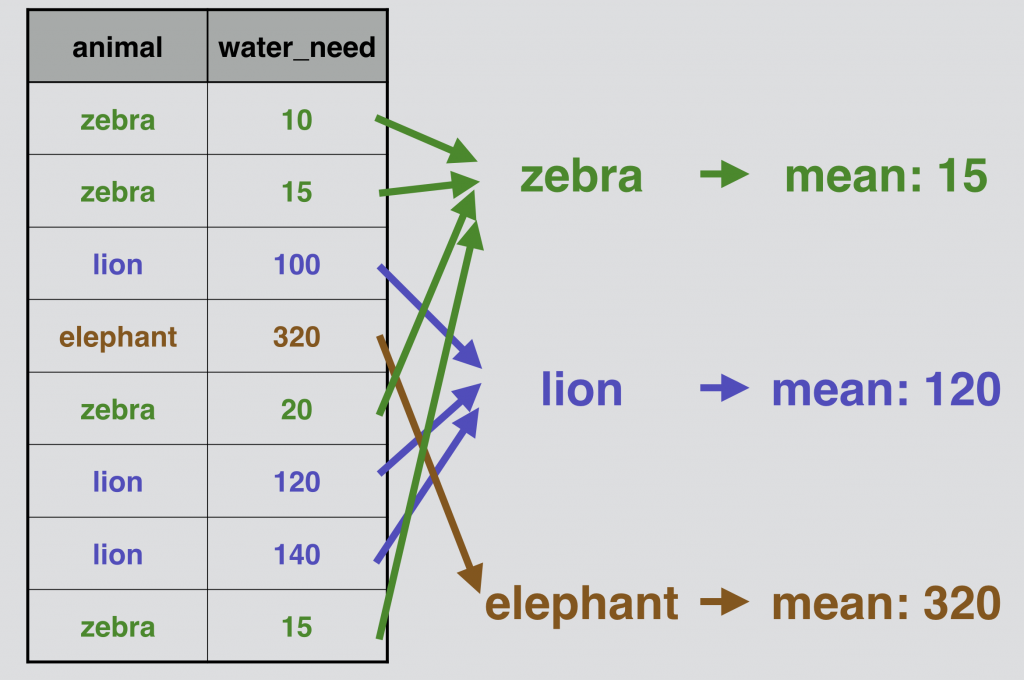
|  |
| --- |
| **Grouping in Pandas** |

As a Data Analyst or Scientist you will probably do segmentations all the time. For instance, it’s nice to know the mean water\_need of all animals (we have just learned that it’s 347.72).

But very often it’s much more actionable to break this number down – let’s say – by animal types. With that, we can compare the species to each other – or we can find outliers.

Here’s a simplified visual that shows how pandas performs “segmentation” (grouping and aggregation) based on the column values!



## Pandas .groupby in action

Let’s do the above presented grouping and aggregation for real, on our zoo dataframe!

We have to fit in a groupby keyword between our zoo variable and our .mean() function:

zoo.groupby('animal').mean()



Just as before, pandas automatically runs the .mean() calculation for all remaining columns (the animal column obviously disappeared, since that was the column we grouped by). You can either ignore the uniq\_id column, or you can remove it afterwards by using one of these syntaxes:

zoo.groupby('animal').mean()[['water\_need']]

(This returns a Dataframe object.)

zoo.groupby('animal').mean().water\_need

(This returns a Series object.)

**Time to test your understanding. Load data from pandas\_tutorial\_read.csv to article\_read and complete the following exercises:**

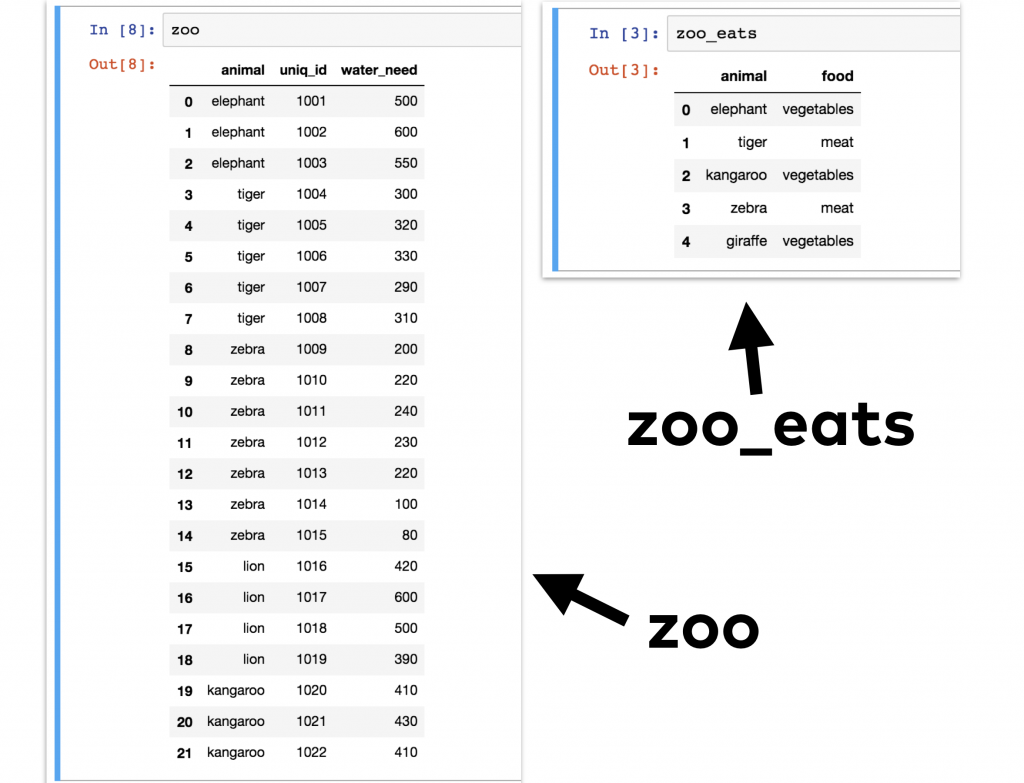
1. **Find the most frequent source in the article\_read dataframe. (Hint: you need .groupby() for ‘source’ column and count them. The correct answer is Reddit!!!!)**
2. **From exercise 20, show only ‘user\_id’**
3. **For the users coming from ‘country\_2’, what is the most frequent topic and source combined? [Hint: Step 1: you need to filter for only ‘country\_2’. Step 2: you need to group ‘topic’ and ‘source’. Step 3: apply .count( )]**

**Data Merging in Pandas**

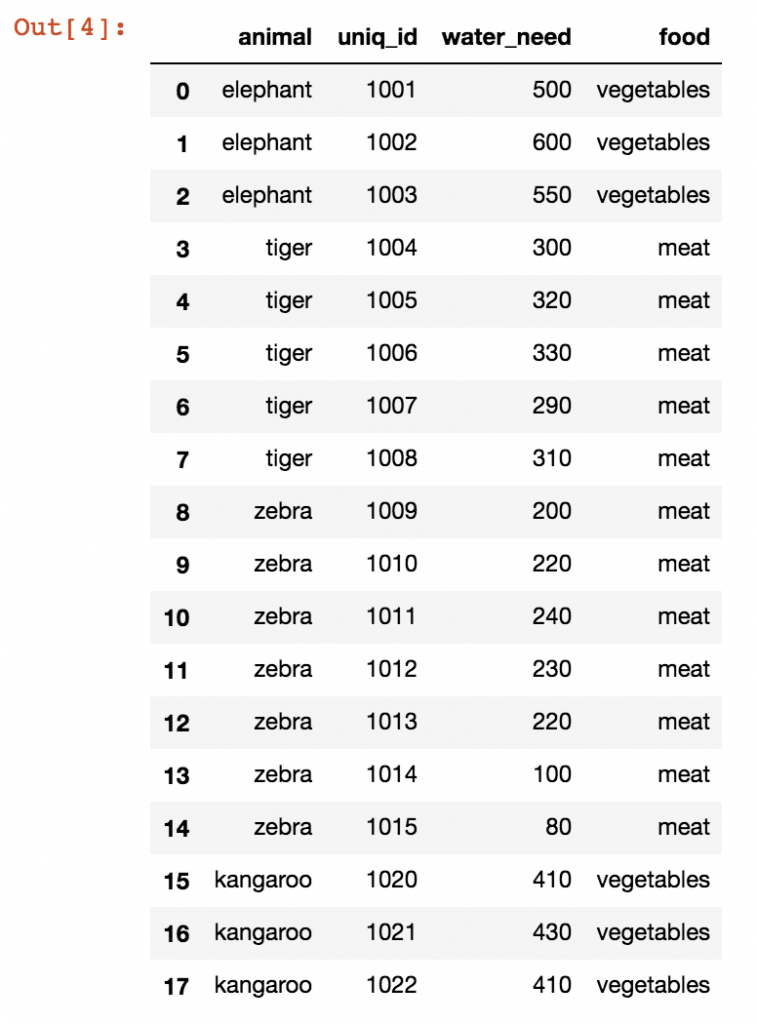
In real life data projects, we usually don’t store all the data in one big data table. We store it in a few smaller ones instead. There are many reasons behind this; by using multiple data tables, it’s easier to manage your data, it’s easier to avoid redundancy, you can save some disk space, you can query the smaller tables faster, etc.

The point is that it’s quite usual that during your analysis you have to pull your data from two or more different tables. The solution for that is called **merge**.

Let’s take our zoo dataframe in which we have all our animals… and let’s say that we have another dataframe, zoo\_eats, that contains information about the food requirements for each species. [Note: both are available in the Portal]



We want to merge these two pandas dataframes into one big dataframe. Something like this:



This can easily be done by using .merge() as shown below.

zoo.merge(zoo\_eats)

**[Note: Originally, there are 21 records in zoo. However, after merging only 17 records remain. Can you guess what is happening here?]**

First, I specified the first dataframe (zoo), then I applied the .merge() pandas method on it and as a parameter I specified the second dataframe (zoo\_eats). I could have done this the other way around:

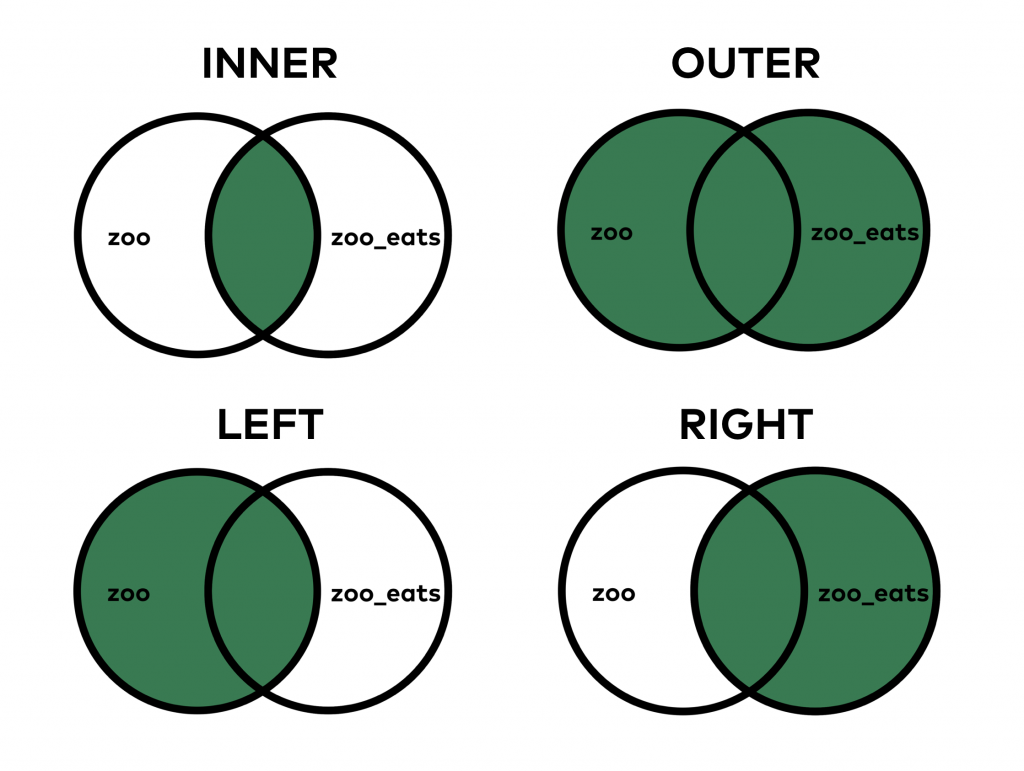
zoo\_eats.merge(zoo) is symmetric to: zoo.merge(zoo\_eats)

The only difference between the two is the order of the columns in the output table. (Just try it!)

1. **Try zoo\_eats.merge(zoo) and zoo.merge(zoo\_eats)**

As you can see, the basic merge method is pretty simple. Sometimes you have to add a few extra parameters though.

One of the most important questions is ***how*** you want to merge these tables. In SQL, we learned that there are different JOIN types.



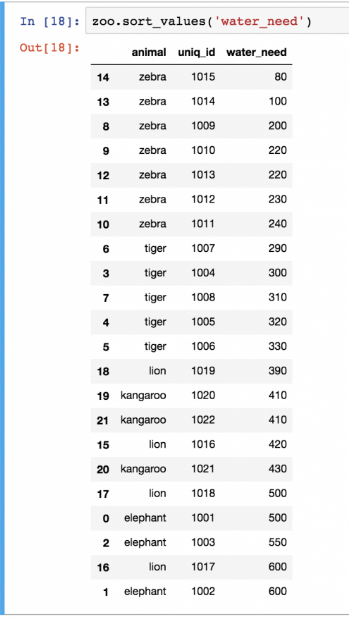
When you do an INNER JOIN (that’s the default both in SQL and pandas), you merge only those values that are found in **both tables**. On the other hand, when you do the OUTER JOIN, it merges all values, even if you can find some of them in only one of the tables.

To specify how we are going to merge data, a syntax ‘how = ’ is needed.

1. **Try**
   1. **zoo.merge(zoo\_eats, how = 'outer')**
   2. **zoo.merge(zoo\_eats, how = 'left')**
   3. **zoo.merge(zoo\_eats, how = 'right')**

**Data Sorting and Data Munging (Cleansing) in Pandas**

Sorting is essential. The basic sorting method is not too difficult in pandas. The function is called sort\_values() and it works like this:



The only parameter I used here was the name of the column I want to sort by, in this case the water\_need column. Quite often, you have to sort by multiple columns, so in general, I recommend using the by keyword for the columns:

zoo.sort\_values(by = ['animal', 'water\_need'])

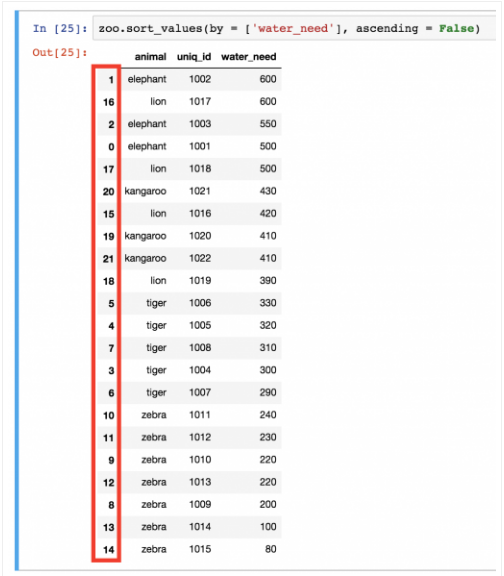
1. **Try the above Python code.**
2. **From exercise 25, swap positions of ‘animal’ and ‘water\_need’, and observe the result.**

Note: you can use the by keyword with one column only, too, like zoo.sort\_values(by = ['water\_need']).

sort\_values sorts in ascending order, but obviously, you can change this and do descending order as well:

zoo.sort\_values(by = ['water\_need'], ascending = False)

1. **Try the above Python code. You should get the following result.**



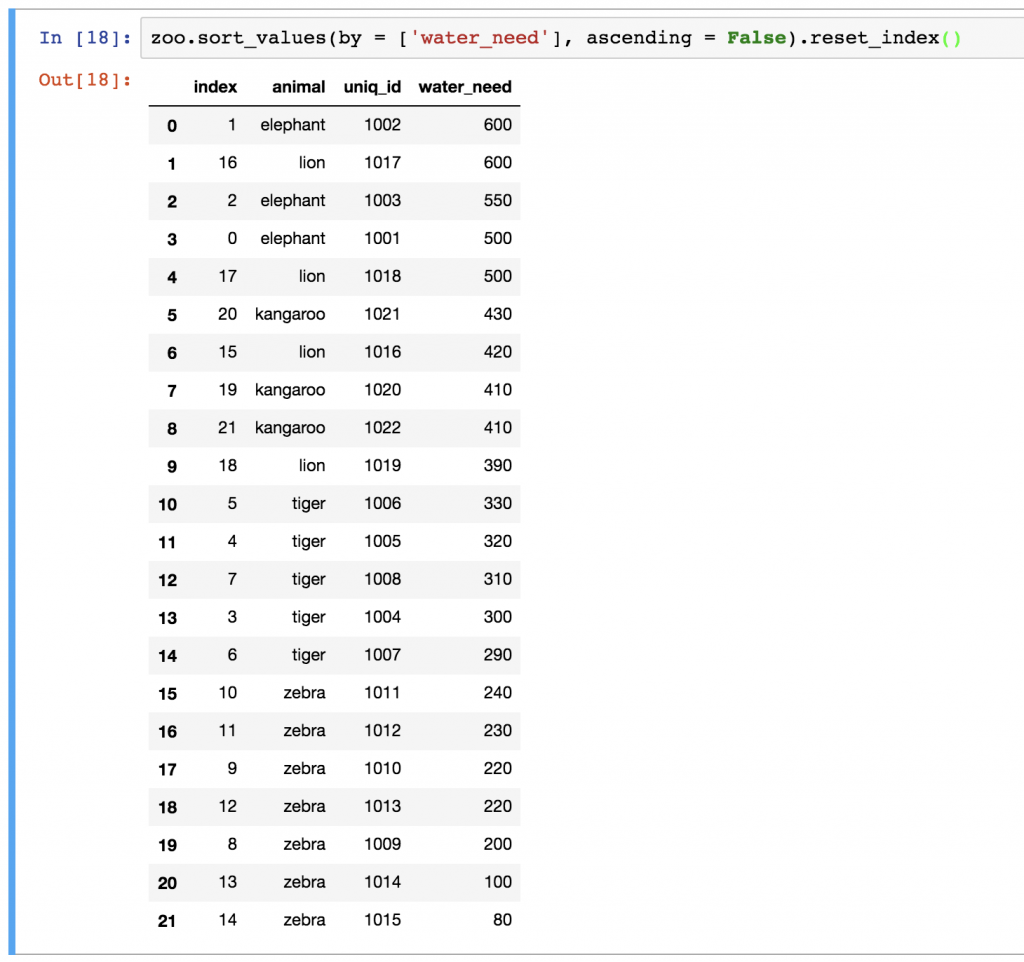
**What a mess with all the indexes after that last sorting, right?**

It’s not just that it’s ugly… wrong indexing can mess up your visualizations or even your machine learning models.

The point is: in certain cases, when you have done a transformation on your dataframe, you have to re-index the rows. For that, you can use the reset\_index() method. For instance:

zoo.sort\_values(by = ['water\_need'], ascending = False)**.reset\_index()**

1. **Try the above Python code. You should get.**



As you can see, our new dataframe kept the old indexes, too. If you want to remove them, just add the drop = True parameter:

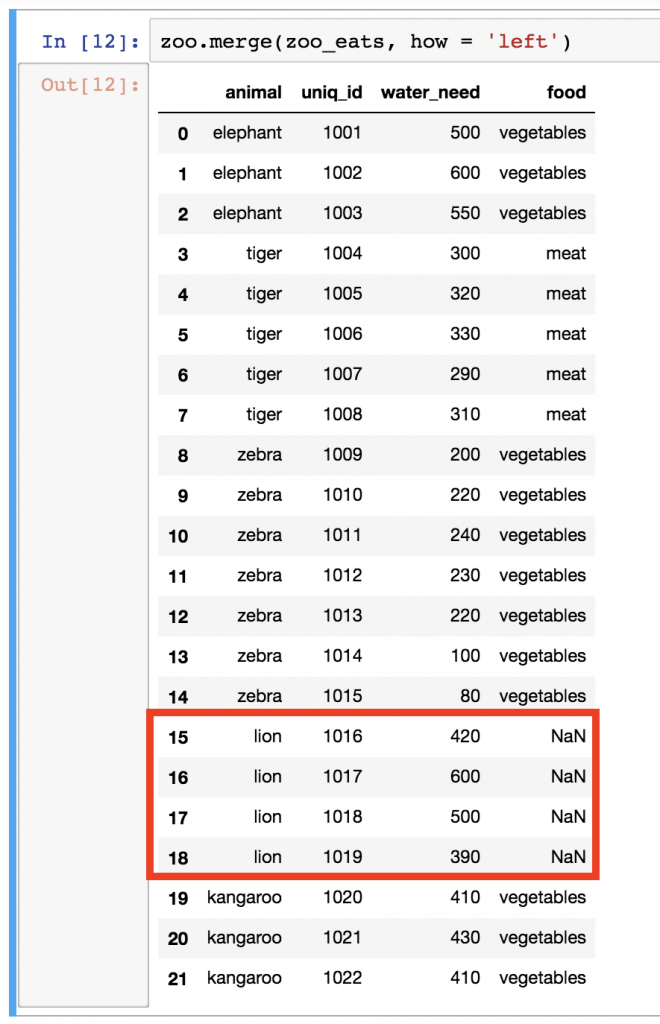
zoo.sort\_values(by = ['water\_need'], ascending = False).reset\_index**(drop = True)**

1. **Try the above Python code.**

|  |
| --- |
| **Data Munging (Cleansing)** |

Let’s rerun the left-merge method that we have used above:

zoo.merge(zoo\_eats, how = 'left')



Remember? These are all our animals. The problem is that we have NaN values for lions. NaN itself can be really distracting, so I usually like to replace it with something more meaningful. In some cases, this can be a 0 value, or in other cases a specific string value, but this time, I’ll go with unknown. Let’s use the fillna() function, which basically finds and replaces all NaN values in our dataframe:

1. **Try the below Python code and see the results.**

zoo.merge(zoo\_eats, how = 'left')**.fillna('unknown')**

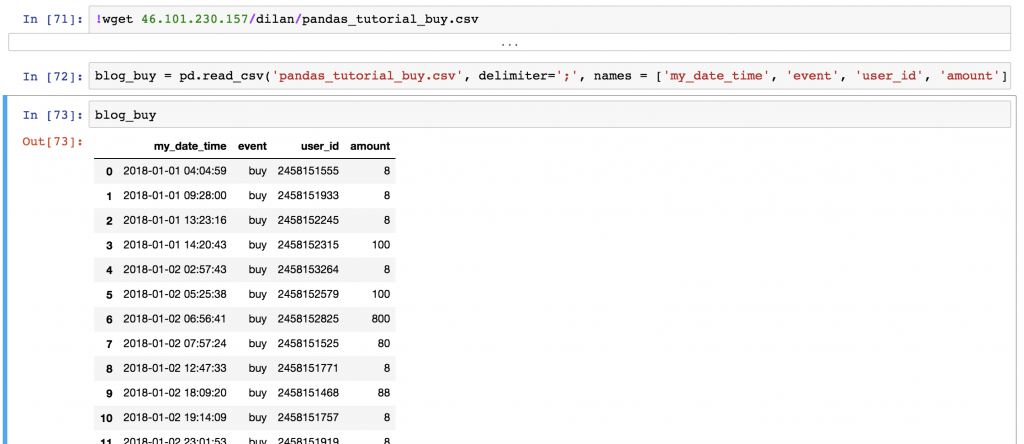
Note: since we know that lions eat meat, we could have filled?

1. **Fill in the appropriate food for lion (meat or vegetables????).**

|  |
| --- |
| **Study the following example.** |

Download pandas\_tutorial\_buy.csv from the portal. Load pandas\_tutorial\_read.csv to article\_read and load pandas\_tutorial\_buy.csv to blog\_buy.

The article\_read dataset shows all the users who read an article on the blog, and the blog\_buy dataset shows all the users who bought something on the very same blog between 2018-01-01 and 2018-01-07.



I have two questions for you:

* **TASK #1: What’s the average (mean) revenue between**2018-01-01**and**2018-01-07**from the users in the**article\_read**dataframe?**
* **TASK #2: Print the top 3 countries by total revenue between**2018-01-01**and**2018-01-07**! (Obviously, this concerns the users in the**article\_read**dataframe again.)**

**Solution to Task#1**

**The average revenue is: 1.0852**

Here’s the code:



A short explanation:

* *(On the screenshot, at the beginning, I included the two extra cells where I import pandas and numpy, and where I read the csv files into my Jupyter Notebook.)*
* **In step\_1**, I merged the two tables (article\_read and blog\_buy) based on the user\_id columns. I kept all the readers from article\_read, even if they didn’t buy anything, because 0s should be counted in to the average revenue value. And I removed everyone who bought something but wasn’t in the article\_read dataset (that was fixed in the task). So all in all that led to a *left join*.
* **In step\_2,** I removed all the unnecessary columns, and kept only amount.
* **In step\_3,** I replaced NaN values with 0s.
* And eventually I did the .mean() calculation.

Solution to Task#2



A short explanation:

* **At step\_1,** I used the same merging method that I used in TASK #1.
* **At step\_2,** I filled up all the NaN values with 0s.
* **At step\_3,** I summarized the numerical values by countries.
* **At step\_4,** I took away all columns but amount.
* And**at step\_5,** I sorted the results in descending order, so I can see my top list!
* **Finally**, I printed the first 3 lines only.