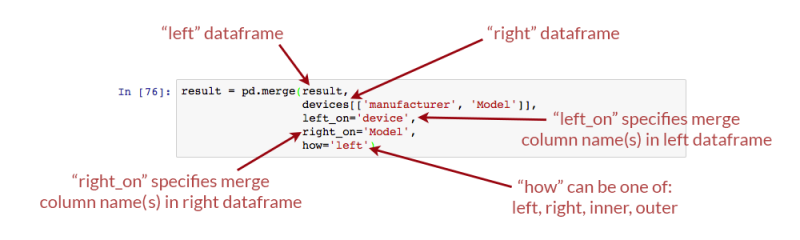
Merge and Join DataFrames with Pandas in Python

[10 Comments](https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/" \l "comments) / [blog](https://www.shanelynn.ie/category/blog/), [data science](https://www.shanelynn.ie/category/blog/data-science/), [Pandas](https://www.shanelynn.ie/category/pandas/), [python](https://www.shanelynn.ie/category/blog/python/), [Tutorials](https://www.shanelynn.ie/category/tutorials/) / By [shanelynn](https://www.shanelynn.ie/author/shanelynn/" \o "View all posts by shanelynn)

In any real world data science situation with [Python](http://python/), you’ll be about 10 minutes in when you’ll need to merge or join [Pandas](http://pandas.pydata.org/) Dataframes together to form your analysis dataset. [Merging and joining dataframes](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) is a core process that any aspiring data analyst will need to master. This blog post addresses the process of merging datasets, that is, joining two datasets together based on common columns between them. Key topics covered here:

* [What is a merge or join of two dataframes](https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/#whatis)?
* [What are inner, outer, left and right merges](https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/#mergetypes)?
* [How do I merge two dataframes with different common column names? (left\_on and right\_on syntax)](https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/#differentnames)

If you’d like to work through the tutorial yourself, I’m using a [Jupyter notebook](http://jupyter.org/) setup with [Python 3.5.2 from Anaconda](https://www.continuum.io/downloads), and I’ve posted the code on GitHub here. I’ve included the sample datasets in the GitHub repository.

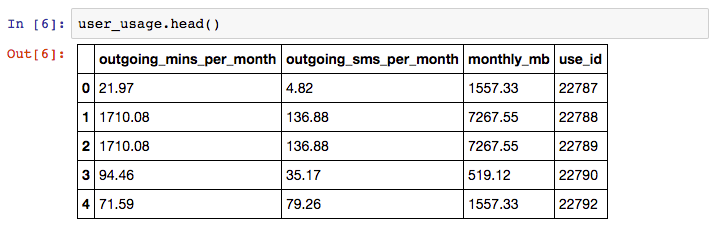
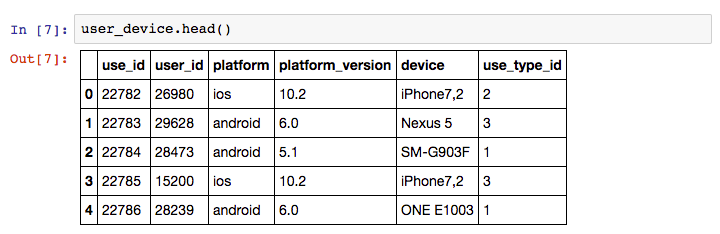
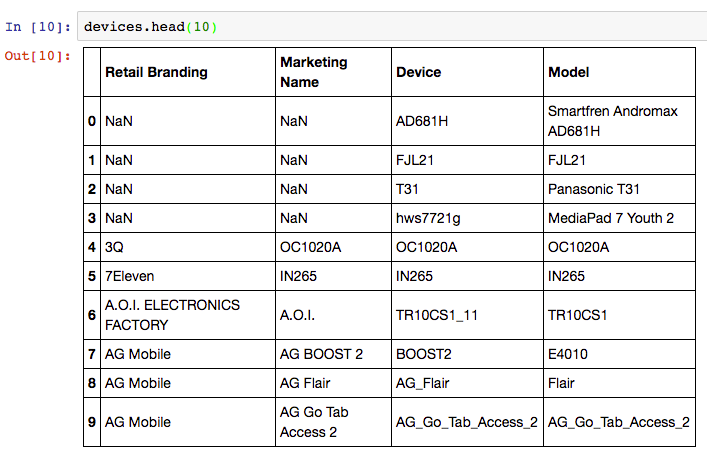
[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-join-different-variable-names-copy-e1488722312527.png)Merging overview if you need a quickstart (all explanations below)! The Pandas merge() command takes the left and right dataframes, matches rows based on the “on” columns, and performs different types of merges – left, right, etc.

**Example data**

For this post, I have taken some real data from the [KillBiller application](http://www.killbiller.com/) and some downloaded data, contained in three CSV files:

* [user\_usage.csv](https://github.com/shanealynn/Pandas-Merge-Tutorial/blob/master/user_usage.csv) – A first dataset containing users monthly mobile usage statistics
* [user\_device.csv](https://github.com/shanealynn/Pandas-Merge-Tutorial/blob/master/user_device.csv) – A second dataset containing details of an individual “use” of the system, with dates and device information.
* [android\_devices.csv](https://github.com/shanealynn/Pandas-Merge-Tutorial/blob/master/android_devices.csv) – A third dataset with device and manufacturer data, which lists all Android devices and their model code, obtained from Google [here](https://support.google.com/googleplay/answer/1727131?hl=en-GB).

We can load these CSV files as [Pandas DataFrames](http://pandas.pydata.org/pandas-docs/stable/dsintro.html?highlight=dataframe) into pandas using the Pandas [read\_csv](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html" \t "_blank)command, and examine the contents using the [DataFrame head()](http://pandas.pydata.org/pandas-docs/stable/basics.html" \l "basics) command.

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/02/pandas_merge_user_usage_head.png)Sample usage information from the KillBiller application showing monthly mobile usage statistics for a subset of users.[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/02/pandas_merge_user_device_head.png)User information from KillBiller application giving the device and OS version for individual “uses” of the KillBiller application.[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/02/pandas_merge_devices_head.png)Android Device data, containing all Android devices with manufacturer and model details.

There are linking attributes between the sample datasets that are important to note – “use\_id” is shared between the user\_usage and user\_device, and the “device” column of user\_device and “Model” column of the devices dataset contain common codes.

**Sample problem**

We would like to determine if the usage patterns for users differ between different devices. For example, do users using [Samsung](http://www.samsung.com/ie/) devices use more call minutes than those using  [LG](http://www.lg.com/uk" \t "_blank) devices? This is a toy problem given the small sample size in these dataset, but is a perfect example of where merges are required.

We want to form a single dataframe with columns for user usage figures (calls per month, sms per month etc) and also columns with device information (model, manufacturer, etc). We will need to “merge” (or “join”) our sample datasets together into one single dataset for analysis.

**Merging DataFrames**

*“Merging” two datasets is the process of bringing two datasets together into one, and aligning the rows from each based on common attributes or columns.*

The words “merge” and “join” are used relatively interchangeably in Pandas and other languages, namely [SQL](https://en.wikipedia.org/wiki/SQL) and [R](https://en.wikipedia.org/wiki/R_(programming_language)). In Pandas, there are separate “[merge](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.merge.html)” and “[join](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.join.html?highlight=join#pandas.DataFrame.join)” functions, both of which do similar things.

In this example scenario, we will need to perform two steps:

1. For each row in the user\_usage dataset – make a new column that contains the “device” code from the user\_devices dataframe. i.e. for the first row, the use\_id is 22787, so we go to the user\_devices dataset, find the use\_id 22787, and copy the value from the “device” column across.
2. After this is complete, we take the new device columns, and we find the corresponding “Retail Branding” and “Model” from the devices dataset.
3. Finally, we can look at different statistics for usage [splitting and grouping data](https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/) by the device manufacturers used.

**Can I use a for loop?**

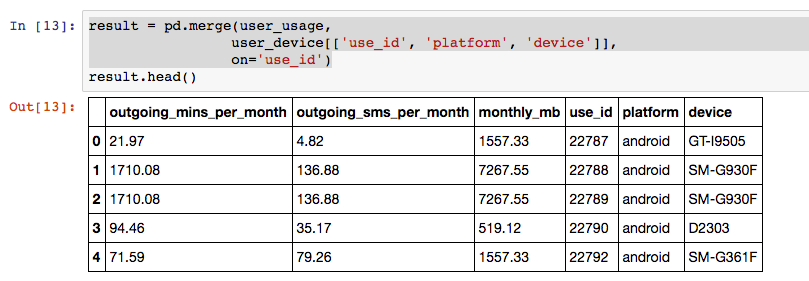
Yes. You could write for loops for this task. The first would loop through the use\_id in the user\_usage dataset, and then find the right element in user\_devices. The second for loop will repeat this process for the devices.

However, using for loops will be much slower and more verbose than using Pandas merge functionality. So,  if you come across this situation – **don’t use for loops**.

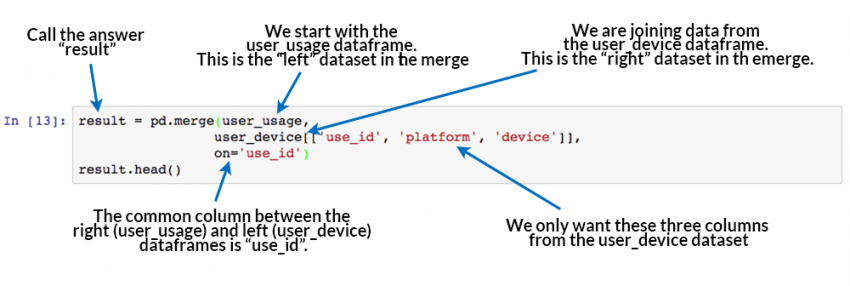
**Merging user\_usage with user\_devices**

Lets see how we can correctly add the “device” and “platform” columns to the user\_usage dataframe using the [Pandas Merge](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) command.

1. result = pd.merge(user\_usage,
2. user\_device[['use\_id', 'platform', 'device']],
3. on='use\_id')
4. result.head()

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/02/pandas_merge_result_1.png)Result of merging user usage with user devices based on a common column.

So that works, and very easily! Now – how did that work? What was the pd.merge command doing?

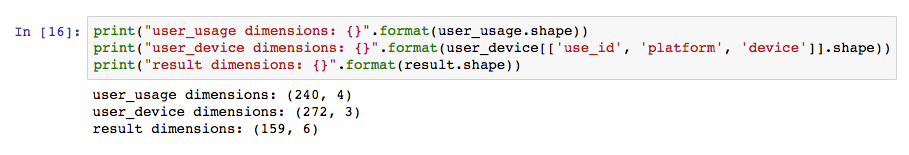
[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/02/pd-merge_intro_image.png)How Pandas Merge commands work. At the very least, merging requires a “left” dataset, a “right” dataset, and a common column to merge “on”.

The merge command is the key learning objective of this post. The merging operation at its simplest takes a left dataframe (the first argument), a right dataframe (the second argument), and then a merge column name, or a column to merge “on”. In the output/result, rows from the left and right dataframes are matched up where there are common values of the merge column specified by “on”.

With this result, we can now move on to get the manufacturer and model number from the “devices” dataset. However, first we need to understand a little more about merge types and the sizes of the output dataframe.

**Inner, Left, and right merge types**

In our example above, we merged user\_usage with user\_devices. The [head](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.head.html)() preview of the result looks great, but there’s more to this than meets the eye. First, let’s look at the [sizes or shapes](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.shape.html) of our inputs and outputs to the merge command:

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/02/dataset_sizes_after_merge.png)The resultant size of the dataset after the merge operation may not be as expected. Pandas merge() defaults to an “inner” merge operation.

Why is the result a different size to both the original dataframes?

By default, the [Pandas merge operation](http://pandas.pydata.org/pandas-docs/stable/merging.html)acts with an “inner” merge. An inner merge, (or inner join) keeps only the common values in both the left and right dataframes for the result. In our example above, only the rows that contain use\_id values that are common between user\_usage and user\_device remain in the result dataset. We can validate this by looking at how many values are common:

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas_merge_blog_use_id_values-1.png)Only common values between the left and right dataframes are retained by default in Pandas, i.e. an “inner” merge is used.

There are 159 values of use\_id in the user\_usage table that appear in user\_device. These are the same values that also appear in the final result dataframe (159 rows).

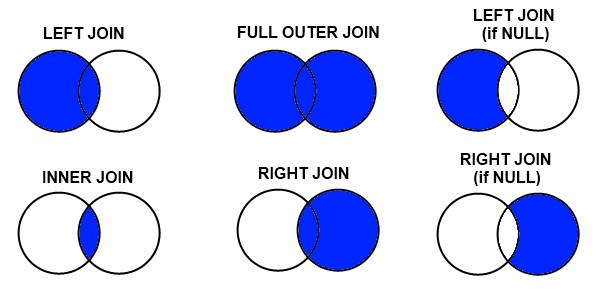
**Other Merge Types**

There are three different types of merges available in Pandas. These merge types are common across most database and data-orientated languages ([SQL](https://www.w3schools.com/sql/sql_join.asp), [R](https://stat.ethz.ch/R-manual/R-devel/library/base/html/merge.html), [SAS](https://www.analyticsvidhya.com/blog/2015/01/introduction-merging-sas/)) and are typically referred to as “joins”. If you don’t know them, learn them now.

1. **Inner Merge / Inner join** – The default Pandas behaviour, only keep rows where the merge “on” value exists in both the left and right dataframes.
2. **Left Merge / Left outer join** – (aka left merge or left join) Keep every row in the left dataframe. Where there are missing values of the “on” variable in the right dataframe, add empty / NaN values in the result.
3. **Right Merge** / **Right outer join** – (aka right merge or right join) Keep every row in the right dataframe. Where there are missing values of the “on” variable in the left column, add empty / NaN values in the result.
4. **Outer Merge / Full outer join –**A full outer join returns all the rows from the left dataframe, all the rows from the right dataframe, and matches up rows where possible, with NaNs elsewhere.

The merge type to use is specified using the “how” parameter in the merge command, taking values “left”, “right”, “inner” (default), or “outer”.

Venn diagrams are commonly used to exemplify the different merge and join types. See this example from [Stack overflow](http://stackoverflow.com/questions/38549/what-is-the-difference-between-inner-join-and-outer-join):

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/join-types-merge-names.jpg)Merge/Join types as used in Pandas, R, SQL, and other data-orientated languages and libraries. Source: Stack Overflow.

If this is new to you, or you are looking at the above with a frown, take the time to watch this video on [“merging dataframes” from Coursera](https://www.coursera.org/learn/python-data-analysis/lecture/08sf6/merging-dataframes) for another explanation that might help. We’ll now look at each merge type in more detail, and work through examples of each.

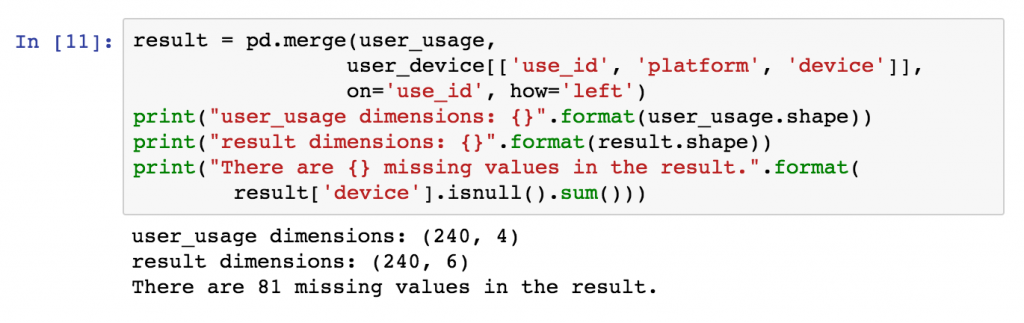
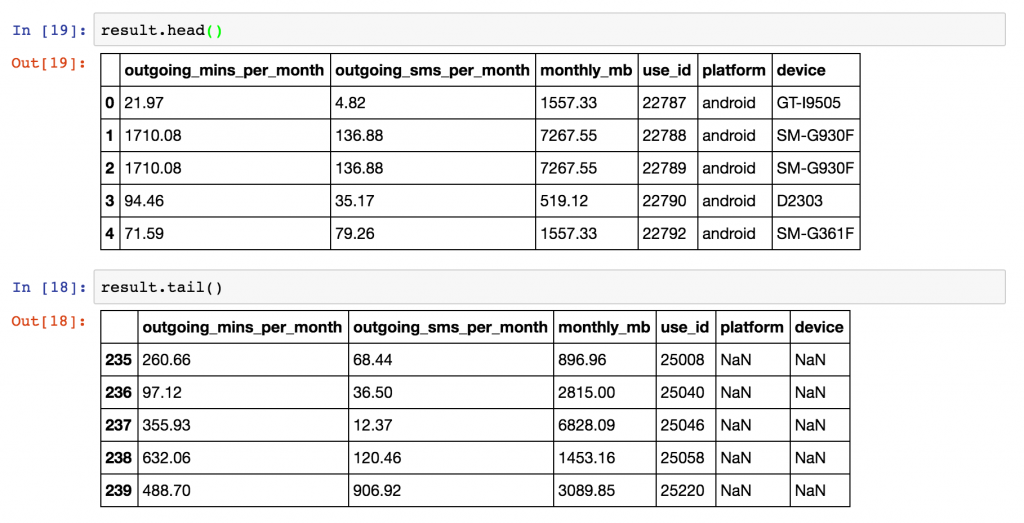
**Example of left merge / left join**

Let’s repeat our merge operation, but this time perform a “left merge” in Pandas.

* Originally, the result dataframe had 159 rows, because there were 159 values of “use\_id” common between our left and right dataframes and an “inner” merge was used by default.
* For our left merge, we expect the result to have the same number of rows as our left dataframe “user\_usage” (240), with missing values for all but 159 of the merged “platform” and “device” columns (81 rows).
* We expect the result to have the same number of rows as the left dataframe because each use\_id in user\_usage appears only once in user\_device. A one-to-one mapping is not always the case. In merge operations where a single row in the left dataframe is matched by multiple rows in the right dataframe, multiple result rows will be generated. i.e. if a use\_id value in user\_usage appears twice in the user\_device dataframe, there will be two rows for that use\_id in the join result.

You can change the merge to a left-merge with the “how” parameter to your merge command. The top of the result dataframe contains the successfully matched items, and at the bottom contains the rows in user\_usage that didn’t have a corresponding use\_id in user\_device.

1. result = pd.merge(user\_usage,
2. user\_device[['use\_id', 'platform', 'device']],
3. on='use\_id',
4. how='left')

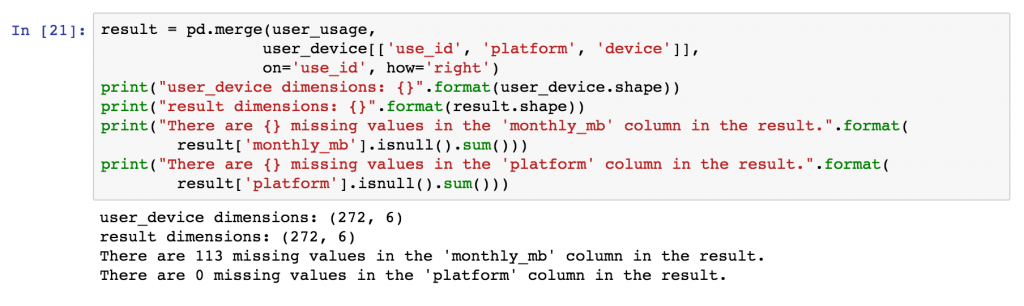
[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas_merge_left_example.png)Left join example in pandas. Specify the join type in the “how” command. A left join, or left merge, keeps every row from the left dataframe.[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-left-join-result.png)Result from left-join or left-merge of two dataframes in Pandas. Rows in the left dataframe that have no corresponding join value in the right dataframe are left with NaN values.

**Example of right merge / right join**

For examples sake, we can repeat this process with a right join / right merge, simply by replacing *how=’left’* with *how=’right’* in the Pandas merge command.

1. result = pd.merge(user\_usage,
2. user\_device[['use\_id', 'platform', 'device']],
3. on='use\_id',
4. how='right')

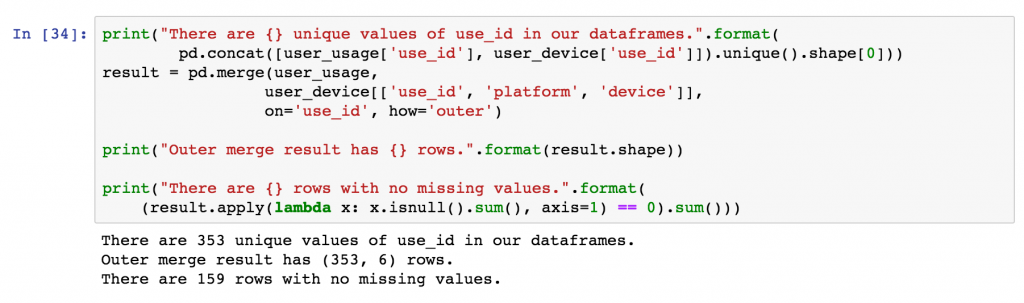
The result expected will have the same number of rows as the right dataframe, user\_device, but have several empty, or NaN values in the columns originating in the left dataframe, user\_usage (namely “outgoing\_mins\_per\_month”, “outgoing\_sms\_per\_month”, and “monthly\_mb”). Conversely, we expect no missing values in the columns originating in the right dataframe, “user\_device”.

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-right-dataframe-result-missing-values.png)Example of a right merge, or right join. Note that the output has the same number of rows as the right dataframe, with missing values only where use\_id in the left dataframe didn’t match anything in the left.

**Example of outer merge / full outer join**

Finally, we will perform an [outer merge](https://www.w3schools.com/Sql/sql_join_full.asp) using Pandas, also referred to as a “full outer join” or just “outer join”. An outer join can be seen as a combination of left and right joins, or the opposite of an inner join. In outer joins, every row from the left and right dataframes is retained in the result, with NaNs where there are no matched join variables.

As such, we would expect the results to have the same number of rows as there are distinct values of “use\_id” between user\_device and user\_usage, i.e. every join value from the left dataframe will be in the result along with every value from the right dataframe, and they’ll be linked where possible.

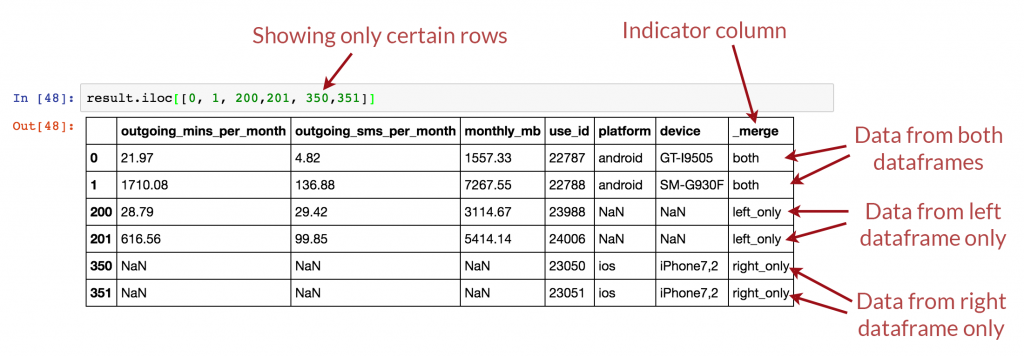
[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-outer-join-result-e1488446880892.png)Outer merge result using Pandas. Every row from the left and right dataframes is retained in the result, with missing values or numpy NaN values where the merge column doesn’t match.

In the diagram below, example rows from the outer merge result are shown, the first two are examples where the “use\_id” was common between the dataframes, the second two originated only from the left dataframe, and the final two originated only from the right dataframe.

**Using merge indicator to track merges**

To assist with the identification of where rows originate from, Pandas provides an [“indicator” parameter](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) that can be used with the merge function which creates an additional column called “\_merge” in the output that labels the original source for each row.

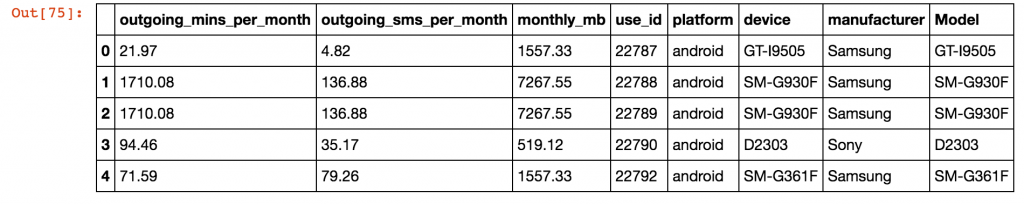
1. result = pd.merge(user\_usage,
2. user\_device[['use\_id', 'platform', 'device']],
3. on='use\_id',
4. how='outer',
5. indicator=True)

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-outer-join-example-1.png)Example rows from outer merge (full outer join) result. Note that all rows from left and right merge dataframes are included, but NaNs will be in different columns depending if the data originated in the left or right dataframe.

**Final Merge – Joining device details to result**

Coming back to our original problem, we have already merged user\_usage with user\_device, so we have the platform and device for each user. Originally, we used an “[inner merge](https://www.w3schools.com/Sql/sql_join_inner.asp)” as the default in Pandas, and as such, we only have entries for users where there is also device information. We’ll redo this merge using a left join to keep all users, and then use a second left merge to finally to get the device manufacturers in the same dataframe.

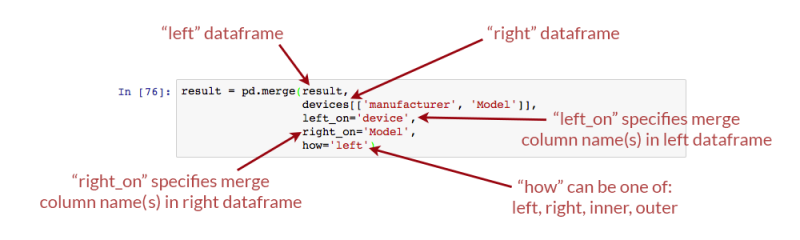
1. # First, add the platform and device to the user usage - use a left join this time.
2. result = pd.merge(user\_usage,
3. user\_device[['use\_id', 'platform', 'device']],
4. on='use\_id',
5. how='left')
6. # At this point, the platform and device columns are included
7. # in the result along with all columns from user\_usage
8. # Now, based on the "device" column in result, match the "Model" column in devices.
9. devices.rename(columns={"Retail Branding": "manufacturer"}, inplace=True)
10. result = pd.merge(result,
11. devices[['manufacturer', 'Model']],
12. left\_on='device',
13. right\_on='Model',
14. how='left')
15. **print**(result.head())

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-final-merge-result.png)Final merged result with device manufacturer information merged onto the user usage table. Two left merges were used to get to this point.

**Using left\_on and right\_on to merge with different column names**

The columns used in a merge operator do not need to be named the same in both the left and right dataframe. In the second merge above, note that the device ID is called “device” in the left dataframe, and called “Model” in the right dataframe.

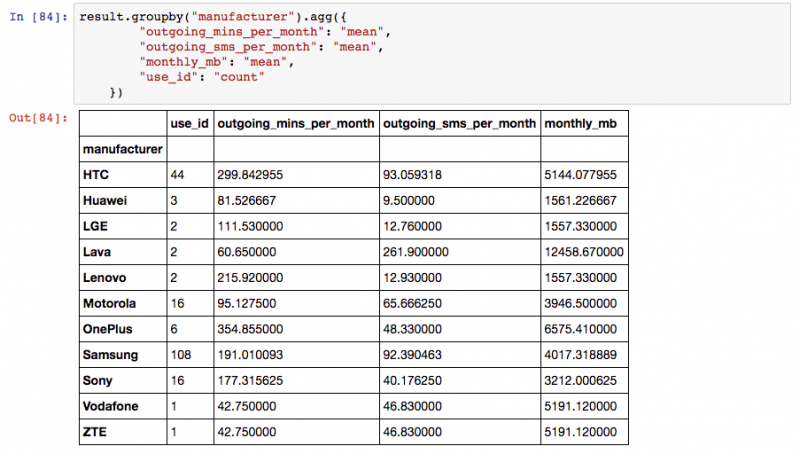
Different column names are specified for merges in Pandas using the “left\_on” and “right\_on” parameters, instead of using only the “on” parameter.

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-join-different-variable-names-copy-e1488722312527.png)Merging dataframes with different names for the joining variable is achieved using the left\_on and right\_on arguments to the pandas merge function.

**Calculating statistics based on device**

With our merges complete, we can use the [data aggregation functionality of Pandas](https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/) to quickly work out the mean usage for users based on device manufacturer. Note that the small sample size creates even smaller groups, so I wouldn’t attribute any statistical significance to these particular results!

1. result.groupby("manufacturer").agg({
2. "outgoing\_mins\_per\_month": "mean",
3. "outgoing\_sms\_per\_month": "mean",
4. "monthly\_mb": "mean",
5. "use\_id": "count"
6. })

[](https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2017/03/pandas-merge-statistics-result.png)Final result using agg() pandas aggregation to group by device manufacturer and work out mean statistics for different columns.

**Becoming a master of merging – Part 2**

That completes the first part of this merging tutorial. You should now have conquered the basics of merging, and be able to tackle your own merging and joining problems with the information above. Part 2 of this blog post addresses the following more advanced topics:

* How do you merge dataframes using multiple join /common columns?
* How do you merge dataframes based on the index of the dataframe?
* What is the difference between the merge and join fucntions in Pandas?
* How fast are merges in Python Pandas?

**Other useful resources**

Don’t let your merging mastery stop here. Try the following links for further explanations and information on the topic:

* [Join and Merge Dataframes](http://chrisalbon.com/python/pandas_join_merge_dataframe.html) examples by Chris Albon.
* [Merge, join, and concatenate](http://pandas.pydata.org/pandas-docs/stable/merging.html) official reference documentation from Pandas.
* [Combining DataFrames with Pandas](http://www.datacarpentry.org/python-ecology-lesson/04-merging-data/) on “Python for Ecologists” by DataCarpentry
* [YouTube tutorial on Joining and Merging Dataframes](https://www.youtube.com/watch?v=XMjSGGej9y8) by “sentdex”
* [High performance database joins with Pandas](http://wesmckinney.com/blog/high-performance-database-joins-with-pandas-dataframe-more-benchmarks/), a comparison of merge speeds  by Wes McKinney, creator of Pandas.
* [SQL Join tutorial](https://www.w3schools.com/sql/sql_join.asp) on W3Schools
* [SQL Join tutorial](http://www.dofactory.com/sql/join) on DoFactory
* [Comparing pandas merge with SQL](http://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html#join) in official Pandas documentation.