**Part-2**

**Use Cases: Finding the Winning Strategy in a Card Game using python**

A box of poker cards consists of 5252 cards, and are divided randomly into 1313smaller decks of 44 cards. Two players play a game as following:

* Each time a player pick a card from a deck and remove that deck from the table
* The number (22 to 1010, Jacks, Queen, King, Ace) of the chosen card in each move must not repeat

The person who cannot pick any other card loses the game. If all 1313 cards are chosen, then it is a tie. Is there a winning strategy for any of the players?

P/S: I would say that as a tie result can happen, a person would prefer to consider "not losing the game" before thinking about winning it.

I have tried this game and believe that the number of pairs in a 4-card deck can affect the winning strategy and that the first one can "not lose" (at least tie). However, I cannot prove this. I created this problem is based on the question: "Prove that if only one person plays this game, there is always a way to pick 13 cards of different types."

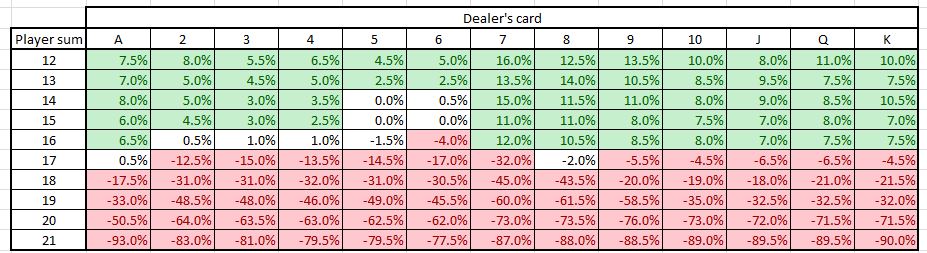
To address the problem at hand, we will need to know how to

Compute the probabilities of observable events using sample space analysis.

Plot the probabilities of events across a range of interval values.

Simulate random processes, such as coin flips and card shuffling, using Python.

Evaluate our confidence in decisions drawn from simulations using confidence interval analysis.



***Win Rate = 41.4%***

***Tie Rate = 9.5%***

***Loss Rate = 49.1%***

SHOCKER!!! Our win rate is far lower than the loss rate of the game. It would have been much better if we just tossed a coin. The biggest difference  is that the dealer wins if both the player and the dealer gets burst. If you remove that single condition, here are the win/loss rate.

***Win Rate = 41.4%***

***Tie Rate = 17.1%***

***Loss Rate = 41.5%***

from random import shuffle

class Card:

suits = ["spades",

"hearts",

"diamonds",

"clubs"]

values = [None, None,"2", "3",

"4", "5", "6", "7",

"8", "9", "10",

"Jack", "Queen",

"King", "Ace"]

def \_\_init\_\_(self, v, s):

"""suit + value are ints"""

self.value = v

self.suit = s

def \_\_lt\_\_(self, c2):

if self.value < c2.value:

return True

if self.value == c2.value:

if self.suit < c2.suit:

return True

else:

return False

return False

def \_\_gt\_\_(self, c2):

if self.value > c2.value:

return True

if self.value == c2.value:

if self.suit > c2.suit:

return True

else:

return False

return False

def \_\_repr\_\_(self):

v = self.values[self.value] +\

" of " + \

self.suits[self.suit]

return v

class Deck:

def \_\_init\_\_(self):

self.cards = []

for i in range(2, 15):

for j in range(4):

self.cards\

.append(Card(i,

j))

shuffle(self.cards)

def rm\_card(self):

if len(self.cards) == 0:

return

return self.cards.pop()

class Player:

def \_\_init\_\_(self, name):

self.wins = 0

self.card = None

self.name = name

class Game:

def \_\_init\_\_(self):

name1 = input("p1 name ")

name2 = input("p2 name ")

self.deck = Deck()

self.p1 = Player(name1)

self.p2 = Player(name2)

def wins(self, winner):

w = "{} wins this round"

w = w.format(winner)

print(w)

def draw(self, p1n, p1c, p2n, p2c):

d = "{} drew {} {} drew {}"

d = d.format(p1n,

p1c,

p2n,

p2c)

print(d)

def play\_game(self):

cards = self.deck.cards

print("beginning War!")

while len(cards) >= 2:

m = "q to quit. Any " + \

"key to play:"

response = input(m)

if response == 'q':

break

p1c = self.deck.rm\_card()

p2c = self.deck.rm\_card()

p1n = self.p1.name

p2n = self.p2.name

self.draw(p1n,

p1c,

p2n,

p2c)

if p1c > p2c:

self.p1.wins += 1

self.wins(self.p1.name)

else:

self.p2.wins += 1

self.wins(self.p2.name)

win = self.winner(self.p1,

self.p2)

print("War is over.{} wins"

.format(win))

def winner(self, p1, p2):

if p1.wins > p2.wins:

return p1.name

if p1.wins < p2.wins:

return p2.name

return "It was a tie!"

game = Game()

game.play\_game()

Then you create a new Deck object, store it in the instance variable deck, and create two Player objects using the names in name1 and name2. The play\_game method of the Game class starts the game. There is a loop in the method that maintains the game as long as there are two or more cards left in the deck, and as long as the variable response is not equal to q .

On each loop turn, you assign the variable response to user input. The game continues until the user types “q” or when there are less than two cards left in the deck. Two cards are drawn each time in the loop and the play\_game method assigns the first card to p1 and the second card to p2.

Then he prints each player’s name and the card he drew, compares the two cards to see which card is the bigger one, increments the wins instance variable for the player with the highest card, and prints a message indicating who won. The Game class also has a method called winning which takes two player items, looks at the number of tricks they’ve won, and returns the player who has won the most tricks.

When the Deck object runs out of cards, the play\_game method displays a message that the war is over, calls the winning method (passing both p1 and p2), and displays a message with the result – player name who won.