PANDAS PART 1

Feb 27, 2015 – General Assembly, Santa Monica

Mohsen Chitsaz, Ph.D.

Sr. Principal Data Scientist at Symantec

Python Data Analysis Library

Introduction to Pandas

- Wes McKinney started the project in **2008**
- Data structures with labeled axes supporting automatic or explicit **data alignment**
- **Data aggregation** (like summing across an axis)
- Flexible handling of **missing data**
- **SQL-like** operations on the data, similar to databases

Getting started with Pandas

Data Structures

- **Series**
  - A **one-dimensional** array-like object containing data and label

```python
s = pd.Series(data=[1,2,3,4,5],\
              index=['first element','second element',\
                     'third','4th','5th element'])
```

- **DataFrame**
  - A tabular, **spreadsheet-like** data structure containing an ordered collection of columns, each of which can be a different value type.

```python
df = pd.DataFrame({'state':['California','New York','Texas'],\
                   'Party':['D','D','R']})
```

Reindexing (Series)

- **Rearranges** the data according to the **new index**, introducing **missing values**, if any index value were not already present.

```
obj
b      6.5
d      7.2
c     -5.3
dtype: float64

obj2
a      NaN
b      6.5
c     -5.3
d      7.2
dtype: float64
```

```
obj = pd.Series(data=[6.5, 7.2, -5.3],\
                index=['b','d','c'])
obj2 = obj.reindex(['a','b','c','d'])
```

Reindexing (DataFrames)

```python
frame = pd.DataFrame(data=np.arange(9).reshape((3,3)),\
                     index=['a','c','d'],\
                     columns=['Ohio','Texas','California'])
print frame
```

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

```python
frame2 = frame.reindex(['a','b','c','d'])
print frame2
```

	Ohio	Texas	California
a	0	1	2
b	NaN	NaN	NaN
c	3	4	5
d	6	7	8

Dropping entries from an axis (Series)

□ **"*drop*"** method returns a **new** object with the indicated value(s) deleted from an axis:

```python
obj = pd.Series(data=[1,2,3,4],\
                index=['a','b','c','d'])
obj2 = obj.drop(['a','c'])
```

```
obj                 obj2
a       1           b       2
b       2           d       4
c       3           dtype: int64
d       4
dtype: int64
```

Dropping entries from an axis (DataFrames)

```python
data = pd.DataFrame(np.arange(16).reshape((4,4)),\
                    index=['Ohio','Colorado','Utah','New York'],\
                    columns=['one','two','three','four'])
data2 = data.drop(['Colorado','Ohio'])
```

```
data
          one   two   three   four
Ohio        0     1       2      3
Colorado    4     5       6      7
Utah        8     9      10     11
New York   12    13      14     15
```

```
data2
           one   two   three   four
Utah         8     9      10     11
New York    12    13      14     15
```

Indexing, selection and filtering

Type	Description
`Obj[col_name(s)]`	Select column(s)
`Obj.ix[row(s)]`	Select row(s)
`Obj.ix[row(s), col(s)]`	Select row(s) & col(s)

Indexing, selection and filtering

```python
data = pd.DataFrame(np.arange(16).reshape((4,4)),\
                    index=['Ohio','Colorado','Utah','New York'],\
                    columns=['one','two','three','four'])
```

```python
data[['one','two']]
```

```python
data.ix[['Ohio','Colorado'],['two','three']]
```

	one	two
Ohio	0	1
Colorado	4	5
Utah	8	9
New York	12	13

	two	three
Ohio	1	2
Colorado	5	6

Indexing, selection and filtering

```
rows = [0,1] #rows = ['Ohio','Colorado']
cols = [2] #cols = ['three']
data.ix[rows, cols]
```

	three
Ohio	2
Colorado	6

```
rows = [0,1] #rows = ['Ohio','Colorado']
cols = [2] #cols = ['three']
data.ix[:, cols]
```
?

Indexing, selection and filtering

```
data[data['three']>5]
```

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data[data < 5]=0
print data
```

```
          one   two   three   four
Ohio        0     0       0      0
Colorado    0     5       6      7
Utah        8     9      10     11
New York   12    13      14     15
```

```
data.ix[data.three>5,0:3]
```

	one	two	three
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

```
data.ix[data.three>5,0:3] = -1
print data
```

```
          one   two   three   four
Ohio        0     1       2      3
Colorado   -1    -1      -1      7
Utah       -1    -1      -1     11
New York   -1    -1      -1     15
```

Arithmetic and data alignment

□ **df1**+**df2**

□ **df1**.add(**df2**, **fill_value**=0)

```python
df1 = pd.DataFrame(np.arange(9).reshape(3,3),\
                   columns=list('abc'),\
                   index=range(3))

df2 = pd.DataFrame(np.arange(16).reshape(4,4),\
                   columns=list('abcd'),\
                   index=range(4))
```

```
df1+df2
     a    b    c    d
0    0    2    4  NaN
1    7    9   11  NaN
2   14   16   18  NaN
3  NaN  NaN  NaN  NaN
```

```
df1.add(df2,fill_value=0)
     a    b    c    d
0    0    2    4    3
1    7    9   11    7
2   14   16   18   11
3   12   13   14   15
```

Function application and mapping

- **`df.apply(f, axis)`**
  - Applies function *"f"* on DataFrame *"df"* on axis *"axis"* hence, reducing the dimension of the DataFrame by +1.

- **`df.applymap(f)`**
  - Applies function *"f"* on every single element of the DataFrame

Function application and mapping

```python
df = pd.DataFrame(np.random.randn(4,3),
                  columns=list('bde'),
                  index=['Utah','Ohio','Texas','Oregon'])
print df
```

```
               b         d         e
Utah    -1.188977  0.415385  1.703405
Ohio    -1.443891 -2.830466  0.450911
Texas    1.167887 -2.447571 -0.239753
Oregon   0.319426  1.838542 -0.059893
```

```python
f = lambda x: '%.2f' % x
print df.applymap(f)
```

```
            b      d      e
Utah    -1.19   0.42   1.70
Ohio    -1.44  -2.83   0.45
Texas    1.17  -2.45  -0.24
Oregon   0.32   1.84  -0.06
```

Function application and mapping

```
f2 = lambda x: x.max()-x.min()
```

```
df.apply(f2, axis=0)
```

```
b    2.611778
d    4.669008
e    1.943158
```

```
df.apply(f2, axis=1)
```

```
Utah      2.892381
Ohio      3.281378
Texas     3.615457
Oregon    1.898435
```

Sorting and ranking

- **DataFrame**
  - *sort_index(by=[col1, col2])*
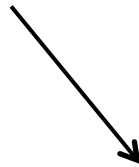 - *sort_index(axis=1, ascending=False)*

- **Series**
  - *order()*

Sorting and ranking

```python
df = pd.DataFrame({'b':[4,7,-3,2,5],
                   'a':[0,1,1,0,-1]})
```

```
df
    a   b
0   0   4
1   1   7
2   1  -3
3   0   2
4  -1   5
```

```python
print df.sort_index(by=['a','b'])
```

```
    a   b
4  -1   5
3   0   2
0   0   4
2   1  -3
1   1   7
```

Missing data

- **data.dropna()**
  - Drops the NaN rows

- **data.fillna(val)**
  - Fills the NaN elements with value "val"

- **data.isnull()**
  - Returns a DataFrame with the same size but boolean values, if each element is null

- **data.notnull()**
  - Returns a DataFrame with the same size but boolean values, if each element is not null

Correlation

```python
start = datetime.datetime(2000,1,1)
stop = datetime.datetime(2014,1,1)
a = pd.io.data.get_data_yahoo(['YHOO','ORCL'],start,stop)
print a.High.ORCL.corr(a.High.YHOO)
```

```
0.687492387025
```

```python
print a.Volume.ORCL.corr(a.Volume.YHOO)
```

```
0.133853803954
```
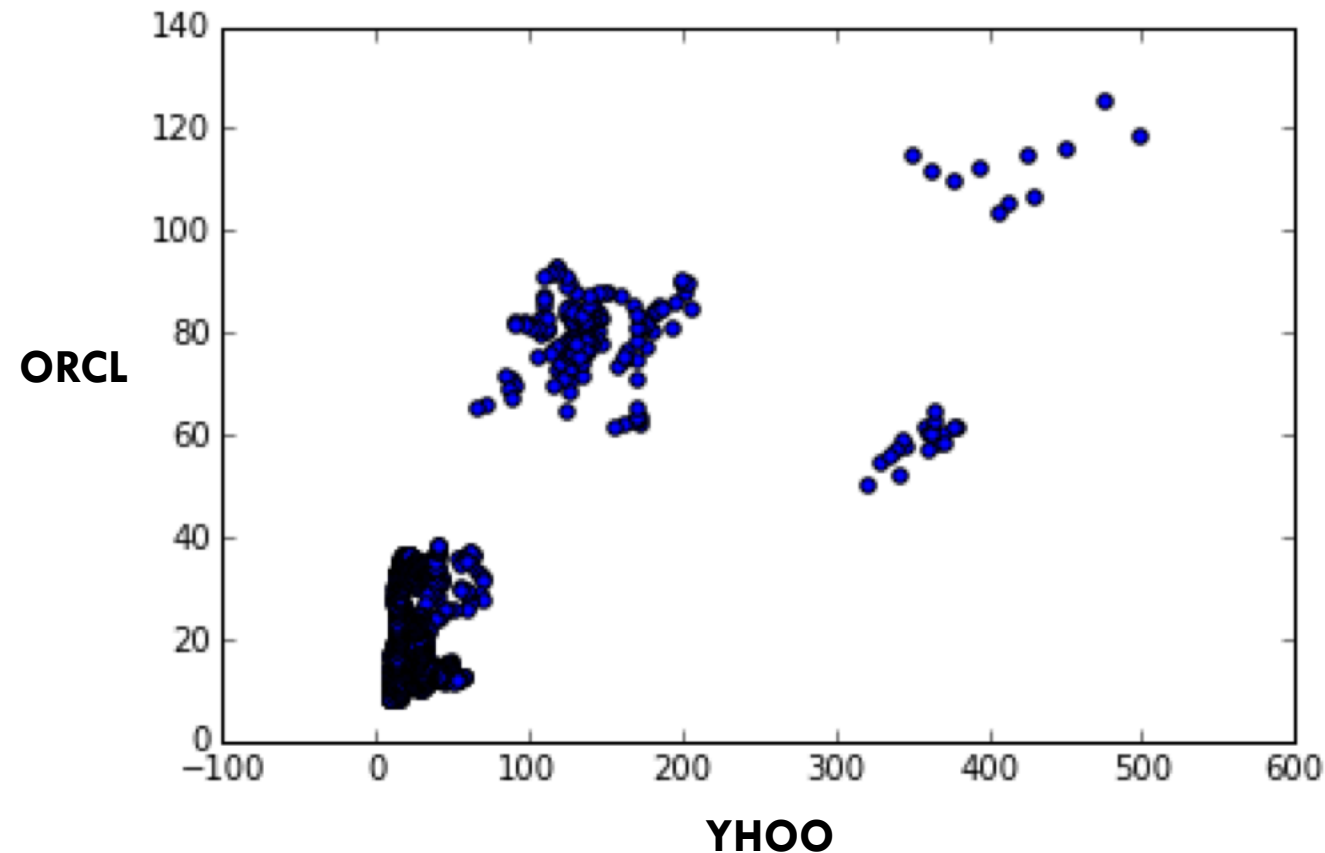
Plotting Series

```
a.High.YHOO.plot()
```

`<matplotlib.axes._subplots.AxesSubplot at 0x10e766e10>`
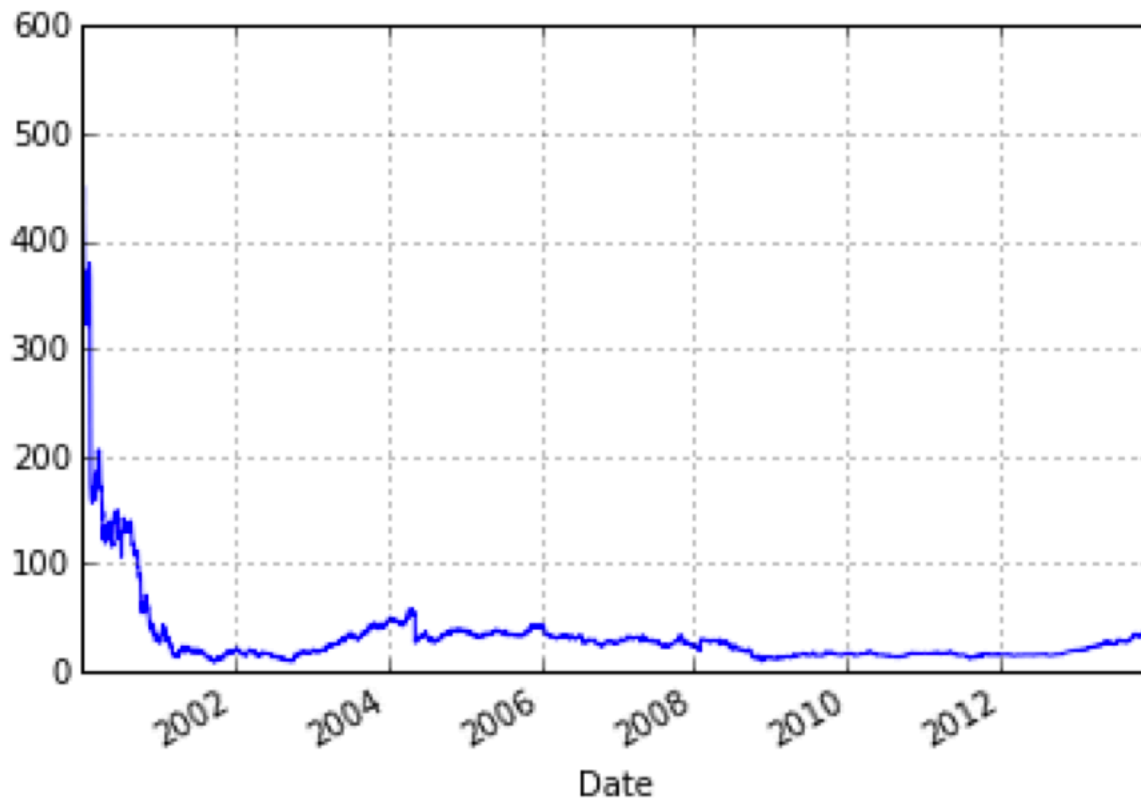
Scatter Plot of two Series

```
plt.scatter(a.High.YHOO,a.High.ORCL)
```

`<matplotlib.collections.PathCollection at 0x10eda84d0>`

Data Loading, Storage and File Formats

Reading and writing to file

- **read_csv**('file_name.csv', nrows=10)
  - Reads the top 10 rows of a csv file

- **to_csv**('file_name.csv', sep='|', na_rep='NULL')
  - Writes a DataFrame object into a file, using the separator *'sep'* and replacing NaN values with *'na_rep'*
  - *sys.stdout* as the name of the file will direct python to write to console as the output

Reading Excel files

- xls_file = ExcelFile('sample.xls')

- table = xls_file.parse('Sheet1')

```
xl = pd.ExcelFile('Curriculum.xlsx')
sylabus = xl.parse('Sylab')
print sylabus.head()
```

```
   Lecture#        Date                                      Topics
0         1  2015-02-11                         Intro to Data Science
1         2  2015-02-13                             Intro to Python
2         3  2015-02-18                                      Numpy
3         4  2015-02-20                                  SQL/Python
4         5  2015-02-25  Pandas - 1. Getting Started, 2. Data Loading, ...
```

```
xl = pd.read_excel('Curriculum.xlsx')
print xl.head()
```

```
   Lecture#        Date                     Topics
0         1  2015-02-11  Intro to Data Science
1         2  2015-02-13      Intro to Python
```

Interacting with HTML and APIs

```python
import requests
url = 'https://api.github.com/repos/pydata/pandas/milestones/28/labels'
resp = requests.get(url)
data = resp.json()
print data
```

[{u'url': u'https://api.github.com/repos/pydata/pandas/labels/Bug', u'colo
pi.github.com/repos/pydata/pandas/labels/Enhancement', u'color': u'4E9A06
thub.com/repos/pydata/pandas/labels/Refactor', u'color': u'FCE94F', u'name
pos/pydata/pandas/labels/Build%20problem', u'color': u'75507B', u'name': u
epos/pydata/pandas/labels/Docs', u'color': u'3465A4', u'name': u'Docs'}, {
labels/Groupby', u'color': u'729FCF', u'name': u'Groupby'}, {u'url': u'htt
0IO', u'color': u'06909A', u'name': u'Data IO'}, {u'url': u'https://api.g
'color': u'8AE234', u'name': u'Visualization'}, {u'url': u'https://api.gi

Interacting with Databases

```python
import sqlite3
query = """CREATE TABLE test (a VARCHAR(20),
        b VARCHAR(20), c REAL, d INTEGER);"""
con = sqlite3.connect(':memory:')
con.execute(query)
con.commit()
data = [('Atlanta','Georgia',1.25,6),\
        ('Sacramento','California',1.7,5),\
        ('Los Angeles','California',1.5,3)]
stmt = "INSERT INTO test VALUES(?,?,?,?)"
con.executemany(stmt, data)
con.commit()
```

Interacting with Databases

Traditional way:

```python
cursor = con.execute('select * from test')
rows = cursor.fetchall()
for row in rows:
    print row
```

```
(u'Atlanta', u'Georgia', 1.25, 6)
(u'Sacramento', u'California', 1.7, 5)
(u'Los Angeles', u'California', 1.5, 3)
```

Pandas way:

```python
print pd.read_sql('select * from test',con)
```

```
             a            b     c  d
0       Atlanta      Georgia  1.25  6
1    Sacramento   California  1.70  5
2   Los Angeles   California  1.50  3
```

Interacting with Databases

`pd.io.sql.*`

http://pandas.pydata.org/
pandas-docs/dev/io.html

Clean, Transform, Merge, Reshape

Combining Datasets

- **pandas.merge**
  - **Connecting rows based on a key or keys**
  - **Similar to '*join*' in SQL**

- **pandas.concat**
  - **Glues or stacks together objects along an axis**

- **combine_first**
  - **Enables splicing together overlapping data to fill in missing values in one object with values from another.**

Database-style DataFrame Merges

```python
df1 = pd.DataFrame({'key':list('bbacaab'),\
                        'data1':range(7)})
df2 = pd.DataFrame({'key':list('abd'),\
                        'data2':range(3)})
print 'df1\n',df1
print 'df2\n',df2
print pd.merge(df1, df2)
```

df1	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

df2	data2	key
0	0	a
1	1	b
2	2	d

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

Database-style DataFrame Merges

```
pd.merge(df1,
         df2,
         on='key1',
         how='inner')
```

Merges the two dataframes on key '*key*' using the "how" argument. In this case it is *inner* join.

Concatenating along an axis

```
df1 = pd.DataFrame(np.random.randn(3,4), columns=['a','b','c','d'])
df2 = pd.DataFrame(np.random.randn(2,3), columns=['b','d','a'])
print 'df1\n',df1
print 'df2\n',df2
print pd.concat([df1, df2])
```

Concatenation

```
df1
            a          b          c          d
0 -0.811979   0.387790   0.134205 -0.425209
1  0.569688 -0.406175   0.553706 -1.318497
2 -0.477620 -0.077196 -0.663000 -0.396227

df2
            b          d          a
0 -0.533802 -1.283319   0.196668
1 -0.271687 -1.788313   1.336632
```



```
            a          b          c          d
0 -0.811979   0.387790   0.134205 -0.425209
1  0.569688 -0.406175   0.553706 -1.318497
2 -0.477620 -0.077196 -0.663000 -0.396227
0  0.196668 -0.533802               NaN -1.283319
1  1.336632 -0.271687               NaN -1.788313
```

Removing duplicates

```python
data = pd.DataFrame({'k1':['one']*3+['two']*4,\
                     'k2':[1,1,2,3,3,4,4]})
print 'data\n',data
print 'data_depulicated\n',data.drop_duplicates()
```

```
data
      k1   k2
0    one    1
1    one    1
2    one    2
3    two    3
4    two    3
5    two    4
6    two    4
```

```
data_depulicated
      k1   k2
0    one    1
2    one    2
3    two    3
5    two    4
```

Replacing Values

```python
data = pd.Series([1, -999, 2, -999, -1000, 3])
print 'data\n',data
data_replaced = data.replace(-999,np.nan)
print 'data_replaced\n',data_replaced
```

```
data
0        1
1     -999
2        2
3     -999
4    -1000
5        3
dtype: int64
```

→

```
data_replaced
0        1
1      NaN
2        2
3      NaN
4    -1000
5        3
dtype: float64
```

Group by

```python
import pandas as pd
import numpy as np
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar','foo', 'bar', 'foo', 'foo'],
                   'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
                   'C' : np.random.randn(8), 'D' : np.random.randn(8)})
```

```python
grouped = df.groupby(['A'])
```

```python
grouped.groups
```

```
{'bar': [1, 3, 5], 'foo': [0, 2, 4, 6, 7]}
```

Aggregation

```python
import pandas as pd
import numpy as np
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar','foo', 'bar', 'foo', 'foo'],
                   'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
                   'C' : np.random.randn(8), 'D' : np.random.randn(8)})
```

```python
grouped = df.groupby(['A'])
```

```python
print grouped.agg(np.sum)
```

```
            C           D
A
bar   0.212190 -0.235641
foo   2.657326 -1.143714
```

Plotting

http://matplotlib.org/gallery.html