Lab 02 - numpy

Name: Nick Satriano Class: CSCI 349 - Intro to Data Mining Semester: Spring 2023 Instructor: Brian King

In [1]:
```
import sys
import numpy as np
print(sys.version)
print("numpy: " + str(np.__version__))
```

```
3.9.16 (main, Jan 11 2023, 16:16:36) [MSC v.1916 64 bit (AMD64)]
numpy: 1.23.5
```

1) Create a 100000 x 75 matrix of zeros, stored as X. Then, print out the shape of the matrix, the base data type, the individual item size in the array, and the total size of the array in bytes (as an integer). Also, print out the total size in megabytes with 3 significant digits.

In [2]:
```
X = np.zeros((100000, 75))
print("X.size = ", X.size)
print("X.shape = ", X.shape)
print("X.dtype = ", X.dtype)
print("X.itemsize = ", X.itemsize)
print(X.itemsize * X.size, "bytes")
print("%.3f mb" %(X.itemsize * X.size / 1000000))
```

```
X.size =  7500000
X.shape =  (100000, 75)
X.dtype =  float64
X.itemsize =  8
60000000 bytes
60.000 mb
```

2) Resize X to have the same number of elements, but with 100 columns. Show the shape. Show the number of bytes (which should be the same as the previous answer)

In [3]:
```
X = np.reshape(X, (int(X.size/100), 100))
print("X.shape = ", X.shape)
print(X.itemsize * X.size, "bytes")
```

```
X.shape =  (75000, 100)
60000000 bytes
```

3) Redo #1, but use a base datatype of 16-bit integers.

In [4]:
```
X = np.zeros((100000, 75), dtype = np.int16)
print("X.size = ", X.size)
print("X.shape = ", X.shape)
print("X.dtype = ", X.dtype)
print("X.itemsize = ", X.itemsize)
print(X.itemsize * X.size, "bytes")
print("%.3f mb" %(X.itemsize * X.size / 1000000))
```

```
X.size =  7500000
X.shape =  (100000, 75)
X.dtype =  int16
X.itemsize =  2
15000000 bytes
15.000 mb
```

4) How many dimensions does X have? Answer using the appropriate property of np.ndarray objects.

In [5]:
```
print("Number of dimesions: ", X.ndim)
```

```
Number of dimesions:  2
```

5) Enter the following Python list in your cell:

str_nums = ['2.14', '-9.300', '42']

Convert this to a numpy array named X. What is the base type? Show the contents of X. We want numeric representation. So, convert X to an array of single precision floating point numbers. (HINT: use astype). Show X again.

In [6]:
```
str_nums = ['2.14', '-9.300', '42']
X = np.array(str_nums)
print("X.dtype = ", X.dtype)
print(X)
X = X.astype(np.single)
print(X)
```

```
X.dtype =  <U6
['2.14' '-9.300' '42']
[ 2.14 -9.3  42.  ]
```

Let's assume you have two weeks' worth of quiz scores. Quizzes are out of 10 points, and are given every day. Copy the following in a new cell:

In [7]:
```
days = ["Mon","Tue","Wed","Thu","Fri"]
scores_1 = [9.5, 8.75, 8, 10, 7.75]
scores_2 = [9, 8, 9.5, 8.75, 7.25]
```

6) Copy the three definitions above for the Python lists days, scores_1, and scores_2. Create a numpy array called scores that has scores_1 as the first row and scores_2 as the second row using np.concatenate. Then, change days into a np.array from the list days. Show the contents of scores and days, and output their shape.

In [8]:
```
scores = np.concatenate(([scores_1], [scores_2]))
print(scores)
print("scores.shape = ", scores.shape)
print(days)
```

```
[[ 9.5   8.75  8.    10.    7.75]
 [ 9.    8.    9.5   8.75  7.25]]
scores.shape =  (2, 5)
['Mon', 'Tue', 'Wed', 'Thu', 'Fri']
```

7) Repeat the previous problem with the creation of scores from the Python lists scores_1 and scores_2, but now do it with np.vstack. The array should be identical.

In [9]:
```
scores = np.vstack(([scores_1], [scores_2]))
print(scores)
print("scores.shape = ", scores.shape)
```

```
[[ 9.5   8.75  8.   10.    7.75]
 [ 9.    8.    9.5   8.75  7.25]]
scores.shape =  (2, 5)
```

8) Compare the result of the expression days == "Fri" if the variable days was a Python list as entered above, vs. days being a numpy array. What is the difference in the result? In general, how does numpy deal with standard comparison operators?

In [10]:
```
print("If the variable days was a Python list, then the expression days == Fri yields:

days = np.array(days)
print("If the variable days was a numpy array, then the expression days == Fri yields:
```

```
If the variable days was a Python list, then the expression days == Fri yields:  []
If the variable days was a numpy array, then the expression days == Fri yields:  [[7.
75]
 [7.25]]
```

When using a python list, we are not able to print the scores from that particular day in the scores array. When stored as a numpy array, we are able to print the scores which are in the same column as the date.

9) The scores array represents quizzes that are out of 10 pts. Show the scores array scaled to be out of 100 pts (but do not change scores itself.)

In [11]:
```
print(scores * 10)
```

```
[[ 95.   87.5  80.  100.   77.5]
 [ 90.   80.   95.   87.5  72.5]]
```

10) Select the scores that fell on Tuesday.

In [12]:
```
scores[:, days == "Tue"]
```

Out[12]:
```
array([[8.75],
       [8.  ]])
```

11) Select all of the scores that are NOT on Tuesday.

In [13]:
```
scores[:, days != "Tue"]
```

Out[13]:
```
array([[ 9.5 ,  8.  , 10.  ,  7.75],
       [ 9.  ,  9.5 ,  8.75,  7.25]])
```

12) Select the scores that were on Tuesday or Thursday.

In [14]:
```
scores[:, np.where((days == "Tue") | (days == "Thu"))]
```

```
Out[14]: array([[[ 8.75, 10.  ]],

                [[ 8.  ,  8.75]]])
```

13) Show the minimum and maximum scores for the entire array of scores.

```
In [15]: print("min =", np.min(scores))
         print("max =", np.max(scores))

min = 7.25
max = 10.0
```

14) Show the maximum scores for each week as a new array. The result should have the same dimensions.

```
In [16]: scores.max(axis = 1, keepdims = True)

Out[16]: array([[10. ],
                [ 9.5]])
```

15) Report the day that the maximum score occurred each week.

```
In [17]: days[np.argmax(scores, axis = 1)]

Out[17]: array(['Thu', 'Wed'], dtype='<U3')
```

16) Report the mean of the scores of each week.

```
In [18]: np.mean(scores, axis = 1)

Out[18]: array([8.8, 8.5])
```

17) Suppose the lowest score was dropped from each week. Report the mean of each week, but without the minimum score for that week

```
In [23]: minScores = scores.min(axis=1, keepdims=True)
         newScores = scores[np.where(scores != minScores)].reshape(2, 4)
         print(np.mean(newScores, axis = 1, keepdims = True))

[[9.0625]
 [8.8125]]
```

1. Show the number of scores that were > 9 each week as an array.

```
In [93]: scores1 = np.array([0, np.count_nonzero(scores[:1] > 9)])
         print(scores1)
         scores2 = np.array([1, np.count_nonzero(scores[1:2] > 9)])
         print(scores2)

         np.vstack((scores1, scores2))

[0 2]
[1 1]
Out[93]: array([[0, 2],
                [1, 1]])
```

```
In [85]: np.random.seed(1234)
         X = np.random.randint(1,100,50).reshape((10,5))
         X
```

```
Out[85]: array([[48, 84, 39, 54, 77],
                [25, 16, 50, 24, 27],
                [31, 44, 31, 27, 59],
                [93, 70, 81, 74, 48],
                [51, 77, 38, 35, 39],
                [68, 12,  1, 76, 81],
                [ 4,  3, 20, 13, 66],
                [76, 82, 15, 72, 61],
                [47, 29, 82, 88, 14],
                [97, 13, 70, 96, 32]])
```

19) Select the first row of X.

```
In [39]: X[:1]
```

```
Out[39]: array([[48, 84, 39, 54, 77]])
```

20) Select the last column of X.

```
In [40]: X[:, -1:]
```

```
Out[40]: array([[77],
                [27],
                [59],
                [48],
                [39],
                [81],
                [66],
                [61],
                [14],
                [32]])
```

21) Select the first AND last column of X.

```
In [41]: np.concatenate((X[:, :1], X[:, -1:]), axis = 1)
```

```
Out[41]: array([[48, 77],
                [25, 27],
                [31, 59],
                [93, 48],
                [51, 39],
                [68, 81],
                [ 4, 66],
                [76, 61],
                [47, 14],
                [97, 32]])
```

22) Select every other row of X.

```
In [43]: X[::2]
```

```
Out[43]: array([[48, 84, 39, 54, 77],
                [31, 44, 31, 27, 59],
                [51, 77, 38, 35, 39],
                [ 4,  3, 20, 13, 66],
                [47, 29, 82, 88, 14]])
```

23) Show the transpose of X, but don't change X itself.

```
In [44]: np.transpose(X)
```

```
Out[44]: array([[48, 25, 31, 93, 51, 68,  4, 76, 47, 97],
                [84, 16, 44, 70, 77, 12,  3, 82, 29, 13],
                [39, 50, 31, 81, 38,  1, 20, 15, 82, 70],
                [54, 24, 27, 74, 35, 76, 13, 72, 88, 96],
                [77, 27, 59, 48, 39, 81, 66, 61, 14, 32]])
```

24) Select the first column of X and set the result to new variable Y.

```
In [45]: Y = X[:, :1]
         print(Y)
```

```
[[48]
 [25]
 [31]
 [93]
 [51]
 [68]
 [ 4]
 [76]
 [47]
 [97]]
```

25) Increment the first value of Y, then show the corresponding value of X. Did both values in X and Y change?

```
In [46]: Y[0] = Y[0] + 1
         print("X: ", X)
         print("Y: ", Y)
```

```
X:  [[49 84 39 54 77]
 [25 16 50 24 27]
 [31 44 31 27 59]
 [93 70 81 74 48]
 [51 77 38 35 39]
 [68 12  1 76 81]
 [ 4  3 20 13 66]
 [76 82 15 72 61]
 [47 29 82 88 14]
 [97 13 70 96 32]]
Y:  [[49]
 [25]
 [31]
 [93]
 [51]
 [68]
 [ 4]
 [76]
 [47]
 [97]]
```

The first values in both X and Y changed as a result of the change to Y.

26) Repeat exercise 24, but ensure that Y is assigned a copy of the selected data. Increment the first value of Y again and ensure that the corresponding value of X did not change.

In [47]:
```
Y = np.copy(X[:, :1])
Y = X[:, :1]

Y[0] = Y[0] + 1
print("X: ", X)
print("Y: ", Y)
```

```
[[49]
 [25]
 [31]
 [93]
 [51]
 [68]
 [ 4]
 [76]
 [47]
 [97]]
X:  [[50 84 39 54 77]
 [25 16 50 24 27]
 [31 44 31 27 59]
 [93 70 81 74 48]
 [51 77 38 35 39]
 [68 12  1 76 81]
 [ 4  3 20 13 66]
 [76 82 15 72 61]
 [47 29 82 88 14]
 [97 13 70 96 32]]
Y:  [[50]
 [25]
 [31]
 [93]
 [51]
 [68]
 [ 4]
 [76]
 [47]
 [97]]
```

27) Create an array that contains the sequence of numbers 0, 0.1, 0.2, ... 9.8, 9.9 using arange, as a 10x10 matrix, stored as X

In [48]:
```
X = np.arange(10, step = 0.1)
X
```

Out[48]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
       5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
       6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
       7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
       9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])
```

28) Set the RNG seed to 1234. Then create an array X of 100 uniformly distributed numbers, with all values between 1.0 and 10.0. Then, show the mean, the median, the minimum and maximum values of X.

In [49]:
```python
np.random.seed(1234)

X = np.random.uniform(1.0, 10.0, 100)

print("The mean of X is: ", np.mean(X))
print("The median of X is: ", np.median(X))
print("The minimum of X is: ", np.min(X))
print("The maximum of X is: ", np.max(X))
```

```
The mean of X is:  5.665266170909222
The median of X is:  5.827284349552116
The minimum of X is:  1.0558766492841647
The maximum of X is:  9.928733195695253
```

29) Define what is meant by a normal distribution. What are the parameters of a normal distribution?

A normal distribution means that the data is distributed in a bell-shaped curve fashion. A normal distribution is observed through two parameters: the mean and the standard deviation.

30) In simple terms, using a normal distribution, what does the Law of Large Numbers tell us?

The Law of Large Numbers tells us that as our data gets larger and larger, it will more accurately represent a normal distribution.

31) Write a function called test_normal_dist. The purpose of this function is to evaluate the law of large numbers.

In [50]:
```python
def test_normal_dist(mu, sd, vec_length, num_trials):
    np.random.seed(1234)
    deviation = 0
    for i in range(num_trials):
        temp = np.random.normal(mu, sd, vec_length)
        avg = np.mean(temp)
        deviation += abs(avg - mu)
    return deviation / num_trials
```

32) Use test_normal_dist to obtain the deviation for vector lengths of 10, 100, 1000, 10000, and

1. Use a fixed number of trials of 100 for each experiment. Report the results as a numpy array with two dimensions. the first being the vector length, and the second being the average deviation resulting from your test_normal_dist function.

In [77]:
```python
np.set_printoptions(suppress=True)

first = np.array([10, test_normal_dist(10, 2, 10, 100)])
second = np.array([100, test_normal_dist(100, 2, 10, 100)])
third = np.array([1000, test_normal_dist(1000, 2, 10, 100)])
fourth = np.array([10000, test_normal_dist(10000, 2, 10, 100)])
```

```
fifth = np.array([100000, test_normal_dist(100000, 2, 10, 100)])

np.vstack((first, second, third, fourth, fifth))
```

Out[77]:

```
array([[    10.        ,      0.47724084],
       [   100.        ,      0.47724084],
       [  1000.        ,      0.47724084],
       [ 10000.        ,      0.47724084],
       [100000.        ,      0.47724084]])
```

In [ ]: