**Lab Assignment 03**

**I. Objectives**: To explore how R and RStudio can be used to perform probability calculations. Students will be introduced to simulations and will install and load a package.

**II. Datasets**: none

**III. Packages**: “mosaic”

**IV. Preparation**

Open RStudio on a lab terminal by double-clicking the icon or selecting RStudio from the Start menu in Windows

**V. Instructions for Lab 3**

When students are being introduced to probability theory, they often see examples of tossing coins, rolling dice, and drawing balls from urns. These are random processes, but it is important to know that R uses a pseudo-random number generator: a list of billions of random numbers is programmed into R, and you choose where in the list to begin. If you wished to begin at the one-millionth entry, you would type > set.seed(1000000)

1. Begin by simulating a coin toss. Type the following three lines into the **Console** window in RStudio: > k <- c(0, 1)

> p <- c(.5, .5)

> x <- sample(k, size=1, prob=p, replace=T)

The first line indicates the values to be sampled, which are 0 (tails) and 1 (heads). The second line indicates the probabilities. The third line uses the built-in sample command, which tells R to draw one number using the probabilities specified by the second line; it also tells R to replace the value previously drawn (in case you repeat the code). If you look in the **Environment** tab, you should see a value for x which is either 0 or 1.

Click next to the cursor in the **Console** window, then use the up arrow once. R will place the last line of code that you typed after the cursor. Hit Enter to get a new draw; if it is the same number, repeat the process (using the up arrow and Enter) until the value of x changes.

2. Now we will simulate a sequence of coin tosses, and save the results. First let us set the number of trials at 200 > n <- seq(1, 200, length = 200)

The above line of code will create a data frame containing just one variable for the trial number. Next let us create empty vectors for the total number of heads and for the cumulative proportion of heads > heads <- numeric(length(n))

> cum <- numeric(length(n))

Now we get to the meat and bones of simulating, which is to create a loop. This will require three lines of code: > for(i in 1:length(n)) {

> heads[i] <- sum(sample(k, size=1, prob=p, replace=T) == 1)

> cum[i] <- sum(heads[1:i])/i }

The first line tells R to repeat the process n times (earlier, we set n to equal 200). The second line tells R to sample a 0 (tails) or 1 (heads) according to the probabilities specified in p, and to save that value as the ith entry for the variable heads. The third line divides the total number of heads so far (summing from 1 to i) by the number of coin tosses so far (equal to i) to get the proportion of heads so far.

3. If you wish to inspect the data, you should create a data frame out of these data. One way is to bind the columns together: > sim <- cbind(n, heads, cum)

What you can see is that every time a head is drawn, shown by heads equaling 0, cum increases. The value of cum should approach 50% pretty rapidly as the number of draws increases. In fact, you can observe this by typing > plot(n, cum, type="l")

4. You might be wondering if there is an automated way to sample coin tosses. One package, named *mosaic*, contains a coin-toss simulator. In the **Packages** window, type mosaic in the search bar, then click the box next to mosaic. Alternatively, type the following commands:

> install.packages(mosaic)

> library(mosaic)

Once mosaic is loaded, type: > rflip(200)

This will flip 200 fair coins for you and calculate the proportion of heads.

5. You will learn more about simulation in a few weeks. Here is a taste of the process, still using the tools in the *mosaic* package. Suppose you wanted to flip 6 coins repeatedly, say 1000 times. This would be time consuming in reality, but can occur instantaneously in R. With two lines of code R will simulate 1000 tosses of 6 coins, and then will show you the distribution of numbers of heads (from 0 to 6) > coins <- do(1000)\*rflip(6)

> tally(~heads, data = coins)

6. Suppose you were interested in rolling dice; you would need to define die and p as follows: > die <- 1:6

> p <- c(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)

> x <- sample(die, size=1, prob=p, replace=T)

You can use the up arrow and Enter key to repeatedly sample. Or, you can roll multiple dice by increasing the number following “size=”.

Let’s compare the results from tossing six coins to rolling a one die. Simulate 1000 dice rolls:

> dice <- do(1000)\*sample(die, size=1, prob=p, replace=T)

Now examine the distribution of the dice rolls (from 1 to 6):

> tally(~dice, data = dice)

7. Flipping coins and choosing balls from urns are both examples of Bernoulli experiments, from which one derives the binomial distribution. R contains built-in binomial distribution calculators.

For the density function, use the dbinom command. Suppose we are tossing six fair coins; then the probability of exactly three heads is > dbinom(3, 6, .5)

The probability of *at most* three heads can be calculated by running the above code four times, using 0, 1, 2, and 3 as the first entry inside the parentheses. On your own, do this, and add up the results: *pr*(*x*=0 | *N*=6, *p*=.5) + *pr*(*x*=1 | *N*=6, *p*=.5) + *pr*(*x*=2 | *N*=6, *p*=.5) + *pr*(*x*=0 | *N*=6, *p*=.5)

= \_\_\_\_\_\_ + \_\_\_\_\_\_ + \_\_\_\_\_\_ + \_\_\_\_\_\_

= \_\_\_\_\_\_

Alternatively, you can use the cumulative distribution to find the probability of at most three heads, which equals: > pbinom(3, 6, .5)

= \_\_\_\_\_\_

8. One of the two key building blocks of the binomial distribution is the number of *combinations*. Both combinations and *permutations* rely on the factorial. In R, the command for a factorial is prod, as in > prod(6:1)

The above command tells you 6! = 1 \* 2 \* 3 \* 4 \* 5 \* 6 = \_\_\_\_\_\_.

Suppose you are at Baskin Robbins, which advertises 31 flavors of ice cream; suppose you are buying a triple-scoop cone, so that you are choosing three different flavors to combine. Then you have 31 options for your first scoop, 30 options for your second scoop, and 29 options for the third scoop. This can be computed: > prod(31:29)

Or, more generally, you can type: > prod(31:1)/prod(28:1)

Check that these two commands give you the same number of permutations = \_\_\_\_\_\_.

9. The Birthday Problem is an example of probability at work. If there are two unrelated people (i.e., not twins), the probability that they have the same birthday is less than 0.3%. Suppose you are watching a football game, with 22 people on the pitch/field at the same time. Write code to allow you to find the probability that two players share a birthday.

10. The difference between permutations and combinations is that they are un-ordered. The answer to 8 will tell you the number of different permutations of three flavors of ice cream, however, is there *really* a difference between a chocolate-strawberry-vanilla and vanilla-chocolate-strawberry? There are six ways to arrange three distinct flavors, so the number of combinations must be smaller than the number of permutations. The equation for permutations is *n*! ÷ (*n*–*r*)! when making *r* choices from *n* options; the corresponding equation for combinations is *n*! ÷ *r*!×(*n*–*r*)! … so the number of unique combinations of three flavors of ice cream is:

> prod(31:1)/(prod(28:1)\*prod(3:1))

11. How many different five-card poker hands are possible from a standard 52-card deck? Write code to find this answer. Note that there are only four possible “royal flushes” (ace, king, queen, jack, and 10 from the same suit); calculate the probability of being dealt a royal flush.

12. To clear your data, type > rm(list=ls())

To clear the Console window, type *Ctrl*-*l*