Welcome to EECS 731. 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 on time so that you can start work on the project and all other homeworks.

1.2 Piazza, and Introduction (due Thursday August 24)

Piazza

Go to Piazza and sign up for the class using your Kansas e-mail address.

You will use Piazza as a forum for discussion, to coordinate with team members, to arrange appointments, and to ask questions. Piazza should be your primary form of communication with me. Use my e-mail only for individual requests, e.g., to excuse yourself from a mandatory guest lecture.

All readings, homeworks, and project descriptions will be updated on the website but Piazza is otherwise the main communication for this course.

Introduction

Once you are signed up to the Piazza course forum, introduce yourself to your classmates and me 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, statistics courses and research projects.

1.3 Programming expectations

All the assignments for this class will use R and, for the most part, the browser-based IPython notebook format you are currently viewing. Knowledge of R is not a prerequisite for this course, provided you are comfortable learning on your own as needed. While I have strived to make the programming component of this course straightforward, I will not devote any time to teaching programming or R syntax. Basically, you should feel comfortable with:
• How to look up R syntax on Google and StackOverflow.
• How to read an R help page
• 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 R programming. Here is one that is useful.

1.4 Getting R

You will be using R throughout the course, including many popular 3rd party R packages for scientific computing (dplyr, ggplot2, forecast, sna, etc). You can download R from the CRAN. Many of you new to R may find R-studio a useful tool. Anaconda is an easy-to-install bundle for R-studio and most of these libraries and is the easiest way to adapt Jupyter notebooks to the R language. I recommend that you use Anaconda for this course.

1.5 Hello R

The Jupyter notebook is an application to build interactive computational notebooks. You’ll be using them to complete labs and homework. Once you’ve set up R and modified the default Jupyter notebook setup to allow for R programming language (see here for recommended instructions), please download this HW0 ipython notebook and open it with R by typing

```R
ipython notebook <name_of_downloaded_file>
```

For the rest of the assignment, use your local copy of this page, running on jupyter notebook with the language as R. Note if you cannot see R as the language, you have not correctly updated jupyter notebook.

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 [2]: x = c(10, 20, 30, 40, 50)
   for (item in x) {
        print(paste("Item is", item))
   }
```

[1] "Item is 10"
[1] "Item is 20"
[1] "Item is 30"
[1] "Item is 40"
[1] "Item is 50"

```
In [4]: for (item in seq(from=10, to=50, by=10)) {print(paste(item, 'is the item'))}
```
1.6 Installing Packages

For other packages that don’t come with the default anaconda R package, open R or R-studio on your local machine. To launch R from the terminal/powershell, type R after the prompt. This should launch R (I am on a mac, so no promises this works for everyone).

From here you can install packages from inside of R via the regular install.packages (from CRAN mirrors), or devtools::install_github (from GitHub), they work fine. @alistaire

How to do this for jupyter: Open your (independent) R installation, then run the following command:
install.packages("packagename", lib="~/anaconda2/lib/R/library", dependencies=T) to add new package to the correct R library used by Jupyter, otherwise the package will be installed locally as mentioned in .libPaths().

Try installing package 'ISLR'. It is the codebase for the book we will read for some assignments.

In the future you may need to install other packages for particular assignments. Please make sure you feel comfortable doing this.

In [1]: library(ISLR)
   sessionInfo()

R version 3.4.1 (2017-06-30)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: macOS Sierra 10.12.6

Matrix products: default
BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versions/A/libBLAS.dylib
LAPACK: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versions/A/libLAPACK.dylib

locale:

attached base packages:
[1] stats graphics grDevices utils datasets methods base

other attached packages:
[1] ISLR_1.0

loaded via a namespace (and not attached):
[1] compiler_3.4.1 R6_2.2.1 magrittr_1.5 IRdisplay_0.4.4
[5] pbdZMQ_0.2-6 tools_3.4.1 crayon_1.3.2 uuid_0.1-2
[9] stringi_1.1.5 IRkernel_0.7.1 jsonlite_1.5 stringr_1.2.0
[13] digest_0.6.12 repr_0.10 evaluate_0.10

1.7 Hello ggplot2

The notebook mostly integrates nicely with ggplot2, the premier plotting package for R. Let's use a built in data set and visualize it.

In [1]: head(iris) # data set that's already in R environment
   options(repr.plot.width=4, repr.plot.height=3) # control size of figure for submission
   ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width)) + geom_point(size=1.5)
   # the last plot looks okay but it doesn't help us understand the data. Instead let's add
   # if there are different characteristics for each species. We'll also add a title and
   options(repr.plot.width=5.2, repr.plot.height=3) # control size of figure for submission
   ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) + geom_point(size=1.5)
   ggtitle('Iris species', subtitle = NULL) + xlab('Length of Sepal') + ylab('Width of Sepal')
### Table

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
</tbody>
</table>

Error in ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)): could not find function "ggplot"

Traceback:

1.8 **Hello R**

Most functions you will need are built into R and don’t require downloading any packages. Here’s a 30 second crash course. For more details, consult ISLR or R tutorials. Note that the default class for many R structures is either "numeric" (array) or "matrix" (matrix) but for many things we need "data.frame". If you run into an error check the class of the variable (class(var)).

```
In [41]: # getting help in R
    ?runif()
```

```
In [2]: print("Make a 3 row x 4 column array of continuous uniform random numbers between 0 and 10")
    x = matrix(runif(12, min=0, max=10), ncol=4)
    print(x)
    cat('
')
    print("Print the dimensions of x")
    print(dim(x))
    print(paste(dim(x)[1], 'rows'))
    print(paste(dim(x)[2], 'columns'))
    cat('
')
    print("Add 1 to every element")
    x = x + 1
    print(x)
    cat('
')
    print("Get the element at row 1, column 2")
    print(x[1, 2])
    cat('
')
    # The colon syntax is called "slicing" the array.
    print("Get the first row")
    print(x[1, ])
    cat('
')
```

print("Get every 2nd column of the first row")
print(x[1, seq(from=2,by=2,to=4)])
cat("\n")

[1] "Make a 3 row x 4 column array of continuous uniform random numbers between 0 and 10"
[1,]  7.043562  9.119865  7.987828  9.704893
[2,]  5.339570  0.5826924  4.597450  6.781032
[3,]  6.924235  6.7049252  7.350322  6.017965

[1] "Print the dimensions of x"
[1] 3 4
[1] "3 rows"
[1] "4 columns"

[1] "Add 1 to every element"
[1,]  8.043562 10.119865  8.987828 10.704893
[2,]  6.339570  1.582692  5.597450  7.781032
[3,]  7.924235  7.704925  8.350322  7.017965

[1] "Get the element at row 1, column 2"
[1] 10.11986

[1] "Get the first row"

[1] "Get every 2nd column of the first row"
[1] 10.11986 10.70489

Display the maximum, minimum, and mean of the array. This does not require writing a loop.
In [1]: # your code here

Call the max function again, but use the apply function to display the maximum of each row in x. You should not need to use a loop here.

In [2]: # your code here

Here’s a way to quickly simulate 500 coin "fair" coin tosses (where the probability of getting Heads is 50%, or 0.5)
In [2]: ?rbinom()
   flips=rbinom(n=1,size=500, prob=.5)
   print(paste(flips, 'flips came up heads'))
[1] "264 flips came up heads"
Repeat this simulation 500 times. Summarize the results using `summary()`. Use the `ggplot2 geom_histogram()` function to plot a histogram of the number of Heads (1s) in each simulation.

- **Hint 1**: `ggplot2` needs a dataframe. To convert things to data frame use `data.frame(x)`.
- **Hint 2**: It may throw an error but still plot. You can get rid of some of the error by using `scale_x_continuous(limits=c(lower,upper))`.
- **Hint 3**: Make sure to have a title and meaningful labels on your axes.

```
In [1]: # your code here
```

### 1.9 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 game show, 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 switching 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 []: ""

Function
--------
simulate_prizedoor

Generate a random array of 1s, 2s, and 3s, representing hiding a prize between door 1, door 2, and door 3

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

Returns
-------
sims : array
   Random array of 1s, 2s, and 3s

Example
-------
>>> print(simulate_prizedoor(3))
[1] 1 1 3
"""
Next, write a function that simulates the contestant’s guesses for \( n_{\text{sim}} \) simulations. Call this function `simulate_guess`. The specs:

```python
In [ ]: ""
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 1, 2, or 3

Example
-------
>>> print simulate_guess(5)
[1] 1 1 1 1 1
""

#your code goes here
```

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

```python
In [ ]: ""
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 1, 2, or 3, and is
different from both prizedoors and guesses

Examples
-------
>>> print goat_door(c(1, 2, 3), c(2, 2, 2))
[1] 3 3 1

#your code here

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

In [ ]: ""
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 goatdoors

Examples
-------
>>> print switch_guess(c(1, 2, 3), c(2, 3, 2))
[1] 3 1 1

#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 [ ]: ""
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(c(1, 2, 3), c(1, 1, 1))
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 [ ]: #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?

In [ ]: