Pandas_and_Friends

February 23, 2015

1 Pandas and Friends

- Austin Godber
- Mail: godber@uberhip.com
- Twitter: @godber

- Presented at DesertPy, Jan 2015.

2 What does it do?

Pandas is a Python data analysis tool built on top of NumPy that provides a suite of data structures and data manipulation functions to work on those data structures. It is particularly well suited for working with time series data.

3 Getting Started - Installation

Installing with pip or apt-get::

```
pip install pandas
# or
sudo apt-get install python-pandas
```

- Mac - Homebrew or MacPorts to get the dependencies, then pip
- Windows - Python(x,y)?
- Commercial Pythons: Anaconda, Canopy

4 Getting Started - Dependencies

Dependencies, required, recommended and optional

```
# Required
numpy, python-dateutil, pytx
# Recommended
numexpr, bottleneck
# Optional
cython, scipy, pytables, matplotlib, statsmodels, openpyxl
```

5 Pandas' Friends!

Pandas works along side and is built on top of several other Python projects.

- IPython

- Numpy
- Matplotlib

5.1 Pandas gets along with EVERYONE!

6 Background - IPython

IPython is a fancy python console. Try running `ipython` or `ipython --pylab` on your command line. Some IPython tips

```
# Special commands, 'magic functions', begin with %
%quickref, %who, %run, %reset
# Shell Commands
ls, cd, pwd, mkdir
# Need Help?
help(), help(obj), obj?, function?
# Tab completion of variables, attributes and methods
```

7 Background - IPython Notebook

There is a web interface to IPython, known as the IPython notebook, start it like this

```
ipython notebook
# or to get all of the pylab components
ipython notebook --pylab
```

8 IPython - Follow Along

Follow along by connecting to TMPNB.ORG!

- http://tmpnb.org

9 Background - NumPy

- NumPy is the foundation for Pandas
- Numerical data structures (mostly Arrays)
- Operations on those.
- Less structure than Pandas provides.

10 Background - NumPy - Arrays

```
In [1]: import numpy as np
        # np.zeros, np.ones
        data0 = np.zeros((2, 4))

        data0

Out[1]: array([[ 0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.]])
```

```
In [2]: # Make an array with 20 entries 0..19
        data1 = np.arange(20)
        # print the first 8
        data1[0:8]

Out[2]: array([0, 1, 2, 3, 4, 5, 6, 7])
```

10.1 Background - NumPy - Arrays

```
In [3]: # make it a 4,5 array
        data = np.arange(20).reshape(4, 5)
        data

Out[3]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19]])
```

10.2 Background - NumPy - Arrays

Arrays have NumPy specific types, `dtypes`, and can be operated on.

```
In [4]: print "dtype: ", data.dtype
        result = data * 20.5
        print result

dtype:  int64
[[   0.    20.5   41.    61.5   82. ]
 [ 102.5  123.   143.5  164.   184.5]
 [ 205.   225.5  246.   266.5  287. ]
 [ 307.5  328.   348.5  369.   389.5]]
```

10.3 Now, on to Pandas

10.4 Pandas

- Tabular, Timeseries, Matrix Data - labeled or not
- Sensible handling of missing data and data alignment
- Data selection, slicing and reshaping features
- Robust data import utilities.
- Advanced time series capabilities

10.5 Data Structures

- Series - 1D labeled array
- DataFrame - 2D labeled array
- Panel - 3D labeled array (More D)

11 Assumed Imports

In my code samples, assume I import the following

```
In [5]: import pandas as pd
        import numpy as np
```

12 Series

- one-dimensional labeled array
- holds any data type
- axis labels known as index
- implicit integert indexes
- `dict`-like

13 Create a Simple Series

```
In [6]: s1 = pd.Series([1, 2, 3, 4, 5])
        s1

Out[6]: 0    1
        1    2
        2    3
        3    4
        4    5
        dtype: int64
```

14 Series Operations

```
In [7]: # integer multiplication
        print s1 * 5

0     5
1    10
2    15
3    20
4    25
dtype: int64
```

15 Series Operations - Cont.

```
In [8]: # float multiplication
        print s1 * 5.0

0     5
1    10
2    15
3    20
4    25
dtype: float64
```

16 Series Index

```
In [9]: s2 = pd.Series([1, 2, 3, 4, 5],
                       index=['a', 'b', 'c', 'd', 'e'])
        s2

Out[9]: a    1
        b    2
```

```
c    3
d    4
e    5
dtype: int64
```

17 Date Convenience Functions

A quick aside ...

```
In [10]: dates = pd.date_range('20130626', periods=5)
         print dates
         print
         print dates[0]
```

```
<class 'pandas.tseries.index.DatetimeIndex'>
[2013-06-26, ..., 2013-06-30]
Length: 5, Freq: D, Timezone: None

2013-06-26 00:00:00
```

18 Datestamps as Index

```
In [11]: s3 = pd.Series([1, 2, 3, 4, 5], index=dates)
         print s3
```

```
2013-06-26    1
2013-06-27    2
2013-06-28    3
2013-06-29    4
2013-06-30    5
Freq: D, dtype: int64
```

19 Selecting By Index

Note that the integer index is retained along with the new date index.

```
In [12]: print s3[0]
         print type(s3[0])
         print
         print s3[1:3]
         print type(s3[1:3])
```

```
1
<type 'numpy.int64'>

2013-06-27    2
2013-06-28    3
Freq: D, dtype: int64
<class 'pandas.core.series.Series'>
```

20 Selecting by value

```
In [13]: s3[s3 < 3]
```

```
Out[13]: 2013-06-26    1
         2013-06-27    2
         Freq: D, dtype: int64
```

21 Selecting by Label (Date)

```
In [14]: s3['20130626':'20130628']
```

```
Out[14]: 2013-06-26    1
         2013-06-27    2
         2013-06-28    3
         Freq: D, dtype: int64
```

21.1 Series Wrapup

Things not covered but you should look into:

- Other instantiation options: `dict`
- Operator Handling of missing data `NaN`
- Reforming Data and Indexes
- Boolean Indexing
- Other Series Attributes:
- `index` - `index.name`
- `name` - Series name

21.2 DataFrame

- 2-dimensional labeled data structure
- Like a SQL Table, Spreadsheet or `dict` of `Series` objects.
- Columns of potentially different types
- Operations, slicing and other behavior just like `Series`

22 DataFrame - Simple

```
In [15]: data1 = pd.DataFrame(np.random.rand(4, 4))
         data1
```

```
Out[15]:          0         1         2         3
         0  0.371333  0.788792  0.869380  0.084323
         1  0.490858  0.134202  0.816307  0.019157
         2  0.839547  0.721638  0.544628  0.042547
         3  0.098594  0.954038  0.306561  0.689759
```

23 DataFrame - Index/Column Names

```
In [16]: dates = pd.date_range('20130626', periods=4)
         data2 = pd.DataFrame(
             np.random.rand(4, 4),
             index=dates, columns=list('ABCD'))
         data2
```

Out[16]:

	A	B	C	D
2013-06-26	0.572954	0.785437	0.089758	0.872083
2013-06-27	0.857868	0.779294	0.453022	0.836332
2013-06-28	0.715369	0.355922	0.750194	0.770045
2013-06-29	0.409056	0.452993	0.937368	0.118998

24 DataFrame - Operations

```
In [17]: data2['E'] = data2['B'] + 5 * data2['C']
         data2
```

Out[17]:

	A	B	C	D	E
2013-06-26	0.572954	0.785437	0.089758	0.872083	1.234225
2013-06-27	0.857868	0.779294	0.453022	0.836332	3.044402
2013-06-28	0.715369	0.355922	0.750194	0.770045	4.106891
2013-06-29	0.409056	0.452993	0.937368	0.118998	5.139832

See? You never need Excel again!

25 DataFrame - Column Access

Deleting a column.

```
In [18]: # Deleting a Column
         del data2['E']
         data2
```

Out[18]:

	A	B	C	D
2013-06-26	0.572954	0.785437	0.089758	0.872083
2013-06-27	0.857868	0.779294	0.453022	0.836332
2013-06-28	0.715369	0.355922	0.750194	0.770045
2013-06-29	0.409056	0.452993	0.937368	0.118998

26 DataFrame

Remember this, data2, for the next examples.

```
In [19]: data2
```

Out[19]:

	A	B	C	D
2013-06-26	0.572954	0.785437	0.089758	0.872083
2013-06-27	0.857868	0.779294	0.453022	0.836332
2013-06-28	0.715369	0.355922	0.750194	0.770045
2013-06-29	0.409056	0.452993	0.937368	0.118998

27 DataFrame - Column Access

As a dict

```
In [20]: data2['B']
```

```
Out[20]: 2013-06-26    0.785437
         2013-06-27    0.779294
         2013-06-28    0.355922
         2013-06-29    0.452993
         Freq: D, Name: B, dtype: float64
```

28 DataFrame - Column Access

As an attribute

```
In [21]: data2.B
```

```
Out[21]: 2013-06-26    0.785437
         2013-06-27    0.779294
         2013-06-28    0.355922
         2013-06-29    0.452993
         Freq: D, Name: B, dtype: float64
```

29 DataFrame - Row Access

By row label

```
In [22]: data2.loc['20130627']
```

```
Out[22]: A    0.857868
         B    0.779294
         C    0.453022
         D    0.836332
         Name: 2013-06-27 00:00:00, dtype: float64
```

30 DataFrame - Row Access

By integer location

```
In [23]: data2.iloc[1]
```

```
Out[23]: A    0.857868
         B    0.779294
         C    0.453022
         D    0.836332
         Name: 2013-06-27 00:00:00, dtype: float64
```

31 DataFrame - Cell Access

Access column, then row or use iloc and row/column indexes.

```
In [24]: print data2.B[0]
         print data2['B'][0]
         print data2.iloc[0,1]  # [row,column]
```

```
0.785436968548
0.785436968548
0.785436968548
```

32 DataFrame - Taking a Peek

Look at the beginning of the DataFrame

```
In [25]: data3 = pd.DataFrame(np.random.rand(100, 4))
         data3.head()
```

Out[25]:

	0	1	2	3
0	0.127258	0.981462	0.820096	0.650613
1	0.471623	0.118745	0.595012	0.205356
2	0.802777	0.398816	0.383789	0.025648
3	0.368724	0.922648	0.850099	0.659503
4	0.889618	0.565936	0.056413	0.768219

33 DataFrame - Taking a Peek

Look at the end of the DataFrame.

```
In [26]: data3.tail()
```

Out[26]:

	0	1	2	3
95	0.332815	0.920505	0.808580	0.161875
96	0.348941	0.532944	0.921147	0.736528
97	0.305776	0.747903	0.161359	0.808279
98	0.141267	0.878946	0.537137	0.157560
99	0.862024	0.519265	0.172454	0.665810

34 DataFrame Wrap Up

Just remember,

- A `DataFrame` is just a bunch of `Series` grouped together.
- Any one dimensional slice returns a `Series`
- Any two dimensional slice returns another `DataFrame`.
- Elements are typically NumPy types or Objects.

35 Panel

Like DataFrame but 3 or more dimensions.

36 IO Tools

Robust IO tools to read in data from a variety of sources

- CSV - pd.read_csv()
- Clipboard - pd.read_clipboard()
- SQL - pd.read_sql_table()
- Excel - pd.read_excel()

37 Plotting

- Matplotlib - s.plot() - Standard Python Plotting Library
- Trellis - rplot() - An 'R' inspired Matplotlib based plotting tool

38 Bringing it Together - Data

The csv file (`phx-temps.csv`) contains Phoenix weather data from GSOD::

```
1973-01-01 00:00:00,53.1,37.9
1973-01-02 00:00:00,57.9,37.0
...
2012-12-30 00:00:00,64.9,39.0
2012-12-31 00:00:00,55.9,41.0
```

39 Bringing it Together - Code

Simple `read_csv()`

```
In [27]: # simple readcsv
         phxtemps1 = pd.read_csv('phx-temps.csv')
         phxtemps1.head()
```

```
Out[27]:    1973-01-01 00:00:00  53.1  37.9
         0  1973-01-02 00:00:00  57.9  37.0
         1  1973-01-03 00:00:00  59.0  37.0
         2  1973-01-04 00:00:00  57.9  41.0
         3  1973-01-05 00:00:00  54.0  39.9
         4  1973-01-06 00:00:00  55.9  37.9
```

40 Bringing it Together - Code

Advanced `read_csv()`, parsing the dates and using them as the index, and naming the columns.

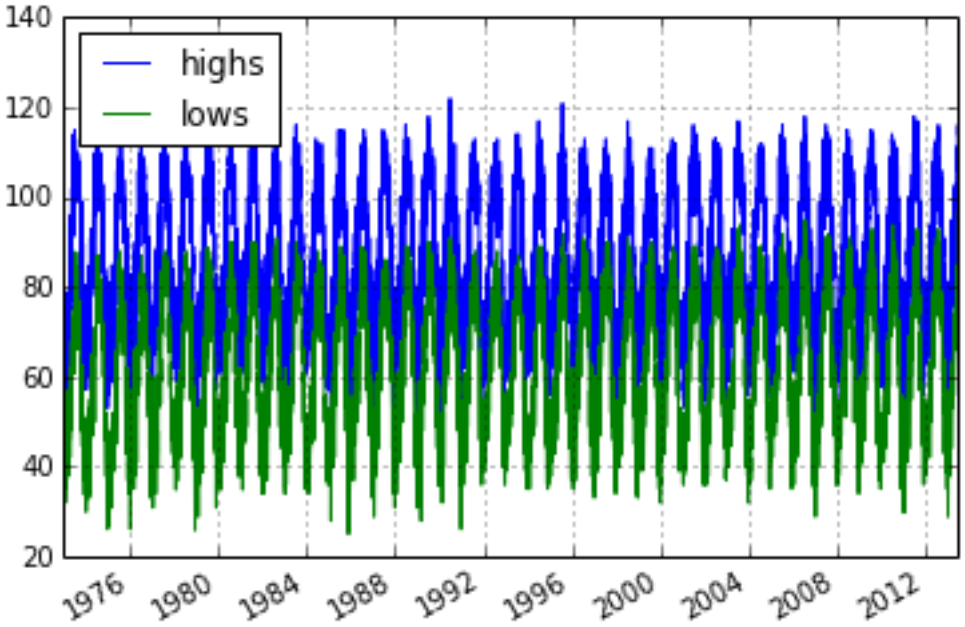
```
In [28]: # define index, parse dates, name columns
         phxtemps2 = pd.read_csv(
             'phx-temps.csv', index_col=0,
             names=['highs', 'lows'], parse_dates=True)
         phxtemps2.head()
```

```
Out[28]:             highs  lows
         1973-01-01   53.1  37.9
         1973-01-02   57.9  37.0
         1973-01-03   59.0  37.0
         1973-01-04   57.9  41.0
         1973-01-05   54.0  39.9
```

41 Bringing it Together - Plot

```
In [29]: import matplotlib.pyplot as plt
         %matplotlib inline
         phxtemps2.plot()  # pandas convenience method
```

```
Out[29]: <matplotlib.axes.AxesSubplot at 0x7f9916ef4d90>
```



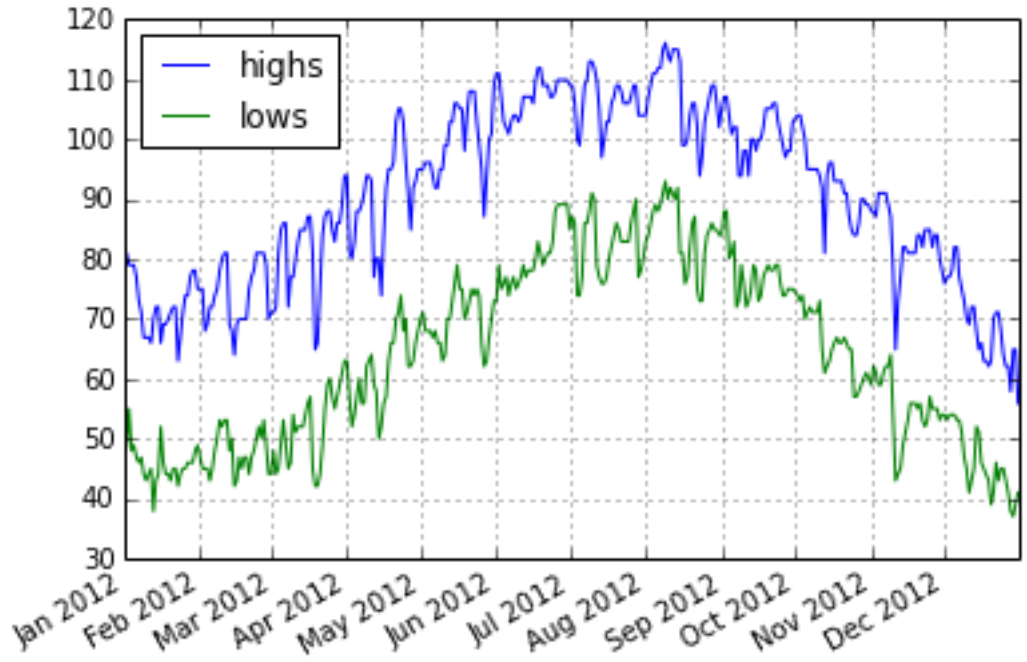
Boo, Pandas and Friends would cry if they saw such a plot.

42 Bringing it Together - Plot

Lets see a smaller slice of time:

```
In [30]: phxtemps2['20120101':'20121231'].plot()
```

```
Out[30]: <matplotlib.axes.AxesSubplot at 0x7f9916ae8290>
```

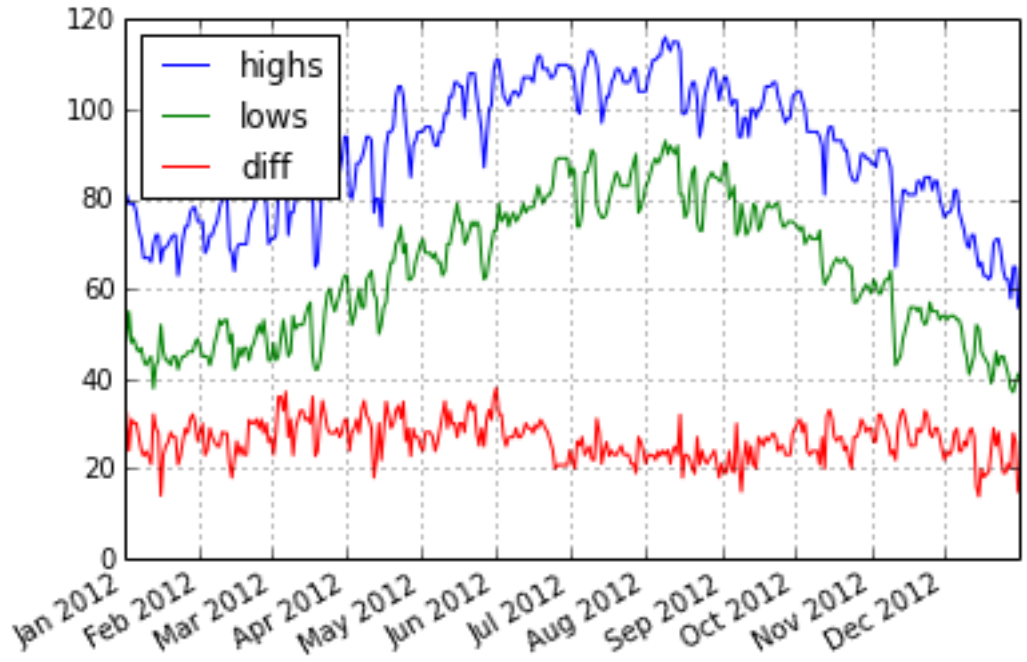


43 Bringing it Together - Plot

Lets operate on the `DataFrame` ... lets take the differnce between the highs and lows.

```
In [31]: phxtemps2['diff'] = phxtemps2.highs - phxtemps2.lows
         phxtemps2['20120101':'20121231'].plot()
```

```
Out[31]: <matplotlib.axes.AxesSubplot at 0x7f99168bec10>
```



44 Pandas Alternatives

- AstroPy seems to have similar data structures.
- I suspect there are others.

44.1 References

- Pandas Documentation
- Python for Data Analysis
- Presentation Source

45 Thanks! - Pandas and Friends

- Austin Godber
- Mail: godber@uberhip.com
- Twitter: @godber
- Presented at DesertPy, Jan 2015.