EXPERIMENT 9: Implementation of Uncertain Methods for an Application

**Name :** D V Ravi Sridhar

**Reg No. :** RA1811003010732

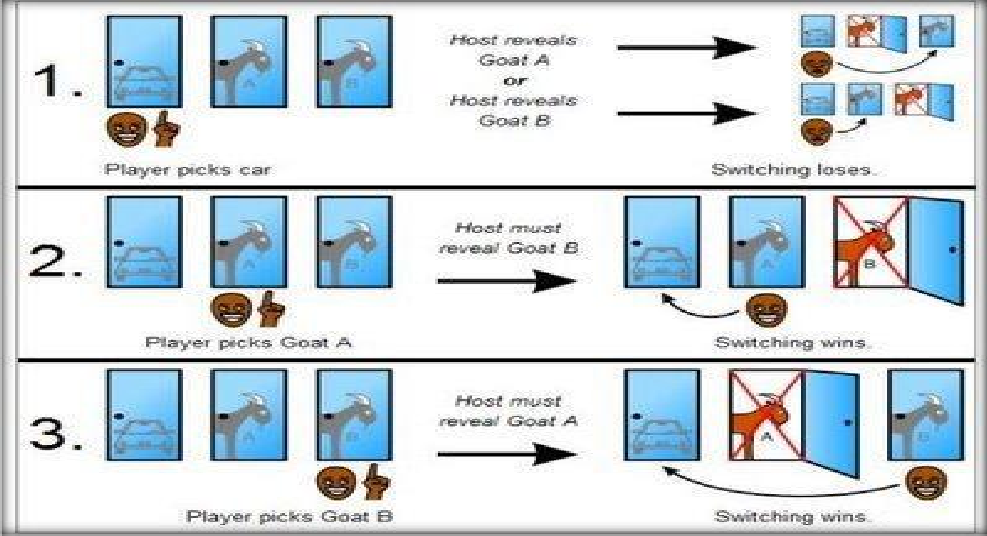
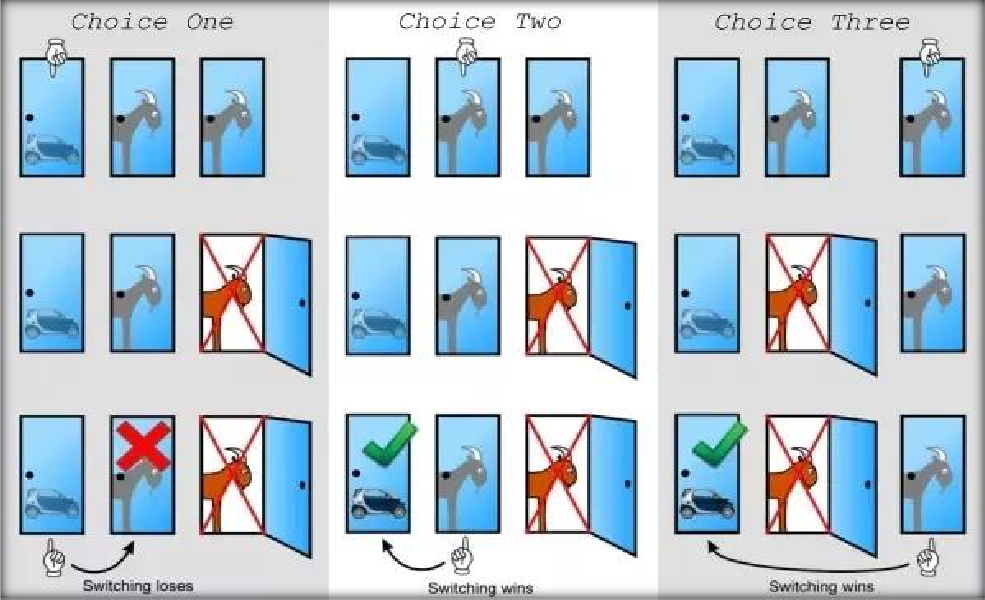
**Section :** CSE – D2

# Date : 26 /04/2021

**Problem Description :- The Monty Hall Problem**

## The Monty Hall Problem is a famous game show problem which involves a host, a contestant, three doors, 2 goats, and 1 Ferrari. The game show goes like this:

1. The contestant picks any door (1, 2, or 3). Let's say this is door 1.
2. The host then reveals what is behind one of the doors that is not selected and not surprisingly it is a goat. Let's say this is door 3. So door 1 (the contestant's original selection) and door 2 (the door that has yet to be revealed) are still in play.
3. The host then asks the contestant if they want to stay with the original guess, door 1 in our example, or switch to the remaining door, which is door 2 in our example.



**Code :-**

#Import Libraries import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

* Visualizations

%matplotlib inline sns.set\_style('darkgrid')

#Generate Multiple Environments

* containers to house 0 or 1 depending on the outcome stay = []

switch = [] doors = [1,2,3]

* we will consider 10^4 "environments" where the contestant either stays or switches environments = range(10\*\*4)

for environment in environments:

* + randomly assign the Ferrari behind door 1,2, or 3 ferrari\_door = np.random.randint(1,4)
  + random assign a guess for the contestant contestant\_guess = np.random.randint(1,4)
  + goat doors are the doors that are not hiding the Ferrari goat\_doors = [door for door in doors if door != ferrari\_door]
  + host reveals a goat. Note, if the host has 2 doors to choose from, # it is because the contestant\_guess == ferrari\_door
  + If the host has only 1 door to choose from, then contestant\_guess != ferrari\_do

or

possible\_reveal\_doors = [door for door in goat\_doors if door != contestant\_guess] if len(possible\_reveal\_doors) == 2:

reveal\_door = np.random.choice(possible\_reveal\_doors) else:

reveal\_door = possible\_reveal\_doors[0]

* Define the door that the contest would switch to # We only care about the sole element of this list

switch\_door = [door for door in doors if door != contestant\_guess and door != rev

eal\_door][0]

* Let's record the results of our model over 10^4 switching or staying outcomes if ferrari\_door == switch\_door:

switch.append(1) stay.append(0)

else:

switch.append(0) stay.append(1)

#Calculate Probability of Winning for Each Strategy # Probabilities

prob\_win\_switch = np.mean(switch) prob\_win\_stay = np.mean(stay)

print(f"Probability of Winning if Switch: {prob\_win\_switch : 0.2%}") print(f"Probability of Winning if Stay: {prob\_win\_stay : 0.2%}")

#Vizualize Our Results

* Create figure and add subplot fig = plt.figure(figsize = (10,8)); ax = fig.add\_subplot(111);
* Redefine environments and define winning % for each environments in environments environments = range(1,10\*\*4+1)

switch\_probs = [np.mean(switch[:environment]) for environment in environments] stay\_probs = [np.mean(stay[:environment]) for environment in environments]

* Generate line graphs of win probabilites for each environment ax.plot(environments, switch\_probs, label=f"Switch Avg: {prob\_win\_switch : 0.2%}"); ax.plot(environments, stay\_probs, label=f"Stay Avg: {prob\_win\_stay : 0.2%}");
* Add labels

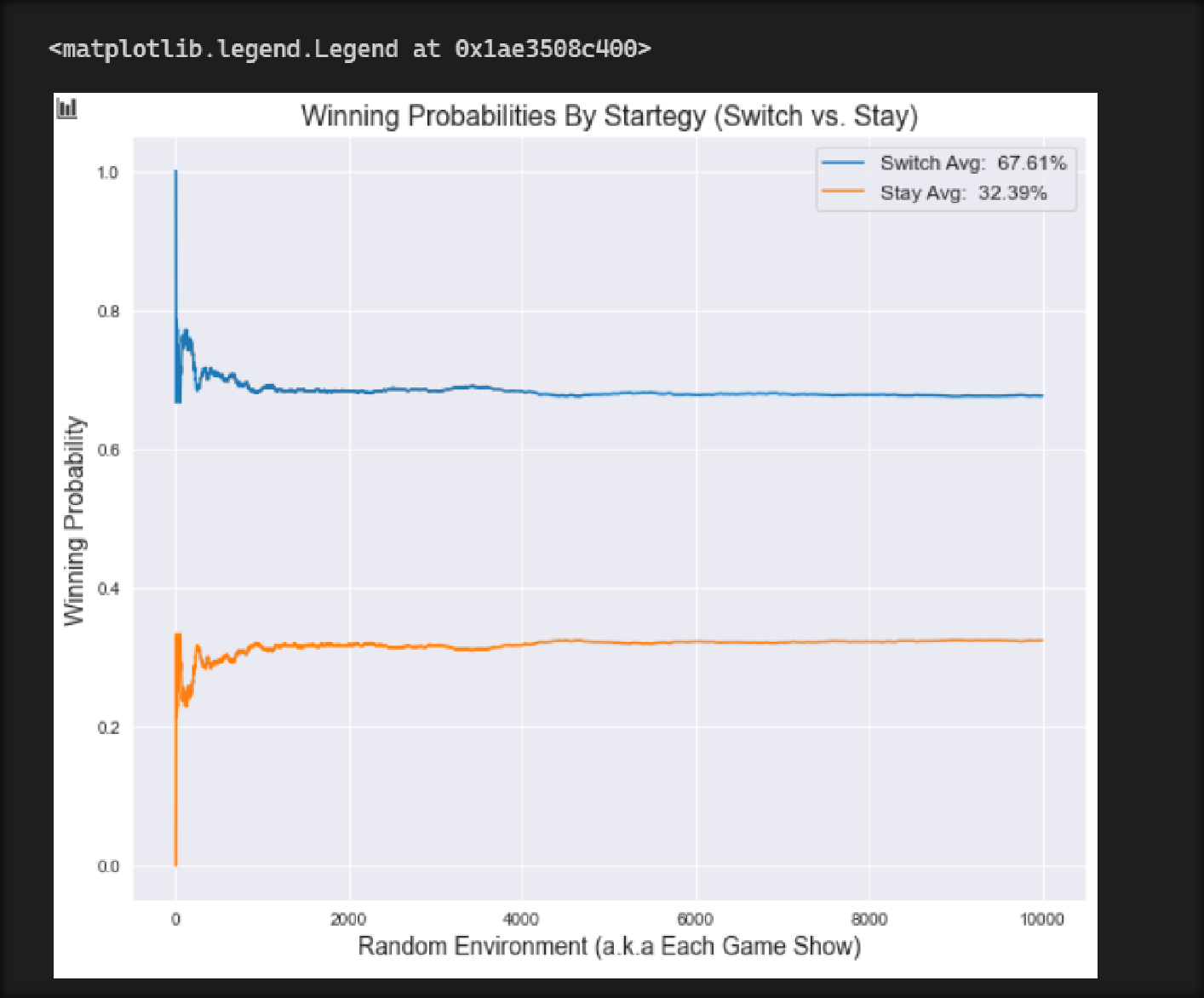
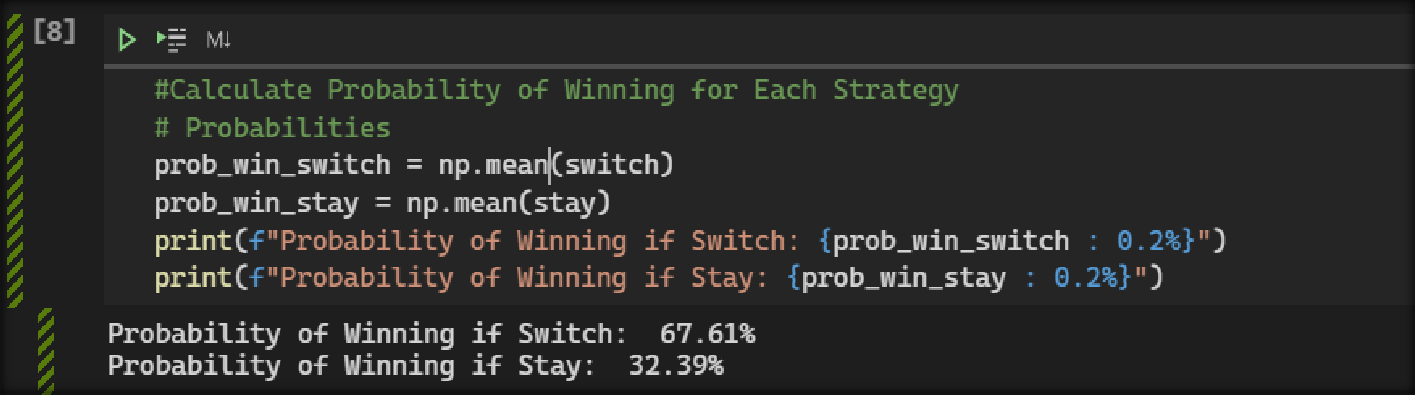
ax.set\_title("Winning Probabilities By Startegy (Switch vs. Stay)", {'fontsize' : 16

});

ax.set\_ylabel("Winning Probability", {'fontsize' : 14}); ax.set\_xlabel("Random Environment (a.k.a Each Game Show)", {'fontsize' : 14}); # Add legend

ax.legend(fontsize = 'large');

# Screenshot from Output :-



**Result**

## The experiment was successfully implemented and executed.