**What is pandas?**

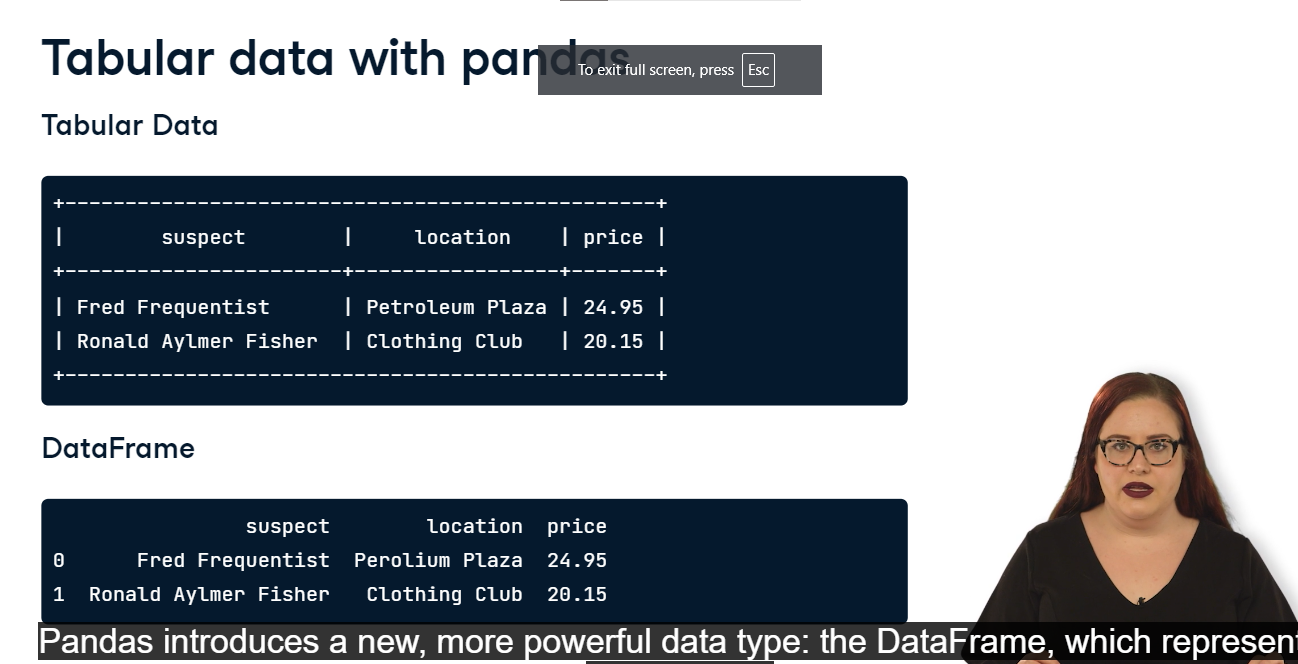
In the previous chapter, you learned about basic Python, including defining variables and loading modules. One of the modules that you loaded was called pandas. Pandas is a module for working with tabular data, or data that has rows and columns. Common examples of tabular data are spreadsheets or database tables. In this chapter, you'll learn the basics of Pandas, including how to load and inspect data.

**What can pandas do for you?**

Pandas gives you many tools for working with tabular data. You can load tabular data from multiple sources (like spreadsheets or databases), search for particular rows or columns in your loaded data, calculate aggregate statistics (like averages or standard deviations), and combine data from multiple sources.

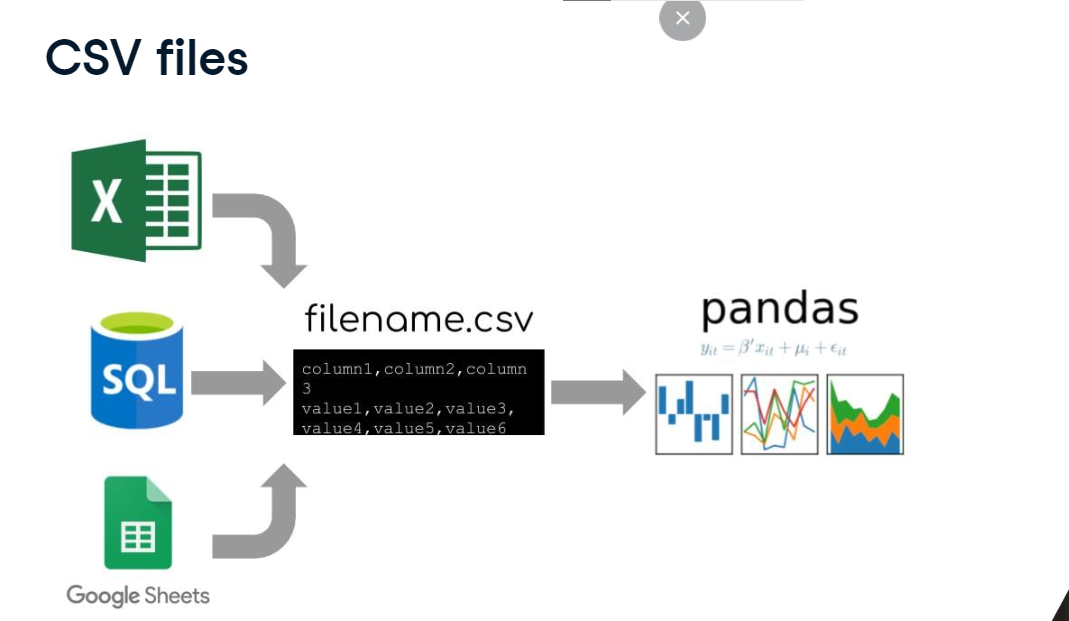
**Tabular data with pandas**

You already know two data types: floats and strings. Pandas introduces a new, more powerful data type: the DataFrame, which represents tabular data. Loading data into a DataFrame is the first step in using Pandas.



**CSV files**

One of the easiest ways to create a DataFrame is by using a CSV file. CSV stands for comma-separated values, and is a common way of storing tabular data as a text-only file. You can download a CSV version of data from an Excel spreadsheet, a SQL database, or a Google Sheet.



**Loading a CSV**

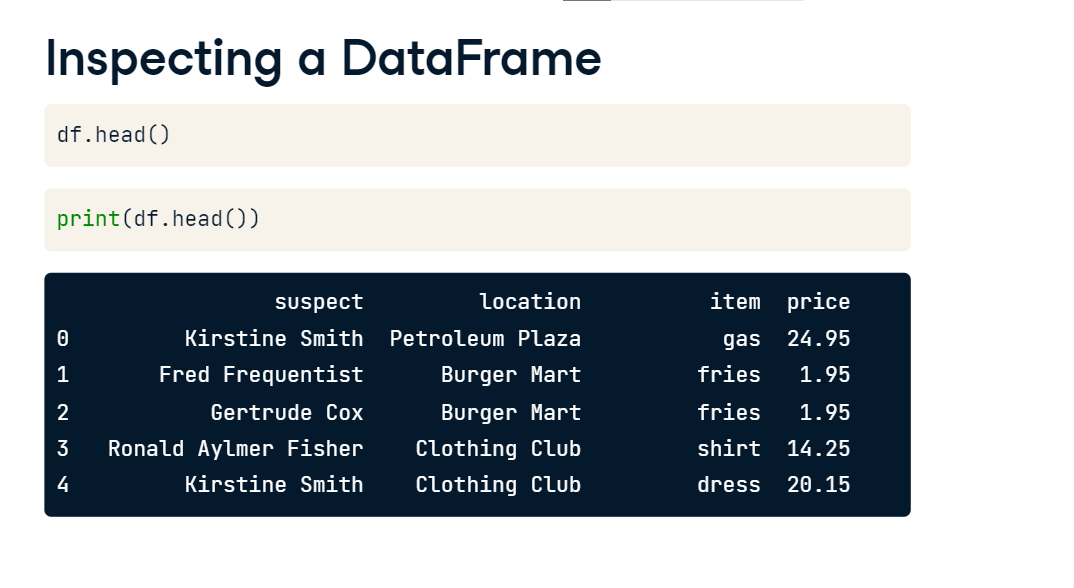
Before we can start using Pandas, we have to import the pandas module. Recall that we always import Pandas under the alias "pd". Next, we create our first DataFrame from a CSV. Turning a CSV into a DataFrame is easy. We use a function: pd-dot-read\_csv. This function takes one argument, the name of a CSV file as a string. In this example, the name of the file is ransom-dot-csv.

**Displaying a DataFrame**

Notice that we saved this DataFrame to a variable called "df". We can display this variable by using the "print" function, which we learned about in a previous lesson. When we print a DataFrame, we get to see every row in the DataFrame. In this example, the DataFrame is so large that it doesn't entirely fit on the slide!

**7. Inspecting a DataFrame**

Usually, we don't want to print an entire DataFrame to inspect it; we just want to view the first few lines. We can do this by using head. head behaves slightly differently than the functions we've learned about previously. Rather than coming at the beginning of a line of code, this function "belongs" to the DataFrame variable and comes at the end of the line, after a period. We call this type of function a "method". The .head method just selects the first five rows of "df". In order to display then, we need to put df-dot-head inside of the print function. Printing the first few lines of a DataFrame using .head is much easier to read than printing the entire DataFrame.



Another way of learning about a DataFrame is using the info method. Like head, we type the variable name of the DataFrame (in this case "df"), followed by a period, and then followed by our method dot-info. Again, in order to display the output, we put the whole statement inside of a print function. Let's take a closer look at the output of info.

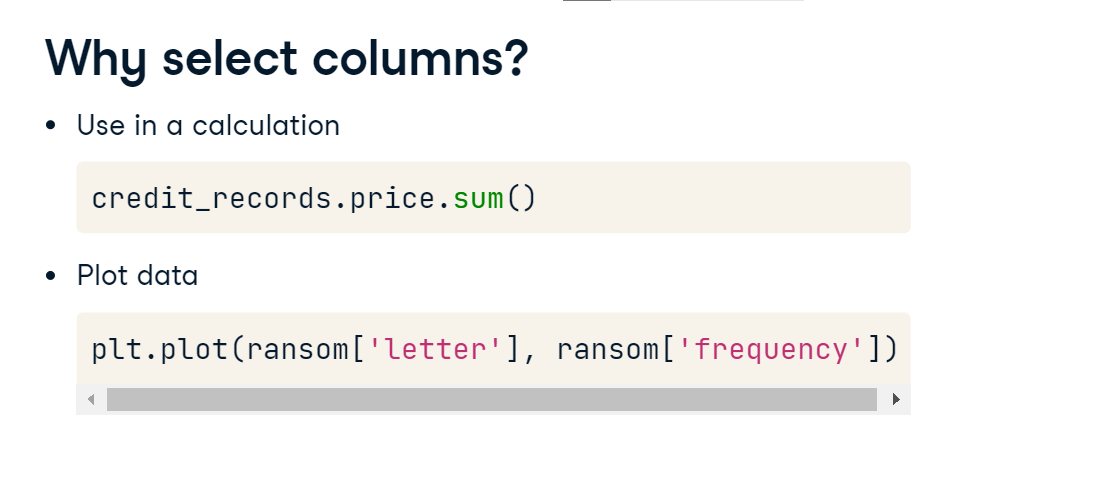
Notice that we can see the names of the columns, the number of rows, and data types for each column. This method is particularly useful for DataFrames with many columns that are difficult to display using head.

**Selecting columns**

Now that we've loaded data from a CSV into a DataFrame, we're going to want to access that data. The first way we'll use that data is by selecting columns. This means we'll be getting all values from a particular column in a DataFrame. In this lesson, we'll continue working with the credit card reports for our suspects in the kidnapping of Bayes, the Golden Retriever, as well as a few other pieces of evidence from the case.

**Why select columns?**

Why do we want to select columns? We might use them in a calculation. For example, this code selects the column "price" and then calls the method "sum" on that column to get the total amount of money spent by our suspects. We might also want to use the data as the input to a function. You might recognize this code from when we learned about functions. It will create a plot of the frequencies of each letter in the ransom note. The data comes from a DataFrame called "ransom" with columns "letter" and "frequency".



**Columns names are strings**

So how do we select columns? We know that we can view the column names for our data using either dot-head or dot-info. If we examine the credit\_records DataFrame, we see that there are 5 columns: suspect, location, date, item, and price. Each of these column names is a string. Recall that a string is just a way of representing text in Python. In this case the strings are all single words, but strings can have spaces or special characters. That means that columns in DataFrames can also have spaces or special characters.



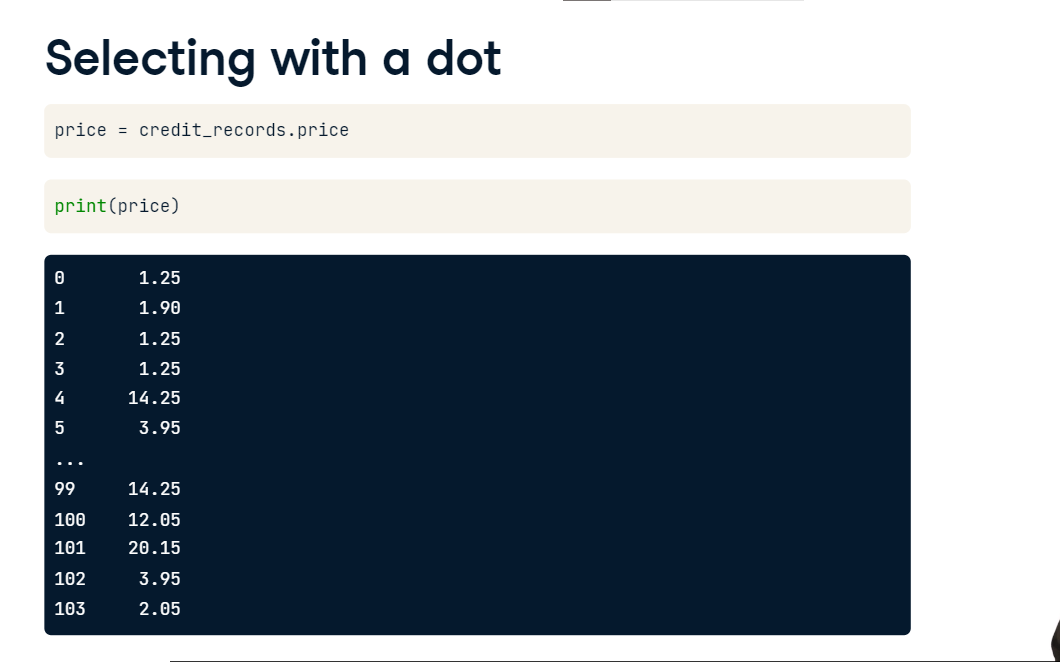
**Selecting with brackets and string**

We can use these column name strings to select a column. We type the name of the DataFrame, followed by an open square bracket, followed by the column string (in quotation marks), and closed with a second square bracket. If we print the column, we see all values for that particular column.



**Selecting with a dot**

There's a second way we can select columns from a DataFrame. If the columns string only contains letters, numbers, and underscores, we can use dot notation. For dot notation, we simply type the name of the variable, followed by a dot, followed by the name of the column. In this case, we don't use quotation marks around the column name.



**Common mistakes in column selection**

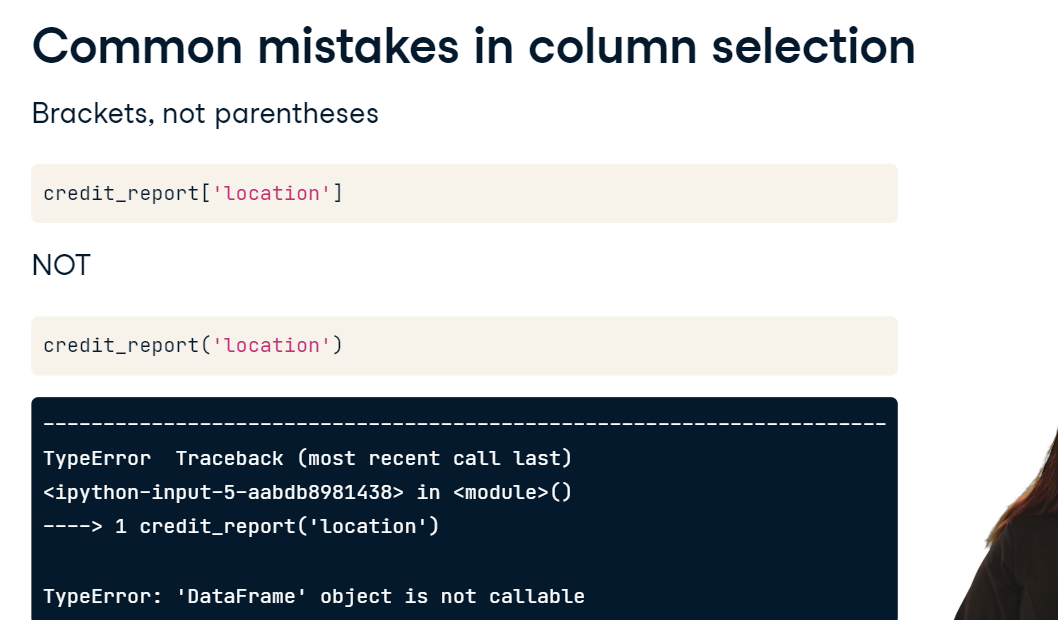
Let's go over some common errors you might encounter when selecting columns. Remember that when your column name contains spaces or special characters, you need to use bracket and string notation, and not dot notation to select the column. If you use dot notation, Python gets confused and thinks that you're referring to several different variables.



Another common error is forgetting to put quotation marks around the column string when using brackets and string notation. If you forget the quotation marks, then Python thinks that the column name is actually a variable that hasn't been defined yet.



Finally, remember that square brackets are not the same at parentheses. If you use parentheses, Python will think that you're using the DataFrame as a function, and will give a "TypeError".



**Select rows with logic**

Welcome back. Now that we know how to select columns of a DataFrame, let's learn how to select specific rows.

**Continuing the investigation**

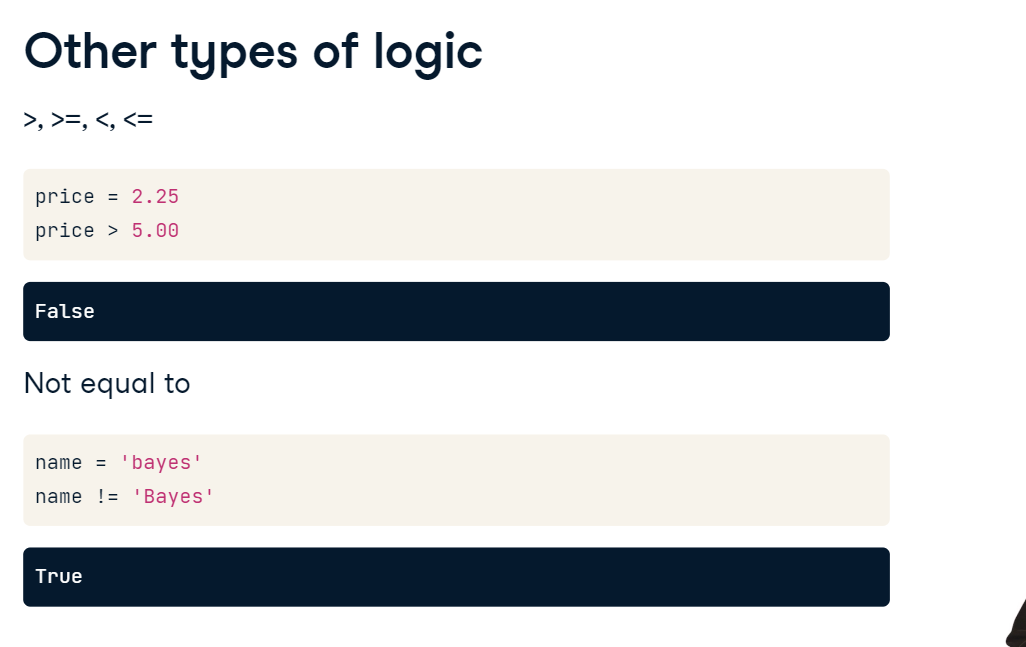
We've collected credit card statements from all of our suspects into a DataFrame called "credit\_records". If we can prove that some of these suspects made suspicious purchases, we can ask them for writing samples and match those samples to the ransom note found at the scene of the crime. In order to search for the suspicious purchases, we'll need to select rows from the DataFrame using logical statements.

**Logical statements in Python**

In Python, a logical statement checks for a relationship between two values (such as "equal to" or "greater than"), and returns True or False. For example, suppose we want to know if 12 times 8 is equal to 96. We can compare the variables "question" and "solution" using a double equals sign. This tests whether the two numbers are the same. In this case, Python returns "True" because 12 times 8 is equal to 96. That "True" that Python returns is a special data type called a "Boolean". There are only two Booleans: True and False. Recall that you've already learned three other data types: floats, strings, and DataFrames. We'll be using Booleans to select rows where the value of a column matches a specific value.

**Other types of logic**

We can make many types of comparisons in Python, such as greater than, greater than or equal to, less than, and less than or equal to. In this example, we ask if the variable "price" is greater than 5. Python returns False because 2-point-25 is not greater than 5. We can also check if two variables are not equal to each other. We do this using exclamation point followed by an equals sign. In this example, we see that lowercase "bayes" is not the same as title-case "Bayes".



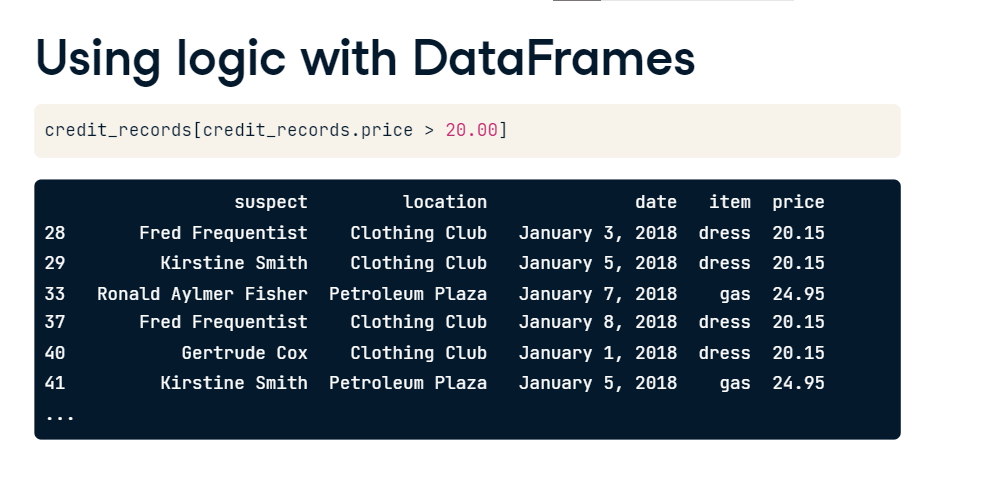
**Using logic with DataFrames**

In the previous examples, we were just comparing two values. In a DataFrame we can compare one value to all values in a DataFrame. For example, we can compare if each purchase in our credit cared records had a price that was greater than $20. This returns an entire column of True or False.

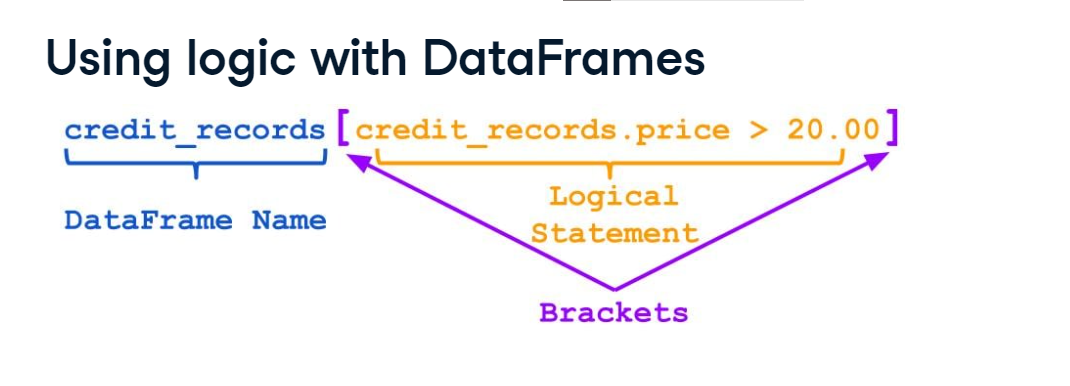


**Using logic with DataFrames**

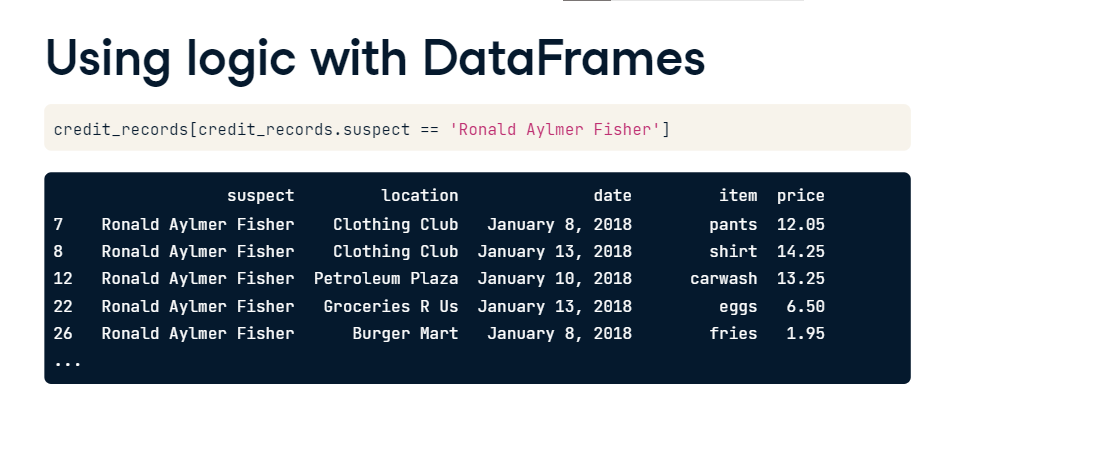
If we put that truth testing statement inside of brackets, we can select only the rows where the statement is True. Here, we select rows of credit\_records where the price is greater than 20 dollars.



Let's examine this line of code more closely. We start with the name of the DataFrame we want to select rows from. In this case, "credit\_records". Next we have a set of square brackets. Inside of the square brackets we put our logical test. In this case, the logical test is whether the "price" column of "credit records" is greater than $20. This statement will select all rows of credit\_records where the column price is greater than $20.



Let's try another example. Suppose, we want to select all rows of credit\_records where the "suspect" is equal to "Ronald Aylmer Fisher". Again, we have the name of the DataFrame, followed by square brackets. Inside of the brackets, we check if credit\_records-dot-suspect is equal to "Ronald Aylmer Fisher". Note that we use the double equals sign to test equality.



**Creating line plots**

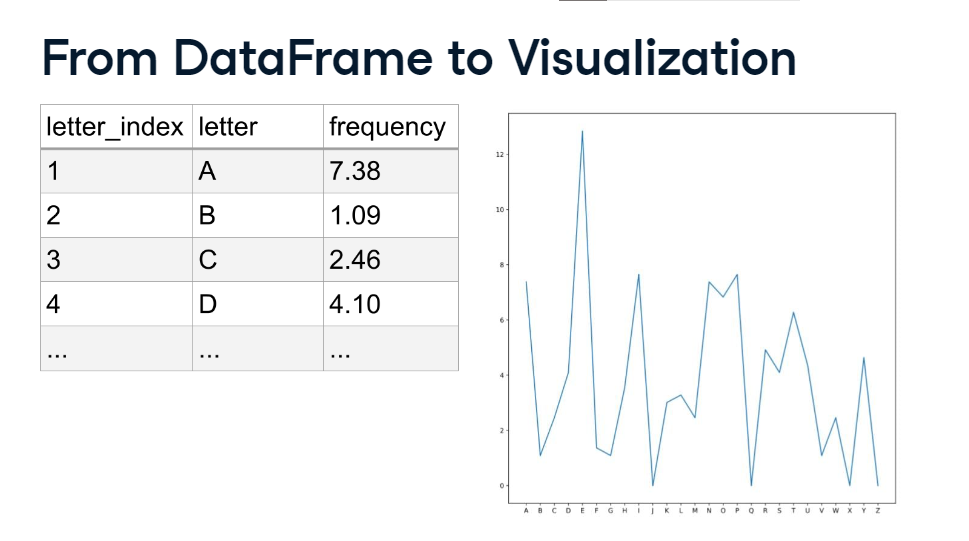
In this video, we'll learn how to create line plots using another Python module: matplotlib. You've seen matplotlib before in Chapter 1, when you learned about the function plt-dot-plot. Recall that plt is the alias that we use when importing matplotlib.

**The plot thickens**

We're close to solving the mystery of Bayes, the kidnapped golden retriever. Remember that a ransom note was found at the scene of the crime. In the previous chapter, we narrowed the list of suspects to Freddie Frequentist and Gertrude Cox. We've asked Freddie and Gertrude to give us writing samples to compare against the ransom note. We'll compare the letter frequencies in Freddie and Gertrude's samples to the note found at the scene of the crime. We'll do this by creating line plots.

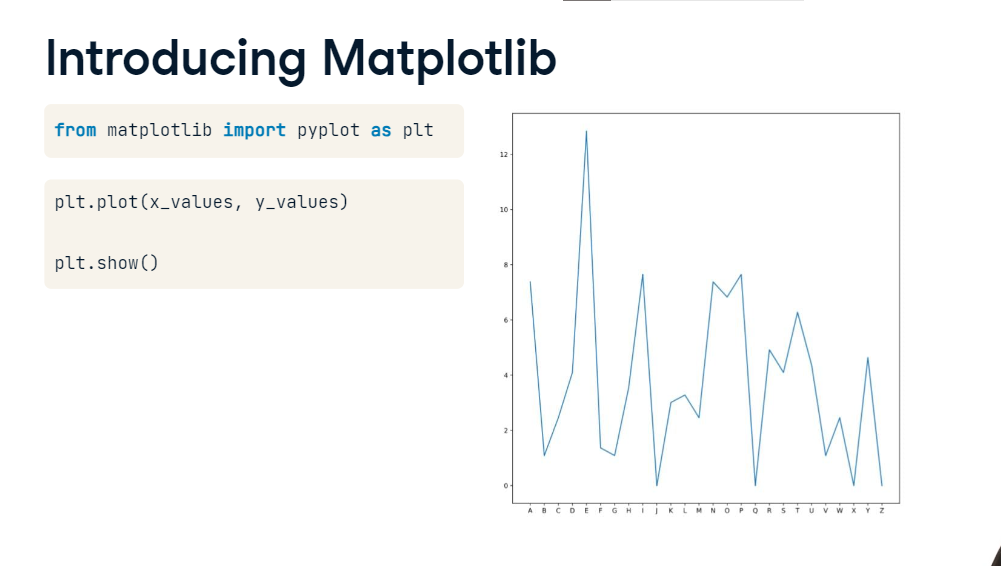
**From DataFrame to Visualization**

We've stored the letter frequencies in a Pandas DataFrame. We'll need to transform that DataFrame into a two-dimensional line plot. A line plot uses a coordinate grid to plot a series of points and then connects each point using a line. In this case, the x-axis represents the letters of the alphabet and the y-axis represents the frequency of each letter.



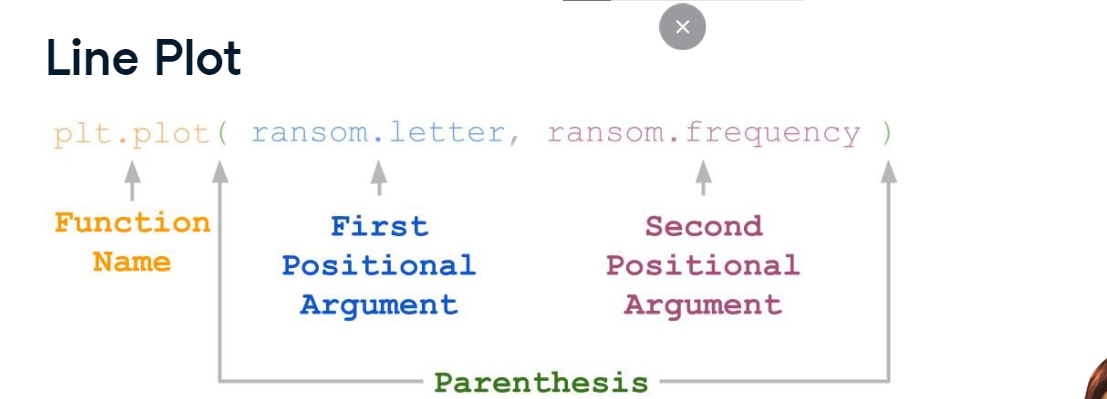
**Introducing Matplotlib**

In order to create a line plot, we'll need to import the module matplotlib. Generally, Data Scientists only load in the submodule pyplot and use the alias plt. We do this by typing "from matplotlib import pyplot as plt". Once we've loaded our module, we can use the functions plt-dot-plot and plt-dot-show to create and display our line plot.



**Line Plot**

Let's examine the line plot function. We begin with the name of the function: plt-dot-plot Like all functions, we'll be putting the arguments inside of parentheses. The first argument is the x-values we want to plot. In this case, we use the letter column from the DataFrame ransom. Recall that we select a column by typing the name of the DataFrame, followed by a dot, followed by the column name. The second argument is the y-values we want to plot. In this case, we use the frequency column from the DataFrame ransom. Again, we type DataFrame name, then dot, then column name. Finally, we close the parentheses.



**Displaying the Results**

If we only type plt-dot-plot, nothing will show up. That's because Python wants to give us the opportunity to enter other commands. For example, we might want to add another line or a title for the plot. When we want to display everything that we've made, we use a second function: plt-dot-show. This function takes no arguments. You just type it, and then your plot will pop open in a new window.

**Multiple Lines**

If we want to plot multiple lines on the same axis, we can just add a second plt-dot-plot command before adding plt-dot-show. In this example, we plotted two different datasets: data1, which represents a straight line and data2, which represents a parabola. Matplotlib will automatically give the two lines different colors.

