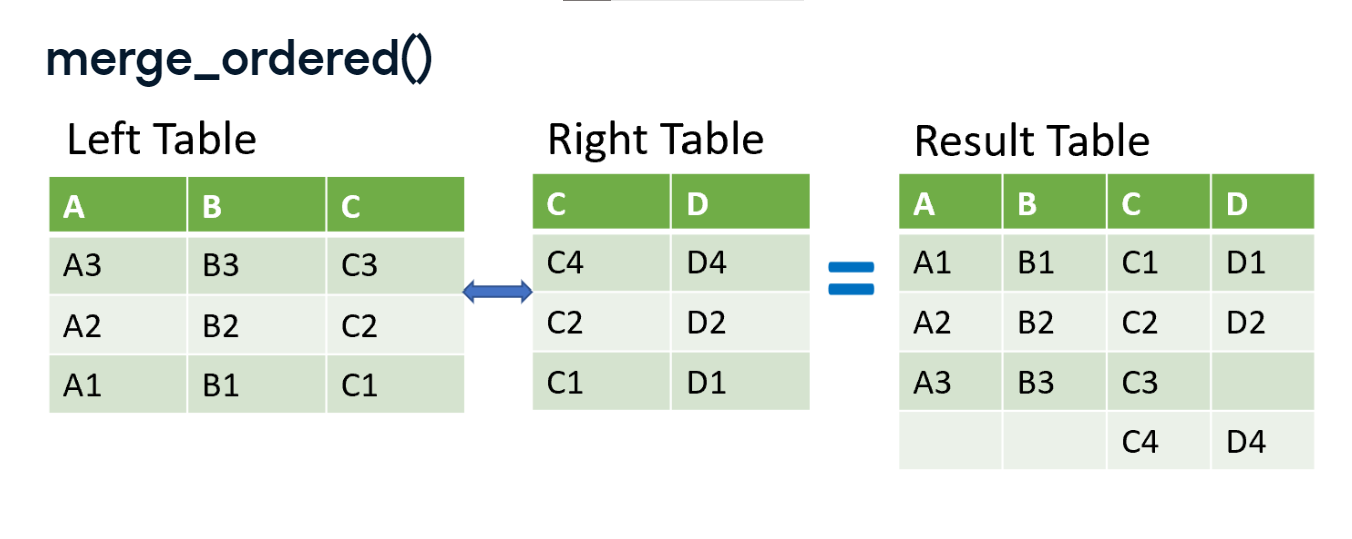
**Using merge\_ordered()**

Welcome back! In this last chapter we'll start discussing merge\_ordered(). This method can merge time-series and other ordered data.

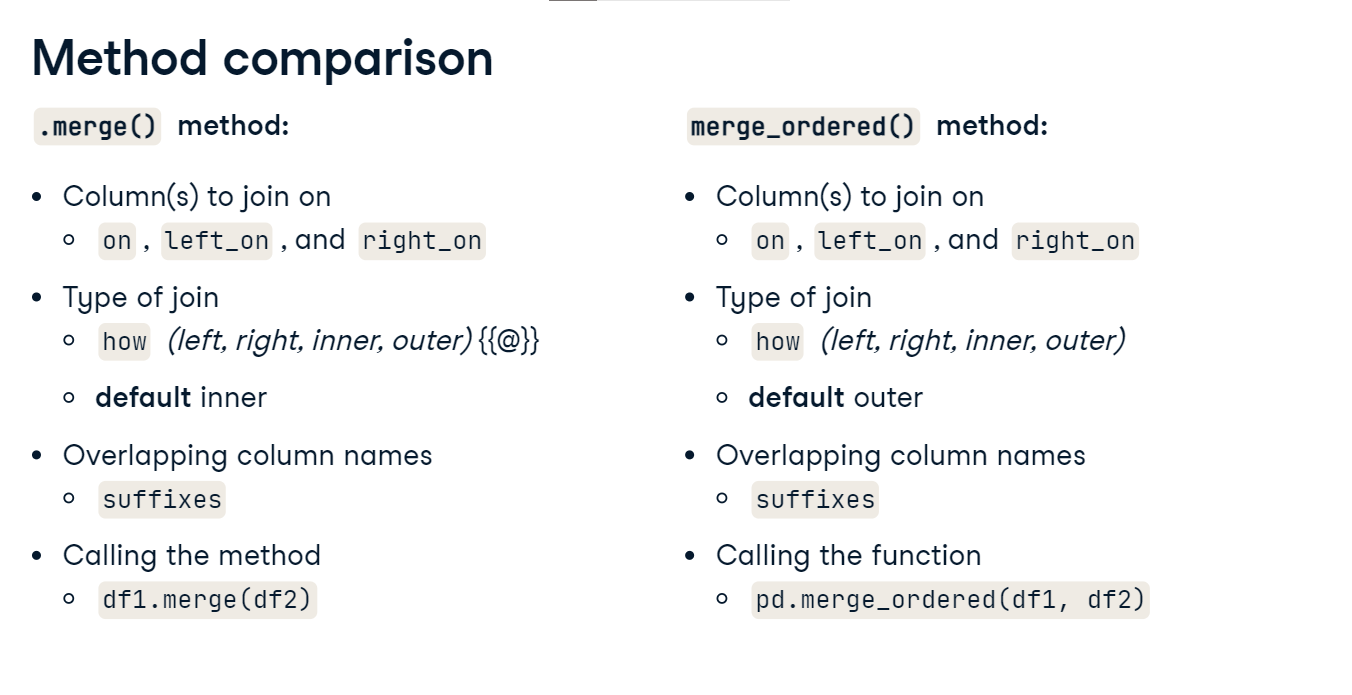
**merge\_ordered()**

The merge\_ordered method will allow us to merge the left and right tables shown here. We can see the output of the merge when we merge on the "C" column. The results are similar to the standard merge method with an outer join, but here that the results are sorted. The sorted results make this a useful method for ordered or time-series data.



**Method comparison**

So, let's first give context to this method. It has many of the same arguments we have already covered with the merge method. They both contain arguments to allow us to merge two tables on different columns. Both methods support different types of joins. Although, the default for the merge method is "inner", it is "outer" for merge\_order method. Also, both methods support suffixes for overlapping column names. However, how you call each of the methods is different. Earlier in the course, we called the merge method by first listing a table and calling the method afterward. For merge\_ordered(), you'll need to first call pandas then merge\_ordered(). Let's look at an example.



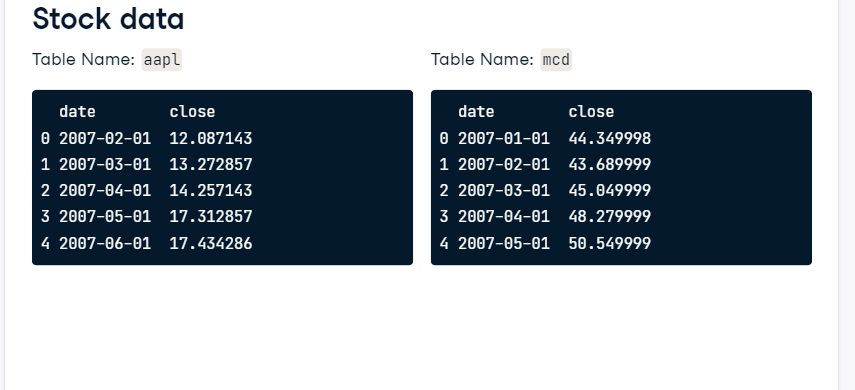
**Financial dataset**

In this chapter, we will be working with financial, macroeconomic, and stock market data.

1. 1 Photo by Markus Spiske on Unsplash

**Stock data**

In this example, we have a table of the stock prices of the Apple corporation from February to June 2007. We also have a table of the stock price for McDonald's corporation from January to May 2007, and we want to merge them.



**Merging stock data**

The first two arguments are the left and right tables. We set the "on" argument equal to date. Finally, we set the suffixes argument to determine which table the data originated. This results in a table sorted by date. There isn't a value for Apple in January or a value for McDonald's for June since values for these time periods are not available in the two original tables.



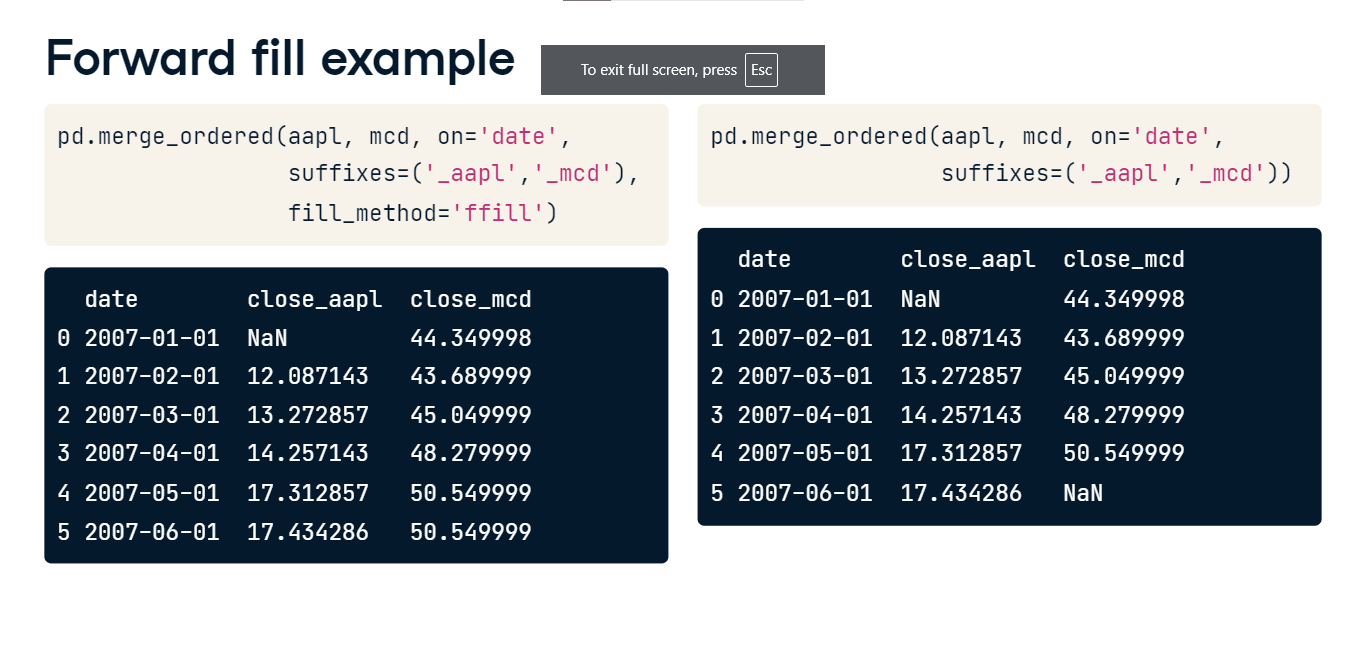
**Forward fill**

We can fill in this missing data using a technique called forward filling. It will interpolate missing data by filling the missing values with the previous value. In the table shown here, the second and fourth rows of column B are filled with the values of B in the rows proceeding them.



**Forward fill example**

Going back to our stock example from before, we now set the fill\_method argument to "ffill" for forward fill. In the result, notice that the missing value for McDonald's in the last row is now filled in with the row before it. The table from before is shown on the right for easier comparison. Notice the missing value for Apple in the first row is still missing since there isn't a row before the first row to copy into the missing value for Apple.



**When to use merge\_ordered()?**

You might think about using the merge\_ordered method instead of the regular merge method when you are working with order or time-series data like in our example. Additionally, the fill forward feature is useful for handling missing data, as most machine learning algorithms require that there are no missing values.

**Using merge\_asof()**

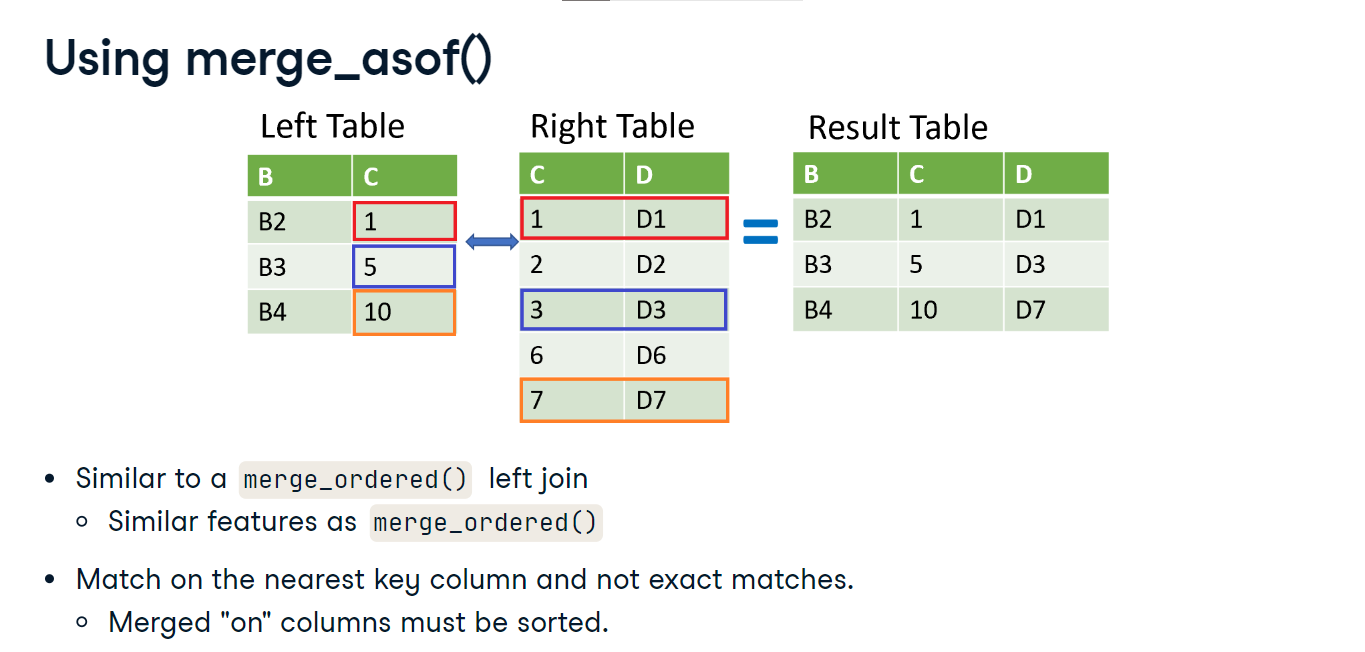
In this lesson we will talk about another method for ordered or time-series data called merge\_asof().

**Using merge\_asof()**

The merge\_asof() method is similar to an ordered left join. It has similar features as merge\_ordered(). However, unlike an ordered left join, merge\_asof() will match on the nearest value columns rather than equal values. This brings up an important point - whatever columns you merge on must be sorted. In the table shown here, when we merge on column "C", we bring back all of the rows from the left table.

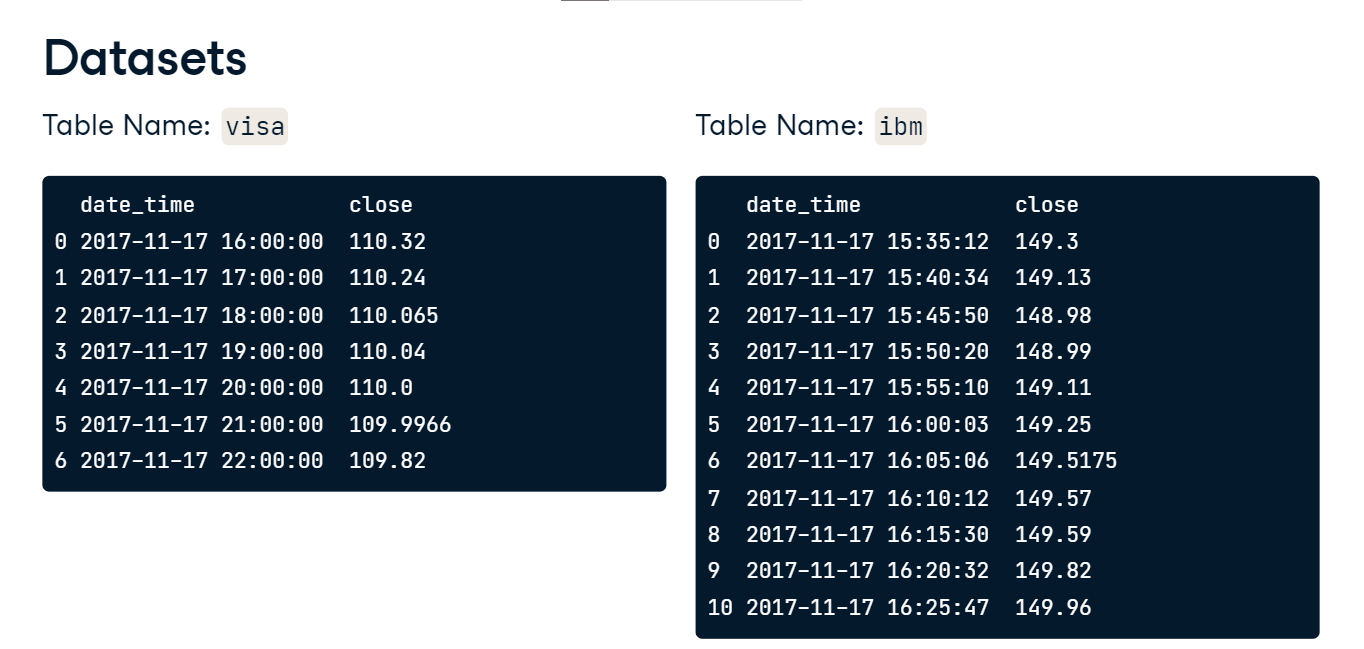
**Using merge\_asof()**

However, the row selected from the right table is the last row whose "C" value is less than or equal to the "C" value in the left table. So, for example, the second row in the left table is matched with the third row in the right table. This because 3 is the closest value in the right table that is still less than or equal to 5.



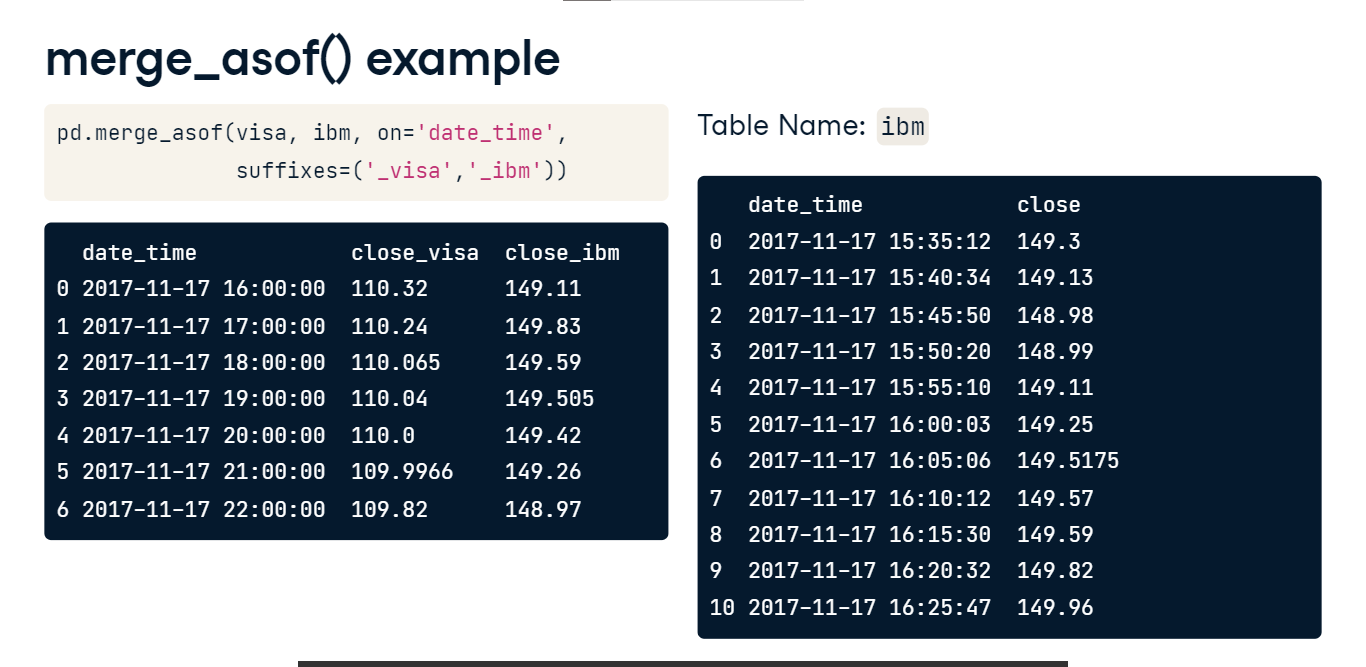
**Datasets**

For this example, we will look at merging two tables. The first is stock price data for the Visa company with entries for every hour on Nov, 11, 2017. The second table is IBM stock prices on the same day with entries for roughly every five minutes.



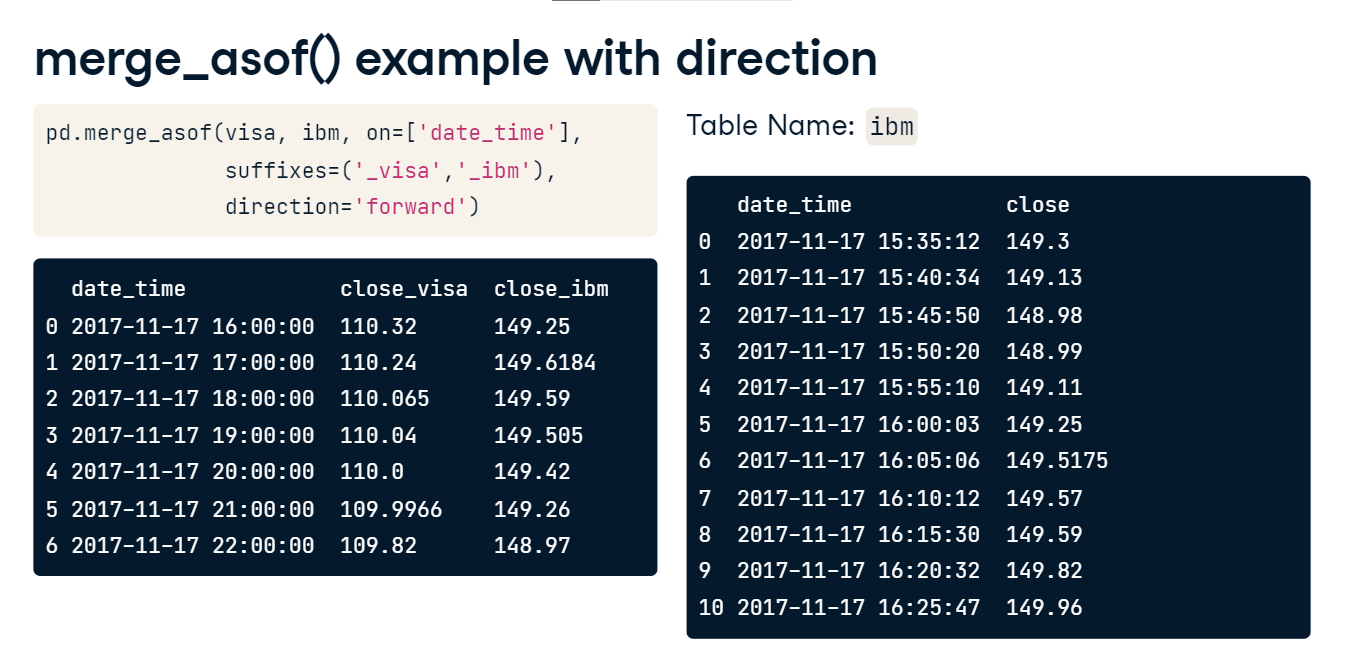
**merge\_asof() example**

Let's use merge\_asof() to merge the tables. The input arguments are very similar to what we have already seen in the course. Here we list the left and right tables first. Then we define that we want to merge on the "date\_time" column. Finally, we provide a set of suffixes. Our output is similar to a left join, so we see all of the rows from the left Visa table. However, the values from the IBM table are based on how close the date\_time values match with the Visa table. Notice the first row and the IBM price of 149.11. Let's show the IBM table again and see why this value was chosen in the merger. It comes from the row indexed as 4. This row has the closest date\_time that is less than the date\_time in the Visa table. The next row has a date\_time that is slightly greater. We will adjust this behavior in our coming example.



**merge\_asof() example with direction**

This time in our merge\_asof() method, we list the direction argument as "forward". This will change the behavior of the method to select the first row in the right table whose "on" key column is greater than or equal to the left's key column. The default value for the direction argument is "backward". When we look at our results, we see different values for the IBM column. Let's again look at the first IBM value and trace it back to the IBM table. We see it in the row indexed as 5. Its date\_time is slightly greater than the date\_time in the visa table. Finally, you can set the direction argument to "nearest" which returns the nearest row in the right table regardless if it is forward or backwards.



**When to use merge\_asof()**

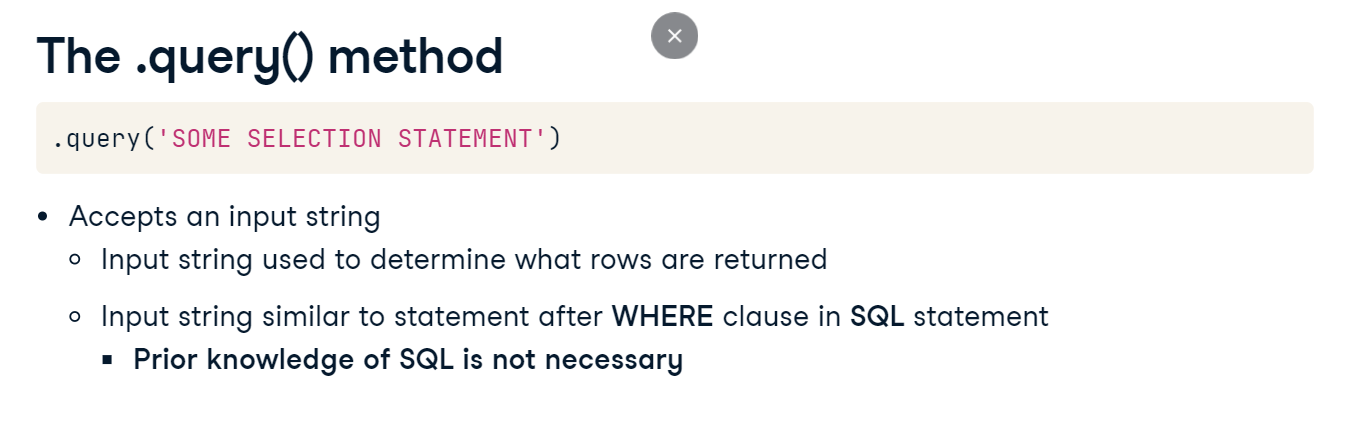
Now that we reviewed the merge\_asof() method, here are a couple of thoughts on when you might want to use it. First, you might think of this method when you are working with data sampled from a process and the dates or times may not exactly align. This is similar to what we did in our example. It could also be used when you are working on a time-series training set, where you do not want any events from the future to be visible before that point in time.

**Selecting data with .query()**

Now that you have learned quite a bit about combining data from different data sources, let's review a pandas method for selecting data from the table called the query() method. pandas provides many methods for selecting data, and query() is one of them.

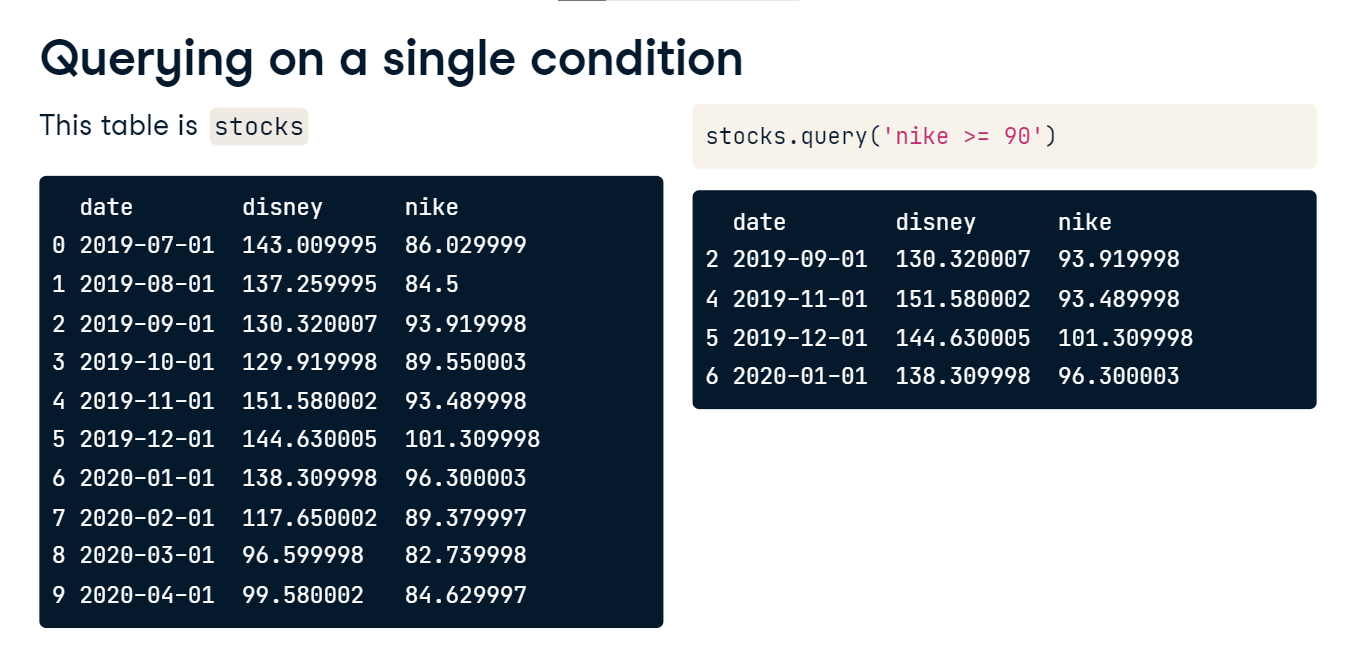
**The .query() method**

The query() method accepts an input string that it will use to select rows to return from the table. For those familiar with SQL, the string you provide to the query function is similar to the portion after the WHERE clause of a SQL statement. However, don't let the SQL statement scare you, because prior knowledge of SQL isn't required. Let's look at an example.



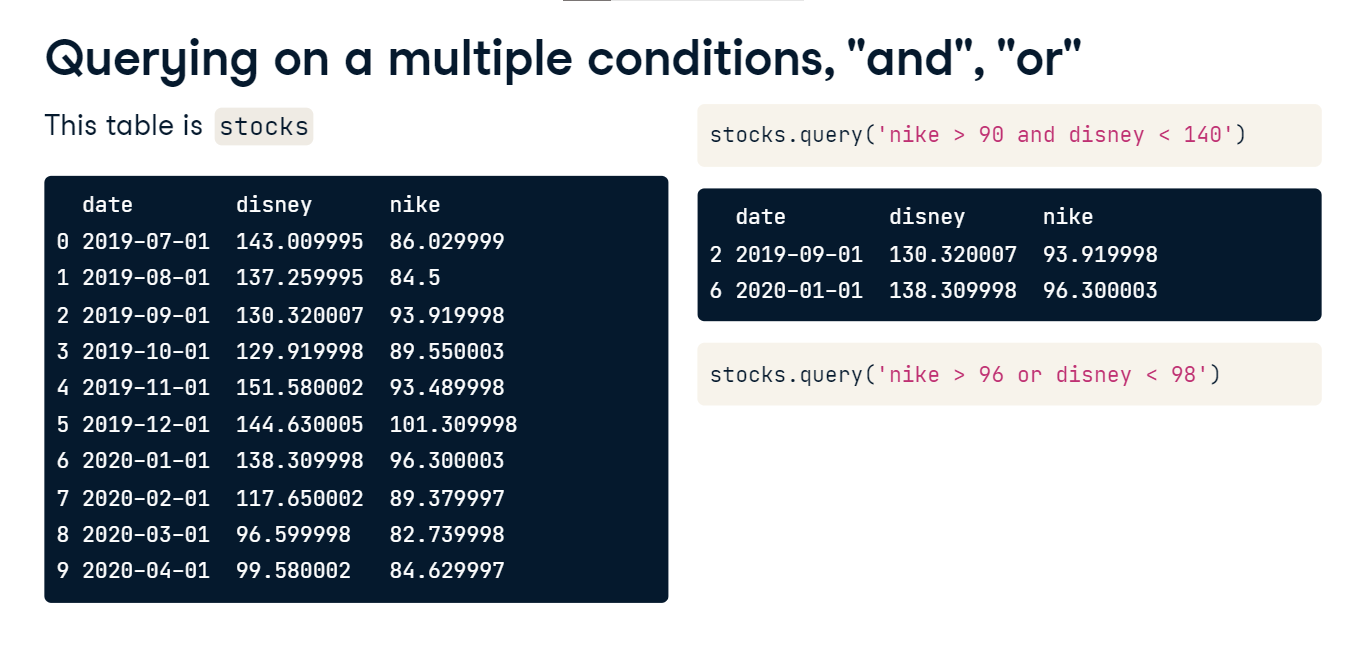
**Querying on a single condition**

We have the following table named stocks with the stock price of Disney and Nike on different days. Now imagine we would like to select the rows where Nike is equal to or above 90. Here we provide a string to the query method. The string identifies that we want to condition which rows are returned by the value of the Nike column. We simply input "nike >= 90". The method returns all rows in stocks where Nike is greater than or equal to 90.



**Querying on a multiple conditions, "and", "or"**

Let's look at another example. Here we use the "and" keyword to select rows where Nike is greater than 90 and Disney is less than 140. The method returns two rows of data that match our criteria. Next, instead of using "and" we can also use the "or" keyword. This input string should select all rows where Nike is over 96 or Disney is less than 98. Now the function returns three rows that meet our criteria.



**Updated dataset**

Our next example shows that you can use the query method to select strings. Imagine now that we have an updated dataset, which is the stocks table in a slightly different format.

**Using .query() to select text**

We are interested in selecting all of the rows were the column stock equals "disney" or the column stock equals "nike" and close is less than 90. Let's pause here for a moment to look at our query string. Within the parentheses of our string, we check if the stock column is nike and the close column is less than 90. Both of these conditions have to be true for the parentheses section to return true. We then add that to the condition to check if stock is listed as "disney". When checking text, we use the double equal signs, similar to an if statement in Python. Also, when checking a text string, we used double quotes to surround the word. This is to avoid unintentionally ending our string statement since we used single quotes to start the statement. In our results, we see all of our Disney rows returned. Also, those rows were Nike is the stock name and the close price is less than 90 are returned.



**Reshaping data with .melt()**

In our last lesson of the course, let's talk about the melt method. This method will unpivot a table from wide to long format. This is often a much more computer-friendly format, therefore making this a valuable method to know.

**Wide versus long data**

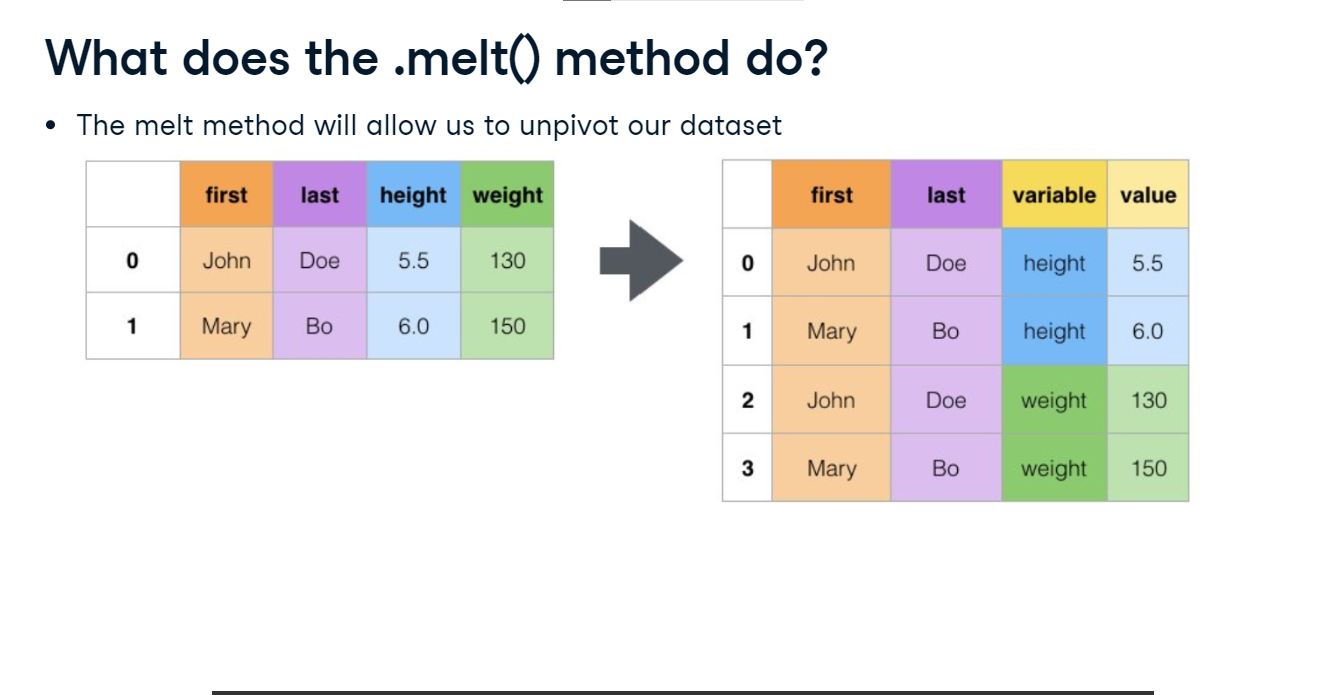
Sometimes we will come across data where every row relates to one subject, and each column has different information about an attribute of that subject. Data formatted in this way is often called wide. There are other times when the information about one subject is found over many rows, and each row has one attribute about that subject. Data formatted in this way is often called long or tall. In general, wide formatted data is easier to read by people than long formatted. However, long formatted data is often more accessible for computers to work with.

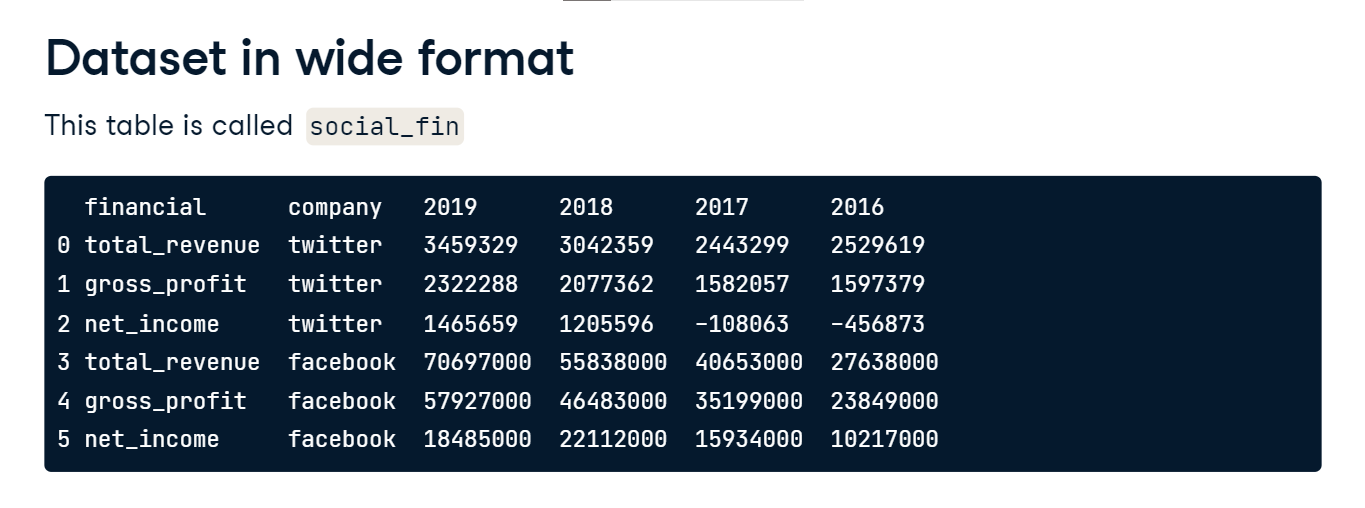
**What does the .melt() method do?**

The melt method will allow us to unpivot, or change the format of, our dataset. In this image, we change the height and weight columns from their wide horizontal placement to a long vertical placement.

**Dataset in wide format**

To demonstrate the melt method, let's start with this dataset of financial metrics of two popular social media companies. Notice that the years are horizontal. Let's change them so that they are vertically placed.



****

**Example of .melt()**

Here we call the melt() method on the table social\_fin. The first input argument to the method is id\_vars. These are columns to be used as identifier variables. We can also think of them as columns in our original dataset that we do not want to change. In our output, we print the first ten rows. Our years are listed vertically. Our final column now has all of our values in one column versus multiple columns. Again, this is a much more computer-friendly format than our original table. We unpivoted each of the separate columns 2016 through 2019. Our output has data for every year in our starting table, but again, we are only showing the first couple of rows. In the next example, we will look at how to control what columns are unpivoted.



**Melting with value\_vars**

This time, let's use the argument value\_vars with the melt() method. This argument will allow us to control which columns are unpivoted. Here, we unpivot only the 2018 and 2017 columns. Our output now only has data for the years 2018 and 2017. Additionally, the order of the value\_var was kept. The output starts with 2018, then moves to 2017. Finally, notice that the column with the years is now named variable, and our values column is named value. We will adjust that in our next example.



**Melting with column names**

In this example, we have added some additional inputs to our melt() method. The var\_name argument will allow us to set the name of the year column in the output. Similarly, the value\_name argument will allow us to set the name of the value column in the output. We again print the first few rows of the output. It is the same as before, except our variable and value columns are renamed year and dollars, respectively. We have seen how the melt() method is useful for reshaping our tables. Imagine a situation where you have merged many columns, making your table very wide. The merge() method can then be used to reshape that table into a more computer-friendly format.



