Poker: The Unsolvable Game

## Application of Machine Learning Algorithms to 6-Player Texas Hold’em

Motivation 1:

As a child, my dream was always to grow up and play in the NHL. Then I realized that my hockey career peaked in grade 8. So for the longest time I did not know what to do with my life, until my math teacher told me: “If you like money, and if you like probability, you should become an actuary”. The truth is, I did like money and probability—but the main selling reason I was drawn to actuarial science was because I heard that actuaries were the best poker players in town. And I could live with being the best poker player in town.

For the last four years, I have pursued the road of actuarial science, but somewhere along the journey I had forgotten the reason for the pursuit altogether. Or perhaps I had not realized it until now. Either way, after finishing my previous work term, I decided I was far enough down the road to test what actuarial science has done for me in terms of poker, so I put everything I’ve learned to this day into this project. Hopefully by the end of the project, I will see whether there is potential to become the best poker player in town, how to do so, and most importantly, whether it is worth it or not for me to continue studying actuarial science.

Motivation 2:

This project has been coded entirely using Python. The reason for this is because during an interview with a company I want to intern with, I underperformed greatly due to my lack of knowledge with Python. Since this company mainly uses Python, I decided that was something I needed to become more familiar with if I wanted a shot at working there. The reason why I wish to work in this company is for 3 reasons.

Firstly, I have always heard that the top talent in not just my field, but in all fields, work here, and I want to see just how good they are. I want to learn from them, compete with them, and ultimately be considered amongst them. I believe this is analogous to my desire to play in the NHL; it is not enough for me to be a top liner in a less competitive league if I am never able to taste the show. The show is not about the money or the fame; it’s about the prestige of being ‘the’ show.

Secondly, I want to take what I’ve learned and bring it back into a field such as actuarial science. I feel that currently, the field of tech, and especially Fin-tech, there is an overload of talent. All the money is in tech, and the talent goes where the money is, and great products come from great talent, and money goes to great products. I don’t want to go down as another one of millions of money grabbing tech kids who are looking for the easiest way out. There is plenty of talent in fields outside of tech, I want to prove that talent can be bred through a great environment, and I want to use what I learn to create that environment. Whether Actuarial Science is the field I want to create this environment in is another question.

Lastly, I have a CS friend who is always showing up to the functions and talking about how much money he makes because he’s in CS. I just really want to have the ammunition to shut him up because he really pisses me off.

Phase 1: Prototype

The initial goal of this project was to create an AI capable of playing through the Pokerstars Cash Game UI. The AI would use machine-learning algorithms in order to determine the optimal play at each blind level. Analysis would be done on the bot’s play and calculations behind decision-making to see how the actions of the bot could be mimicked for optimal play (i.e. the existence of a solvable strategy).

# Action Plan:

* Take Hand History from PokerStars Cash Games
* Parse Hand History into readable Data Frames, each 1 hand, with the decision assigned a numeric value as the dependent variable (Fold = 0, Call/Raise = num big blinds put into the pot) and explanatory variables such as amount to call, number of callers, hand strength
* Run simple linear regression on decisions with respect to explanatory variates, weighing hands with high winnings higher as observations

# Timeline:

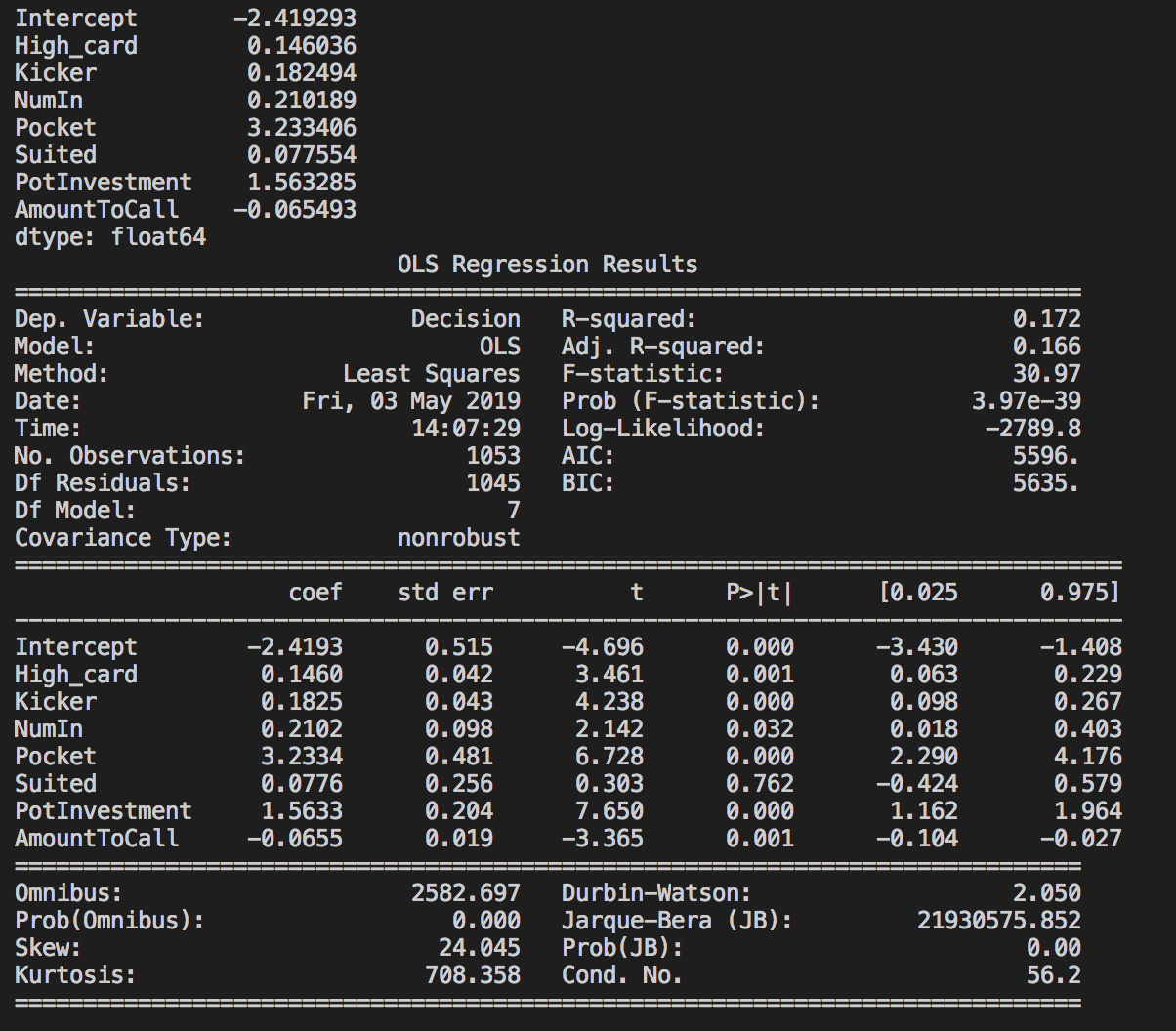
* Gathering hand history data (i.e. playing poker) – 1 week
* Building internal poker hand interpreter and parsing Hand History into readable Dataframe (Preflop) – 1 week
* Fixing hand interpreter to handle postflop – 2 weeks
* Difficulty was in handling of algorithms to handle extrapolation of parameters unique to postflop, such as number of outs and calculating the hand value. Such algorithms have yet to be perfected
* Parsing Hand History into readable Postflop dataframe – 1 week
* Building linear regression model and running model on preflop and postflop dataframes—1 day

# Conclusions:

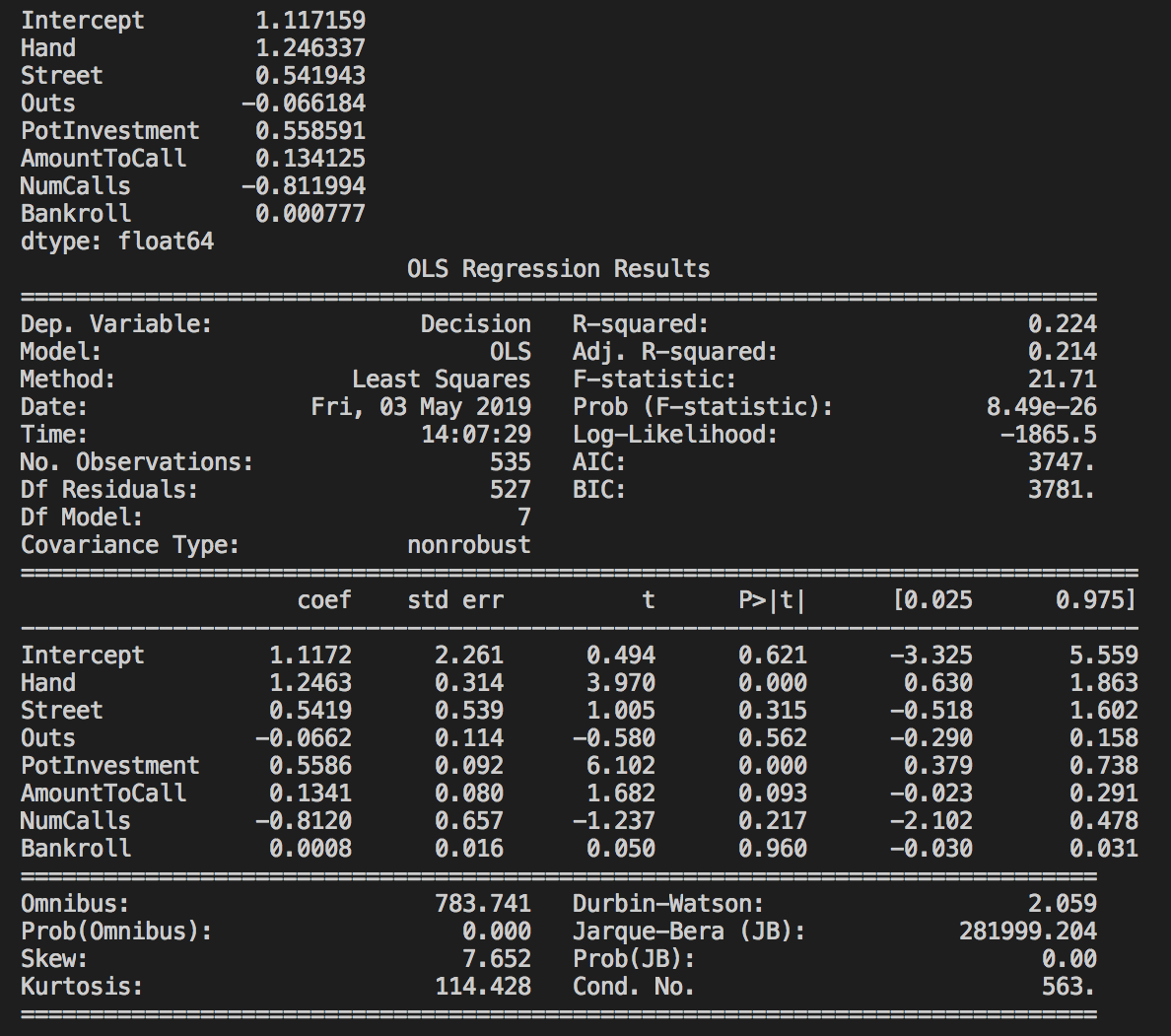
The coefficients determined by the regression appeared to be reasonable. However, the prototype method was not an effective method to derive a solution for the following reasons:

* Since regression was ran on Hand History of my play, the bot would learn to play like me, not like the ideal poker player
* Although decisions that led to large wins had more weight, those decisions were usually risky decisions that led to large losses as well, there was no punishment system for bad decisions, only reward for good decisions
* Running regression through the data did not provide a good fit, the parameters with high weight values

Preflop



Postflop



Phase 2: Genetic Algorithm

After deciding the initial strategy was inconclusive, I decided to use a variant of the Genetic Algorithm to find the optimal weights for gameplay

# Action Plan

* Transform hand parsing tool into full poker game simulator for genetic algorithm (reinforcement learning)
* Determine preflop and postflop coefficients randomly for 6 players
* Simulate 100 hands. The top two players after 100 hands in terms of stack size stay on the table, and 3 new players are breeded from the weights of the top 2. The last player will be random.
* After 100 generations, see optimal coefficients calculated from the algorithm

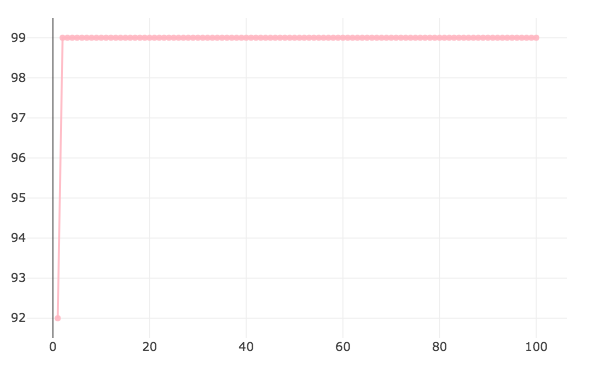
# Timeline

* Transform hand parsing tool into full poker game simulator—1 week
* Build in capability of generating random players with random weights, and capability for keeping top 2 players and breeding new players from top 2—3 days
* Build in capability of human controlled player(s) to play against bots, also for debugging purposes—2 days
* Run simulations and analysis—3 days

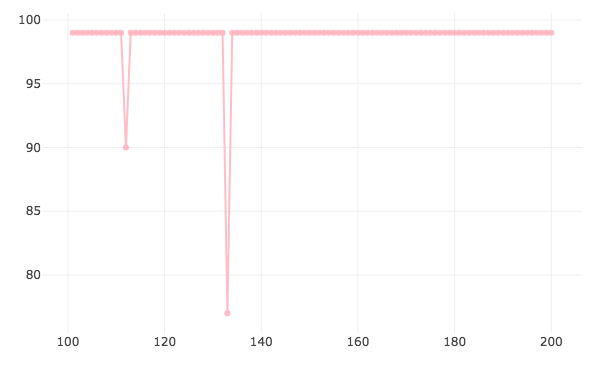
# Conclusions:

Game Length: Measures overall aggressiveness of players

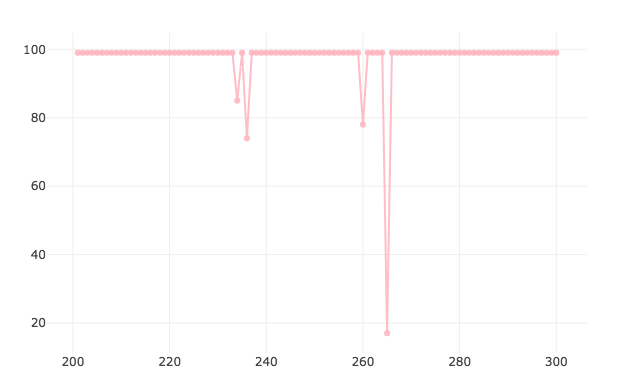
First 100 Generations



First 200 Generations

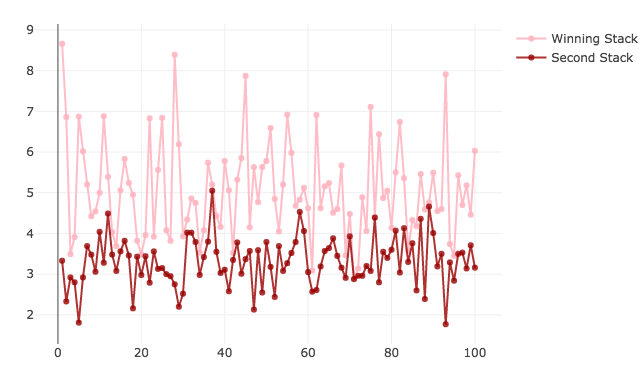


First 300 Generations

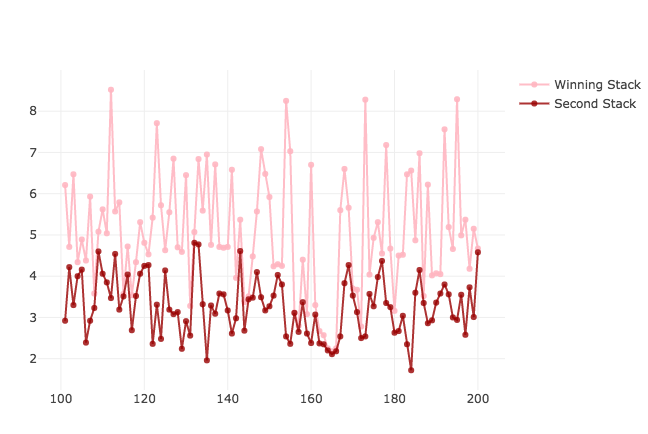


Stack sizes—measures performance of top players vs. Overall

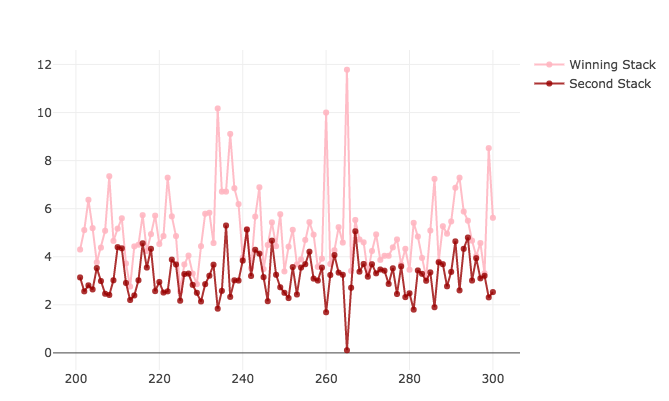
First 100 Generations

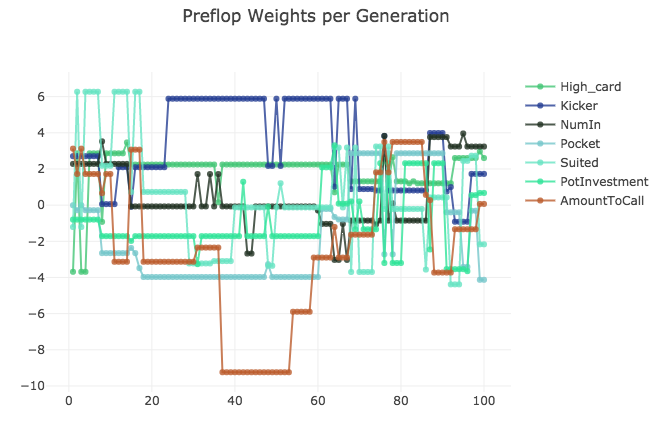


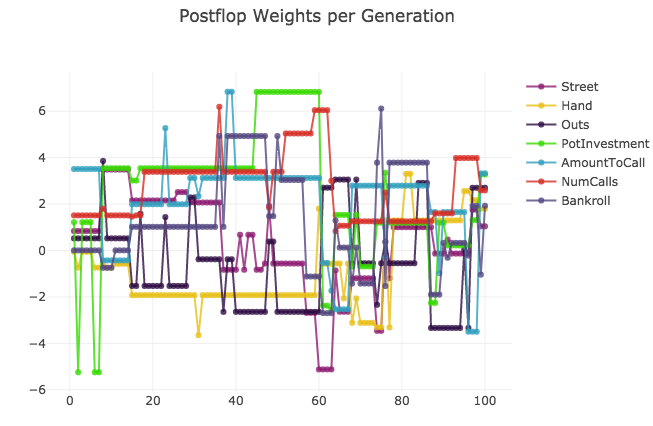
First 200 Generations



First 300 Generations

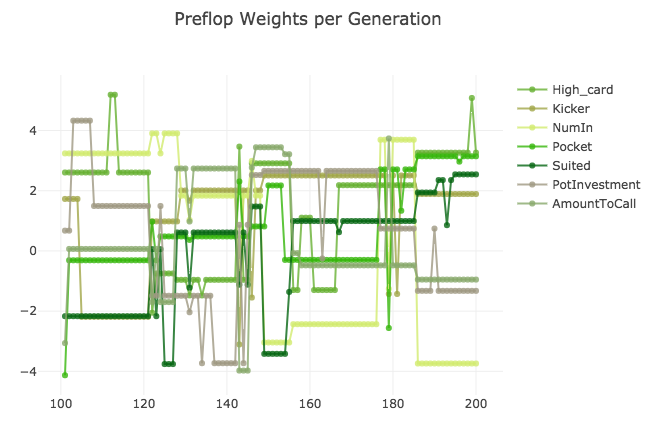


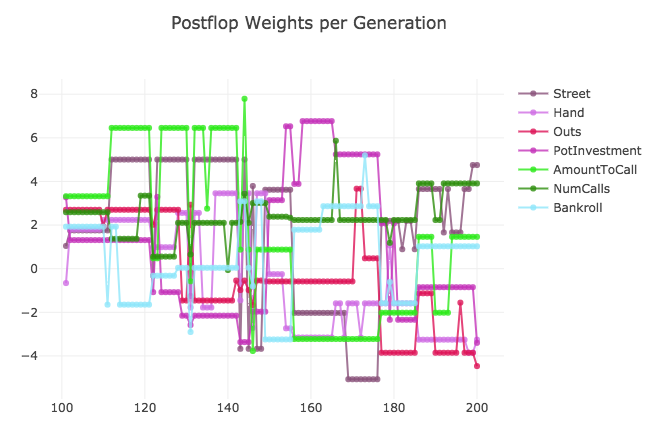




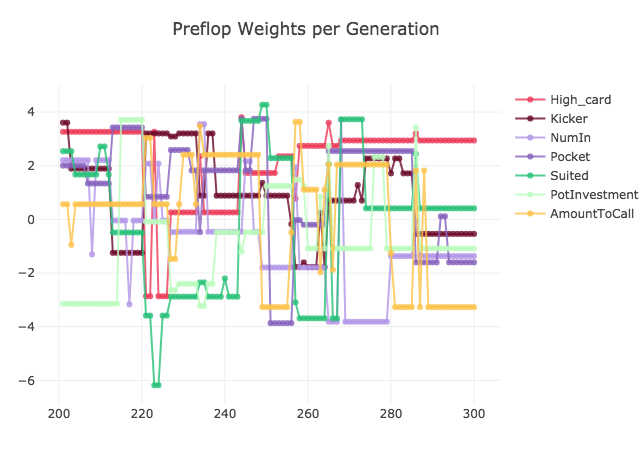
Hand seems to be a strong motivator to call/raise after 100 simulations

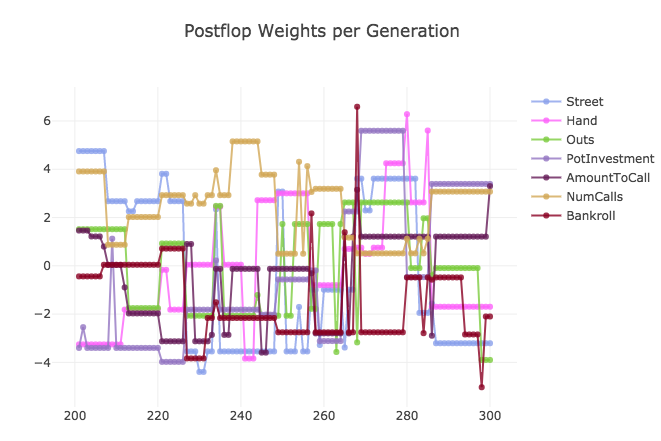
First 200 Simulations





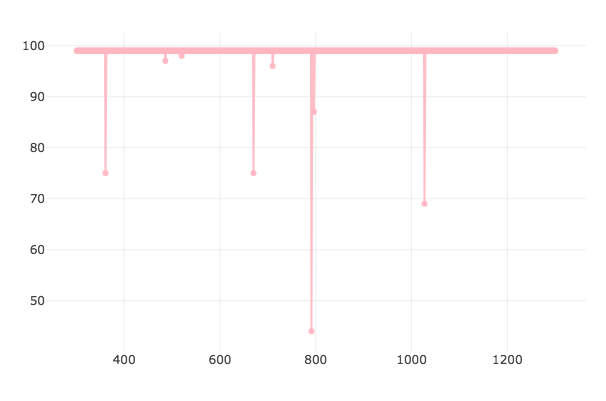
First 300 Simulations

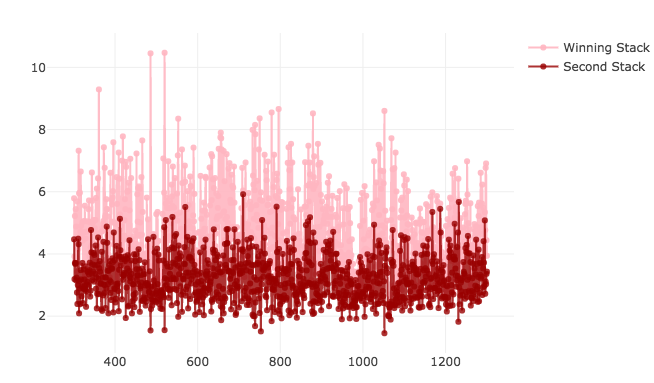


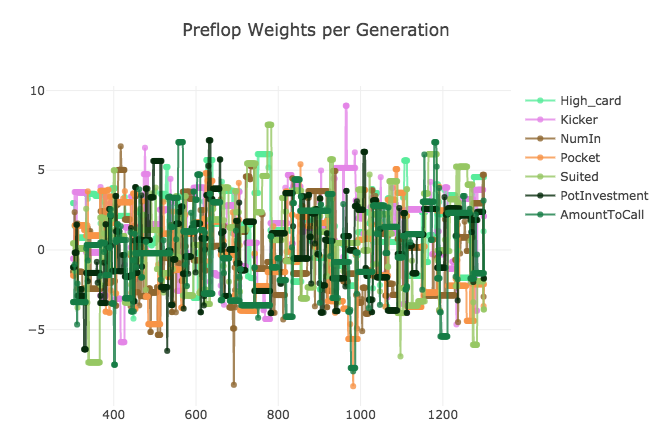


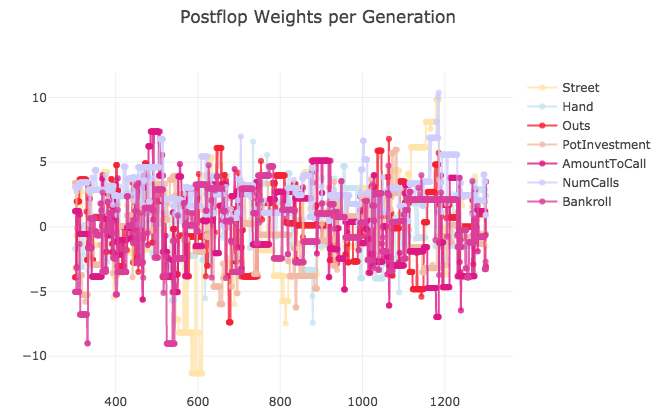
1000 Generations Later

No sign of convergence

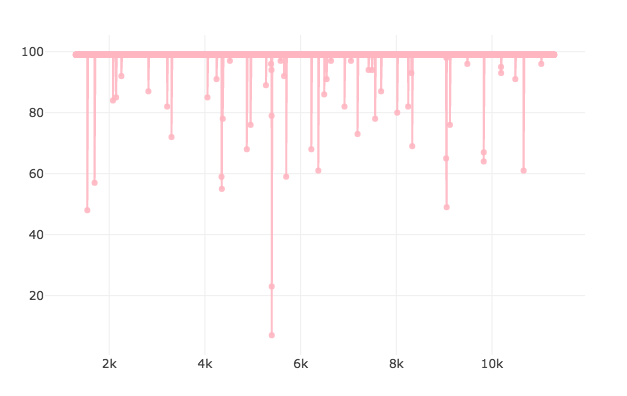


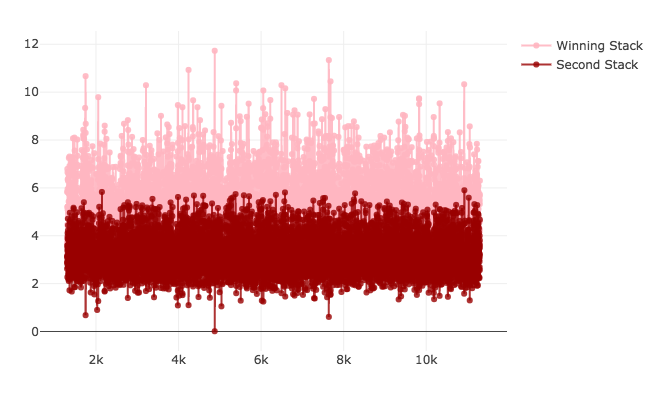


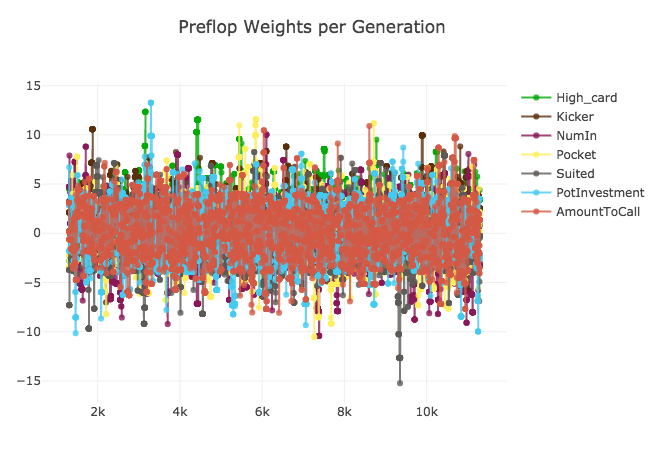


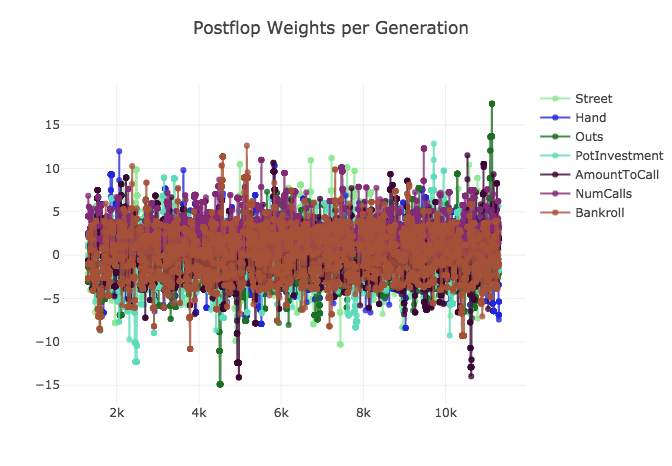


10000 Generations Later

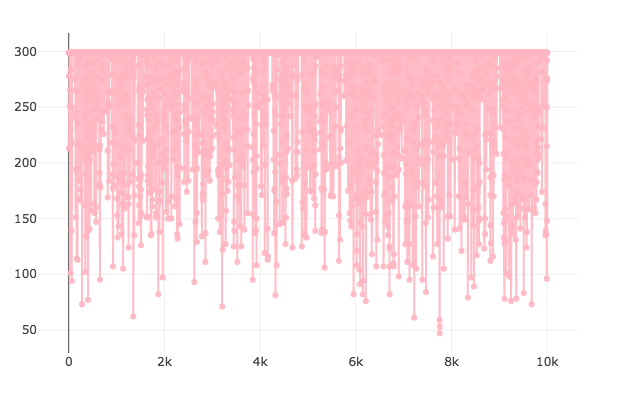


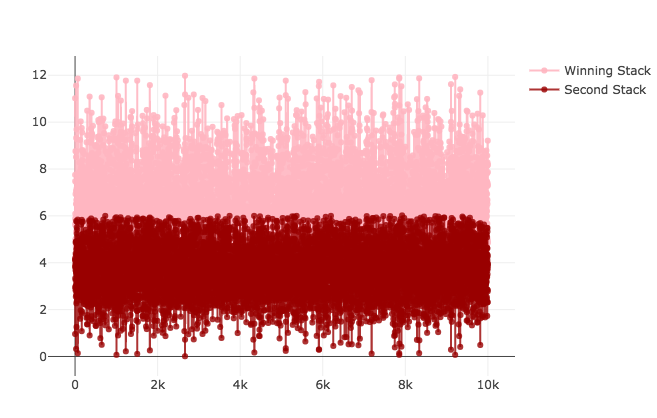


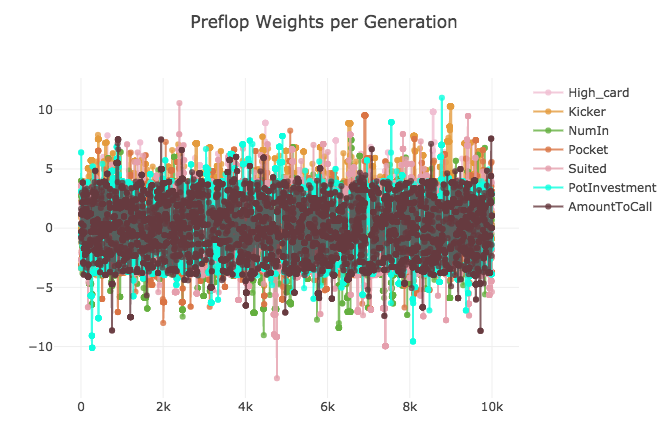


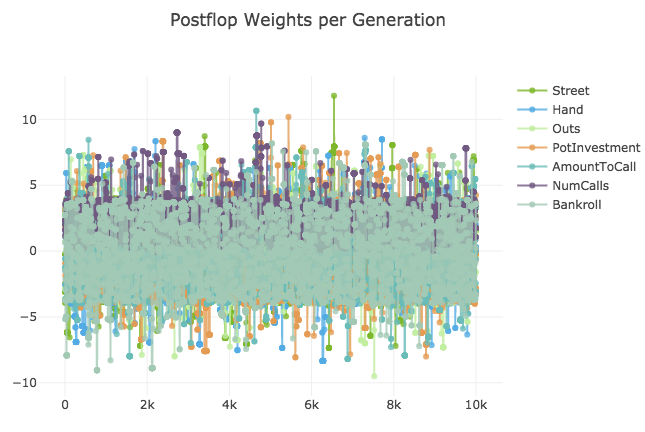


No signs of convergence: extend to 300 hands, 2 random players for faster convergence









No convergence

Phase 3: Genetic Algorithm with Neural Network

# Action Plan

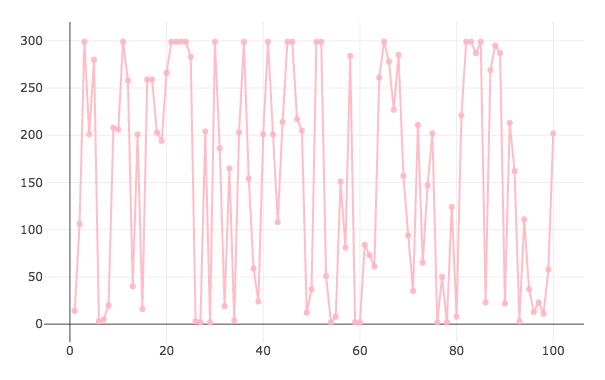
* Use a two layer Neural Network to make decisions instead of standard linear regression
* Second layer has 2 nodes: motivation is one node for your hand strength, and one node for opponent’s hand strength, later may experiment with more nodes
* Positives include better decision-making algorithm, there is a passive (check, fold), passive-aggressive (call, bet) or aggressive (bet, raise) options, better than a range between passive to aggressive (sometimes either fold or raise), also extra layer adds extra understanding up for interpretation of the model.
* Downside is cannot monitor weights on parameters as well, more black-box, not as intuitive

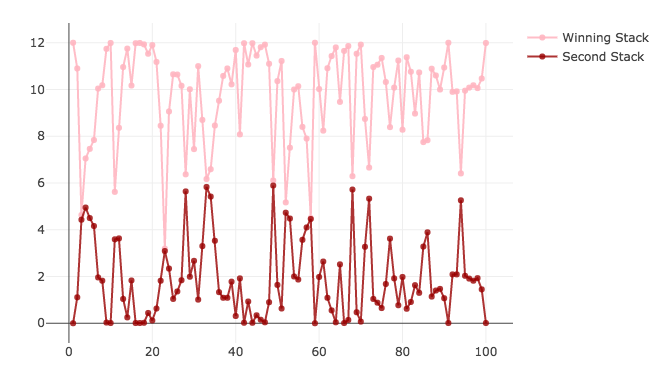
# Timeline

* Construction of Neural Network, functionality to interchange between Neural Network and Linear Regression decision-making – 2 days
* Functionality to load previous file to continue training from a prior generation – 2 days

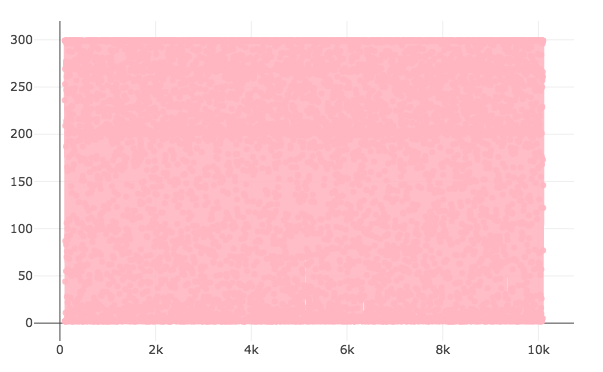
# Conclusions

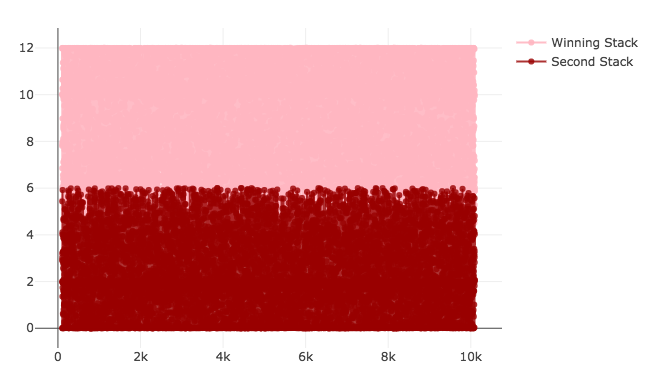
100 Generations





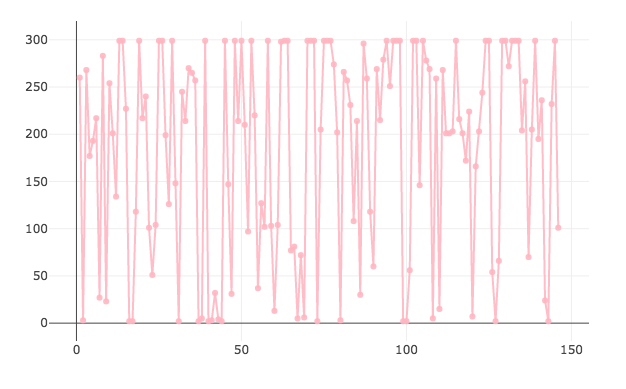
10000 Generations

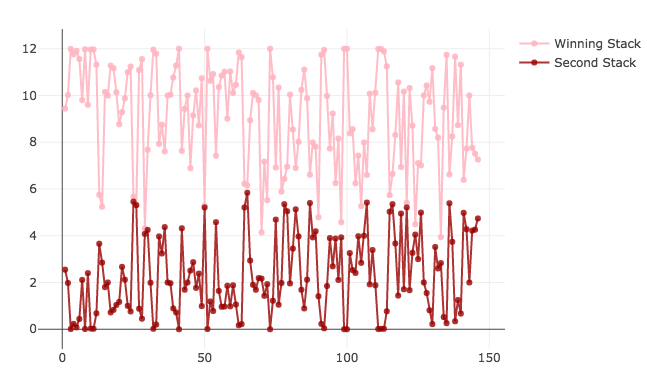




Clearly, it is difficult to determine whether convergence has occurred or not simply by looking at these two graphs. A new method is to track the wins of a certain player, and if that player has won (not been kicked out of the table) at least 10 times, then we can determine convergence has occurred.

Added new parameter called age (number of generations ago player was born), if age > 10 then player is a candidate for convergence.





Player1 in log146 managed to survive 10 rounds. However, after reviewing player1’s logs, noticed that this player has a very simple strategy: call preflop, bet postflop. Noticed that although this strategy is effective, it is very easy to catch on to. Therefore, maybe an additional parameter that accounts for opponent historical aggressiveness would be very effective here. Also, perhaps strategy is only effective because it is easy to stay in the game 10 times against randomly generated opponents with random behaviour patterns.

Convergence occurred after 146 Generations. Let’s try extending convergence age requirement to 20.

After a successful run without bugs, convergence occurred after 2454 generations. Because the number of generations is so high, information gathered from charts is minimal. However, player seems to again play very aggressively when taking a look at the player logs.

Since we are close to having a viable solution, want to make sure infrastructure for playability/compatibility with the PokerStars App is built. So will take a short break from strategizing and build the computer vision component of the AI.

Automated Playability:

# Action Plan:

Use computer vision techniques on Pokerstars app to parse gameplay information for decision-making, so bot gameplay can be fully independent. Ideally would like for the bot to be compatible with all window sizes and resolutions, but for that would need intense object detection models that are either too complex or just did not work.

# Timeline:

* Researching different computer vision techniques for automated gameplay (1 day)
* **Technique #1: Haar Cascades (3 weeks)**

The Haar Cascade technique uses the Python OpenCV library and trains a model to identify the pixel-pattern characteristics of a positive vs. a negative of a certain object, and imposes the cascade pattern over various areas in the picture to identify potential positives in high-matching areas.

I tried to create a cascade for a player on PokerStars (a circle with a rectangle beside it, with text in the rectangle), but since the shape was too basic and had no unique pixel patterns compared to the background, this technique was ultimately ineffective no matter how the data was generated

* **Technique #2: Tensorflow (1 week, and continued)**

Similar to the Haar Cascade, except uses Google’s Tensorflow GPU model to train based on self-labeled images

Installing Tensorflow properly took a while, and labelling the 300+ train and test images for all objects is also quite time consuming, so in the meantime decided to try an alternate technique

* **Technique #3: Hard-Coding (1 day—CURRENTLY HERE)**

The initial window location for the Pokerstars Game will always be at the bottom right of the screen, so hard coding the player and card-locations was definitely a possibility. Coded the AIPlayer.py to read information based on pixel colors at certain locations and converting certain areas into text using the Pytesseract OCR library

Currently experiencing issues with the OCR library, not reading in player names and stack sizes as expected, perhaps giving it a smaller picture to read will help