Exercises02Solutions

February 21, 2015

0.1 Computer lab 02

These exercises provide more practice in data manipulation and working with numpy arrays.

```
In [1]: import os
        import sys
        import glob
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        %matplotlib inline
        %precision 4
        plt.style.use('ggplot')
```

Exercise 1. Write a 12 by 12 times table chart without explicit looping (i.e. no for, while or comprehsnsions). Your code should gneerate this output:

```
[[  1   2   3   4   5   6   7   8   9  10  11  12]
 [  2   4   6   8  10  12  14  16  18  20  22  24]
 [  3   6   9  12  15  18  21  24  27  30  33  36]
 [  4   8  12  16  20  24  28  32  36  40  44  48]
 [  5  10  15  20  25  30  35  40  45  50  55  60]
 [  6  12  18  24  30  36  42  48  54  60  66  72]
 [  7  14  21  28  35  42  49  56  63  70  77  84]
 [  8  16  24  32  40  48  56  64  72  80  88  96]
 [  9  18  27  36  45  54  63  72  81  90  99 108]
 [ 10  20  30  40  50  60  70  80  90 100 110 120]
 [ 11  22  33  44  55  66  77  88  99 110 121 132]
 [ 12  24  36  48  60  72  84  96 108 120 132 144]]
```

```
In [2]: # Your code here

        # version 1
        xs = np.arange(1, 13)
        print xs[:, None] * xs[None, :]
```

```
[[  1   2   3   4   5   6   7   8   9  10  11  12]
 [  2   4   6   8  10  12  14  16  18  20  22  24]
 [  3   6   9  12  15  18  21  24  27  30  33  36]
 [  4   8  12  16  20  24  28  32  36  40  44  48]
 [  5  10  15  20  25  30  35  40  45  50  55  60]
 [  6  12  18  24  30  36  42  48  54  60  66  72]
 [  7  14  21  28  35  42  49  56  63  70  77  84]
 [  8  16  24  32  40  48  56  64  72  80  88  96]
 [  9  18  27  36  45  54  63  72  81  90  99 108]
```

```
 [ 10  20  30  40  50  60  70  80  90 100 110 120]
 [ 11  22  33  44  55  66  77  88  99 110 121 132]
 [ 12  24  36  48  60  72  84  96 108 120 132 144]]
```

```
In [3]: # version 2
        xs = np.arange(1, 13)
        print np.outer(xs, xs)
```

```
[[  1   2   3   4   5   6   7   8   9  10  11  12]
 [  2   4   6   8  10  12  14  16  18  20  22  24]
 [  3   6   9  12  15  18  21  24  27  30  33  36]
 [  4   8  12  16  20  24  28  32  36  40  44  48]
 [  5  10  15  20  25  30  35  40  45  50  55  60]
 [  6  12  18  24  30  36  42  48  54  60  66  72]
 [  7  14  21  28  35  42  49  56  63  70  77  84]
 [  8  16  24  32  40  48  56  64  72  80  88  96]
 [  9  18  27  36  45  54  63  72  81  90  99 108]
 [ 10  20  30  40  50  60  70  80  90 100 110 120]
 [ 11  22  33  44  55  66  77  88  99 110 121 132]
 [ 12  24  36  48  60  72  84  96 108 120 132 144]]
```

Exercise 2. Create a new matrix that normalize the given matrix so that all *columsns* sum to 1.0 without using any loops. Create another matrix so that all *rows* sum to 1.0. In other words, if the 3 matrices were `xs` (given), `ys` (column normalized) and `zs` (row normalized), we would have

`ys.sum(axis=0)` = `[ 1., 1., 1., 1., 1., 1.]`

and

`zs.sum(axis=1)` = `[ 1., 1., 1., 1.]`

Start by creating the following matrix `xs`

```
[[  1.   2.   3.   4.   5.   6.]
 [  7.   8.   9.  10.  11.  12.]
 [ 13.  14.  15.  16.  17.  18.]
 [ 19.  20.  21.  22.  23.  24.]]
```

```
In [7]: # Your code here

        # all columns sum to 1
        xs = np.arange(1.0, 25.0).reshape(4,6)
        ys = xs/np.sum(xs, axis=0)
        print ys
        print np.sum(ys, axis=0)

        print

        # all rows sum to 1
        zs = xs/np.sum(xs, axis=1)[:, np.newaxis]
        print zs
        print np.sum(zs, axis=1)
```

```
[[ 0.025   0.0455  0.0625  0.0769  0.0893  0.1   ]
 [ 0.175   0.1818  0.1875  0.1923  0.1964  0.2   ]
 [ 0.325   0.3182  0.3125  0.3077  0.3036  0.3   ]
 [ 0.475   0.4545  0.4375  0.4231  0.4107  0.4   ]]
[ 1.  1.  1.  1.  1.  1.]
```

```
[[ 0.0476  0.0952  0.1429  0.1905  0.2381  0.2857]
 [ 0.1228  0.1404  0.1579  0.1754  0.193   0.2105]
 [ 0.1398  0.1505  0.1613  0.172   0.1828  0.1935]
 [ 0.1473  0.155   0.1628  0.1705  0.1783  0.186 ]]
[ 1.  1.  1.  1.]
```

Exercise 3. In this exercise, we will practice using Pandas dataframes to explore and summarize a data set `heart`.

This data contains the survival time after receiving a heart transplant, the age of the patient and whether or not the survival time was censored

- Number of Observations - 69
- Number of Variables - 3

Variable name definitions::

- survival - Days after surgery until death
- censors - indicates if an observation is censored. 1 is uncensored
- age - age at the time of surgery

Answer the following questions with respect to the `heart` data set:

- How many patients were censored?
- What is the correlation coefficient between age and survival for uncensored patients?
- What is the average age for censored and uncensored patients?
- What is the average survival time for censored and uncensored patients under the age of 45?
- What is the survival time of the youngest and oldest uncensored patient?

In [9]:
```
import statsmodels.api as sm
heart = sm.datasets.heart.load_pandas().data

heart.head(n=6)
```

Out[9]:

	survival	censors	age
0	15	1	54.3
1	3	1	40.4
2	624	1	51.0
3	46	1	42.5
4	127	1	48.0
5	64	1	54.6

In [25]:
```
# How many patients were censored?

print '# censroed', sum(heart.censors == 0)
print

# What is the correlation coefficient between age and survival for uncensored patients?

uncensored = heart[heart.censors == 1]
print 'Correlation coefficient', np.corrcoef(uncensored.age, uncensored.survival)[0,1]
print

# What is the average age for censored and uncensored patients?

print heart.groupby('censors')['age'].mean()
```

```
        print

        # What is the average survival time for censored and uncensored patients under the age of 45?

        young = heart[heart.age < 45]
        print young.groupby('censors')['survival'].mean()
        print

        # What is the survival time of the youngest and oldest uncensored patient?

        print 'Survival of youngest', uncensored.survival[np.argmin(uncensored.age)]
        print 'Survival of olderst', uncensored.survival[np.argmax(uncensored.age)]
```

```
# censroed 24

Correlation coefficient 0.00325649928321

censors
0          41.729167
1          48.484444
Name: age, dtype: float64

censors
0          712.818182
1          169.909091
Name: survival, dtype: float64

Survival of youngest 228.0
Survival of olderst 60.0
```

Exercise 4. Normalize the given matrix M so that all rows sum to 1.0 (as in Exercise 2). This cna then be considered as a transition matrix P for a Markov chain. Find the stationary distribution of this matrix in the following ways using `numpy` and `numpy.linalg` (or `scipy.linalg`):

- By repeated matrix multiplication of a ranodm probabilty vector v (a row vector normalized to sum to 1.0) with P using matrix multiplication with `np.dot`.
- By raising the matrix P to some large power unitl it doesn't change with higher powers (see `np.linalg.matrix_power`) and then calculating vP
- From the equation for stationarity $wP = w$, we can see that w must be a left eigenvector of P with eigenvalue 1 (Note: np.linalg.eig returns the right eigenvectors, but the left eighenvector of a matrix is the right eigenvector of the transposed matrix). Use this to find w using `np.linalg.eig`.
- Suppose $w = (w_1, w_2, w_3)$. Then from $wP = w$, we have:

$$w_1P_{11} + w_2P_{21} + w_3P_{31} = w_1 \tag{1}$$
$$w_1P_{12} + w_2P_{22} + w_3P_{32} = w_2 \tag{2}$$
$$w_1P_{13} + w_2P_{23} + w_3P_{331} = w_3 \tag{3}$$
$$\tag{4}$$

  This is a singular system, but we also know that $w_1 + w_2 + w_3 = 1$. Use these facts to set up a linear system of equations that can be solved with `np.linalg.solve` to find w.

Given matrix M

```
[[7, 8, 8],
 [1, 3, 8],
 [9, 2, 1]]
```

```
In [79]: M = np.array([7,8,8,1,3,8,9,2,1.0]).reshape(3,3)
         print M
         print

         # Normalize the matrix so that rows sum to 1
         P = M/np.sum(M, 1)[:, np.newaxis]
         print P
         print

         # By repeated matrix multiplication of a ranodm probabilty vector £v£
         v = np.random.random(P.shape[0])
         v /= np.sum(v)
         for i in range(50):
             v = np.dot(v, P)

         # Check that v is a statioary distribution
         print v, '=', np.dot(v, P)
         print

         # By raising the matrix £P£ to some large power
         P50 = np.linalg.matrix_power(P, 50)
         P51 = np.dot(P50, P)
         # check that P50 is stationary
         np.testing.assert_allclose(P50, P51)
         print np.dot(v, P51)
         print

         # Left eigenvector with egienvalue 1
         # note transpose of P to find left eigenvectors
         eigenvalues, eigenvectors = np.linalg.eig(P.T)
         # find index of eigenvalue = 1
         idx = np.argmin(np.abs(eigenvalues - 1))
         w = np.real(eigenvectors[:, idx]).T
         # remember to normalize eigenvector to get a probability distribution
         print w/np.sum(w)
         print

         # Solving linear system
         # Use the first 2 rows of  the matrix P - I = 0
         # and construnct the last row from £w_1 + w_2 + w_3 = 1£
         A = P.T - np.eye(3)
         A[2] = [1,1,1]
         print np.linalg.solve(A, [0,0,1])
```

```
[[ 7.  8.  8.]
 [ 1.  3.  8.]
 [ 9.  2.  1.]]

[[ 0.3043  0.3478  0.3478]
 [ 0.0833  0.25    0.6667]
 [ 0.75    0.1667  0.0833]]

[ 0.3986  0.2606  0.3408] = [ 0.3986  0.2606  0.3408]
```

```
[ 0.3986  0.2606  0.3408]

[ 0.3986  0.2606  0.3408]

[ 0.3986  0.2606  0.3408]
```

In []: