Data Analysis in Python, MAT/PHY 230: Mid-term Project (40 pts)

The goal of this project is to complete our exploration of the statistics of the "Snakes and Ladders" game.

Read and follow all of these instructions carefully and precisely to ensure full credit. You will be given some class time to work on the project before the deadline. I need to receive the URL of your uploaded notebook in a ***secret gist*** *by midnight on Saturday, March 5th (right before Spring Break). If you have a documented accommodation for longer deadlines, your deadline will be Wednesday, March 9th. Use the numbered sub-sections below to organize your notebook using markdown cells.*

Feel free to show me your work-in-progress to get feedback. I expect you ***to present this in the form of a notebook*** *that shows me your data and explains what you are trying to do. An easy way to share a notebook like this is to post online as a secret gist and send me the URL. That way, I can read it directly online and get back to you more rapidly, and no one else will be able to see it without the URL.*

1) Definitions of different game editions

1.1) [2 pts] Write dictionary objects to define *two* different 10x10 boards using pictures you find on the internet (e.g. classic editions of the board game). Make a third board setup of your own by making four or more variations or recombinations of snake and ladder positions from the first two. Provide appropriate metadata for them, and save them to JSON format.

1.2) [2 pts] An updated protocol and tools can be found on github in the `board_compare.ipynb` notebook. According to the stated protocol, edit the `validate_setup` function to ensure that there is *at most* one snake or ladder per square and that no square appears twice across the two sequences (hint: consider how you would check this by hand). Show code that demonstrates that the three definitions are indeed valid and different from each other.

2) Creating the simulation data

2.1) [1 pt] Write code to determine how many snakes and ladders each board has, and present the data in the form of a Pandas table. (Think of the most efficient way to obtain this information in this situation.)

2.2) [2 pts] Run simulations of your three game boards using one regular 6-sided die. Save all the move record data to CSV files (three in total). This is the last thing that should happen in your first notebook.

3) Basic comparison of the editions

3.1) [4 pts] In a new notebook, load the CSV data. Explore the basic statistics of the behavior over many runs. Follow the same steps that we did in class for the 4x4 board. Present 2–4 plots that suitably summarize similarities or differences in their behavior, in addition to any data tables that you display.

The plots could be bar charts, line plots, scatterplots, or any other kind you think are appropriate. You may use functions from any of the libraries we have installed, especially Matplotlib and Pandas. Read about them in your book or online. Add markdown text as commentary about your findings.

4) Defining the "loopiness" of a game

4.1 [5 pts] Write a function to compute how many times the *same* snake or ladder was used in *one* game, based on a parameter to the function that selected the "start" of that snake or ladder. You will have to choose how to design this effectively, based on the code and workflow we have already developed. Think about the design and use case before writing the function.

4.2 [5 pts] Use the function to find characteristic statistics about number of times any snake or ladder was used in any run of the game (let's call that condition a *recurrence*). Hint: consider the Pandas `describe` method that we have previously used. Present a comparison of the statistics of recurrences between each of your three game setups. Let us now define the *loopiness* feature of a game setup to be the mean average number of recurrences.

5) Another metric for recurrent behavior

There are many possible ways to quantify the recurrent movement of a player in a board game such as this. Let's just consider one more metric.

Suppose there is a ladder at square 10 linking to 16 and a snake *starting after that endpoint* at square 20 linking to 8, *before the foot of the ladder*. Regardless of the location of any other snakes and ladders, it is possible that the player could go around this pair of snake and ladder more than once in a game. We can call this behavior *cycling*. A more complex cycle could involve additional snakes and ladders that may be present between squares 10 and 20. To keep things simple, we will say *2-cycling* specifically when a cycle happens between one snake and one ladder with no other snakes or ladders being used in between.

5.1) [4 pts] For this example ladder-snake pair `(L, S) = ((10, 16), (20, 8))` we therefore see that there is a requirement that `L[1] < S[0]` and `S[1] < L[0]` for this 2-cycle to be possible. No other jumps are possible if there is no starting point for any other snake or ladder between 16 and 20 or between 8 and 10. For each game board, write code to compute how many valid snake-ladder combinations can 2-cycle with no other jumps in between? By this definition, note that there is just one 2-cycle possible in the 4x4 game we first defined (8→14→15→6→8…).

5.2) Extra Credit [3 pts] Using your answer to (4.1), write code to detect all 2-cycles occurring during a game simulation. You can start exploring this by hand and then write a function to help you automate. If you find cycling, identify the associated player move data and display the sequences of moves in the notebook cell so that I can see the relevant parts of it myself.

5.3) Extra credit [4 pts] Write a python loop (or otherwise perform an operation on the dataframe of moves) to find the largest number of recurrences and cycles of *any* pairing of snake and ladder. Repeat these measurements for each of your three game board setups and describe your findings.

6) Game variations

6.1 [5 pts] Pick one game board setup. Now, pick any one snake or ladder whose "start" occurs between position 35 and 65. Vary the position of that start point by 5 in each direction, in steps of 1, and make a new game setup for each by writing a function to automate the process. However, note that the start of a snake or ladder cannot be placed on a square where there is already the start of a different snake or ladder, so your code must avoid that situation. *You may therefore have fewer than ten new game boards.*

6.2 [3 pts] For each game setup, compare their loopiness metrics and number of possible 2-cycles (as defined above) and present your findings with a table or graph and some explanation. If you did the extra credit above, include statistics about the number of 2-cycles actually found in the simulations for an additional EC point.

Overall presentation

[5 pts] You are expected to reuse the code we developed in the class, making any adjustments where needed. Credit will be awarded for clear variable names and using markdown cells to add commentary and explanation to the code or results data where you see fit. (Don't overdo it! Good code is pretty self-explanatory). Make sure you are using the latest version of the module from my github repository before starting.

I cannot give credit to code that does not run or does not make sense. Make sure you test all your code, and provide a clear citation (such as a URL in a markdown cell) to any new code that you borrow and adapt from other sources. If you're unsure of something, ask privately on gitter or by email about it while sharing your work in progress as a gist.

[2 pts] Upload all your investigations as **two Jupyter notebooks** with embedded graphs to a gist and send me its link.

Extra credit [2 pts] What, if any, correlation exists between our loopiness metrics and the average and standard deviations of game lengths for the three boards?

Start this project as soon as possible, so that you can be effective during the limited class time that I will allocate and have opportunity to ask questions outside of class.

You may ask your colleagues for general advice about understanding the meaning of the questions, and general programming advice *only*. Don't submit work that is *at all* identically written or coded to that of your colleagues. (See the syllabus for the rules and consequences of plagiarism.)

You will be penalized for late submissions pro rata to their lateness.