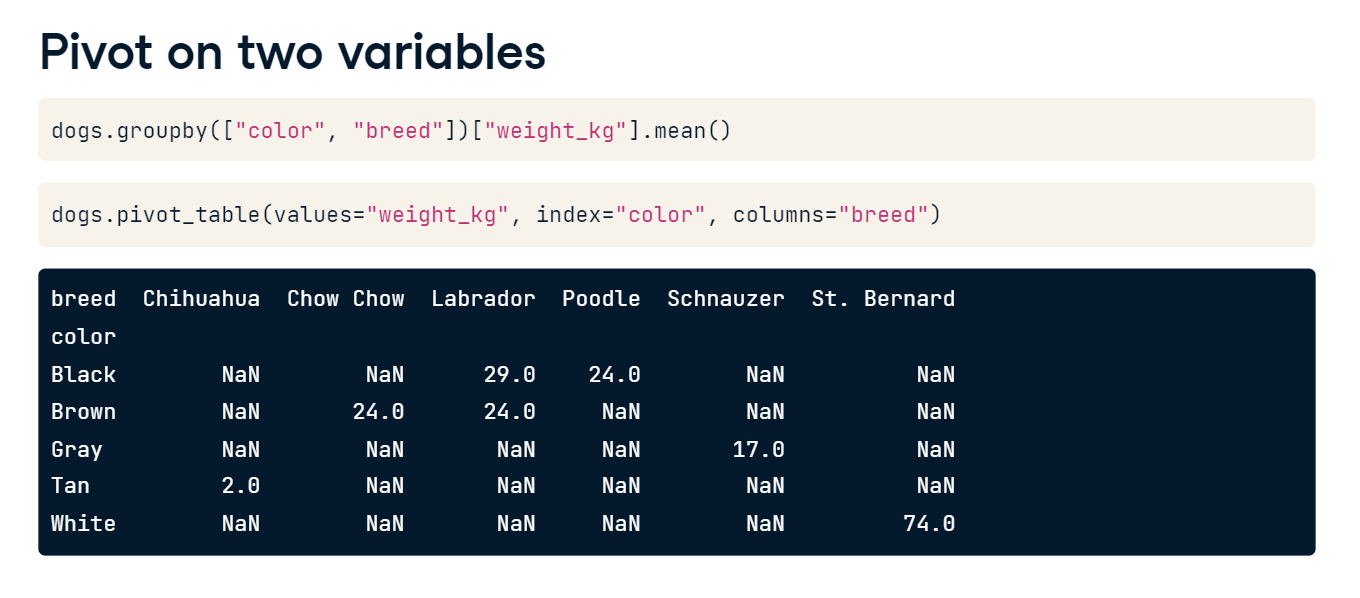
**Multiple statistics**

To get multiple summary statistics at a time, we can pass a list of functions to the aggfunc argument. Here, we get the mean and median for each dog color.



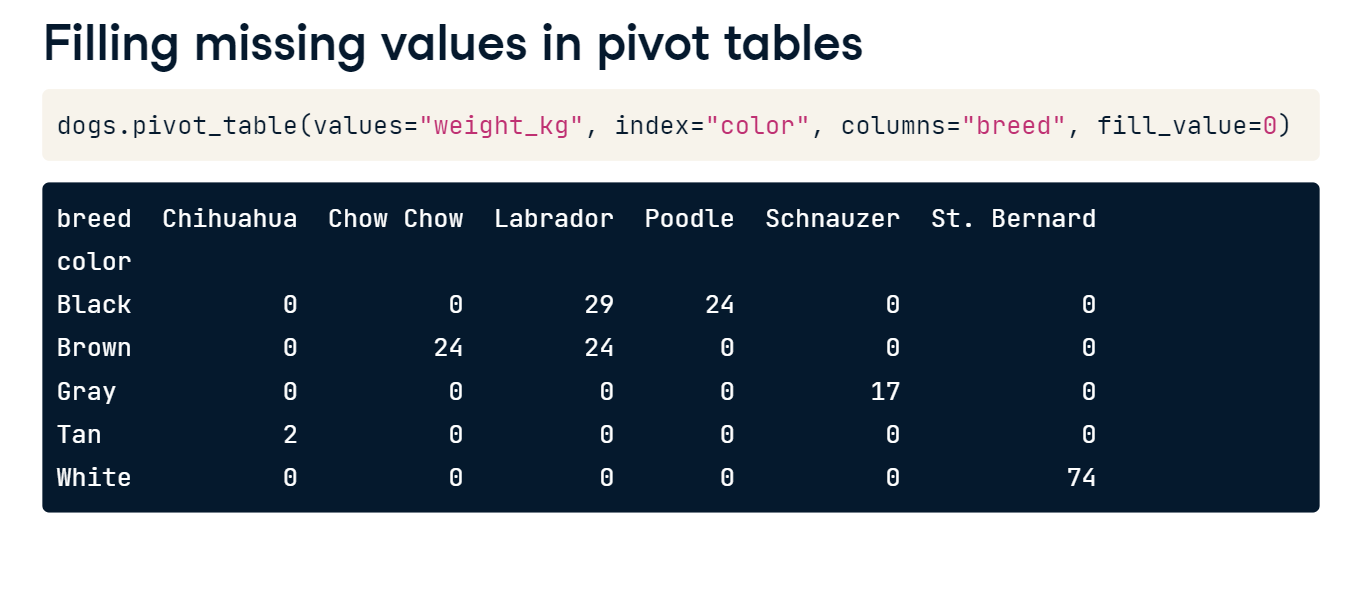
**Pivot on two variables**

You also previously computed the mean weight grouped by two variables: color and breed. We can also do this using the pivot\_table method. To group by two variables, we can pass a second variable name into the columns argument. While the result looks a little different than what we had before, it contains the same numbers. There are NaNs, or missing values, because there are no black Chihuahuas or gray Labradors in our dataset, for example.



**Filling missing values in pivot tables**

Instead of having lots of missing values in our pivot table, we can have them filled in using the fill\_value argument. Here, all of the NaNs get filled in with zeros.



**Summing with pivot tables**

If we set the margins argument to True, the last row and last column of the pivot table contain the mean of all the values in the column or row, not including the missing values that were filled in with Os. For example, in the last row of the Labrador column, we can see that the mean weight of the Labradors is 26 kilograms. In the last column of the Brown row, the mean weight of the Brown dogs is 24 kilograms. The value in the bottom right, in the last row and last column, is the mean weight of all the dogs in the dataset. Using margins equals True allows us to see a summary statistic for multiple levels of the dataset: the entire dataset, grouped by one variable, by another variable, and by two variables.

