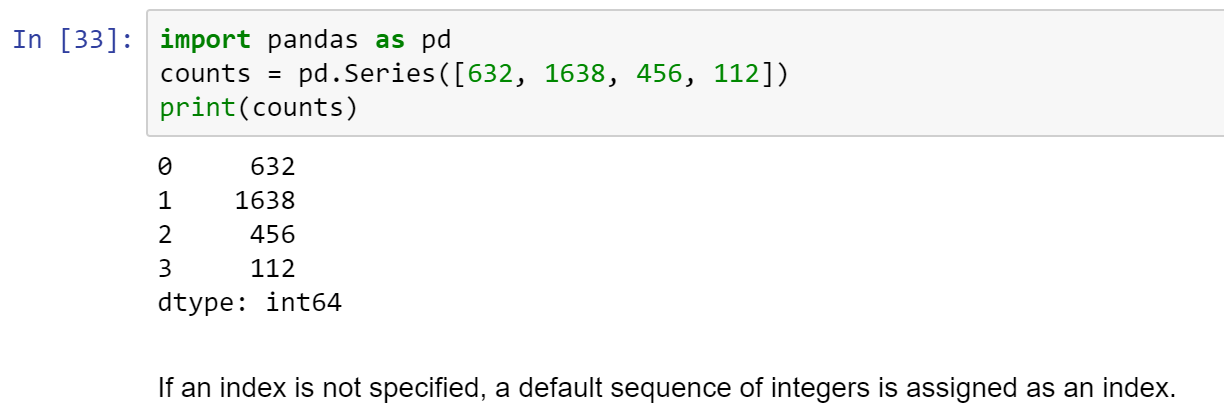
Let’s practice to learn more . . .

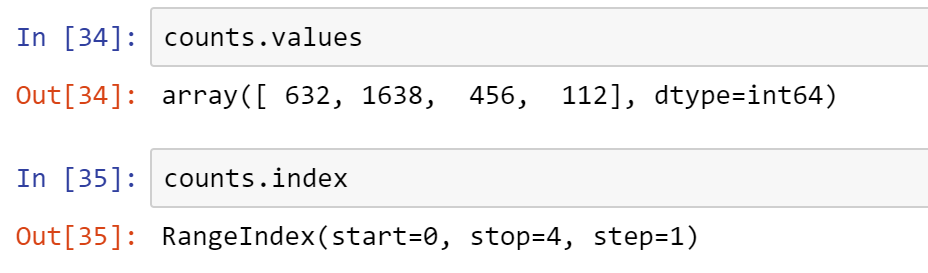
pandas is a Python package providing fast, flexible, and expressive data structures designed to work with relational or labeled data. It is well suited for:

* Tabular data with heterogeneously-typed columns, as you might find in SQL table or Excel spreadsheet
* Ordered and unordered time series data
* Arbitrary matrix data with row and column labels

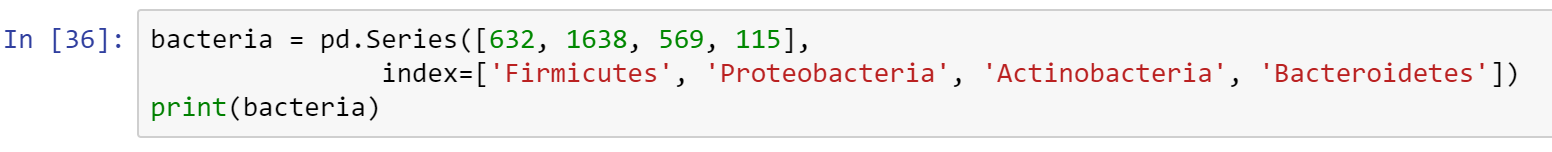
Virtually any statistical dataset, labeled or unlabeled, can be converted to a pandas data structure for clearing, transformation, and analysis.

A Series is a single vector of data with an index that labels each element in the vector.



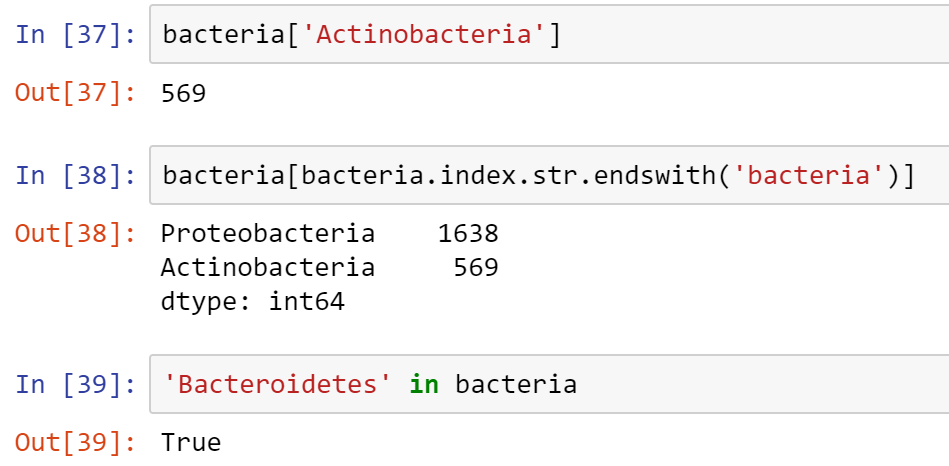


We can assign meaningful labels to the index, if they are available. These counts are of bacteria taxa constituting the microbiome of hospital patients, so using the taxon of each bacterium is a useful index.



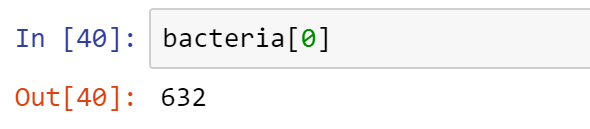


These labels can be used to refer to the values in the Series.

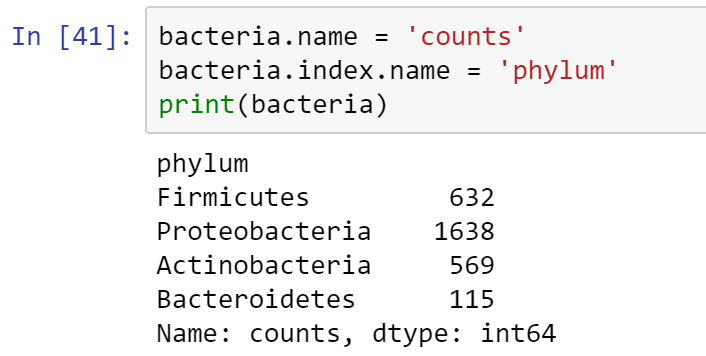


Notice that the indexing operation preserved the association between the values and the corresponding indices.

We can still use positional indexing if we wish.

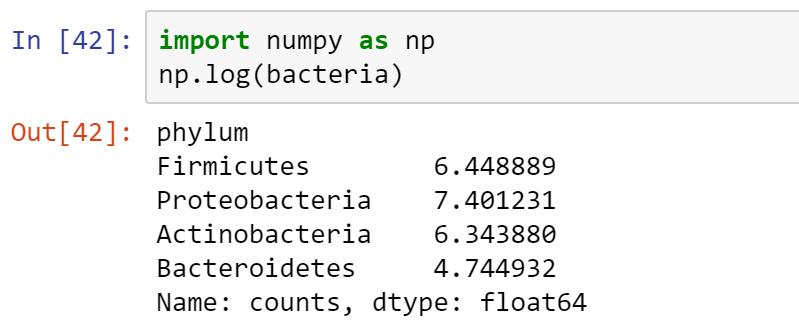


We can give both the array of values and the index meaningful labels themselves.

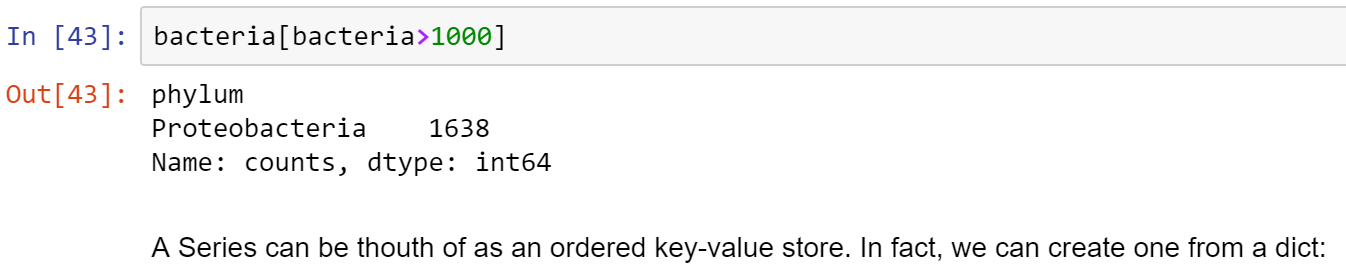


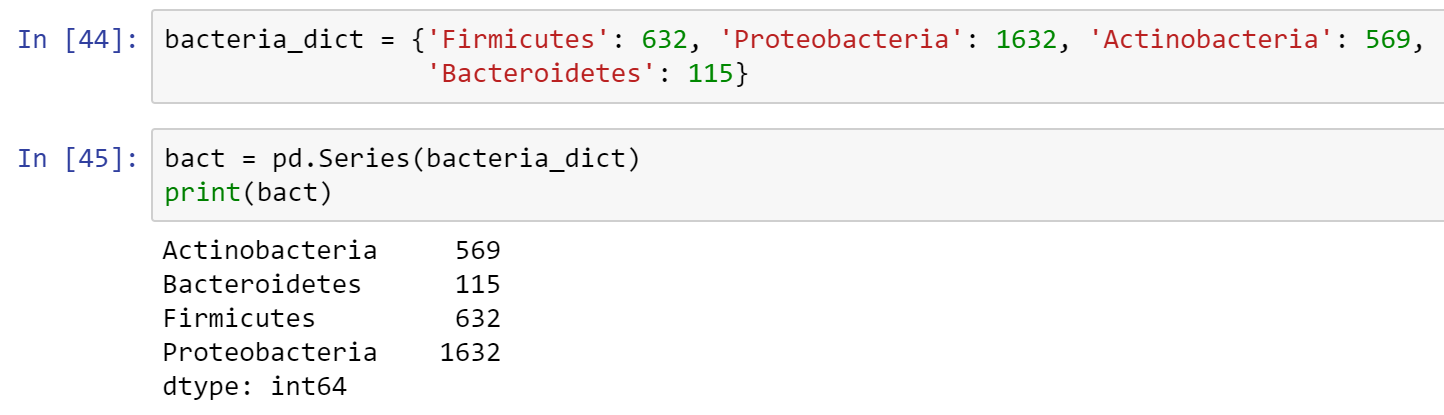
pandas and NumPy are tightly integrated

NumPy's math functions and other operations can be applied to Series without losing the data structure.



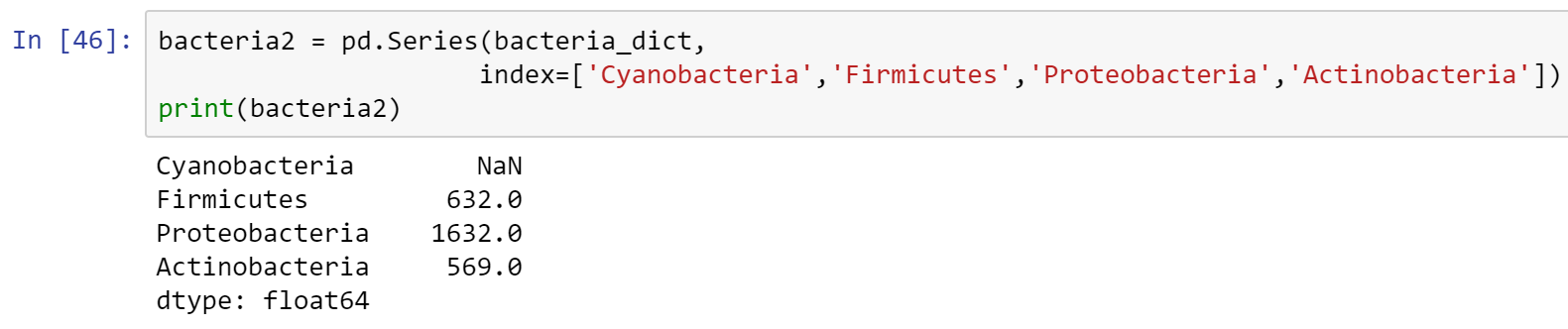
We can also filter according to the values in the Series:

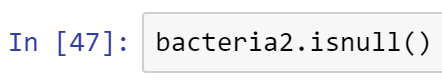


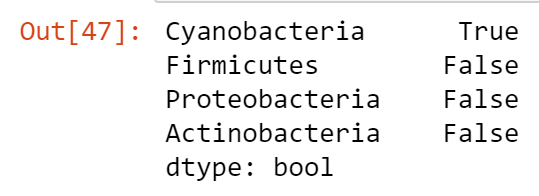


Notice that the Series is created in key-sorted order.

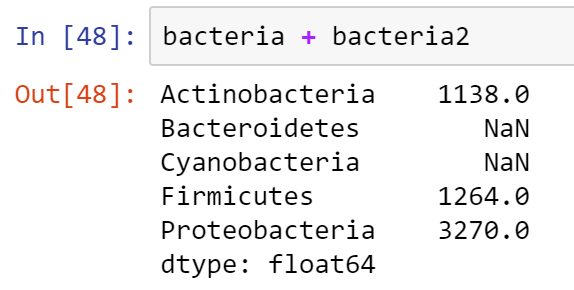
If we pass a custom index to Series, it will select the corresponding values from the dict, and treat indices without corresponding values as missing. pandas uses the NaN (not a number) type for missing values.





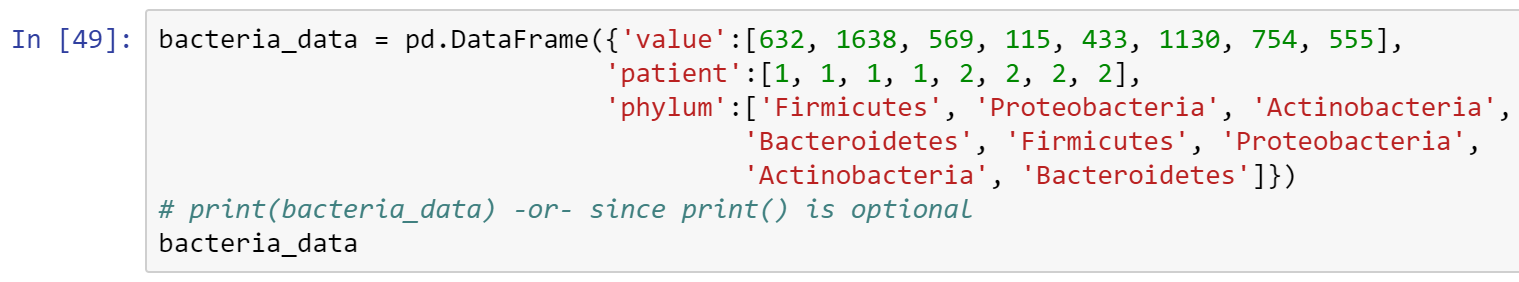


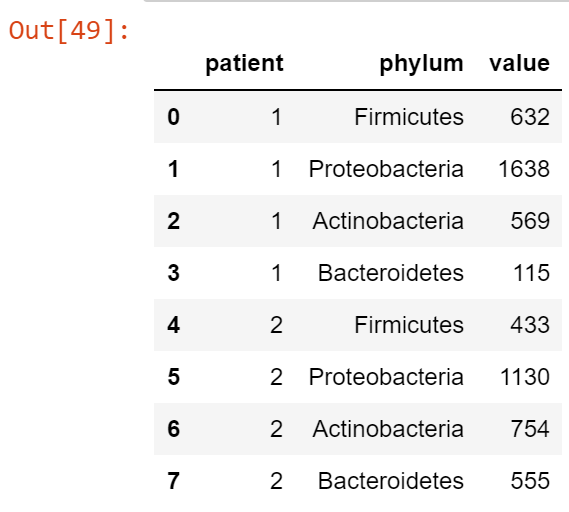
Critically, the labels are used to align data when used in operations with other Series objects:



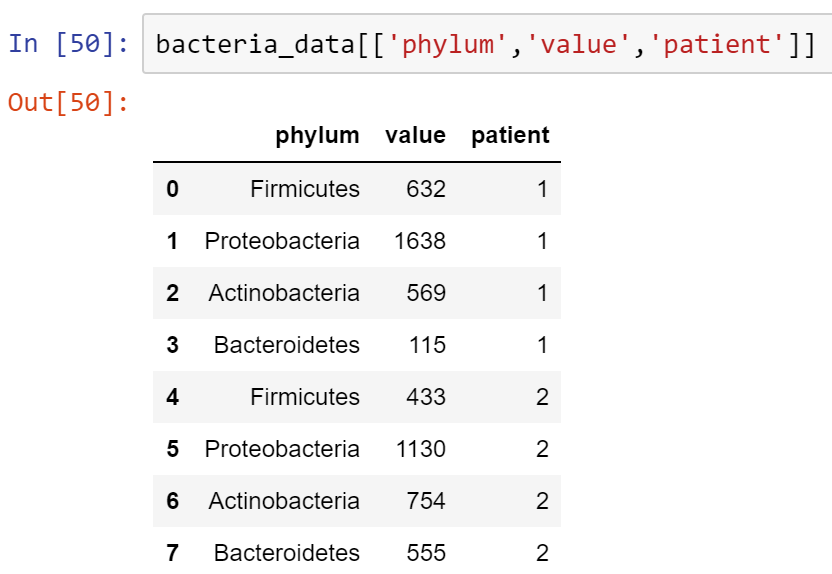
A DataFrame is a tabular data structure, encapsulating multiple series like columns in a spreadsheet. Data are stored internally as a 2-dimensional object, but the DataFrame allows us to represent and manipulate higher-dimensional data.

DataFrame is used to generalize the series.

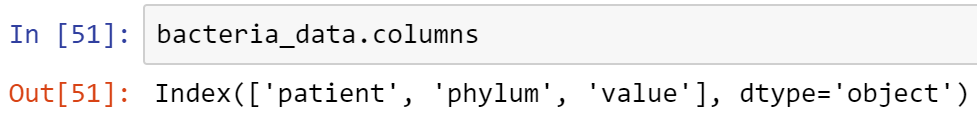




Notice the DataFrame is sorted by column name. We can change the order by indexing them in the order we desire.

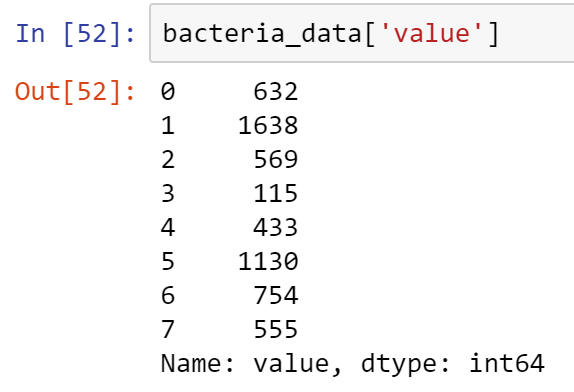


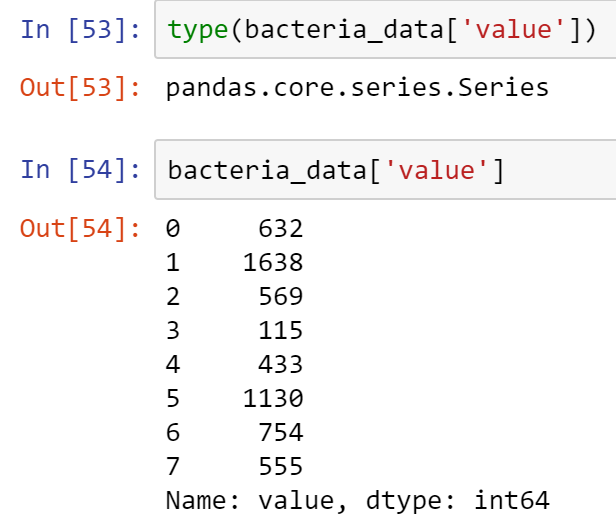
A DataFrame has a second index, representing the columns:

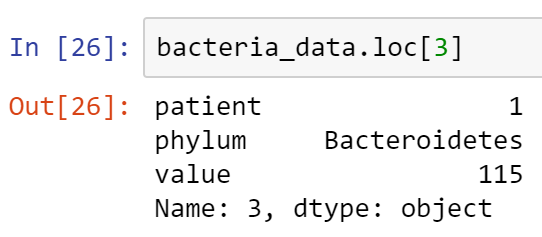


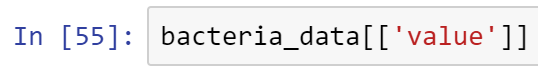
So, above are the indexes of the columns--that is the second index.

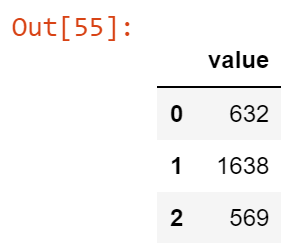
If we wish to access columns, we can do so either by dict-like indexing or by attribute:





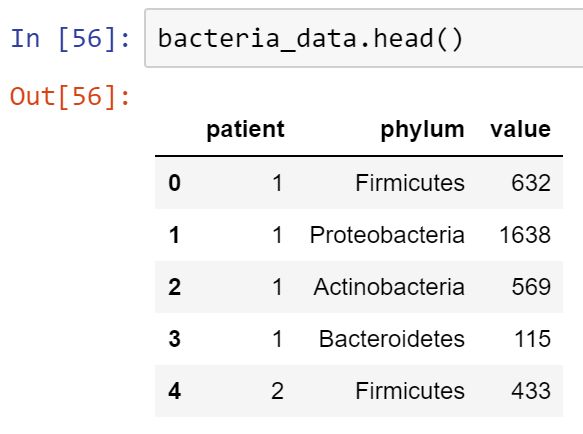


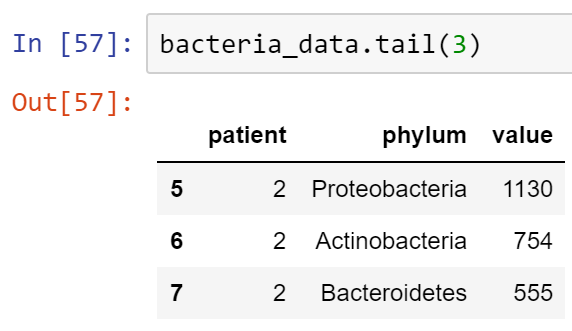


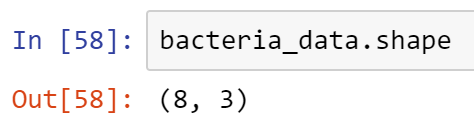




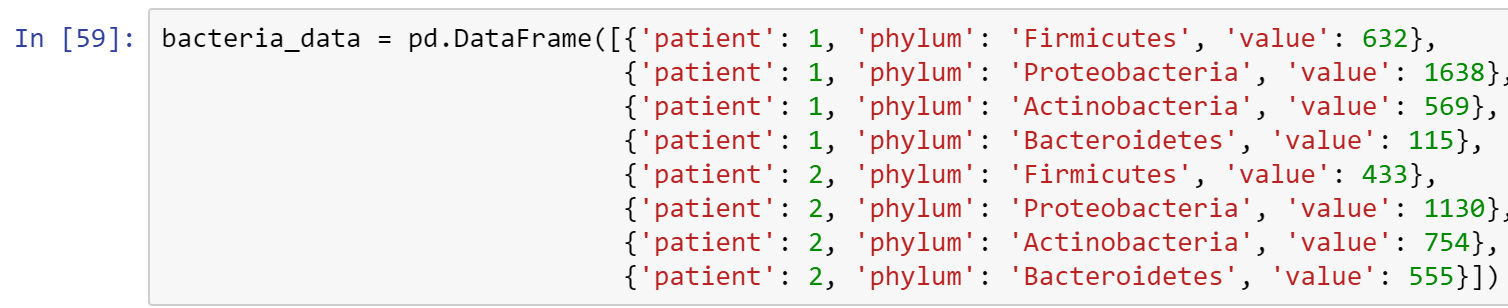
Since a row potentially contains different data types, the returned Series of values is of the generic object type.

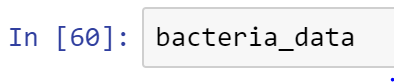


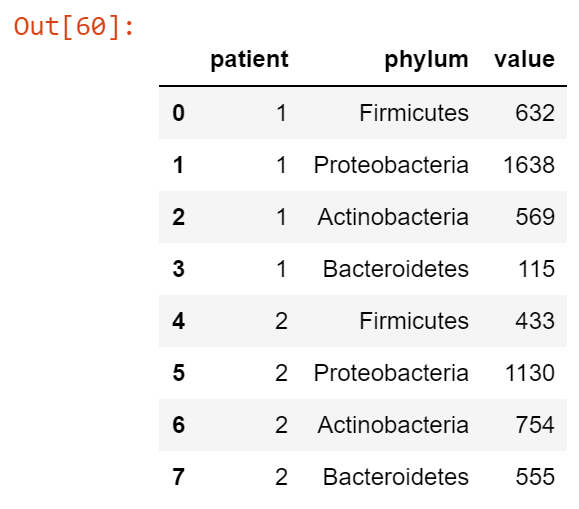




If we want to create a DataFrame row-wise rather than column-wise, we can do so with a dict of dicts.



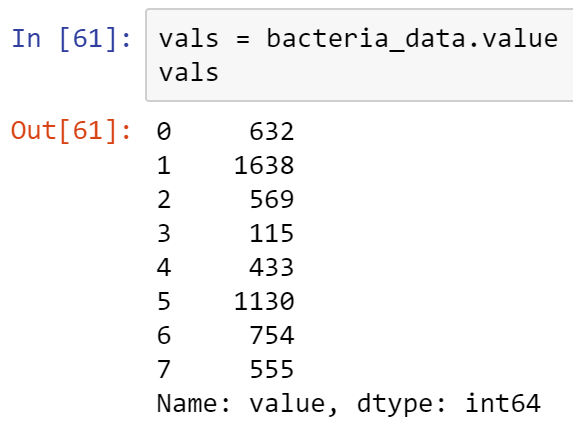




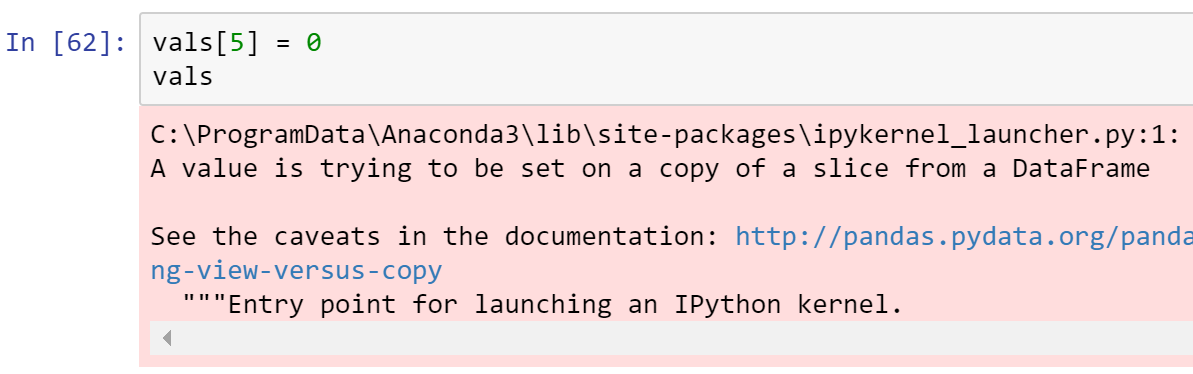
As you can see, there are lots of ways of creating DataFrame.

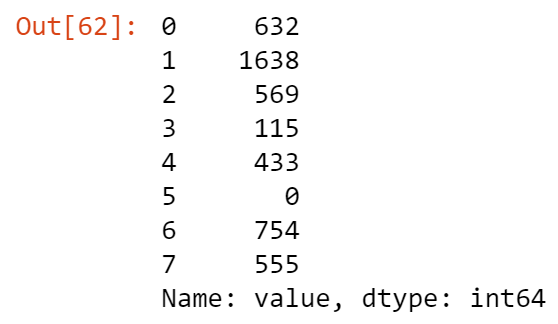
In panda, when indexing data, you are getting a view of that data, and not a copy of the data itself. So you must be cautious when manipulating this data.

For example, let's isolate a column of our dataset by assigning it as a Series to a variable.'

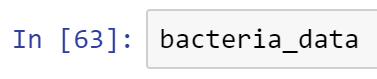


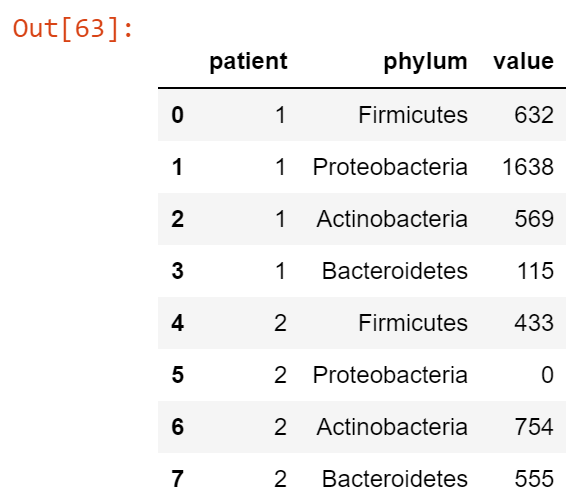
Now, let's assign a new value to one of the elements of the Series.





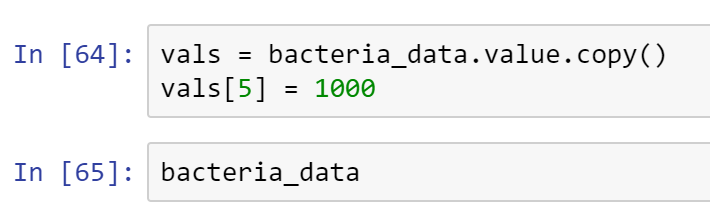
A warning is given that the changes to a "view" (a Series), will also be applied to the original DataFrame.

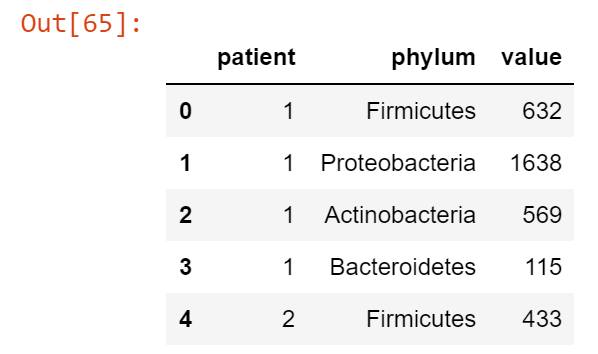


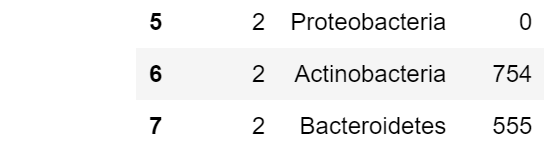


Note that row 5 in the original DataFrame above had been altered as well.

We can avoid this by working with a copy when modifying subsets of the original data.

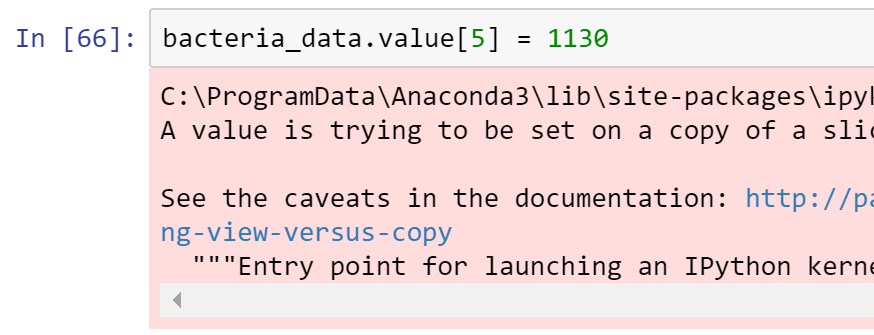




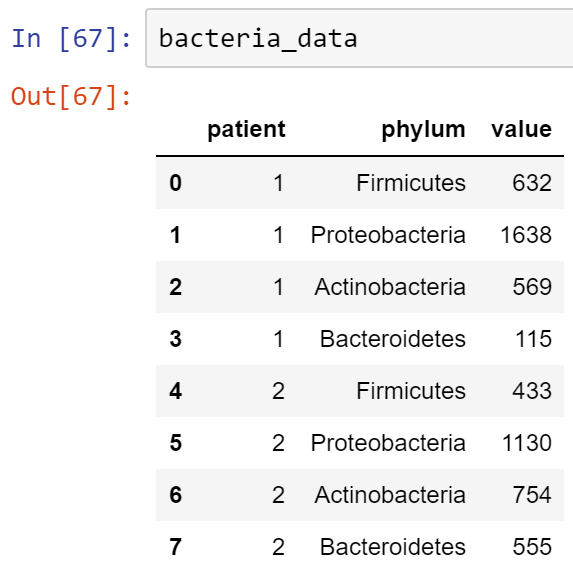


Note that changes to row 5 of the copy (a clone) did not affect the original DataFrame.

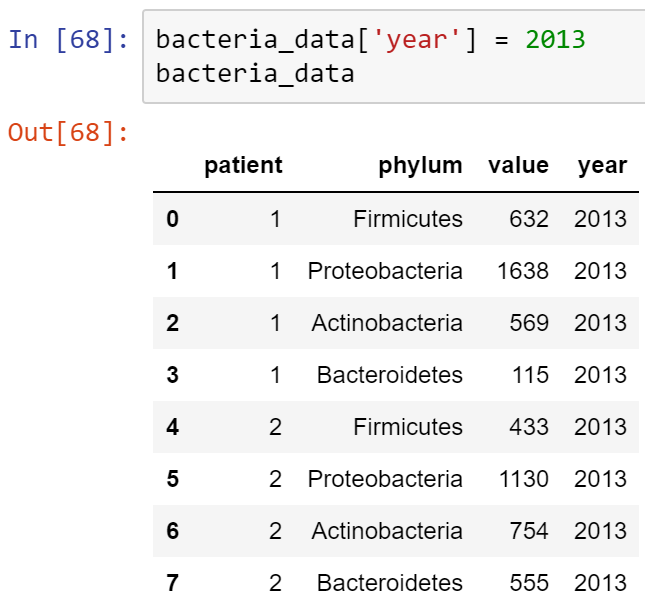
Now let's put back the value we accidentally changed.

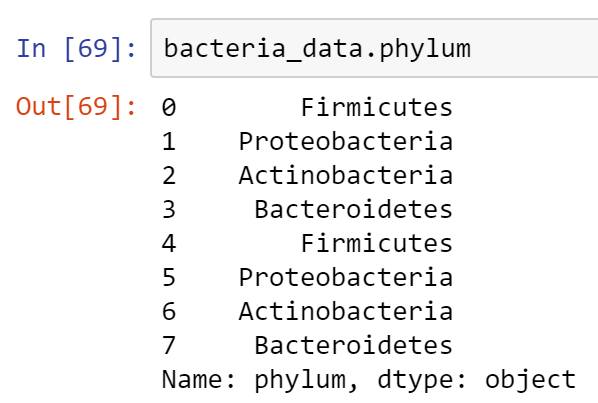


A warning is given, and checking the original DataFrame shows row 5 was restored.



You can also add a new column.

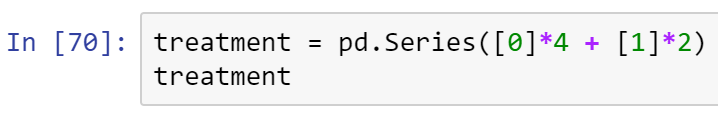


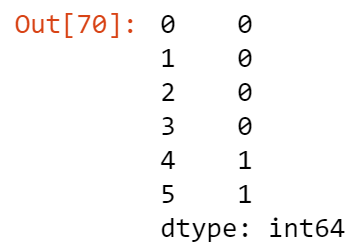


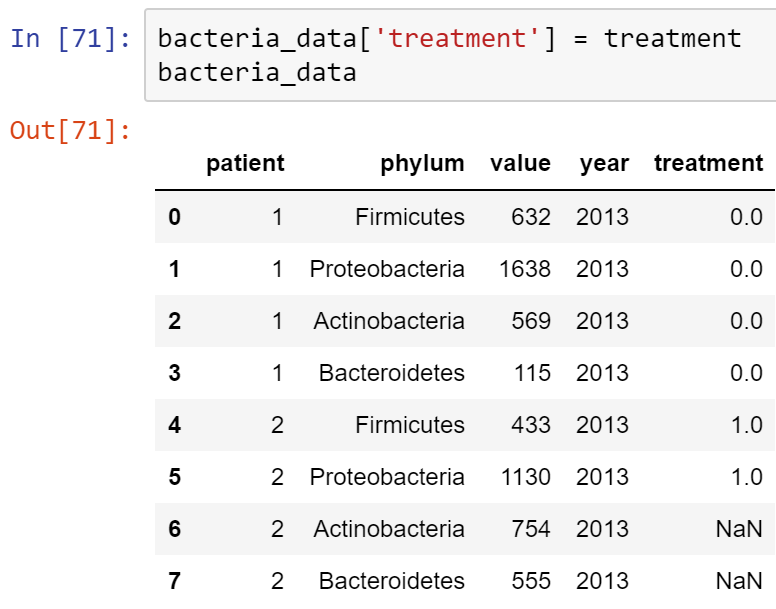
Above returns a column view.

Auto-alignment:

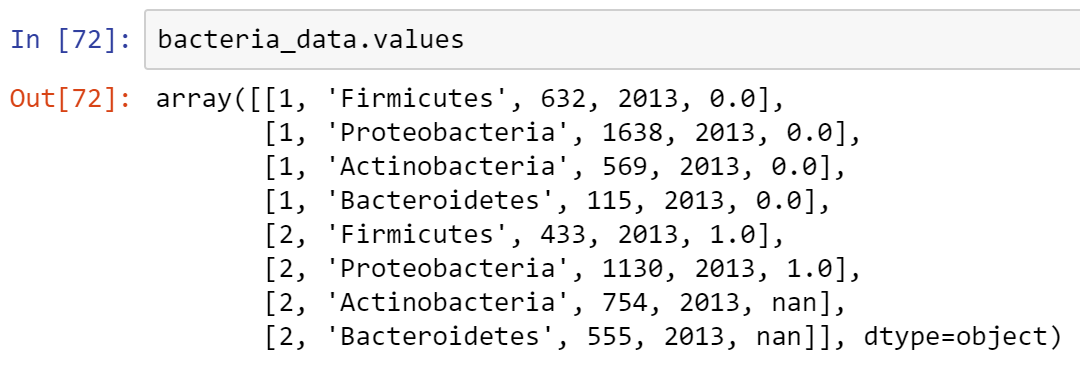
When adding a column that is not a simple constant, we need to be a bit more careful. Due to pandas' auto-alignment behavior, specifying a Series as a new column causes its values to be added according to the DataFrame's index.



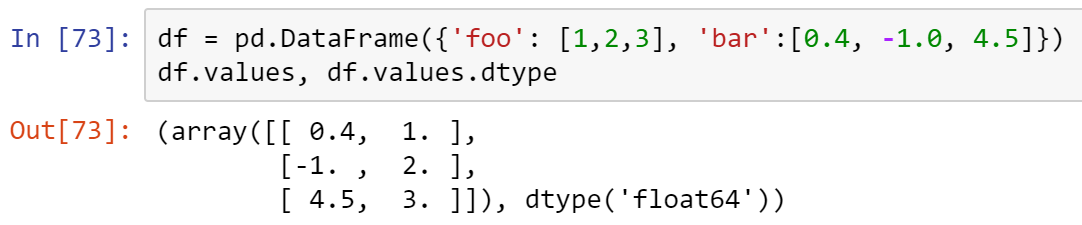




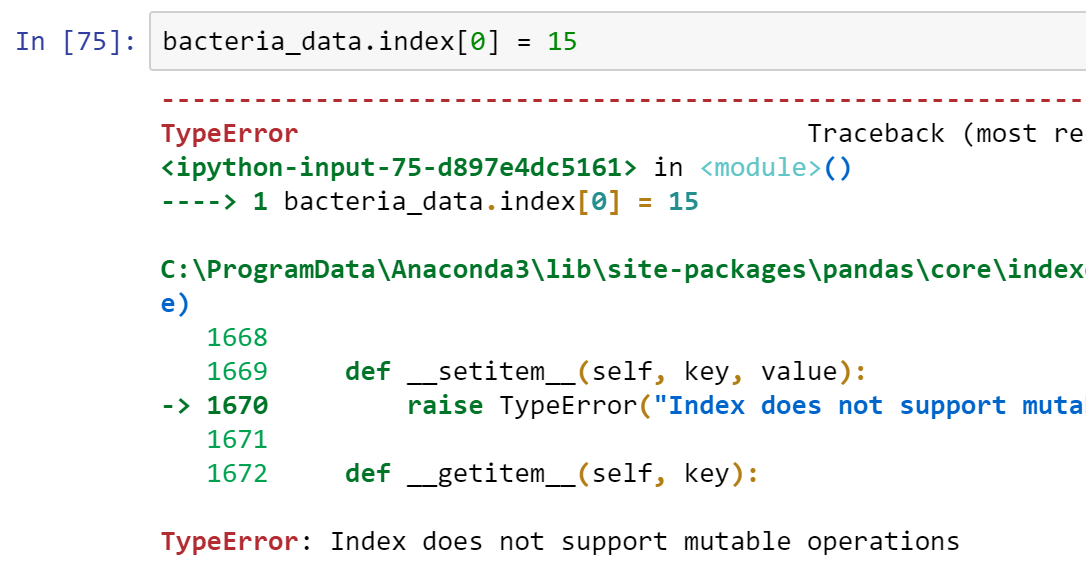
We can extract the underlying data as a simple ndarray by accessing the values attribute.

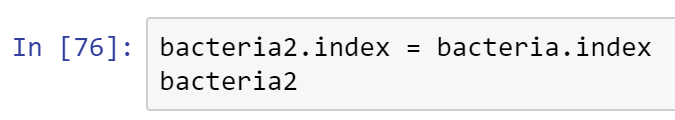


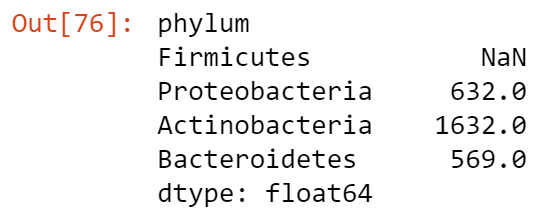
Notice that because of the mix of string, integer and float (and NaN) values, the dtyple of the array is object. The dtype will automatically be chosen to be as general as needed to accomodate all the columns.



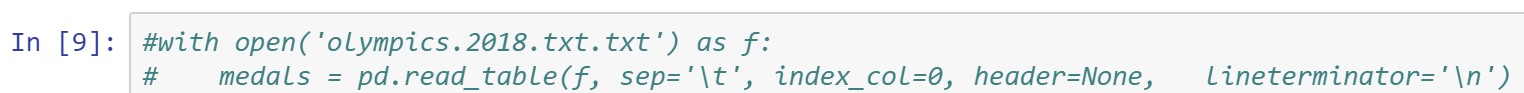
pandas indexes are immutable. For example, you can't just change index 0 to 15.

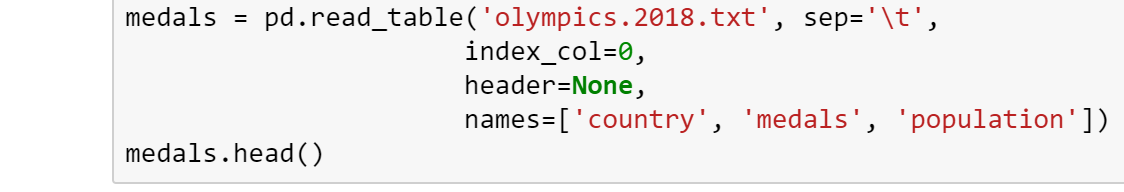


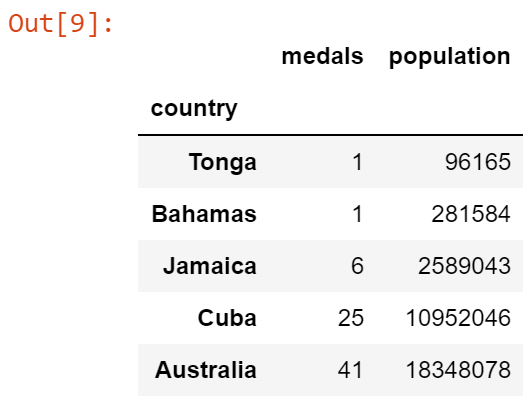




pandas makes importing data easy.







There is no header row in this dataset, so we specified this, and provided our own header names.

If we did not specify header=None the function would have assumed the first row contained column names.

The tab separator was passed to the (separator) sep agrument as \t.

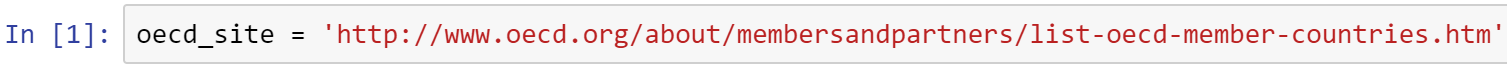
The sep argument can be customized as needed to accommodate arbitrary separators. For example, we can use a regular expression to define a variable amount of whitespace, which is unfortunately, common in some datasets:

sep = '\s+'

Scraping Data from the Web

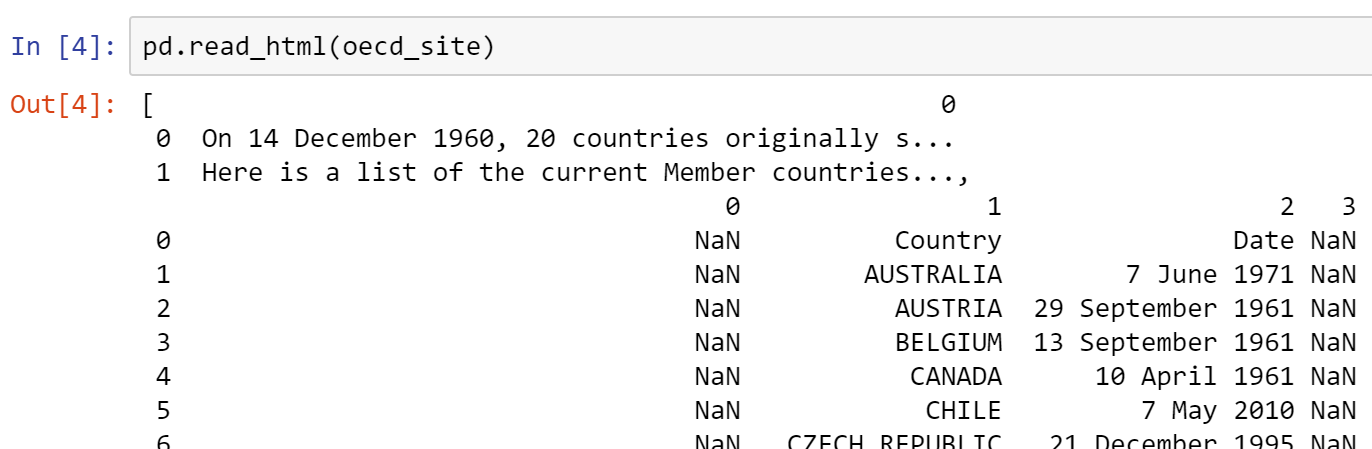
We would like to add another variable to this dataset. Along with population, a country's economic development may be a useful predictor of Olympic success. A very simple indicator of this might be OECD membership status.

The read\_html function accepts a URL argument, and will attempt to extract all the tables from that address, returning whatever it finds in a list of DataFrames.



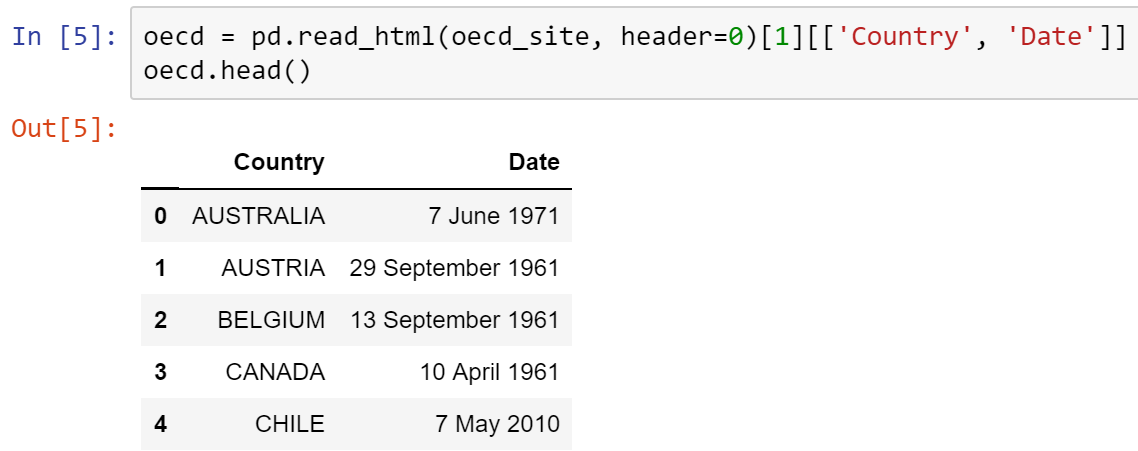
Make sure you check the URL that you typed above that it will render in a browser.

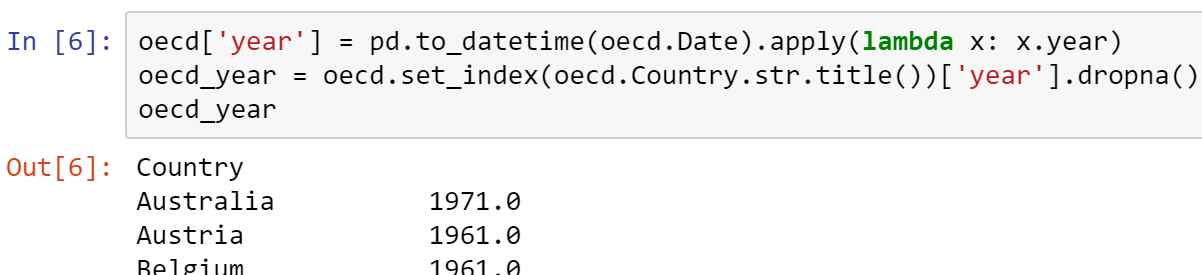
Then execute the line below, and wait for a few seconds for the result to return.



There is typically some cleanup that is required of the returned data, such as the assignment of column names and conversion of types.

The table of interest is at index 1 (Country), and we will extract two columns from the table. Otherwise, this table is pretty clean.





We can create an indicator (binary) variable for OECD status by checking if each country is in the index of countries with membership year less than 1997.

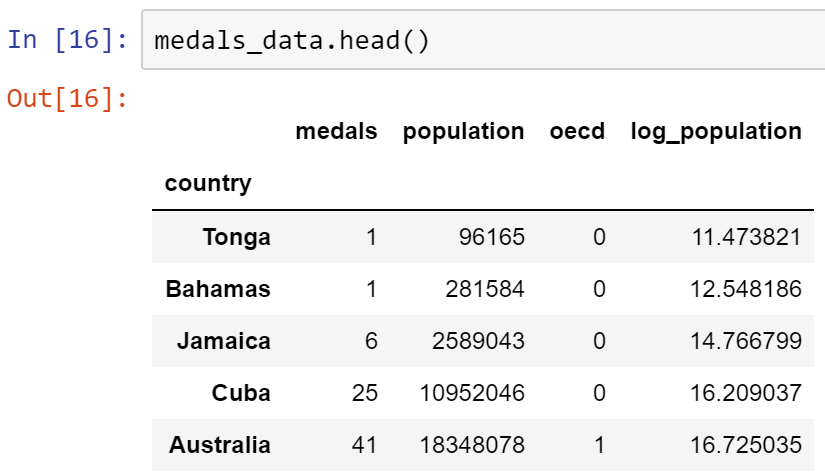
The new DataFrame method assign is a convenient means for creating the new column from this operation.



Since the distribution of populations spans several orders of magnitude, we may wish to use the logarithm of the population size, which may be created similarly.



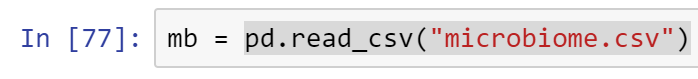
The NumPy log function will return a pandas Series(or DataFrame when applied to one) instead of a ndarray; all of NumPy's functions are compatible with pandas in this way.



Comma-separated Values (CSV)

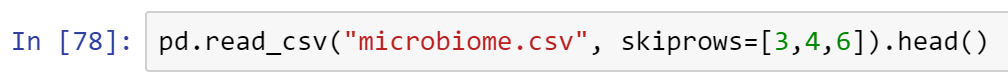
The most common form of delimited data. Since CSV is so ubiquitous, the read\_csv is available as a convenience function for read\_table.

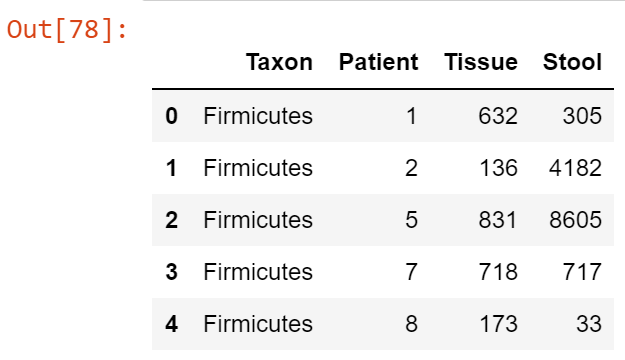
Consider some more microbiome data. Read into a DataFrame using read\_csv.



skiprows

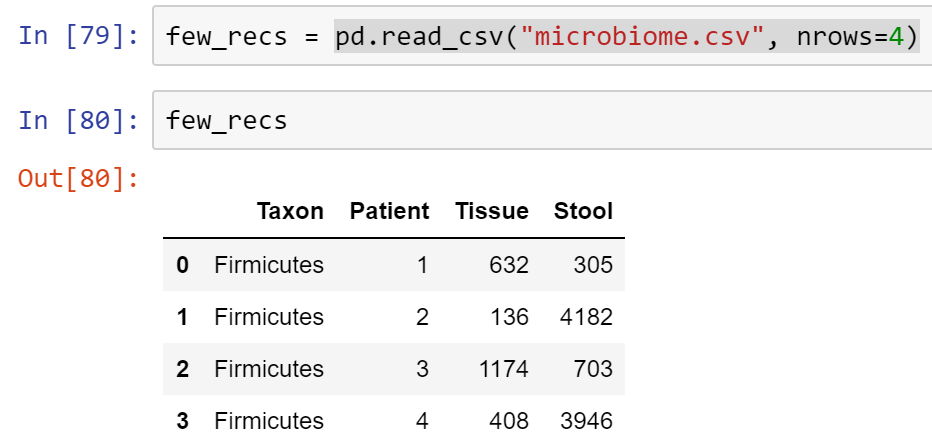
If we have sections of data that we do not wish to import (for example, known bad data), we can populate the skiprows argument:





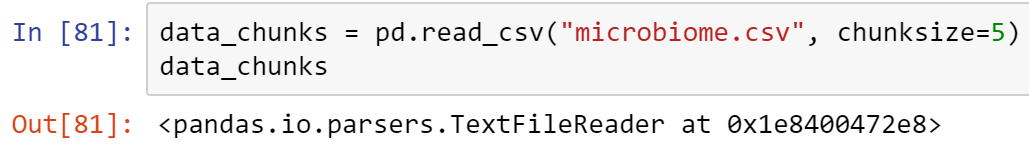
nrows

Conversely, if we only want to import a small number of rows from, say, a very large data file we can use nrows:



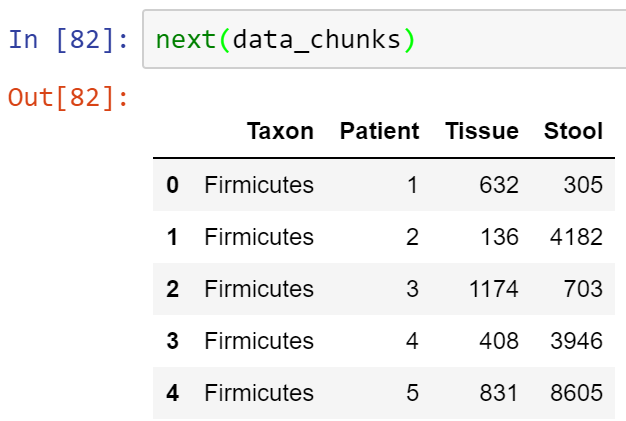
chunksize

Alternatively, if we want to process our data in reasonable chunks, the chunksize argument will return an iterable object that can be employed in a data processing loop. For example, our microbiome data are organized by bacterial phylum, with 15 patients represented in each.

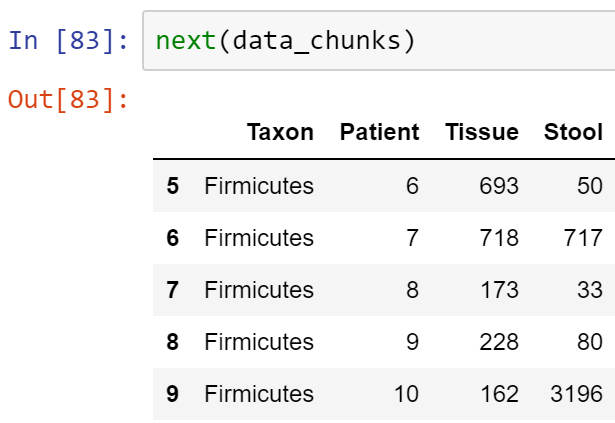


The weird object returned, TextFileReader, ia a python generator. It pass you one chunk at a time.

For example: to get the first 4.



Here's the next 4:

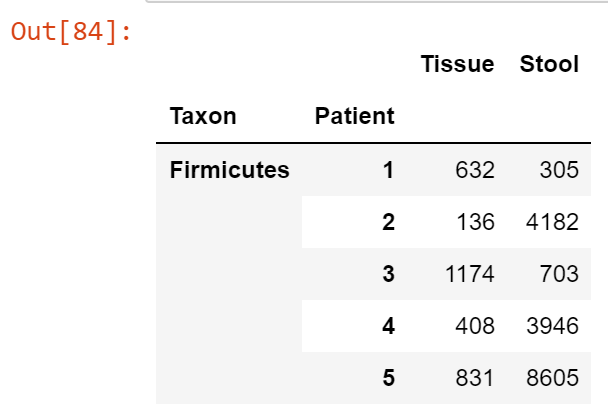


And so on, and so forth . . .

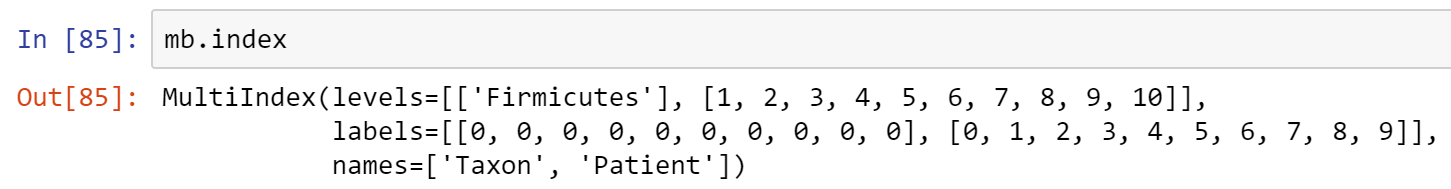
Hierarchical Indices

For a more useful index, we can specify the first two columns, which together provide a unique index to the data.



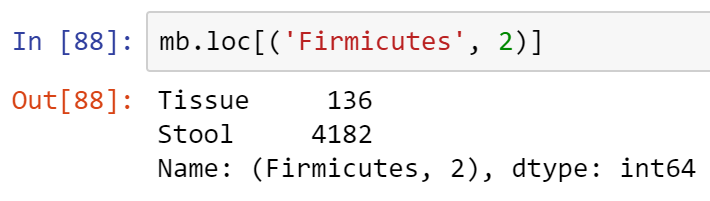


This is called a hierarchical index, which allows multiple dimensions of data to be represented in tabular form.

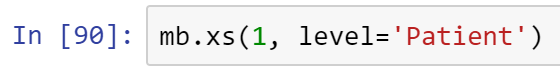


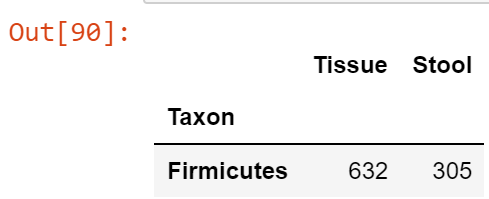
The corresponding index is a MultiIndex object that consists of a sequence of tuples, the elements of which is some combination of the three columns used to create the index. Where there are multiple repeated values, pandas does not print the repeats, making it easy to identify groups of values.

Rows can be indexed by passing the appropriate tuple.

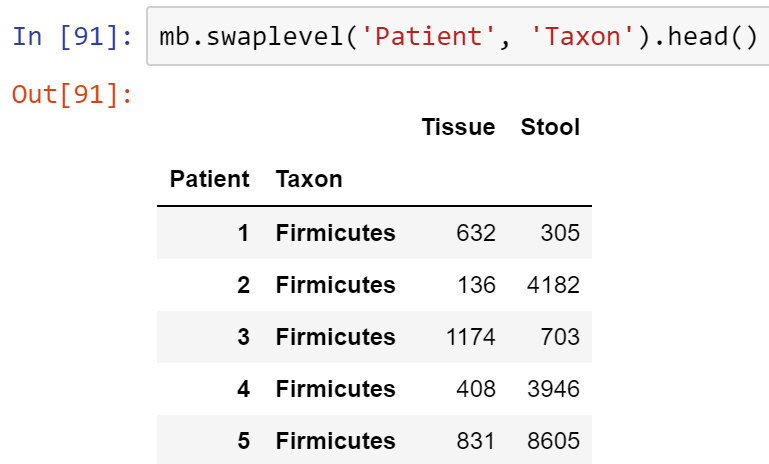


To extract arbitrary levels from a hierarchical row index, the cross-section method xs can be used.





We may also reorder levels as we like.



* Copy all your code into a Word doc, place your name on it, and submit in Canvas.