Blackjack Simulation - ISYE 6644 Project

Author: Bhargav Narasimha Shandilya

(Solo Project)

Abstract:

Although it has several variants, the classic version of Blackjack, which involves a 3 to 2 payoff for drawing an ace and a face card, is widely regarded as the most authentic version of the game. The strategy to come as close as possible to winning (popularly known as 'basic', 'optimal', or 'perfect' strategy) without card counting, has been fairly well-established. In this analysis, an attempt is made to replicate this strategy to its full extent while accounting for as many corner cases as possible. Broadly, based on the number of runs, the simulation shows that casinos will invariably make money over time. The expected amount of loss can be minimized by the player by following certain strategies outlined in [1]. This project also analyzes and compares a variant of classic Blackjack known as 6 to 5 Blackjack. The difference that a 6 to 5 payout makes is quite significant, and it drastically reduces the number of runs required for the expected returns to converge to a negative value.

Background and Problem Description:

Game objective: To get a cumulative score that is higher than the dealer. The game can theoretically be played with just the dealer and a single player. If the score (sum of card values) crosses 21, the player loses the round.

To begin, let us first establish the basic rules of Blackjack:

- Blackjack uses 4 to 8 packs of standard 52 card decks. For the purpose of this experiment, we assume that the number of decks is 6 (this turns out to be a popular choice amongst casinos).

- Card values :
  J, K, Q = 10
  Cards from 2 to 10 = Number on the card
  Ace = 1 or 11 (depending on what's best for that hand)
  Suites do not matter in Blackjack.

- The dealer gives each player two cards before drawing two cards, one face up and another face down.

- The player has two information points to try and beat the dealer: the value of their cards and the dealer's face-up card.

- Bets must be placed before the round begins. After the dealer deals the cards, players can double their bet by choice.

- The player's options:

  Stand: The player is satisfied with their cards and is willing to try and beat the dealer with the cards they have.

  Double Down: Double the value of the bet and draw a card.

  Hit: Ask for another card.

  Split: If the player has two identical cards (such as 2 aces), they can ask the dealer to treat each of the cards as a separate hand. They will then be given additional cards for each extra hand.

  These are the primary options across all casinos. Some casinos have additional options such as surrendering, buying insurance, etc. We will not be dealing with these additional options in this project.

- If the player or the dealer manages to score 21 points with 2 cards, it is considered a blackjack. Players are generally paid 1.5 times their bet amount if they score a blackjack. If the dealer scores a blackjack, the player automatically loses the round and their bet money.
  If no blackjacks are scored, rounds continue until all players decide to stay, the dealer goes over 21, or all the players go over 21. If neither the player nor the dealer goes over 21, the one with the higher score wins the round.

All of these scenarios are modeled in the attached python script.

The optimum strategy for blackjack was formally documented in the Journal of American Statistical Research [2] all the way back in 1956. The following table has been popularized as the best way to apply the strategy.

H - Hit
DH - Double if possible; if not, hit
S - Stay
P - Split

Card Number	2	3	4	5	6	7	8	9	10	Ace
4 to 8	H	H	H	H	H	H	H	H	H	H
9	H	DH	DH	DH	DH	DH	H	H	H	H
10	DH	DH	DH	DH	DH	DH	DH	DH	H	H
11	DH	DH	DH	DH	DH	DH	DH	DH	DH	H
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
17	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S
21	S	S	S	S	S	S	S	S	S	S
Ace - 2	H	H	H	DH	DH	H	H	H	H	H
Ace - 3	H	H	H	DH	DH	H	H	H	H	H
Ace - 4	H	H	DH	DH	DH	H	H	H	H	H
Ace - 5	H	H	DH	DH	DH	H	H	H	H	H
Ace - 6	H	H	DH	DH	DH	H	H	H	H	H
Ace - 7	S	H	DH	DH	DH	S	S	H	H	H
Ace - 8	S	S	S	S	S	S	S	S	S	S
Ace - 9	S	S	S	S	S	S	S	S	S	S
Ace - 10	S	S	S	S	S	S	S	S	S	S
Pair of 2's	H	H	P	P	P	P	H	H	H	H
Pair of 3's	H	H	P	P	P	P	H	H	H	H
Pair of 4's	H	H	H	H	H	H	H	H	H	H
Pair of 5's	DH	DH	DH	DH	DH	DH	DH	DH	H	H
Pair of 6's	H	P	P	P	P	H	H	H	H	H
Pair of 7's	P	P	P	P	P	P	H	H	H	H
Pair of 8's	P	P	P	P	P	P	P	P	P	P
Pair of 9's	P	P	P	P	S	P	P	P	S	S
Pair of 10's	S	S	S	S	S	S	S	S	S	S
Pair of A's	P	P	P	P	P	P	P	P	P	P

Playing Blackjack with basic strategy is expected to minimize the expected loss percentage to 50.5% to 51%. Several advanced strategies continue to be in development. As shown in [3], if the dealer has a 7 against a player standing on a double 9 (known as a serendipitous hand), the expected value for a unit bet (for the player) can be estimated to be:
E = (Wins - Losses) / (Wins + Losses + Non-Results).

In [4], we find that the 'house edge' or the advantage that the casino has over the player can be computed in various scenarios. Generally, it is given by the formula: HA = {(R - T) - 2T} / R, where HA is the house advantage, R is the remaining cards in the deck after the cards have been dealt, and T is the number of tens remaining in the deck. [4] also contains detailed computations for a wide range of scenarios that can potentially occur in Blackjack.

Main Findings:

Several references have been used to build the functions in the attached Python notebook. [5] has a compact function with some minor faults, but which covers all the base cases. [6] is also an interesting implementation of the problem. [5] has served as the primary inspiration for the structure of the Notebook.

All the functions along with the corresponding results for certain scenarios have been attached below. The notebook is self-explanatory in most cases. Using a good approximation of the optimal strategy, the code explores:

- Basic statistics such as mean and variance for 100,000 runs of Blackjack
- The number of zero transactions, i.e. no one wins anything in the round
- The number of blackjacks, positive results, and negative results overall
- The net amount gained or lost by the player
- The player's loss or gain percentage
- A 95% Confidence interval for the sample mean

Since 'n' is a very large number, it is perhaps safe to assume that the sample variance is very close to the real value of the parameter. Hence we use a standard normal distribution to obtain the confidence interval for the mean, assuming that the variance is known.

The cases explored are:
- Above-mentioned stats for 3 to 2 winnings for a Blackjack draw
- Above-mentioned stats for 6 to 5 winnings for a Blackjack draw
- Casinos require dealers to stop drawing cards when they reach a cumulative total of 17 or higher. Why is this important? What happens if this bar is raised to 19?

Kernel density plots are used instead of histograms to visualize the distribution of winnings, blackjacks, busts, and losses.

Blackjack Simulation - Project Notebook

March 13, 2021

1 Blackjack Simulation

In this notebook, we will attempt to perform Monte Carlo simulations of a player vs. dealer Blackjack contest. We assume that the casino follows the classic rules of Blackjack. We also explore the possibility of the 6 to 5 payoff instead of the tradional 3 to 2. This is known to result in a worse outcome for the player, but to what extent? All this and more in the blocks that follow!

[357]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
```

A guide for decision-making in blackjack. This is based on the strategy table provided by : https://www.blackjackapprenticeship.com/how-to-play-blackjack/.

[358]:
```
def PlayerGuide(player_cardset, dealers_pick, surr_allowed=False):

    num_aces = 0
    hard = 0

    for card in player_cardset:
        if card == 1:
            num_aces += 1

    num_player_cards = len(player_cardset)

    for val in player_cardset:
        hard += val
    # If the total hard count is greater than 11, it is best to not consider␣
↪the ace as an '11' card

    if hard <= 11:
        # The value of the ace is 11 in this case
        soft = hard + 11*num_aces - num_aces
        # To handle the case of two aces
        if soft > 21:
            soft = (hard-1) + 11
```

```
    elif hard > 11:
        soft = hard
    # Common conditions based only on the total hard and soft counts

    if hard >= 17:
        return 'Stand'

    if num_player_cards == 2 and soft == 21:
        return 'Blackjack'

    if soft >= 19:
        return 'Stand'

    # When the player has chosen to stand or not yet drawn a card

    if num_player_cards == 2:

        if player_cardset[0] == player_cardset[1]:
            card = player_cardset[0]

            split_conditions = (card == 1) or (card == 8 and dealers_pick != 1)␣
↪or (card in [2, 3, 7] and dealers_pick not in [1, 8, 9, 10]) or (
                card == 9 and dealers_pick not in [1, 7, 10]) or (card == 6 and␣
↪dealers_pick not in [1, 7, 8, 9, 10]) or (card == 4 and dealers_pick in [5,␣
↪6])
            if split_conditions:
                return 'Split'

            if card == 9 and dealers_pick in [1, 7, 10]:
                return 'Stand'

        if hard == soft:

            if(surr_allowed == False):
                double_conditions = (hard == 11 and dealers_pick != 1) or (hard␣
↪== 10 and dealers_pick not in [
                    1, 10]) or (hard == 9 and dealers_pick not in ([1, 2, 7, 8,␣
↪9, 10]))
                if double_conditions:
                    return 'Double'

                hit_conditions = (hard == 12 and dealers_pick in [1, 2, 3]) or (
                    hard <= 11) or (dealers_pick in [1, 7, 8, 9, 10] and hard <␣
↪17)
                if(hit_conditions):
                    return 'Hit'
```

```
            return 'Stand'

        if hard != soft:

            double_conditions2 = (soft == 17 and dealers_pick in [3, 4, 5, 6])␣
↪or (soft in [
                15, 16] and dealers_pick in [4, 5, 6]) or (soft in [13, 14] and␣
↪dealers_pick in [5, 6])

            if double_conditions2:
                return 'Double'

            if soft == 18:

                if dealers_pick not in ([1, 2, 7, 8, 9, 10]):
                    return 'Double'

                if dealers_pick in [2, 7, 8]:
                    return 'Stand'

            return 'Hit'

    # After a player has asked for a card

    if num_player_cards > 2:

        if soft <= 17 or (soft == 18 and dealers_pick in [1, 9, 10]):
            return 'Stand'

        if hard == soft:
            if hard <= 11 or (hard == 12 and dealers_pick <= 3) or␣
↪(dealers_pick in [1, 7, 8, 9, 10]):
                return 'Hit'
            return 'Stand'
```

Simulating a single blackjack round. Code references:

https://towardsdatascience.com/python-blackjack-simulator-61b591ffb971

https://github.com/molron94/Blackjack-Sim

Note: These are references only. The setting and the idea of what a single round should constitute is inspired from the links above. The rules and payouts remain the same for each round.

[378]:
```
def blackjack_round(bet_amount=10, player_cardset='no_initial_condition',␣
↪dealer_stay_num = 17, single_dealer_card='no_initial_condition',␣
↪payback_type = "3to2", deck = [1,2,3,4,5,6,7,8,9,10,10,10,10]*24):
```

```python
def end_round():
    card_list=[1,2,3,4,5,6,7,8,9,10,10,10,10]*4
    num_decks = 6
    deck = card_list*num_decks
    return deck

if (deck == [1,2,3,4,5,6,7,8,9,10,10,10,10]*24):
    card_list=[1,2,3,4,5,6,7,8,9,10,10,10,10]*4
    num_decks = 6
    deck = card_list*num_decks

stand = False


if player_cardset == 'no_initial_condition':
    ch1 = np.random.choice(deck, replace=False)
    deck.remove(ch1)
    ch2 = np.random.choice(deck, replace=False)
    deck.remove(ch2)
    player_cardset=[ch1, ch2]


if single_dealer_card == 'no_initial_condition':
    dealer_cardset = []
    ch = np.random.choice(deck, replace=False)
    single_dealer_card = ch
    deck.remove(ch)

    dealer_cardset.append(single_dealer_card)


while len(player_cardset) < 2:

    ch = np.random.choice(deck, replace=False)
    player_cardset.append(ch)
    deck.remove(ch)


# Check if a blackjack has been scored
if PlayerGuide(player_cardset, single_dealer_card) == 'Blackjack':

        # Has the casino also scored a blackjack?
        dealer_cardset.append(np.random.choice(deck, replace=False))
        if 1 in (dealer_cardset):
            if sum(dealer_cardset)==11:
                deck = end_round()
```

```
                    return 0
            else :
                if(payback_type == "3to2"):
                    deck = end_round()
                    return bet_amount*1.5

                elif(payback_type == "6to5"):
                    deck = end_round()
                    return bet_amount*6/5

    if PlayerGuide(player_cardset, single_dealer_card) == 'Stand':
            stand = True


    if stand == False:


        if PlayerGuide(player_cardset, single_dealer_card) == 'Hit':
            ch = np.random.choice(deck, replace=False)
            player_cardset.append(ch)
            deck.remove(ch)

            if sum(player_cardset)>21:
                deck = end_round()
                return 0 - bet_amount


        if PlayerGuide(player_cardset, single_dealer_card)== 'Split':

            new_turn = blackjack_round(bet_amount, [player_cardset[0]], single_dealer_card, deck = deck)
            new_turn += blackjack_round(bet_amount, [player_cardset[1]], single_dealer_card, deck = deck)

            return new_turn

        if PlayerGuide(player_cardset, single_dealer_card)=='Double':

            ch = np.random.choice(deck, replace=False)
            player_cardset.append(ch)
            deck.remove(ch)
            bet_amount= bet_amount * 2
            if sum(player_cardset)>21:
                deck = end_round()
                return 0- bet_amount
```

```
    while True:

        ch = np.random.choice(deck, replace=False)
        dealer_cardset.append(ch)
        deck.remove(ch)

        # Compute the dealer's score and the player's score - represented by␣
↪d_score and p_score
        d_score= sum(dealer_cardset)
        p_score=sum(player_cardset)
        if p_score<=11 and 1 in player_cardset:
            p_score+=10

        dealer_soft_score= d_score
        if d_score<=11 and 1 in dealer_cardset:
            dealer_soft_score+=10

        # Dealer scores a blackjack!
        if len(dealer_cardset)==2 and dealer_soft_score==21:
            return 0-bet_amount

        # Conditions to win or lose money
        # We'll assume that this is a traditional casino where the dealer␣
↪cannot hit when on a soft 17. This can be changed in the function parameters.
        if dealer_soft_score>=dealer_stay_num:

            if dealer_soft_score>21:
                return bet_amount

            if p_score==dealer_soft_score:
                return 0

            if p_score>dealer_soft_score:
                return bet_amount

            if p_score<dealer_soft_score:
                return 0 - bet_amount
```

[379]:
```
import pandas as pd

def compute_stats(sim):

    condition = sim[0]
```

```
    num_runs = sim[2]
    k = sim[1]
    bet = sim[3]


    ax = sns.kdeplot(k)
    ax.set(xlabel='Distribution of Wins and Losses', ylabel='Frequency')


    zero_count = 0
    positive_count = 0
    blackjacks = 0
    negative_count = 0

    for ele in k:
        if ele == 0:
            zero_count += 1
        elif ele > 0:
            positive_count += 1
        elif ele < 0:
            negative_count += 1

        if ele > bet and ele < bet*2:
            blackjacks += 1

    results = {'Number of Zero Transactions' : [zero_count],
               "Number of Positive Results (Including Blackjacks)" :␣
↪[positive_count],
               "Number of Negative Results" : [negative_count],
               "Number of Blackjacks" : [blackjacks]
              }

    print("Mean of winnings for each round = {}".format(np.mean(k)))
    print("Variance of winnings = {}".format(np.std(k)**2))
    print ("Total bet: " + str(bet * num_runs))
    print ("Net amount gained or lost by the player: " + str(np.
↪mean(k)*num_runs))

    print("Loss or Gain percentage: " + str((np.mean(k)*num_runs)*100/(bet *␣
↪num_runs)) + "%")

    print()
    print("95% Confidence interval:")

    conf_int = (np.mean(k) - 1.96*(np.sqrt(np.std(k)/num_runs)), np.mean(k) + 1.
↪96*(np.sqrt(np.std(k)/num_runs)))
    print(conf_int)
```

```
    display(pd.DataFrame.from_dict(results))


def blackjack_sim(init_hand='no_initial_condition', dealer_stay_num = 17,␣
 ↪init_dealer='no_initial_condition', bet=10, payback_type = "6to5",␣
 ↪num_runs=100000):
    sim_out=[]

    for i in range(num_runs):
        sim_out.append(blackjack_round(bet_amount=bet, dealer_stay_num =␣
 ↪dealer_stay_num, player_cardset=init_hand, single_dealer_card=init_dealer,␣
 ↪payback_type = payback_type))
    return (init_hand, np.array(sim_out), num_runs, bet)
```

Running 100,000 rounds and examining the statistics:

[380]:
```
sim = blackjack_sim(num_runs=10000, bet=20, payback_type = "3to2")
compute_stats(sim)
```

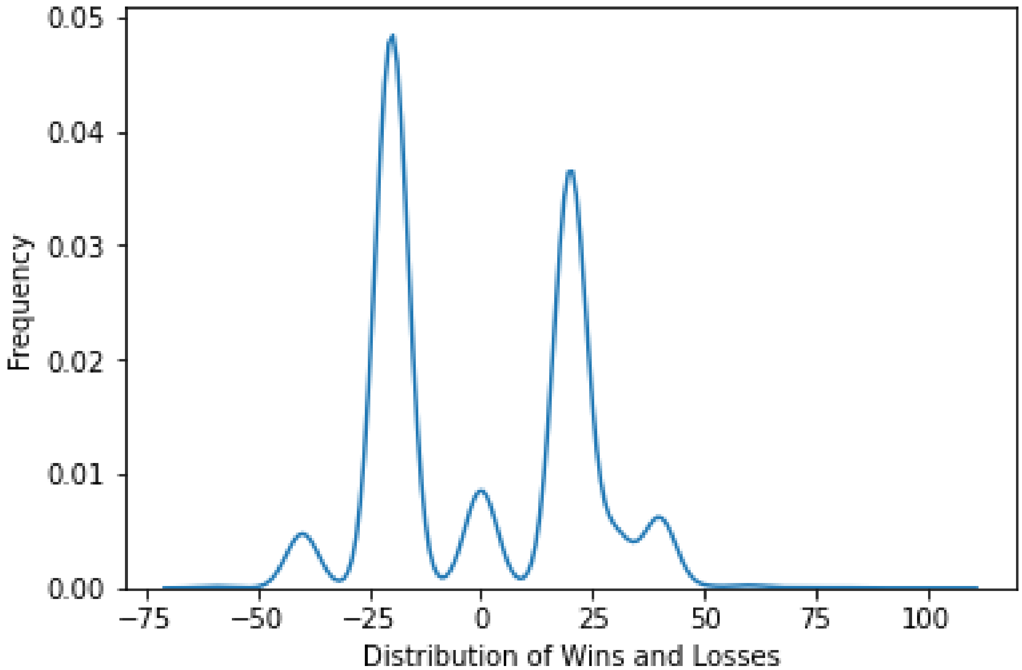
```
Mean of winnings for each round = -0.257
Variance of winnings = 526.183951
Total bet: 200000
Net amount gained or lost by the player: -2570.0
Loss or Gain percentage: -1.285%

95% Confidence interval:
(-0.3508729510347921, -0.16312704896520794)

   Number of Zero Transactions  \
0                          780

   Number of Positive Results (Including Blackjacks)  \
0                                               4351

   Number of Negative Results  Number of Blackjacks
0                        4869                   411
```



How much worse does a 6 to 5 payoff make it for the player?

[362]:
```
sim2 = blackjack_sim(num_runs=100000, bet=20, payback_type = "6to5")
compute_stats(sim2)
```

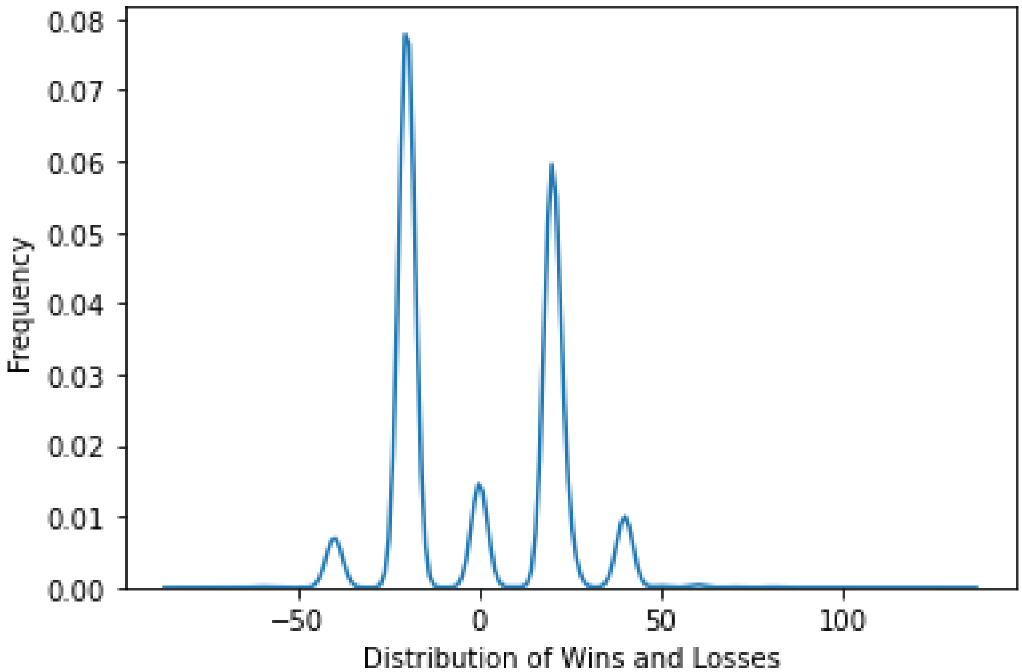
```
Mean of winnings for each round = -0.55716
Variance of winnings = 504.84613273439993
Total bet: 2000000
Net amount gained or lost by the player: -55716.0
Loss or Gain percentage: -2.7858%

95% Confidence interval:
(-0.5865395966974034, -0.5277804033025966)

   Number of Zero Transactions  \
0                         8267

   Number of Positive Results (Including Blackjacks)  \
0                                              42879

   Number of Negative Results  Number of Blackjacks
0                       48854                  4032
```



The block below shows how crucial it is for the dealer to stay at 17. Staying at 19 instead of 17 causes the player to start beating the casino quite convincingly as shown below:

[363]:
```
sim3 = blackjack_sim(num_runs=10000, bet=20, payback_type = "6to5",␣
 ↪dealer_stay_num = 19)
compute_stats(sim3)
```

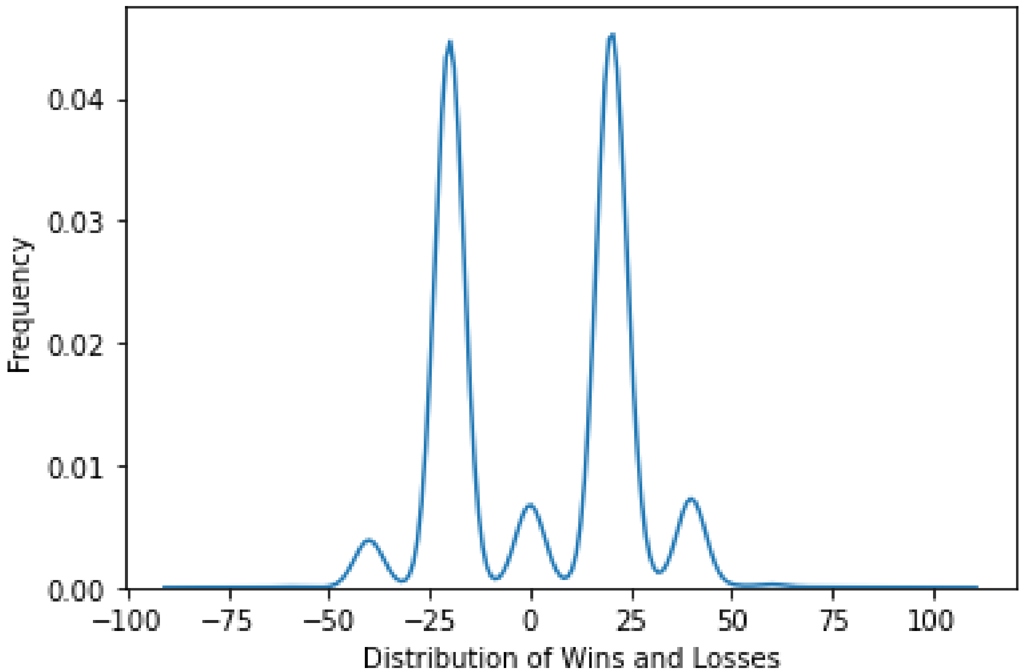
```
Mean of winnings for each round = 2.1528
Variance of winnings = 516.2246521599999
Total bet: 200000
Net amount gained or lost by the player: 21528.0
Loss or Gain percentage: 10.764%

95% Confidence interval:
(2.059374429951673, 2.246225570048327)

   Number of Zero Transactions  \
0                          610

   Number of Positive Results (Including Blackjacks)  \
0                                               5001

   Number of Negative Results  Number of Blackjacks
0                        4389                   420
```



Here, the player makes a tidy profit of about 10.76%. The kernel density plot also shows how the positive peak is slightly higher then the negative peak. With the added advantage of a blackjack payoff, the player can eventually beat the casino and make money.

```
[ ]:
```

Conclusions:
It seems to be overwhelmingly clear that casinos hold a huge majority in the long run when it comes to games such as Blackjack. It is also an excellent illustration of the Law of Large Numbers. Although this implementation does not cover certain nuanced strategies, it gets close to the 0.8% mark for percentage loss (supposedly the best possible result with optimal strategy).

It is also interesting to note that the player's chances of winning begin to improve if the dealer decides to choose a higher threshold to stop drawing cards. The KDE plots clearly indicate how in both the '3 to 2' case and the '6 to 5' case, the number of negative earnings (losses) outweighs the gains made over time.

Future work: I plan on incorporating more strategies and performing a more thorough probabilistic analysis of the game. While I do not expect to come up with anything novel that has not already been discovered, it would be interesting to explore the possibilities and add them to the 'Player Guide' function.

References:
[1] https://www.888casino.com/blog/blackjack-strategy/best-blackjack-strategies

[2] Baldwin, Roger R., et al. "The optimum strategy in blackjack." Journal of the American Statistical Association 51.275 (1956): 429-439.

[3]Charlie H. Cooke, Probability models for blackjack poker, Computers & Mathematics with Applications, Volume 59, Issue 1, 2010, Pages 108-114, ISSN 0898-1221, https://doi.org/10.1016/j.camwa.2009.08.062.
(https://www.sciencedirect.com/science/article/pii/S0898122109006543)

[4]
https://forums.saliu.com/blackjack-natural-odds-probability.html#:~:text=We%20calculate%20first%20all%20combinations,0483%20%3D%204.83%25

[5] https://towardsdatascience.com/python-blackjack-simulator-61b591ffb971

[6] https://github.com/seblau/BlackJack-Simulator