PANDAS

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   -1.435335
1   -2.523561
2   -1.356375
3    0.016566
4   -0.297398
dtype: float64
```

SERIES IS LIKE AN ARRAY

```
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

```
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
```

FROM OTHER DATAFRAME

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

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

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

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

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
```

```
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.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 -0.206765  0.350524 -1.617857 -0.736760
2013-01-02 -0.519446  1.185063  1.176936  0.385541
2013-01-03 -0.612372  0.164813 -0.433153  0.154663
2013-01-04  0.493483  0.294654  1.509413  1.070845
2013-01-05  0.853607 -0.638706 -1.781561  2.053478
2013-01-06 -0.268534 -0.759996 -1.456812  0.028835
```

INSPECTION

```
df.head()
```

```
                   A         B         C         D
2013-01-01 -0.206765  0.350524 -1.617857 -0.736760
2013-01-02 -0.519446  1.185063  1.176936  0.385541
2013-01-03 -0.612372  0.164813 -0.433153  0.154663
2013-01-04  0.493483  0.294654  1.509413  1.070845
2013-01-05  0.853607 -0.638706 -1.781561  2.053478
```

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

```
                   A         B         C         D
2013-01-04  0.493483  0.294654  1.509413  1.070845
2013-01-05  0.853607 -0.638706 -1.781561  2.053478
2013-01-06 -0.268534 -0.759996 -1.456812  0.028835
```

COLUMNS AND VALUES

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

```
(Index(['A', 'B', 'C', 'D'], dtype='object'),
 array([[-0.20676519,  0.35052391, -1.61785684, -0.73675974],
        [-0.51944644,  1.18506269,  1.17693572,  0.38554099],
        [-0.61237228,  0.16481312, -0.43315343,  0.15466335],
        [ 0.49348278,  0.29465366,  1.50941344,  1.07084483],
        [ 0.85360735, -0.63870612, -1.78156098,  2.05347844],
        [-0.26853374, -0.75999558, -1.45681235,  0.02883511]]))
```

DESCRIBE A DATAFRAME

```
df.describe()
```

```
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean  -0.043338  0.099392 -0.433839  0.492767
std    0.586615  0.716588  1.458462  0.961538
min   -0.612372 -0.759996 -1.781561 -0.736760
25%   -0.456718 -0.437826 -1.577596  0.060292
50%   -0.237649  0.229733 -0.944983  0.270102
75%    0.318421  0.336556  0.774413  0.899519
max    0.853607  1.185063  1.509413  2.053478
```

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   -0.206765
2013-01-02   -0.519446
2013-01-03   -0.612372
2013-01-04    0.493483
2013-01-05    0.853607
2013-01-06   -0.268534
Freq: D, Name: A, dtype: float64
```

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

```
                   A         B         C         D
2013-01-01 -0.206765  0.350524 -1.617857 -0.736760
2013-01-02 -0.519446  1.185063  1.176936  0.385541
2013-01-03 -0.612372  0.164813 -0.433153  0.154663
```

BY INDEX

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

```
                   A         B         C         D
2013-01-02 -0.519446  1.185063  1.176936  0.385541
2013-01-03 -0.612372  0.164813 -0.433153  0.154663
2013-01-04  0.493483  0.294654  1.509413  1.070845
```

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

```
                   A         B         C         D
2013-01-02 -0.519446  1.185063  1.176936  0.385541
2013-01-03 -0.612372  0.164813 -0.433153  0.154663
2013-01-04  0.493483  0.294654  1.509413  1.070845
```

BY INTEGER LOCATION

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

```
                   A         B         C         D
2013-01-05  0.853607 -0.638706 -1.781561  2.053478
2013-01-03 -0.612372  0.164813 -0.433153  0.154663
```

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 -0.720462  0.450766
1  bar    one -0.970592  0.532224
2  foo    two  0.095800  2.589090
3  bar  three -0.320740 -1.847557
4  foo    two -0.539154  2.315864
5  bar    two  0.798654 -0.344383
6  foo    one -0.452392 -0.302850
7  foo  three  0.500077 -0.207130
```

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

```
            C         D
A
bar -0.492679 -1.659716
foo -1.116131  4.845739
```

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

```
                  C         D
A   B
bar one   -0.970592  0.532224
    three -0.320740 -1.847557
    two    0.798654 -0.344383
foo one   -0.586427  0.073958
    three  0.500077 -0.207130
    two   -0.221677  2.452477
```

MERGING

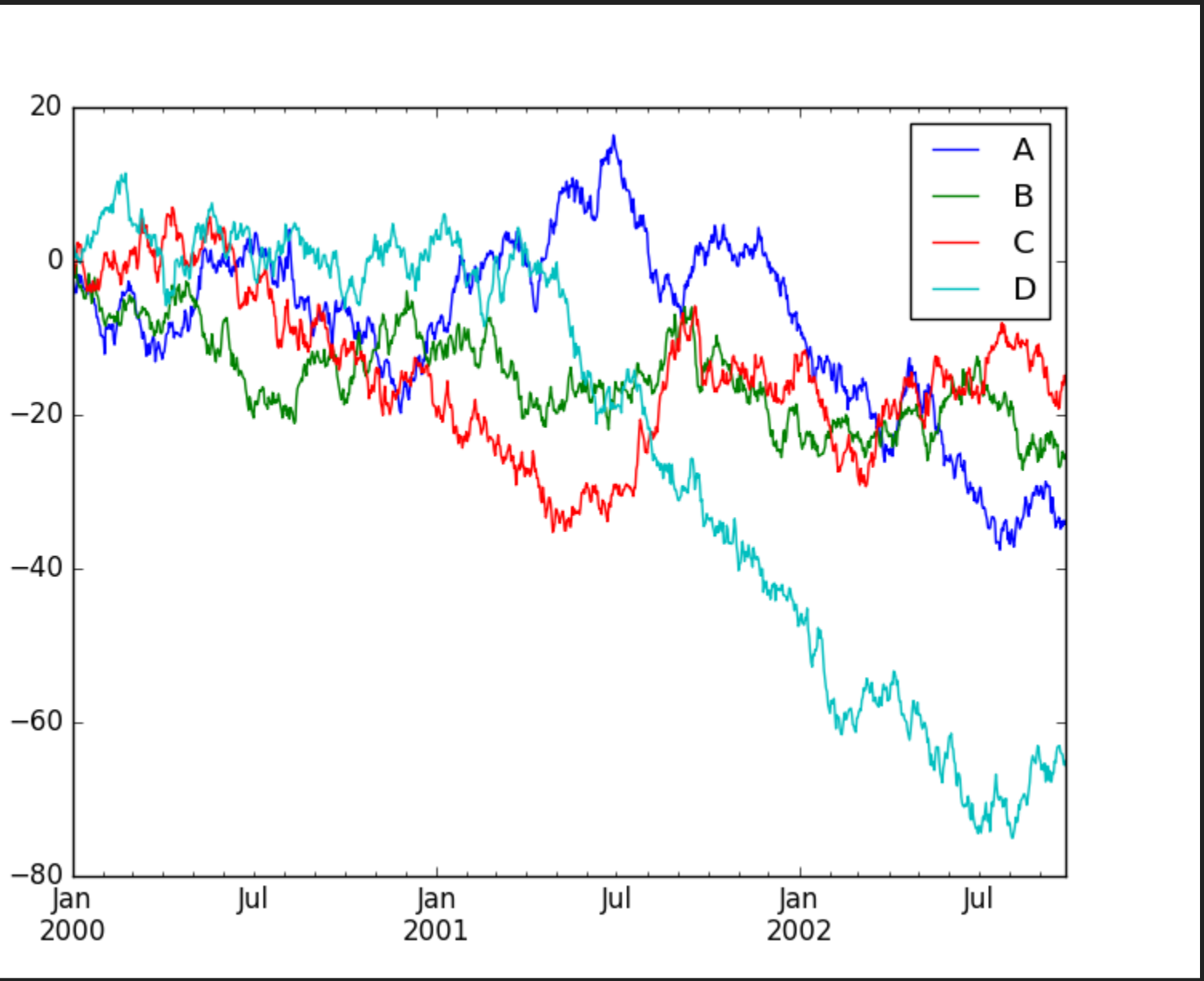
```
left = pd.DataFrame({'key': ['one', 'two'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['two', 'one'], 'rval': [4, 5]})
pd.merge(left, right, on='key')
```

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

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()
```


20
0
−20
−40
−60
−80
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

```
# 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,
                    encoding="latin-1")

r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=r_cols,
                      encoding="latin-1")

# 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),
                     encoding='latin-1')

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

WHAT DID WE READ?

```
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  ratin
g  \
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
```

WHAT ARE THE 10 MOST RATED MOVIES?

```
most_rated = lens.groupby('title').size().sort_values(ascending=Fals
e)[: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

WHICH MOVIES ARE MOST HIGHLY RATED?

Sort them by mean rating

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

	rating	
	size	mean
title		
They Made Me a Criminal (1939)	1	5
Marlene Dietrich: Shadow and Light (1996)	1	5
Saint of Fort Washington, The (1993)	2	5
Someone Else's America (1995)	1	5
Star Kid (1997)	3	5

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()
```

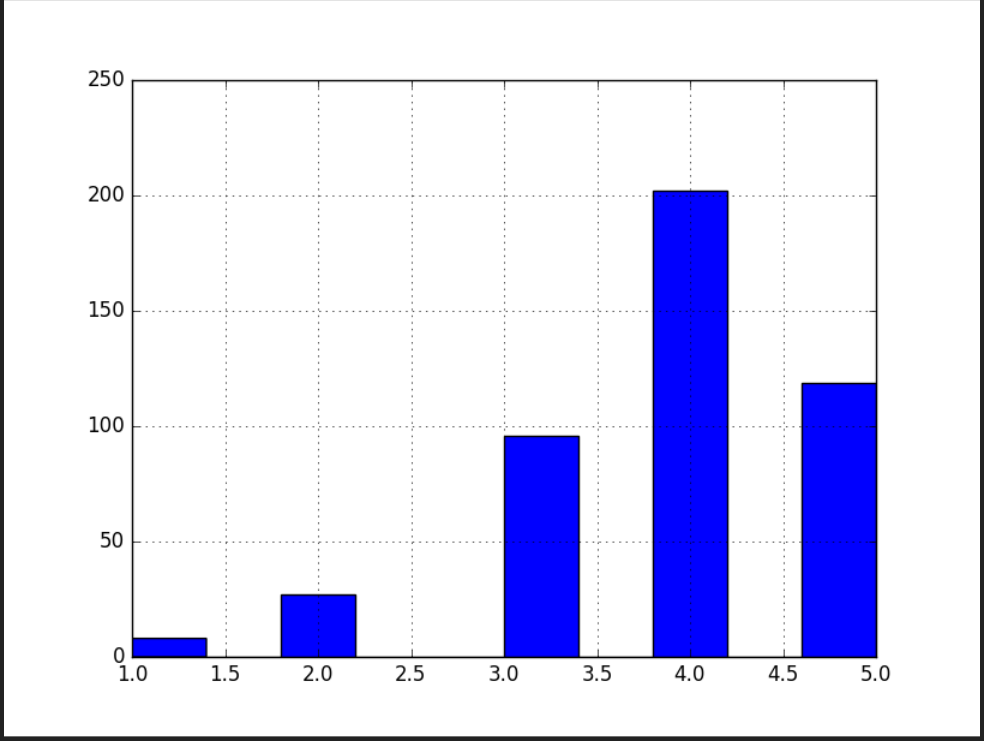
	rating	
	size	mean
title		
Close Shave, A (1995)	112	4.491071
Schindler's List (1993)	298	4.466443
Wrong Trousers, The (1993)	118	4.466102
Casablanca (1942)	243	4.456790
Shawshank Redemption, The (1994)	283	4.445230

EXERCISE

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

SOLUTION

```
toy_story = lens[lens.title=='Toy Story (1995)']
plt.figure()
ax = toy_story.rating.hist()
plt.savefig('hist.png')
'hist.png'
```

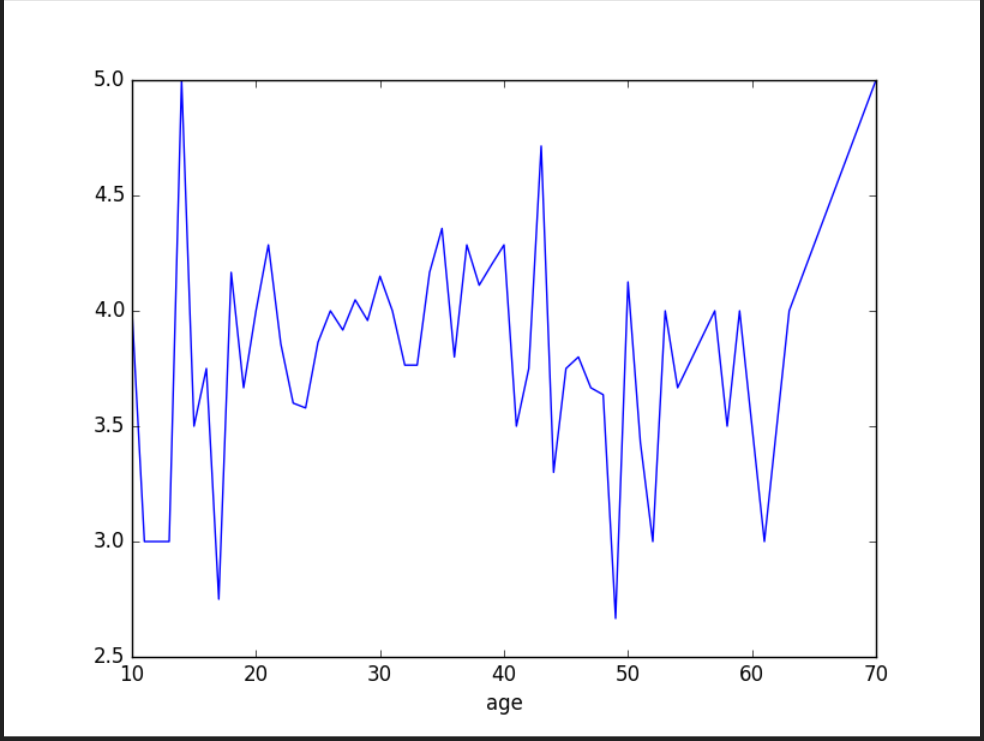


EXERCISE 2

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

SOLUTION

```
age_grouped = toy_story.groupby('age').mean()
plt.figure()
ax = age_grouped['rating'].plot()
plt.savefig('age-ratings.png')
'age-ratings.png'
```



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)