Exercises for "A Beginner's Introduction to Pydata: How to Build a Minimal Recommendation System"

Systems check: imports and files

In [15]:
```
import numpy as np
import pandas as pd
import tables as tb
!find ./data
```

```
./data
./data/.DS_Store
./data/ml-1m
./data/ml-1m/movies.dat
./data/ml-1m/ratings.dat
./data/ml-1m/README
./data/ml-1m/users.dat
./data/movielens_test.csv
./data/movielens_train.csv
```

Systems check: how to load the users and movies portions of MovieLens

In [30]:
```
import pandas as pd

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

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

Systems check: how to load the training and testing subsets

In [2]:
```
# 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)
```

```
print movielens_train
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5838 entries, 593263 to 466639
Data columns:
user_id        5838  non-null values
movie_id       5838  non-null values
rating         5838  non-null values
timestamp      5838  non-null values
gender         5838  non-null values
age            5838  non-null values
occupation     5838  non-null values
zip            5838  non-null values
title          5838  non-null values
genres         5838  non-null values
for_testing    5838  non-null values
dtypes: bool(1), int64(6), object(4)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2668 entries, 693323 to 713194
Data columns:
user_id        2668  non-null values
movie_id       2668  non-null values
rating         2668  non-null values
timestamp      2668  non-null values
gender         2668  non-null values
age            2668  non-null values
occupation     2668  non-null values
zip            2668  non-null values
title          2668  non-null values
genres         2668  non-null values
for_testing    2668  non-null values
dtypes: bool(1), int64(6), object(4)
```

Numpy Questions: Indexing

1. Access an individual element in a NumPy array

In [3]:
```
# given the following ndarray, access the its third element
arr = np.arange(10)
arr
```

Out[3]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

2. Access the last column of a 2d array

In [4]:
```
# given the following ndarray, access its last column
```

```
arr = np.array([[5,4,2,5],[4,5,1,12],[0,1,5,4]])
arr
```

```
Out[4]: array([[ 5,  4,  2,  5],
               [ 4,  5,  1, 12],
               [ 0,  1,  5,  4]])
```

3. Select all elements from a 2d array that are larger than zero

```
In [21]: # given the following ndarray, obtain all elements that are larger than ze
         arr = np.array([[-0.28179535,  1.80896278, -1.08991099, -1.20264003,  0.61
                          [ 0.49983669,  0.28402664, -0.12685554,  0.81266623,  0.96
         arr
```

```
Out[21]: array([[-0.28179535,  1.80896278, -1.08991099, -1.20264003,  0.61651465],
                [ 0.49983669,  0.28402664, -0.12685554,  0.81266623,
          0.96586634]])
```

4. Set all negative values of an array to 1

```
In [22]: # given the following ndarray, set the last two elements to 10
         arr = np.array([1,2,-10,5,-6])
         arr
```

```
Out[22]: array([  1,   2, -10,   5,  -6])
```

Numpy Questions: Operations

1. Compute the sum of a 1D array

```
In [19]: # given the following ndarray, compute its sum
         arr = np.arange(10)
         arr
```

```
Out[19]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

2. Compute the mean of a 1D array

In [20]:
```
# given the following ndarray, compute its mean
arr = np.array([50,-79,80,35])
arr
```

Out[20]:
```
array([ 50, -79,  80,  35])
```

3. How do you detect the presence of NANs in an array?

In [9]:
```
# given the following ndarray, detect all elements that are nans
arr = np.array([np.nan] * 10)
arr[2:4] = 5
arr
```

Out[9]:
```
array([ nan,  nan,   5.,   5.,  nan,  nan,  nan,  nan,  nan,  nan])
```

Pandas questions: Series and DataFrames

1. Adding a column in a DataFrame

In [10]:
```
# given the following DataFrame, add a new column to it
df = pd.DataFrame({'col1': [1,2,3,4]})
df
```

Out[10]:

	col1
0	1
1	2
2	3
3	4

2. Deleting a row in a DataFrame

In [11]:
```
# given the following DataFrame, delete row 'd' from it
df = pd.DataFrame({'col1': [1,2,3,4]}, index = ['a','b','c','d'])
df
```

Out[11]:

	col1

a	1
b	2
c	3
d	4

3. Creating a DataFrame from a few Series

In [12]:
```
# given the following three Series, create a DataFrame such that it holds
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 [32]:
```
# given the DataFrame 'movielens' that we loaded in the previous step, try
# into the 'zip' column
movielens_train[?]
```

2. Label-based indexing

In [29]:
```
# using the same 'movielens' DataFrame, index into the row whose index is
movielens_train.ix[?]
```

Reco systems questions: estimation functions

1. Simple content filtering using mean ratings

In [ ]:
```
# write an 'estimate' function that computes the mean rating of a particul
```

```python
def estimate(user_id, movie_id):
    # first, index into all ratings by this user
    # second, compute the mean of those ratings
    # return


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

2. Simple collaborative filtering using mean ratings

In [ ]:
```python
# write an 'estimate' function that computes the mean rating of a particul
def estimate(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
estimate(4653, 2648)
```

Mini-Challenge

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

In [13]:
```python
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 [14]:
```python
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)
```

In [ ]:
```python
# write your estimate function here
def my_estimate_func(user_id, movie_id):
    # your code
```

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

In [ ]:
```python
print 'RMSE for my estimate function: %s' % evaluate(my_estimate_func)
```

```
In [ ]: print "RMSE for my estimate function: %s" % evaluate(my_estimate_func)
```

Once you are happy with your score, you can submit your RMSE by running this function (in the hosted notebook only):

```
In [ ]: from update_score import update_score
        update_score(evaluate(my_estimate_func))
```

[BONUS] Pytables questions: file and node creation

1. Create a PyTables file in your working environment

```
In [ ]: # write your answer in this code block
```

2. Within the file you created, create a new group

```
In [ ]: # write your answer in this code block
```

3. Within the group you created, create a new array of integers and save it

```
In [ ]: # write your answer in this code block
```

4. For the group created, set a datetime attribute, with the value of ‘utcnow’

```
In [ ]: # write your answer in this code block
```