Stephen Mouch

ID: 2618503

May 17, 2023

PACE Python/Data Analysis

Programming-Oriented Capstone Project Report

The code functions very simply. First, the necessary imports are made for pandas, random, and datetime, then the csv files are read in and their dates converted to a usable format. After this the rules, i.e. cost per play and basic prizes (0-3 matches) are defined.

The simulate_game function is where the bulk of the magic happens. This function uses the hardcoded dates and user input in main() to run the simulation according to random or specified numbers. It utilizes a for loop to rerun the simulation for every date in the date range specified. In this for loop, it tabulates the additional cost of each run and, if reselect = false (which is only the case for user specified, fixed numbers), then it uses the numbers input by the user. If reselect = true, it generates a new set of numbers in the range 1 to 40 for every iteration. The function then uses the prepared, imported data in winning number datafile, df_numbers, to pull the numbers, and uses the len() operator on a logical and of the set of numbers and set of winning numbers. This gives the quantity of matching numbers. If the number of matches is less than four, the predefined prize array is accessed with matches as the index and it is added to the total won. If the matches are 4 or 5, it finds the prize via the imported and formatted data in the prizes datafile, df_prizes, adding it to the total one. In addition to this, in order to track this unusual occurrence, it also increments a counter variable for four or five matched numbers. When the iteration has run it's course, the roi is calculated and the totals spent and won, roi, and quantity of 4/5 number matches are returned to the main method.

The main method is very straightforward, taking user input to determine random or user specified numbers, with error handling built in for non-unique or out-of-bound numbers. The results that it prints are dependent on the user's initial choice: if user-specified numbers are used then the reselection option is not available as only those fixed numbers are being used throughout the simulation. If the user selected random, it outputs both the random, fixed results as well as the randomly reselected results.

While I ran this simulation many times, I found that it overwhelmingly resulted in negative ROIs.

Analysis:

Because the code I wrote according to the original specifications of the project didn't give me enough granularity to make good analysis of the data, I modified it slightly to force it to produce multiple simulations of each scenario and then graph them showing cumulative loss or gain. While practically, this is a bit nonsensical as one couldn't replay the same 8 years of lottery over and over again, in the context of data gathering it represents the overall trend in the data given the possibility in each scenario for the randomized numbers to perform differently.

1. Find the ROI for a user selected set of numbers

```
Scenario 1 - Fixed numbers (user-selected): [1, 3, 5, 7, 11]
Total money spent: $2757
Total money won: $424
Return on investment: -84.62%

Four or Five number matches: 0
```

2. Find the ROI for a randomly selected set of numbers

```
Scenario 1 - Fixed numbers (randomly generated): [13, 36, 25, 4, 28]
Total money spent: $2757
Total money won: $684
Return on investment: -75.19%

Four or Five number matches: 1
```

3. Find the ROI for randomly reselecting numbers every day.

```
Scenario 2 - Reselecting numbers every day
Total money spent: $2757
Total money won: $641
Return on investment: -76.75%
Four or Five number matches: 1
```

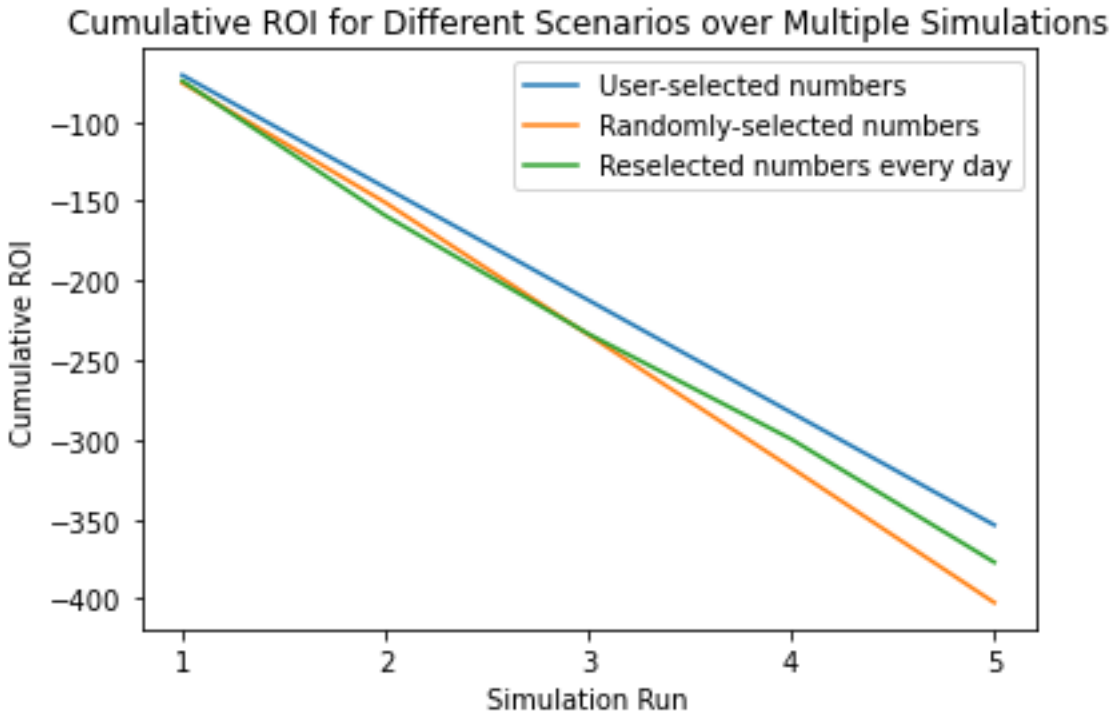
4. Note any scenario that resulted in a 4 or 5 number match.

All scenarios where that occurred have a counter variable printed.

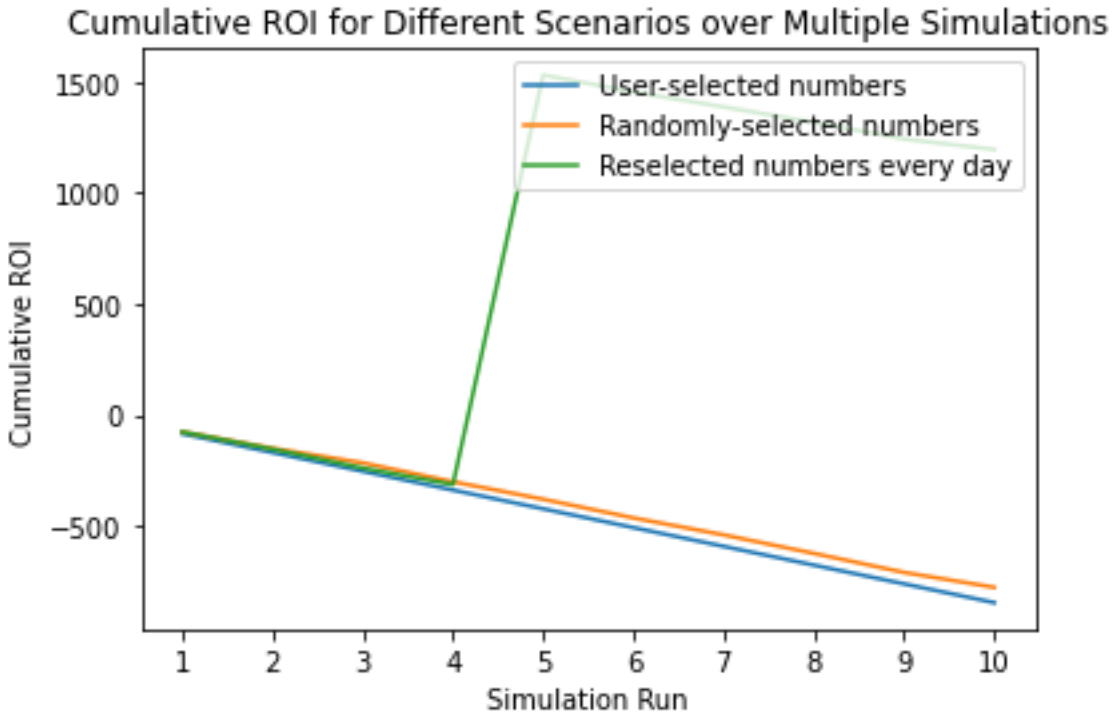
5. Based on your simulations, does any one scenario (user selected, randomly selected, or randomly reselected) influence the final ROI? Why might that be (or not)?

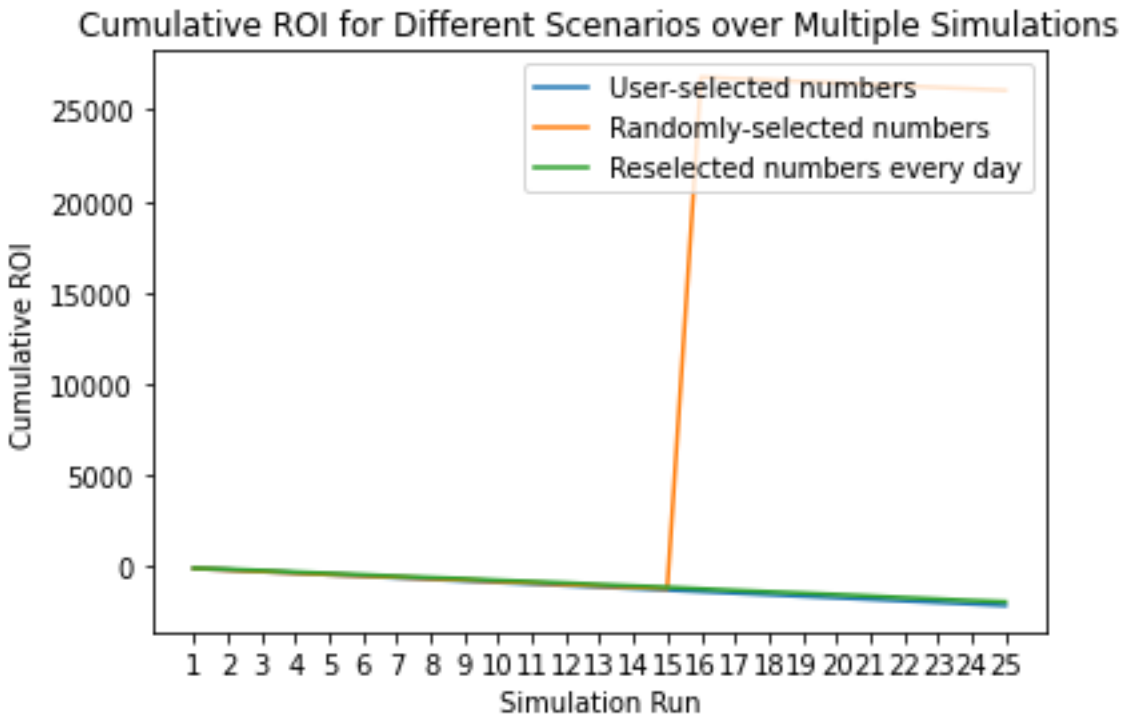
In all of the running of these simulations I didn't see any trends in the individual data, so I refactored to a couple different graph generating files. Doing so allowed me to examine cumulative ROI as well as the number of 4/5 matches, which ended up seeming to be a major determinant of a less-negative ROI. For all of these the user-sequence was the first 5 primes, less 2.

Most of the ROI graphs I tested at 5 cumulative runs were linearly decreasing in trend, like the following graph:



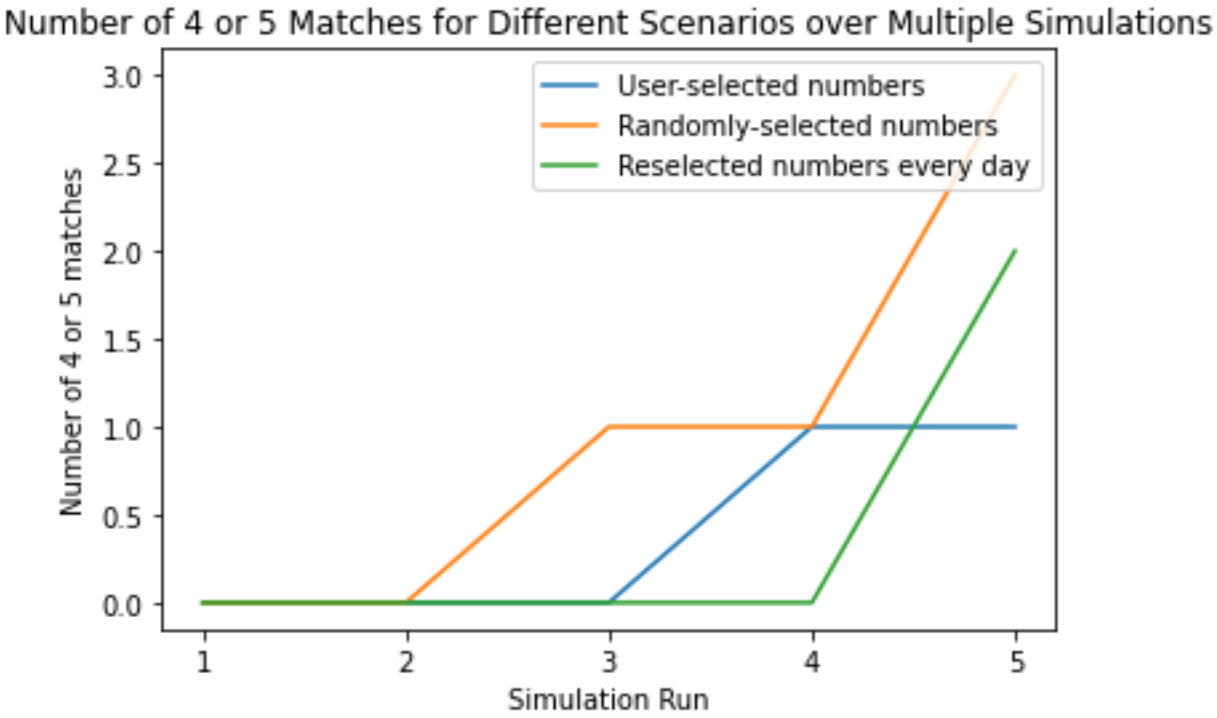
With the total number of simulations increasing to 10 and 25, there were incidences of two simulations, resulting in one each of the randomly-selected and reselected numbers did exceedingly well.

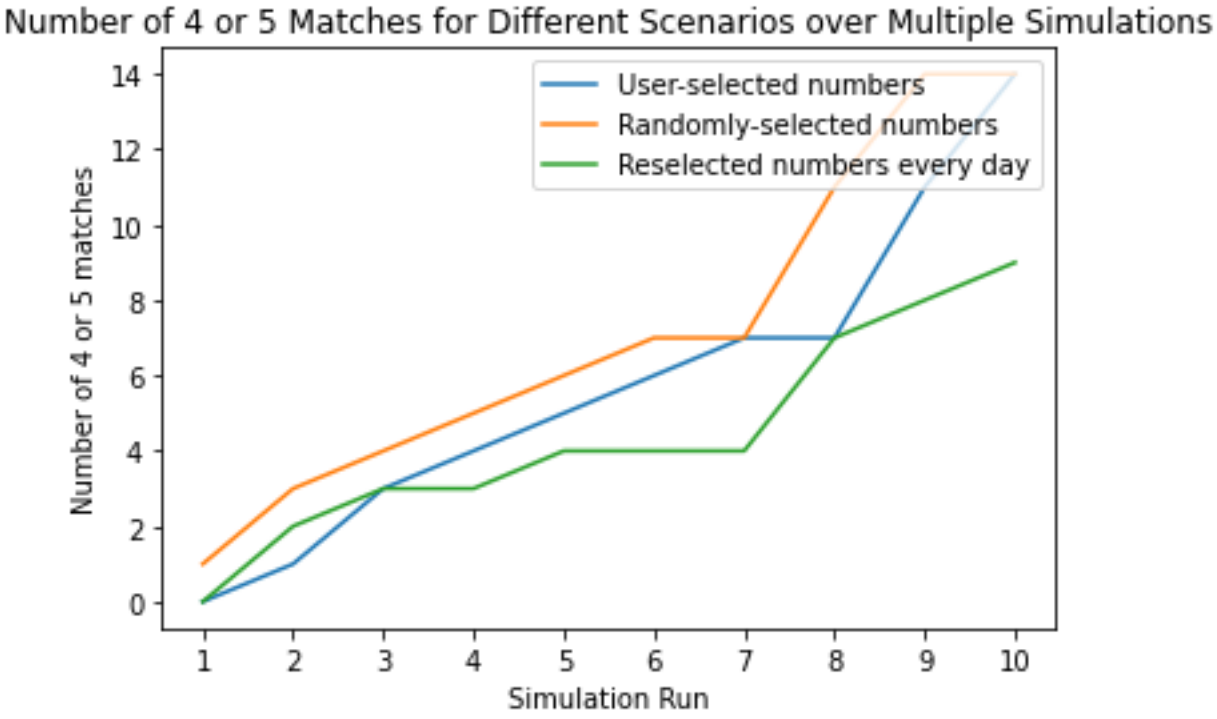


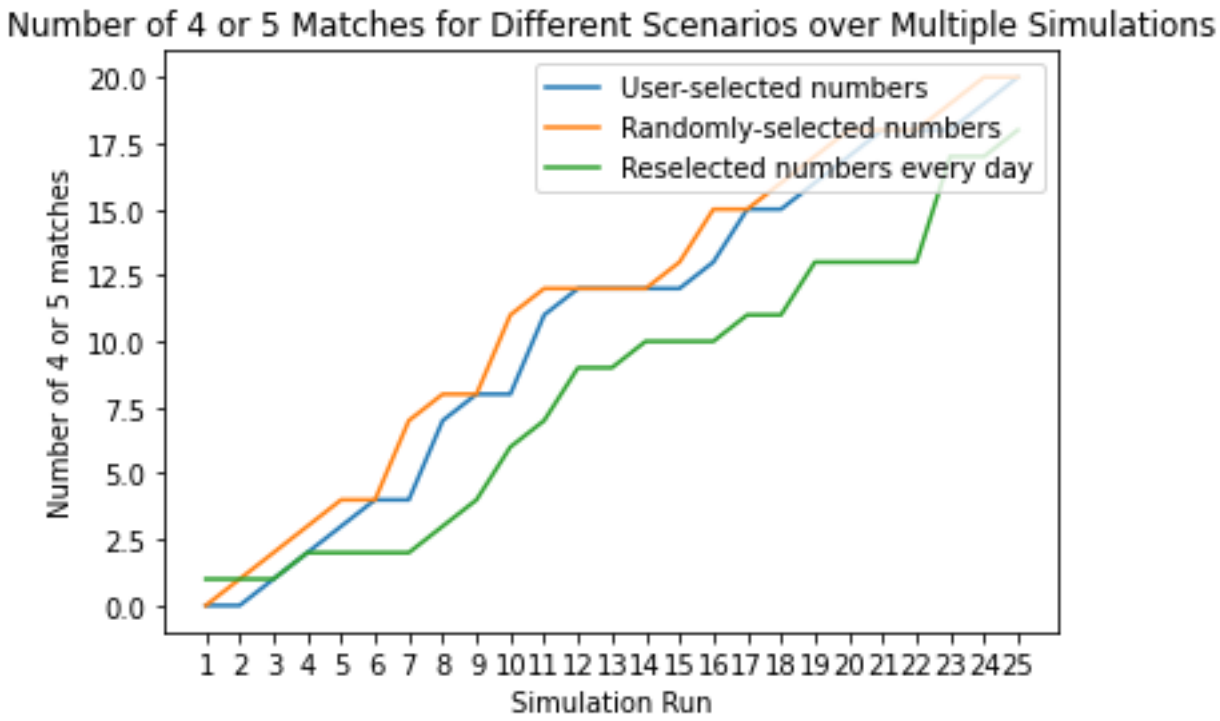


The general slope trending towards zero indicated to me that, given enough simulations, it could potentially break even. Beyond that there was no indication that any specific scenario outperformed the others in a meaningful way.

In the individual simulations I noticed that the ROI was higher if there were more 4/5 matches, which makes sense as it results in a far larger prize. As such I refactored to instead graph the cumulative 4/5 matches over multiple simulations.







As for playing the same numbers, it does seem that strategy, even if random, trends mildly stronger in winning larger prizes.