Exercises for "Hands-on with Pydata: How to Build a Minimal Recommendation Engine"

Systems check: imports and files

In [57]:

```
import numpy as np
import pandas as pd
```

Pandas questions: Series and DataFrames

1. Adding a column in a DataFrame

In [58]:

```
# given the following DataFrame, add a new column to it
df = pd.DataFrame({'col1': [1,2,3,4]})
df
```

Out[58]:

	col1
0	1
1	2
2	3
3	4

2. Deleting a row in a DataFrame

In [59]:

```
# given the following DataFrame, delete row 'd' from it
df = pd.DataFrame({'col1': [1,2,3,4]}, index = ['a','b','c','d'])
df
```

Out[59]:

	col1
a	1
b	2
c	3
d	4

3. Creating a DataFrame from a few Series

In [60]:

```
# given the following three Series, create a DataFrame such that it holds them
ser_1 = pd.Series(np.random.randn(6))
ser_2 = pd.Series(np.random.randn(6))
ser_3 = pd.Series(np.random.randn(6))
```

Pandas questions: Indexing

1. Indexing into a specific column

In [61]:

```
# given the following DataFrame, try to index into the 'col_2' column
df = pd.DataFrame(data={'col_1': [0.12, 7, 45, 10], 'col_2': [0.9, 9, 34, 11]}
                  columns=['col_1', 'col_2', 'col_3'],
                  index=['obs1', 'obs2', 'obs3', 'obs4'])
df
```

Out[61]:

	col_1	col_2	col_3
obs1	0.12	0.9	NaN
obs2	7.00	9.0	NaN
obs3	45.00	34.0	NaN
obs4	10.00	11.0	NaN

2. Label-based indexing

2. Label-based indexing

In [62]:

```
# using the same DataFrame, index into the row whose index is 'obs3'
```

2. Position-based indexing

In [63]:

```
# using the same DataFrame, index into into its first row
```

Mini-Challenge prep: data loading

1. How to load the `users` and `movies` portions of MovieLens

In [64]:

```
import pandas as pd

users = pd.read_table('data/ml-1m/users.dat',
                      sep='::', header=None,
                      names=['user_id', 'gender', 'age', 'occupation', 'zip'])

movies = pd.read_table('data/ml-1m/movies.dat',
                       sep='::', header=None,
                       names=['movie_id', 'title', 'genres'])
```

2. How to load the training and testing subsets

In [65]:

```
# subset version (hosted notebook)
movielens_train = pd.read_csv('data/movielens_train.csv', index_col=0)
movielens_test = pd.read_csv('data/movielens_test.csv', index_col=0)
```

In [66]:

```
movielens_train.head()
```

Out[66]:

	user_id	movie_id	rating	timestamp	gender	age	occupation	zip	title	genres
593263	3562	3798	4	967332344	F	25	6	32812	What Lies Beneath (2000)	Thriller
235597	1051	3793	4	974958593	F	25	0	60513	X-Men (2000)	Action|
219003	3727	2366	3	966309522	M	35	7	74401	King Kong (1933)	Action|
685090	4666	1094	3	963843918	M	35	1	53704	Crying Game, The (1992)	Drama
312377	3261	1095	4	968251750	M	45	20	87505	Glengarry Glen Ross (1992)	Drama

In [67]:

```
movielens_test.head()
```

Out[67]:

	user_id	movie_id	rating	timestamp	gender	age	occupation	zip	title	ge
693323	4653	2648	4	975532459	M	35	12	95051	Frankenstein (1931)	Ho
24177	2259	1270	4	974591524	F	56	16	70503	Back to the Future (1985)	Co
202202	3032	1378	5	970343147	M	25	0	47303	Young Guns (1988)	Ac
262003	3029	2289	4	972846393	M	18	4	92037	Player, The (1992)	Co
777848	4186	2403	3	1017931262	M	25	7	33308	First Blood (1982)	Ac

Mini-Challenge prep: evaluation functions

These are the two functions that you will need to test your `estimate` method.

In [68]:

```
def compute_rmse(y_pred, y_true):
    """ Compute Root Mean Squared Error. """

    return np.sqrt(np.mean(np.power(y_pred - y_true, 2)))
```

In [69]:

```
def evaluate(estimate_f):
    """ RMSE-based predictive performance evaluation with pandas. """

    ids_to_estimate = zip(movielens_test.user_id, movielens_test.movie_id)
    estimated = np.array([estimate_f(u,i) for (u,i) in ids_to_estimate])
    real = movielens_test.rating.values
    return compute_rmse(estimated, real)
```

Test a dummy solution!

In [70]:

```
def my_estimate_func(user_id, movie_id):
    return 3.0
```

You can test for performance with the following line, which assumes that your function is called `my_estimate_func`:

In [73]:

```
print 'RMSE for my estimate function: %s' % evaluate(my_estimate_func)
```

```
RMSE for my estimate function: 1.23237195265
```

Reco systems questions: Minimal reco engine v1.0

1. Simple collaborative filtering using mean ratings

In [72]:

```
# write an 'estimate' function that computes the mean rating of a particular u
def collab_mean(user_id, movie_id):
    # first, index into all ratings of this movie
    # second, compute the mean of those ratings
    #
    return


# try it out for a user_id, movie_id pair
collab_mean(4653, 2648)
```

Mini-Challenge: first round

Implement an `estimate` function of your own using other similarity notions, eg.:

- collaborative filter based on age similarities
- collaborative filter based on zip code similarities
- collaborative filter based on occupation similarities
- content filter based on movie genre

Mini-Challenge: second round

Implement an `estimate` function of your own using other custom similarity notions, eg.:

- euclidean
- cosine