Exploring Science on Twitter

with IPython Notebook and Python Pandas

Brenda Moon
@brendam

Exploring Science on Twitter

with IPython Notebook and Python Pandas

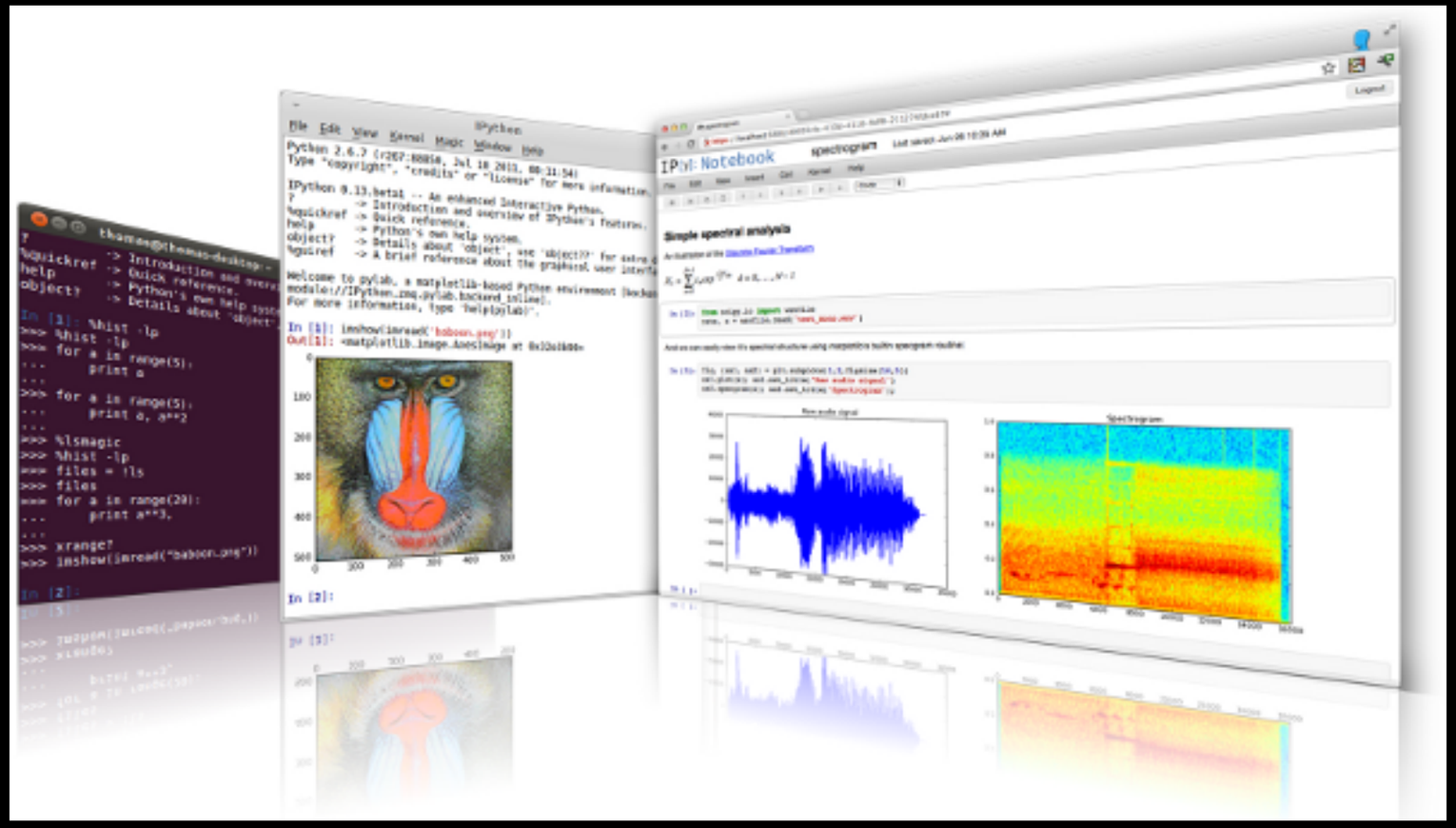
Brenda Moon
@brendam

Python Environment

- virtualenv - keep environment isolated
- virtualenvwrapper - manage virtualenv easily
- cpvirtualenv - copy virtualenv
- setvirtualenvproject - set work directory

IPython Notebook

http://www.ipython.org/



1. bash

```
ruby:~ brenda$ workon pyconau2013
(pyconau2013)ruby:pyconau2013 brenda$ ipython notebook --pylab inline
```

```
$ ipython notebook --pylab inline
] Using existing profile dir: u'/Users/brenda/.ipython/profile_default'
] Using MathJax from CDN: http://cdn.mathjax.org/mathjax/latest/MathJax.js
] Serving notebooks from local directory: /Volumes/HDD/Documents/pyconau2013
] The IPython Notebook is running at: http://127.0.0.1:8888/
] Use Control-C to stop this server and shut down all kernels.
```

IPy IPython Dashboard | IPy Example of an IPython Not

127.0.0.1:8888

IP[y]: Notebook

Notebooks | Clusters

To import a notebook, drag the file onto the listing below or **click here**.

Refresh | New Notebook

/ Volumes / HDD / Documents / pyconau2013 /

Example of an IPython Notebook — Shutdown

IPy IPython Dashboard | IPy Example of an IPython Not

127.0.0.1:8888/34e0fc74-de32-4b10-a61b-0714edc332bb

IP[y]: Notebook Example of an IPython Notebook

File Edit View Insert Cell Kernel Help

Heading 1 Cell Toolbar: None

```
Example of an IPython Notebo
```

virtualenv: pyconau2013

6 July 2013

Simple example of how you can record notes about what you are doing interleaved with the code you are writing.

In [ ]:
```
import pandas
```

Example of an IPython Notebook

virtualenv: pyconau2013

6 July 2013

Simple example of how you can record notes about what you are doing interleaved with the code you are writing.

In [ ]:
```
import pandas
```

IP[y]: Notebook Example of an IPython Notebook

you are writing.

In [10]:
```
import pandas
pandas.load?
```

```
Type:        function
String Form:<function load at 0x10c262c08>
File:        /Users/brenda/.virtualenvs/pyconau2013/lib/python2.7/site-packages/pa
ndas-0.12.0.dev_2bde9e0-py2.7-macosx-10.8-intel.egg/pandas/core/common.py
Definition: pandas.load(path)
Docstring:
Load pickled pandas object (or any other pickled object) from the specified
file path

Warning: Loading pickled data received from untrusted sources can be unsafe.
See: http://docs.python.org/2.7/library/pickle.html

Parameters
----------
path : string
```

IP[y]: Notebook Example of an IPython Notebook

you are writing.

In [11]:
```
import pandas
pandas.load??
```

```
Type:        function
String Form:<function load at 0x10c262c08>
File:        /Users/brenda/.virtualenvs/pyconau2013/lib/python2.7/site-packages/pa
ndas-0.12.0.dev_2bde9e0-py2.7-macosx-10.8-intel.egg/pandas/core/common.py
Definition: pandas.load(path)
Source:
def load(path):  # TODO remove in 0.13
    """
    Load pickled pandas object (or any other pickled object) from the specified
    file path

    Warning: Loading pickled data received from untrusted sources can be unsafe.
    See: http://docs.python.org/2.7/library/pickle.html

    Parameters
```

IP[y]: Notebook Example of an IPython Notebook

In [12]:
```
%magic
```

```
IPython's 'magic' functions
===========================

The magic function system provides a series of functions which allow you to
control the behavior of IPython itself, plus a lot of system-type
features. There are two kinds of magics, line-oriented and cell-oriented.

Line magics are prefixed with the % character and work much like OS
command-line calls: they get as an argument the rest of the line, where
arguments are passed without parentheses or quotes.  For example, this will
time the given statement::

        %timeit range(1000)

Cell magics are prefixed with a double %%, and they are functions that get as
an argument not only the rest of the line, but also the lines below it in a
separate argument.  These magics are called with two arguments: the rest of the
```

IP[y]: Notebook Example of an IPython Notebook

In [13]:
```
%%timeit?
```

```
Definition: %%timeit(self, line='', cell=None)
Docstring:
Time execution of a Python statement or expression

Usage, in line mode:
  %timeit [-n<N> -r<R> [-t|-c]] statement
or in cell mode:
  %%timeit [-n<N> -r<R> [-t|-c]] setup_code
  code
  code...

Time execution of a Python statement or expression using the timeit
module.  This function can be used both as a line and cell magic:

- In line mode you can time a single-line statement (though multiple
  ones can be chained with using semicolons).

- In cell mode, the statement in the first line is used as setup code
  (executed but not timed) and the body of the cell is timed.  The cell
```

IP[y]: Notebook Example of an IPython Notebook

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

In [14]: %pastebin?

```
String Form:<bound method CodeMagics.pastebin of <IPython.core.magics.code.CodeMa
gics object at 0x10adc95d0>>
Namespace:  IPython internal
File:       /Users/brenda/Documents/python/ipython/IPython/core/magics/code.py
Definition: %pastebin(self, parameter_s='')
Docstring:
Upload code to Github's Gist paste bin, returning the URL.

Usage:
  %pastebin [-d "Custom description"] 1-7

The argument can be an input history range, a filename, or the name of a
string or macro.

Options:

  -d: Pass a custom description for the gist. The default will say
      "Pasted from IPython".
```

IP[y]: Notebook Example of an IPython Notebook

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

In [15]: `%save?`

```
Definition: %save(self, parameter_s='')
Docstring:
Save a set of lines or a macro to a given filename.

Usage:
  %save [options] filename n1-n2 n3-n4 ... n5 .. n6 ...

Options:

  -r: use 'raw' input.  By default, the 'processed' history is used,
  so that magics are loaded in their transformed version to valid
  Python.  If this option is given, the raw input as typed as the
  command line is used instead.

  -f: force overwrite.  If file exists, %save will prompt for overwrite
  unless -f is given.

  -a: append to the file instead of overwriting it.

This function uses the same syntax as %history for input ranges
```

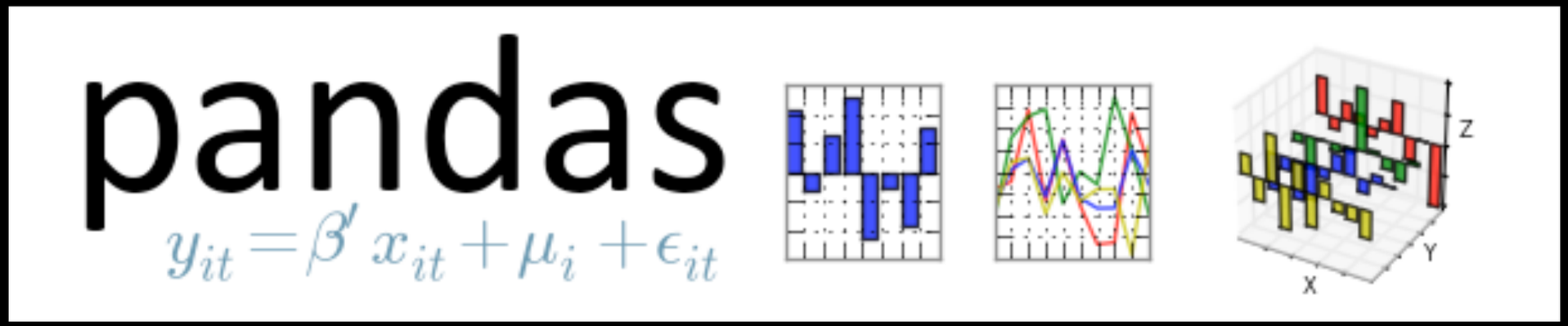
IP[y]: Notebook Example of an IPython Notebook

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

In [16]: %run?

```
Definition: %run(self, parameter_s='', runner=None, file_finder=<function get_py_
filename at 0x10a307848>)
Docstring:
Run the named file inside IPython as a program.

Usage:
  %run [-n -i -e -G]
       [( -t [-N<N>] | -d [-b<N>] | -p [profile options] )]
       ( -m mod | file ) [args]

Parameters after the filename are passed as command-line arguments to
the program (put in sys.argv). Then, control returns to IPython's
prompt.

This is similar to running at a system prompt:
  $ python file args
but with the advantage of giving you IPython's tracebacks, and of
loading all variables into your interactive namespace for further use
(unless -p is used, see below).
```



http://pandas.pydata.org/

"high-performance, easy-to-use data structures and data analysis tools"

- series (array like)
- dataFrame (table)

strong support for time based indexing

Tweets per Day

- Tweets containing the word ‘science’
- 12 million ‘science’ tweets in 2011
- lets look at tweets per day

```
import pandas
import couchdbkit

server = couchdbkit.Server('http://brenda:XXXXX@127.0.0.1:5984/')
tweetdb = server.get_db('tweets')
tweets = list(
    tweetdb.view(
        "tweetsPerDay/tweetsPerDay",
        reduce=True,
        group_level=3,
        startkey=[2011, 1, 1],
        endkey=[2012, 1, 1]))
date_list = [
    pandas.datetime(tweet["key"][0], (tweet["key"][1]), tweet["key"][2])
    for tweet in tweets]
date_index = pandas.DatetimeIndex(date_list)
data_list = [tweet["value"] for tweet in tweets]
science_tweets = pandas.Series(data_list, date_index)
# change to float so can have NaN values
science_tweets = science_tweets.astype(float)
# mask out the missing data period so it doesn't plot
science_tweets['2011-03-31':'2011-04-12'] = numpy.NaN
# final check that start and end dates are correct
print 'first element: ', science_tweets.first('1D')
print 'last element: ', science_tweets.last('1D')
# check than the missing values don't plot.
science_tweets.plot()
science_tweets.to_pickle('dataFiles/2011TweetsPerDay-final.pkl')
```

```
import pandas
import couchdbkit
```

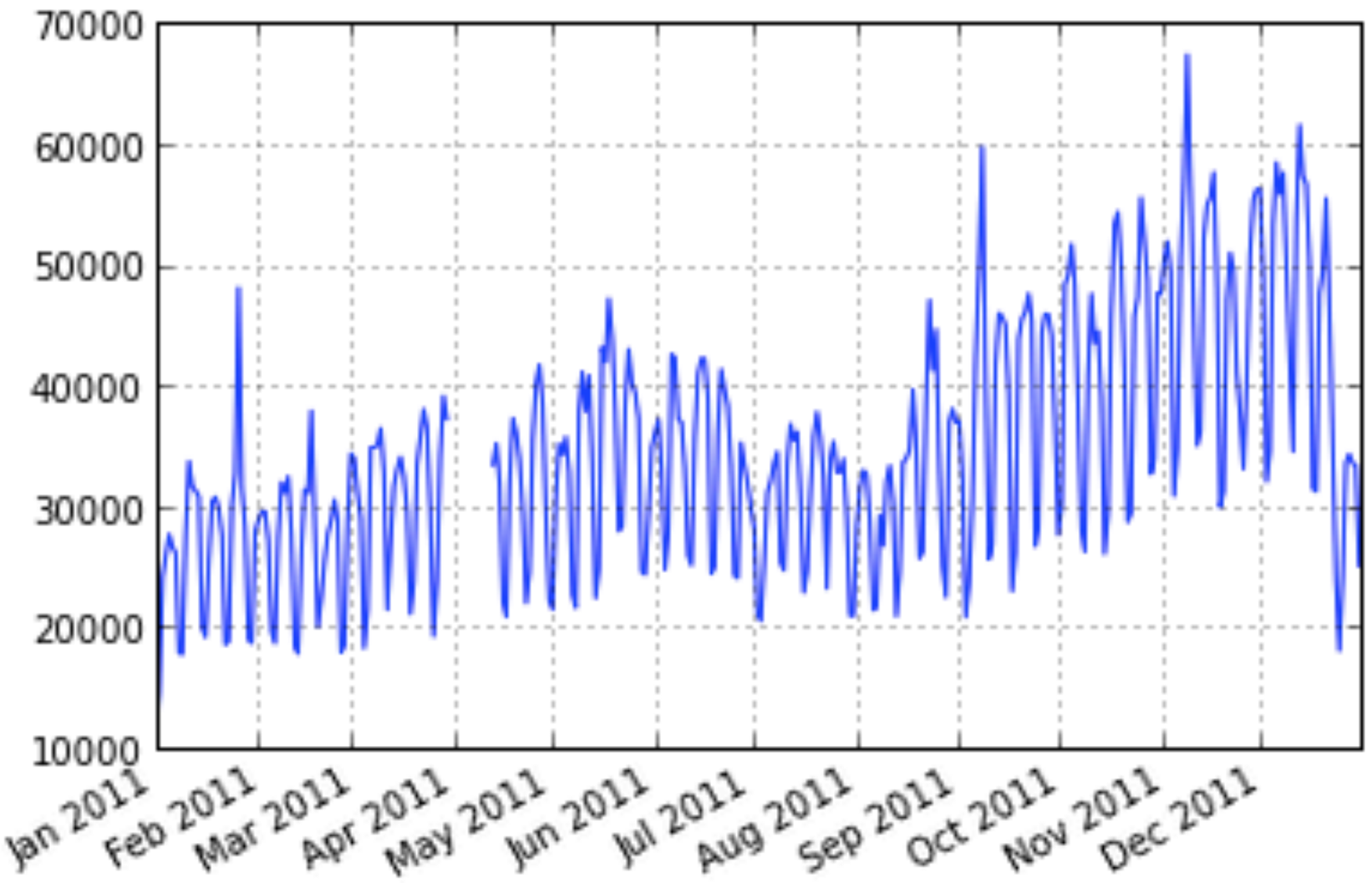
```
server = couchdbkit.Server('http://brenda:XXXXX@127.0.0.1:5984/')
tweetdb = server.get_db('tweets')
tweets = list(
    tweetdb.view(
        "tweetsPerDay/tweetsPerDay",
        reduce=True,
        group_level=3,
        startkey=[2011, 1, 1],
        endkey=[2012, 1, 1]))
date_list = [
    pandas.datetime(tweet["key"][0], (tweet["key"][1]), tweet["key"][2])
    for tweet in tweets]
date_index = pandas.DatetimeIndex(date_list)
data_list = [tweet["value"] for tweet in tweets]
science_tweets = pandas.Series(data_list, date_index)
# change to float so can have NaN values
science_tweets = science_tweets.astype(float)
# mask out the missing data period so it doesn't plot
science_tweets['2011-03-31':'2011-04-12'] = numpy.NaN
# final check that start and end dates are correct
print 'first element: ', science_tweets.first('1D')
print 'last element: ', science_tweets.last('1D')
# check than the missing values don't plot.
science_tweets.plot()
science_tweets.to_pickle('dataFiles/2011TweetsPerDay-final.pkl')
```

```
import pandas
import couchdbkit

server = couchdbkit.Server('http://brenda:XXXXX@127.0.0.1:5984/')
tweetdb = server.get_db('tweets')
tweets = list(
    tweetdb.view(
        "tweetsPerDay/tweetsPerDay",
        reduce=True,
        group_level=3,
        startkey=[2011, 1, 1],
        endkey=[2012, 1, 1]))
date_list = [
    pandas.datetime(tweet["key"][0], (tweet["key"][1]), tweet["key"][2])
    for tweet in tweets]
date_index = pandas.DatetimeIndex(date_list)
data_list = [tweet["value"] for tweet in tweets]
science_tweets = pandas.Series(data_list, date_index)
# change to float so can have NaN values
science_tweets = science_tweets.astype(float)
# mask out the missing data period so it doesn't plot
science_tweets['2011-03-31':'2011-04-12'] = numpy.NaN
# final check that start and end dates are correct
print 'first element: ', science_tweets.first('1D')
print 'last element: ', science_tweets.last('1D')
# check than the missing values don't plot.
science_tweets.plot()
science_tweets.to_pickle('dataFiles/2011TweetsPerDay-final.pkl')
```

```
import pandas
import couchdbkit

server = couchdbkit.Server('http://brenda:XXXXX@127.0.0.1:5984/')
tweetdb = server.get_db('tweets')
tweets = list(
    tweetdb.view(
        "tweetsPerDay/tweetsPerDay",
        reduce=True,
        group_level=3,
        startkey=[2011, 1, 1],
        endkey=[2012, 1, 1]))
date_list = [
    pandas.datetime(tweet["key"][0], (tweet["key"][1]), tweet["key"][2])
    for tweet in tweets]
date_index = pandas.DatetimeIndex(date_list)
data_list = [tweet["value"] for tweet in tweets]
science_tweets = pandas.Series(data_list, date_index)
# change to float so can have NaN values
science_tweets = science_tweets.astype(float)
# mask out the missing data period so it doesn't plot
science_tweets['2011-03-31':'2011-04-12'] = numpy.NaN
# final check that start and end dates are correct
print 'first element: ', science_tweets.first('1D')
print 'last element: ', science_tweets.last('1D')
# check than the missing values don't plot.
science_tweets.plot()
science_tweets.to_pickle('dataFiles/2011TweetsPerDay-final.pkl')
```

```
# final check that start and end dates are correct
print 'first element: ', science_tweets.first('1D')
print 'last element: ', science_tweets.last('1D')
# check than the missing values don't plot.
science_tweets.plot()
science_tweets.to_pickle('dataFiles/2011TweetsPerDay-final.pkl')
```

```
first element:  2011-01-01    11070
dtype: float64
last element:  2011-12-31    25067
dtype: float64
```



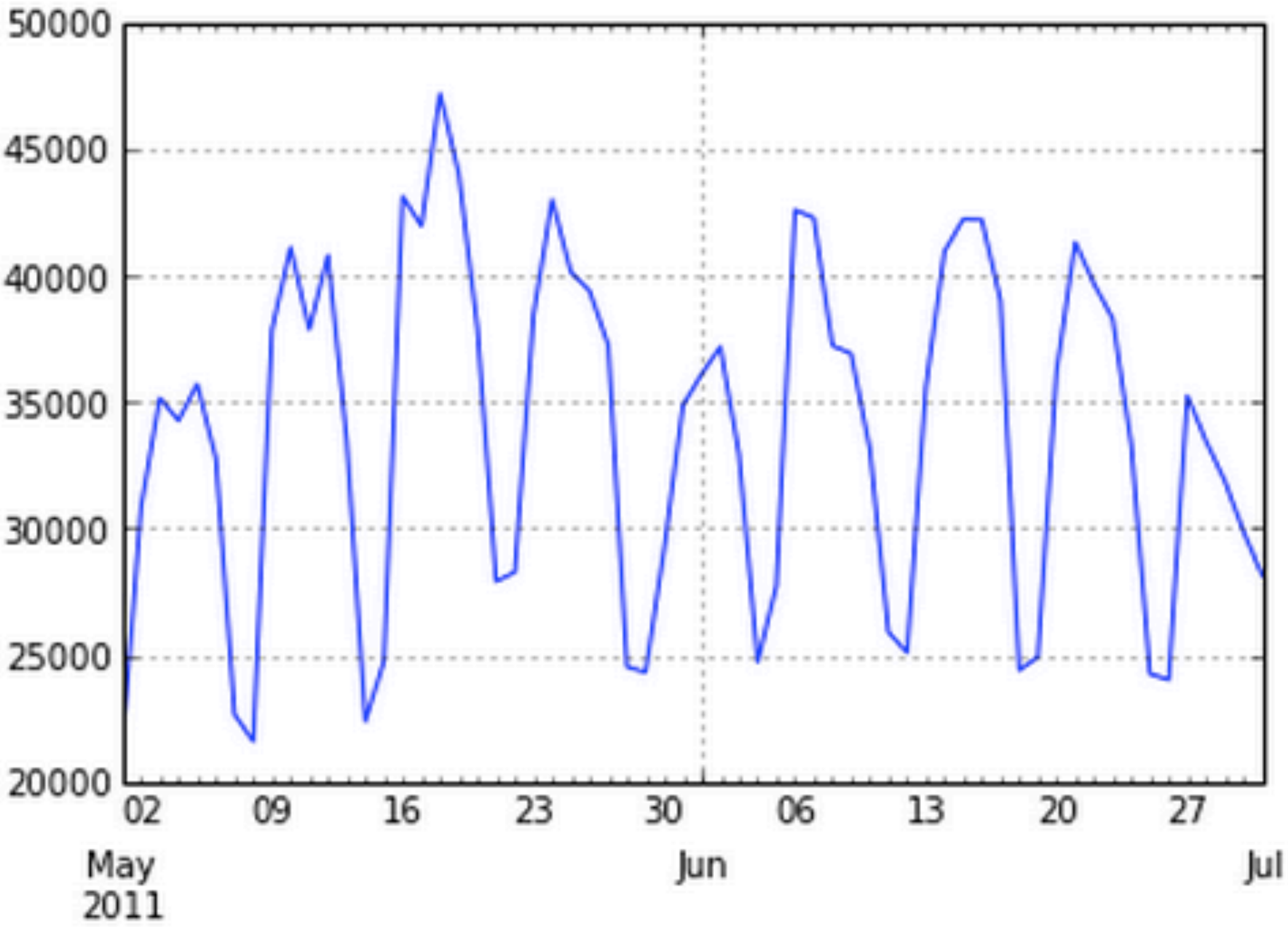
```python
# different ways to access the start of the timeseries
print "science_tweets.first('1D')", science_tweets.first('1D')
print '\nscience_tweets.head(1)', science_tweets.head(1)
print '\nscience_tweets[0]', science_tweets[0]
print "\nscience_tweets['2011-01-01']", science_tweets['2011-01-01']
print "\nscience_tweets.first('1W')", science_tweets.first('1W')
```

```
science_tweets.first('1D') 2011-01-01    11070
dtype: float64

science_tweets.head(1) 2011-01-01    11070
dtype: float64

science_tweets[0] 11070.0

science_tweets['2011-01-01'] 11070.0

science_tweets.first('1W') 2011-01-01    11070
2011-01-02    14542
dtype: float64
```

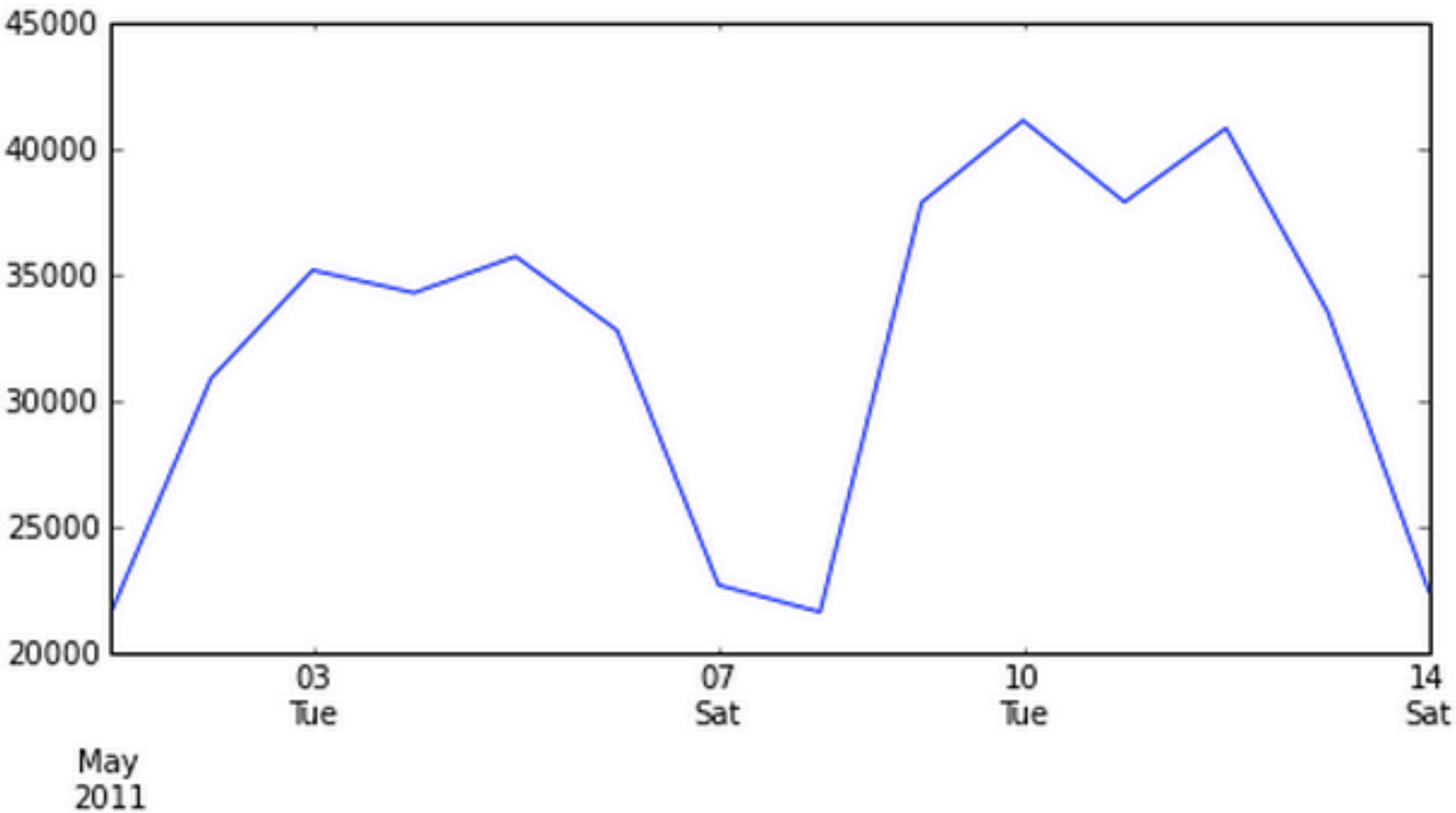
```
# ambiguity of date strings
print "\nscience_tweets['2011-07-01']", science_tweets['2011-07-01']
print "\nscience_tweets['2011-01-07']", science_tweets['2011-01-07']
print "\nscience_tweets[pandas.datetime(2011,1,7)]"
print science_tweets[pandas.datetime(2011,1,7)]
print "\npandas.to_datetime(['07-01-2011'], dayfirst=True)"
science_tweets[pandas.to_datetime(['07-01-2011'], dayfirst=True)]
```

```
science_tweets['2011-07-01'] 28146.0

science_tweets['2011-01-07'] 26095.0

science_tweets[pandas.datetime(2011,1,7)]
26095.0

pandas.to_datetime(['07-01-2011'], dayfirst=True)
2011-01-07    26095
dtype: float64
```

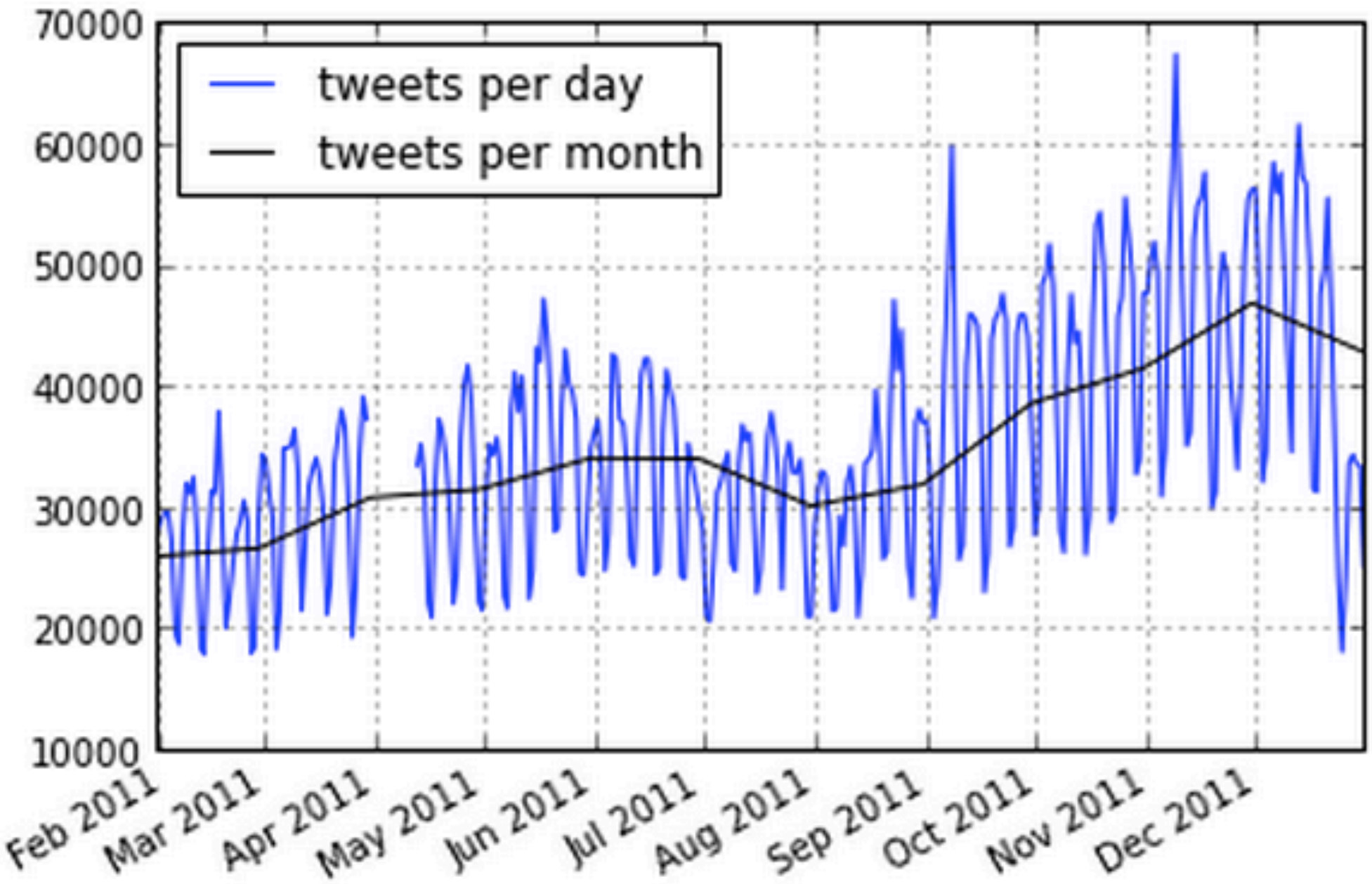
```
# zoom in using a date range
science_tweets['2011-05-01':'2011-07-01'].plot();
```



```
ax = plt.figure(figsize=(7,4), dpi=300).add_subplot(111)
two_weeks = science_tweets['2011-05-01':'2011-05-14']
ax.plot_date(two_weeks.index.to_pydatetime(), two_weeks, '-')
ax.xaxis.set_minor_locator(mpl.dates.WeekdayLocator(byweekday=(1,5),
                                                    interval=1))
ax.xaxis.set_minor_formatter(mpl.dates.DateFormatter('%d\n%a'))
ax.xaxis.set_major_locator(mpl.dates.MonthLocator())
ax.xaxis.set_major_formatter(mpl.dates.DateFormatter('\n\n\n%b\n%Y'))
plt.tight_layout()
plt.show()
```



```
monthly_tweets = science_tweets.resample('M', how='mean')
science_tweets.plot(label='tweets per day')
monthly_tweets.plot('tweets per month', color='k')
legend(loc=0);
```



```
science_tweets.describe()
```

```
count      352.000000
mean     34646.264205
std      10178.287440
min      11070.000000
25%      27626.750000
50%      33414.500000
75%      40963.500000
max      67375.000000
dtype: float64
```

```
df = pandas.DataFrame(data={'tweets per day':science_tweets,
                            'tweets per month':monthly_tweets})
df[28:33]
```

	tweets per day	tweets per month
2011-01-29	18913	NaN
2011-01-30	18651	NaN
2011-01-31	27853	25897.903226
2011-02-01	28808	NaN
2011-02-02	29517	NaN

```
df['cumulative'] = df['tweets per day'].cumsum()
total_tweets = df['tweets per day'].sum()
df['percent'] = df['tweets per day'].map(lambda x:
                                         x*100/total_tweets)
df['dayofweek'] = df.index.map(lambda x: x.dayofweek)
df.head()
```

	tweets per day	tweets per month	cumulative	percent	dayofweek
2011-01-01	11070	NaN	11070	0.090771	5
2011-01-02	14542	NaN	25612	0.119241	6
2011-01-03	24121	NaN	49733	0.197786	0
2011-01-04	25984	NaN	75717	0.213062	1
2011-01-05	27639	NaN	103356	0.226633	2

```
df.groupby('dayofweek').sum()
```

	tweets per day	tweets per month	cumulative	percent
dayofweek				
0	1833426	93883.930876	274073252	15.033646
1	2010642	33999.741935	276083894	16.486774
2	2084425	78640.716129	280636847	17.091776
3	2012464	64687.800000	280180783	16.501714
4	1758167	38536.800000	281938950	14.416540
5	1233825	74176.073477	283172775	10.117064
6	1262536	30059.516129	272239826	10.352487

```
df.groupby('dayofweek').sum()[['tweets per day', 'percent']]
```

	tweets per day	percent
dayofweek		
0	1833426	15.033646
1	2010642	16.486774
2	2084425	17.091776
3	2012464	16.501714
4	1758167	14.416540
5	1233825	10.117064
6	1262536	10.352487

```
df.groupby('dayofweek').sum()['percent'].sum()
```

```
100.0
```

```
print df.groupby('dayofweek').sum()['tweets per day'].sum()
print df['tweets per day'].sum()
```

```
12195485.0
12195485.0
```

```
df2 = df.groupby('dayofweek').sum()
print df2.to_latex(columns=['tweets per day', 'percent'])
```

```
\begin{tabular}{lrr}
\toprule
{} &  tweets per day &     percent \\
\midrule
dayofweek &                 &             \\
0         &         1833426 &   15.033646 \\
1         &         2010642 &   16.486774 \\
2         &         2084425 &   17.091776 \\
3         &         2012464 &   16.501714 \\
4         &         1758167 &   14.416540 \\
5         &         1233825 &   10.117064 \\
6         &         1262536 &   10.352487 \\
\bottomrule
\end{tabular}
```

Questions?

Slides and notebook available on GitHub

https://github.com/brendam/pycon2013talk