Exercises for "Hands-on with Pydata"

April 8, 2014

1 Systems check

```
In [1]: import numpy as np
        import pandas as pd
```

2 Numpy Questions: Indexing

2.1 1. Access an individual element in a NumPy array

```
In [2]: # given the following ndarray, access the its third element
        arr = np.arange(10)
        arr

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

2.2 2. Access the last column of a 2d array

```
In [3]: # 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[3]: array([[ 5,  4,  2,  5],
               [ 4,  5,  1, 12],
               [ 0,  1,  5,  4]])
```

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

```
In [4]: # given the following ndarray, obtain all elements that are larger than zero
        arr = np.array([[-0.28179535,  1.80896278, -1.08991099, -1.20264003,  0.61651465],
                        [ 0.49983669,  0.28402664, -0.12685554,  0.81266623,  0.96586634]])
        arr

Out[4]: array([[-0.28179535,  1.80896278, -1.08991099, -1.20264003,  0.61651465],
               [ 0.49983669,  0.28402664, -0.12685554,  0.81266623,  0.96586634]])
```

2.4 4. Set all negative values of an array to 1

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

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

3 Numpy Questions: Operations

3.1 1. Compute the sum of a 1D array

```
In [6]: # given the following ndarray, compute its sum
        arr = np.arange(10)
        arr
```

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

3.2 2. Compute the mean of a 1D array

```
In [7]: # given the following ndarray, compute its mean
        arr = np.array([50,-79,80,35])
        arr
```

```
Out[7]: array([ 50, -79,  80,  35])
```

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

```
In [8]: # given the following ndarray, detect all elements that are nans
        arr = np.array([np.nan] * 10)
        arr[2:4] = 5
        arr
```

```
Out[8]: array([ nan,  nan,   5.,   5.,  nan,  nan,  nan,  nan,  nan,  nan])
```

4 Pandas questions: Series and DataFrames

4.1 1. Adding a column in a DataFrame

```
In [9]: # given the following DataFrame, add a new column to it
        df = pd.DataFrame({'col1': [1,2,3,4]})
        df
```

```
Out[9]:    col1
        0     1
        1     2
        2     3
        3     4

        [4 rows x 1 columns]
```

4.2 2. Deleting a row in a DataFrame

```
In [10]: # given the following DataFrame, delete row 'd' from it
         df = pd.DataFrame({'col1': [1,2,3,4]}, index = ['a','b','c','d'])
         df
```

```
Out[10]:    col1
         a     1
         b     2
         c     3
         d     4

         [4 rows x 1 columns]
```

4.3 3. Creating a DataFrame from a few Series

```
In [11]: # given the following three Series, create a DataFrame such that it holds them in its columns
         ser_1 = pd.Series(np.random.randn(6))
         ser_2 = pd.Series(np.random.randn(6))
         ser_3 = pd.Series(np.random.randn(6))
```

5 Pandas questions: Indexing

5.1 1. Indexing into a specific column

```
In [12]: # 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[12]:

	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

[4 rows x 3 columns]

5.2 2. Label-based indexing

```
In [13]: # using the same DataFrame, index into the row whose index is 'obs_3'
         df
```

Out[13]:

	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

[4 rows x 3 columns]

6 Reco systems questions: Data Loading

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

```
In [14]: 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'])
```

6.2 2. How to load the training and testing subsets

In [15]:
```
# 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 [16]:
```
movielens_train.head()
```

Out[16]:

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

[5 rows x 11 columns]

In [17]:
```
movielens_test.head()
```

Out[17]:

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

[5 rows x 11 columns]

7 Reco systems questions: Estimation Functions

7.1 1. Simple content filtering using mean ratings

In [18]:
```
# write an 'estimate' function that computes the mean rating of a particular user
def estimate1(user_id, movie_id):
    # first, index into all ratings by this user
    # second, compute the mean of those ratings
    # for now, we'll just return None
    return None


# try it out for a user_id, movie_id pair
estimate1(4653, 2648)
```

7.2 2. Simple collaborative filtering using mean ratings

```
In [19]: # write an 'estimate' function that computes the mean rating of a particular user
         def estimate2(user_id, movie_id):
             # first, index into all ratings of this movie
             # second, compute the mean of those ratings
             # for now, we'll just return None
             return None


         # try it out for a user_id, movie_id pair
         estimate2(4653, 2648)
```

8 Mini-Challenge

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

```
In [20]: 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 [21]: 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 [22]: # write your estimate function here
         def my_estimate_func(user_id, movie_id):
             return 3.0
```

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

```
In [23]: print 'RMSE for my estimate function: %s' % evaluate(my_estimate_func)
```

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

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))
```