csv-parsing-plotting

Michele Pratusevich

December 11, 2013

1 Data analysis with Python

What we want to do: draw an interesting conclusion from data. Go from table of data to a picture that tells us more.

Steps:

1. Get data
2. Put data into Python
    1. Use python `csv` package
    2. String parsing / formatting (turn into two lists of numbers)
3. Draw a plot / decide what plot to draw
    1. matplotlib
4. Draw conclusions

Big ideas: iteration, planning, big picture

Definitions

- **plot** - any picture that shows data in an interesting way (line graph, bar graph, histogram, etc.)
- **graph** - same as a plot
- **parsing** - getting useful information out of something. For example, getting the month out of the date string
- **formatting** - turn something into a different way of representing (usually about strings). For example, turning dates from American format to European format

ipython notebook

A **better** python shell. "interactive python" – run one piece of python at a time and see what happens. It's also in a web browser.

Run it from the folder in the terminal you want to save you work:

```
ipython notebook --pylab inline
```

It will open Firefox and you'll edit python from there. (If you are working on Windows, this is almost the same. Take a look at the Windows install instructions when you start the lab for more information.)

1.1 String parsing / formatting

Context: computers can do some things better than humans. One of these is change formats of strings. (e.g. change date format from American to European)

Problem: I run the American branch of the Red Cross. For every patient that we treat I collect their birthday. I want to give this data to my counterpart in France to compare the age demographic of our patients. We have over 3 million records for the last year alone - I need to change all the date formats before I send the data over.

2013-10-30 –> 2013-30-10

- By hand this is LONG AND TEDIOUS
- By computer this is FAST AND EASY

In [1]:
```
# d is in form 2013-10-23 but we want
# in form 2013-23-10

dates = ["2013-10-30"]

for d in dates:
print "old = ", d
b = d.split("-")
new_date = b[0] + "-" + b[2] + "-" + b[1]
print "new = ", new_date
```

```
old =  2013-10-30
new =  2013-30-10
```

1.2 When do you have these kinds of problems?

- Modify LOTS of strings
- Tables (of data) (like Excel sheets)
- Move data from one storage to another (the example above)

1.3 How data is stored

- csv = "**C**omma-**S**eparated **V**alue"
- store values in a text document, but special formatting for data (in a table)
- sometimes the first row are "headers" (titles of columns)
- a file format Python can understand
- Example:
  - Data about oysters
  - Save it as a csv

1.4 Getting data into Python

Use the **csv** package. Here is a good tutorial. And here is a very simple example.

In [33]:
```
import csv

# the name of the file
# 'rb' means "for reading the file"
# (if you write 'wb' that is
# "for writing the file")
f = open('30oysters.csv', 'rb')

# the actual reader object that has the file
reader = csv.reader(f)

#using this to count the row numbers
rownum = 0

# the reader object is made of rows of data
for row in reader:

    # the header is the first row
    if rownum == 0:
        header = row
    else:

        # for now just print the row
        # later we will add more here
        print row

    # count the row numbers
    rownum += 1

# you always must close your file,
# or you will use too much memory
f.close()
```

```
['1         12.92          13.04          5136699   47907']
['2         11.40          11.71          4795151   41458']
['3         17.42          17.42          6453115   60891']
['4          6.79           7.23          2895239   29949']
['5          9.62          10.03          3672746   41616']
['6         15.50          15.59          5728880   48070']
['7          9.66           9.94          3987582   34717']
['8          7.02           7.53          2678423   27230']
['9         12.56          12.73          5481545   52712']
['10        12.49          12.66          5016762   41500']
['11        10.12          10.53          3942783   31216']
['12        10.64          10.84          4052638   41852']
['13        12.99          13.12          5334558   44608']
['14         8.09           8.48          3527926   35343']
['15        14.09          14.24          5679636   47481']
['16        10.73          11.11          4013992   40976']
['17        15.17          15.35          5565995   65361']
['18        15.50          15.44          6303198   50910']
['19         5.22           5.67          1928109   22895']
['20         7.75           8.26          3450164   34804']
['21        10.71          10.95          4707532   37156']
['22         7.91           7.97          3019077   29070']
['23         6.93           7.34          2768160   24590']
['24        13.63          13.21          4945743   48082']
['25         7.67           7.83          3138463   32118']
['26        11.27          11.38          4410797   45112']
['27        10.98          11.22          4558251   37020']
```

```
['28          8.87            9.25            3449867    39333']
['29         13.68           13.75            5609681    51351']
['30         14.27           14.37            5292105    53281']
```

Rows from the reader are returned in the form of a list. Each row is a list.

Lab 3, Exercise 1: data, csv, and ipython notebook

Now that we have data in python, the goal is to turn that data into a list of numbers, so we can plot it.

1.5 String formatting

Each row from the reader is given to us as a string inside a list or a list of strings (it depends on the data set, so you have to iteratively figure out what you need to do).

A combination of string formatting and list manipulation.

String methods

Methods to remember (these act on strings):

- `.replace()` - replace characters in a string with another string
- `.split()` - cut a string into pieces based on whitespace

In [21]:
```
a = "Alpine \t\t15.4      6.7"
print a
```

```
Alpine          15.4      6.7
```

In [22]:
```
b = a.replace("\t", " ")
print b
# for reference, \t is a tab character
```

```
Alpine    15.4      6.7
```

In [23]:
```
c = b.split()
print c
```

```
['Alpine', '15.4', '6.7']
```

List manipulation

Methods to remember (these act on lists):

- `.append()` - adds one element to the end of the list
- `.extend()` - adds a list of elements to the end of the list
- `len()` - returns the number of elements in the list

And index manipulations:

```
a = [1, 2, 3, 4]
a[1:3] → [2, 3]
a[1:] → [2, 3, 4]
a[-2:0] → [3, 4]
a[:] → [1, 2, 3, 4]
```

In [9]:
```
a = ["a", "b", "c", "d"]
print "length =", len(a)
```

```
length = 4
```

In [10]:
```
a.append("e")
print "after append", a
print "length", len(a)
```

```
after append ['a', 'b', 'c', 'd', 'e']
length 5
```

In [11]:
```
a.extend(["f", "g"])
print "after extend", a
print "length", len(a)
```

```
after extend ['a', 'b', 'c', 'd', 'e', 'f', 'g']
length 7
```

In [12]:
```
a[1:3]
```

Out [12]:
```
['b', 'c']
```

In [13]:
```
a[0]
```

Out [13]:
```
'a'
```

In [15]:
```
a[2:]
```

Out [15]:
```
['c', 'd', 'e', 'f', 'g']
```

In [16]:
```
a[:3]
```

Out [16]:
```
['a', 'b', 'c']
```

In [17]:
```
a[:]
```

Out [17]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

In [18]:
```
a[-2:]
```

Out [18]: ['f', 'g']

In [19]:
```
a[::-1]
```

Out [19]: ['g', 'f', 'e', 'd', 'c', 'b', 'a']

And many more! You have a shell, just try it!

Strings to numbers

Methods to remember:

- `float()` - turns a string into a float
- `int()` - turns a string into an int
- `map()` - does a function on each member of a list

In [28]:
```
n = "15"
print int(n)
print float(n)
```

```
15
15.0
```

In [31]:
```
d = ['15', '4']
print "original = ", d
print "int map = ", map(int, d)
print "float map = ", map(float, d)
```

```
original =  ['15', '4']
int map =  [15, 4]
float map =  [15.0, 4.0]
```

In [32]:
```
d = ['maya13', 'nicole13']
def meetify(string):
    return string + "-meet"
print d
print map(meetify, d)
```

```
['maya13', 'nicole13']
['maya13-meet', 'nicole13-meet']
```

Now that we know how to format strings, where do we do this with our data? With each row of course!

In [34]:
```
import csv
f = open('30oysters.csv', 'rb')
reader = csv.reader(f)
rownum = 0
weight = []
vol = []
for row in reader:
    if rownum == 0:
        header = row
    else:
        print "row = ", row
```

```
        r = map(float, row[0].split())
        print "formatted row = ", r
    rownum += 1
f.close()
```

```
row =  ['1         12.92           13.04            5136699   47907']
formatted row =  [1.0, 12.92, 13.04, 5136699.0, 47907.0]
row =  ['2         11.40           11.71            4795151   41458']
formatted row =  [2.0, 11.4, 11.71, 4795151.0, 41458.0]
row =  ['3         17.42           17.42            6453115   60891']
formatted row =  [3.0, 17.42, 17.42, 6453115.0, 60891.0]
row =  ['4          6.79            7.23            2895239   29949']
formatted row =  [4.0, 6.79, 7.23, 2895239.0, 29949.0]
row =  ['5          9.62           10.03            3672746   41616']
formatted row =  [5.0, 9.62, 10.03, 3672746.0, 41616.0]
row =  ['6         15.50           15.59            5728880   48070']
formatted row =  [6.0, 15.5, 15.59, 5728880.0, 48070.0]
row =  ['7          9.66            9.94            3987582   34717']
formatted row =  [7.0, 9.66, 9.94, 3987582.0, 34717.0]
row =  ['8          7.02            7.53            2678423   27230']
formatted row =  [8.0, 7.02, 7.53, 2678423.0, 27230.0]
row =  ['9         12.56           12.73            5481545   52712']
formatted row =  [9.0, 12.56, 12.73, 5481545.0, 52712.0]
row =  ['10        12.49           12.66            5016762   41500']
formatted row =  [10.0, 12.49, 12.66, 5016762.0, 41500.0]
row =  ['11        10.12           10.53            3942783   31216']
formatted row =  [11.0, 10.12, 10.53, 3942783.0, 31216.0]
row =  ['12        10.64           10.84            4052638   41852']
formatted row =  [12.0, 10.64, 10.84, 4052638.0, 41852.0]
row =  ['13        12.99           13.12            5334558   44608']
formatted row =  [13.0, 12.99, 13.12, 5334558.0, 44608.0]
row =  ['14         8.09            8.48            3527926   35343']
formatted row =  [14.0, 8.09, 8.48, 3527926.0, 35343.0]
row =  ['15        14.09           14.24            5679636   47481']
formatted row =  [15.0, 14.09, 14.24, 5679636.0, 47481.0]
row =  ['16        10.73           11.11            4013992   40976']
formatted row =  [16.0, 10.73, 11.11, 4013992.0, 40976.0]
row =  ['17        15.17           15.35            5565995   65361']
formatted row =  [17.0, 15.17, 15.35, 5565995.0, 65361.0]
row =  ['18        15.50           15.44            6303198   50910']
formatted row =  [18.0, 15.5, 15.44, 6303198.0, 50910.0]
row =  ['19         5.22            5.67            1928109   22895']
formatted row =  [19.0, 5.22, 5.67, 1928109.0, 22895.0]
row =  ['20         7.75            8.26            3450164   34804']
formatted row =  [20.0, 7.75, 8.26, 3450164.0, 34804.0]
row =  ['21        10.71           10.95            4707532   37156']
formatted row =  [21.0, 10.71, 10.95, 4707532.0, 37156.0]
row =  ['22         7.91            7.97            3019077   29070']
formatted row =  [22.0, 7.91, 7.97, 3019077.0, 29070.0]
row =  ['23         6.93            7.34            2768160   24590']
formatted row =  [23.0, 6.93, 7.34, 2768160.0, 24590.0]
row =  ['24        13.63           13.21            4945743   48082']
formatted row =  [24.0, 13.63, 13.21, 4945743.0, 48082.0]
row =  ['25         7.67            7.83            3138463   32118']
```

```
formatted row =  [25.0, 7.67, 7.83, 3138463.0, 32118.0]
row =  ['26        11.27           11.38           4410797   45112']
formatted row =  [26.0, 11.27, 11.38, 4410797.0, 45112.0]
row =  ['27        10.98           11.22           4558251   37020']
formatted row =  [27.0, 10.98, 11.22, 4558251.0, 37020.0]
row =  ['28         8.87            9.25           3449867   39333']
formatted row =  [28.0, 8.87, 9.25, 3449867.0, 39333.0]
row =  ['29        13.68           13.75           5609681   51351']
formatted row =  [29.0, 13.68, 13.75, 5609681.0, 51351.0]
row =  ['30        14.27           14.37           5292105   53281']
formatted row =  [30.0, 14.27, 14.37, 5292105.0, 53281.0]
```

Lab 3, Exercise 2: string formatting

1.6 Plotting

We now have our data in number format so we can use it for drawing graphs. But the data is not useful for plotting. When we plot, we need two lists of numbers (two variables to plot, one on the `x` axis and one on the `y` axis).

Parsing the right data

We need to save the data we want into two separate lists to plot. Right now, data from our `csv` file is in COLUMNS:

But we want our data in two LISTS instead:*How do I know what columns to use?* Look at the headering of the columns in the original file!

In [36]:
```
import csv
f = open('30oysters.csv', 'rb')
reader = csv.reader(f)
rownum = 0
weight = []
vol = []
for row in reader:
    if rownum == 0:
        header = row
    else:
        r = map(float, row[0].split())
        weight.append(r[1])
        vol.append(r[2])
    rownum += 1

print "weight = ", weight
print "vol = ", vol
f.close()
```

```
weight =  [12.92, 11.4, 17.42, 6.79, 9.62, 15.5, 9.66, 7.02, 12.56,
12.49, 10.12, 10.64, 12.99, 8.09, 14.09, 10.73, 15.17, 15.5, 5.22,
7.75, 10.71, 7.91, 6.93, 13.63, 7.67, 11.27, 10.98, 8.87, 13.68,
14.27]
vol =  [13.04, 11.71, 17.42, 7.23, 10.03, 15.59, 9.94, 7.53, 12.73,
12.66, 10.53, 10.84, 13.12, 8.48, 14.24, 11.11, 15.35, 15.44, 5.67,
8.26, 10.95, 7.97, 7.34, 13.21, 7.83, 11.38, 11.22, 9.25, 13.75,
14.37]
```

1.7 Plotting the data

Two types of plots: `dot / scatter plots` and `line plots`.

Methods to remember:

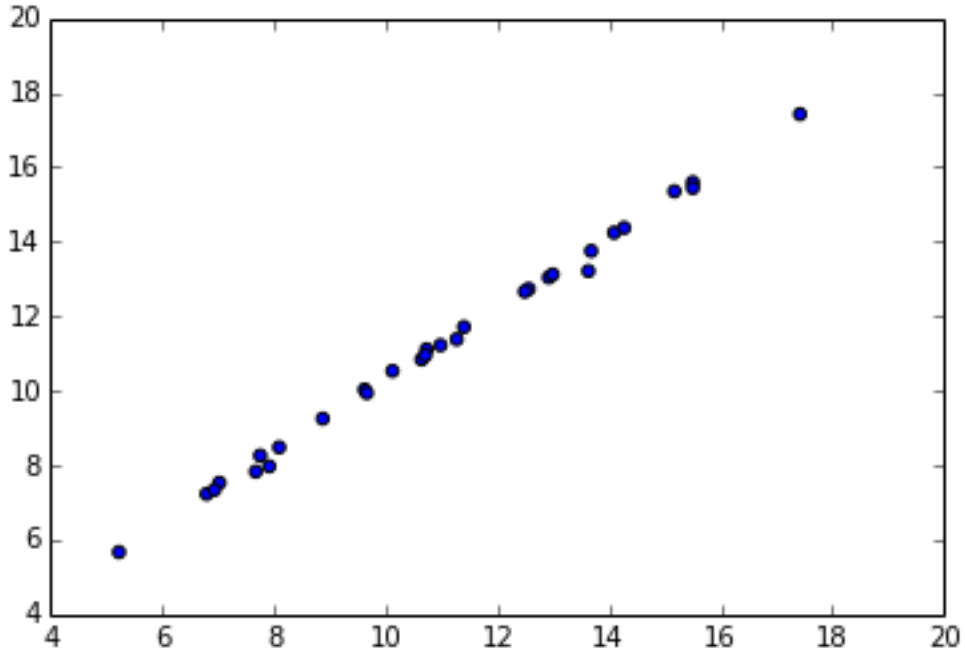
- `scatter(x, y)` - makes a dot plot with `x` on the bottom and `y` on the side
- `plot(x, y)` - makes a line plot with `x` on the bottom and `y` on the side

The type of data you have and what you want to show decides what type of plot to use.

In [40]:
```
# need to import this so we can actually plot stuff
from pylab import *
```

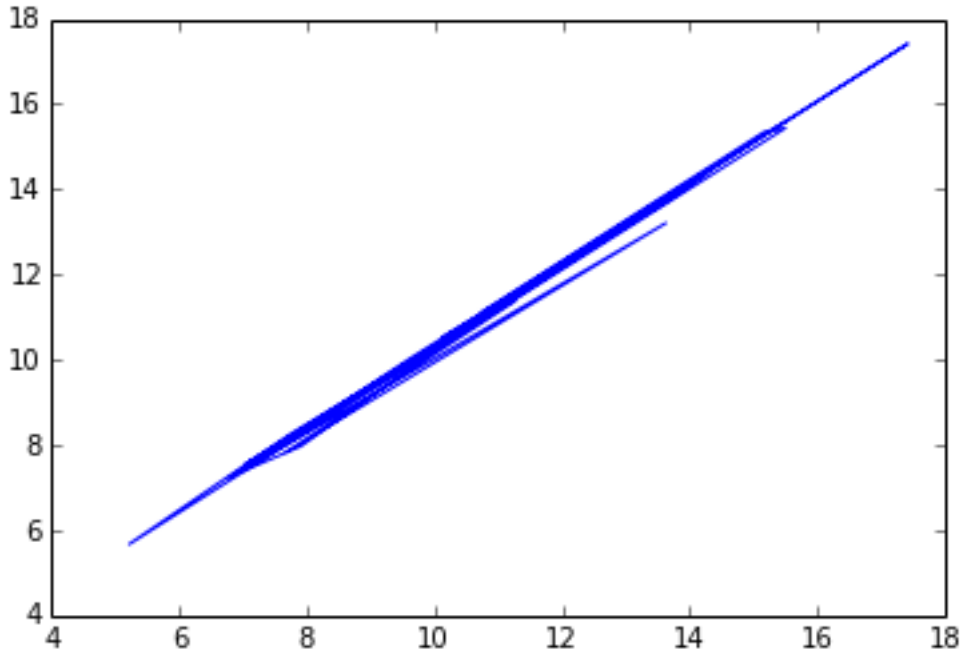
In [41]:
```
scatter(weight, vol)
```

Out [41]: `<matplotlib.collections.PathCollection at 0x42a3750>`



In [42]:
```
plot(weight, vol)
```

Out [42]: `[<matplotlib.lines.Line2D at 0x44a5710>]`



Lab 3, Exercise 3: making your first plot

1.8 Other types of plots

Bar plots

- http://stackoverflow.com/questions/11617719/how-to-plot-a-very-simple-bar-chart-python-matplotlib-using-input-txt-file
- http://scienceoss.com/bar-plot-with-custom-axis-labels/

1.9 Drawing conclusions

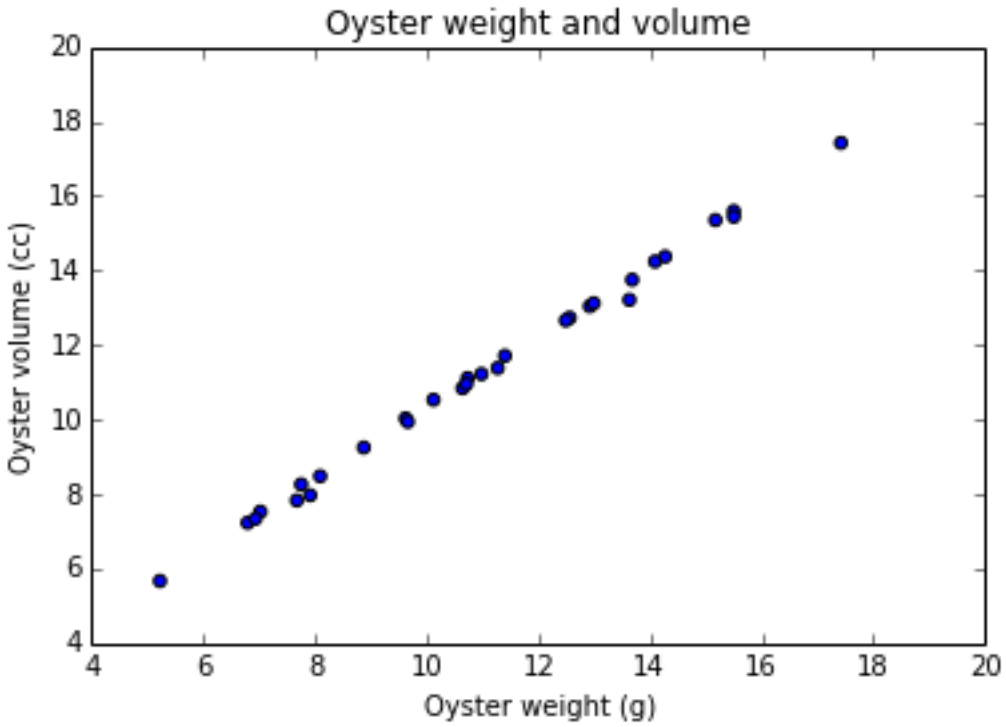
1. Adding labels
2. Writing a sentence summary

Adding labels

Pretty self-explanatory with the sample code. The things you keep are the `plt.figure()` and `.add_subplot(111)`. The `111` just means that you are adding a subplot to fill the entire space of the plot window.

In [44]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('Oyster weight and volume')
ax.set_xlabel('Oyster weight (g)')
ax.set_ylabel('Oyster volume (cc)')
ax.scatter(weight, vol)
# this saves the figure in a file
fig.savefig("oyster_with_titles.png")
```

Out [44]: <matplotlib.collections.PathCollection at 0x44db390>



There is a clear relationship between the weight and volumes of oysters - the bigger the oyster, the heavier it is.