Data Bootcamp: Class #1

Revised: May 27, 2015

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.
- Team: Dave Backus, Glenn Okun, Sarah Beckett-Hile, and a rotating group of ninjas.
- High-level languages: Python, R, and Matlab. All of these languages are useful, but we think Python is the tool of choice. It's a general purpose program, which means it has more flexibility than the others, and it's gradually building up comparable or better functionality in specific areas we care about: data management, graphics, and so on. Also, it's free (so is R).

  Here's a nice comparison of Python and R for data science.
- **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.
- Course 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; the pdf file comes with links. Programs are in the Code folder under Python.
- Other resources:
  - Python the Hard Way: http://learnpythonthehardway.org/book/. Many of our former students say this is the best introduction to Python. Click on "try it now for free" and ignore the offers to buy books and videos. Use the arrows on the right to go to the next or previous lesson. It's more systematic than our introduction (that is, slower and with somewhat more basic programming coverage), but does not go into graphics and data. In short: good starting point, but different focus.

One thing to be aware of: It's based on Python 2. It's not wildly different from Python 3, which is what we use, but you may notice that the print command has changed from `print x` to `print(x)`.

- Codecademy: http://www.codecademy.com/en/tracks/python. This also comes recommended. One clear advantage over Hard Way: you don't need to install Python, you do everything online.
- Coursera: https://www.coursera.org/course/pythonlearn. This is a nice beginner's course in Python. All the materials are online if you want to work through them yourself.

If you try any of them, let us know what you think. Ditto other resources.

- These notes. They're intended to accompany class and may be mysterious on their own. If you see ??, that's a reminder to us to fix something up later.

2 Today's plan

- Python fundamentals.
- Reading csv and xls files.
- Projects.
- Examples.

3 Python fundmentals

- Environments: Python 3.4 in Spyder, editor, IPython console.
- References: Python tutorial, Section 3.
- ** check this out: https://wiki.python.org/moin/SimplePrograms
- ***The logic of a Python program: work through a set of commands in order, not like Excel
- Lines. Each line is a new command. Unlike Excel, the program goes through the lines one at a time.

  Continuation automatic if you have parens or brackets that need to be closed ("implicit"). Blank lines fine, often helpful in making code easier to read. More here and here.
- Operations: `2*3, 2 * 3, 2/3, 2^3, 2**3, log(3), etc`
- Assignments: `x = 2, y = 3.5, z = x/y, etc`
- Comments and `print`: `#, """, #%%, print(x), etc.`
- Strings: `a = 'some', b = 'thing', c = a+b, d = '11.32', etc.`

- Single and double quotes.

  [??] triple quotes can go over several lines,

  we use them for multi-line comments
  https://docs.python.org/3/tutorial/introduction.html#strings

  Also used to supply docs for functions http://stackoverflow.com/a/7057988/804513 **

- Functions and methods. ??

  We'll see shortly that there are often two ways to do things: functions and methods. We can tell functions because they come with arguments in parentheses. In some cases, we even use function() if we don't have any input. [??]

  Here's a list. print... len... [??]

- Types and conversions. [??] `len` and `type`: `len(first), len(x), type(first), type(x)`.

  Add conversions...

- **Exercise.** Wbat do the functions `float`, `int`, and `str` do? What other functions would you like to know about? (We'll never use most of them, but you never know.)

- Type conversions. There are lots of occasions when we want — or need – to change type. Suppose, for example, we want to convert `year = '2011'`, which is defined as a string, to a number. We can do that with `int(year)`. (Or `float`, but we have something else in mind.

- **Exercise.** Suppose we take `year` is a string containing the year of a particular piece of data. How would we construct a string containing the following year?

- Slicing. Find an element of `a`: `a[1], a[0], a[1:3], a[-1], a[:2], a[1] = 'z', etc.`

- **Exercise.** Set `first` equal to your first name, `last` equal to your last name.
  (a) Set `full` equal to your first and last names together with a space between them.
  (b) Set `lastfirst` equal to your last name, comma, space, first name.

- Objects and methods. Most things in Python are **objects** (for example, `x, z, first`). (Experts may say at this point: an **object** is an **instance** of a **class**. Ignore them.) **Methods** are ready-to-go things we can do with them. The available methods depend on the object. To get a list of available methods for a given object, type in the IPython console: `object.[tab]`. When you find a method you're interested in, go to Spyder's Object explorer and type `object.method` for a description and syntax. Examples: `first.find('a'), x.is_integer()`.

- **Exercise.** Take `lastfirst` and see what methods are available.
  (a) Use `lastfirst.find` to find the location of the comma and decompose `lastfirst` into `first` and `last`.
  (b) Find a method that converts `last` to all lower case.

- Lists: collections of things, possibly different, defined by square brackets `[]` with commas between items. Slicing conventions similar to strings. Examples: `numbers = [x, y, z]`, `type(numbers)`, `print(numbers)`, `strings = [a, b, c, d]`, `all = numbers + strings`, `type(all)`, `print(all)`, `all[0]`, `all[-1]`, `all[3:]`, `etc.`

- ***** Style *****

  PEP8 etc
  https://google-styleguide.googlecode.com/svn/trunk/pyguide.html

  Examples:
  http://www.reddit.com/r/Python/comments/3639nl/what_is_the_most_beautiful_piece_of_python_code/
  https://github.com/mitsuhiko/werkzeug/blob/master/werkzeug/routing.py

- Assignment idea: convert list of strings to list with number of letters in each.
  Do in class?
  http://www.reddit.com/r/Python/comments/35ubwo/newbie_for_programming_i_am_working_on_this/

- **Other data structures.** [??] The term data structure refers to the organization of a collection of data. A list is a data structure. ...

  http://en.wikipedia.org/wiki/Data_structure

  Examples: tuples, dictionaries, and sets. We'll cover them later as needed. If you run across one of these terms, keep in mind it's just a different form of organization, we can look up its properties when we need to.

4 Inputting data

- Packages: Collections of commands that do more than basic Python. There are packages that do almost anything, but we'll focus on a couple: Matplotlib and pandas. Matplotlib is a graphics package, we'll talk about it another time. We touch on pandas — briefly — below.

  In Python — and many other languages — we need to tell Python to make them available. Here are some options for doing this with a mythical package `xyz`:

  – `from xyz import *`. This imports all the commands from the package `xyz`. You can replace the star with specific commands and import only them. A command `foo` is then executed by typing `foo`.

  – `import xyz`. This imports the whole thing. A command `foo` is then executed by adding `xyz.` (note the period) before its name: `xyz.foo`.

  – `import xyz as x` is the most common syntax. It allows the shorthand `x.foo`. We'll do this momentarily with the package `pandas`.

  For more, see the Section 6 of the Python tutorial. This refers to modules, which could be packages, subsets of packages, or even code you've written yourself.

- The pandas package: pandas = (Python Data Analysis Library). This is the central package for data work, it allows you to use Python for data management along similar lines to the program R. We'll import it typically with the command *import pandas as pd* and execute commands with syntax of the form pd.command.

- Reference: pandas has the usual high-quality documentation we expect from Python packages. Search "python pandas" or go to http://pandas.pydata.org/. We'll focus here on the input-output tools: http://pandas.pydata.org/pandas-docs/stable/io.html.

- Reading csv files. We've found that csv files are the most useful common data format, far better than xls or xlsx. Its simple structure (entries separated by commas, tabs, or spaces) allows easy and rapid input. That's a general statement, not a statement about Python.

  Suppose you have (as we do) a csv file in a parallel directory `Data`. Then we can read it with the `read_csv` command in pandas:

  ```
  import pandas as pd
  file = '../Data/test1.csv'
  df = pd.read_csv(file)
  ```

  This reads the file `test1.csv` into `df`. If you try `type(df)` you'll find that `df` is a DataFrame, the standard object type in pandas. It's a table of numbers, much like a sheet in a spreadsheet program, with labels for rows (the index) and columns (the variables).

  Some of the methods you could try: `df.columns` (the column labels), `df.columns.tolist()` (the column labels as a list), `df.index` (the row labels), `df.head(2)` (the top two rows), `df.tail()` (the bottom five rows), `df.tail(2)` (the bottom two rows), `df.describe()` (summary statistics), and `df.info()` (information about the form of the data). There are only three rows in this one, so you could simply print the whole thing, but with large data sets this is very useful.

  Comment. The tricky part of this is setting the working directory, where Python looks for things. More on this another time.

- ?? mention read table

- Reading csv's from url's. Even better, `read_csv` also reads from url's. The same file is in our GitHub repository (talk about how to find it, and the Raw button). We can read it with

  ```
  import pandas as pd     # this is redundant if we've already imported it
  url1 = 'https://raw.githubusercontent.com/DaveBackus/'
  url2 = 'Data_Bootcamp/master/Code/Data/test1.csv'
  url = url1 + url2
  df = pd.read_csv(url)
  ```

  Or we could read the World Economic Outlook (WEO) database from the IMF. This reads in the whole thing:

```
url = 'http://www.imf.org/external/pubs/ft/weo/2014/01/weodata/WEOApr2014all.xls'
weo = pd.read_csv(url, sep='\t')    # tab = \t
```

Try this and list the contents. What does the file look like?

- Spreadsheets. They have a more complex structure and the possibility of more than one sheet. Most programmers avoid these files when they can: they create technical problems that are hard to predict. See, for example, this blog post.

  Neverthless, spreadsheets are extremely common in the business world and elsewhere, so we need to be able to work with them. Here's an example that reads in the first sheet:

```
file = '../Data/test2.xlsx'
xls = pd.read_excel(file)       # default is first sheet
```

- Copying from url's. Here's an example that copies a file from a url:

```
import urllib.request           # this is a module from the package urllib
file = 'foo.csv'
urllib.request.urlretrieve(url, file)
```

  This way you can copy the file to your own computer and read it in from there. This does the same thing, courtesy of Sarah:

```
f = urllib.request.urlopen(url)
file = 'foo_sbh.csv'
with open(file, 'wb') as local_file:
    local_file.write(f.read())
```

- **Exercise.** Create a spreadsheet and save it as both csv and xlsx. Write code to read them into Python.

- Copy from clipboard. We're not fans of this, it makes replication hard, but it's awful convenient and recommended by some of our former students.

```
clip = pd.read_clipboard()
```

- Zip files. The `zipfile` module (part of basic Python) lets you do whatever you want with zip files. Some examples:

```
import pandas as pd
import urllib
import zipfile
import os

url  = 'http://databank.worldbank.org/data/download/WDI_csv.zip'
file = os.path.basename(url)            # strip out file name
urllib.request.urlretrieve(url, file)   # copy to disk
```

```
# see what's there
print(['Is zipfile?', zipfile.is_zipfile(file)])
zf = zipfile.ZipFile(file, 'r')
print('List of zipfile contents (two versions)')
[print(file) for file in zf.namelist()]
zf.printdir()

# extract a component
csv = zf.extract('WDI_Data.csv')        # copy to disk
df1 = pd.read_csv('WDI_Data.csv')       # read
print(df1.columns)                      # check contents

# alternative:  open and read
csv = zf.open('WDI_Data.csv')
df2 = pd.read_csv(csv)
print(df3.columns)
```

For more, see the section of the Python tutorial that describes the zipfile module.

5 Basic Pandas

*** Intro to indexing, selecting variables...

6 Projects

It's best to have a goal, something you're working toward. Might be something you can show potential employers to demonstrate your skill set.

7 Examples

These illustrate some of the possibilities of Python with data that's available free from online sources. This goes beyond what we've learned so far, but gives you a sense of what we'll be doing. Some examples:

- FRED. Great online source of macroeconomic data.
- Fama-French. Great source of equity returns on portfolios sorted by type and industry. Used to illustrate boxplots.
- World Bank. Country indicators. Used to illustrate bubble plot (scatter plot with third variable indicated by size of "bubble").

See the code for details. We're still working on Yahoo's Options data, which went from working to not working in the last week.

What examples would you like to see? Tell us, or post a suggestion on the Group.

8 Coming attractions

In the works:

- Basic programming tools: control flow, functions.
- Data: the pandas package.
- Graphics: the Matplotlib package.

If there's interest:

- SQL databases.
- Scraping data from websites.
- Interactive web graphics: Plot.ly and Google charts.
- Anything else?

9 Before the next class

- Play around with Python. Play around with some of the code snippets you've seen. Experiment with new methods. Try to read data you're using for other purposes. Run programs you find in the documentation.
- Think about projects. Skim through our list of projects and data sources, see if anything crosses your mind. Or ask your employer. Or anywhere else you find data.
- Do the assignment below. Really, give it a try. Tell us what undergrad major pays the most. Think about graphics you can use to make the case. Bounce around ideas for a (small) project of your own.
- Get help when you need it. If you get stuck, ask friends for help, or post a question on the Google Group. Uee StackOverflow rules if possible: include a self-contained example of your problem, code that others can fix.
- **Exercise.** Read the article from the FiveThirtyEight (first link below) about income by college major of recent graduates. It takes data from a survey run by the US government (the American Community Survey) and produces figures summarizing the salaries of students who majored in various subjects. They've kindly posted the data online — also the R code — in the second link.

  Your mission:

  (a) Read the data in the file `recent-grads.csv` in the 538 GitHub repository (link below). You should be able to read it into Python straight from the url. Make sure you use the link from the Raw file. How many data points do you have? How many variables?

  (b) Think about graphs you'd like to see, whether in the article or not. You could, for example, look through the Matplotlib gallery (link below) and see if there are any graph types that you think you could use to present this data effectively.

(d) If you're feeling brave, produce a graph of your own. We haven't talked much about graphics yet, so if you get stuck, ask questions or stop.

Links
Article: http://fivethirtyeight.com/features/the-economic-guide-to-picking-a-college-major/
Code and data: https://github.com/fivethirtyeight/data/tree/master/college-majors
Gallery: http://matplotlib.org/1.2.1/gallery.html

Today's code

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

Basics

```
"""
For Class #1 of an informal mini-course at NYU Stern, Fall 2014.

Topics:  calculations, assignments, strings, slicing, lists, data frames,
reading csv and xls files

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
"""
"""
Check Python version
"""
# https://docs.python.org/3.4/library/sys.html
import sys

print('What version of Python? \n', sys.version, '\n', sep='')

if float(sys.version_info[0]) < 3.0:
    raise Exception('***** Program halted, old version of Python *****')

"""
Calculations and assignments (best in IPython console)
"""
x = 2*3
y = 2**3
z = 2/3

"""
Strings
"""
a = 'some'
b = 'thing'
```

```
c = a + b
print(['a[1:3]', a[1:3]])

# names
first, last = 'Dave', 'Backus'
full = first + ' ' + last

"""
Output and input
"""
print(full)
print(first, last)
print(last, ', ', first)
print(last, ', ', first, sep='')

x = input('Type your name here --> ')
print(x, end='\n\n')

"""
Lists
"""
numbers = [x, y, z]
strings = [a, b, c]

both = numbers + strings
print(['both[3:]', both[3:]])

#%%
"""
Inputting data
"""
import pandas as pd
# check version
print('Pandas version ', pd.__version__)

# read from local file
file = '../Data/test1.csv'
df = pd.read_csv(file)

#%%
# some properties
print(df)
print(type(df))
print(['Shape is', df.shape])
print(df.mean())
print(df.columns)
print(['column labels', df.columns])
print(['row labels', df.index])

#%%
# read from url
url = 'https://raw.githubusercontent.com/DaveBackus/Data_Bootcamp/master/Code/Data/test1.csv'
```

```
dfurl = pd.read_csv(url)

#%%
# read IMF's WEO data from
url = 'http://www.imf.org/external/pubs/ft/weo/2014/01/weodata/WEOApr2014all.xls'
weo = pd.read_csv(url, sep='\t')    # tab = \t
print(weo.head())
print(['column labels', weo.columns])
print(['row labels', weo.index])

#%%
# copy file from url to hard drive
import urllib.request           # this is a module from the package urllib
file = 'foo.csv'
url = 'https://raw.githubusercontent.com/DaveBackus/Data_Bootcamp/master/Code/Data/test1.csv'
urllib.request.urlretrieve(url, file)

#%%
# Sarah's version
f = urllib.request.urlopen(url)
file = 'foo_sbh.csv'
with open(file, 'wb') as local_file:
    local_file.write(f.read())

#%%
# read from xls
file = '../Data/test2.xlsx'
xls = pd.read_excel(file)       # default is first sheet

#%%
# zip files
import pandas as pd
import urllib
import zipfile
import os

# this is a big file, best to test with something smaller
url  = 'http://databank.worldbank.org/data/download/WDI_csv.zip'
file = os.path.basename(url)            # strip out file name
urllib.request.urlretrieve(url, file)   # copy to disk

# see what's there
print(['Is zipfile?', zipfile.is_zipfile(file)])
zf = zipfile.ZipFile(file, 'r')
print('List of zipfile contents (two versions)')
[print(file) for file in zf.namelist()]
zf.printdir()

# extract a component
csv = zf.extract('WDI_Data.csv')        # copy to disk
df1 = pd.read_csv('WDI_Data.csv')       # read
print(df1.columns)                      # check contents
```

```
# alternative:  open and read
csv = zf.open('WDI_Data.csv')
df2 = pd.read_csv(csv)
print(df2.columns)
```

Examples

```
"""
Examples for Class #1 of an informal mini-course at NYU Stern, Fall 2014.
Examples are
* US GDP growth via FRED
* GDP per capita in a set of countries via World Bank
* Fama-French US equity factors via French
* Stock options via Yahoo (not yet)

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

Prepared by Dave Backus, Sarah Beckett-Hile, and Glenn Okun
Created with Python 3.4
"""
"""
1. Read in US real GDP from FRED (code GDPC1)

References
* http://research.stlouisfed.org/fred2/
* http://pandas.pydata.org/pandas-docs/stable/
* http://pandas.pydata.org/pandas-docs/stable/remote_data.html#fred
*
"""
import datetime as dt
import pandas.io.data as web       # data import tools
import matplotlib.pyplot as plt    # plotting tools

# get data from FRED
fred_series = ["GDPC1"]
start = dt.datetime(1985, 1, 1)
data = web.DataReader(fred_series, "fred", start)
print(data.tail())                  # to see what we've got

# compute annualized growth rate, change label, and print stats
g = 4*data.pct_change()
g.columns = ['US GDP Growth']
print(['Mean and std dev', g.mean(), g.std()])

# quick and dirty plot
g.plot()
plt.show()

#%%
```

```
"""
2. Fama-French equity factors from Ken French's website

References
* http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
* http://quant-econ.net/pandas.html
* http://pandas.pydata.org/pandas-docs/dev/remote_data.html#fama-french
* http://pandas.pydata.org/pandas-docs/stable/10min.html#selection
* http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.boxplot
* http://pandas.pydata.org/pandas-docs/dev/generated/pandas.DataFrame.hist.html
"""
# load packages (if it's redundant it'll be ignored)
import pandas.io.data as web

# read data from Ken French's website
ff = web.DataReader('F-F_Research_Data_Factors', 'famafrench')[0]
# NB:  ff.xs is a conflict, rename to xsm
ff.columns = ['xsm', 'smb', 'hml', 'rf']

print(type(ff))

# see what we've got
print(ff.head(3))
print(ff.describe())

# stats
moments = [ff.mean(), ff.std(), ff.skew(), ff.kurtosis() - 3]
print('Summary stats for Fama-French factors (mean, std, skew, ex kurt)') #, end='\n\n')
print(moments)

# some plots
ff.hist(bins=50, sharex=True)
plt.show()

ff.boxplot(whis=0, return_type='axes')
plt.show()

#%%
"""
3. GDP per capita and life expectancy in selected countries (World Bank data)

References
* http://data.worldbank.org/
* http://pandas.pydata.org/pandas-docs/stable/remote_data.html#world-bank
* http://quant-econ.net/pandas.html#data-from-the-world-bank
* http://matplotlib.org/examples/shapes_and_collections/scatter_demo.html
"""
# load packages (ignored if redundant)
# load package under name wb
from pandas.io import wb
import numpy as np
import matplotlib.pyplot as plt
```

```
# specify dates, variables, and countries
start = 2011
# GDP per capita, population, life expectancy
variable_list = ['NY.GDP.PCAP.KD', 'SP.POP.TOTL', 'SP.DYN.LE00.IN']
country_list  = ['US', 'FR', 'JP', 'CN', 'IN', 'BR', 'MX']

# Python understands we need to go to the second line because ( hasn't been closed by )
data = wb.download(indicator=variable_list,
                   country=country_list, start=start, end=start).dropna()
# see what we've got
print(data)

# check the column labels, change to something simpler
print(data.columns)
data.columns = ['gdppc', 'pop', 'le']
print(data)

# scatterplot
# life expectancy v GDP per capita
# size of circles controlled by population

plt.scatter(data['gdppc'], data['le'], s=0.000001*data['pop'], alpha=0.5)
plt.ylabel('Life Expectancy')
plt.xlabel('GDP Per Capita')
plt.show()
```