Homework 0

Due Tuesday, September 10 (but no submission is required)

Welcome to CS109 / STAT121 / AC209 / E-109 (http://cs109.org/). In this class, we will be using a variety of tools that will require some initial configuration. To ensure everything goes smoothly moving forward, we will setup the majority of those tools in this homework. While some of this will likely be dull, doing it now will enable us to do more exciting work in the weeks that follow without getting bogged down in further software configuration. This homework will not be graded, however it is essential that you complete it timely since it will enable us to set up your accounts. You do not have to hand anything in, with the exception of filling out the online survey.

Class Survey, Piazza, and Introduction

Class Survey

Please complete the mandatory course survey located here (https://docs.google.com/spreadsheet/viewform?formkey=dFg1ZFJwLWJ6ZWhWR1JJb0tES3lGMEE6MA#gid=0). It should only take a few moments of your time. Once you fill in the survey we will sign you up to the course forum on Piazza and the dropbox system that you will use to hand in the homework. It is imperative that you fill out the survey on time as we use the provided information to sign you up for these services.

Piazza

Go to Piazza (https://piazza.com/harvard/fall2013/cs109/home) and sign up for the class using your Harvard e-mail address.

You will use Piazza as a forum for discussion, to find team members, to arrange appointments, and to ask questions. Piazza should be your primary form of communication with the staff. Use the staff e-mail (staff@cs109.org) only for individual requests, e.g., to excuse yourself from a mandatory guest lecture. All readings, homeworks, and project descriptions will be announced on Piazza first.

Introduction

Once you are signed up to the Piazza course forum, introduce yourself to your classmates and course staff with a follow-up post in the introduction thread. Include your name/nickname, your affiliation, why you are taking this course, and tell us something interesting about yourself (e.g., an industry job, an unusual hobby, past travels, or a cool project you did, etc.). Also tell us whether you have experience with data science.

Programming expectations

All the assignments and labs for this class will use Python and, for the most part, the browser-based IPython notebook format you are currently viewing. Knowledge of Python is not a prerequisite for this course, **provided you are comfortable learning on your own as needed**. While we have strived to make the programming component of this course straightforward, we will not devote much time to teaching prorgramming or Python syntax. Basically, you should feel comfortable with:

- How to look up Python syntax on Google and StackOverflow.
- Basic programming concepts like functions, loops, arrays, dictionaries, strings, and if statements.
- How to learn new libraries by reading documentation.
- Asking questions on StackOverflow or Piazza.

There are many online tutorials to introduce you to scientific python programming. Here is one (https://github.com/jrjohansson/scientific-python-lectures) that is very nice. Lectures 1-4 are most relevant to this class.

Getting Python

You will be using Python throughout the course, including many popular 3rd party Python libraries for scientific computing. Anaconda (http://continuum.io/downloads) is an easy-to-install bundle of Python and most of these libraries. We recommend that you use Anaconda for this course.

Please visit this page (https://github.com/cs109/content/wiki/Installing-Python) and follow the instructions to set up Python

Hello, Python

The IPython notebook is an application to build interactive computational notebooks. You'll be using them to complete labs and homework. Once you've set up Python, please download this page (https://raw.github.com/cs109/content/master/HW0.ipynb), and open it with IPython by typing

```
ipython notebook <name_of_downloaded_file>
```

For the rest of the assignment, use your local copy of this page, running on IPython.

Notebooks are composed of many "cells", which can contain text (like this one), or code (like the one below). Double click on the cell below, and evaluate it by clicking the "play" button above, for by hitting shift + enter

In [1]:
```
x = [10, 20, 30, 40, 50]
for item in x:
    print "Item is ", item
```

Python Libraries

We will be using a several different libraries throughout this course. If you've successfully completed the installation instructions (https://github.com/cs109/content/wiki/Installing-Python), all of the following statements should run.

In [2]:
```
#IPython is what you are using now to run the notebook
import IPython
print "IPython version:      %6.6s (need at least 1.0)" % IPython.__vers
ion__

# Numpy is a library for working with Arrays
import numpy as np
print "Numpy version:       %6.6s (need at least 1.7.1)" % np.__version
__

# SciPy implements many different numerical algorithms
import scipy as sp
print "SciPy version:       %6.6s (need at least 0.12.0)" % sp.__versio
n__

# Pandas makes working with data tables easier
import pandas as pd
print "Pandas version:      %6.6s (need at least 0.11.0)" % pd.__versio
n__

# Module for plotting
import matplotlib
print "Mapltolib version:   %6.6s (need at least 1.2.1)" % matplotlib._
_version__

# SciKit Learn implements several Machine Learning algorithms
import sklearn
print "Scikit-Learn version: %6.6s (need at least 0.13.1)" % sklearn.__v
ersion__

# Requests is a library for getting data from the Web
import requests
print "requests version:    %6.6s (need at least 1.2.3)" % requests.__v
ersion__

# Networkx is a library for working with networks
import networkx as nx
print "NetworkX version:    %6.6s (need at least 1.7)" % nx.__version__

#BeautifulSoup is a library to parse HTML and XML documents
import BeautifulSoup
print "BeautifulSoup version:%6.6s (need at least 3.2)" % BeautifulSoup.
__version__
```

```
#MrJob is a library to run map reduce jobs on Amazon's computers
import mrjob
print "Mr Job version:       %6.6s (need at least 0.4)" % mrjob.__versio
n__

#Pattern has lots of tools for working with data from the internet
import pattern
print "Pattern version:      %6.6s (need at least 2.6)" % pattern.__vers
ion__
```

If any of these libraries are missing or out of date, you will need to install them (https://github.com/cs109/content/wiki/Installing-Python#installing-additional-libraries) and restart IPython

Hello matplotlib

The notebook integrates nicely with Matplotlib, the primary plotting package for python. This should embed a figure of a sine wave:

In [3]:
```
#this line prepares IPython for working with matplotlib
%matplotlib inline

# this actually imports matplotlib
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 30)  #array of 30 points from 0 to 10
y = np.sin(x)
z = y + np.random.normal(size=30) * .2
plt.plot(x, y, 'ro-', label='A sine wave')
plt.plot(x, z, 'b-', label='Noisy sine')
plt.legend(loc = 'lower right')
plt.xlabel("X axis")
plt.ylabel("Y axis")
```

If that last cell complained about the `%matplotlib` line, you need to update IPython to v1.0, and restart the notebook. See the installation page (https://github.com/cs109/content/wiki/Installing-Python)

Hello Numpy

The Numpy array processing library is the basis of nearly all numerical computing in Python. Here's a 30 second crash course. For more details, consult Chapter 4 of Python for Data Analysis, or the Numpy User's Guide (http://docs.scipy.org/doc/numpy-dev/user/index.html)

In [4]:
```
print "Make a 3 row x 4 column array of random numbers"
x = np.random.random((3, 4))
```

```
print x
print

print "Add 1 to every element"
x = x + 1
print x
print

print "Get the element at row 1, column 2"
print x[1, 2]
print

# The colon syntax is called "slicing" the array.
print "Get the first row"
print x[0, :]
print

print "Get every 2nd column of the first row"
print x[0, ::2]
print
```

Print the maximum, minimum, and mean of the array. This does **not** require writing a loop. In the code cell below, type `x.m<TAB>`, to find built-in operations for common array statistics like this

In [5]:
```
#your code here
```

Call the `x.max` function again, but use the `axis` keyword to print the maximum of each row in x.

In [6]:
```
#your code here
```

Here's a way to quickly simulate 500 coin "fair" coin tosses (where the probabily of getting Heads is 50%, or 0.5)

In [7]:
```
x = np.random.binomial(500, .5)
print "number of heads:", x
```

Repeat this simulation 500 times, and use the plt.hist() function (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist) to plot a histogram of the number of Heads (1s) in each simulation

In [8]:
```
#your code here
```

The Monty Hall Problem

Here's a fun and perhaps surprising statistical riddle, and a good way to get some practice writing python functions

In a gameshow, contestants try to guess which of 3 closed doors contain a cash prize (goats are behind the other two doors). Of course, the odds of choosing the correct door are 1 in 3. As a twist, the host of the show occasionally opens a door after a contestant makes his or her choice. This door is always one of the two the contestant did not pick, and is also always one of the goat doors (note that it is always possible to do this, since there are two goat doors). At this point, the contestant has the option of keeping his or her original choice, or swtiching to the other unopened door. The question is: is there any benefit to switching doors? The answer surprises many people who haven't heard the question before.

We can answer the problem by running simulations in Python. We'll do it in several parts.

First, write a function called `simulate_prizedoor`. This function will simulate the location of the prize in many games -- see the detailed specification below:

In [9]:
```
"""
Function
--------
simulate_prizedoor

Generate a random array of 0s, 1s, and 2s, representing
hiding a prize between door 0, door 1, and door 2

Parameters
----------
nsim : int
    The number of simulations to run

Returns
-------
sims : array
    Random array of 0s, 1s, and 2s

Example
-------
>>> print simulate_prizedoor(3)
array([0, 0, 2])
"""
def simulate_prizedoor(nsim):
    #compute here
    return answer
#your code here
```

Next, write a function that simulates the contestant's guesses for `nsim` simulations. Call this function `simulate_guess`. The specs:

In [10]:
```
"""
Function
--------
simulate_guess
```

```
Return any strategy for guessing which door a prize is behind. This
could be a random strategy, one that always guesses 2, whatever.

Parameters
----------
nsim : int
    The number of simulations to generate guesses for

Returns
-------
guesses : array
    An array of guesses. Each guess is a 0, 1, or 2

Example
-------
>>> print simulate_guess(5)
array([0, 0, 0, 0, 0])
"""
#your code here
```

Next, write a function, goat_door, to simulate randomly revealing one of the goat doors that a contestant didn't pick.

In [11]:
```
"""
Function
--------
goat_door

Simulate the opening of a "goat door" that doesn't contain the prize,
and is different from the contestants guess

Parameters
----------
prizedoors : array
    The door that the prize is behind in each simulation
guesses : array
    THe door that the contestant guessed in each simulation

Returns
-------
goats : array
    The goat door that is opened for each simulation. Each item is 0, 1,
or 2, and is different
    from both prizedoors and guesses

Examples
```

```
--------
>>> print goat_door(np.array([0, 1, 2]), np.array([1, 1, 1]))
>>> array([2, 2, 0])
"""
#your code here
```

Write a function, `switch_guess`, that represents the strategy of always switching a guess after the goat door is opened.

In [12]:
```
"""
Function
--------
switch_guess

The strategy that always switches a guess after the goat door is opened

Parameters
----------
guesses : array
     Array of original guesses, for each simulation
goatdoors : array
     Array of revealed goat doors for each simulation

Returns
-------
The new door after switching. Should be different from both guesses and g
oatdoors

Examples
--------
>>> print switch_guess(np.array([0, 1, 2]), np.array([1, 2, 1]))
>>> array([2, 0, 0])
"""
#your code here
```

Last function: write a `win_percentage` function that takes an array of `guesses` and `prizedoors`, and returns the percent of correct guesses

In [13]:
```
"""
Function
--------
win_percentage

Calculate the percent of times that a simulation of guesses is correct

Parameters
-----------
```

```
guesses : array
    Guesses for each simulation
prizedoors : array
    Location of prize for each simulation

Returns
--------
percentage : number between 0 and 100
    The win percentage

Examples
---------
>>> print win_percentage(np.array([0, 1, 2]), np.array([0, 0, 0]))
33.333
"""
#your code here
```

Now, put it together. Simulate 10000 games where contestant keeps his original guess, and 10000 games where the contestant switches his door after a goat door is revealed. Compute the percentage of time the contestant wins under either strategy. Is one strategy better than the other?

In [14]:
```
#your code here
```

Many people find this answer counter-intuitive (famously, PhD mathematicians have incorrectly claimed the result must be wrong. Clearly, none of them knew Python).

One of the best ways to build intuition about why opening a Goat door affects the odds is to re-run the experiment with 100 doors and one prize. If the game show host opens 98 goat doors after you make your initial selection, would you want to keep your first pick or switch? Can you generalize your simulation code to handle the case of n doors?
