Homework 3

[ 100 points - due by 11:59 pm, Sunday, February 19, 2017 ]

Submit these files to the CS submission system at the usual place by 11:59. You may work on your own or with 1-2 partners on the programming portions of this assignment. (The reading/response is individual only.) Groups larger than 3, please split into smaller groups! Remember that partners need to work in the same physical location, share composition time equally (or each compose on their own machines) and be fully equal owners and producers of their work. *Have fun experimenting!*  [cs35 homepage](https://www.cs.hmc.edu/~dodds/cs35/)

**Downloads**

Starter files to download -- grab them at the start of class & follow along:

* [The zip file to start all of this week's problems…](https://drive.google.com/open?id=0BwPWh-3AmiLxZmtWZXlMcE5pRTA)
* Here is the [rps.csv](https://drive.google.com/open?id=0BwPWh-3AmiLxUjlXWWkzNHR1VlU) file that we created in class (plus some machine-generated strings!)
* [hw3pr4](https://github.com/ScriptingBeyondCS/CS-35/tree/master/week_3)

**Submission**

Again we ask you to submit a zipped archive named hw3.zip -- our graders have wisely (!) asked that we standardize our filenames. So, this week we ask for at least these five files:

hw3pr1.py which follows the matplotlib/visualization tutorial

plot\_scatter.png (or another image type) the first "other" plot's variation

plot\_bar.png (or another image type) the second "other" plot's variation

plot\_choice.png (or another image type) your third choice from the "other" plot's novel variation

hw3pr2.py which will end up classifying each rps-string in rps.csv as *human* or *machine*

hw3pr3.py which creates two new visualizations using matplotlib and your own choice of data

datavis1.png (or another image type) the first data-visualization

datavis2.png (or another image type) the second data-visualization

As usual, submit your reading response in its own spot at the [submission site](http://cicero.cs.hmc.edu/).

As always, extra-credit is available for posting code and a write-up of any one of these problems to your GitHub repository (be sure to let us know you've done this -- and provide a direct link)

**Problem 0**: *Developing an ethics of AI* [5 pts]

This week's reading is a recent [NYTimes article](https://www.nytimes.com/2016/09/02/technology/artificial-intelligence-ethics.html) on the development of an "Ethics of AI." It covers two recent events: first, a recent effort by large tech companies to agree on principles for AI development. The second is a deeper, longer-term effort named the [100-year study on Artificial Intelligence](https://ai100.stanford.edu/2016-report/preface), planning regular reports every five years. (Here's the [first](https://ai100.stanford.edu/sites/default/files/ai100report10032016fnl_singles.pdf).)

After reading over that NYTimes article, reflect on your agreement or disagreement -- or a completely different take -- on one of these two quotes from the Stanford 100-year report:

1. *"Contrary to the more fantastic predictions for AI in the popular press, the Study Panel found no cause for concern that AI is an imminent threat to humankind."*  ([p.4](https://ai100.stanford.edu/sites/default/files/ai100report10032016fnl_singles.pdf))
2. *"As a society, we are ... determining how to deploy AI-based technologies in ways that promote, not hinder, democratic values such as freedom, equality, and transparency."* ([p.6](https://ai100.stanford.edu/sites/default/files/ai100report10032016fnl_singles.pdf))

As with each week's reading, responses should carefully considered, but need not be very long (4-5 sentences is wonderful).

**[Lab problem] Problem 1: Data Visualization Tutorial**

[30 pts: 15 for the tutorial and 5 each for your custom variations on the "other" plots]

* This problem asks you to write all of your code in the **hw3pr1.py** file.
* In addition, you'll submit three screenshot files (their names are noted below…)
* This problem provides a hands-on introduction to the widely-used matplotlib visualization library in Python. In addition, you'll experience a bit of the numpy (numeric python) library -- the numpy library is also a large part of the machine-learning routines we'll use over the next couple of weeks.
* The matplotlib and numpy libraries are so large, however, that it may be better to dive in and try out different plotting commands, styles, attributes, and techniques via tutorial, rather than from documentation. This lab problem asks you to work through a tutorial on matplotlib graphics -- and to create and submit a variation each step of the way. The tutorial is at this site: <http://www.labri.fr/perso/nrougier/teaching/matplotlib/>
* [**work through the tutorial...**] In particular, this problem asks you to
  + read the short introduction
  + Use ipython's %matplotlib command to help render graphics smoothly
  + Read through and try the ***Simple Plot*** series of [tutorials](http://www.labri.fr/perso/nrougier/teaching/matplotlib/). By the end, all of them should have worked...
  + Feel free to look at the next two sections (optional), entitled "figures, subplots, axes, and ticks" and "animation." Entirely optional, though, the animation is neat!
  + Look over the "Other Types of Plots" section and ***do*** complete the challenges for two other types of plots - specifically, the very useful
    - *Scatter Plot*  [with a hint on changing points by color](http://matplotlib.org/examples/pie_and_polar_charts/polar_scatter_demo.html) and
    - *Bar Plot* and then
  + Choose one more type of plot from the "Other Types of Plots" section and try that one out, as well…
    - Note that, if you try the *Polar* plot, you'll need to add the line of code

import matplotlib.cm as cm in order for that to work

* [**Create three screenshots of your other plots**] For each of these last three types of plots, create a *distinctive variation of your own design* of the given example and include a screenshot or a saved-image of each in your hw3.zip folder. For instance, you could (a) change shapes, (b) change colormaps, (c) alter the underlying data in an unusual way, (d) add your own text, or (e) check out any of the very large matplotlib documentation and try one novel feature out. Include your screenshots as
  + - *plot\_scatter.png*
    - *plot\_bar.png* and
    - *plot\_choice.png*
* Of course, you're welcome to delve further -- remember that, if you join one of the labs, the lab incentive applies: credit for this problem, even if you don't make it to the end of all of the above challenges… Also, since this kind of tutorial is a bit different than other problems, if you feel strongly -- positive or negative -- about having tutorials such as this one as part of the course, please do let me know… . And, if you \_like\_ tutorials, here's the main [Numpy tutorial](http://www.numpy.org/)

**Problem 2: Batch-mode RPS ~ *Human or Machine?***

[35 pts; with EC certainly available!]

* This problem asks you to write all of your code in the **hw3pr2.py** file.
* In addition, you'll download the [**rps.csv**](https://drive.google.com/open?id=0BwPWh-3AmiLxUjlXWWkzNHR1VlU) file and submit an augmented version of it.
* In lecture, we will create a csv file (from a Google spreadsheet) that includes everyone's "batch-mode" rock-paper-scissors entries. Each of these will be a string of 200 or more of the characters r, p, and s. Grab it from this [rps.csv link](https://drive.google.com/open?id=0BwPWh-3AmiLxUjlXWWkzNHR1VlU).
* Your challenge is *not* to determine who wins in rock, paper, and scissors with these strings -- that was a prior course's challenge (and extra-credit, if you'd like). Instead, this problem asks you to determine which rps strings were human-made and which were "machine-made"…
* First, write a function **extract\_features( rps )** that
  + takes in a single rps string named, naturally, **rps**
  + creates a default dictionary that counts ***something*** about the data in the **rps** string -- the exact thing(s) you'd like to measure or count are up to you. Each of those things you measure or count are usually called *features* in a classification or machine-learning application. For example, one feature could be *the number of the character 's' in the string.* (This is not necessarily a *useful* feature!)
  + your function should return that dictionary of features
  + a starter function that *does* count the number of 's's (and only that) is provided.
* Then, you should write another function named **score\_features( dict\_of\_features )** that
  + takes in a dictionary of features (the output of **extract\_features**)
  + and it returns a single floating-point number that "scores" how human-made or how machine-made those features are. *This algorithm is entirely up to you.* For example, higher values might mean "more likely to be human-made," or vice-versa -- or some completely different approach. The values do *not* have to be probabilities, but it's ok if they are (or a variant). The goal is to compute a quantity -- a "score" -- that can distinguish human-made rps strings from machine-made rps strings.
  + Note that the machine-created rps strings were generated by choosing r, p, or s uniformly randomly and independently for each spot in the string… ***Hint***: people will \_not\_ have chosen their characters that way!
* You should include, in the triple-quoted string provided at the top of the file, a description of what features your **extract\_features** function computes and how they're being used by your **score\_features** function to determine which batch-rps strings are machine-like and which are more human-like.
* Finally, write a function **read\_data( filename="rps.csv" )** that
  + uses prior weeks' experience as a guide to open the file of **filename** (by default, "rps.csv") and extracts all of its data
  + it should return a *list* of all of the rps strings in that csv file (it will need to grab them from the appropriate column)
  + watch out -- there is a header row in the rps.csv file. Be sure your code doesn't consider it a data row… .
* Then, use these three functions to give a score to *each* of the batch rps strings in the rps.csv file you downloaded. Include those scores in your rps.csv file, either by writing out a new file or simply by pasting the scores into the appropriate column in the csv file. Then, make a decision as to which strings were human and which were machine and place the string "human" or "machine" in the appropriate column in the csv file.
* Half of the strings are human-generated and half are machine-generated. And -- there is one additional string (there are an odd number of strings). An additional challenge is to determine if that "extra" string was generated by a person or a machine. 20 of the 35 points on this problem are for implementing the above functions and the write-up. The final 15 of the 35 points is allocated based on how discerning your algorithm is -- how well it tells the difference between machine and human rps strings!
* Be sure to include docstrings inside your functions, as usual…
* ***EC Options***: For up to +5 points extra-credit, also create a function named **batch\_play( rps1, rps2 )** that takes two rps-strings and "plays them against each other" by comparing the gestures ('r' == 'rock', 'p' == 'paper', 's' == 'scissors') corresponding in each of the two strings, e.g., rps1[0] vs. rps2[0] and so on… . It should return 1 if rps1 wins more often, 2 if rps2 wins more often and 0 if it's a draw. For +3 additional points, determine which rps string in the csv file defeats the largest number of other strings?! (Is it a human- or a machine-generated champion?)

**Problem 3: Your own data -- and visualizations**

[30 pts; 15 each for two of your own data-visualizations…]

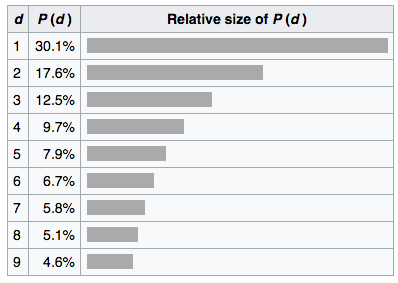
* This problem asks you to write all of your code in the **hw3pr3.py** file.
* In addition, you'll submit two screenshot files (their names are noted below…)
* This problem is an open-ended invitation to combine the previous two problems (your rps-string features and scoring) -- **or**, if you prefer, to visualize a data of your own choosing with the data-visualization toolset that matplotlib offers.
* To that end, first lookover the [matplotlib gallery](http://matplotlib.org/devdocs/gallery.html) to get a sense of some of the things matplotlib can do… (other internet snooping welcome! Animations are great fun, but also lots of work -- if you'd like to try those, it's totally up to you)
* Then, look over some of the [possible datasets](https://docs.google.com/document/d/1dr2_Byi4I6KI7CQUTiMjX0FXRo-M9k6kB2OESd7a2ck/edit) that might be of interest… (again, it can be anything - thesis data? Project data for another class? Something else?)
* Create at least two visualizations using your choice of data and matplotlib tools. Include the data (whether in a file or simply pasted into Python) and code to run your visualizations in your hw3pr3.py file. (There's a couple of default starter items there -- don't use those!)
* Include a brief explanation of your two visualizations in a triple-quoted string in hw3r3.py (there's a spot at the top of the file already)
* Take at least one screenshot of each of your two visualizations -- *please name them* ***datavis1****.png and* ***datavis2****.png*. Actually, any image format for the screenshot is completely ok, but calling all of your screenshots *datavis#* will help us find them.
* ***Uninspired?*** Try using the rps-string data from the class -- plotting your own choice of feature(s) from it and/or the [raw birthday-frequency data](https://drive.google.com/open?id=0BwPWh-3AmiLxN1ZadEpsVG4xZzQ) linked here (or, if you'd prefer, this [table of borthday ranks, NYT](http://www.nytimes.com/2006/12/19/business/20leonhardt-table.html)), and/or the [xkcd matplotlib gallery](http://matplotlib.org/xkcd/gallery.html) (!?!) If you're looking for specific ideas -- create your own [heatmap](http://matplotlib.org/examples/pylab_examples/colorbar_tick_labelling_demo.html) of rps features and another of birthday frequencies ([nice](https://www.labnol.org/internet/most-popular-birthday-months/21283/) / [hmm](http://blog.revolutionanalytics.com/2012/06/birthday-problem-ctd.html)). Plus, try different [stylesheets](http://matplotlib.org/api/style_api.html): they are what make matplotlib (almost) as sophisticated as one might want. :-)
* ***EC Options***: Certainly, this is wide open! Data visualization is a field unto itself; these tools are capable ones, but there are many others out there - if this is something of interest, feel free to explore more in this problem (let us know you have) or as a final project… . And, if visualization is a particular interest, the author of the matplotlib tutorial also has an elegant paper you might like, entitled [*Ten Simple Rules for Better Figures*](http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003833).

**Problem 4: Benford's Law**

[??? pts]

[Benford's law](https://en.wikipedia.org/wiki/Benford%27s_law), also called the first-digit law, is an observation about the frequency distribution of leading digits in many real-life sets of numerical data. The law states that in many naturally occurring collections of numbers, the leading significant digit (the leftmost digit) is likely to be small. For example, in sets which obey the law, the number 1 appears as the most significant digit about 30% of the time, while 9 appears as the most significant digit less than 5% of the time. By contrast, if the digits were distributed uniformly, they would each occur about 11.1% of the time.

It has been shown that this result applies to a wide variety of data sets, including electricity bills, street addresses, stock prices, house prices, population numbers, death rates, lengths of rivers, physical and mathematical constants, and processes described by power laws (which are very common in nature). It tends to be most accurate when values are distributed across multiple orders of magnitude. Below shows the percentages of each first digit according to Benford's Law.



For this problem you need to download the data set [city\_populations.csv](https://github.com/ScriptingBeyondCS/CS-35/tree/master/week_3), which gives the populations the world's ~27,000 most populous cities. (To download a file from github, click the file. Then, right-click the "Raw" button and select "Save Link As" to download.) Parse the data and use it to produce a matplotlib bar graph showing that the first digits of city populations follow Benford's Law.

\* Description and graph courtesy of Wikipedia

**Extra-credit: Showing off your results…**

[up to +8 pts extra-credit...]

* As with each week, you're invited to include both your source code and a short write-up of one of the week's problems within your GitHub repo(s). If you do, let us know (and provide a direct link). We look forward to checking it out!