Data Bootcamp: Class #3

Revised: November 9, 2014

1 Course overview

- Objective: Learn enough about Python to do useful things with data.
- Target audience: Programming newbies. Anyone can do this with a little persistence and the help of friends.
- Trigger warning: This will take some effort. But it's worth it.
- **Work habits:**
  - Advice. This will seem mysterious at first, but if you stick with it, you'll find it starts to look familiar, even make sense.
  - Practice. Any time you have something to do with data, try it out in Python. Play around. Have fun!
  - Friends. Work with friends, and make new friends who know how to code.
  - Help. If you get stuck, ask for help — from friends, from the Bootcamp group (post a problem), Stack Overflow, etc.
- Team: Dave Backus, Glenn Okun, Sarah Beckett-Hile, and a rotating group of ninjas.
- Resources:
  - Bootcamp Group: https://groups.google.com/forum/#!forum/nyu_data_bootcamp. Post comments and questions here.
  - GitHub repository: https://github.com/DaveBackus/Data_Bootcamp. All the docs and programs are here. This document is in the Notes folder.
  - This document: The online version comes with links.

2 Overall plan

- First class: Python basics, examples.
- Today's class: graphics.
- Next (and last class) in two weeks: data management, SQL
- **It's possible we'll run a more formal course in January**

3 Today's plan

- Theme: graphics.
- Standard "scientific" packages.
- Reminders.
- Graphics with Matplotlib: two routes, one destination.
- Exercise review (Sarah).

4 Standard packages

Python is not just a program, it's an open source collection of tools that includes both basic Python and a collection of packages written by different people. The good news is that you can often find a package to do whatever it is you want to do. The bad news is that there's not necessarily any uniformity or quality control. The standard packages — which we'll talk more about in a minute — are well written, well documented, and have armies of users who spot and correct problems. Some of the others less so. Our suggestion is to stick to the mainstream packages, such as those in the Anaconda distribution, until you're reasonably comfortable with Python.

Some of the leading mainstream packages for numerical ("scientific") computation — all of which are in the Anaconda — are

- NumPy. Tools for numerical computing, including linear algebra. Compare this to Excel, where the basic unit is a cell, a single number. In NumPy the basic unit is a vector (a column) or matrix (a table). In this respect, it's a direct competitor to Matlab. We won't see much of NumPy here, but it's the foundation for Pandas.
- Matplotlib. The leading graphics library for Python and our focus today.
- Pandas. Tools for managing and manipulating data. The basic unit is a DataFrame, which is a table of data with labels for observations and variables. If you've used R, you'll be familiar with the idea. If not, you'll be familiar with it in a couple weeks. We'll touch on this today and spend a fair amount of time with it at our next meeting.

We love the term "wrapper," it makes it sound like we know what we're doing. It's an informal programming term that means a simplified interface for more basic programming functions. Cool, eh? Here are some popular wrappers for Matplotlib:

- Pyplot. This has two interfaces for Matplotlib, a simple one similar to Matlab and a more complicated object-oriented one (which means we'll use objects and methods). We'll describe both.
- Seaborn. Another interface for Matplotlib. Not part of the Anaconda distribution, but it seems to us to have some potential. Take a look at the examples, see what you think.

- PyLab. A combination of (basically) NumPy and Pyplot designed to replace Matlab. Convenient if you're familiar with Matlab, probably not otherwise. The Matplotlib FAQ adds: "it is no longer recommended."

We'll use Pyplot today. And Sarah may use Seaborn when she goes through last week's exercise.

5 Reminders

- Environment: Python 3.4 in Spyder.
- Assignments, strings, lists, slicing:

```
x, y, z = 2*3, 2**3, 2/3    # yes, three at once!
a, b = 'some', 'thing'
c = a + b

numbers = [x, y, z]
strings = [a, b, c]
all = numbers + strings
print(all)
print([type(all), all[:3], all[3:])
```

- Objects and methods. Most of the things we've seen are objects: `x, a, all`, etc. They have methods defined for them – useful things you can do with them. For more on the methods available for the object `all`, go to the IPython console and type `all.[tab]`. We call this "tab completion." For more on a specific method called `foo`, go to the Object explorer and type `all.foo`.
- Reading csv files. This was part of last week's assignment:

```
import pandas as pd
url1 = 'https://raw.githubusercontent.com/fivethirtyeight/data/master/'
url2 = 'college-majors/recent-grads.csv'
df = pd.read_csv(url1+url2)
```

What is `df`? What's in it?

```
properties = [type(df), df.columns, df.index, df.shape, df.head()]
for property in properties :
    print(property, end='\n\n')
```

Use tab completion to find more.

- **Exercises.**
  - How do we find the first item in the list `all`? The last?
  - What is `len(all)`?
  - How do we find the third column name in `df`? What is it?

6 Graphics overview

Graphics are horrendously complicated. Programs like Excel do their best to hide this fact, but if you ever try to customize a chart it quickly rears its ugly head. A graph might include, for example: the type (line, bar, etc); the color and thickness of lines, bars, or markers; title and axis labels; their location, fonts, and font sizes; tick marks (location, size); background color; and so on.

Matplotlib gives you control over all of these things and more — but it's complicated, there are lots of moving parts. The set of defaults is collected in what they call the `matplotlibrc` file, and it's enormous. That's why people have come up with simpler interfaces, Pyplot and Seaborn among them.

Bottom line: Graphics are complicated, but that's the nature of the beast, not something specific to Python. What Python gives you is total control. As always, you should start simple, work up to more complicated things gradually.

7 Pyplot's simple interface

- Overview. Pyplot is a module of Matplotlib. Here we'll use its simple interface. It looks like Matlab, but if that means nothing to you forget we said it. Either way, it's a good place to start.
- References:
  - Matplotlib's Pyplot tutorial. (This is excellent, check it out.)
  - Matplotlib's gallery of examples. Graphs along with the code that produced them, a great place to start.
  - Sargent and Stachurski's "Quantitative Economics," the section on Matplotlib. Good, but a little terse for our taste.
- Line plots. Here we take 11 years of annual US GDP data and plot it against time. Also consumption ("personal consumption expenditures"), so that we can show how to add a second line to a graph. We put all of these things, plus time, in lists and go to work:

```
import matplotlib.pyplot as plt     # import pyplot module

# data
gdp  = [13271.1, 13773.5, 14234.2, 14613.8, 14873.7, 14830.4, 14418.7,
        14783.8, 15020.6, 15369.2, 15710.3]
pce  = [8867.6, 9208.2, 9531.8, 9821.7, 10041.6, 10007.2, 9847.0, 10036.3,
        10263.5, 10449.7, 10699.7]
year = [2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013]

# plot
plt.plot(year, gdp)         #  plot(x,y)
```

We can do lots of things to dress up the plot. Here are some examples, but find your own with `plt.[tab]`:

```
plt.plot(year, pce, color='magenta', linewidth=2)  # add another line
plt.title('US Real GDP and Consumption')    # title
plt.ylabel('Billions of 2009 USD')          # y axis label
plt.legend(('GDP', 'Consumption'), loc=0)   # legend
plt.text(2010, 14000, 'GDP')                # add text at specific location
plt.text(2010, 9200, 'Consumption')         # same again
plt.ylim((0, 16000))                        # change y axis limits
plt.tick_params(length=6, width=2, colors='red')    # yikes!
plt.savefig('bootcamp_test.pdf')            # export as (in this case) pdf
plt.show()                                  # close plot
```

- **Exercise.** Run the basic plot GDP on your computer and
  - Change the color of the GDP line to green.
  - Add a marker.
  - Left-justify the title.
  - Extra credit. Change the title's font size to 20.
  - More extra credit. What does the `plot` argument `alpha=0.5` do?
- Bar charts. We'll work with two examples here. The first is to express our line plot of US GDP as a bar chart. We do that with the command:

```
plt.bar(year, gdp)                          # bar(x,y)
```

This isn't the most attractive bar chart in the world, but it works. Here's an example that looks better:

```
plt.bar(year, gdp, width=0.9, color='blue', align='center', alpha=0.7)
plt.show()
```

Here's another example, a standard chart in the MBA Global Economy class. The difference here is that one axis is a country, which is not numerical data. (There's a similar example in the Matplotlib gallery.) Here's the data:

```
codes     = ['USA', 'FRA', 'JPN', 'CHN', 'IND', 'BRA', 'MEX']
countries = ['United States', 'France', 'Japan', 'China', 'India',
             'Brazil', 'Mexico']
gdppc     = [53.1, 36.9, 36.3, 11.9, 5.4, 15.0, 16.5]
```

Now think about constructing a bar chart. The Pyplot command is `bar`, which works just like `plot`: the first two arguments are the x and y axes. Here there's no obvious x axis — the country codes and names won't work — so we need to make one. These two lines (i) make up a counter of the same length as `gdppc` that we can use and (ii) generate a bar chart.

```
other_axis = range(len(gdppc))
print(other_axis)
plt.bar(other_axis, gdppc, align='center')
```

This isn't the most elegant chart you'll see, but it's functional.

You might stop here and think: What would you do to dress this chart up, make it more attractive and compelling?

Here are some examples that might get you started:

```
plt.xticks(other_axis, codes, fontsize=12)
plt.xlim((-0.75, 6.75))
plt.ylabel('GDP Per Capita (thousands of USD)')
plt.title('GDP Per Capita', fontsize=16, loc='left')
```

What do you think? What would you add?

- Horizontal bar charts. The same chart looks better if we rotate it. That's easily done by replacing the command `bar` with `barh`. Even better, we now have room to add the country names, not just the three-letter country codes.

```
plt.barh(other_axis, gdppc, align='center', alpha=0.7)
plt.ylim((-0.6, 6.6))
plt.yticks(other_axis, countries, fontsize=14)
```

8 Pyplot's object-oriented interface

- Overview. This is close to classic Matplotlib, which is said to give you greater flexibility and control. You'll have to decide for yourself whether it's worth it. Eventually yes, but it probably makes sense to start with simple interface. We'll go through this at high speed to give you a sense of the syntax.
- References:
  - Matplotlib doesn't have a tutorial. The documentation sends you to the gallery of examples and outside sources.
  - Sargent and Stachurski's "Quantitative Economics" is pretty good, if a little terse. See the section on Matplotlib.
- Basic syntax. A typical graph has commands like these:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplot()
```

`fig` and `ax` here are variable names; we can call them anything we want, but these are standard. `fig` is referred to as a figure object, it controls the overall figure area. If we stop here, we just get a blank box. `ax` is referred to as an axis object, it controls the content of the figure. We'll give some examples below.

- Line plots. We'll start with the same data we used earlier:

```
gdp  = [13271.1, 13773.5, 14234.2, 14613.8, 14873.7, 14830.4, 14418.7,
        14783.8, 15020.6, 15369.2, 15710.3]
pce  = [8867.6, 9208.2, 9531.8, 9821.7, 10041.6, 10007.2, 9847.0, 10036.3,
        10263.5, 10449.7, 10699.7]
year = [2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013]
```

  To generate a plot of US real GDP, we apply the method `plot` to `ax` much as we did before:

```
fig, ax = plt.subplots()                    # sets up single plot
ax.plot(year, gdp, color='blue', linewidth=2, alpha=0.8)
plt.show()                                  # closes plot
```

- Multiple plots.

```
fig, ax = plt.subplots(nrows=2, ncols=1, sharex=True)
# first figure
ax[0].plot(year, gdp, color='blue', linewidth=2, alpha=0.8)
ax[0].text(2010, 14000, 'GDP')
# second figure
ax[1].plot(year, pce, color='magenta', linewidth=2, alpha=0.8)
ax[1].text(2010, 9200, 'Consumption')
plt.show()
```

- **Exercise.** Convert the double line plot above to bar charts.

9 Before the next class

Exercise. Randy Olson has a nice blog post (search "randy olson beautiful") about creating effective graphics. He starts with this gif: http://gfycat.com/ImprobableFemaleBasenji. Skim his post, play the gif, and

- Write a Python script that recreates the data in the figure, both numbers and labels. (There are only a few points, you can just type them into lists.)
- Approximate the various graphs using Matplotlib.
- Play around with the graphics parameters. What is the best graph you can produce? What do you like about it?
- Variant: Do the same with our GDP per capita bar charts.

Today's code

Attached. Download this pdf file, open in Adobe Acrobat or the equivalent, and click on the pushpins: 

Also here:

```
"""
For Class #2 of an informal mini-course at NYU Stern, Fall 2014.

Topic:  reminders, matplotlib graphics

Repository of materials (including this file):
* https://github.com/DaveBackus/Data_Bootcamp

Written by Dave Backus, Sarah Beckett-Hile, and Glenn Okun
Created with Python 3.4
"""

"""
Reminders 0:  editor, IPython console, Object inspector

Reminders 1:  calculations, strings, slicing
"""
x, y, z = 2*3, 2**3, 2/3    # yes, three at once!
a, b = 'some', 'thing'      # strings
c = a + b

numbers = [x, y, z]
strings = [a, b, c]
both = numbers + strings
print([type(both), both[:3], both[3:]])

#%%
"""
Reminders 2:  reading csv's (the exercise from last class)
"""
import pandas as pd
url1 = 'https://raw.githubusercontent.com/fivethirtyeight/data/master/'
url2 = 'college-majors/recent-grads.csv'
df = pd.read_csv(url1+url2)

#%%
# print various properties of df one at a time
properties = [type(df), df.columns, df.index, df.shape, df.head()]
for prop in properties :
    print(prop, end='\n\n')

#%%
"""
Graphics 1:  Pyplot's simple (Matlab-like) interface
```

```
"""
import matplotlib.pyplot as plt             # import pyplot module

# data
# this is FRED series GDPCA, billions of 2009 USD, accessed November 2014
gdp  = [13271.1, 13773.5, 14234.2, 14613.8, 14873.7, 14830.4, 14418.7,
        14783.8, 15020.6, 15369.2, 15710.3]
# FRED series DPCERX1A020NBEA
pce  = [8867.6, 9208.2, 9531.8, 9821.7, 10041.6, 10007.2, 9847.0, 10036.3,
        10263.5, 10449.7, 10699.7]
year = [2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013]

#%%
# line plot
plt.plot(year, gdp, color='#ffcc00', linewidth=2,
         marker='o', alpha=0.8)
plt.plot(year, pce, color='magenta', linewidth=2, alpha=0.5)
#%%
# dress up the figure
plt.title('US Real GDP and Consumption')
plt.ylabel('Billions of 2009 USD')
plt.legend(('GDP', 'Consumption'), loc=0)   # legend
plt.text(2010, 14000, 'GDP')                # add label at specific location
plt.text(2010, 9200, 'Consumption')         # another one
plt.ylim((0, 16000))                        # change y axis limits
plt.tick_params(length=6, width=2, colors='red')
plt.savefig('bootcamp_test.pdf')
plt.show()                                  # this closes the plot

#%%
# bar charts

# Example 1
plt.bar(year, gdp)
#%%                        # bar(x,y)
# prettier version
plt.bar(year, gdp, width=0.8, color='blue', align='center', alpha=1)
plt.show()

#%%
# Example 2
# See:  http://matplotlib.org/examples/lines_bars_and_markers/barh_demo.html
codes     = ['USA', 'FRA', 'JPN', 'CHN', 'IND', 'BRA', 'MEX']
countries = ['United States', 'France', 'Japan', 'China', 'India',
             'Brazil', 'Mexico']
# World Bank data, thousands of 2013 USD, PPP adjusted
gdppc     = [53.1, 36.9, 36.3, 11.9, 5.4, 15.0, 16.5]
other_axis = range(len(gdppc))

plt.bar(other_axis, gdppc, align='center')

# prettify it
```

```
plt.xticks(other_axis, codes, fontsize=12)
plt.xlim((-0.75, 6.75))
plt.ylabel('GDP Per Capita (thousands of USD)')
plt.title('GDP Per Capita', fontsize=16, loc='left')
plt.show()

#%%
# Example 2':  horizontal bars
plt.barh(other_axis, gdppc, align='center', edgecolor='red',
         linewidth=2, alpha=0.2)
plt.ylim((-0.6, 6.6))
plt.yticks(other_axis, countries, fontsize=14)
plt.show()

#%%
"""
Graphics 2:  Object-oriented graphics
"""
import matplotlib.pyplot as plt             # redundant if already done
gdp  = [13271.1, 13773.5, 14234.2, 14613.8, 14873.7, 14830.4, 14418.7,
        14783.8, 15020.6, 15369.2, 15710.3]
pce  = [8867.6, 9208.2, 9531.8, 9821.7, 10041.6, 10007.2, 9847.0, 10036.3,
        10263.5, 10449.7, 10699.7]
year = [2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013]

# set up a plot
fig, ax = plt.subplots()                    # sets up single plot
print([type(fig), type(ax)])                # see what we have

# add content to the object ax
ax.plot(year, gdp, color='blue', linewidth=2, alpha=0.8)

#%%
# multiple plots
fig, ax = plt.subplots(nrows=2, ncols=1, sharex=True)

# first figure
ax[0].plot(year, gdp, color='blue', linewidth=2, alpha=0.8)
ax[0].text(2010, 14000, 'GDP')

# second figure
ax[1].plot(year, pce, color='magenta', linewidth=2, alpha=0.8)
ax[1].text(2010, 9200, 'Consumption')

plt.show()
```