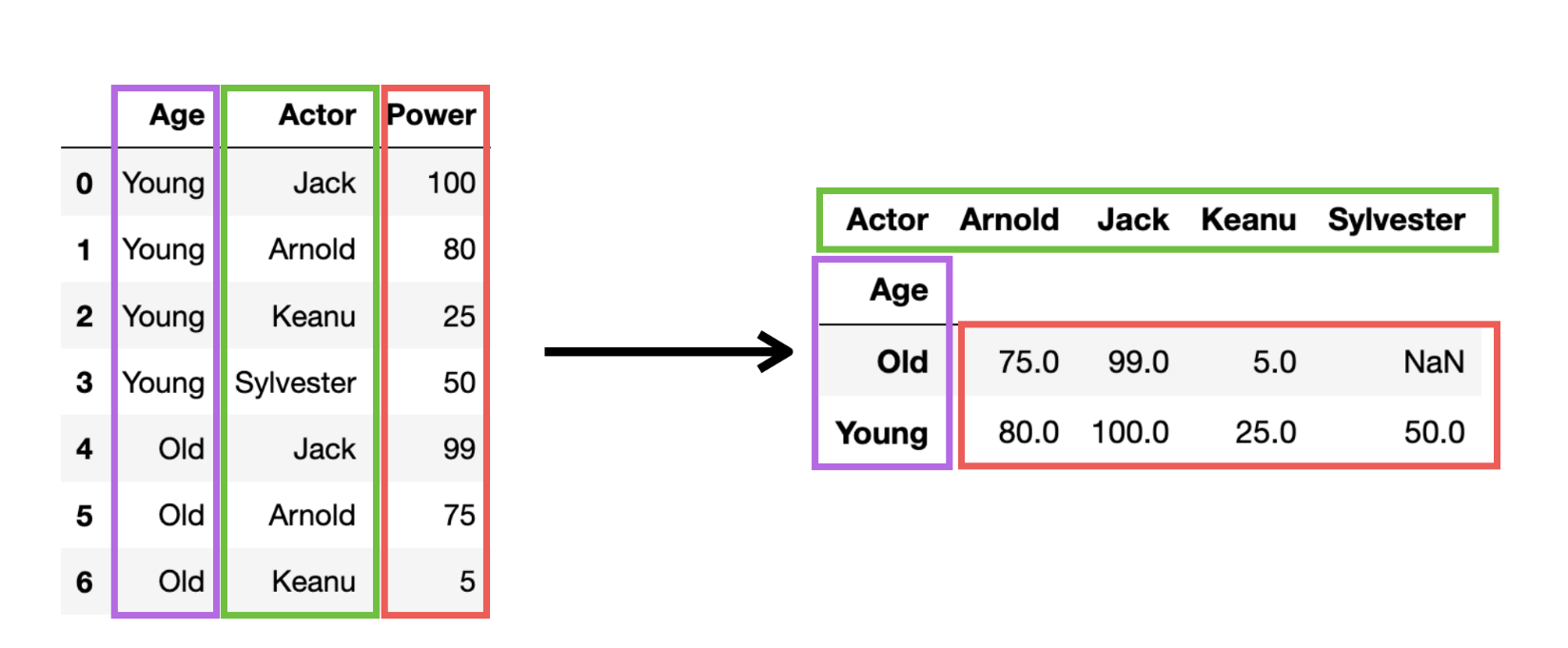
**Data Manipulation: Pivoting DataFrames**

**The**[**.pivot()**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html)**method**

Sometimes you want to convert your DataFrame so that each category has its own column. For example, suppose you needed to take the table below and create a separate column for each actor, where each row is the Age of the actor:



How would you do this with the DataFrame below?

1. test\_df = pd.DataFrame({'Age': ['Young', 'Young', 'Young', 'Young', 'Old', 'Old', 'Old', 'Old'],
2. 'Actor': ['Jack', 'Arnold', 'Keanu', 'Sylvester', 'Jack', 'Arnold', 'Keanu', 'Sylvester'],
3. 'Power': [100, 80, 25, 50, 99, 75, 5, 30]})
4. test\_df

The easiest way to accomplish this is by using the .pivot() method in Pandas. Try the example for yourself. The thing to understand is how to supply the correct aguments to get the desired outcome. The index are the categories for the rows. The columns are the categories for the columns. And the values are what you want in the new cells.

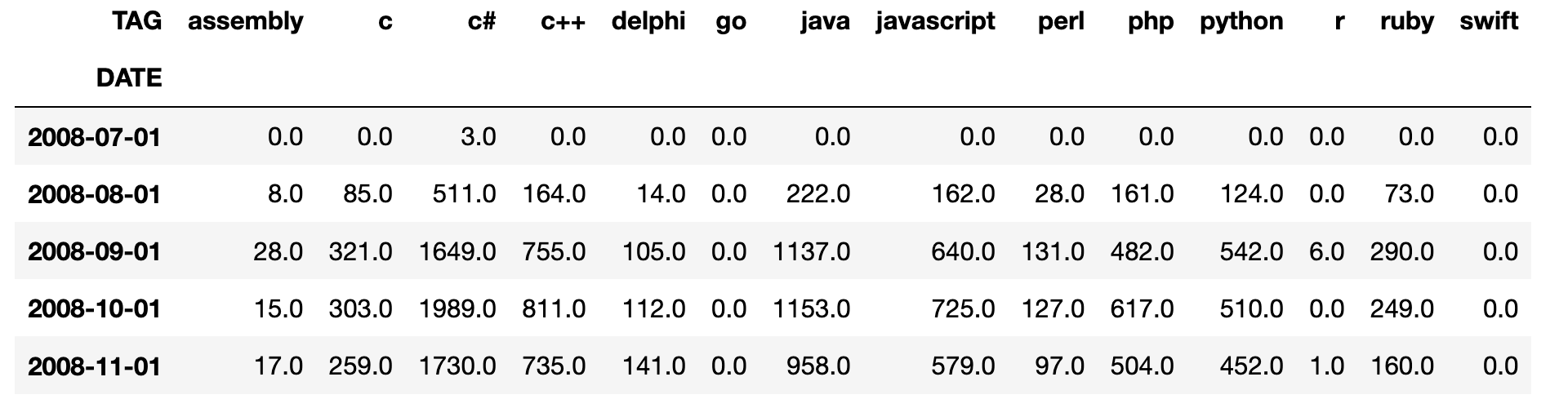
1. pivoted\_df = test\_df.pivot(index='Age', columns='Actor', values='Power')
2. pivoted\_df

However, there's one very important thing to notice. What happens if a value is missing? In the example above there's no value for old Sylvester. In this case, the .pivot() method will insert a NaN value.

**Mini-Challenge**

* Can you pivot the df DataFrame so that each row is a date and each column is a programming language? Store the result under a variable called reshaped\_df.
* Examine the dimensions of the reshaped DataFrame. How many rows does it have? How many columns?
* Examine the head and the tail of the DataFrame. What does it look like?
* Print out the column names.
* Count the number of entries per column.

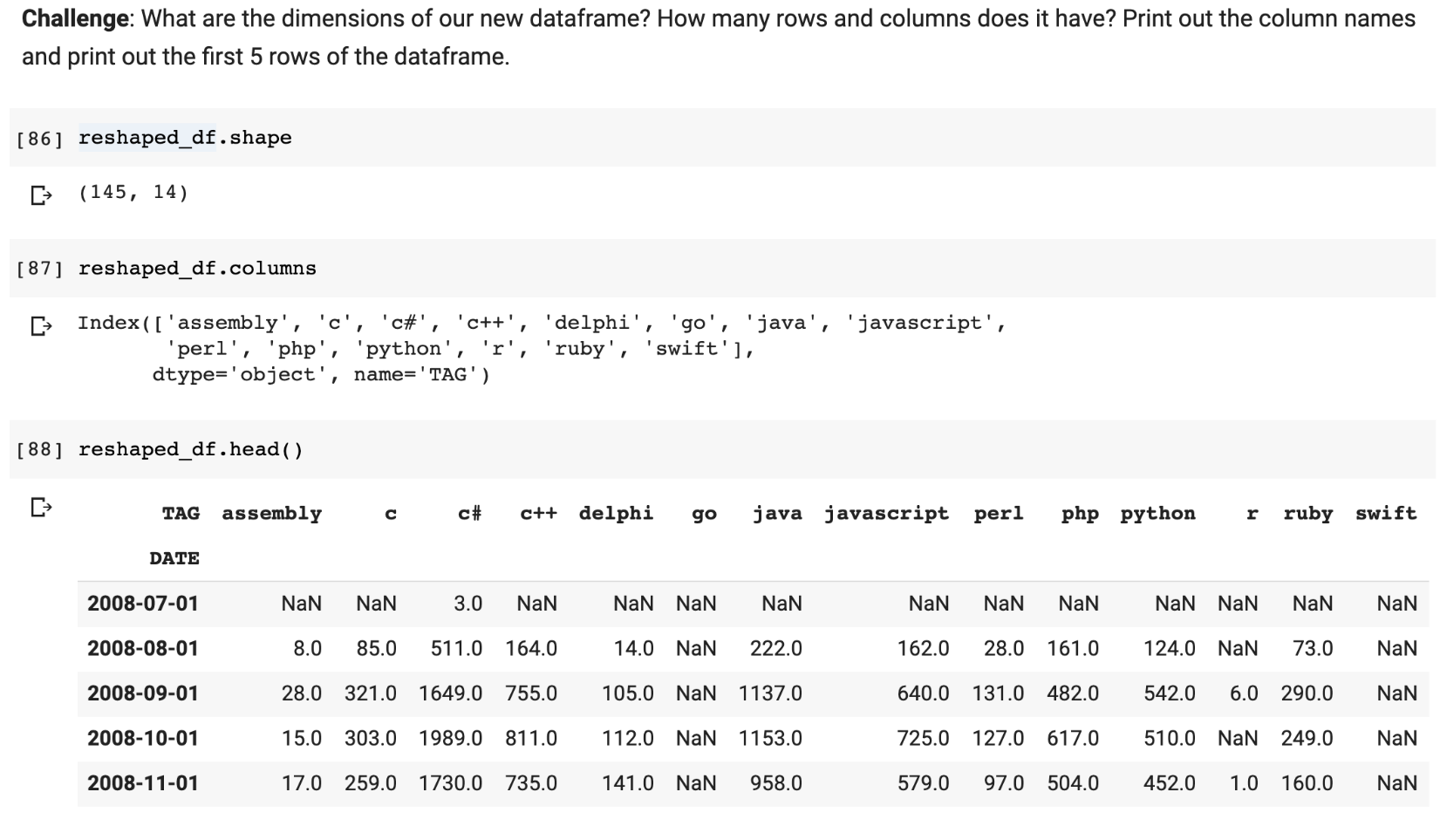
You should get something like this:



**Solution**

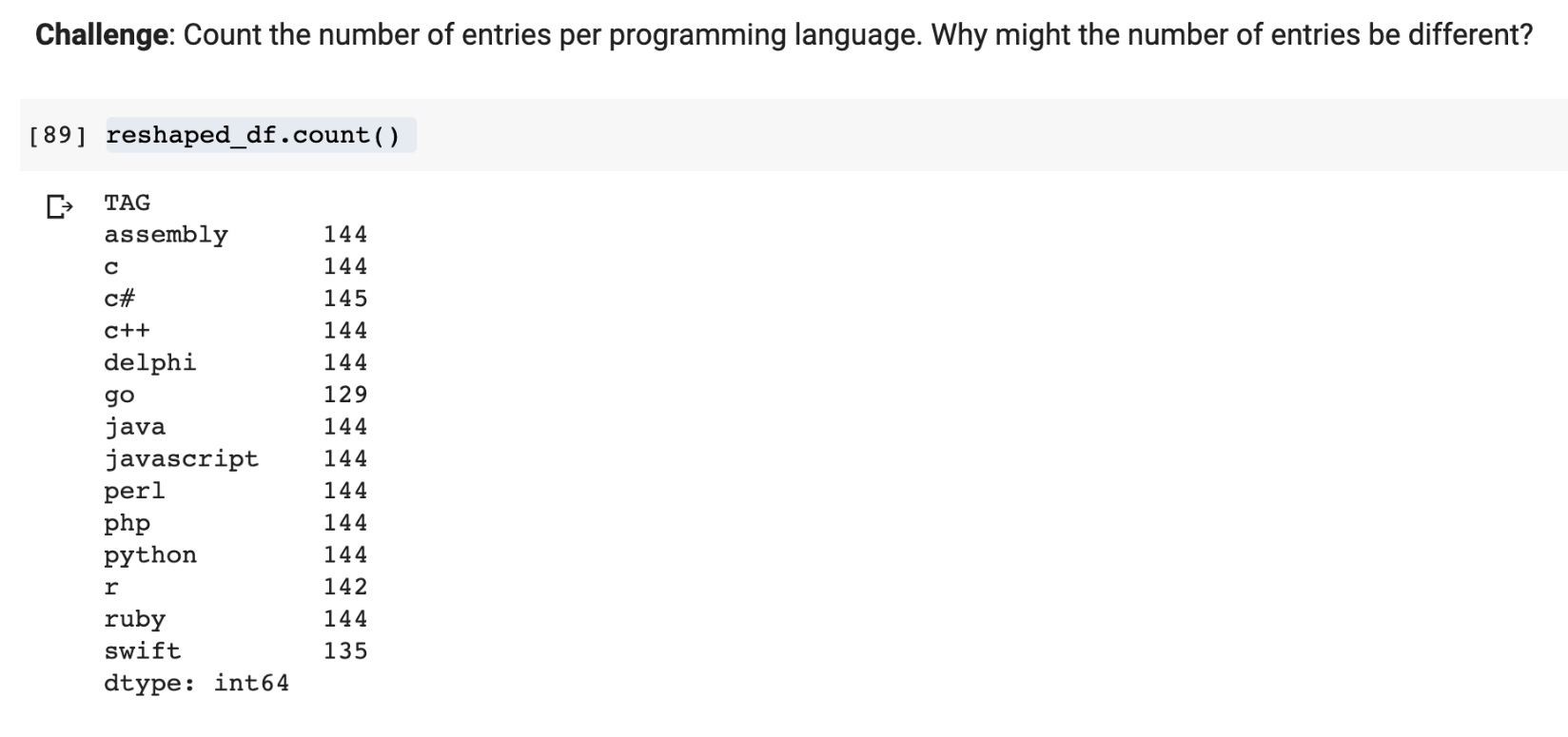
Here's how you pivot our existing DataFrame to get the outcome above:

1. reshaped\_df = df.pivot(index='DATE', columns='TAG', values='POSTS')



We have 145 rows and 14 columns in the new DataFrame. Each programming language became a column and our date column became the new index (i.e., the label for the rows).

When we count the number of entries per column we see that not all languages are the same. The reason is that the .count() method excludes NaN values. When we pivoted the DataFrame the NaN values were inserted when there were no posts for a language in that month (e.g., Swift in July, 2008).



**Dealing with NaN Values**

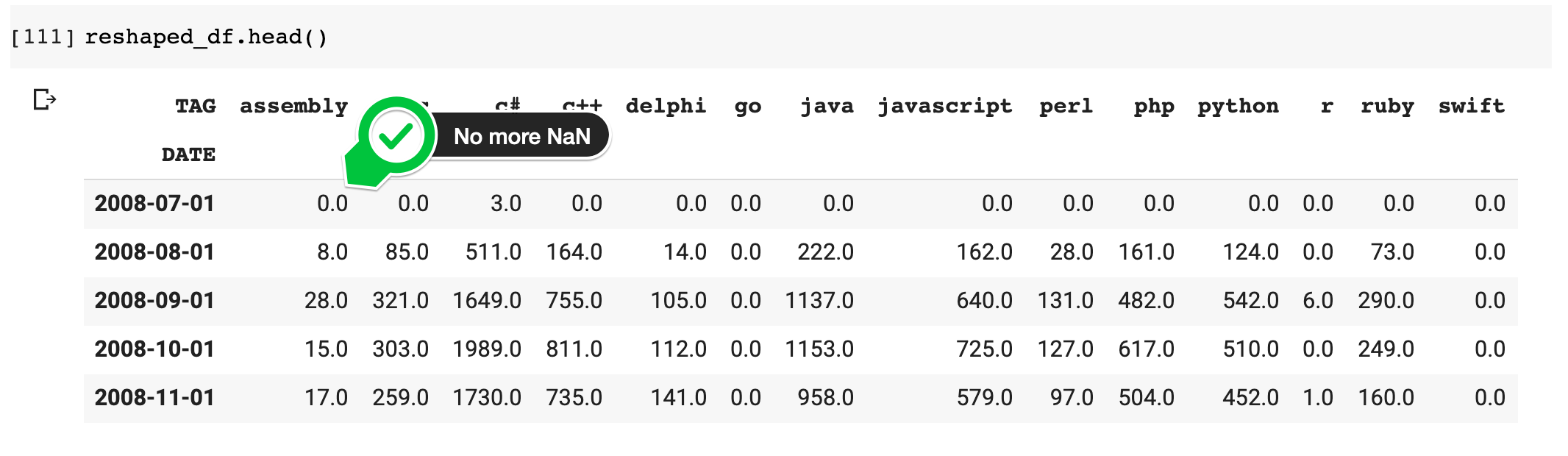
In this case, we don't want to drop the rows that have a NaN value. Instead, we want to substitute the number 0 for each NaN value in the DataFrame. We can do this with the .fillna() method.

1. reshaped\_df.fillna(0, inplace=True)

The inplace argument means that we are updating reshaped\_df. Without this argument we would have to write something like this:

1. reshaped\_df = reshaped\_df.fillna(0)

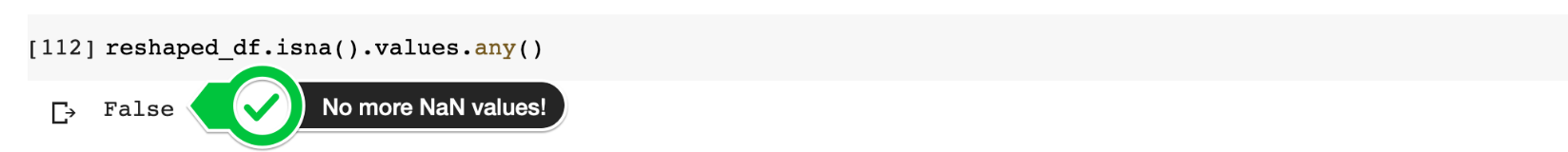
Let's check if we successfully replaced all the NaN values in our DataFrame.



We can also check if there are any NaN values left in the entire DataFrame with this line:

1. reshaped\_df.isna().values.any()

Here we are using the .isna() method that we've used before, but we're chaining two more things: the values attribute and the any() method. This means we don't have to search through the entire DataFrame to spot if .isna() is True.



Now we're all set to create some charts and visualize our data. For all of that and more, I'll see you in the next lesson!