Lect 17 – pandas

Rob Capra

INLS 490-172

- We are going to talk about pandas today....

- But first, a small aside about floating point numbers...

Try this...

```
In [86]: a = 0

In [87]: for i in range(10):
    ...:     a += 0.1
    ...:

In [88]: a
Out[88]: 0.9999999999999999
```

What?!?!?!?

Try this...

```
In [86]: a = 0

In [87]: for i in range(10):
    ...:     a += 0.1
    ...:

In [88]: a
Out[88]: 0.9999999999999999
```

What?!?!?!?

```
In [91]: 0.1 + 0.2
Out[91]: 0.30000000000000004
```

Uhm... is Python broken?

Should we go back to IDLE?

Floating Point Numbers

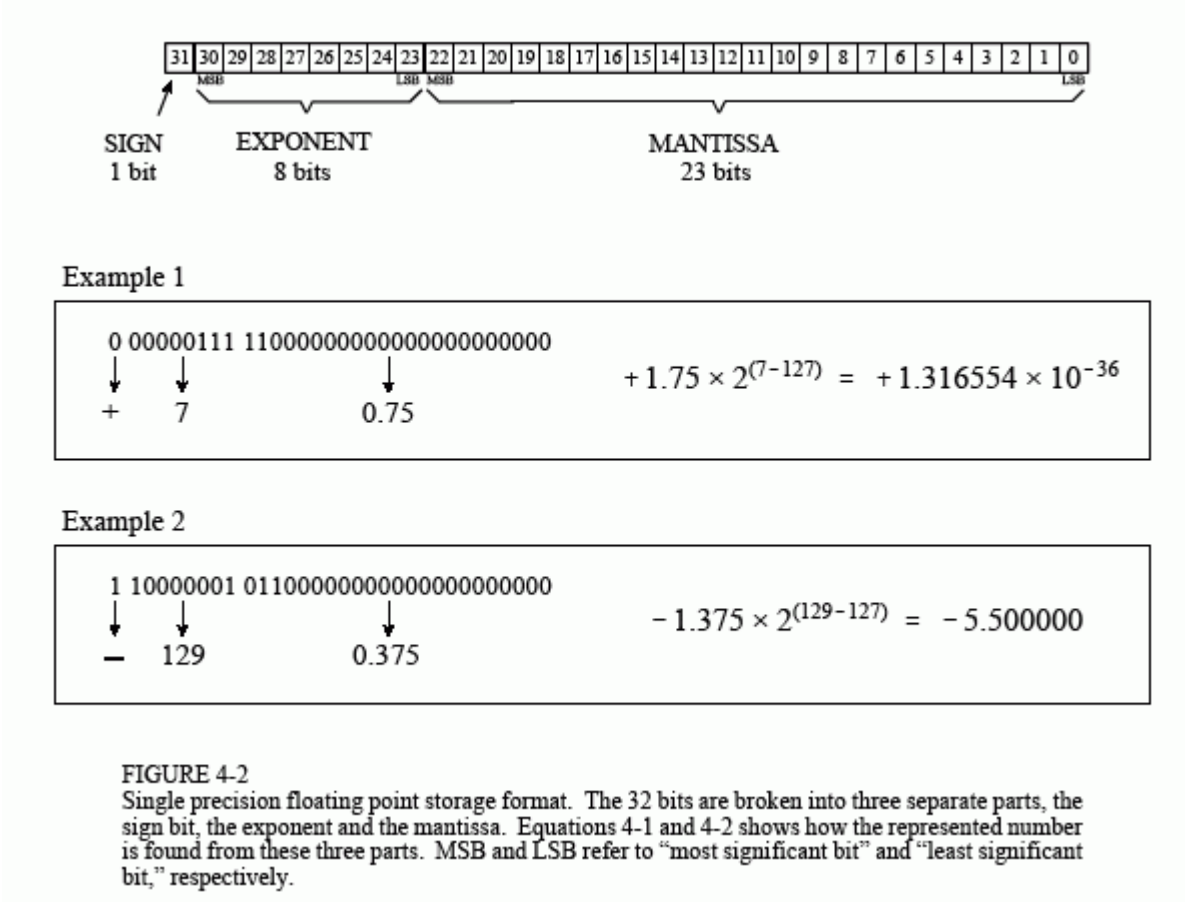


FIGURE 4-2
Single precision floating point storage format. The 32 bits are broken into three separate parts, the sign bit, the exponent and the mantissa. Equations 4-1 and 4-2 shows how the represented number is found from these three parts. MSB and LSB refer to "most significant bit" and "least significant bit," respectively.

Python, along with most other high-level programming languages and the chips on your computer, represent floating-point numbers as base 2 (binary) fractions.

from: http://www.dspguide.com/ch4/3.htm

Floating Point Numbers

- Python, along with most other programming languages and the chips on your computer, represent floating-point numbers as base 2 (binary) fractions.

- Unfortunately, many decimal fractions cannot be represented *exactly* as binary fractions.

- Thus, the decimal FP numbers we enter are stored as binary FP numbers that are *very close approximations*.

from: https://docs.python.org/2/tutorial/floatingpoint.html

Floating Point Numbers

- Actually, we have this problem with decimal fractions also.

- Consider 1/3:
    - 0.3, 0.333, 0.3333333333333333
    - all are approximations

- With binary fractions, the decimal value 0.1 has the same problem.

- Good news – typically off by no more than 1 part in 2^{53}

from: https://docs.python.org/2/tutorial/floatingpoint.html

format

```
In [93]: a = 0.1 + 0.2

In [94]: a
Out[94]: 0.30000000000000004

In [95]: print ("{0:.2f}".format(a))
0.30
```

https://docs.python.org/2/library/string.html#formatstrings
http://stackoverflow.com/questions/455612/python-limiting-floats-to-two-decimal-points

- And now... pandas

pandas

- pandas is a powerful Python library for data analysis and analytics
- Series, DataFrame
- Indexing
- Aggregation and groupby operations

```
from pandas import Series, DataFrame
import pandas as pd
from numpy.random import randn
import numpy as np
```

Series

- One-dimensional data structure
  - Contains an array of data
  - And an associated array of data labels called its ***index***

```
In [40]: s = Series([31, 25, 18])

In [41]: s
Out[41]:
0    31
1    25
2    18
dtype: int64

In [42]: s.values
Out[42]: array([31, 25, 18], dtype=int64)

In [43]: s.index
Out[43]: Int64Index([0, 1, 2], dtype=int64)
```

In this example, we did not specified an index, so pandas creates a default index of integers.

Series Index

- When creating a Series, you can specify an index

```
In [44]: s = Series([31, 25, 18],
                    index=['inls285', 'inls382', 'inls523'])

In [45]: s
Out[45]:
inls285    31
inls382    25
inls523    18
dtype: int64

In [46]: s.values
Out[46]: array([31, 25, 18], dtype=int64)

In [47]: s.index
Out[47]: Index([u'inls285', u'inls382', u'inls523'],
dtype=object)
```

Series Operations & Indexing

- Values can be selected by their index or position

```
In [52]: s13 = Series([31, 25, 18], index=['inls285', 'inls382', 'inls523'])

In [53]: s14 = Series([29, 23, 14], index=['inls285', 'inls382', 'inls523'])
```

```
In [61]: s13 + 1
Out[61]:
inls285    32
inls382    26
inls523    19
dtype: int64

In [62]: s14 * 2
Out[62]:
inls285    58
inls382    46
inls523    28
dtype: int64

In [63]: s13 + s14
Out[63]:
inls285    60
inls382    48
inls523    32
dtype: int64
```

```
In [64]: s13[s13>20]
Out[64]:
inls285    31
inls382    25
dtype: int64

In [65]: s14[s14>20]
Out[65]:
inls285    29
inls382    23
dtype: int64

In [66]: s14[s13>20]
Out[66]:
inls285    29
inls382    23
dtype: int64
```

Series from Dicts

- Series can be created from dicts
- Can be used like an ordered dict
- map index vals to data vals

```
In [68]: d = {'a': 5, 'b': 10, 'c': 15}

In [69]: s = Series(d)

In [70]: d
Out[70]: {'a': 5, 'b': 10, 'c': 15}

In [71]: s
Out[71]:
a     5
b    10
c    15
dtype: int64

In [72]: if 'a' in s:
    ...:     print s['a']
    ...:
5
```

Series from Dict

- When creating a Series from a dict
- If an index is provided, only items that match will be included.
- If the index has “extra” items, they will get **`NaN`**
  - not a number – treated as “missing” data

```
In [73]: d = {'a': 5, 'b': 10, 'c': 15}

In [74]: t = ['b', 'c', 'd']

In [75]: s = Series(d, index=t)

In [76]: s
Out[76]:
b    10
c    15
d   NaN
dtype: float64
```

Note that in this case, the Series used float64 even though we gave it integers.

This is because NaN is only supported for floats.

Series Name

- Series objects also have a name attribute.
- The index of a series also has a name attribute.

```
In [77]: s14
Out[77]:
inls285    29
inls382    23
inls523    14
dtype: int64

In [78]: s14.name = "Spring 2014"

In [79]: s14.index.name = "Course names"

In [80]: s14
Out[80]:
Course names
inls285         29
inls382         23
inls523         14
Name: Spring 2014, dtype: int64
```

Series – Index assignment

- The index of a series can be changed by assignment.

```
In [81]: s14
Out[81]:
Course names
inls285         29
inls382         23
inls523         14
Name: Spring 2014, dtype: int64

In [82]: s14.index = ['INLS 285', 'INLS 382', 'INLS 523']

In [83]: s14
Out[83]:
INLS 285    29
INLS 382    23
INLS 523    14
Name: Spring 2014, dtype: int64
```

Series Exercise
(Britney's back!)

- Create two Series objects:
  - aug_plays, sept_plays
  - Both should use the same index vals:
    - Britney Spears, Depeche Mode, Lady Gaga
  - Use the play counts shown on the right as values (just type them in)
- Use aug_plays and sept_plays to create avg_plays
  - You can use integers
  - Aug + Sept / 2

```
In [91]: aug_plays
Out[91]:
Britney Spears    190
Depeche Mode      274
Lady Gaga         344
dtype: int64

In [92]: sept_plays
Out[92]:
Britney Spears    123
Depeche Mode      497
Lady Gaga         273
dtype: int64

In [94]: avg_plays
Out[94]:
Britney Spears    156
Depeche Mode      385
Lady Gaga         308
dtype: int64
```

Data Frame

- Tabular data structure, like a spreadsheet
  - Ordered collection of columns
  - Each column can be a diff data type
  - Row and column indexes

	A	B	C
0	v0	v1	v2
1	v3	v4	v5
2	v6	v7	v8
3	v9	v10	v11
4	v12	v13	v14

DataFrame

- Create a DataFrame from a dict of equal-length lists

	course	semester	enrollment
0	inls285	s13	31
1	inls285	s14	58
2	inls382	s13	26
3	inls382	s14	46
4	inls523	s13	19
5	inls523	s14	28

```
d = {'course': ['inls285', 'inls285', 'inls382', 'inls382', 'inls523', 'inls523'],
'semester': ['s13', 's14', 's13', 's14', 's13', 's14'], 'enrollment': [31, 58, 26,
46, 19, 28]}


df = DataFrame(d)
```

DataFrame

- DataFrame output in Canopy

```
In [29]: d = {'course': ['inls285', 'inls285', 'inls382', 'inls382', 'inls523', 'inls523'],
    ...:     'semester': ['s13', 's14', 's13', 's14', 's13', 's14'],
    ...:     'enrollment': [31, 58, 26, 46, 19, 28]}

In [30]: df = DataFrame(d)

In [31]: df
Out[31]:
```

	course	enrollment	semester
0	inls285	31	s13
1	inls285	58	s14
2	inls382	26	s13
3	inls382	46	s14
4	inls523	19	s13
5	inls523	28	s14

```
In [32]: print str(df)
    course  enrollment semester
0  inls285          31      s13
1  inls285          58      s14
2  inls382          26      s13
3  inls382          46      s14
4  inls523          19      s13
5  inls523          28      s14
```

Retrieving Columns

- Retrieve columns by dict-like notation, or by attribute
- Columns are retrieved as a Series

```
In [33]: type(df)
Out[33]: pandas.core.frame.DataFrame

In [34]: print str(df)
    course  enrollment semester
0  inls285          31      s13
1  inls285          58      s14
2  inls382          26      s13
3  inls382          46      s14
4  inls523          19      s13
5  inls523          28      s14

In [35]: s = df['course']

In [36]: type(s)
Out[36]: pandas.core.series.Series
```

```
In [37]: s
Out[37]:
0    inls285
1    inls285
2    inls382
3    inls382
4    inls523
5    inls523
Name: course, dtype: object

In [38]: s2 = df.course

In [39]: s2
Out[39]:
0    inls285
1    inls285
2    inls382
3    inls382
4    inls523
5    inls523
Name: course, dtype: object
```

DataFrame Index

- DataFrames can also have a customized index (like Series do)

```
In [42]: d = {'course': ['inls285', 'inls285', 'inls382', 'inls382', 'inls523',
'inls523'], 'semester': ['s13', 's14', 's13', 's14', 's13', 's14'], 'enrollment':
[31, 58, 26, 46, 19, 28]}

In [51]: d
Out[51]:
{'course': ['inls285', 'inls285', 'inls382', 'inls382', 'inls523', 'inls523'],
 'enrollment': [31, 58, 26, 46, 19, 28],
 'semester': ['s13', 's14', 's13', 's14', 's13', 's14']}

In [43]: df = DataFrame(d, index=['c1234', 'c2345', 'c8822', 'c7654', 'c5512',
'c4321'])

In [44]: print str(df)
        course  enrollment semester
c1234  inls285          31      s13
c2345  inls285          58      s14
c8822  inls382          26      s13
c7654  inls382          46      s14
c5512  inls523          19      s13
c4321  inls523          28      s14
```

DataFrame Index

- Columns are retrieved as a Series w/ same index as DF

```
In [44]: print str(df)
        course  enrollment semester
c1234  inls285          31      s13
c2345  inls285          58      s14
c8822  inls382          26      s13
c7654  inls382          46      s14
c5512  inls523          19      s13
c4321  inls523          28      s14

In [45]: df.course
Out[45]:
c1234    inls285
c2345    inls285
c8822    inls382
c7654    inls382
c5512    inls523
c4321    inls523
Name: course, dtype: object
```

Retrieve Rows using .ix

- Rows can be retrieved using .ix

```
In [54]: print str(df)
         course  enrollment semester
c1234  inls285          31      s13
c2345  inls285          58      s14
c8822  inls382          26      s13
c7654  inls382          46      s14
c5512  inls523          19      s13
c4321  inls523          28      s14

In [55]: s = df.ix['c7654']

In [56]: type(s)
Out[56]: pandas.core.series.Series

In [57]: s
Out[57]:
course        inls382
enrollment         46
semester          s14
Name: c7654, dtype: object

In [58]: s.values
Out[58]: array(['inls382', 46, 's14'], dtype=object)

In [59]: s.index
Out[59]: Index([u'course', u'enrollment', u'semester'], dtype=object)
```

Notice how the row is retrieved as a Series whose index is the columns of the DF.

DataFrame

- You can create a new column and assign values to it using assignment

```
In [54]: print str(df)
        course  enrollment semester
c1234  inls285          31      s13
c2345  inls285          58      s14
c8822  inls382          26      s13
c7654  inls382          46      s14
c5512  inls523          19      s13
c4321  inls523          28      s14

In [64]: df['tmp'] = [1, 3, 5, 7, 8, 9]

In [65]: print str(df)
        course  enrollment semester  tmp
c1234  inls285          31      s13    1
c2345  inls285          58      s14    3
c8822  inls382          26      s13    5
c7654  inls382          46      s14    7
c5512  inls523          19      s13    8
c4321  inls523          28      s14    9
```

DataFrame

- A dict of dicts will create a DF with outer dict keys as the columns and inner dicts keys as row indices

```
In [76]: d = {'unc': {2012: 4.1, 2013: 4.3, 2014: 4.5}, 'duke': {2012:
3.8, 2013: 3.8, 2014: 4.1}}

In [77]: df = DataFrame(d)

In [79]: print str(df)
      duke  unc
2012   3.8  4.1
2013   3.8  4.3
2014   4.1  4.5

In [80]: df.columns
Out[80]: Index([u'duke', u'unc'], dtype=object)

In [81]: df.index
Out[81]: Int64Index([2012, 2013, 2014], dtype=int64)

In [82]: print str(df.T)
      2012  2013  2014
duke   3.8   3.8   4.1
unc    4.1   4.3   4.5
```

Can transpose using T, like with numpy arrays

DataFrame

- DF columns can be extracted and operated on as either Series or numpy arrays

```
In [95]: print str(df)
      duke  unc
2012   3.8  4.1
2013   3.8  4.3
2014   4.1  4.5

In [96]: s = df.unc

In [97]: type(s)
Out[97]: pandas.core.series.Series

In [98]: a = df.unc.values

In [99]: type(a)
Out[99]: numpy.ndarray

In [100]: s.sum()
Out[100]: 12.899999999999999

In [101]: a.sum()
Out[101]: 12.899999999999999
```

DataFrame Exercise
(this is Ex #11 to turn in)

- Create a DataFrame with the following play count data:

	Aug	Sept	Nov
David Bowie	571	623	409
The Beatles	725	518	822
New Order	274	492	368

- After creating the DF:
  1. Extract the Sept column and compute the total # of plays
  2. Extract the row for 'David Bowie' and compute the total # of plays
- Save your code as `youronyen_ex11.py` and submit it to Sakai "Exercise #11" before Thursday, April 10 at 11:00am.

DF Indexing

```
In [27]: d = {'unc': {'aug': 4.1, 'sep': 4.3, 'oct': 4.5},
'duke': {'aug': 3.8, 'sep': 3.8, 'oct': 4.1}}

In [28]: df = DataFrame(d)

In [29]: print str(df)
     duke  unc
aug   3.8  4.1
oct   4.1  4.5
sep   3.8  4.3

In [30]: df.unc
Out[30]:
aug    4.1
oct    4.5
sep    4.3
Name: unc, dtype: float64
```

```
In [31]: df.unc['oct']
Out[31]: 4.5

In [33]: df.unc[1]
Out[33]: 4.5

In [34]: df['unc'][1]
Out[34]: 4.5

In [39]: df['unc']['oct']
Out[39]: 4.5
```

NOTE:
`df[0][1]`
does NOT work

- There are several ways to refer to specific elements in a DataFrame:
  - Columns
    - The first [], using the column name: **`df['unc']`**
    - Or can be accessed as an attribute: **`df.unc`**
  - Rows
    - The second [], using the row name: **`df.unc['oct']`**
    - The second [], using the row position: **`df.unc[1]`**

More Complex Indexing

```
In [114]: d = {'unc': {2012: 4.1, 2013: 4.3, 2014: 4.5},
'duke': {2012: 3.8, 2013: 3.8, 2014: 4.1}, 'ncstate': {2012:
3.9, 2013: 3.8, 2014: 4.3}}

In [115]: df = DataFrame(df)

In [116]: print str(df)
      duke  ncstate  unc
2012   3.8      3.9  4.1
2013   3.8      3.8  4.3
2014   4.1      4.3  4.5
```

```
In [117]: print str(df['unc'])
2012    4.1
2013    4.3
2014    4.5
Name: unc, dtype: float64
```

Retrieve a column

```
In [118]: print str(df[['unc','ncstate']])
      unc  ncstate
2012  4.1      3.9
2013  4.3      3.8
2014  4.5      4.3
```

Retrieve multiple columns in the order specified

```
In [119]: print str(df[:2])
      duke  ncstate  unc
2012   3.8      3.9  4.1
2013   3.8      3.8  4.3
```

Slice of rows

```
In [120]: print str(df[df['duke']>4.0])
      duke  ncstate  unc
2014   4.1      4.3  4.5
```

Boolean array

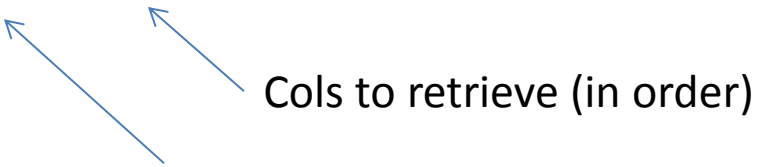
Arithmetic between DF and Series

```
In [157]: print str(df)
      duke  ncstate  unc
2012   3.8      3.9  4.1
2013   3.8      3.8  4.3
2014   4.1      4.3  4.5

In [158]: d = {'unc': 0.4, 'duke': 0.8, 'ncstate': 0.6}

In [159]: s = Series(d)

In [160]: s
Out[160]:
duke       0.8
ncstate    0.6
unc        0.4
dtype: float64

In [161]: print str(df - s)
      duke  ncstate  unc
2012   3.0      3.3  3.7
2013   3.0      3.2  3.9
2014   3.3      3.7  4.1
```

Matches the index of the Series to the columns of the DF, then *broadcasts* down the rows

Sorting by row or column index

```
In [168]: print str(df)
      duke  ncstate  unc
2012   3.8      3.9  4.1
2013   3.8      3.8  4.3
2014   4.1      4.3  4.5

In [169]: df2 = df.ix[[2014, 2012, 2013], ['unc', 'duke',
'ncstate']]

In [170]: type(df2)
Out[170]: pandas.core.frame.DataFrame

In [171]: print str(df2)
      unc  duke  ncstate
2014  4.5   4.1      4.3
2012  4.1   3.8      3.9
2013  4.3   3.8      3.8

In [172]: print str(df2.sort_index())
      unc  duke  ncstate
2012  4.1   3.8      3.9
2013  4.3   3.8      3.8
2014  4.5   4.1      4.3

In [173]: print str(df2.sort_index(axis=1))
      duke  ncstate  unc
2014   4.1      4.3  4.5
2012   3.8      3.9  4.1
2013   3.8      3.8  4.3
```



Summarizing Data

(Descriptive Statistics)

- pandas objects such as Series and DataFrame support many common mathematical operations to summarize data.
- For example:
  - count
  - sum
  - mix, max
  - mean, median, std

- A nice feature of these operations is that pandas typically handles *missing data* in an elegant way

Table 5-10. Descriptive and summary statistics

Method	Description
`count`	Number of non-NA values
`describe`	Compute set of summary statistics for Series or each DataFrame column
`min, max`	Compute minimum and maximum values
`argmin, argmax`	Compute index locations (integers) at which minimum or maximum value obtained, respectively
`idxmin, idxmax`	Compute index values at which minimum or maximum value obtained, respectively
`quantile`	Compute sample quantile ranging from 0 to 1
`sum`	Sum of values
`mean`	Mean of values
`median`	Arithmetic median (50% quantile) of values
`mad`	Mean absolute deviation from mean value
`var`	Sample variance of values
`std`	Sample standard deviation of values
`skew`	Sample skewness (3rd moment) of values
`kurt`	Sample kurtosis (4th moment) of values
`cumsum`	Cumulative sum of values
`cummin, cummax`	Cumulative minimum or maximum of values, respectively
`cumprod`	Cumulative product of values
`diff`	Compute 1st arithmetic difference (useful for time series)
`pct_change`	Compute percent changes

sum()

```
In [206]: d = {'unc': {2012: 4.1, 2013: 4.3, 2014: 4.5}, 'duke': {2012:
3.8, 2013: 3.8, 2014: 4.1}, 'ncstate': {2012: 3.9, 2013: 3.8, 2014: 4.3}}

In [207]: df = DataFrame(d)

In [211]: print str(df)
      duke  ncstate  unc
2012   3.8      NaN  4.1
2013   3.8      3.8  4.3
2014   4.1      4.3  4.5

In [212]: df.unc.sum()
Out[212]: 12.899999999999999

In [213]: df['unc'].sum()
Out[213]: 12.899999999999999

In [214]: df['ncstate'].sum()
Out[214]: 8.0999999999999996

In [216]: df.ix[2012].sum()
Out[216]: 7.8999999999999995
```

sum(), mean(), idxmax()

```
In [206]: d = {'unc': {2012: 4.1, 2013: 4.3, 2014: 4.5}, 'duke': {2012:
3.8, 2013: 3.8, 2014: 4.1}, 'ncstate': {2012: 3.9, 2013: 3.8, 2014: 4.3}}

In [45]: df = DataFrame(d)

In [46]: print df
      duke  ncstate  unc
2012   3.8      3.9  4.1
2013   3.8      3.8  4.3
2014   4.1      4.3  4.5

In [47]: df['ncstate'][2012] = np.nan

In [48]: print df
      duke  ncstate  unc
2012   3.8      NaN  4.1
2013   3.8      3.8  4.3
2014   4.1      4.3  4.5

In [49]: df['ncstate'].mean()
Out[49]: 4.0499999999999998
```

```
In [50]: df.sum()
Out[50]:
duke       11.7
ncstate     8.1
unc        12.9
dtype: float64

In [51]: df.sum(axis=1)
Out[51]:
2012     7.9
2013    11.9
2014    12.9
dtype: float64

In [52]: df.idxmax()
Out[52]:
duke       2014
ncstate    2014
unc        2014
dtype: int64
```

Sorting a Series

```
In [8]: s = Series([6, 2, 8, 4], index=['b', 'd', 'a', 'c'])

In [9]: s
Out[9]:
b    6
d    2
a    8
c    4
dtype: int64

In [10]: s.sort_index()
Out[10]:
a    8
b    6
c    4
d    2
dtype: int64

In [11]: s.order()
Out[11]:
d    2
c    4
b    6
a    8
dtype: int64
```

Series can be sorted by:

1. the index using .sort_index()

2. the values using .order()

Sorting a DataFrame on an Index

```
In [23]: df = DataFrame({'cb': [6, 2, 8, 4], 'ca': [7, 3, 1, 5]},
index=['id', 'ia', 'ib', 'ic'])

In [24]: print df
    ca  cb
id   7   6
ia   3   2
ib   1   8
ic   5   4

In [25]: print df.sort_index()
    ca  cb
ia   3   2
ib   1   8
ic   5   4
id   7   6

In [26]: print df.sort_index(axis=1, ascending=False)
    cb  ca
id   6   7
ia   2   3
ib   8   1
ic   4   5
```

axis=1 indicates to sort the column index
ascending = False indicates reverse order

Sorting a DataFrame on a Column

```
In [28]: print df
    ca  cb
id   7   6
ia   3   2
ib   1   8
ic   5   4

In [29]: print df.sort_index(by='cb')
    ca  cb
ia   3   2
ic   5   4
id   7   6
ib   1   8

In [30]: print df.sort_index(by='ca')
    ca  cb
ib   1   8
ia   3   2
ic   5   4
id   7   6
```

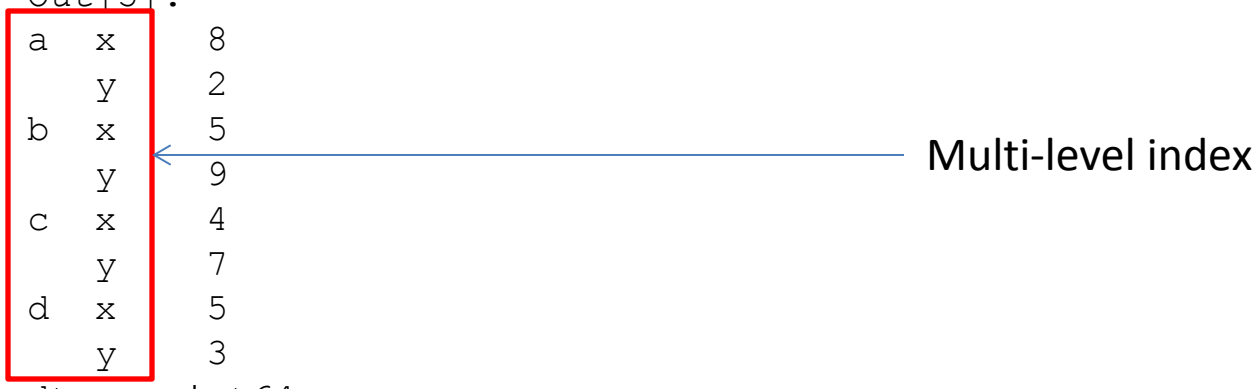
Sorting on Multiple Columns

```
In [31]: df = DataFrame({'a': [5, 7, 1, 1], 'b': [2, 4, 8, 6]})

In [32]: print df
   a  b
0  5  2
1  7  4
2  1  8
3  1  6

In [33]: print df.sort_index(by=['a','b'])
   a  b
3  1  6
2  1  8
0  5  2
1  7  4
```

Hierarchical Indexing

- Allows multiple index ***levels*** on an axis (row or column)

```
In [4]: s = Series([8, 2, 5, 9, 4, 7, 5, 3],
index=[['a','a','b','b','c','c','d','d'],
['x','y','x','y','x','y','x','y']])

In [5]: s
Out[5]:
a  x    8
   y    2
b  x    5
   y    9
c  x    4
   y    7
d  x    5
   y    3
dtype: int64

In [6]: s.index
Out[6]:
MultiIndex
[(u'a', u'x'), (u'a', u'y'), (u'b', u'x'),
(u'b', u'y'), (u'c', u'x'), (u'c', u'y'),
(u'd', u'x'), (u'd', u'y')]
```



Hierarchical Indexing

- Partial indexing to select subsets of the data

```
In [11]: s
Out[11]:
a  x    8
   y    2
b  x    5
   y    9
c  x    4
   y    7
d  x    5
   y    3
dtype: int64

In [12]: s['b']
Out[12]:
x    5
y    9
dtype: int64
```

```
In [13]: s['b':'c']
Out[13]:
b  x    5
   y    9
c  x    4
   y    7
dtype: int64

In [14]: s.ix[['a','c']]
Out[14]:
a  x    8
   y    2
c  x    4
   y    7
dtype: int64
```

```
In [17]: s[:,'y']
Out[17]:
a    2
b    9
c    7
d    3
dtype: int64
```

unstack() and stack()

```
In [20]: print s.unstack()
   x  y
a  8  2
b  5  9
c  4  7
d  5  3

In [20]:

In [20]:

In [21]: s
Out[21]:
a  x    8
   y    2
b  x    5
   y    9
c  x    4
   y    7
d  x    5
   y    3
dtype: int64
```

```
In [22]: s2 = s.unstack()

In [23]: print s2
   x  y
a  8  2
b  5  9
c  4  7
d  5  3

In [24]: print s2.stack()
a  x    8
   y    2
b  x    5
   y    9
c  x    4
   y    7
d  x    5
   y    3
dtype: int64
```

DataFrame – Hierarchical Index

- In a DataFrame, either axis (row or column) can have a hierarchical index

```
In [31]: d = np.arange(12).reshape((4,3))

In [32]: d
Out[32]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [33]: df = DataFrame(d, index=[['a','a','b','b'], [1, 2, 1,
2]], columns=[['unc','unc','duke'], ['x','y','x']])

In [34]: print df
     unc      duke
       x   y     x
a 1    0   1     2
  2    3   4     5
b 1    6   7     8
  2    9  10    11
```

Summary Statistics by Level

- Many summary stats functions have a ***level*** option that can be used with a hierarchical index

```
In [42]: print df
     unc      duke
       x   y     x
a 1    0   1     2
  2    3   4     5
b 1    6   7     8
  2    9  10    11

In [43]: print df.sum(level=0)
   unc      duke
     x   y     x
a    3   5     7
b   15  17    19

In [44]: print df.sum(level=1)
   unc      duke
     x   y     x
1    6   8    10
2   12  14    16
```

```
In [45]: print df.sum(level=0,
axis=1)
     duke  unc
a 1     2    1
  2     5    7
b 1     8   13
  2    11   19

In [46]: print df.sum(level=1,
axis=1)
      x   y
a 1   2   1
  2   8   4
b 1  14   7
  2  20  10
```

Using a column as an index

- You may want to use a column of a DataFrame as a row index (or vice-versa)

```
In [59]: a = range(7)

In [60]: b = range(7,0,-1)

In [61]: c =
['x','x','x','y','y','y','y']

In [62]: d = [0,1,2,0,1,2,3]

In [63]: df = DataFrame({'a': a,
'b': b, 'c':c, 'd':d})

In [64]: print df
   a  b  c  d
0  0  7  x  0
1  1  6  x  1
2  2  5  x  2
3  3  4  y  0
4  4  3  y  1
5  5  2  y  2
6  6  1  y  3
```

```
In [65]: df2 =
df.set_index(['c','d'])

In [66]: print df2
     a  b
c d
x 0  0  7
  1  1  6
  2  2  5
y 0  3  4
  1  4  3
  2  5  2
  3  6  1
```

reset_index

- reset_index goes the other direction, moving a hierarchical index levels into columns

```
In [66]: print df2
     a  b
c d
x 0  0  7
  1  1  6
  2  2  5
y 0  3  4
  1  4  3
  2  5  2
  3  6  1

In [67]: df3 = df2.reset_index()

In [68]: print df3
   c  d  a  b
0  x  0  0  7
1  x  1  1  6
2  x  2  2  5
3  y  0  3  4
4  y  1  4  3
5  y  2  5  2
6  y  3  6  1
```

Hierarchical Index Exercise
(this is Ex #12 to turn in)

- Create a DataFrame using a hierarchical index with the following play count data:

		Aug	Sep	Nov
uid123	Bowie	12	15	26
	Gaga	2	0	4
	Spears	1	0	3
uid345	Bowie	3	0	4
	Gaga	24	18	31
	Spears	8	12	5
uid678	Bowie	6	3	0
	Gaga	8	14	27
	Spears	28	21	16

- After creating the DF, use summary statistics by level to:
  1. Output a summary table of the total plays for each uid for each month (i.e. collapse artists)
  2. Output a summary table of the total plays for each artist for each month (i.e. collapse uids)
  3. Output a summary table of the total plays for each uid for each artists (i.e. collapse months)

- Save your code as `youronyen_ex12.py` and submit it to Sakai "Exercise #11" before Sunday, April 13at 5:00pm.