## 1 Monty Hall Problem

Suppose that there are three closed doors on the set of the ***Let's Make a Deal***, a TV game show presented by Monty Hall. Behind one of these doors is a car; behind the other two are goats. The contestant does not know where the car is, but Monty Hall does.

The contestant selects a door, but not the outcome is not immediately evident. Monty opens one of the remaining "wrong" doors. If the contestant has already chosen the correct door, Monty is equally likely to open either of the two remaining doors.

After Monty has shown a goat behind the door that he opens, the contestant is always given the option to switch doors. What is the probability of winning the car if she stays with her first choice? What if she decides to switch?

## 1.1 Implementation (part 1)

We have 3 doors to choose from, so we will define a vector of three door names "A","B" and "C". The command `sample(,n)` takes a sample of size n from a specified set of values. Here we just want to select one door to be our "correct door" and another to be "selected" door (i.e. the contestant selects)

These events are independent. We will perform the selection for both doors separately, but this can be implemented in one command.

```
Doors = c("A","B","C")
Correct = sample(Doors,1)
Choice = sample(Doors,1)
```

A wrong door must be selected to be opened. The door selected by the contestant can't be chosen. First let us select the doors that must stay closed, then find the ones we can choose from to open

```
StayClosed = union(Correct, Choice)
CanOpen = setdiff(Doors, StayClosed)
```

```
NotOpen = setdiff(Doors,Open)

Stick = Choice

Switch = setdiff(NotOpen,Choice)
```

```
# Was sticking the right decision? Or was switching?
# The following logicial statements  will tell us.

Stick==Choice
Switch==Choice
```

## 1.2 Writing a Function

We are going to create a function `MHfunc()` to help us simulate a solution for the Monty Hall Problem. The function is constructed using `R` code we have seen already. The output of the function is returned as a data frame, with three columns:

**Correct** : The number of the correct door.

**Choice** : The door that the contestant chose originally, and the door selected if the contestant decided to "stick".

**Switch** : The door selected if the contestant chose to "switch".

Necessarily the ***Correct*** option must correspond with a value in one of the other two columns.

```
MHfunc <- function(numdoors=3){
 Doors = LETTERS[1:numdoors]
 Correct = sample(Doors,1)
 Choice = sample(Doors,1)

 StayClosed = union(Correct, Choice)
 CanOpen = setdiff(Doors, StayClosed)
 DoesOpen = sample(CanOpen,1)

 NotOpen = setdiff(Doors,DoesOpen)
 Switch = setdiff(NotOpen,Choice)

 Stick=Choice
 MHoutput=c(Correct,Choice,Stick,Switch)
 names(MHoutput)= c("Correct","Choice","Stick","Switch")
 return(MHoutput)
}
MHfunc()
```

No arguments are needed to be passed to the function. The output should appear something like this:

```
> MHfunc()
Correct  Choice   Stick  Switch
"A"      "C"      "C"      "A"
> MHfunc()
Correct  Choice   Stick  Switch
"C"      "A"      "A"      "C"
> MHfunc()
Correct  Choice   Stick  Switch
"C"      "C"      "C"      "A"
> MHfunc()
Correct  Choice   Stick  Switch
"C"      "B"      "B"      "C"
>
```

## 1.3 Transforming logical values

The R command `as.numeric()` can be use to convert logical values, such as TRUE or FALSE into binary form (i.e. 1 and 0).

```
LogVec=c(TRUE,FALSE,TRUE)
as.numeric(LogVec)
```

```
> output = MHfunc()
> output
Correct  Choice   Stick  Switch
"B"     "B"     "B"     "A"
>
> output[1]
Correct
"B"
> output[3]
Stick
"B"
> output[4]
Switch
"A"
>
> output[1] == output[3]
Correct
TRUE
> output[1] == output[4]
Correct
FALSE
> as.numeric(output[1] == output[3])
[1] 1
> as.numeric(output[1] == output[4])
[1] 0
```

## 1.4 A Simulation Study of the Monty Hall Problem

Let us perform a simulation study of the Monty Hall problem. We are putting the code we have written already, placed in a `for` loop, although we alter the ending to make for more readable output.

```
StickCount = 0
SwitchCount = 0
M=1000

for(i in 1:M)
 {
 output=MHfunc()
 Correct=output[1]
 Stick=output[3]
 Switch=output[4]
 # Install counters for both count variables

 StickCount = StickCount + as.numeric(Stick  ==  Correct)
 SwitchCount = SwitchCount + as.numeric(Switch  ==  Correct)
}
StickCount
SwitchCount
```

The output of this program would look like this. The proportion is approximately 2 to 1 in favour of the switching option.

```
> StickCount
[1] 323
> SwitchCount
[1] 677
```