Moving from datascience to Pandas

Given the sizable number of questions about differences between the datascience library and Pandas, I wanted to put together some thoughts and reference material on the subject.

What is the datascience library?

Written here at Berkeley specifically for the Data 8 curriculum, datascience is a library used to load/manage/manipulate/analyze data. As some of you have started to see, it is *very* similar to Pandas. That's because it essentially is - the point of the library was to take similar concepts and abstractions used in more mainstream libraries like Pandas and make them more accessible and intuitive (particularly for students who are newer to programming). It's a useful pedagogical tool and one of the really cool productions of Berkeley's data science program.

What is Pandas and why have we switched?

Pandas is the wider standard for data analysis tools in Python. It's extensively used in industry and academia and has a robust body of documentation. It also has the same central idea of the datascience library: building a table and operating on that.

The previous iteration of this class relied on the datascience library but it quickly turned out that some of the more involved analyses of social science/legal datasets we wanted to do were difficult to execute without resorting to Pandas. Given that and the fact that the world outside of Berkeley is largely unfamiliar with the datascience library, it made sense to switch LS123 entirely to Pandas. There may be a slight initial learning curve but the long-term dividends are significant, particularly for those of you who may want to go on to data science after this course.

What is the difference between datascience and Pandas?

In a practical sense, there's not actually a huge difference. As I mentioned above they both are built on the same data model (tables) and operations. The most significant change is the syntax (the code you type to make operations happen).

For example, here is how you might generate the following table in both datascience and Pandas

State	Senators	Representatives
NY	2	27
PA	2	18

WA	2	10

datascience

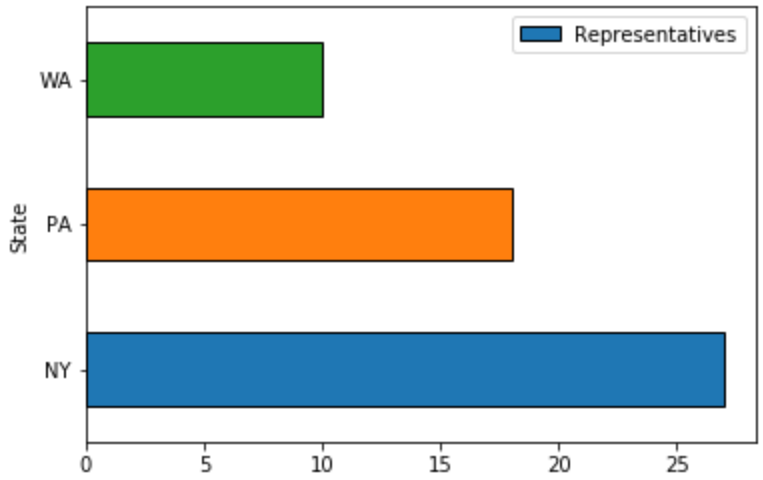
```
from datascience import *
states = Table().with_columns([
    'State', ['NY', 'PA', 'WA'],
    'Senators',  [2,   2,   2],
    'Representatives', [27,   18,
10],
])
```

Pandas

```
import pandas as pd
states = pd.DataFrame({'State':
['NY', 'PA', 'WA'],
    'Senators':  [2,   2,   2],
    'Representatives': [27,   18,
10]})
```

As you can see, the format of the two pieces of code are fairly similar, with minor differences in the actual code typed.

Turning our attention to visualization, if we wanted to generate the following bar graph of states versus representative count, the code is again very similar.



datascience

```
states.barh('State',
'Representatives')
```

Pandas

```
states.plot.barh(x='State',
y='Representatives')
```

It turns out this is the general case for understanding how datascience commands translate to Pandas: the structure and arguments for the commands are mostly similar with minor differences in the command names (e.g. plot.barh vs barh)

How do I find the Pandas equivalent of datascience commands?

While hopefully helpful, the examples above are only two of a broad range of commands you will encounter in this class. It may be more useful to think about how do you take command X in datascience and find the command Y equivalent in Pandas.
The answer ends up being in the documentation for Pandas here which provides clear and explicit guidance on how to do pretty much everything with the library. Admittedly it is pretty dense so we work through an example below. Let's say we want to relabel the columns on a table. In datascience, we would do it as such

```
states.relabeled(['State', 'Senators', 'Representatives'], ['A', 'B',
'C'])
```

Now go to the Pandas documentation and either look for the function in the list or use the search feature. We can quickly find the documentation for Pandas rename here.

`DataFrame.`**`rename`**(*mapper=None*, *index=None*, *columns=None*, *axis=None*, *copy=True*, *inplace=False*, *level=None*) [source]

Alter axes labels.

Function / dict values must be unique (1-to-1). Labels not contained in a dict / Series will be left as-is. Extra labels listed don't throw an error.

See the user guide for more.

mapper, index, columns : *dict-like or function, optional*
dict-like or functions transformations to apply to that axis' values. Use either `mapper` and `axis` to specify the axis to target with `mapper`, or `index` and `columns`.

This view tells us what the arguments of the Pandas function are. The important insight is that the function wants a 'dict-like' object. This means that we need to provide it a dictionary mapping old column names to new columns as opposed to simply the two columns that datascience requests. Combining this with the example provided in the documentation we find that the Pandas syntax we want is

```
states.rename(index=str, columns={'State': 'A', 'Senators': 'B',
'Representatives': 'C'})
```

This approach of consulting the documentation and figuring out the format of the arguments is a pretty universal way to move from datascience to Pandas. That being said, if you want a quick guide to the syntax for some commands Pandas has published a really useful cheatsheet linked here. It provides a short summary of common commands and the intuition for what they're doing to the tables.

Resources

Pandas documentation
- The authoritative reference on Pandas commands, what they do, and how to call them

10 Minutes to Pandas
- An abbreviated guide that covers some common topics, takes slightly longer than 10 minutes to get through but definitely worth the time

Official Pandas Cheat Sheet
- A single-page visualization of useful day-to-day operations as well as intuition for what they're doing

Dataquest Cheat Sheet
- Another one-pager that has a list of frequent commands and a quick explanation