**Inner join**

Welcome! I am Aaren Stubberfield and I will be your instructor for this course. The pandas package is a powerful tool for manipulating and transforming data in Python. However, when working on an analysis, the data needed could be in multiple tables. This course will focus on the vital skill of merging tables together.

**For clarity**

As we start, two quick clarifications. First, through other courses on DataCamp, you may have learned how to import tabular data as DataFrames. In this course, you may hear the words table and DataFrame, but they are equivalent here. Second, we will refer to combining different tables together as merging tables, but note that some refer to this same process as joining.

1. 1 Photo by David Travis on Unsplash

**Chicago data portal dataset**

To help us learn about merging tables, we will use data from the city of Chicago data portal.

1. 1 Photo by Pedro Lastra on Unsplash

**Datasets for example**

The city of Chicago is divided into fifty local neighborhoods called wards. We have a table with data about the local government offices in each ward. In this example, we want to merge the local government data with census data about the population of each ward.

1. 1 Ward image By Alissapump, Own work, CC BY-SA 3.0

**The ward data**

If we look at the wards table, we have information about the local government of each ward, such as the government office address. This table has 50 rows and 4 columns, or one row for each ward.

**Census data**

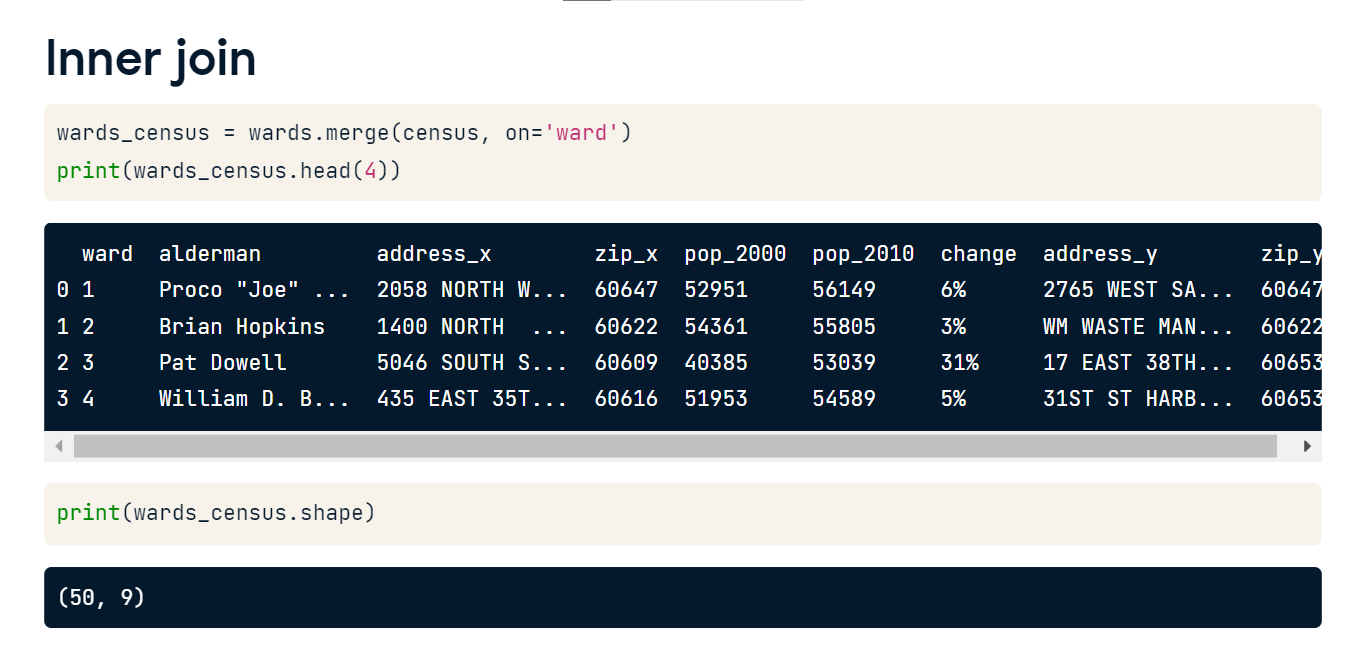
The census table contains the population of each ward in 2000 and 2010, and that change as a percentage. Additionally, it includes the address for the center of each ward. This table has 50 rows and 6 columns.

**Merging tables**

The two tables are related by their ward column. We can merge them together, matching the ward number from each row of the wards table to the ward numbers from the census table. For example, the second ward in the wards table with Alderman Brian Hopkins would be matched with row 2 of the census table where the population in 2000 was 54,361.

**Inner join**

The pandas package has an excellent DataFrame method for performing this type of merge called merge. The merge method takes the first DataFrame, wards, and merges it with the second DataFrame, census. We use the on argument to tell the method that we want to merge the two DataFrames on the ward column. Since we listed the wards table first, its columns will appear first in the output, followed by the columns from the census table. In this example, the merge returns a DataFrame with 50 rows and 9 columns, where the returned rows have matching values for the ward column in both tables. This is called an inner join.



**Inner join**

An inner join will only return rows that have matching values in both tables.

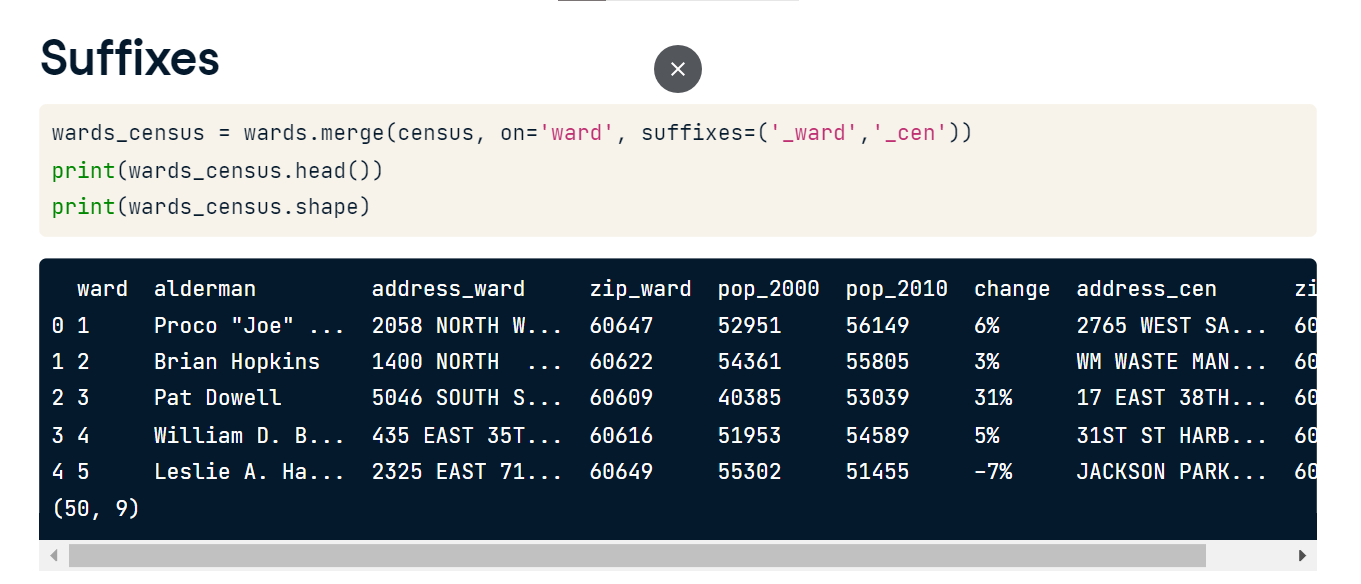
**Suffixes**

You may have noticed that the merged table has columns with suffixes of underscore x or y. This is because both the wards and census tables contained address and zip columns. To avoid multiple columns with the same name, they are automatically given a suffix by the merge method.



**Suffixes**

We can use the suffix argument of the merge method to control this behavior. We provide a tuple where all of the overlapping columns in the left table are given the suffix '\_ward', and those of the right table will be given the suffix '\_cen'. This makes it easier for us to tell the difference between the columns.



**One to many relationships**

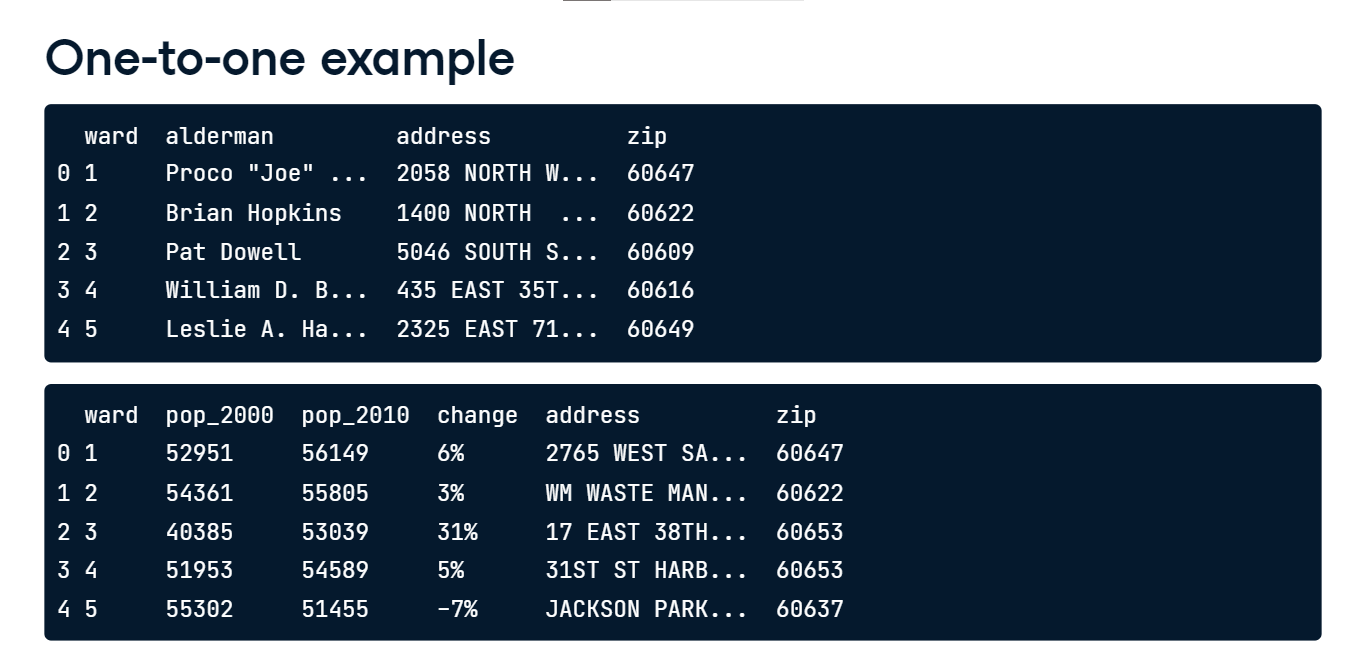
Welcome back! In the last lesson, we learned how to merge two DataFrames together with the merge method. In this lesson, we'll discuss different types of relationships between tables. In particular, we will discuss the one-to-many relationship. But first, let's quickly consider what a one-to-one relationship is.

**One-to-one**

In a one-to-one relationship, every row in the left table is related to one and only one row in the right table.

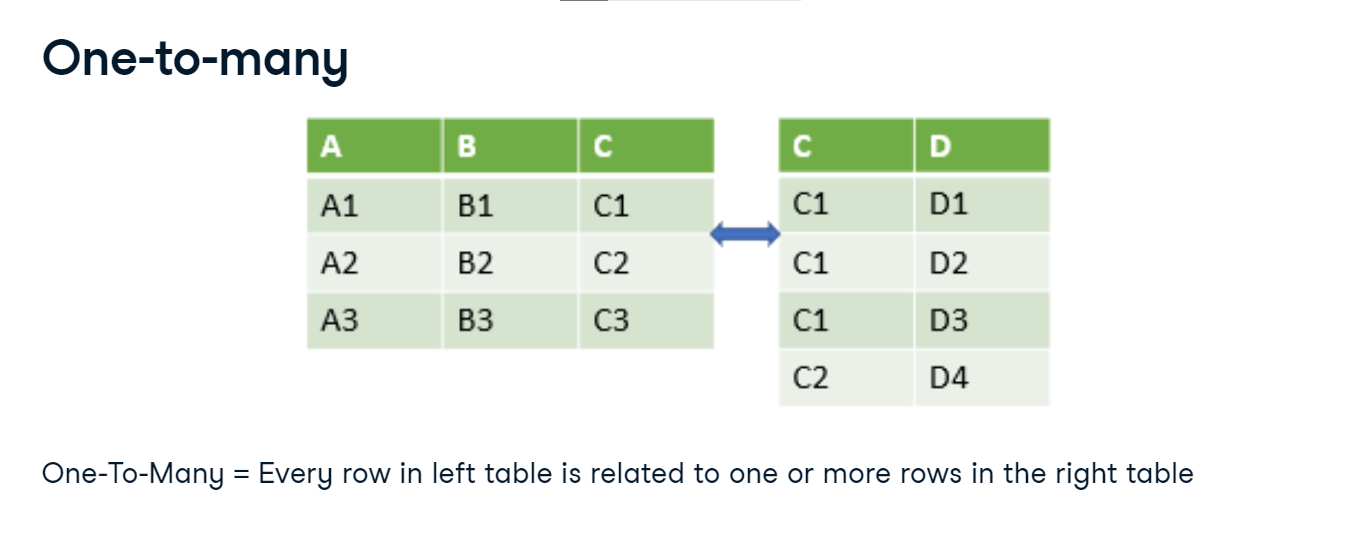
**One-to-one example**

We looked at a one-to-one relationship earlier. Recall the relationship between the wards table and the census table. Every row in the wards table is related to only one row in the census table, so there is only one row for ward 3 in each table. Practically speaking, it only makes sense that there is one row of population information for each ward. It wouldn't make sense if the census table contained multiple population values in 2000 for the third ward.



**One-to-many**

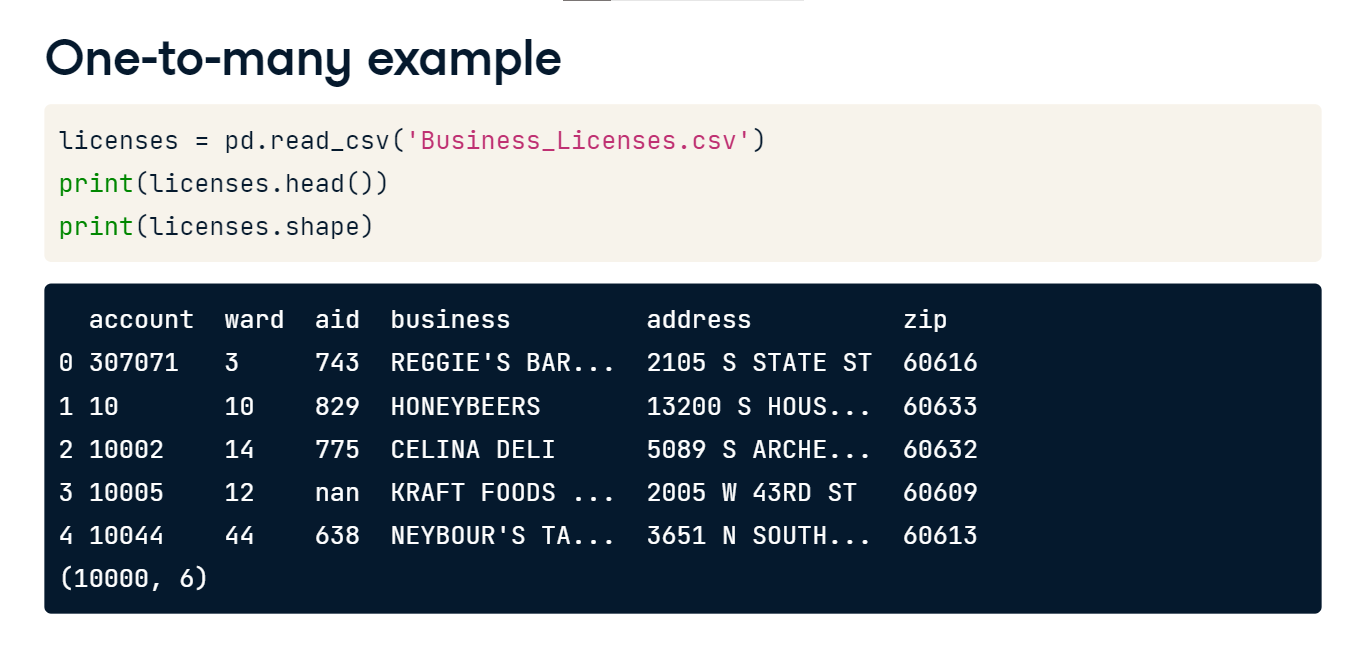
So, what is a one-to-many relationship? Well, in a one-to-many relationship, every row in the left table is related to one or more rows in the right table.



**One-to-many example**

To provide an example of a one-to-many relationship, let's think back to our wards table. Within each ward, there are many businesses. We will merge the wards table with a table of licensed businesses in each ward.

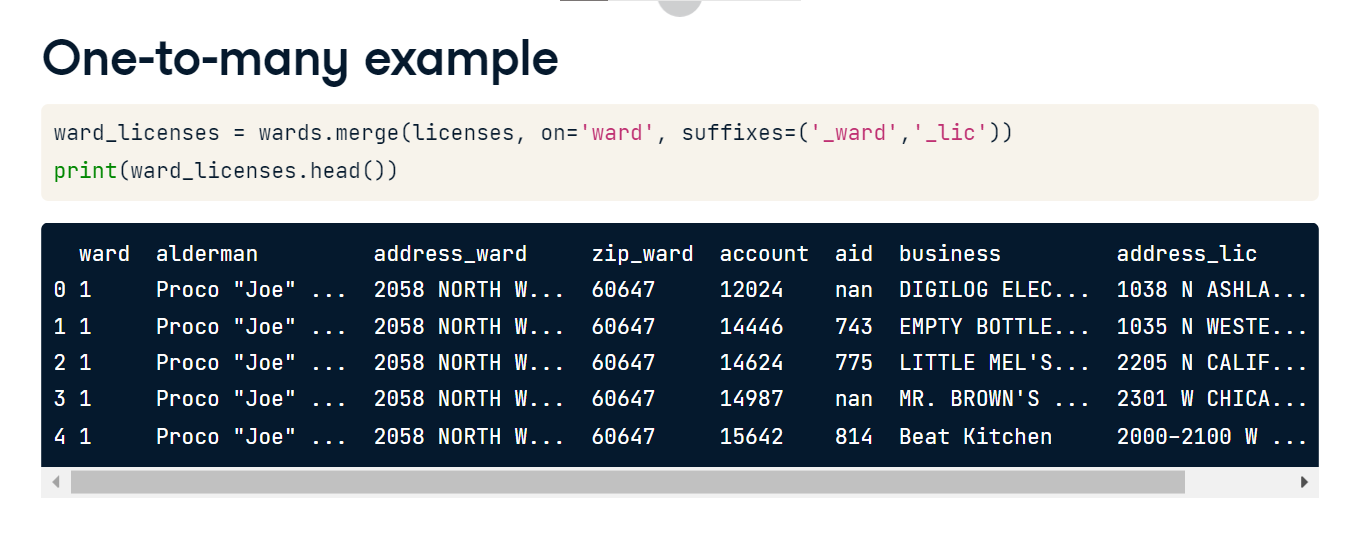
The business license data is stored in another table called licenses. It holds info such as the business address and ward the business is located within.



**One-to-many example**

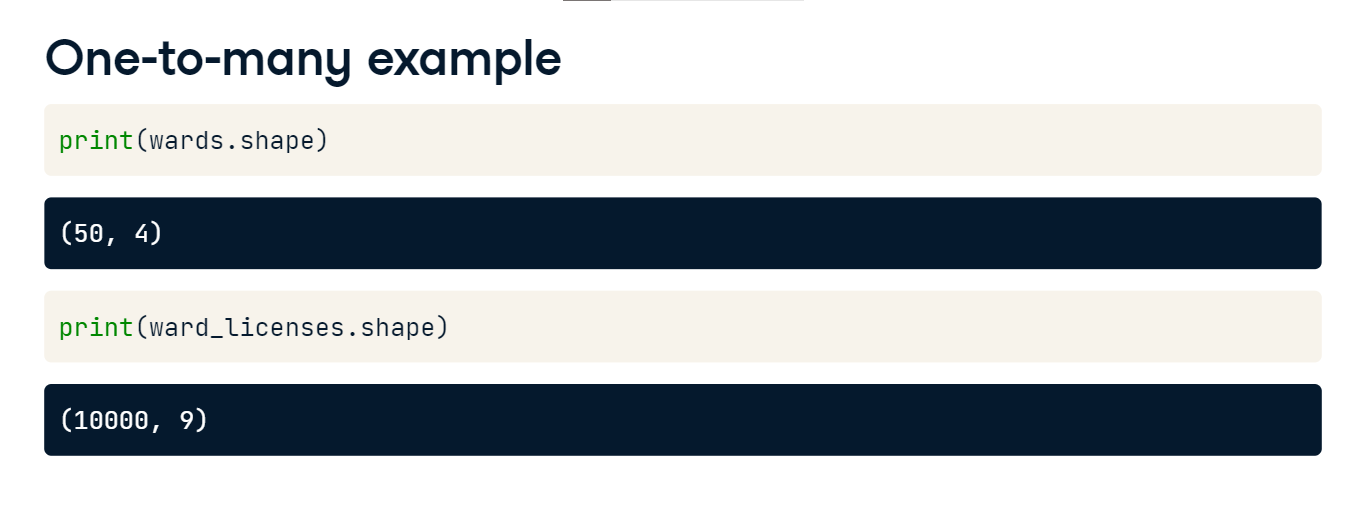
The two DataFrames are related to each other by their ward column.

When we merge the two tables together with the merge method, setting the 'on' attribute to the column ward, the resulting table has both local ward data and business license data. Notice that ward 1 and its alderman Joe is repeated in the resulting table because the licenses table has many businesses in the 1st ward. pandas takes care of the one-to-many relationships for us and doesn't require anything special on our end. We can use the same syntax as we did with one-to-one relationships.



**One-to-many example**

By printing the shape, we can see that our original wards table has 50 rows. After merging the wards table with the licenses table, the resulting table has 10,000 rows. When you merge tables that have a one-to-many relationship, the number of rows returned will likely be different than the number in the left table.

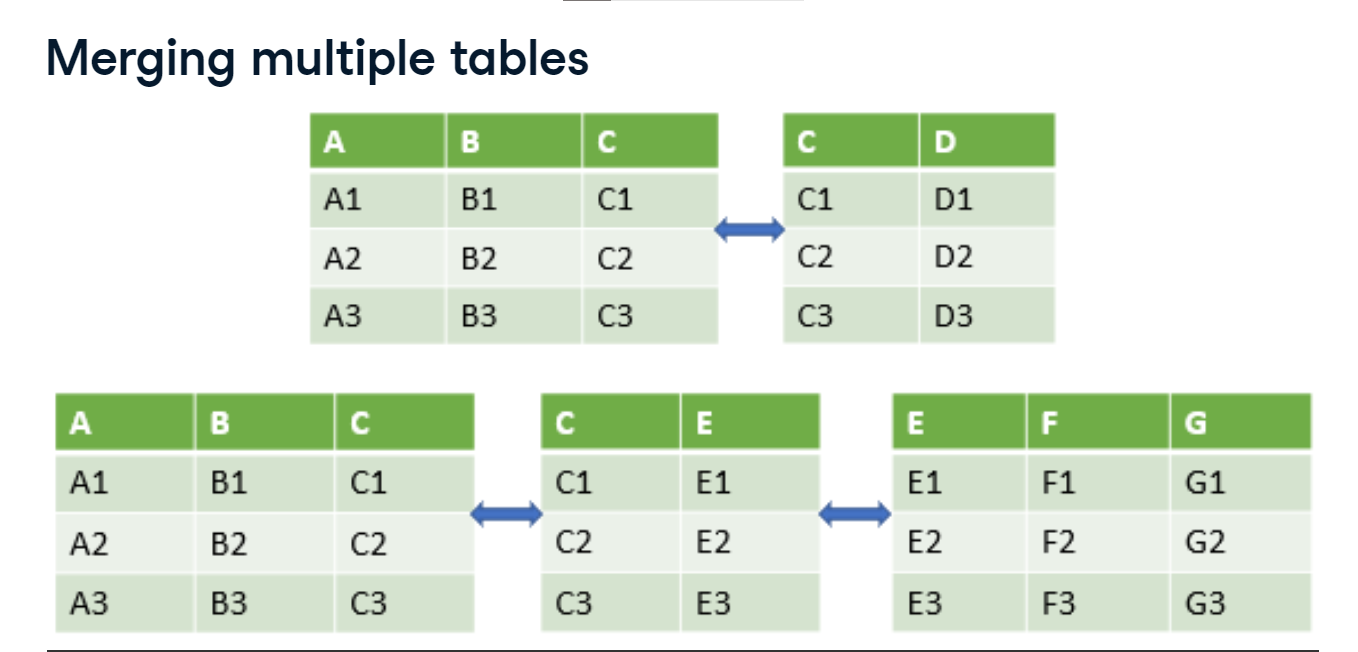


**Merging multiple DataFrames**

Welcome back. In our last lesson, we learned how to merge two tables with a one-to-many relationship using the merge method. Merging data like this is a necessary skill to bring together data from different sources to answer some more complex data questions.

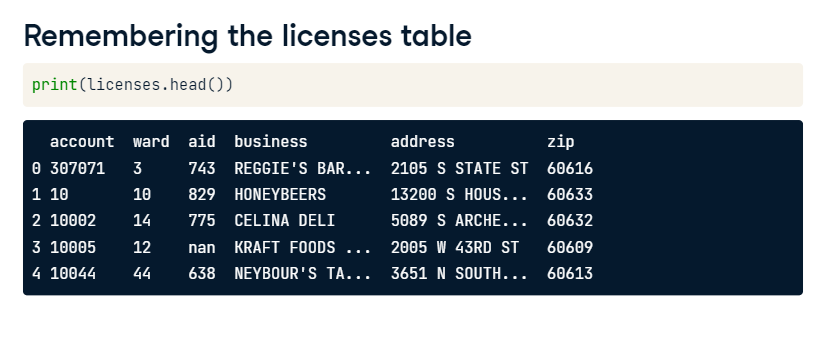
**Merging multiple tables**

Sometimes we need to merge together more than just two tables to complete our analysis.



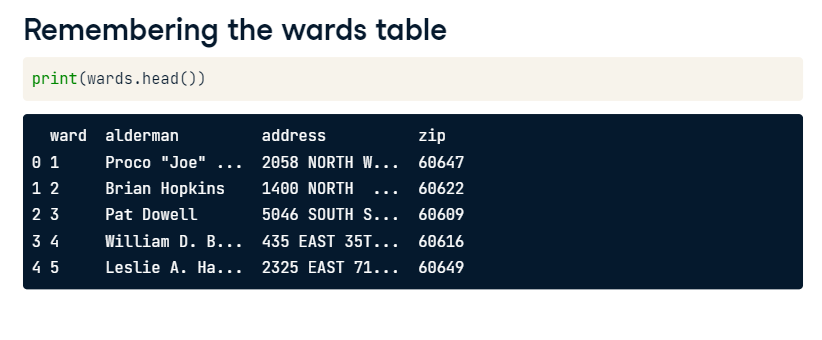
**Remembering the licenses table**

In the previous lesson, we used two tables from the city of Chicago. One table contained business licenses issued by the city.



**Remembering the wards table**

The other table listed info about the local neighborhoods called wards, including the local government official's office.



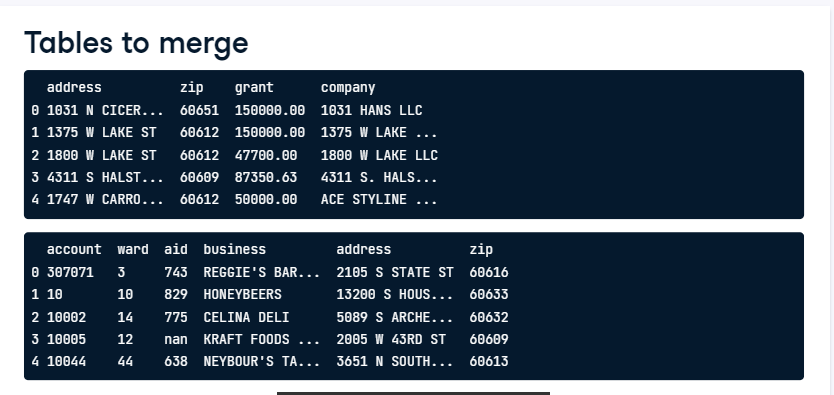
**Review new data**

Now, we also have a table of businesses that have received small business grant money from Chicago. The grants are funded by taxpayer money. Therefore, it would be helpful to analyze how much grant money each business received and in what ward that business is located. We then could determine if one ward's businesses received a disproportionately large amount of grant money.



**Tables to merge**

To pull all of this information together, let's first connect our grants table to our licenses table. The two tables are related by their company name and location. Let's pause here for a moment.



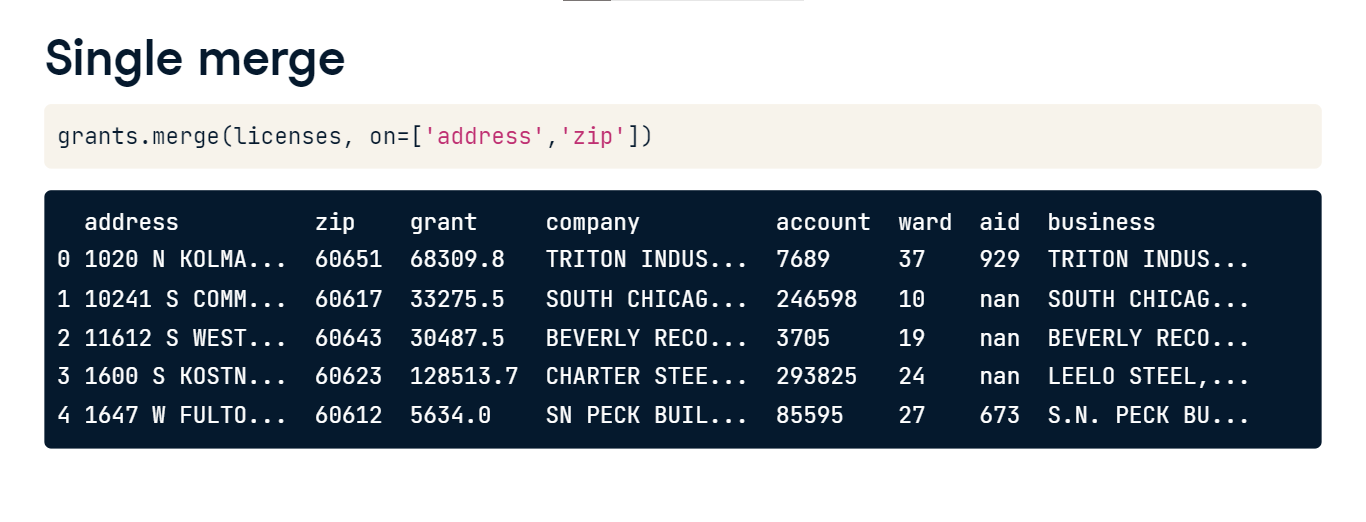
**Theoretical merge**

If we merge the two tables only using the zip column, then the 60616 zip of Reggie's bar from the licenses table will be matched to multiple businesses in the grants table with the same zip. Our code sample prints the first few rows and some columns of the merged table. The output of the merge duplicates Reggie's bar for each matching zip in the grants table, which is not what we want. If instead, we merged on address only, there's a small risk that the address would repeat in different parts of the city. Therefore, the best option is to merge the tables using the combination of both address and zip code.



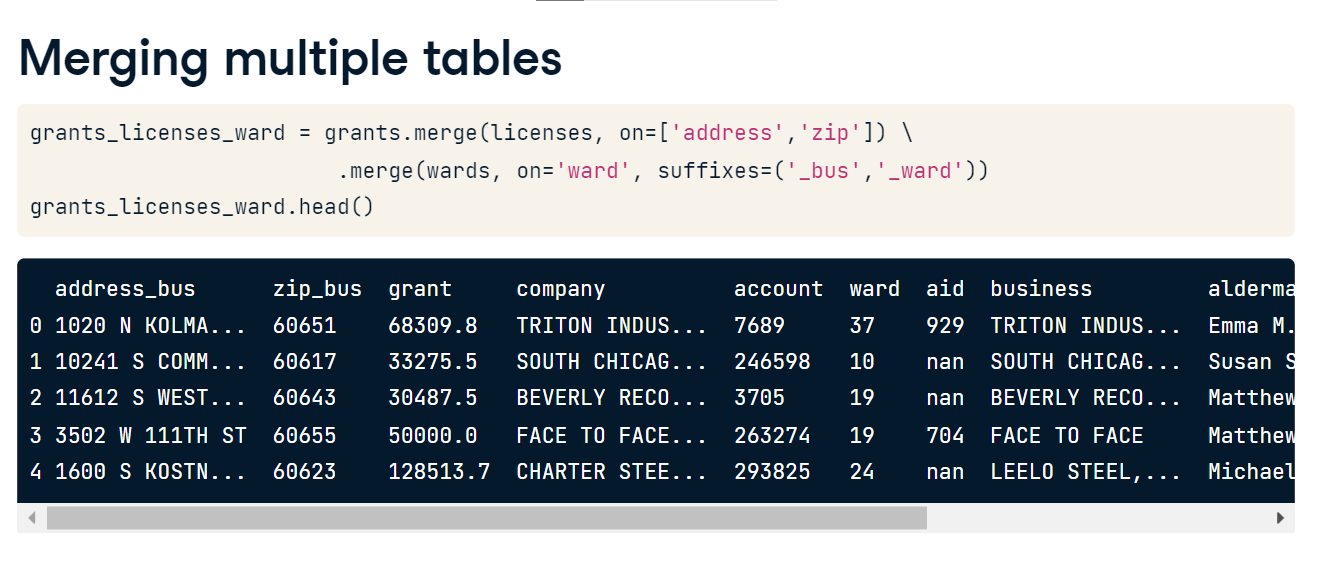
**Single merge**

We merge the two DataFrames as shown before, except in this case, we pass a list of the column names we want to merge on to the 'on' argument. This allows us to use multiple columns in the merge. As before, the matching rows between the two DataFrames are returned with the columns from the grant table listed first. However, when we merge on two columns, in this case address and zip code, we are requiring that both the address and zip code of a row in the left table match the address and zip code of a row in the right table in order for them to be linked to each other in the merge.



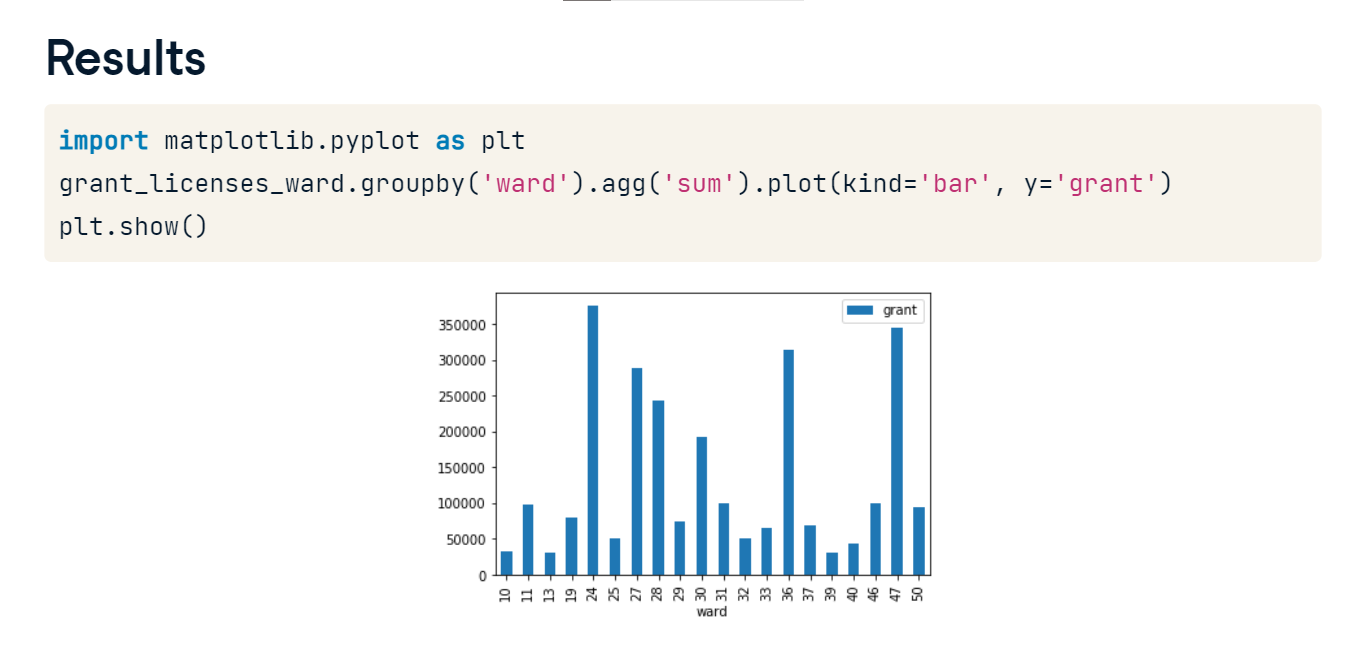
**Merging multiple tables**

We can now extend this example to a third table. First, we merge the grants table with the wards table on the ward column again, adding suffixes to the repeated column names. Note that we're using Python's backslash line continuation method to add the second merge on the next line. Python will read this as just one line of code. Without this, Python will throw a syntax error since it will parse it as two separate lines of code, so don't forget your backslash. Now our output table has information about grants, business, and wards. We can now complete our analysis.



**Results**

We can now sum the grants by ward and plot the results. Some wards have received more grants than others.



**Merging even more...**

We could continue to merge additional tables as needed. We stopped at three, but if needed, we could continue to add more. The code here shows the pattern you would follow as you merge more tables.

