Lab 02 - numpy

Submission: `lab02.ipynb`, pushed to your Git repo on `master`.
Points: 10
Due: Wednesday, January 22, 2pm

Objectives

- `numpy`. More `numpy`. Becauase, one can never have enough `numpy`.

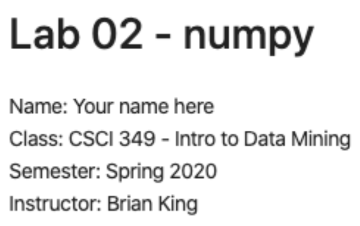
Introduction

The first lab got you set up with your tools, and briefly introduced you to `numpy`. Hopefully you used Google to help you seek good reference material for `numpy` as you were working through the previous lab. A common challenge Python programmers new to `numpy` have to deal with is understanding some of the fundamental differences between Python lists, and `numpy` arrays (`ndarray`). This lab is going to dive into numpy.

NOTE: One final reminder - please understand that this course will require you to seek answers using your own information-seeking skills (and common sense!) Most of these packages we use will have a standard web site that provides a user guide that you will find immensely helpful. For example, the main site for `numpy` is: https://docs.scipy.org/doc/numpy-1.12.0/user/index.html . I also posted a link to a very good book on Moodle (McKinney's book, "Python for Data Analysis, 2nd ed."). I am not going to assign readings out of the books and related material for Python and related packages (numpy, pandas, matplotlib, etc.). There are references for your own use. They also have some great github repos that I find even more informative than the book itself.

Getting started

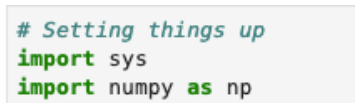
Create you **`lab02.ipynb`** file. Create your header cell (which every submitted file will need)



Lab 02 - numpy

Name: Your name here
Class: CSCI 349 - Intro to Data Mining
Semester: Spring 2020
Instructor: Brian King

Next you should have a cell that sets up your imports that you'll be using.



```
# Setting things up
import sys
import numpy as np
```

You're ready to go. Remember, answer each question with [P] with Python code that produces your answer. No answer should be hard coded! If there is output required, then your code should clearly `print` the answer desired. Generally, any variable as the last line of a cell will automatically be shown in the output. However, if the answer is going to require pages and pages of output, then please do NOT print it unless I specifically ask for the output.

For example, if a line states, "Create a 2x3 matrix of zeros, single precision floating point format, stored as X", then your answer should be something like:

```
X = np.zeros((2,3), dtype=np.float32)
print(X)
```

However, if the line states, "Create a matrix of zeros, stored as integers, of 10000 rows and 100 columns, named X", then please don't show the entire array! A simple cell containing the following will suffice:

```
X = np.zeros((10000,100), dtype=np.float32)
```

Exercises

Some of you did not fill out your **`README.md`**. Please do that and push that file to your remote repo.

1) [P] Create an 52500 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 places of significance.

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

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

You should have a reduction in memory by a factor of 4. There are times when this type of change is simple, but highly effective in speeding up your algorithm. Integer computations will always outperform floating point computations. Use the simplest data type available that can accurately store your data.

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

5) [P] Enter the following list in your cell:

   ```
   str_nums = ['2.14', '-9.3', '42']
   ```

   Convert this to a `numpy` array named `X`. What is the base type? Show the contents of `X`.

   Then, convert `X` to an array of single precision floating point numbers. (HINT: use `astype`). Show `X` again.

Let's assume you have two week's worth of quiz scores. Quizzes are out of 10 points, and are given every day. Enter the following in your cell:

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

The array `scores_1` represents quiz scores from week 1, and `scores_2` is week 2. The `days` array will be used for data selection purposes.

6) [P] Copy the three definitions above for `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.

7) [P] Repeat the previous problem with the creation of `scores` from `scores_1` and `scores_2`, but repeat it with `np.vstack`. The array should be identical.

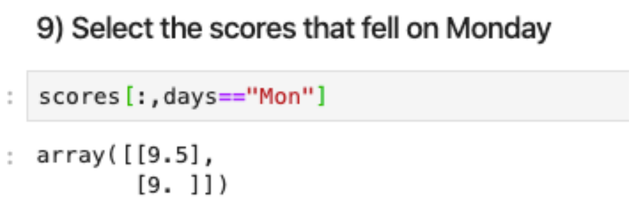
For the next several questions, you are going to exercise some advanced data selection techniques using the `days` array to assist with looking up data. For example, if you are asked to reveal the scores from only Friday, then you must write the numpy selection to do this. For example:

```
scores[:, days == "Fri"]
```

8) [M] 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?**

9) [P] Select the scores that fell on Monday.

   HINT: For cells that ask for only one output, you can sometimes avoid a `print` statement. The expression evaluated as the last line of a cell will also be the output of the cell. For example, if I did not use the `print` function, here's what the answer would look like:



9) Select the scores that fell on Monday

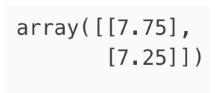
```
: scores[:,days=="Mon"]

: array([[9.5],
         [9. ]])
```

10) [P] Select all of the scores that are NOT on Monday (Hint – look up the ~ operator)

11) [P] Select the scores that were on Tuesday or Thursday

12) [P] Show the minimum and maximum scores for the entire array of scores

13) [P] Show the minimum scores for each week as a new array. The result should have the same dimensions (hint – use the `keepdims` parameter). Your result should be something like:



```
array([[7.75],
       [7.25]])
```

14) [P] Report the day that the *maximum* score occurred each week. (HINT: use `argmax` and use that result to index `days`.)

15) [P] Report the mean of the scores of each week

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

17) [P] Convert the scores to fall on a scale from 0-100 instead of 0-10. Show the result.

Copy and paste this code into a cell:

```
np.random.seed(1234)
X = np.random.randint(1,100,50).reshape((10,5))
X
```

You may use standard selection techniques with integers and slicing for the following exercises. Be sure to show your results of each. (Most of these are one-line statements, and thus you won't need a print function.) For each of these, you should only show the result of the selection. Do NOT reassign X.

18) [P] Select the first row of X

19) [P] Select the last column of X

20) [P] Select the first AND last column of X

21) [P] Select every other row of X

22) [P] Show the transpose of X, but don't change X itself

23) [P] Select the first column of X and set the result to Y.

24) [P] Increment the first value of Y, then show the corresponding value of X. Did both values in X and Y change? THIS IS AN IMPORTANT OBSERVATION WHEN DEALING WITH SELECTING DATA!

25) [P] Repeat exercise 23, 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.

26) [P] 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`.

27) [P] 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.

28) [M] Define what is meant by a **normal distribution**. What are the parameters of a normal distribution?

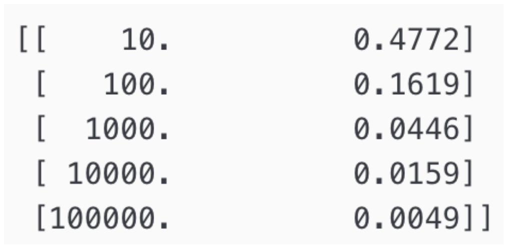
29) [M] In simple terms, using a normal distribution, what does the Law of Large Numbers tell us?

30) [P] Write a function called `test_normal_dist`. The purpose of this function is to evaluate the law of large numbers. It should have four parameters:

    `mu` = mean of distribution
    `sd` = standard deviation
    `vec_length` = length of the vector to generate randomly from a normal distribution, with `mu` and `sd` as parameters
    `num_trials` = number of times to repeat the experiment

    The function should behave as follows. First, initialize the seed value to 1234, before your loop. Then, loop for `num_trials`, generating `vec_length` numbers from a normal distribution. Compute the mean of that vector, then compute its deviation (absolute value of the observed mean - expected mean). This should be repeated for all trials, and then return the *average* deviation over all trials.

31) [P] Use `test_normal_dist` to obtain the deviation for vector lengths of 10, 100, 1000, 10000, and 100000. 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. Your resulting array should look like:



```
[[     10.            0.4772]
 [    100.            0.1619]
 [   1000.            0.0446]
 [  10000.            0.0159]
 [ 100000.            0.0049]]
```

In other words, your results should confirm the Law of Large Numbers.

Deliverables

Commit and push `lab02.ipynb`. Be sure you have every cell run, and output generated. Verify that your file is pushed properly on Gitlab.