Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

**Module:** STU22004
**Date:** 03/12/2021

# Applied Probability Report
# Group Project

------------------------------------------------------------

## Contents

## Individuals participated in project

Abdelaziz Abushark: 20332134
Ali Al Ani: 19314029
Noel Peter: 20330157
Charlie Christiansson: 12303004

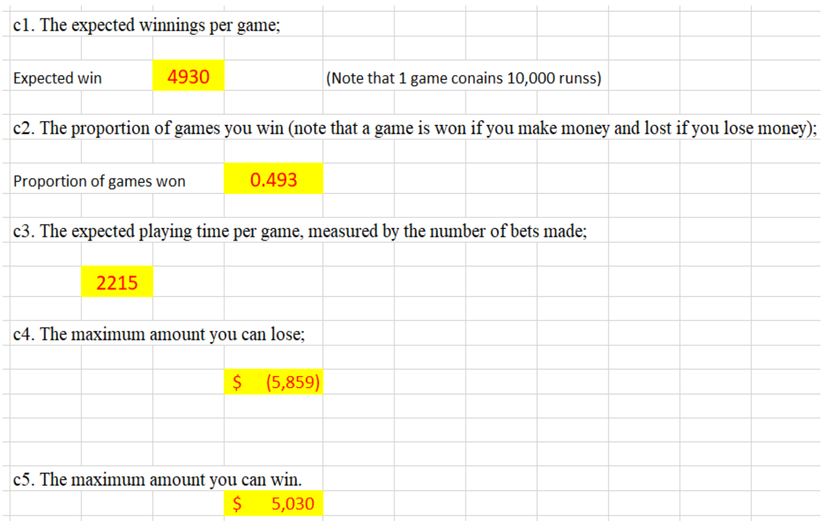
## Work distribution

Abdelaziz Abushark: (Q2) + part (a)   games 1 ,2 and 3
Ali Al Ani: Q1) part b + part (a) games 1 ,2 and 3
Noel Peter: Q1) part c
Charlie Christiansson : Q1) parts a game 4 & d

## The process of doing Q1 part A.

We decided to divide this task into 3 people: Ali , Charlie and Aziz . In which Ali and Aziz did the questions relating to games 1,2and 3 by simulating 10000 repetitions on excel and charlie did game 4 using java.
The Answers for the first 3 games (g1,g2 and g3) are

### Game 1:



c1. The expected winnings per game;

| | | |
|---|---|---|
| Expected win | 4930 | (Note that 1 game conains 10,000 runss) |

c2. The proportion of games you win (note that a game is won if you make money and lost if you lose money);

| | |
|---|---|
| Proportion of games won | 0.493 |

c3. The expected playing time per game, measured by the number of bets made;

2215

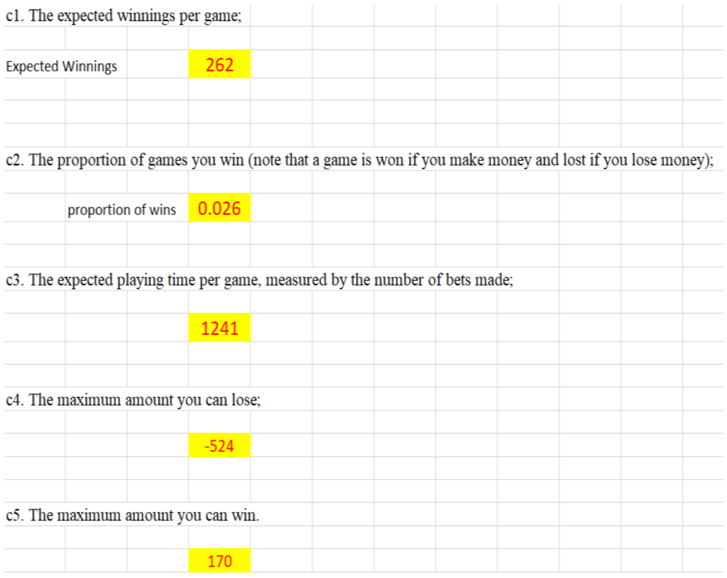
c4. The maximum amount you can lose;

$ (5,859)

c5. The maximum amount you can win.

$ 5,030

## Game 2:



c1. The expected winnings per game;

| | |
|---|---|
| Expected Winnings | 262 |

c2. The proportion of games you win (note that a game is won if you make money and lost if you lose money);

| | |
|---|---|
| proportion of wins | 0.026 |

c3. The expected playing time per game, measured by the number of bets made;

1241

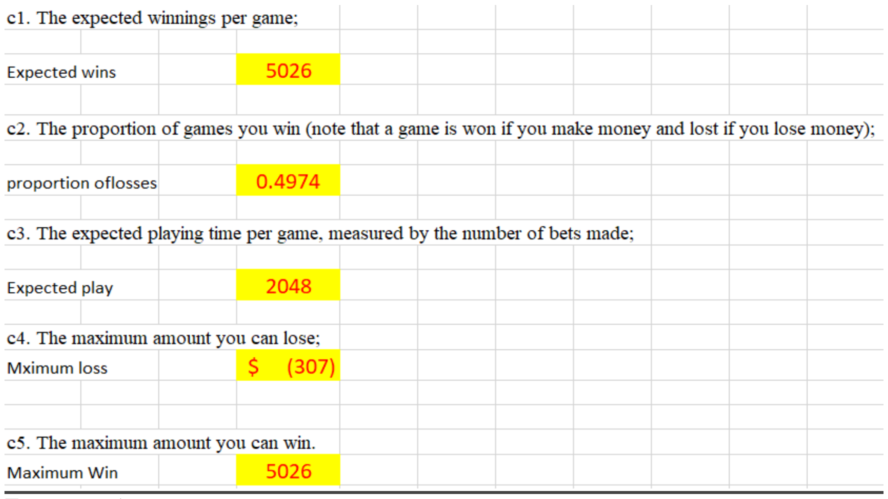
c4. The maximum amount you can lose;

-524

c5. The maximum amount you can win.

170

## Game 3:



c1. The expected winnings per game;

| | |
|---|---|
| Expected wins | 5026 |

c2. The proportion of games you win (note that a game is won if you make money and lost if you lose money);

| | |
|---|---|
| proportion oflosses | 0.4974 |

c3. The expected playing time per game, measured by the number of bets made;

| | |
|---|---|
| Expected play | 2048 |

c4. The maximum amount you can lose;

| | |
|---|---|
| Mximum loss | $ (307) |

c5. The maximum amount you can win.

| | |
|---|---|
| Maximum Win | 5026 |

For game 1:

We got the expected win by using countif function to count all the ‘+1’
We got the proportion of games won was using countif to calculate all the ‘+1’/10000
We got the expected playing time per game by using excel and later verified by python as we used the same seed for both excel and python (123456789)
We got the minimum amount you can lose using MIN to count out 10000 reps.
We used the maximum MAX formula in excel to calculate the maximum amount you can win.

For game 2:
We got the expected win by using countif function to count all the ‘+35’
We got the proportion of games won was using countif to calculate all the ‘+35’/10000
We got the expected playing time per game by using excel and later verified by python as we used the same seed for both excel and python (123456789)
We got the minimum amount you can lose using MIN to count out 10000 reps.
We used the maximum MAX formula in excel to calculate the maximum amount you can win.

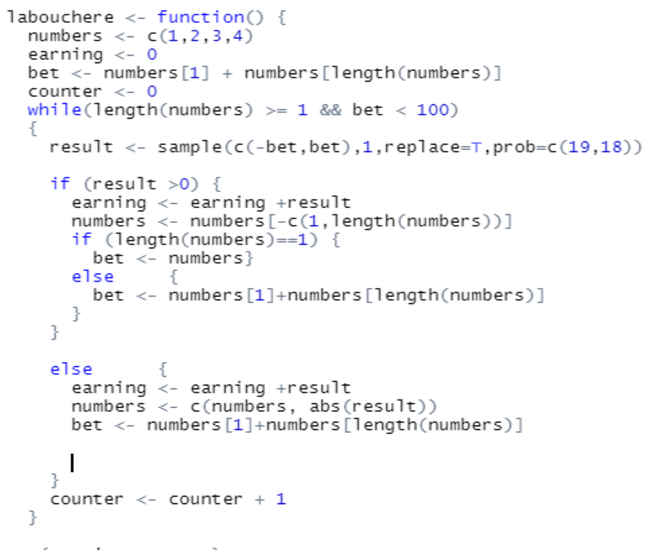
For game 3:
We got the expected win by using countif function to count all the ‘+1’
We got the proportion of games won was using countif to calculate all the ‘0’/10000
We got the expected playing time per game by using excel and later verified by python as we used the same seed for both excel and python (123456789)
We got the minimum amount you can lose using MIN to count out 10000 reps.

We used the maximum MAX formula in excel to calculate the maximum amount you can win.
For game 4 (Charlie has to do this part)

## Game 4: Labouchere System

Using the following code from R we can simulate the labouchere system to calculate:
C1. The expected winnings per game
C2. The proportion of games you win.
C3. The expected playing time per game, measured by the number of bets made.
C4. The maximum amount you can lose.
C5. The maximum amount you can win.

The following is the code from R to simulate the labouchere system:
C1. Expected Winnings per game.



```
labouchere <- function() {
  numbers <- c(1,2,3,4)
  earning <- 0
  bet <- numbers[1] + numbers[length(numbers)]
  counter <- 0
  while(length(numbers) >= 1 && bet < 100)
  {
    result <- sample(c(-bet,bet),1,replace=T,prob=c(19,18))

    if (result >0) {
      earning <- earning +result
      numbers <- numbers[-c(1,length(numbers))]
      if (length(numbers)==1) {
        bet <- numbers}
      else      {
        bet <- numbers[1]+numbers[length(numbers)]
      }
    }

    else       {
      earning <- earning +result
      numbers <- c(numbers, abs(result))
      bet <- numbers[1]+numbers[length(numbers)]

    }
    counter <- counter + 1
  }
```

To simulate the the code 10,000 times as required we use the following function:

```
earnings <- function() {
  mean(replicate(100000,labouchere()[1]))
}
earnings()
```

After simulating 10,000 times we get a result of **-3.63985.**

C2. The proportion of games you win

```
probabilitywin <- function() {
  labouchere <- replicate(100000, probabilitywin )

  return(list(mean(labouchere), sd(labouchere)))
}
```

After simulating 10,000 times we get a result of **.95713**, which gives us a win rate of **95.713%.**

C3. The expected playing time per game, measured by the number of bets made.

```
numberofbets <- function() {
  (sum(replicate(100000,labouchere()[2])))/100000
}
numberofbets()
```

After simulating 10,000 times we get a result of **8.7.**

C4. The expected amount of money a player can lose = **$4940.**

C5. The maximum amount of money a player can earn = **$10.**

## The process of doing Q1 part B.

I was tasked with doing Q1 part B and to do this question, I had to wait until my colleague had finished Q1 A to be able to use his tests and simulation to calculate my winnings, games won and percentage error. However, as I waited, I contemplated whether to do my own simulation in code such as Python. This became daunting as it would make no sense for me to have to do the same code as my colleague because it would be a waste of time. Thus, I turned to excel where it would've been much easier for me to simulate the games in a shorter period of time and still be able to get good results that I can share.

Question 1 B required me to run simulations for Game 1 and 2, checking my estimates for C1 and C2 and calculating the percentage error.

## Game 1:

- I used a Rand (0,1) and dragged it down 10,000 times.
- I calculated the probability of a red outcome. 18/37 = 0.48649
- If my random number is greater than 0.48649 then it is considered a loss (-1). Otherwise it is a win (+1).
- My theoretical winnings were -270.27 (0.48649 * 10000 - 10000 + 4864.9)
- I found the formula for percentage error (Measured Value - Theoretical Value / Theoretical Value * 100).
- My expected games won /10000 = 4864.

To calculate my percentage error for C1 (winnings per game) I used the percentage error formula Measured Value - Theoretical Value / Theoretical Value * 100.
My theoretical value is -270.27. After 10,000 simulations I ended with an actual winnings of -248 (sum of all wins or losses (+1 or -1s). Plugging these two numbers into my formula I ended up with an answer of -8.24 (8.24%).

As for games won (C2). Our theoretical games won is simply 0.4864 * 10000 = 4864 games.
My actual games won after 10,000 simulations was 4876.
With my result of 4876 and theoretical value of 4864, I once again plugged the values into the percentage formula [4876 - 4864] / 4864 * 100.

## Game 2:

Game 2 was very similar to Game 1 with a few different probability changes, such as the probability of winning changing to 1/37 (2.702%) (Theoretical Average).
I used the same method for Game 2:

- Using Rand(0,1) and dragging down 10,000 times.
- Probability of winning is 1/37 (2.702%), if my random number is greater than 0.02702 then it is considered a loss, otherwise it is a win.
- My winnings were calculated as +35 for a win and -1 for a loss.
- I once again used the percentage error formula (Measured Value - Theoretical Value / Theoretical Value * 100)
- I used the countif function in excel to calculate how many times we received a +35 and -1 to determine how many games we lost and won.

To calculate the error percentage of winnings. I first multiplied the theoretical games won * 10,000 which is 270.2. I took 270.2 and multiplied it by 35 to get 9457 theoretical total winnings and multiplied 9729.8 by -1 to get my theoretical total losses. I then calculated my winnings by summing all the -1s and +35 I received in my 10,000 simulations to get my measured winnings.
My measurements were:
271(games won) * 35 = 9485 winnings
9729 (games lost) * -1 = -9729 total loses
-244 total winnings + losses

Theoretical winnings + losses =
9457 - 9729.8 = -272.8
-272.8 theoretical winnings + losses

Percentage error = (((-244 - (-272.8)) / -272.8 * 100) = 10.55%

As for games won, using the countif function I was able to find that we won 271 games out of 10000 which is 2.71% of the games we simulated. Plugging this value as well as our theoretical value of 2.72% we can calculate the percentage error of [(0.0271 - 0.0272) / 0.0272 * 100] = 0.296077 which is the percentage error of games won.

Final Answers:
**C1 G1 - 8.24%**
**C2 G1 - 24.67%**

**C1 G2 - 10.55%**
**C2 G2 - 29.6%**

## Q1) Part C

Before we are about to bet real money on one of these roulette games, it is really important that we familiarize ourselves with the odds in these games and what chances we stand in winning. This part of the assignment involves taking the running average and running variance of the
1. The expected winning per game.
2. The proportion of games won.

Game 1:
This game has 2 possibilities: either you get red, or you get either of the other colours (Green or Black). The probability of getting red is 18/37 and the probability of getting either of the other colours is 19/37. In this part I used the data provided by the simulations as done by my lab partner @Aziz

I calculated the expected winning for the game using the formula which came out to be = 47 %
Then I calculated the proportion of games won from the simulation = 4891 games

Game 2:
This game has the same 2 possibilities: either you get the number you bet on or you get either of the other 37 numbers. The probability of winning (getting the number you bet on) is 1/37 and the probability of losing (either of the other numbers) is 16/37. In this part I used the data provided by the simulations as done by my lab partner @Aziz

Then I calculated the expected winning for the game which came up to be = 2.7%
Also calculating the proportion of games won from the simulation = 286 games

Game 3:
This game has the same 2 possibilities: either you get the red or you get either of the other colors. The probability of getting red is 18/37 and the probability of either getting the other colors is 19/37. Again, in this part I used the data provided by the simulations as done by my lab partner @Aziz

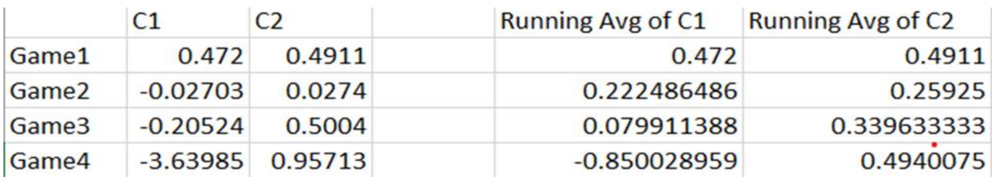
Where I then calculated the expected winning for the game using the formula B(1-(2q) ^n)) (where B = amount of the initial bet (1), q = prob of losing (19/37), n = finite no of bets the gambler can afford (7 bets to reach 100$)) = 20.5%
Then I calculated the proportion of games won from the simulation = 5020 games

Game 4:
This game has the same 2 possibilities: either you get the red or you get either of the other colours. The probability of getting red is 18/37 and the probability of either getting the other colours is 19/37. In this part I used the data provided by the simulations as done by my lab partner @Charlie

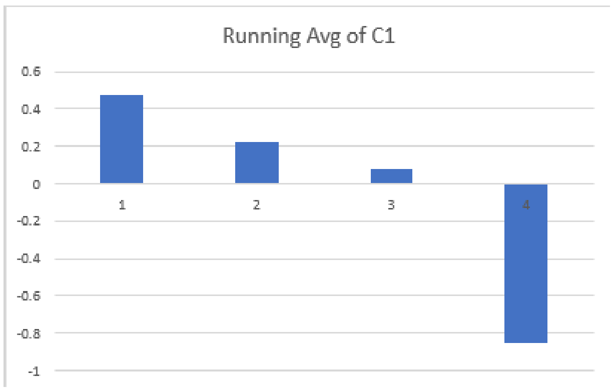
The calculated the expected winning for the game which came up to be = 3.6%
Also calculating the proportion of games won from the simulation = 9571 games

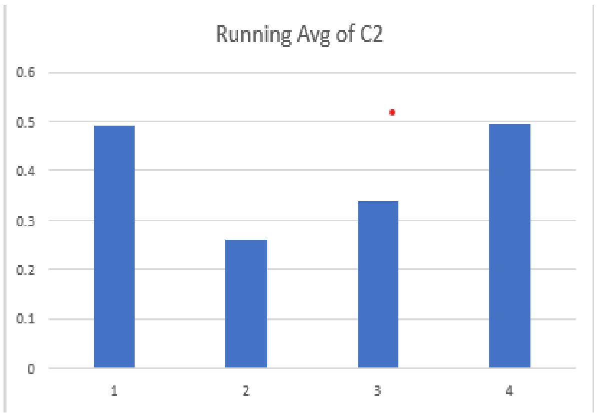
Then to calculate the running average and the running variance of the games I created a new sheet in excel with all the c1's and c2's from all the games

For the Running Average I used the formula Average () in excel by using the absolute of the first cell and dragging it to the rest of the c1's and c2's. The value projected from the formula used were as follows



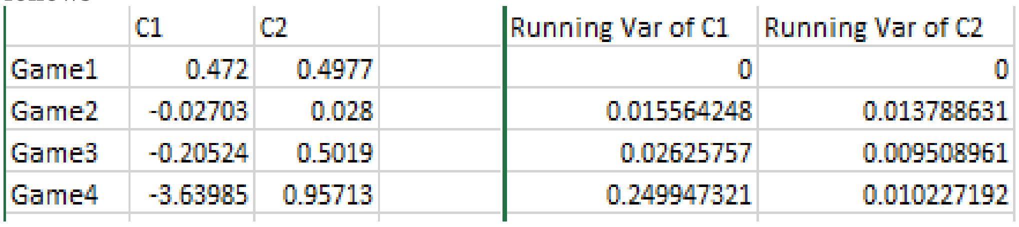
| | C1 | C2 | | Running Avg of C1 | Running Avg of C2 |
|---|---|---|---|---|---|
| Game1 | 0.472 | 0.4911 | | 0.472 | 0.4911 |
| Game2 | -0.02703 | 0.0274 | | 0.222486486 | 0.25925 |
| Game3 | -0.20524 | 0.5004 | | 0.079911388 | 0.339633333 |
| Game4 | -3.63985 | 0.95713 | | -0.850028959 | 0.4940075 |





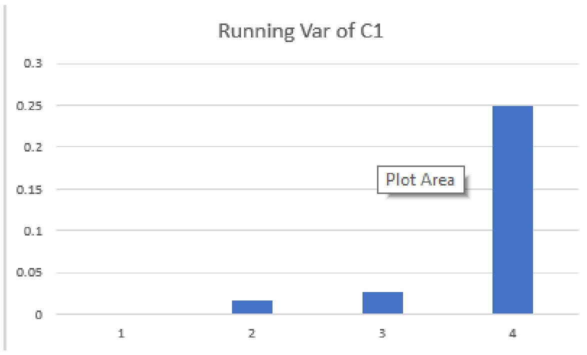
Also, I used the graph in excel to visualize the running average of C1 and C2.

For the Running Variance I used the formula Var.P() in excel by using the absolute of the first cell and dragging it to the rest of the c1's and c2's. The value projected from the formula used were as follows

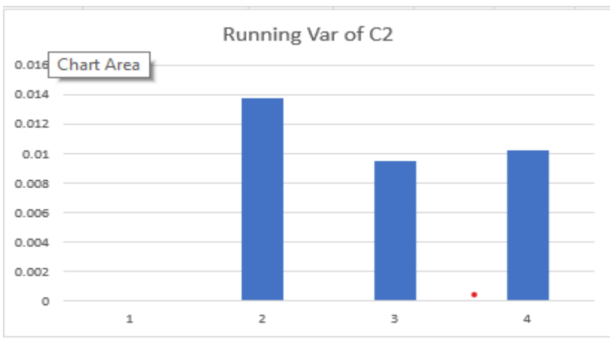


| | C1 | C2 | | Running Var of C1 | Running Var of C2 |
|---|---|---|---|---|---|
| Game1 | 0.472 | 0.4977 | | 0 | 0 |
| Game2 | -0.02703 | 0.028 | | 0.015564248 | 0.013788631 |
| Game3 | -0.20524 | 0.5019 | | 0.02625757 | 0.009508961 |
| Game4 | -3.63985 | 0.95713 | | 0.249947321 | 0.010227192 |

Also, I used the graph in excel to visualize the running variance of C1 and C2



To



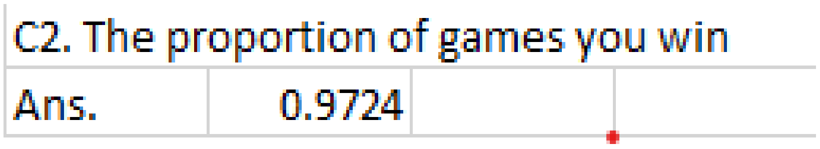
answer the second part of the question I thought to myself could it be reliable as even 10,000 is a large number can be relying on these numbers so regardless, I tried to increase the number of repetitions for the first game

Game1
To simulate randomness in the winning and losing the game I used the formula Rand (0,1) which returned and evenly distributed random real numbers in (0,1). Then I used the same formula to simulate 20,000 repetitions of the game. Then I used the same concept as used by my group partner in which, if the random number is greater than the probability of getting a RED then it is considered a loss (-1). And otherwise, it is considered a win (1).

Game 2
Again, in this game I used the same concept as used in game 1 in which if the random number is greater than the probability of getting the number then it is considered a loss (-1), and otherwise, it is considered a win (35) to simulate 20,000 repetitions of the game.

Game 3
Again, to simulate randomness in the winning and losing the game I used the formula Rand (0,1) which returned and evenly distributed random real numbers in (0,1) and used the same formula to simulate 20,000 repetitions of the game. Then I used the same concept as used in game 1 in which if the random number is greater than the probability of getting the number then it is considered a loss (You must double the previous bet). And otherwise, it is considered a win (you bet 1$ again).

After calculating both the C1 and C2 from the simulation for the games it was observed that the more times you play the odds of you winning increases substantially for every game as well as game 4. Given below is the C2 for game1 which shows the increase in the proportion of games won from 0.483 in 10,000 repetitions to 0.9724 in 20.000 repetitions.


C2. The proportion of games you win

| Ans. | 0.9724 | | |
|---|---|---|---|

Q1) Part D

| Game | Expected Winning min-max | Proportions won min-max | Expected Player time min-max |
|---|---|---|---|
| G1 | -0.02813,-0.022 | 0.484, 0.48762 | 1,1 |
| G2 | -0.043 ,-0.0011 | 0.02586,0.02746 | 1,1 |
| G3 | -1.0293 ,-0.658 | 0.9064,0.917 | 17.647,17.665 |
| G4 | -3.89,-3.4314 | 0.945,0.972 | 8.173, 8.81142 |

| Game | Winning - mean, std dev | Proportions - won mean, std dev | Player time - mean, std dev |
|---|---|---|---|
| G1 | -0.02749 ,0.9996 | 0.48668,0.4998 | 1,0 |
| G2 | -0.029 , 0.0043 | 0.02681,0.09531 | 1,0 |
| G3 | -0.98972 ,37.142 | 0.9214,0.27364 | 17.5138,4.19 |
| G4 | -3.641 ,69.0473 | 0.9478, 0.20477 | 8.614, 7.789 |

After simulating each game we have found that the **Labouchere System** has the most variable game by amount won and by expected playing time.

## The process of doing the code for the 2nd Question

I started doing the second question using **Python (Jupyter notes)** as my software as I felt it is the easiest software to use and get the right answer as it has special libraries and functions that will save me some time doing my mathematical equations and double checking the answers.

## The Functions and Libraries used to write the code

I used functions and libraries to make my work easier and simpler:

- Function append which was used to add the card to the list instantly
- Library numpy which I used to do the complex mathematical equations and functions as later I used it to verify my answer for example variance = math.var(shuffle) in which does the math equation for variance for me which also done manually but later verified by numpy as the manual equation I used for variance was var =sum(x-avg)**2 for x in shuffleCardList)/len(shuffleCardList) which can be done in a more simple way using numpy which is variance = math.var(shuffleCardList) . In addition I used numpy to verify my expectation equation and answer for me which I wrote in code which was avg = sum(shuffleCardList)/len(shuffleCardList) which was later verified by numpy using that code expectation=math.mean(shuffleCardList) .
- Library tensfor flow which I later used to double check numpy library and my manual code that I wrote in which my code for numpy and manual code were variance = math.var(shuffleCardList) , var =sum(x-avg)**2 for x in shuffleCardList)/len(shuffleCardList) which I later verified using tensor flow by variance=tf(shuffle) which gave me the same answer as well as I used it for double checking my expectation in which my numpy and manual code were avg = sum(shuffleCardList)/len(shuffleCardList) and expectation=math.mean(shuffleCardList) which was later verified by tensor flow using this code tf.mean(shuffleCardList)
- Function random which was used to generate random floating numbers and to randomize the cards process as we have 100 cards as it was used to randomly shuffle the cards
- Function shuffle which I was pretty surprised to find it on Python which helped me a lot to shuffle the cards as I linked function random with it to make the process random as function shuffle takes a sequence, eg list of items and change the original list then after than I linked the numpy library to it to do the mathematical calculations for me which resulted in math.random.shuffle(cards)
- Library matplotlib.pyplot which I later used to draw me the graphs of the expectation and variance of 10000 repetitions which I will be attaching below

## The code and the methods used

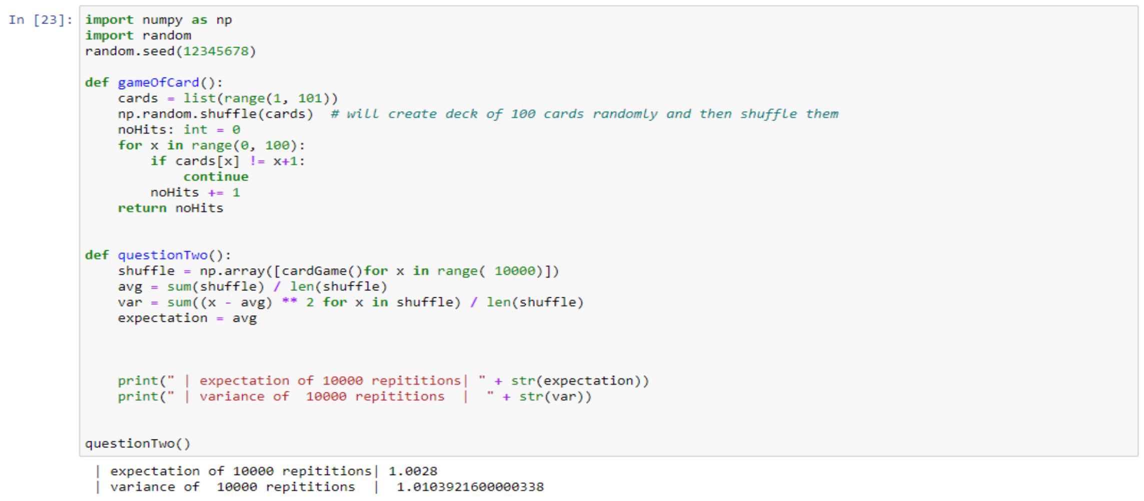
- I used one method which is called gameOfCard and main method which is called questionTwo as I Interlinked the two methods to generate the final answer for me
- I started my main method def questionTwo() by writing an array list for the cards which is shuffle and
- I added a for loop for the 10000 repetitions that was asked for the question to do which was for x in range (0,1000)
- I used the function append to add the card list instantly and linked it to the 2nd method which is def gameOfCard() .
- I then wrote the math equation for the variance and the expectation which were avg = sum(shuffleCardList)/len(shuffleCardList) in which avg represents the expectation and then the variance which was var = sum(x-avg)**2 for x in shuffleCradList) / len(shuffleCardList).
- I finally ended my code by printing out the result which is print("expectation) and print("var)
- The 2nd method was was gameOfCards() in which I coded the first part of the question in which it asked that the deck has 100 cards so I created a card list that has range from (0,100)
- I then interlinked numpy library and the 2 functions random and shuffle to make the procedure random and simple as I explained what numpy , shuffle and random do in the previous point which is **The Functions and Libraries used to write the code**
- I then proceeded to calculate the number of hits as the question asked us to count the numbers of hits when the cards turned over so it would have the same range as the number of cards.
- I created a for loop to generate the number of hits as we will have a hit whenever a card is turned over and I printed out the different combinations of hits
- I followed it by an If statement to test as if the hit was not in the same range as cards which was (0,100) ro continue and not return anything but if it was in the same range it would return the number of hits which was referred as noHits
- I added the library matplotlib.pyplot to draw me the graphs by using plt.plot so I can visualize the final results and added a title to it by using plt.title
- I then added a random seed which will generate me specific random numbers to check both results from my manual calculations and numpy library
- Finally, the code is now done and ready to be used to generate the 10000 simulations.

## Pictures of the code used and the output to generate the final answer

**Code I wrote that used my manual calculations:**

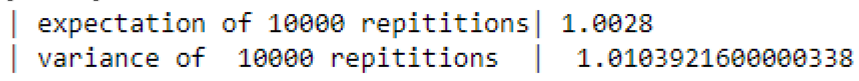
- You can see the calculations clearly and the 2 methods used

- You can see the output of the function, which is below, and I will post it again below it to confirm it.



```
In [23]: import numpy as np
         import random
         random.seed(12345678)

         def gameOfCard():
             cards = list(range(1, 101))
             np.random.shuffle(cards)  # will create deck of 100 cards randomly and then shuffle them
             noHits: int = 0
             for x in range(0, 100):
                 if cards[x] != x+1:
                     continue
                 noHits += 1
             return noHits


         def questionTwo():
             shuffle = np.array([cardGame()for x in range( 10000)])
             avg = sum(shuffle) / len(shuffle)
             var = sum((x - avg) ** 2 for x in shuffle) / len(shuffle)
             expectation = avg



             print(" | expectation of 10000 repititions| " + str(expectation))
             print(" | variance of  10000 repititions  |  " + str(var))


         questionTwo()

          | expectation of 10000 repititions| 1.0028
          | variance of  10000 repititions  |  1.0103921600000338
```

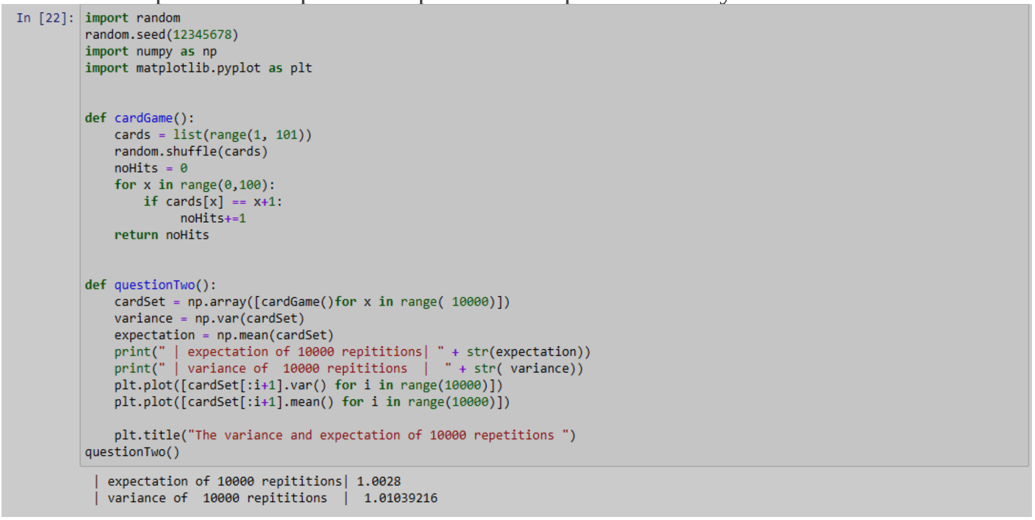
**Sample output:**



```
| expectation of 10000 repititions| 1.0028
| variance of  10000 repititions  |  1.0103921600000338
```

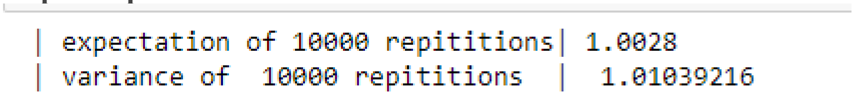
**Code I wrote that using library numpy to confirm my answer:**

- You can clearly see the numpy library
- You can clearly notice that the answers are the same
- Compare this output to me previous output to see they are the same



```
In [22]: import random
         random.seed(12345678)
         import numpy as np
         import matplotlib.pyplot as plt


         def cardGame():
             cards = list(range(1, 101))
             random.shuffle(cards)
             noHits = 0
             for x in range(0,100):
                 if cards[x] == x+1:
                      noHits+=1
             return noHits


         def questionTwo():
             cardSet = np.array([cardGame()for x in range( 10000)])
             variance = np.var(cardSet)
             expectation = np.mean(cardSet)
             print(" | expectation of 10000 repititions| " + str(expectation))
             print(" | variance of  10000 repititions  |  " + str( variance))
             plt.plot([cardSet[:i+1].var() for i in range(10000)])
             plt.plot([cardSet[:i+1].mean() for i in range(10000)])

             plt.title("The variance and expectation of 10000 repetitions ")
         questionTwo()

          | expectation of 10000 repititions| 1.0028
          | variance of  10000 repititions  |  1.01039216
```

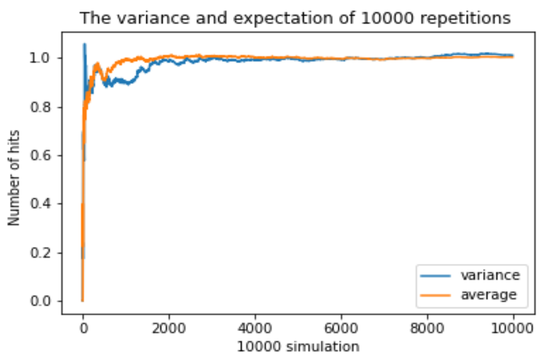
**Sample output :**



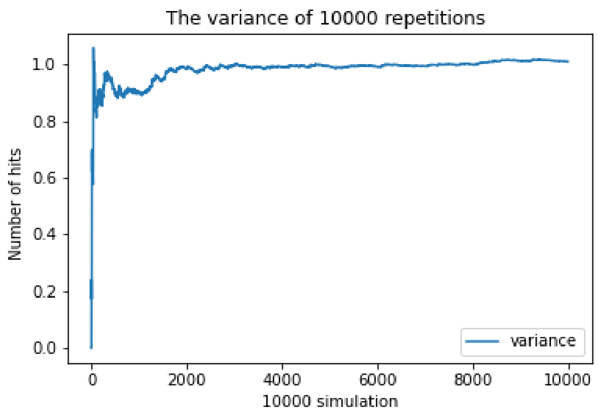
```
| expectation of 10000 repititions| 1.0028
| variance of  10000 repititions  |  1.01039216
```

## Graphs that I drew to make me visualize the final results :

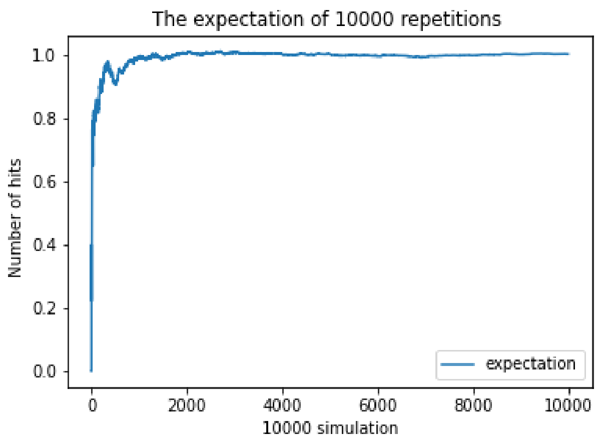
- The variance and expectation of 10000 repetitions



- The variance of 10000 repetitions



-
- The Expectation of 10000 repetitions



By adding a random seed function I can say that it limits the random numbers to a limitation to 10000 simulations and the final answer estimate is :

- Notice that if I did not add the random.seed it will keep generating different simulations of numbers and will give me different results every time I run my program so random.seed() function is very important that help you generate a limitation of 10000 simulations and get the estimation for the final answer.

- **The expectation of the total number of hits: 1.0028**
- **The variance of the total number of hits: 1.0103921600000338**

**For the full code please visit : https://github.com/azizosharke/AppliedPorbability**