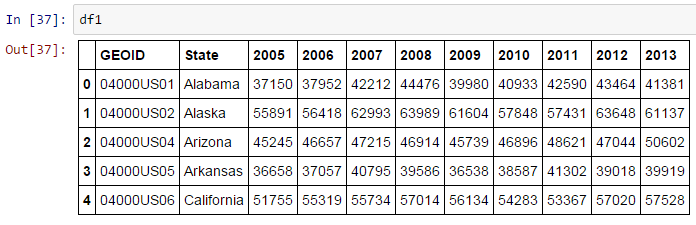
**Accessing pandas dataframe columns, rows, and cells**

At this point you know how to load CSV data in Python. In this lesson you will learn how to access rows, columns, cells and subsets of rows and columns from a pandas dataframe. Let’s open the CSV file again, but this time we will work smarter. We will not download the CSV from the web manually. We will let Python directly access the CSV download URL. Here is the code:

* import pandas as pd
* df1=pd.read\_csv("http://pythonhow.com/wp-content/uploads/2016/01/Income\_data.csv")

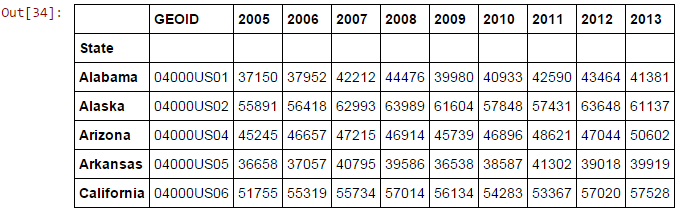
The dataframe will be identical to the dataframe we used in the previous lesson. Again, once you have the dataframe loaded in your iPython session, you can perform operations to your dataframe. Just for reference, here is how the complete dataframe looks like:



And before extracting data from the dataframe, it would be a good practice to assign a column with unique values as the index of the dataframe. The State column would be a good choice:

* df2=df1.set\_index("State")

As you see you needed to store the result in a new dataframe because this is not an in-place operation. The *df2* dataframe would look like this now:

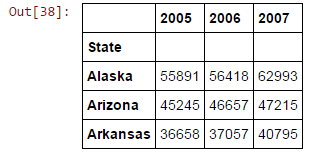


Now, let’s extract a subset of the dataframe. Here is the general syntax rule to subset portions of a dataframe:

* df2.loc[startrow:endrow,startcolumn:endcolumn]

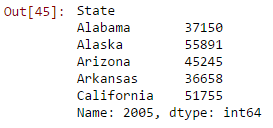
If you can’t wrap your mind around that, here is a nice example that extract the values for the rows from Alaska through Arkansas for years 2005 to 2007:

* df2.loc["Alaska":"Arkansas","2005":"2007"]



And here is how to slice a column:

* df2.loc[: , "2005"]



Note that when you extract a single row or column, you get a one-dimensional object as output. That is called a pandas Series. The values on the left are just labels taken from the dataframe index So, the formula to extract a column is still the same, but this time we didn’t pass any index name before and after the first colon. That tells Python to include all the rows. And we passed only one column name.

To extract only a row you would do the inverse:

* df2.loc["California", : ]

And for a single cell:

* df2.loc["California","2013"]

You could apply methods to the subsets:

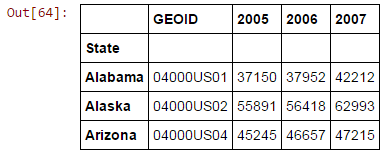
* df2.loc[:,"2005"].mean()

That for example would return the mean income value for year 2005 for all states of the dataframe.

**Position based indexing**

Now, sometimes, you don’t have row or column labels. In such case you will have to rely on position based indexing which is implemented with iloc instead of loc:

* df2.iloc[0:3,0:4]



Note that when we used label based indexing both the start and the end labels were included in the subset. With position based slicing, only the start index is included. So, in this case Alabama had an index of 0, Alaska 1, and Arizona 2. Same goes for the columns.

And one more thing you should now about indexing is that when you have labels for either the rows or the columns, and you want to slice a portion of the dataframe, you wouldn’t know whether to use loc or iloc. In this case, you would want to use ix:

* df2.ix[0:3,"2005":"2007"]

**Pandas data analysis functions**

You now know how to load CSV data into Python as pandas dataframes and you also know how to manipulate a dataframe. Let’s now see what data analysis methods we can apply to the pandas dataframes.

You know that the dataframe is the main pandas object. So, if you have a some data loaded in dataframe *df*, you could apply methods to analyze those data. For instance, here is how you apply the mean method to the dataframe we have been working on:

df.mean()  
  
And you would get:  
2005 45339.8  
2006 46680.6  
2007 49789.8  
2008 50395.8  
2009 47999.0  
2010 47709.4  
2011 48662.2  
2012 50038.8  
2013 50113.4  
dtype: float64

You can get a list of available DataFrame methods using the Python *dir* function:

dir(pd.DataFrame)

And you can get the description of each method using help:  
  
help(pd.DataFrame.mean)

You can also apply methods to columns of the dataframe:  
  
df2.loc[:,"2005"].mean()

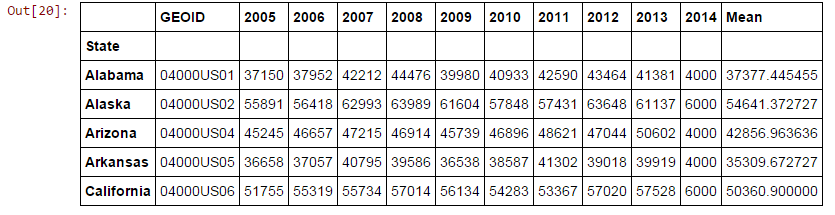
Note though that in this case you are not applying the mean method to a pandas dataframe, but to a pandas series object:  
  
type(d2.loc[:,"2005"])

So, checking the type of the object would give the type of the object:

pandas.core.series.Series  
  
And again you can pass the Series object to the dir method to get a list of available methods.

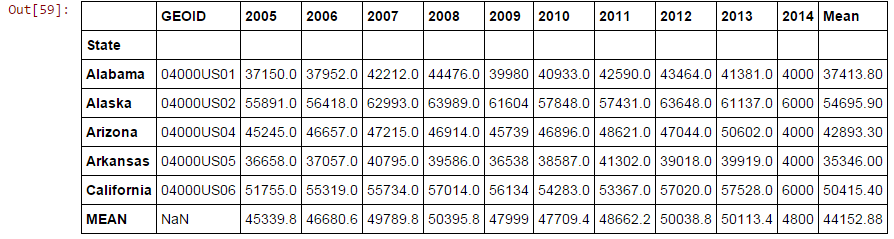
Adding columns to a DataFrame is quite straightforward:

df2["2014"]=[4000,6000,4000,4000,6000]   
That would add a new column with label “2014” and the values of the Python list. You can also add a column containing the average income for each state:

df2["Mean"]=df2.mean(axis=1)  
  
And you would get this:  
  
The axis parameter tells Python to compute the mean along axis 1 which means along the columns. Axis set to 0 would go along the rows. Let’s see how to calculate the mean for each year and add them as a new row:

df2.loc["MEAN"]=df2.mean(axis=0)

This would add a new row with index “MEAN”:



Instead of throwing an error, pandas generated a NaN datatype for the GEOID column which is a good thing because operations won’t break when the dataframe has non-numeric values.