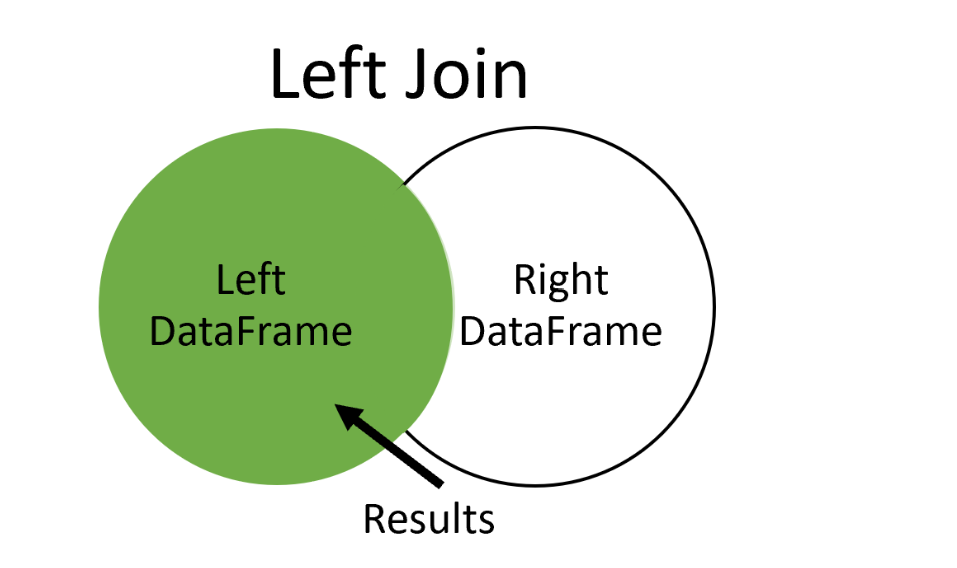
**Left join**

Greetings, and welcome back! In this lesson, we will discuss how a left join works, which is another way to merge two tables. Before we start talking about left joins, let's quickly review what we have learned so far.

**Quick review**

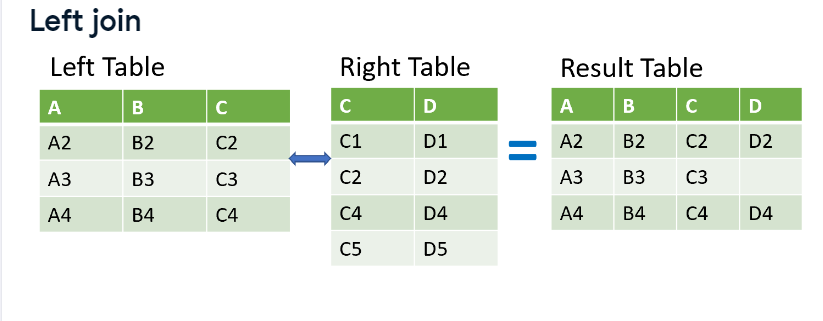
In chapter 1, we introduced the pandas merge method that allows us to combine two tables by specifying one or more key columns to link the tables by. By default, the merge method performs an inner join, returning only the rows of data with matching values in the key columns of both tables.

**Left join**

In this lesson, we'll talk about the idea of a left join. A left join returns all rows of data from the left table and only those rows from the right table where key columns match. 

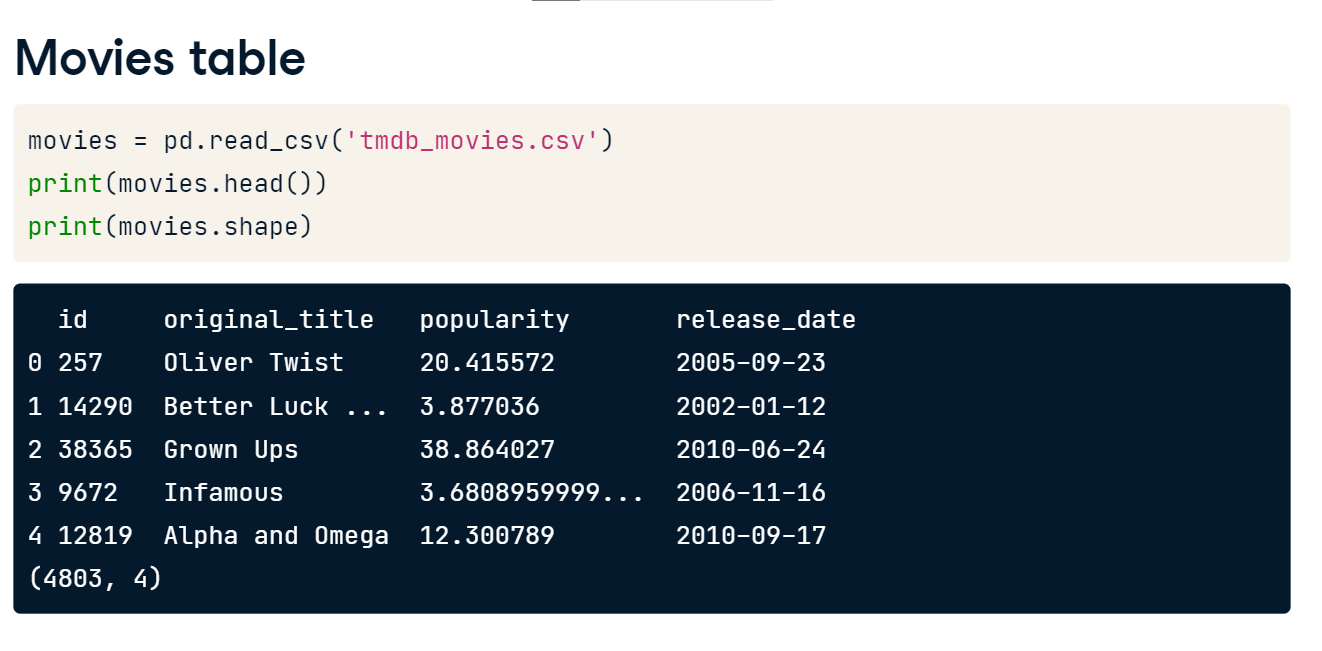
**Left join**

Here we have two tables named left and right. We want to use a left join to merge them on key column C. A left join returns all of the rows from the left table and only those rows from the right table where column C matches in both. Notice the second row of the merged table. The columns from the left table are filled in, while the column from the right table is not since there wasn't a match found for that row in the right table. Let's review another example.



**New dataset**

To help us learn more about left joins and other concepts in this chapter, we will use data from The Movie Database, a community-built movie database with info on thousands of movies, their casts, and popularity. In our next example, we have two tables from The Movie Database that we want to merge.



**Movies table**

Our first table, named movies, holds information about individual movies such as the title name and its popularity. Additionally, each movie is given an ID number. Our table starts with 4,803 rows of data.

**Tagline table**

Our second table is named taglines, which contains a movie ID number and the tag line for the movie. Notice that this table has almost 4,000 rows of data, so it contains fewer movies than the movies table.



**Merge with left join**

To merge these two tables with a left join, we use our merge method similar to what we learned in chapter 1. Here we list the movie table first and merge it to the taglines table on the ID column in both tables. However, notice an additional argument named 'how'. This argument defines how to merge the two tables. In this case, we use 'left' for a left join. The default value for how is 'inner', so we didn't need to specify this in Chapter 1 since we were only working with inner joins. The result of the merge shows a table with all of the rows from the movies table and a value for tag line where the ID column matches in both tables. Wherever there isn't a matching ID in the taglines table, a null value is entered for the tag line. Remember that pandas uses NaN to denote missing data.



**Number of rows returned**

After the merge, our resulting table has 4,805 rows. This is because we are returning all of the rows of data from the movies table, and the relationship between the movies table and taglines is one-to-one. Therefore, in a one-to-one merge like this one, a left join will always return the same number rows as the left table.

**Other joins**

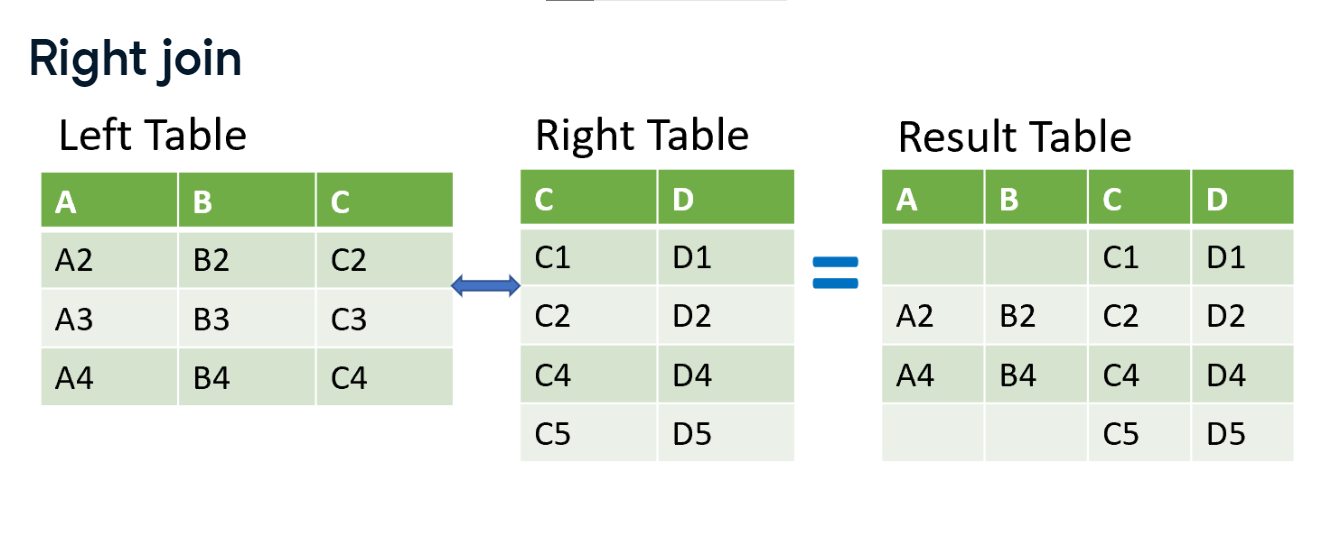
All right, let's continue on. You now know how to use the merge method to perform an inner and left join. The merge method supports two other join types.

**Right join**

Let's start with the right join. It will return all of the rows from the right table and includes only those rows from the left table that have matching values. It is the mirror opposite of the left join.

**Right join**

These example tables show the result of a right join. Only rows from the left table where the column C matches are returned. Where there isn't a match, the columns from the left table will be missing in the result table, like rows one and four.



**Looking at data**

For this lesson, let's look at another table called movie\_to\_genres. Movies can have multiple genres, and this table lists different genres for each movie.



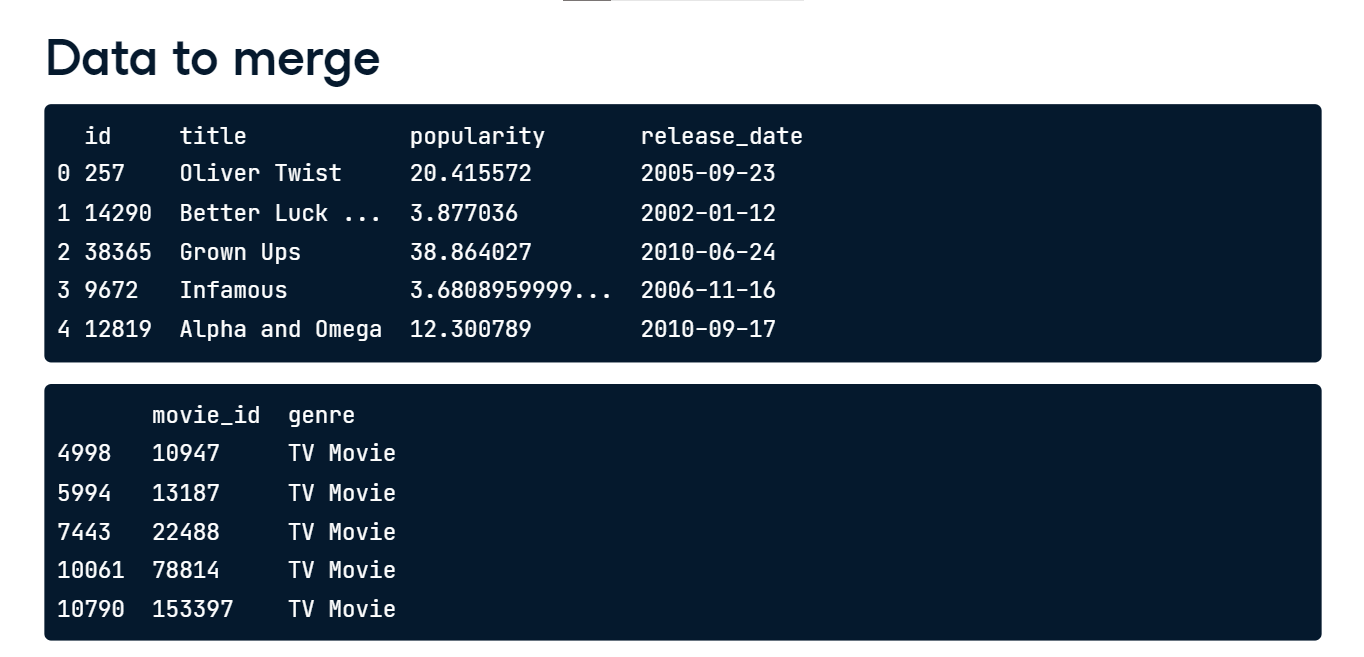
**Filtering the data**

For our right join example, let's take a sample of this data subsetting to develop a table of movies from the TV Movie genre.



**Data to merge**

Our goal is to merge it with the movies table. We will set movies as our left table and merge it with the tv\_genre table. We want to use a right join to check that our movies table is not missing data. In addition to showing a right join, this example also allows us to look at another feature. Notice that the column with the movie ID number in the movies table is named id, and in the tv\_genre table it is named movie\_id. The merge method has a feature to take this into account.

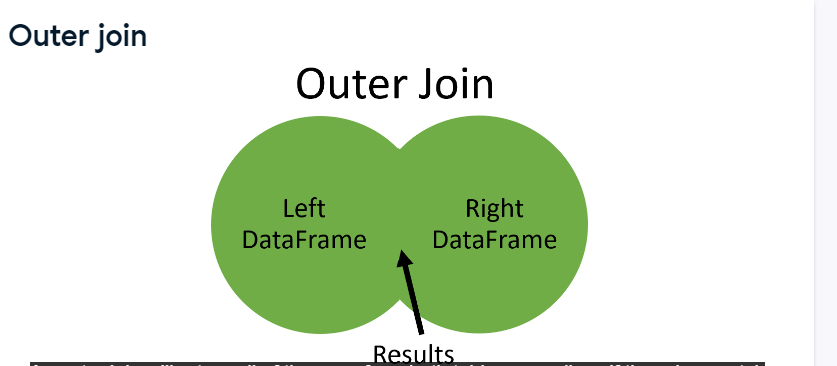


**Merge with right join**

The code for this merge has some new elements. First of all, we set the how argument to right so that the merge performs a right join. Additionally, we introduce two new arguments, named left\_on and right\_on. They allow us to tell the merge which key columns from each table to merge the tables. We list movies as the left table, so we set left\_on to id and right\_on to movie\_id. Our returned table has movies that match our table of tv\_genres. There does not appear to be any null values in the columns from the movies table. We could explore further. However, let's move on to our last type of join. 

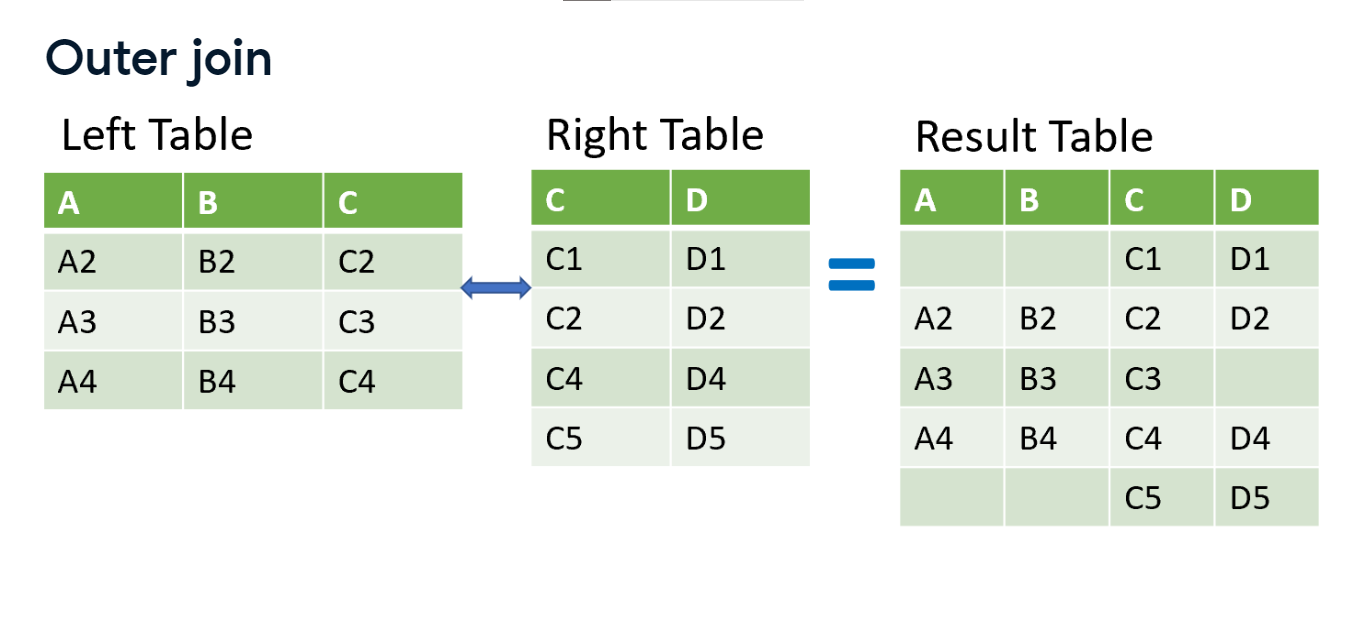
**Outer join**

Our last type of join is called an outer join. An outer join will return all of the rows from both tables regardless if there is a match between the tables.



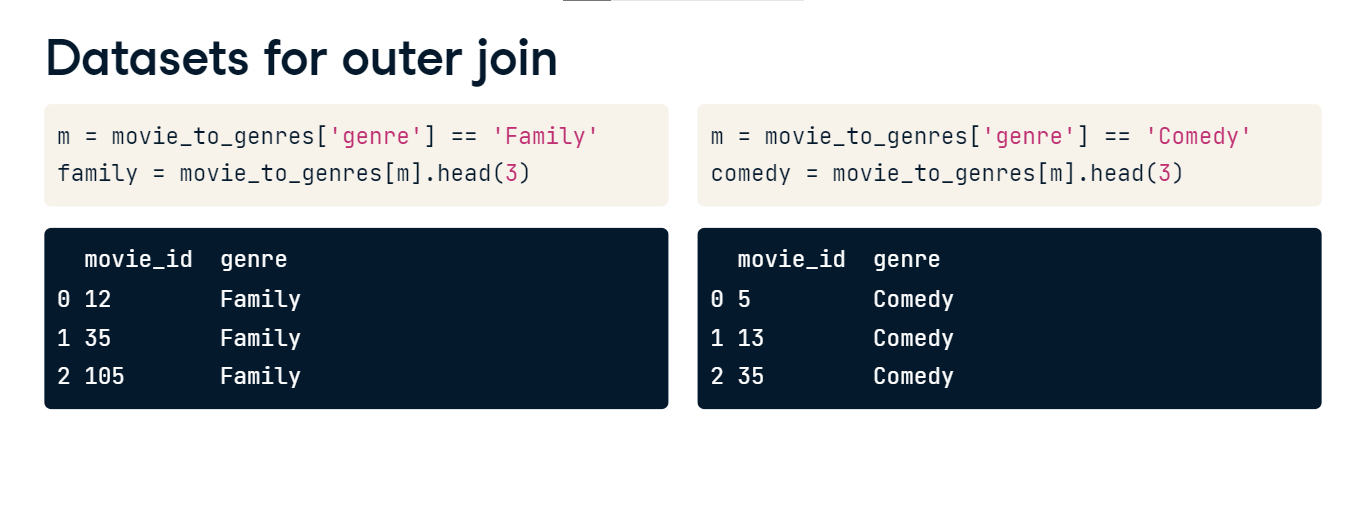
**Outer join**

Here is a simple example of an outer join. Where the key column used to join the tables has no match, null values are returned. That is why in the result, the columns from the left table are missing in rows one and five, and in column D row three is missing.



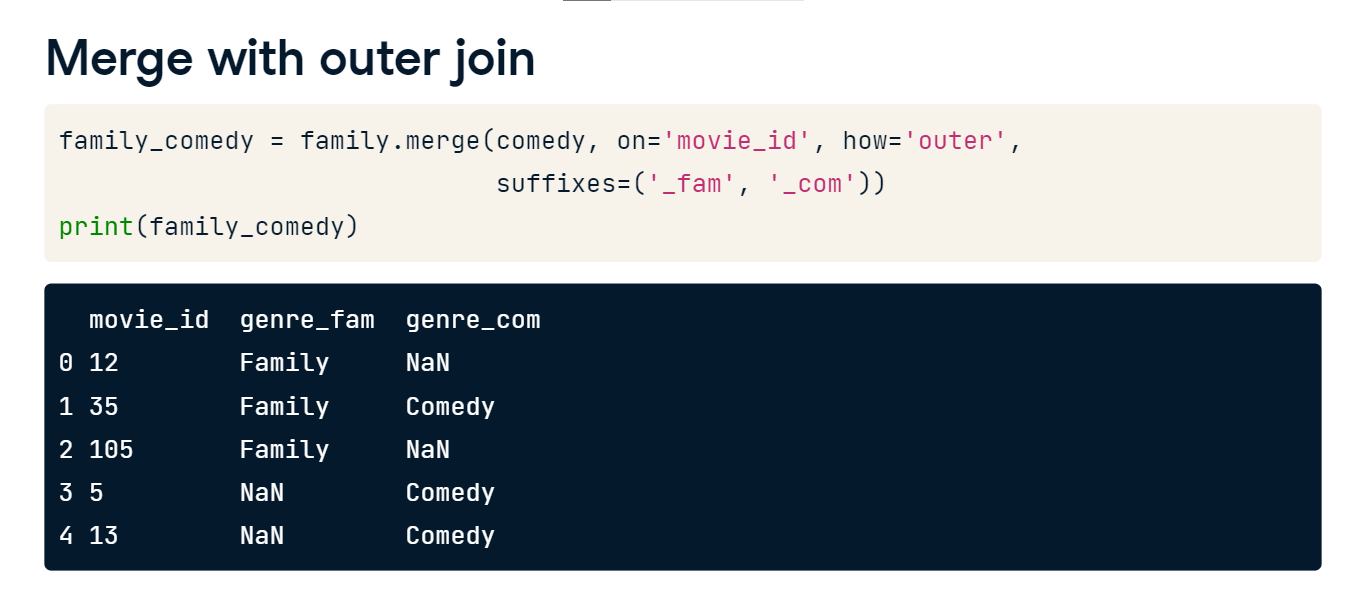
**Datasets for outer join**

For an example of this, we filter the movie\_to\_genres table as before into two very small tables. One table has data on Family movies, and the other has Comedy movies.



**Merge with outer join**

In this merge, we list the family table as the left table and merge it on the movie\_id column. The how argument is set to outer for an outer join. Both of our tables have the same column names. Therefore, we add suffixes to show what table the columns originated. In our result table, every row is returned for both tables and we see some null values. In our original comedy tables ID number 12 does not exist. Therefore a null is shown. Similarly, in our last row, movie ID 13 wasn't in the family dataset so it has a null.



**Merging a table to itself**

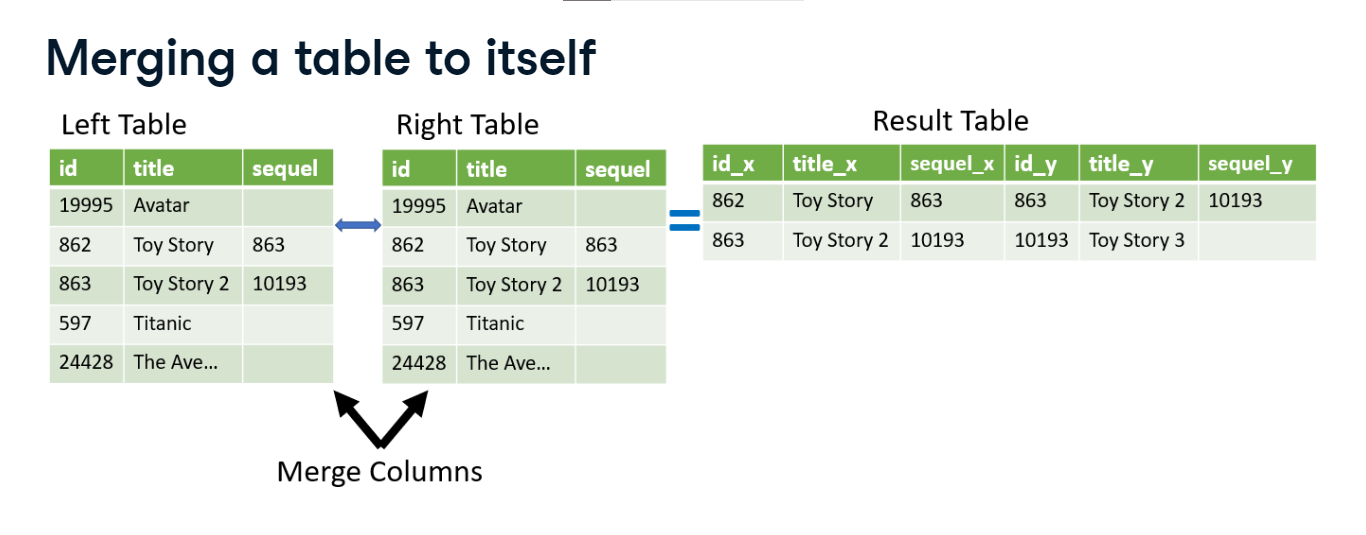
Hello again! In this lesson, we will talk about merging a table to itself. This type of merge is also referred to as a self join. So, let's get started.

**Sequel movie data**

So when would you ever need to merge a table to itself? The table shown here is called sequels and has three columns. It contains a column for movie id, title, and sequel. The sequel number refers to the movie id that is a sequel to the original movie. For example, in the second row the movie is titled Toy Story, and has an id equal to 862. The sequel number of this row is 863. This is the movie id for Toy Story 2, the sequel to Toy Story. If we continue, 10193 is the movie id Toy Story 3 which is the sequel for Toy Story 2.

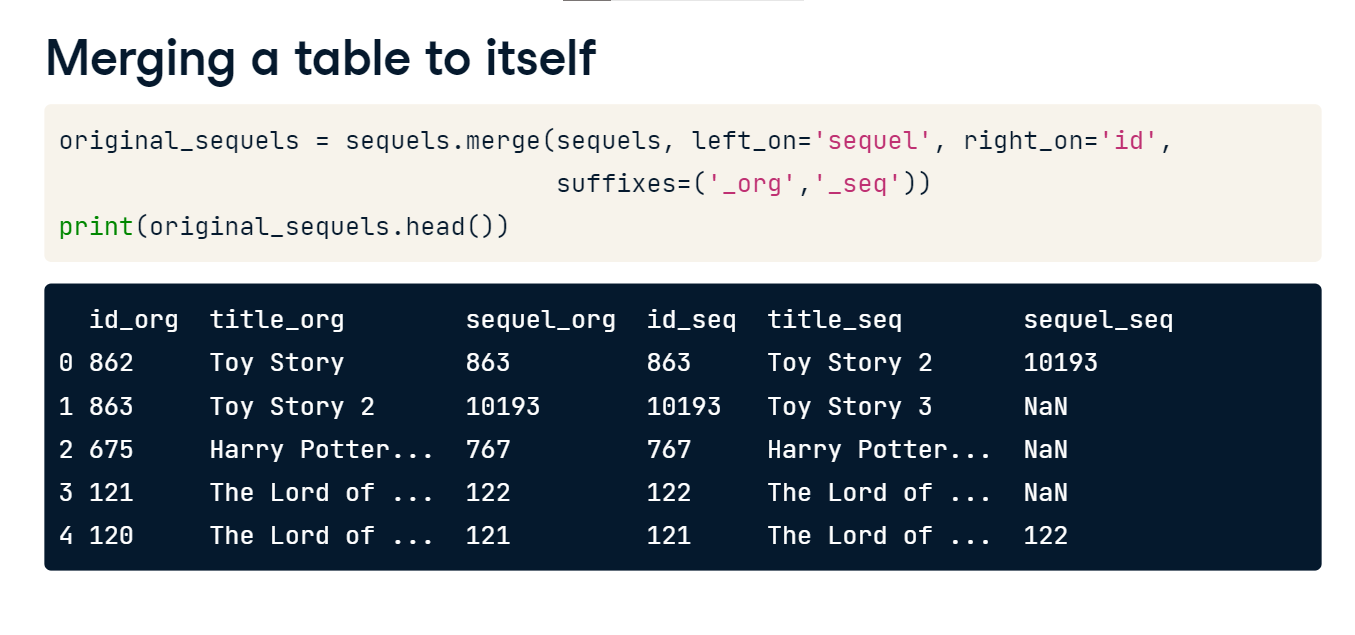
**Merging a table to itself**

If we would like to see a table with the movies and the corresponding sequel movie in one row of the table, we will need to merge the table to itself. In the left table, the sequel ID for Toy Story of 863 is matched with 863 in the ID column of the right table. Similarly, Toy Story 2 of the left table is matched with Toy Story 3 in the right table. We will talk more about this later, but the merge is an inner join. Therefore, we do not see Avatar and Titanic because they do not have sequels.



**Merging a table to itself**

To complete this merge, we set the sequels table as input to the merge method for both the left and right tables. We can think of it as merging two copies of the same table. All of the aspects we have reviewed regarding merging two tables still apply here. Therefore, we can merge the tables on different columns. We'll use the 'left\_on' and 'right\_on' attributes to match rows where the sequel's id matches the original movie's id. Finally, setting the suffixes argument in the merge method allows us to identify which columns describe the original movie and which describe the sequel. When we look at the results of the merge, the 'title\_org' and 'title\_seq' list the original and sequel movies, respectively. Here we listed the original movie and its sequel in one row.



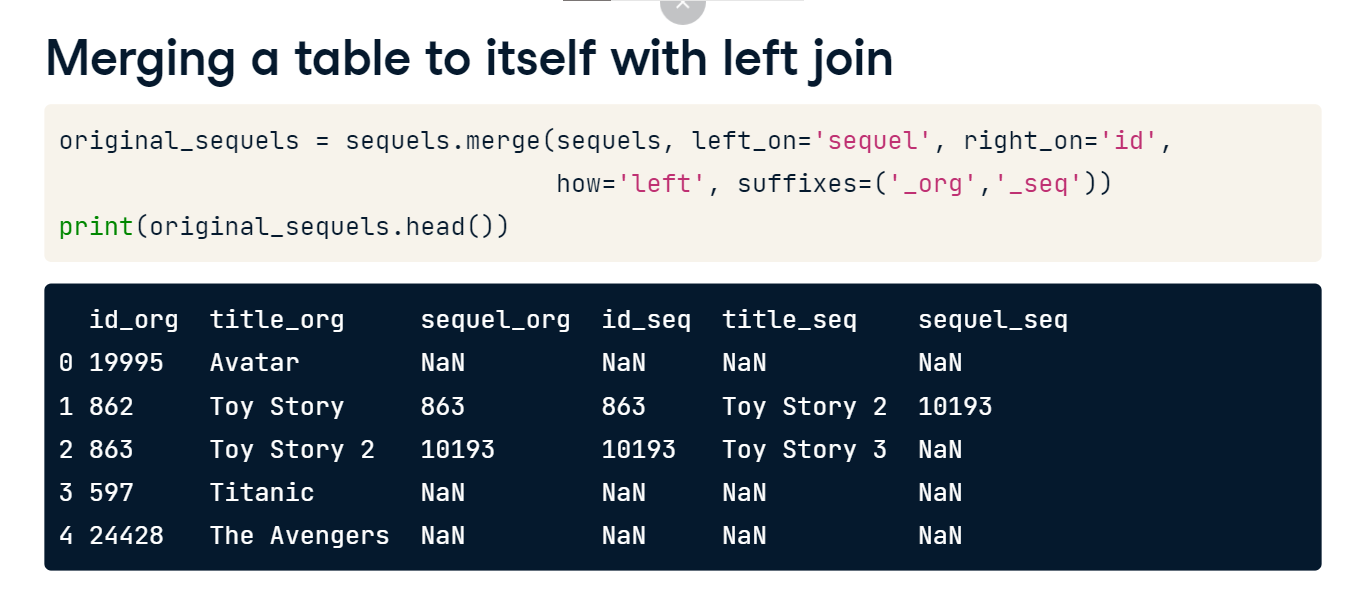
**Continue format results**

Now that we have our result table from the merge, we could select only the `title\_org`, and `title\_seq` columns, and we can see that we've successfully linked each movie to its sequel.



**Merging a table to itself with left join**

Pausing here is a good time to highlight again that when merging a table to itself, we can use the different types of joins we have already reviewed. Let's take the same merge from earlier but make it a left join. The 'how' argument is set in the merge method to left from the default 'inner'. Now the resulting table will show all of our original movie info. If the sequel movie exists in the table, it will fill out the rest of the row. If you compare this to our earlier merger, you now see movies like Avatar and Titanic in the result set.



**When to merge at table to itself**

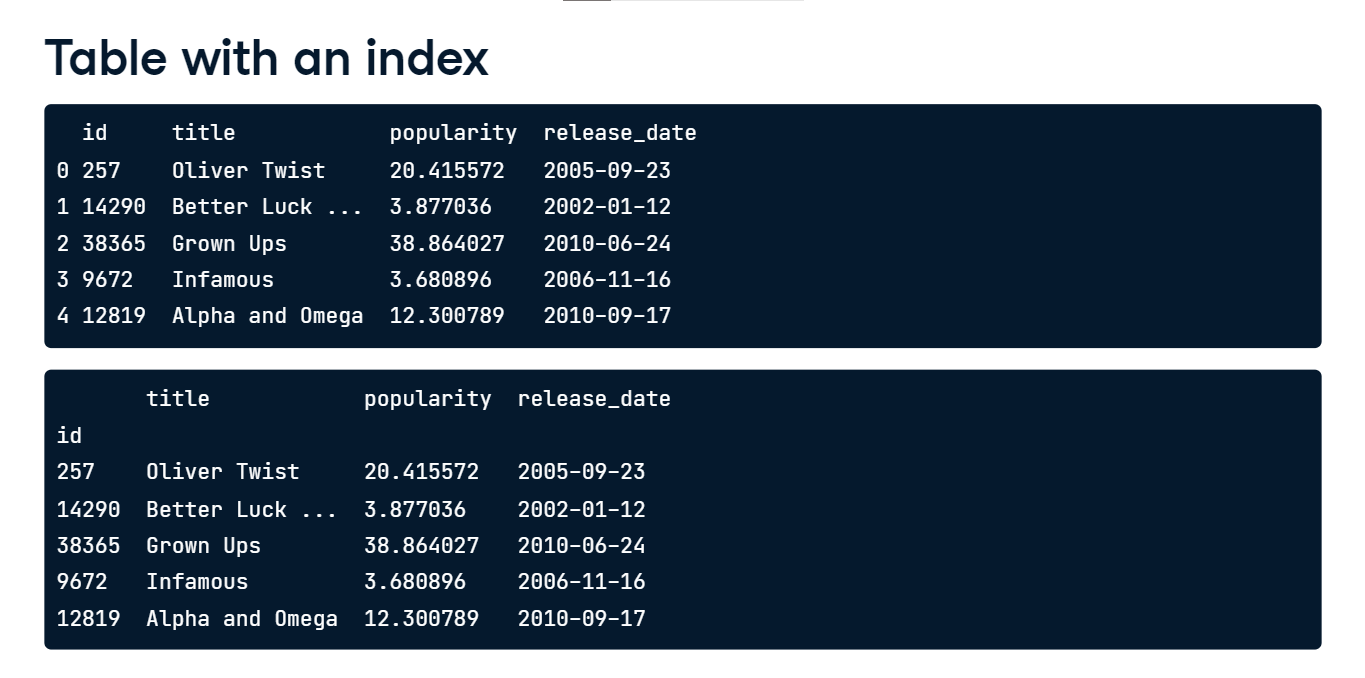
You might need to merge a table to itself when working with tables that have a hierarchical relationship, like employee and manager. You might use this on sequential relationships such as logistic movements. Graph data, such as networks of friends, might also require this technique.

**Merging on indexes**

So far, we've only looked at merging two tables together using their columns. In this lesson, we'll discuss how to merge tables using their indexes. Often, the DataFrame indexes are given a unique id that we can use when merging two tables together.

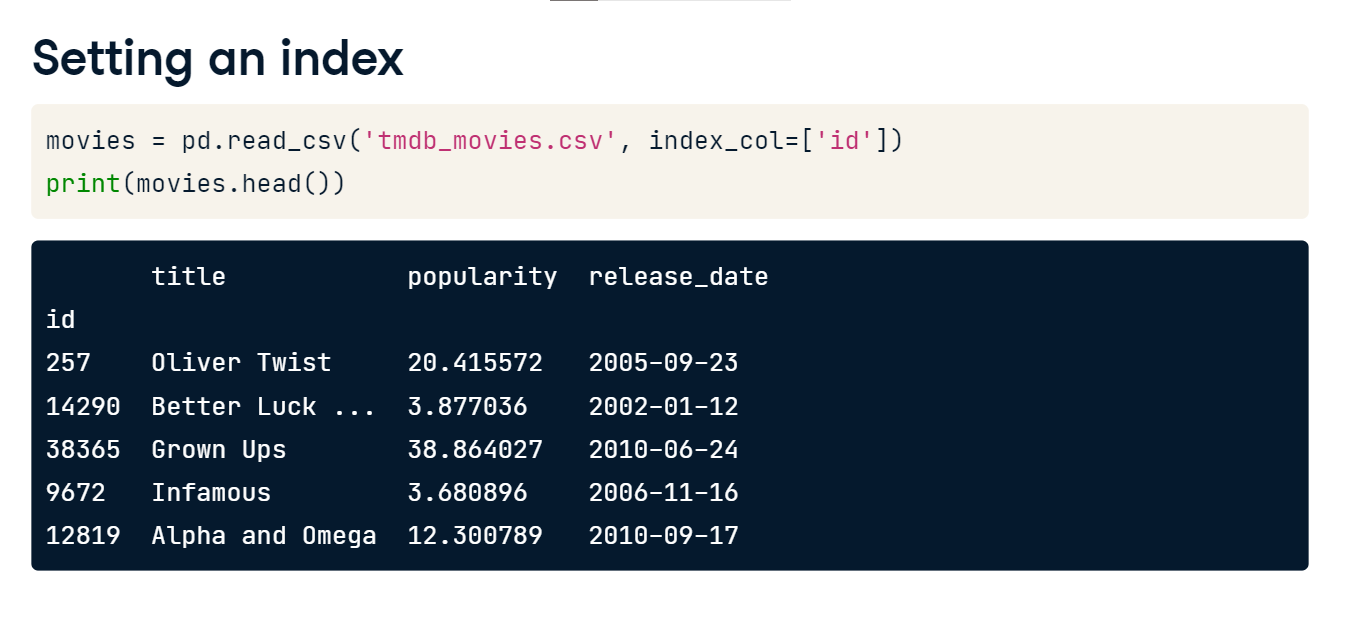
**Table with an index**

Here, we show the movies table that was introduced earlier in this chapter. The index is the default 0, 1, 2, 3, etc., auto-increment. In this second version, the id column is the index for the table.



**Setting an index**

There are different methods to set the index of a table, but if our data starts off in a CSV file, we can use the index\_col argument of the read\_csv method. This lesson will not focus on how to set a table index, but how to use that index to merge two tables together.



**Merge index datasets**

Recall our example to merge the movies and taglines tables using the id column with a left join. Let's recreate that merge using the index which is now the id for tables.



**Merging on index**

Our merge statement looks identical to before. However, in this case we are inputting to the 'on' argument the index level name which is called 'id'. The merge method automatically adjusts to accept index names or column names. The returned table looks as before, except the 'id' is the index.



**MultiIndex datasets**

Let's try a multiIndex merge. Here, we have two tables with a multiIndex that holds the movie ID and cast ID. The first table, named 'samuel', has the movie and cast ID for a group of movies that Samuel L. Jackson acted in. The second table, named cast, has the movie ID and cast ID for a number of movie characters. Let's merge these two tables on their multiIndex.

**MultiIndex merge**

In this merge, we pass in a list of index level names to the 'on' argument, just like we did when merging on multiple columns. Since this is an inner join, both the movie\_id and cast\_id must match in each table to be returned in the result. It's interesting to see that Samuel Jackson has acted in over 65 movies! That's a lot.



**Index merge with left\_on and right\_on**

There is one more thing regarding merging on indexes. If the index level names are different between the two tables that we want to merge, then we can use the left\_on and right\_on arguments of the merge method. Let's go back to our movies table, shown in the top panel, and merge it with our movies\_to\_genres table, shown in the lower panel.



**Index merge with left\_on and right\_on**

In this merge, since we list the movies table as the left table, we set left\_on equal to id and right\_on equal to movie\_id. Additionally, since we are merging on indexes, we need to set left\_index and right\_index to True. These arguments take only True or False. Whenever we are using the left\_on or right\_on arguments with an index, we need to set the respective left\_index and right\_index arguments to True. The left\_index and right\_index tell the merge method to use the separate indexes.

