Post 02 - Simulations Using R

Stella Park

Introduction

One interesting and convenient feature in R is generating random numbers. Its diverse applications includes simulation, which allows us do predictions and calculate statistical data from the sampled values.

I thought this would be an great topic to cover since it connects the study of Statistics with the computational skills of R that we are learning in Stat 133 course.

Specifically in this post, we will first explore how to simulate the flipping a coin and the Monty Hall problem using the sample function in R. Then, we will apply the simulation method to do some statistical testing using bootstrapping.

Flipping a Coin

Let's start with something easy.

I'll start off by explaining how to flip a coin using the function sample(). Follow the code chunk below with comments to simulate a coin flip!

```
# create a coin and distinguish the sides
coin <- c('heads','tails')

# toss the coin indicating the number of tosses with the size variable
# make sure to set replace = TRUE so that we can draw more elements than the length of input    vector
# let's start by simulating 100 coin tosses
tosses <- sample(coin, size=100, replace=TRUE)

# create a table with the number of heads and tails
freqs <- table(tosses)
freqs
```

```
## tosses
## heads tails
##    51    49
```

Now a coin does not always have to be fair. You can set the probability for a certain side, head or tail, to be more or less than 0.5, just about any numbers between 0 to 1!:

```
# argument x = input vector
# argument n = size of sample
# argument prob = vector of probability that sums up to 1
toss <- function(x, n, prob){
  sample(x, size=n, replace=TRUE, prob=prob)
}
```

For instance, let's make an unfair coin that has 70% probability of getting heads and 30% probability of getting tails, and put the results in a table!

```
unfair<-toss(coin, n=30, prob=c(0.7,0.3))
table(unfair)
```

```
## unfair
## heads tails
##    20    10
```

Let's visualize the distribution of the flip of this unfair coin in a histogram! In order to get a distribution that is close to a normal, let's increase the sample size to a big number. In order to see the distribution, we will use a method called bootstrapping and iterate through 1000 trials using randomly sampled values.

```
# ceate a table simulating a coin flip with 1000 trials
unfair_coin<-toss(coin, n=1000, prob=c(0.7,0.3))

  # create an empty list to later fill with sample values
  h <- c()
  # create a for loop to iterate the simulation 1000 times
  for(i in 1:1000){
    # assing a variable to the sampled values from the unfair coin toss
    bootstrap=sample(unfair_coin, replace=TRUE)
    # simulate the probability that you will get heads
    h=c(h,length(which(bootstrap=='heads'))/1000)
    }

#display few values of list z containing the sample probabilities of getting a head
head(h)
```

```
## [1] 0.706 0.692 0.693 0.704 0.680 0.695
```

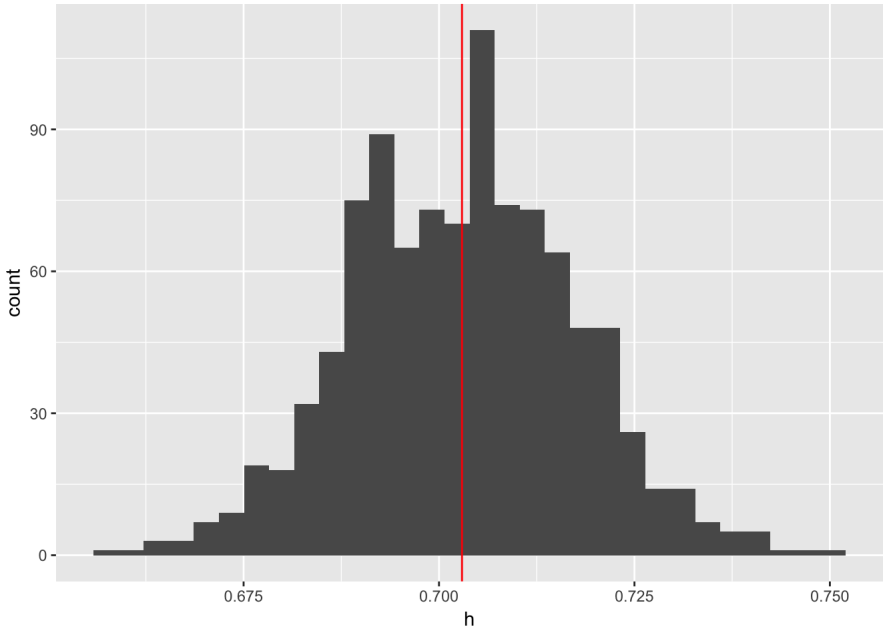
You will notice that you will get a different list of z every time you simulate, since we are getting random samples. Now let's visualize the

distribution in a histogram!

```
# first make a data frame of the coin tosses so that we can create a histogram
h_df <- data.frame(h)

# load the ggplot2 pacakge
library(ggplot2)
# create a histogram with the distribution and a red line to represent the mean
ggplot(h_df, aes(x=h)) + geom_histogram() + geom_vline(xintercept=mean(h_df$h),col=2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Notice how the mean is very close to the probability of heads that we set up in the toss() function, 0.7! So we have figured out the probability of how many heads will appear in a toss by the bootstrapping method, generating random samples and observing its distribution!

Let's do the same for the distribution of tails:

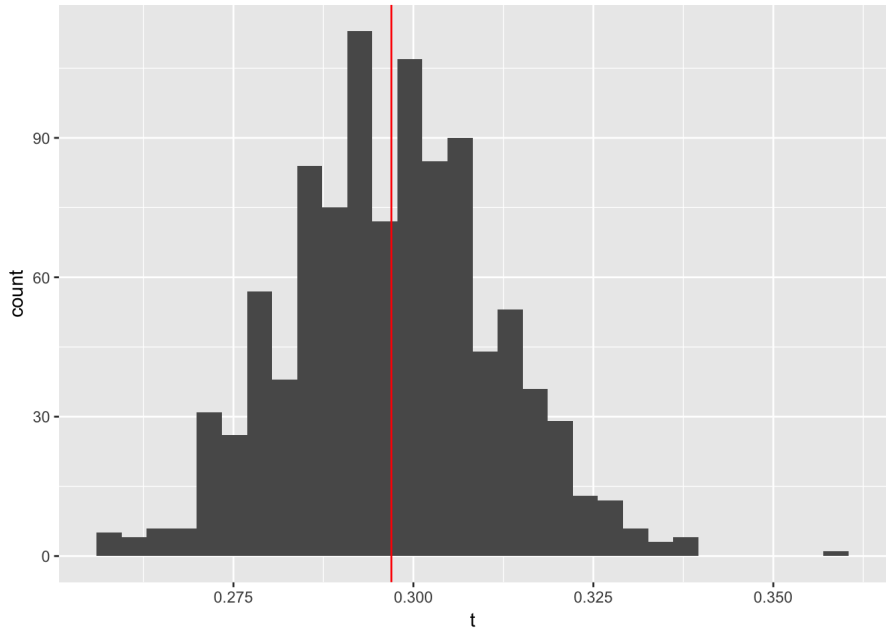
```
  t <- c()
  for(i in 1:1000){
    bootstrap=sample(unfair_coin, replace=TRUE)
    t=c(t,length(which(bootstrap=='tails'))/1000)
  }
head(t)
```

```
## [1] 0.308 0.301 0.296 0.287 0.280 0.279
```

```
t_df <- data.frame(t)

# load the ggplot2 pacakge
library(ggplot2)
# create a histogram with the distribution and a red line to represent the mean
ggplot(t_df, aes(x=t)) + geom_histogram() + geom_vline(xintercept=mean(t_df$t),col=2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The expected probability in which tails will appear is also very close to the probability of tails that we set up in the unfair toss function, 0.3!

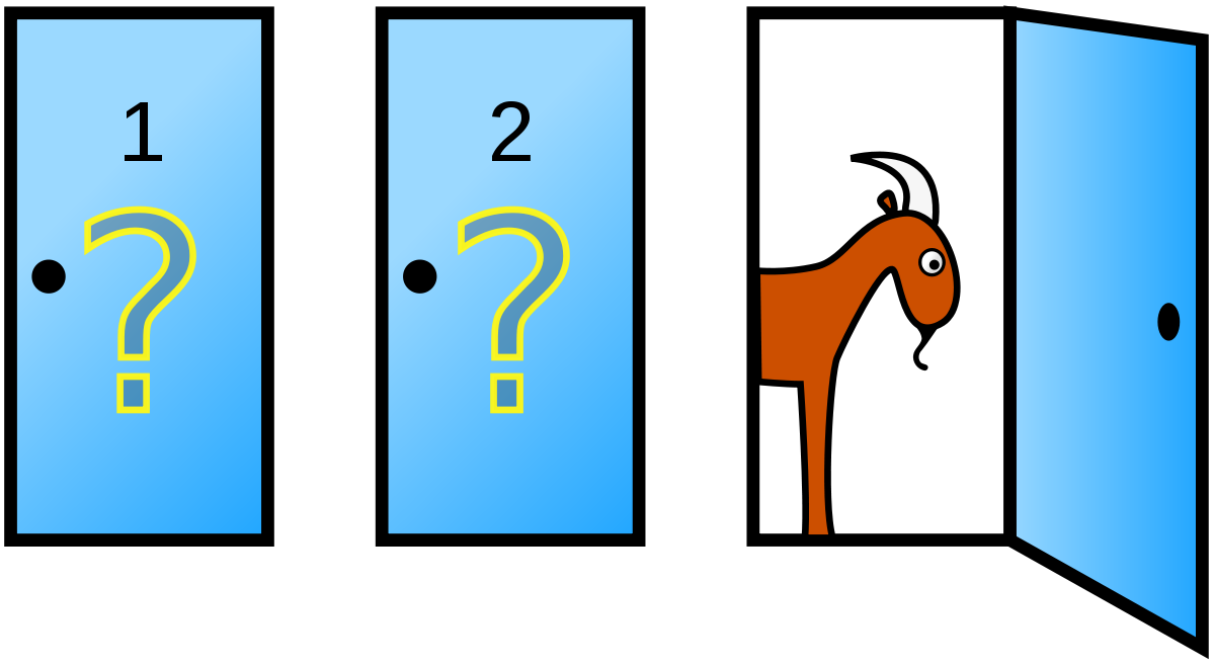
Monty Hall Problem

Now let's talk about some more complex applications of the bootstrapping method and simulations using R!

The Monty Hall problem is a brain teaser, in the form of a probability puzzle, which loosely derived from a television game show called "Let's Make a Deal". The fun part of the show was that when the contestants had the opportunity to win wonderful prizes, they might instead end up with weird prizes that were less desirable.

In the show, the contestant faces three closed doors: behind one door is a car and behind each other two doors is a goat. The contestant has to choose one door under following rules below:

- When the contestant makes his or her initial choice, that door does not open yet.
- The show host opens one of the two other doors that has a goat behind it.
- Two doors remain, including the door of the contestant's choice. One of the door has a car behind it, and the other has a oat. The contestant has to choose one of the remaining two, and is free to switch his or her initial decision.



Which door should the contestant choose if he or she wants the car? Should the contestant stick with the original choice or switch to the other door? This is the Monty Hall paradox.

The logical solution to the paradox is the following:

- Chance that the car is behind the initially chosen door is 1/3
- Since the car is behind either the initially chosen door or the remaining door, the chance that the car is behind the remaining door is 2/3
- Hence, the contestant should switch doors to win the car

Although there are multiple ways to approach this problem, we will use simulations in R!

Solving the Paradox by Simulation

We will try to simulate the experiment by a big number of times in order to best replicate the actual outcome. The crucial part of this method is to generate random samples and strive to calculate the probability of winning the car. We will calculate the probability of win for each possibility of the contestant's choices and compare them to figure out the best one. Let's perform 100 trials because 100 is a big number!

Follow the code chunk below:

```
# Set the seed to make the examples replicable
set.seed(1)

# Make a vector with different values to distinguish the three doors
doors <- c('A','B','C')

# Create an empty list to later fill with randomly generated samples
x <- c()

# Create a function with a for loop to simulate the problem 1,000 times
# Argument n = number of trials, an integer value
# Argument descript = logic whether to display the description of statistics
monty <- function(n,descript=TRUE){
  for(i in 1:n){
  # assign a variable to the door that has a car behind it
    car <- sample(doors)[1]
    # assign a variable to the door that the contestant chooses
    pick <- sample(doors)[1]
    # assign a variable to the door that has a goat behind it among the two doors that are not     chosen by the c
ontestant
    open <- sample(doors[which(doors != pick & doors != car)])[1]
    # assign a variable to the door that the contestant may pick if he decides to switch
    switch <- doors[which(doors != pick & doors != open)]

    if (pick == car){
      x = c(x,'same')       # add 'same' to list x if car is behind the initially picked door
    }
    if (switch == car){
      x=c(x,'switched')     # add 'switched' to x if car is behind the switched door
    }
  }
    # Display results if descript = TRUE
  if(descript==TRUE){
    a <- '\n *Not Switched*\n'
    b <- '\n Wins = '
    c <- '\n Number of Trials'
    d <- '\n Observed Winning Probability = '
    e <- '\n Theoretical Winning Probability = '
    f <- '\n\n *Switched*\n'

    cat(a,
        b, length(which(x=='same')),
        c, n,
        d, length(which(x=='same')) / n,
        e, 1/3,
        f,
        b, length(which(x=='switched')),
        c, n,
        d, length(which(x=='switched')) / n,
        e, 2/3
        )
    }

    # Only return the modified list x if descript = FALSE
  if(descript==FALSE){
    return(x)
  }
}
```

```
# Describe the simulated Monty Hall problem with 100 trials
monty(100)
```

```
##
##  *Not Switched*
##
##  Wins =  34
##  Number of Trials 100
##  Observed Winning Probability =  0.34
##  Theoretical Winning Probability =  0.3333333
##
##  *Switched*
##
##  Wins =  66
##  Number of Trials 100
##  Observed Winning Probability =  0.66
##  Theoretical Winning Probability =  0.6666667
```

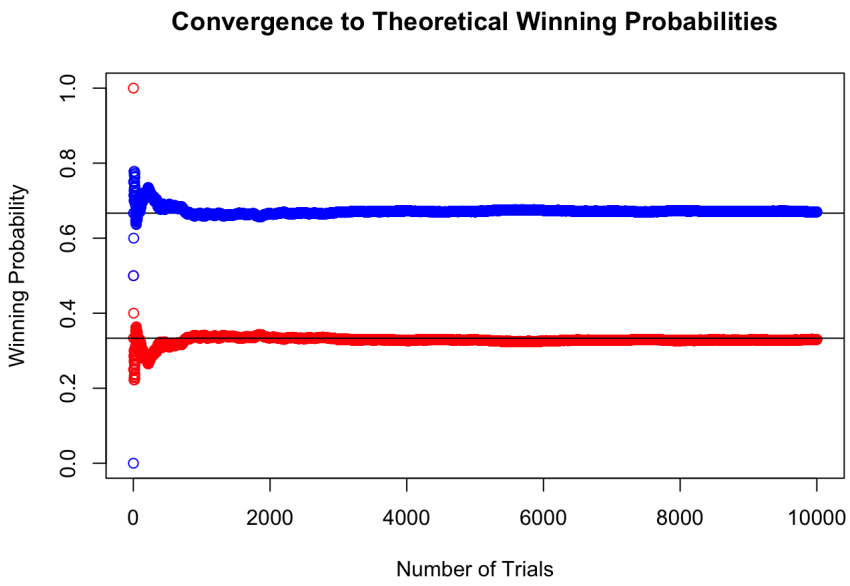
Now let's see what happens if we increase the number of trials to 10,000!

```
set.seed(4)
monty(10000)
```

```
##
##  *Not Switched*
##
##  Wins =  3381
##  Number of Trials 10000
##  Observed Winning Probability =  0.3381
##  Theoretical Winning Probability =  0.3333333
##
##  *Switched*
##
##  Wins =  6619
##  Number of Trials 10000
##  Observed Winning Probability =  0.6619
##  Theoretical Winning Probability =  0.6666667
```

Analyzing and Visualizing the Results

Observe that the observed winning probability gets closer to the theoretical winning probability as the number of trials becomes larger. You may delete the seed and try multiple trials with different randomly generated samples to test more results. The chart below illustrates how the winning probability converges as the number of trials becomes larger:



Now let's observe the distribution of winning probabilities by generating a histogram! First, we will compose a function that will make a bootstrap of creating random samples and make it into an array.

```
# Argument n = number of trials
# Argument value = either same or switched door
distribution <- function(n,value){
  # create an empty list to store sample probability
  y <- c()
  # create a for loop to iterate the simulation n times
  for(i in 1:n){
    # assign a variable bootstrap to store samples of winning proability
    bootstrap=sample(monty(n,FALSE),replace=TRUE)
    # append bootstrap values that match the value argument
    y=c(y,length(which(bootstrap==value))/n)
  }
  # return the new appended array y
  return(y)
}
```

Let's now create a data frame dat to store the bootstrapped values into a table distinguished by columns of the same door or switched doors.

```
dat <- data.frame(same=distribution(300,'same'),
                  switched=distribution(300,'switched'))
```

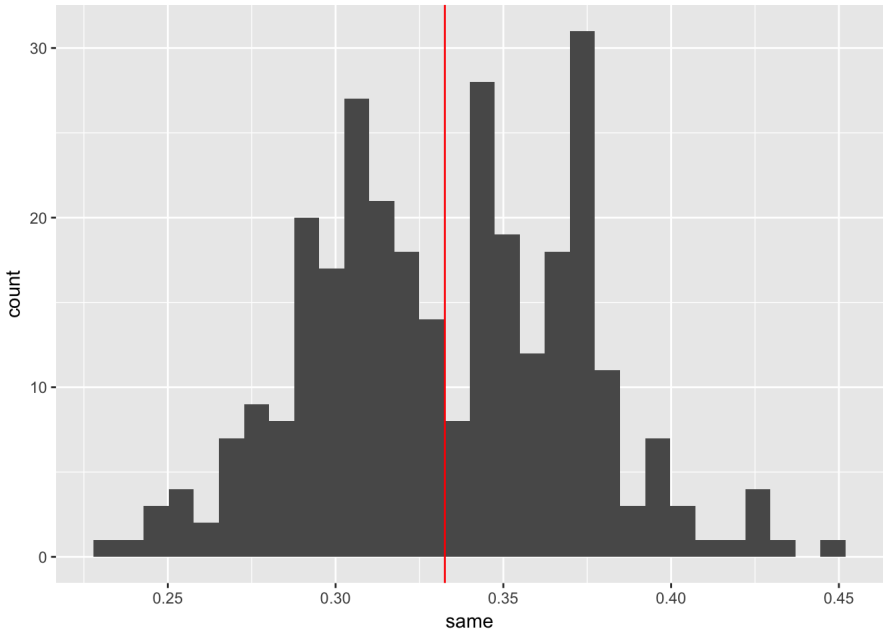
Let's visualize the distribution in a histogram! The red line represents the mean value of winning probability.

- Distribution of winning probability if contestant sticks to his or her initial pick

```r
# Load the ggplot2 package
library(ggplot2)

# Create a histogram of the distribution and include a vertical line for the mean
ggplot(dat, aes(x=same)) + geom_histogram() + geom_vline(xintercept=mean(dat$same),col=2)
```

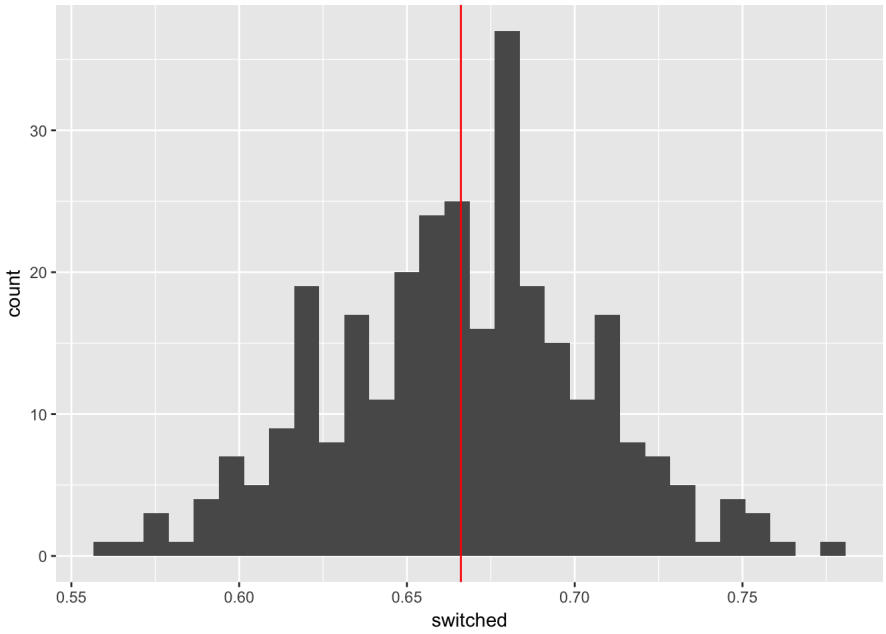
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



- Distribution of winning probability if contestant switches his or her initial pick

```r
ggplot(dat, aes(x=switched)) + geom_histogram() + geom_vline(xintercept=mean(dat$switched),col=2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Since we are getting random samples, you will notice that you will get a different distribution table every time you run the plot. Observe that the mean, indicated by the red line, is close to the theoretical winning probability. The histogram will be more normally distributed as the number of trials increases.

Tentative Solution

Note that according to simulation or random samples, the probability of winning a car if you switch the doors is greater by almost 30%! So we have proved that the theory that switching is better!

Conclusion

That's a wrap! I hope this topic makes you connect the statistical knowledge, such as probability and sampling, with the computational skills that we have learned in Stat 133. Random sampling is a great function to visualize distribution, simulate, and make predictions.

This method can also be used with data from outside sources, such as csv files, and bootstrapped using sample() function. Check out the source URLs below to see some cool applications of random sampling using R!

Sources

- https://en.wikipedia.org/wiki/Monty_Hall_problem
- https://www.inferentialthinking.com/chapters/08/3/monty-hall-problem.html
- https://www.math.utah.edu/~treiberg/M3074MontyEg.pdf
- https://www.youtube.com/watch?v=tvv4IA8PEzw
- http://www.esg.montana.edu/R/rsim.pdf
- http://rstudio-pubs-static.s3.amazonaws.com/8492_b817c712a5f6456fb4c5932e3d957135.html#/1
- https://stats.idre.ucla.edu/r/library/r-library-introduction-to-bootstrapping/