# **Objectives**

The objective of this assignment was to programmatically determine the correlation coefficient of shuffling cards in two different ways on both 52 and 104 card decks. Given a deck of 52 cards in the first run, the cards must be shuffled in a way such that the first card is always 1 and the last card is always 52. The same applies to the deck of 104 cards in the first run, only the last card is always 104 instead of 52. With 52 cards the second run, the shuffling is changed so that the first card is 27, the second is 1, and the third is 26, and so on with the same applying to the 104-card deck only the cards being 53, 1, 54, and so on. Each shuffle must be repeated 15 time on each deck, and the correlation coefficient calculated for each. Finally, the correlation coefficient of each shuffle must be plotted against the number of shuffles.

# **Program Design**

To accomplish the required functionality, six functions were implemented:

1. **shuffle1() –** This function is responsible for shuffling the decks of cards in the first run. It first calculates the half-length of the deck. It then iterates over a new empty array to length n, adding a card from the first half of the deck, then a card from the second until all the cards have been added.
2. **shuffle2() –** This function shuffles the decks of cards in the second run. It functions similarly to shuffle1(), except instead of taking the cards from the first half of the deck it takes the cards from the second half of the deck first.
3. **calculateR() –** This function is a Python implementation of the Pearson’s correlation coefficient.The function first calculates *sumi*, or the sum of the numbers one to *n*. Then it calculates *sumsq*, or the sum of the squares of the numbers one to *n*. The *sqsum* calculated in the next line is a square of *sumi.* The next line calculates *sum\_product,* or the sum of the products of the positions of each card in the shuffled deck and its value. It iterates over each card using *enumerate(deck)* to get both the index and the value of the card, then multiplies the index by the value to and sums each of these. Lastlty, *r* or the correlation coefficient is calculated by .
4. **performFirstRun() –** This function represents the first run, and is responsible for iteratively calling the shuffle1() and calculateR() functions 15 times and adding the shuffled deck with its r value to a list and shuffle number. In addition to this, it also plots the r values vs. shuffle number with some formatting to make the graph easy to interpret.
5. **performSecondRun() –** Functionally similar to performFirstRun(), except it calls shuffle2() instead of shuffle one and contains some different formatting for the plot.
6. **main() –** This is the driver code that calls performFirstRun() and performSecondRun() on both 52 and 104 cards, as well as some additional plot formatting and the final displaying for the plot itself.

Code screenshots:

**A computer screen shot of a deck

Description automatically generatedA screenshot of a computer program

Description automatically generatedA close-up of a computer screen

Description automatically generated**

# **Results**

The terminal output and graph produced by the program can be seen below:

A close up of a pattern

Description automatically generatedA close-up of a white background

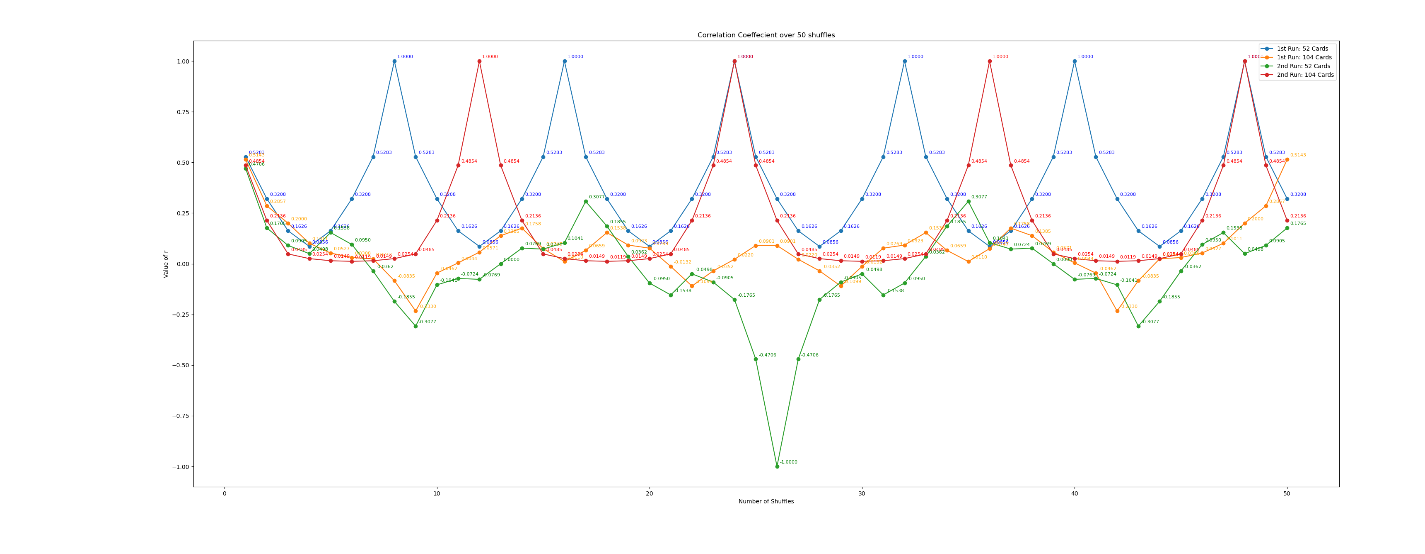
Description automatically generatedA close-up of a pattern

Description automatically generatedA close-up of a white background

Description automatically generated

A graph of different colored lines

Description automatically generated



(Above graph produced by the program after modification to run 50 shuffles)

# **Questions / Analysis**

1. In the first run with 52 cards, r is the lowest at the fourth and twelfth shuffles with a value of 0.0856 meaning that those are the two “most random” configurations of the cards. The second run with 52 cards sees a slightly lower r value of 0.0119. On the other hand, r is lowest at the ninth shuffle with a value of -0.2338 during the first run with 104 cards. Also at the ninth shuffle, the second run with 104 cards saw the lowest r value recorded, -0.3077.
2. In the first run with 52 cards, the cards return to their original order after eight shuffles. Also, in the second run with 104 cards, the cards return to their original order after twelve shuffles. These were the only two runs in which the cards returned to their original values. It is worth noting that when the cards returned to their original order, the correlation coefficient was 1.0. This means there was perfect positive correlation between the indices and the values of the cards.
3. When shuffling using the first method and 52 cards or the second method and 104 cards, 15 runs is enough for the cards to return to their original order. Also, after modifying the code slightly to shuffle 50 times instead of 15, the same results were seen, as well as cyclical peaks and valleys for the r values within the aforementioned runs. Also, at the 26th shuffle during the second run with 52 cards their order was completely reversed, yielding an r value of -1.
4. When using the first method and 104 cards or the second method and 52 cards, the cards never seem to reach their original order, even after modifying the code to run 50 shuffles.

# **Sources**

<https://www.w3schools.com/python/matplotlib_plotting.asp>

<https://rosettacode.org/wiki/Perfect_shuffle>

<https://stackoverflow.com/questions/43146319/annotate-a-plot-using-matplotlib-showing-values-in-the-plot>

<https://www.geeksforgeeks.org/matplotlib-pyplot-annotate-in-python/>

<https://www.socscistatistics.com/tests/pearson/default2.aspx>