Mid Assessment Task 1

Harpreet Singh
s223925166

SIG742-Modern Data Science

Essay Report

Part II
Python Foundations for Data Science

Introduction

Who would've thought that an old TV game show could inspire a statistical problem that has tripped up mathematicians and statisticians with Phds? **The Monty Hall problem** has confused people for decades. In the game show, Let's Make a Deal, Monty Hall asks you to guess which closed door a prize is behind. The answer is so puzzling that people often refuse to accept it! The problem occurs because our statistical assumptions are incorrect.

The Monty Hall problem is a brain teaser, in the form of a probability puzzle, loosely based on the American television game show *Let's Make a Deal* and named after its original host, Monty Hall. The problem was originally posed (and solved) in a letter by Steve Selvin to the *American Statistician* in 1975.[1][2] It became famous as a question from reader Craig F. Whitaker's letter quoted in Marilyn vos Savant's "Ask Marilyn" column in *Parade* magazine in 1990

What is the Monty Hall problem ?

The problem is stated as follows. Assume that a room is equipped with three doors. Behind two are goats, and behind the third is a shiny new car. You are asked to pick a door, and will win whatever is behind it. Let's say you pick door 1. Before the door is opened, however, someone who knows what's behind the doors (Monty Hall) opens *one of the other* two doors, revealing a goat, and asks you if you wish to change your selection to the third door (i.e., the door which neither you picked nor he opened). The Monty Hall problem is deciding whether you do.

Monty Hall asks you to choose one of three doors. One of the doors hides a prize and the other two doors have no prize. You state out loud which door you pick, but you don't open it right away.

Monty opens one of the other two doors, and there is no prize behind it.

At this moment, there are two closed doors, one of which you picked.

The prize is behind one of the closed doors, but you don't know which one.

Monty asks you, "Do you want to switch doors?"

The majority of people assume that both doors are equally like to have the prize. It appears like the door you chose has a 50/50 chance. Because there is no perceived reason to change, most stick with their initial choice.

Purpose

The Monty Hall problem – Do you want to switch the door?

You will need to write a code to find out what is the probability to get the car if you switch the door?

Analysis

In order to derive solution, we will be assessing two questions : -

- **What is the probability to get the car if the user switches the door?**
- **What is the probability to get the car if the user does not switch the door?**

We are using Python program simulation to analyze both the problem statement
For this we are using **random** and **numpy** module from python

From random module, we are using **random.choice()** function
The **choice()** method returns a randomly selected element from the specified sequence. The sequence can be a string, a range, a list, a tuple or any other kind of sequence.
Example:

```
print(f"\nRandomly selected element from {[34,5,12,4]}: {random.choice([34,5,12,4])}")
print(f"\nRandomly selected element from {['HArpreet','DataScience','Masters']}: {random.choice(['Harpreet','DataScience','Masters'])}")
print(f"\nRandomly selected element from {range(1,4)}: {random.choice(range(1,4))}")
```

```
Randomly selected element from [34, 5, 12, 4]: 5

Randomly selected element from ['HArpreet', 'DataScience', 'Masters']: Harpreet

Randomly selected element from range(1, 4): 1
```

From **numpy** module , we are using **numpy.random.randint()** function numpy.random.randint function is used to get random integers from low to high values. The low value is included while the high value is excluded in the calculations. The output values are taken from the discrete uniform distribution of the range values.

```
print(f"\nRandomly generated number from {list(range(1,4))}: {np.random.randint(1,4)}")
print(f"\nRandomly generated number from {list(range(1,10))}: {np.random.randint(1,10)}")
```

```
Randomly generated number from [1, 2, 3]: 3

Randomly generated number from [1, 2, 3, 4, 5, 6, 7, 8, 9]: 8
```

To calculate the probability of both the problem statement,
We need below variables whose values are generated using **numpy.random.randint() & random.choice()** :-

Car_variable_door - generated number among (1,2,3) using random choice to indicate which door has the car

Challenger_door - generated number among (1,2,3) using random choice to indicate the choice that challenger has selected

Host_selected_door - generated number that is not same as the challenger's choice (Challenger_door) and also did not has the car (Car_variable_door)

Challenger_door_changed - generated number using random choice if the challenger wishes to change his door number

```
[47] car_variable_door = np.random.randint(1,4)
     challenger_door = np.random.randint(1,4)
     host_selected_door = random.choice([i for i in range(1,4) if i not in [car_variable_door,challenger_door]])
     challenger_door_changed = random.choice([i for i in range(1,4) if i not in [host,challenger_door]])
     print(f"Door in which car is present: {car_variable_door}")
     print(f"\nDoor that is selected by challenger: {challenger_door}")
     print(f"\nDoor that is not same as the challenger's choice  and also did not has the car: {host_selected_door}")
     print(f"\nDoor that is wished by challenger to be changed when asked: {challenger_door_changed}")
```

```
Door in which car is present: 1

Door that is selected by challenger: 3

Door that is not same as the challenger's choice  and also did not has the car: 2

Door that is wished by challenger to be changed when asked: 2
```

```
[48] car_variable_door = np.random.randint(1,4)
     challenger_door = np.random.randint(1,4)
     host_selected_door = random.choice([i for i in range(1,4) if i not in [car_variable_door,challenger_door]])
     challenger_door_changed = random.choice([i for i in range(1,4) if i not in [host,challenger_door]])
     print(f"Door in which car is present: {car_variable_door}")
     print(f"\nDoor that is selected by challenger: {challenger_door}")
     print(f"\nDoor that is not same as the challenger's choice  and also did not has the car: {host_selected_door}")
     print(f"\nDoor that is wished by challenger to be changed when asked: {challenger_door_changed}")
```

```
Door in which car is present: 3

Door that is selected by challenger: 1

Door that is not same as the challenger's choice  and also did not has the car: 2

Door that is wished by challenger to be changed when asked: 3
```

Now we have all the required variables to calculate the required probability.

Now we define two functions : -

- **no_switch_doors();**
- **switch_doors();**

no_switch_doors() calculate the probability of winning the car if the user does not wish to switch the door
switch_doors() calculate the probability of winning the car if the user wishes to switch the door

Both the functions use the above created variables to calculate the probability
Car_variable_door
Challenger_door
Host_selected_door
Challenger_door_changed

Apart from above variables, we create three more variables:-

Wins - count the number of times user wins the car
If the door selected by challenger has the same door the car is in —---> A
Challenger_door == car_variable_door
Or
If the door changed by user has the same door the car is in —----> B
Challenger_door_changed == car_variable_door

Losses - count the number of times user losses the car
If A and B conditions fails, the user losses

Win_perc- stores the probability or winning percent of the car
Win_perc = (wins) / (wins + losses)

Both function will iterate n times and returns the winning percent / or probability in every iteration

In each iteration, function will create the values of required variables using above logic

Car_variable_door
Challenger_door
Host_selected_door
Challenger_door_changed
Wins
Losses
Win_perc
Number_of_iterations - defined in function to run n iterations

Function definition for no_switch_doors() :-

```
# Function to evaluate the probabilty if the user does not want to switch the door

def no_switch_doors():
    no_switch_doors = []
    wins = 0
    losses = 0
    for n in tqdm(range(1,1000)):
        car_variable_door = np.random.randint(1,4)
        challenger_door = np.random.randint(1,4)
        host = random.choice([i for i in range(1,4) if i not in [car_variable_door,challenger_door]])
        # print()
        # print(car_variable_door,challenger_door,host)
        if challenger_door == car_variable_door:
            wins = wins + 1
        else:
            losses = losses + 1

        win_perc = wins / (wins + losses)
        no_switch_doors.append(win_perc)
        # print(f"\nProbabilty if the User does not want to switch the door running {n} times:--> {win_perc}")
    return (win_perc),no_switch_doors
```

Here,

- Wins and losses are initialized with value 0.
- No_switch_door is an empty list to store the probability values in between the running iterations
- Number_of_iterations - storing the values from 1-1000
- Function is calculating all variable values using a for loop till the values exhausted in range function

- In each iteration,challenger_door value is compared to car_variable_door and if value matches, wins value is incremented by 1
- Otherwise losses value is incremented by when as the challenger losses

- After that win_perc is calculated using the values generated above using wins & losses
- Finally the function return the win_percent value and No_switch_door list after all iterations are completed

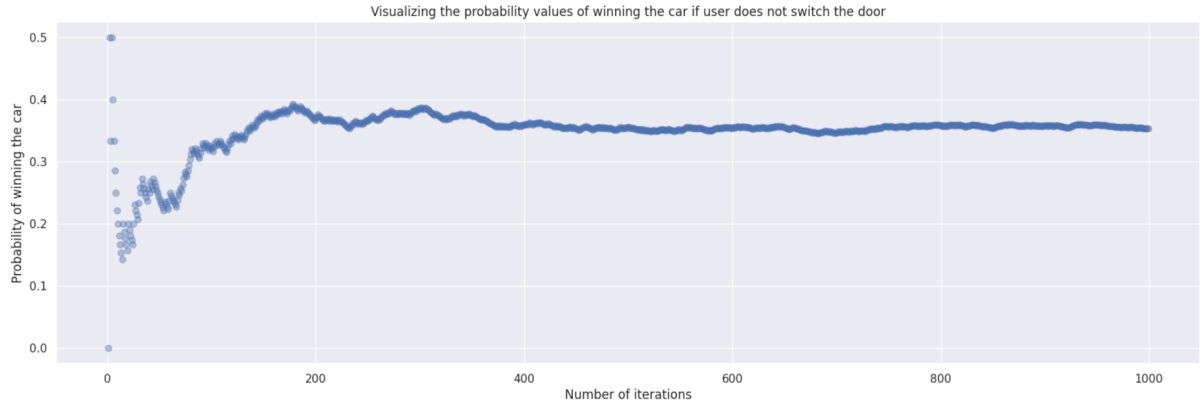
```
            # print(f"\nProbabilty if the User does not want to switch the door running {n} times:--> {win_perc}")
        return (win_perc),no_switch_doors
```

```
[137] result,no_switch_doors = no_switch_doors()
      print(f"\n\nProbabilty if the User does not want to switch the door:--> {result}")
```

```
100%|██████████| 999/999 [00:00<00:00, 62292.57it/s]

Probabilty if the User does not want to switch the door:--> 0.3163163163163163
```

```
[138] plt.figure(figsize = (20,6))
      plt.scatter(x = range(1,1000),y=no_switch_doors,alpha=0.4)
      plt.xlabel("Number of iterations")
      plt.ylabel("Probability of winning the car")
      plt.title("Visualizing the probability values of winning the car if user does not switch the door");
```

```
[149] plt.figure(figsize = (20,6))
      plt.scatter(x = range(1,1000),y=no_switch_doors,alpha=0.4)
      plt.xlabel("Number of iterations")
      plt.ylabel("Probability of winning the car")
      plt.title("Visualizing the probability values of winning the car if user does not switch the door");
```



From the above iterations, we see that the probability of winning the car if the user does not want to switch the door settles down to **0.3**

Now we calculate the probability of winning if user does want to switch the door

Function definition for switch_doors() :-

```
[139] # Function to evaluate the probabilty if the user wants to switch the door
      def switch_doors():
          switch_doors = []
          wins = 0
          losses = 0
          for n in tqdm(range(1,1000)):
              car_variable_door = np.random.randint(1,4)
              challenger_door = np.random.randint(1,4)
              host = random.choice([i for i in range(1,4) if i not in [car_variable_door,challenger_door]])
              challenger_door_changed = random.choice([i for i in range(1,4) if i not in [host,challenger_door]])

              # print()
              # print(car_variable_door,challenger_door,host)
              if challenger_door_changed == car_variable_door:
                  wins = wins + 1
              else:
                  losses = losses + 1

              win_perc = wins / (wins + losses)
              switch_doors.append(win_perc)
              # print(f"\nProbabilty of winning if the User wants to switch the door runnung {n} times:--> {win_perc}")
              # print(win_perc)
          return (win_perc),switch_doors
```

Here,

- Wins and losses are initialized with value 0.
- switch_door is an empty list to store the probability values in between the running iterations
- Number_of_iterations - iterations ranging from 1-1000 (user defined)

- Function is calculating all variable values using a for loop till the values exhausted in range function

- In each iteration,challenger_door_changed value is compared to car_variable_door and if value matches, wins value is incremented by 1
- Otherwise losses value is incremented by when as the challenger losses

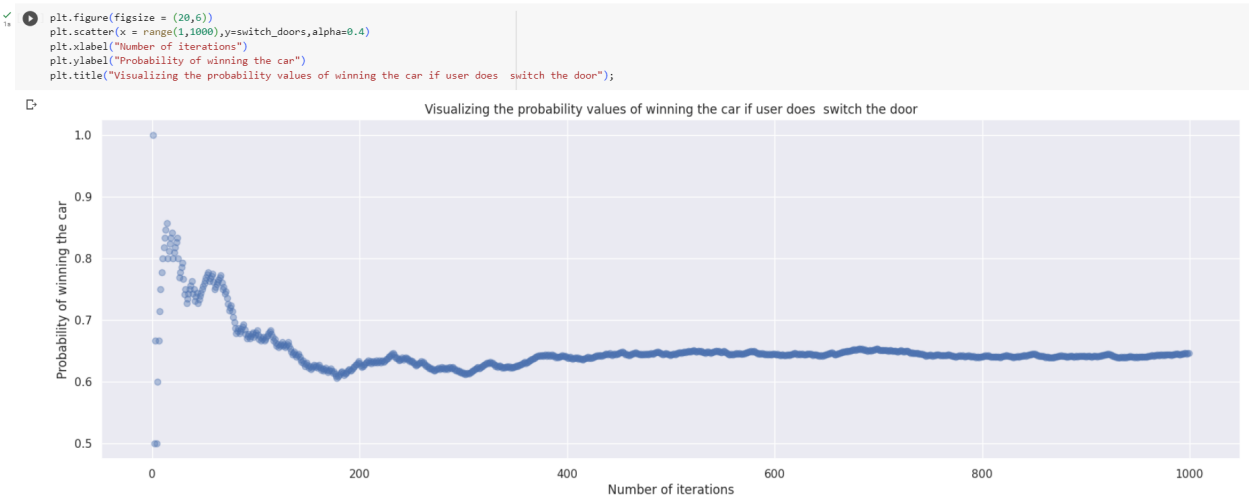
- After that win_perc is calculated using the values generated above using wins & losses
- Finally the function return the win_percent value and switch_door list after all iterations are completed

```
            # print()
            # print(car_variable_door,challenger_door,host)
            if challenger_door_changed == car_variable_door:
                wins = wins + 1
            else:
                losses = losses + 1

            win_perc = wins / (wins + losses)
            switch_doors.append(win_perc)
            # print(f"\nProbabilty of winning if the User wants to switch the door runnung {n} times:--> {win_perc}")
            # print(win_perc)
        return (win_perc),switch_doors
```

```
[140] result,switch_doors = switch_doors()
      print(f"\n\nProbabilty if the User wants to switch the door::--> {result}")
```

```
100%|██████████| 999/999 [00:00<00:00, 44183.16it/s]

Probabilty if the User wants to switch the door::--> 0.6616616616616616
```

```
[141] plt.figure(figsize = (20,6))
      plt.scatter(x = range(1,1000),y=switch_doors,alpha=0.4)
      plt.xlabel("Number of iterations")
      plt.ylabel("Probability of winning the car")
      plt.title("Visualizing the probability values of winning the car if user does  switch the door");
```

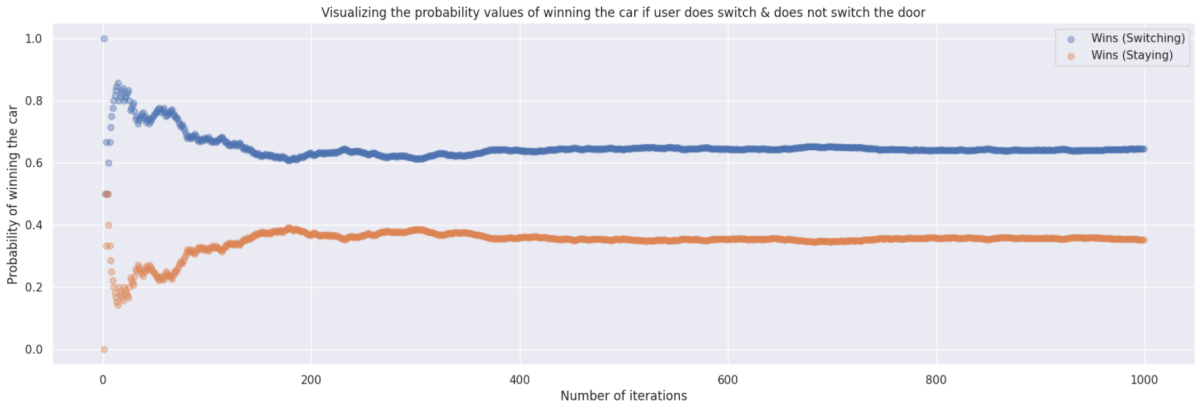


From the above iterations, we see that the probability of winning the car if the user does want to switch the door settles down to **0.6**

Conclusion

- what is the probability to get the car if user switches the door? – > **0.6**
- what is the probability to get the car if user does not switch the door? —> **0.3**

```
plt.scatter(x = range(1,1000),y=switch_doors,alpha=0.4,label = 'Wins (Switching)')
plt.scatter(x = range(1,1000),y=no_switch_doors,alpha=0.4,label = 'Wins (Staying)')
plt.legend()
plt.xlabel("Number of iterations")
plt.ylabel("Probability of winning the car")
plt.title("Visualizing the probability values of winning the car if user does switch & does not switch the door");
```



Number of iterat

```
[33] print(f"The probability to win the car if user switches the door: {result_switch_doors}")
     print(f"The probability to win the car if user does not switch the door: {result_no_switch}")

     The probability to win the car if user switches the door: 0.6466466466466466
     The probability to win the car if user does not switch the door: 0.3533533533533533
```

The Monty Hall problem – Do you want to switch the door?
From the above simulation and results obtained above,it can be said that **user do want to switch the door** as the probability of winning when switching the door is 2 times higher than the probability of winning when not switching the door

The correct answer is that **you *do* want to switch.** If you do not switch, you have the expected 1/3 chance of winning the car, since no matter whether you initially picked the correct door, Monty will show you a door with a goat. But after Monty has eliminated one of the doors for you, you obviously do not improve your chances of winning to better than 1/3 by sticking with your original choice. If you now switch doors, however, there is

a 2/3 chance you will win the car (counterintuitive though it seems).**Therefore, switching is twice as likely to get you the car as staying.**

Alternate Solutions

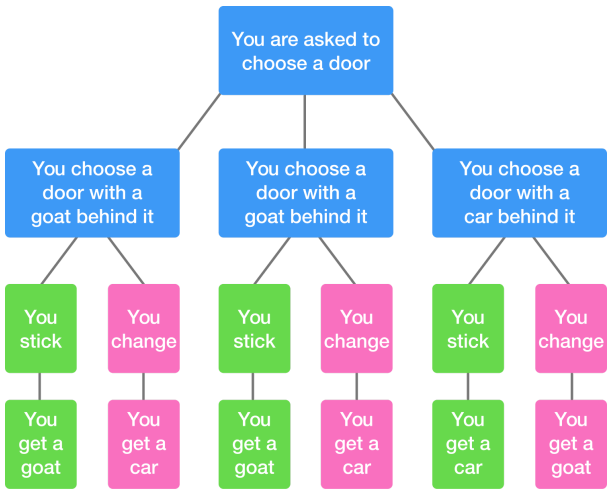
One way to see the solution is to explicitly list out all the possible outcomes, and count how often you get the car if you stay versus switch. Without loss of generality, suppose your selection was door 1

Then the possible outcomes can be seen in this table:

Behind door 1	Behind door 2	Behind door 3	Result if staying at door 1	Result if switching to the door offered
Car	Goat	Goat	**Wins car**	Wins goat
Goat	**Car**	Goat	Wins goat	**Wins car**
Goat	Goat	**Car**	Wins goat	**Wins car**

In two out of three cases, you win the car by changing your selection after one of the doors is revealed. This is because there is a greater probability that you choose a door with a goat behind it in the first go, and then Monty is *guaranteed* to reveal that one of the other doors has a goat behind it. Hence, by changing your option, you double your probability of winning.

Another way of seeing the same set of options is by drawing it out as a decision tree, as in the following image:



Using Bayes' Theorem

There are two aspects of the Monty Hall problem that many struggle to agree with. First, why aren't the odds 50-50 after the host opens the door? Why is it that switching doors has a 2 in 3 chance of winning when sticking with the first pick only has a 1 in 3 chance? Secondly, why is it the case that if Monty opened a door truly randomly and happened to show a goat, then the odds of staying vs. switching doors are now 50-50? Bayes' theorem can answer these questions.

Bayes' theorem is a formula that describes how to update the probability that a hypothesis is correct, given evidence. In this case, it's the probability that our initially picked door is the one with the car behind it (that staying is right) given that Monty has opened a door showing a goat behind it: that Monty has shown us that one of the choices we didn't pick was the wrong choice.
Let H be the hypothesis "door 1 has a car behind it,"
and
E be the evidence that Monty has revealed a door with a goat behind it.
Then the problem can be restated as calculating P (H | E) , the conditional probability of H given E

Since every door either has a car or a goat behind it, the hypothesis "not H" is the same as "door 1 has a goat behind it."

In this case, Bayes' theorem states that

P (H | E) = (P(E | H) / P (E)) * P (H)

= P(E | H) * P (H) / P(E | H) * P(H) + P (E | not H) * P (not H)

The problem as stated says that Monty Hall deliberately shows you a door that has a goat behind it

Breaking down each of the components of this equation, we have the following::

- P(H) is the prior probability that door 1 has a car behind it, without knowing about the door that Monty reveals. This is 1 / 3
- P(not H) is the probability that we did not pick the door with the car behind it. Since the door either has the car behind it or not, P(not H) = 1 - P(H) = ⅔

- P (E | H) is the probability that Monty shows a door with a goat behind it, given that there is a car behind door 1. Since Monty always shows a door with a goat, this is equal to 1
- P (E | not H) is the probability that Monty shows the goat, given that there is a goat behind door 1. Again, since Monty always shows a door with a goat, this is equal to 1

Combining all of this information gives

P (H | E) = (1 * 1 / 3) / 1 * ⅓ + 1 * 2 /3
= (1 / 3) / 1
= 1 / 3

The probability that the car is behind door 1 is completely unchanged by the evidence. However, since the car can only either be behind door 1 or behind the door Monty didn't reveal, the probability it is behind the door that is not revealed is 2 / 3

Therefore, switching is twice as likely to get you the car as staying.

References

- https://betterexplained.com/articles/understanding-the-monty-hall-problem/
- https://brilliant.org/wiki/monty-hall-problem/#:~:text=The%20Monty%20Hall%20problem%20is,to%20choose%20between%20three%20doors.
- https://statisticsbyjim.com/fun/monty-hall-problem/ ,By Jim Frost
- https://en.wikipedia.org/wiki/Monty_Hall_problem
- https://mathworld.wolfram.com/MontyHallProblem.html,Weisstein, Eric W. "Monty Hall Problem." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/MontyHallProblem.html