Packet 3: Simulations

Student Learning Outcomes:

- Download and install RStudio on a computer.
- Import Data into RStudio.
- Utilize R commands that manipulate values within matrices.
- Conduct Simulations to find solutions to probability problems.

Download R and RStudio

R

R is a statistical programming language that we will be using in this course. It is free, open source, and very popular in the statistics and statistical computing world. To download R go to:

https://www.r-project.org/

You will need to select a CRAN (Comprehensive R Archive Network) from which to download the software. Any one will be fine, but I usually select one that is close to my current location. You will then need to select the appropriate download for your operating system and follow instructions on the webpage from there.

RStudio

RStudio is a popular GUI (Graphical User Interface) for R. There are many versions of RStudio available. We will utilize the free *desktop* version:

https://rstudio.com/

As with R, you will need to follow the instructions on the webpage to find the appropriate version for your operating system.

- _____ The area in RStudio where you can enter code that will be immediately run. This does not save your work.
- _____ A document that you can save that contains your code for a project.
- _____ The area of RStudio where you can load data and see what is inside your variables and data frames.

Introduction to R

We will start to understand commands in R by importing and working with a data set. The Import Data set button can be found in the _____. Download the majors.txt data set from Sakai and import it into RStudio. If you haven't already, you should open a new script and save it as packet3Notes.

Understanding the data frame

Currently your data is in the form of a matrix.

- Specific rows of your matrix can be identified by typing

- Specific columns of your matrix by identified by typing

- Specific data points in your matrix can be identified by typing

If your columns are named, you can also refer to them using _____.

Example:

We can combine both atomic vectors and the matrix sub setting abilities to obtain values:

Example:

Commenting your code

At this point you should start commenting your script. That way when you come back to your script in the future, you know what you were intending to do with each line of code. To add a comment to your code type a # in front of your comment. For example type into your code:

```
#To find all of the values in the first row of a data frame we can use
#the code dataframe[1,]
```

Note that if your comment goes onto the next line (or you choose for it to go onto the next line) then you need to use a new # symbol each time. R will read in but not execute any code that starts with '#'.

Changing data within a matrix

Currently the values in the sex column of the majors data frame take on values 1 and 2. Suppose that 1 stands for males and 2 stands for female. Our next goal is to change these from 1s and 2s to Male and Female. Type the following code into your R script and make sure to comment it so you know what you are doing later:

```
for(i in 1:length(majors$sex))(
  if (majors$sex[i] == 1)
    (majors$sex[i] <- "Male"))
```

Describe what this code is doing.

Create a second `for-loop` in your R script. This loop should change the value of 2 in the sex vector to Female.

Note: For those familiar with `if/else` statements, we could have also done that to change the values to Male and Female in a single loop.

Logical and Mathematical Operators in R

The following is a list of logical operators that you might use in R.

- **!** - This logical operator means *not*.
- **&** - This logical operator means *and* for a vector. You should use this operator when you want to perform a logical operation on every item in a vector.
- **&&** - This logical operator means *and*. You should use this operator when you want to perform a logical operation on the *first* item in a vector or a value.
- **|** - This logical operator means *or* for a vector. You should use this operator when you want to perform a logical operation on every item in a vector.
- **||** - This logical operator means *or*. You should use this operator when you want to perform a logical operation on the *first* item in a vector or a value.
- **is.** - This logical operator asks a question. For example is.na() determines if the data is NA or not.

Some mathematical operators you can use include:

- >= (greater than or equal to)
- <= (less than or equal to)
- == (equal to)

Example: What does the code `majors$satm == majors$satv` give you?

Example: What does the code `sum(majors$satm == majors$satv)` give you?

Example: What does the code `sum(majors$satm > 500 & majors$satv >= 500)` give you?

Example: What does the code `sum(!is.na(majors$sex))` give you?

Creating Data frames

We can create our own data frames using R. We will start with a matrix.

```
values <- c(rep("heart",13),rep("diamond", 13), rep("spade",13), rep("club",13))
cards <- matrix(values, 13, 4)
colnames(cards) <- c("suit1", "suit2", "suit3", "suit4")
```

Describe what this code is doing.

```r
set.seed(1234)
hand <- sample(cards, size = 5,replace = F)
hand
```

```
## [1] "spade"   "diamond" "diamond" "spade"   "club"
```

```r
table(hand)
```

```
## hand
##    club diamond   spade
##       1       2       2
```

Describe what this code is doing.

Simulations

A simulation is an _____ of the operation of a process or system that represents its operation over time. A general framework for a simulation may include:

1. Create the simulation process for a single iteration of the experiment.
2. Extend the simulation process to include multiple iterations of the experiment.
3. Use your results to solve problems.

Probability of flush in 5 card hand

In packet 1, we found two ways to determine the probability of a 5 card flush. We will now simulate drawing many five card hands to estimate the probability of a five card flush.

Based on the example we just did, how can we determine if our five card hand will be a flush?

If you set the seed as I did, then your current hand will not be a flush since the length of this table is not 1. So far our running total is 0 flushes out of 1 hand. What if we were to simulate this again?

```r
hand <- sample(cards, size = 5,replace = F)
hand
```

```
## [1] "club"    "heart"   "heart"   "spade"   "diamond"
```

```r
length(table(hand))
```

```
## [1] 4
```

Still not a flush! Now we have 0 flushes out of two hands. This process is tedious and we do not want to do it manually, so we need to set up a loop. Within the loop we will determine how long the length of the table is. *if* it is 1, then we will update our flush count by 1.

```r
#iters represents the number of times we will run the loop.
#Set this value small to start to make sure your loop is working.
#You can increse it to a large value later.
iters <- 100

#To start we have 0 flushes.  We will use this to keep track of
#how many flushes we have drawn.
flush <- 0

for(i in 1:iters){
  hand <- sample(cards,5,replace = F)
  if(length(table(hand))==1)
    flush <- flush +1
}

#After we have finished with the loop we will calculate the
#proprotion of total hands (iters) that were flushes.
flush/iters
```

```
## [1] 0.01
```

This tells us that 1 out of our 100 iterations came out to be a flush. Run the simulation again on your computer. Set the seed to a different value and increase the `iters` value to 10000. What probability did you get? Was it close to the value we found in packet 1?

Play a game twice. Record either WW, WL, LW or LL

Example from Packet 1: Suppose that when you play a video game for the first time, you have a 0.3 probability of winning. Your probability of winning is higher when you play for a second time, but how much higher depends on your first result. If you win on the first game, then your probability of winning on the second game increases to 0.8. But if you lose on the first game, then your probability of winning on the second game increases only to 0.4.

We would like to simulate the wins and losses for this scenario and use the information to determine:

- The probability of WW, WL, LW and LL
- The probability of losing on game 2

We will start by simulating the outcome of the first game. This is a _____ with one trial. The probability of success is 0.3.

```
set.seed(1234)
win1 <- 0.3
outcome1 <- rbinom(1,1,win1)
outcome1
```

```
## [1] 0
```

Note that the `rbinom` function does not output success and failure. It will output 1's and 0's. This person did not win in their first game. This means the probability of success for the second game will be 0.4. We can simulate how this person will do on the second game.

```
win2 <- 0.4
outcome2 <- rbinom(1,1,win2)
outcome2
```

```
## [1] 1
```

This person won on their second game!

What we just did won't work in a simulation, since we set the probability of win2 to be 0.4. We will need to make this *dynamic* so that if the person wins the first game, the value of win2 will be 0.8. If not, then it will be set to 0.4 as we did here.

```
set.seed(1234)
win1 <- 0.3
outcome1 <-rbinom(1,1,win1)
if(outcome1 == 1) {
  win2 <- 0.8
 }else{
  win2 <- 0.4 }
outcome2 <- rbinom(1,1,win2)
```

In our first example, we kept a running total of the number of hands that were flushes. Now we want to keep track of wins and losses from both hands in conjunction with each other. This means we need to create a matrix in which to store all of these values. We should include a column for the first round wins and a column for the second round wins. The length of the matrix should be as long as the number of iterations we intend to run.

```
iters <- 10000
gameMatrix <- matrix(NA,iters,2)
colnames(gameMatrix) <- c("Game1", "Game2")
```

Now we need to put our single set of games simulation into a loop, run it multiple times and keep track of the values we get. Putting it all together we get:

```
iters <- 10000
gameMatrix <- matrix(NA,iters,2)
colnames(gameMatrix) <- c("Game1", "Game2")

set.seed(1234)
win1 <- 0.3

for(i in 1:iters) {
 gameMatrix[i,1] <- rbinom(1,1,win1)

 if(gameMatrix[i,1] == 1){
    win2 <- 0.8
 } else {
    win2 <- 0.4 }

  gameMatrix[i,2] <- rbinom(1,1,win2)
}
```

Our first problem was to determine the probabilities of WW, WL, LW and LL. We can do that by creating a two way table of the information from our gameMatrix. Note that in the order it is given, the values on the left of the table will be the outcome of the first game and the values on the top of the table will be the outcome from the second game.

```
table(gameMatrix[,1], gameMatrix[,2])/iters
```

```
##
##          0      1
##   0 0.4156 0.2856
##   1 0.0577 0.2411
```

What is the probability of winning in the second game?

```
table(gameMatrix[,2])/iters
```

```
##
##      0      1
## 0.4733 0.5267
```

Law of Large Numbers

In calculus, you saw that, under certain conditions, sequences would converge to a value. In probability, we also have convergence theorems.

Weak Law of Large Numbers

Formal Definition:

Let $X_1, X_2, X_3, ...$ be a sequence of independent and identically distributed random variables with $\mu = E(X_i)$ and $\sigma^2 = var(X_i) < \infty, i = 1, 2, ...$ Then $\forall \epsilon > 0$,

$$lim_{n\to\infty} P(|\frac{X_1+X_2+...+X_n}{n} - \mu| > \epsilon) = 0$$

Informal Definition:

Let $X_1, X_2, X_3, ...$ be a sequence of independent and identically distributed random variables with $\mu = E(X_i)$ and a finite variance. If we were take the average value of that sequence and subtract the theoretical mean, the chances that the difference between the two values will be greater than some set tolerance (ϵ), go to 0 as our sample gets larger.

This type of convergence is called convergence in _____.

Strong Law of Large Numbers

Formal Definition:

Let $X_1, X_2, X_3, ...$ be a sequence of independent and identically distributed random variables with $\mu = E(X_i)$. Then

$$P(lim_{n\to\infty}\bar{X}_n = \mu) = 1$$

Informal Definition:

As the number of trials goes to infinity, the average of the observations converges to the expected value, with probability 1.

This type of convergence is called convergence _____.

Comparison of the Weak and Strong Law of Large Numbers

- Convergence in probability leaves open the possibility that the sample mean and theoretical mean being greater than epsilon is possible, and could happen an infinite number of times, although at infrequent intervals.
- Convergence almost surely will not allow this to happen.
- Convergence almost surely implies convergence in probability, however the converse is not true.

Takeaway

- When we simulate values, we are creating a sequence of random variables that follow the same distribution. They will have the same expected value and variance.
- Because of the law of large numbers (either weak or strong) we know that our simulation estimates for the expected value will be approximately equal to the theoretical values, if our number of simulation iterations is large enough.

Duffels

Example From Packet 2: The biology club weekend outing has two groups. One with 7 people will camp at Diamond Lake. The other group with 10 people will camp at Arapahoe Pass. Seventeen duffels were prepared by the outing committee, but 6 of these had the tents accidentally left out of the duffel. The group going to Diamond Lake picked up their duffels at random from the collection and started off down the trail. The group going to Arapahoe Pass used the remaining duffels.

We would like to simulate this scenario and count the number of duffels that ended up at Diamond Lake. Use the following steps to do this.

Make sure the process works for one sample

1. Create a variable called duffel that contains 11 entries that represent duffels with tents, and 6 entries that represent duffels without tents.
2. Take a sample of duffels that represent those that will be taken to Diamond lake.
3. Count how many duffels at Diamond Lake are without tents.

Extend the process to many samples

4. Create a matrix that will store the count of the number of duffels that are without tents. This matrix should be as long as the number of iterations you intend to run.
5. Create a loop that will repeat steps 2 and 3 of this process for as many iterations you set. Make sure to store the values from step 3 in the matrix you created in step 4.

Solve Problems with Simulated Data

6. Use the stored values to find the mean and standard deviation for the number of duffels that go to Diamond Lake without tents. You can use the `mean()` and `sd()` functions to do this.
7. Create a table of the stored values to determine the probability mass function, for the number of duffels that will arrive at Diamond Lake without duffels.

Blood Donation

Example From Packet 2: Blood type A occurs in about 41% of the population. A clinic needs 4 pints of type A blood. Each donor gives a pint of blood each time they donate. Let X be a random variable representing the number of donors needed to provide 4 pints of type A blood.

- Since there is no definitive stopping point for the number of donors we may need, we can't use a `for-loop` on this problem. Instead, we will need a _____.
- With this type of loop, R will run until it reaches a particular stopping criteria. These loops are highly susceptible to becoming _____, or loops that never stop. Make sure to update your stopping criteria within the loop! For us, that will be the number of blood donors that are of blood type A.

Example

```
countA = 0
while(countA <4) {
  countA
  countA = countA + 1
}
```

Describe what this code is doing:

We will now try to solve this problem using a simulation.

Make sure the process works for one sample

1. Create a variable that will keep track of the total number of blood donors in an experiment. Also create a variable that keeps track of the number of type A blood donors.
2. Create a `while-loop` that will run until the count for the number of type A blood donors reaches 4. Inside that loop you should:
    a. Run a binomial experiment to see if a donor is type A blood or not.
    b. If they are type A, update your count for the number of type A donors.
    c. Whether they are type A or not, update your total count of donors.

Extend the process to many samples

3. Create a matrix that keeps track of all of the iterations you plan to run. We will store the total number of donors in one column of this matrix.
4. Create a `for-loop` that runs for as many iterations as you intended. Within the loop make sure that you reset the total count, reset the count for type A donors, and run your `while-loop`. Make sure to store your total count in your matrix.

Solve Problems with Simulated Data

5. Used your stored values to find the mean and standard deviation for the total number of donors. Do these match the theoretical values for this distribution?
6. Create a table of the stored values to determine the probability mass function for the number of donors needed to obtain 4 type A donors.

Monty Hall

Monty Hall was the host of a game show called "Lets Make a Deal". In one of the games, Monty would give the contestant the choice of 3 doors. Behind one of them was a great prize, like a brand new car. Behind the other two doors were booby prizes, like a goat.

The contestant would pick a door. Monty would then walk to one of the other two doors and always reveal a goat, essentially leaving the contestant with two doors, one that had a car and one that had a goat.

Monty then offered the contestant a "deal". They could either stay with their original door or switch to the remaining door. Your goal is to simulate the contestant's probability of winning a car if they stay with the door they originally selected or switch to the other door.

Hint: This is a Bayes' Theorem Problem. You can find the theoretical probability solution to see if your simulated solution is correct.