PANDAS

CHRISTOPH PAULIK

A PYTHON DATA ANALYSIS LIBRARY

- Built on top of numpy to make data analysis easier
- Automatic data alignment based on labels or indices
- Data aggregation, transformation and grouping
- Intuitive merging and joining of datasets
- Hierarchical labeling
- Reading and Writing of CSV, Excel and others

PANDAS.SERIES

- For storing indexed 1D data

creation from numpy array with list as index

```
s = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd', 'e'])
print s
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int64
```

INDEX IS CREATED IF NOT SET

```
pd.Series(randn(5))
```

```
0   -0.294807
1   -0.044998
2   -1.723521
3    1.329945
4   -0.036720
dtype: float64
```

SERIES IS LIKE AN ARRAY

```python
s[0]
print("\n")
s[s > s.median()]
print("\n")
s[[3,2,1]]
```

```
0


d    3
e    4
dtype: int64


d    3
c    2
b    1
dtype: int64
```

SERIES IS LIKE A DICTIONARY

```python
s['a']
s['e'] = 6
s
'e' in s
'f' in s
```

```
0
>>> a    0
b    1
c    2
d    3
e    6
dtype: int64
True
False
```

OPERATIONS ON SERIES

```
s+s
s**2
np.exp(s)
```

```
a      0
b      2
c      4
d      6
e     12
dtype: int64
a      0
b      1
c      4
d      9
e     36
dtype: int64
a      1.000000
b      2.718282
c      7.389056
d     20.085537
e    403.428793
dtype: float64
```

PANDAS.DATAFRAME

A 2D labeled data structure with columns of potentially different types.

Like Series, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

FROM DICTIONARY

```
d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
     'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
```

```
df
```

```
   one  two
a    1    1
b    2    2
c    3    3
d  NaN    4

[4 rows x 2 columns]
```

FROM OTHER DATAFRAME

```python
pd.DataFrame(df, index=['d', 'b', 'a'])
```

```
   one  two
d  NaN    4
b    2    2
a    1    1

[3 rows x 2 columns]
```

```python
pd.DataFrame(d, index=['d', 'b', 'a'], columns=['two', 'three'])
```

```
   two three
d    4   NaN
b    2   NaN
a    1   NaN

[3 rows x 2 columns]
```

COMPLEX CASES

```
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.Timestamp('20130102'),
                    'C': pd.Series(1,index=list(range(4)),
                                   dtype='float32'),
                    'D': np.array([3] * 4,dtype='int32'),
                    'E': 'foo' })
```

```
df2
```

```
   A          B  C  D    E
0  1 2013-01-02  1  3  foo
1  1 2013-01-02  1  3  foo
2  1 2013-01-02  1  3  foo
3  1 2013-01-02  1  3  foo

[4 rows x 5 columns]
```

```python
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.Timestamp('20130102'),
                    'C': pd.Series(1,index=list(range(4)),
                                   dtype='float32'),
                    'D': np.array([3] * 4,dtype='int32'),
                    'E': 'foo' })
```

```python
df2.dtypes
```

```
A             float64
B      datetime64[ns]
C             float32
D               int32
E              object
dtype: object
```

TIME SERIES

```
# Date range
dates = pd.date_range('20130101', periods=6)
# Dataframes
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
```

```
df
```

```
                   A         B         C         D
2013-01-01 -0.850581 -0.448843 -0.656592 -1.009414
2013-01-02 -0.662871 -0.195961 -0.135948 -1.385167
2013-01-03  1.995238  2.545748 -0.300269 -0.088726
2013-01-04  0.396430 -1.945540 -1.785509  0.714793
2013-01-05 -2.605348  0.493118 -0.605733 -0.090220
2013-01-06  2.055708 -0.630673  0.617193 -0.328289

[6 rows x 4 columns]
```

INSPECTION

```
df.head()
```

```
                   A         B         C         D
2013-01-01 -0.850581 -0.448843 -0.656592 -1.009414
2013-01-02 -0.662871 -0.195961 -0.135948 -1.385167
2013-01-03  1.995238  2.545748 -0.300269 -0.088726
2013-01-04  0.396430 -1.945540 -1.785509  0.714793
2013-01-05 -2.605348  0.493118 -0.605733 -0.090220

[5 rows x 4 columns]
```

```
df.tail(3)
```

```
                   A         B         C         D
2013-01-04  0.396430 -1.945540 -1.785509  0.714793
2013-01-05 -2.605348  0.493118 -0.605733 -0.090220
2013-01-06  2.055708 -0.630673  0.617193 -0.328289

[3 rows x 4 columns]
```

COLUMNS AND VALUES

```
df.columns, df.values
```

```
(Index([u'A', u'B', u'C', u'D'], dtype='object'),
 array([[-0.85058067, -0.44884314, -0.65659238, -1.00941366],
        [-0.66287078, -0.19596096, -0.13594813, -1.38516652],
        [ 1.99523764,  2.54574834, -0.30026874, -0.08872582],
        [ 0.39642992, -1.94554049, -1.78550921,  0.71479338],
        [-2.60534819,  0.49311821, -0.60573324, -0.09022009],
        [ 2.05570797, -0.63067295,  0.61719339, -0.32828863]]))
```

DESCRIBE A DATAFRAME

```
df.describe()
```

```
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.054763 -0.030358 -0.477810 -0.364504
std    1.805370  1.492730  0.788141  0.745002
min   -2.605348 -1.945540 -1.785509 -1.385167
25%   -0.803653 -0.585215 -0.643878 -0.839132
50%   -0.133220 -0.322402 -0.453001 -0.209254
75%    1.595536  0.320848 -0.177028 -0.089099
max    2.055708  2.545748  0.617193  0.714793

[8 rows x 4 columns]
```

DATAFRAME SLICING OVERVIEW

Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

BY COLUMN OR ROW SLICE

```
df['A']
```

```
2013-01-01   -0.850581
2013-01-02   -0.662871
2013-01-03    1.995238
2013-01-04    0.396430
2013-01-05   -2.605348
2013-01-06    2.055708
Freq: D, Name: A, dtype: float64
```

```
df[0:3]
```

```
                   A         B         C         D
2013-01-01 -0.850581 -0.448843 -0.656592 -1.009414
2013-01-02 -0.662871 -0.195961 -0.135948 -1.385167
2013-01-03  1.995238  2.545748 -0.300269 -0.088726

[3 rows x 4 columns]
```

BY INDEX

```
df['20130102':'20130104']
```

```
                   A         B         C         D
2013-01-02 -0.662871 -0.195961 -0.135948 -1.385167
2013-01-03  1.995238  2.545748 -0.300269 -0.088726
2013-01-04  0.396430 -1.945540 -1.785509  0.714793

[3 rows x 4 columns]
```

```
from datetime import date
df[date(2013,1,2):date(2013,1,4)]
```

```
                   A         B         C         D
2013-01-02 -0.662871 -0.195961 -0.135948 -1.385167
2013-01-03  1.995238  2.545748 -0.300269 -0.088726
2013-01-04  0.396430 -1.945540 -1.785509  0.714793

[3 rows x 4 columns]
```

BY INTEGER LOCATION

```
df.iloc[[4, 2]]
```

```
                   A         B         C         D
2013-01-05 -2.605348  0.493118 -0.605733 -0.090220
2013-01-03  1.995238  2.545748 -0.300269 -0.088726

[2 rows x 4 columns]
```

GROUPING

```
gp = 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)})
gp
```

```
     A      B         C         D
0  foo    one  0.988164 -1.173990
1  bar    one -0.630121 -0.939856
2  foo    two -1.399150 -1.246932
3  bar  three  1.462258 -0.342918
4  foo    two -1.558261 -1.082188
5  bar    two  2.230090 -0.004736
6  foo    one  0.491665  1.519990
7  foo  three -0.463409 -0.758924

[8 rows x 4 columns]
```

```
gp.groupby('A').sum()
```

```
            C         D
A
bar  3.062226 -1.287510
foo -1.940991 -2.742043

[2 rows x 2 columns]
```

```
gp.groupby(['A','B']).mean()
```

```
                  C         D
A   B
bar one   -0.630121 -0.939856
    three  1.462258 -0.342918
    two    2.230090 -0.004736
foo one    0.739914  0.173000
    three -0.463409 -0.758924
    two   -1.478706 -1.164560

[6 rows x 2 columns]
```

MERGING

```python
left = pd.DataFrame({'key': ['one', 'two'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['two', 'one'], 'rval': [4, 5]})
pd.merge(left, right, on='key')
```

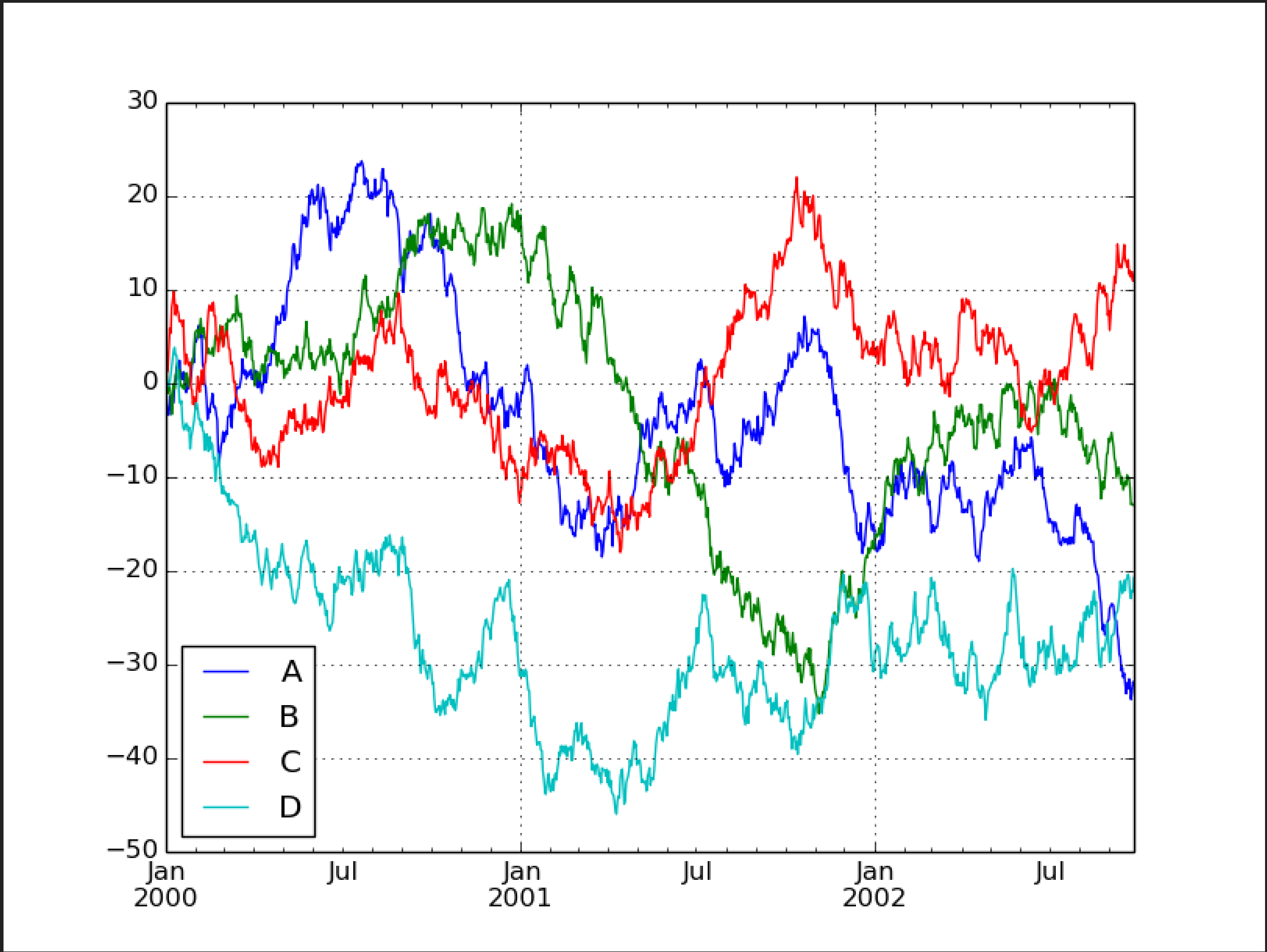
```
   key  lval  rval
0  one     1     5
1  two     2     4

[2 rows x 3 columns]
```

PLOTTING

Pandas has built-in functions for common plot types

```
import matplotlib.pyplot as plt
df = pd.DataFrame(randn(1000, 4),
                  index=pd.date_range('1/1/2000', periods=1000),
                  columns=list('ABCD'))
df = df.cumsum()
ax = df.plot()
```


30
20
10
0
−10
−20
−30
−40
−50
Jan
2000
Jul
Jan
2001
Jul
Jan
2002
Jul
A
B
C
D

WORKING WITH A DATASET

Let's try working with the Movielens 100k dataset

- 1000 Users
- 100,000 Ratings
- 1700 Movies

Extract the ml-100k.zip to a folder `ml-100k` in the same directory as the `lecture7.py`

READING THE DATA

```python
# pass in column names for each CSV
u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
users = pd.read_csv('ml-100k/u.user', sep='|', names=u_cols)

r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=r_cols)

# the movies file contains columns indicating the movie's genres
# let's only load the first five columns of the file with usecols
m_cols = ['movie_id', 'title', 'release_date',
          'video_release_date', 'imdb_url']
movies = pd.read_csv('ml-100k/u.item', sep='|',
                     names=m_cols, usecols=range(5))

# create one merged DataFrame
movie_ratings = pd.merge(movies, ratings)
lens = pd.merge(movie_ratings, users)
```

HOW DOES THE DATA LOOK LIKE?

```
lens.head(3)
```

```
   movie_id              title release_date  video_release_date  \
0         1   Toy Story (1995)  01-Jan-1995                 NaN
1         4  Get Shorty (1995)  01-Jan-1995                 NaN
2         5     Copycat (1995)  01-Jan-1995                 NaN

                                            imdb_url  user_id  rating  \
0  http://us.imdb.com/M/title-exact?Toy%20Story%2...      308       4
1  http://us.imdb.com/M/title-exact?Get%20Shorty%...      308       5
2  http://us.imdb.com/M/title-exact?Copycat%20(1995)      308       4

   unix_timestamp  age sex occupation zip_code
0       887736532   60   M    retired    95076
1       887737890   60   M    retired    95076
2       887739608   60   M    retired    95076

[3 rows x 12 columns]
```

WHAT ARE THE 10 MOST RATED MOVIES?

```
most_rated = lens.groupby('title').size().order(ascending=False)[:10]
print most_rated
```

```
title
Star Wars (1977)                    583
Contact (1997)                      509
Fargo (1996)                        508
Return of the Jedi (1983)           507
Liar Liar (1997)                    485
English Patient, The (1996)         481
Scream (1996)                       478
Toy Story (1995)                    452
Air Force One (1997)                431
Independence Day (ID4) (1996)       429
dtype: int64
```

WHICH MOVIES ARE MOST HIGHLY RATED?

The `agg` function can take multiple functions that are applied to a column

```
movie_stats = lens.groupby('title').agg({'rating': [np.size, np.mean]})
movie_stats.head()
```

```
                           rating
                             size      mean
title
'Til There Was You (1997)       9  2.333333
1-900 (1994)                    5  2.600000
101 Dalmatians (1996)         109  2.908257
12 Angry Men (1957)           125  4.344000
187 (1997)                     41  3.024390

[5 rows x 2 columns]
```

WHICH MOVIES ARE MOST HIGHLY RATED?

Sort them by mean rating

```
movie_stats.sort([('rating', 'mean')], ascending=False).head()
```

```
                                             rating
                                               size  mean
title
Marlene Dietrich: Shadow and Light (1996)         1     5
Prefontaine (1997)                                3     5
Santa with Muscles (1996)                         2     5
Star Kid (1997)                                   3     5
Someone Else's America (1995)                     1     5

[5 rows x 2 columns]
```

WHICH MOVIES ARE MOST HIGHLY RATED?

Lets only look at movies rated at least 100 times

```
atleast_100 = movie_stats['rating'].size >= 100
movie_stats[atleast_100].sort([('rating', 'mean')], ascending=False).head()
```

```
                                  rating
                                    size      mean
title
Close Shave, A (1995)                112  4.491071
Schindler's List (1993)              298  4.466443
Wrong Trousers, The (1993)           118  4.466102
Casablanca (1942)                    243  4.456790
Shawshank Redemption, The (1994)     283  4.445230

[5 rows x 2 columns]
```

EXERCISE

```
### Exercise ###
### Try to plot the ratings distribution of a movie of your choice.
### you can use the hist() function to produce a histogram
```

EXERCISE 2

```
### Exercise ###
### plot the mean rating by age of user
```

ADDITIONAL RESOURCES

- Pandas website - The documentation is very thorough and full of examples
- List of pandas tutorials
- using pandas on the movielens dataset (blogpost from which I took some examples)