Lab 1: R Basics

Statistical Computing, 36-350

Week of Tuesday August 30, 2022

Name:
Andrew ID:
Collaborated with:

This lab is to be done in class (completed outside of class time if need be). You can collaborate with your classmates, but you must identify their names above, and you must submit **your own** lab as an knitted PDF file on Gradescope, by Friday 9pm, this week.

```
## For reproducibility --- don't change this!
set.seed(08312021)
```

This week's agenda: manipulating data objects; using built-in functions, doing numerical calculations, and basic plots; reinforcing core probabilistic ideas.

The binomial distribution

The binomial distribution $\text{Bin}(m, p)$ is defined by the number of successes in m independent trials, each have probability p of success. Think of flipping a coin m times, where the coin is weighted to have probability p of landing on heads.

The R function `rbinom()` generates random variables with a binomial distribution. E.g.,

```
rbinom(n=20, size=10, prob=0.5)
```

produces 20 observations from $\text{Bin}(10, 0.5)$.

Q1. Some simple manipulations

- **1a.** Generate 500 random values from the $\text{Bin}(15, 0.5)$ distribution, and store them in a vector called `bin.draws.0.5`. Extract and display the first 25 elements. Extract and display all but the first 475 elements.

```
# YOUR CODE GOES HERE
```

- **1b.** Add the first element of `bin.draws.0.5` to the fifth. Compare the second element to the tenth, which is larger? A bit more tricky: print the indices of the elements of `bin.draws.0.5` that are equal to 3. How many such elements are there? Theoretically, how many such elements would you expect there to be? Hint: it would be helpful to look at the help file for the `rbinom()` function.

```
# YOUR CODE GOES HERE
```

- **1c.** Find the mean and standard deviation of `bin.draws.0.5`. Is the mean close what you'd expect? The standard deviation?

```
# YOUR CODE GOES HERE
```

- **1d.** Call `summary()` on `bin.draws.0.5` and describe the result.

```
# YOUR CODE GOES HERE
```

- **1e.** Find the data type of the elements in `bin.draws.0.5` using `typeof()`. Then convert `bin.draws.0.5` to a vector of characters, storing the result as `bin.draws.0.5.char`, and use `typeof()` again to verify that you've done the conversion correctly. Call `summary()` on `bin.draws.0.5.char`. Is the result formatted differently from what you saw above? Why?

```
# YOUR CODE GOES HERE
```

Q2. Some simple plots

- **2a.** The function `plot()` is a generic function in R for the visual display of data. The function `hist()` specifically produces a histogram display. Use `hist()` to produce a histogram of your random draws from the binomial distribution, stored in `bin.draws.0.5`.

```
# YOUR CODE GOES HERE
```

- **2b.** Call `tabulate()` on `bin.draws.0.5`. What is being shown? Does it roughly match the histogram you produced in the last question?

```
# YOUR CODE GOES HERE
```

- **2c.** Call `plot()` on `bin.draws.0.5` to display your random values from the binomial distribution. Can you interpret what the `plot()` function is doing here?

```
# YOUR CODE GOES HERE
```

- **2d.** Call `plot()` with two arguments, the first being `1:500`, and the second being `bin.draws.0.5`. This creates a scatterplot of `bin.draws.0.5` (on the y-axis) versus the indices 1 through 500 (on the x-axis). Does this match your plot from the last question?

```
# YOUR CODE GOES HERE
```

Q3. More binomials, more plots

- **3a.** Generate 500 binomials again, composed of 15 trials each, but change the probability of success to: 0.2, 0.3, 0.4, 0.6, 0.7, and 0.8, storing the results in vectors called `bin.draws.0.2`, `bin.draws.0.3`, `bin.draws.0.4.`, `bin.draws.0.6`, `bin.draws.0.7` and `bin.draws.0.8`. For each, compute the mean and standard deviation.

```
# YOUR CODE GOES HERE
```

- **3b.** We'd like to compare the properties of our vectors. Create a vector of length 7, whose entries are the means of the 7 vectors we've created, in order according to the success probabilities of their underlying binomial distributions (0.2 through 0.8).

```
# YOUR CODE GOES HERE
```

- **3c.** Using the vectors from the last part, create the following scatterplots. Explain in words, for each, what's going on.
  - The 7 means versus the 7 probabilities used to generate the draws.
  - The standard deviations versus the probabilities.
  - The standard deviations versus the means.

  Challenge: for each plot, add a curve that corresponds to the relationships you'd expect to see in the theoretical population (i.e., with an infinite amount of draws, rather than just 500 draws).

```
# YOUR CODE GOES HERE
```

Q4. Working with matrices

- **4a.** Create a matrix of dimension 500 x 7, called `bin.matrix`, whose columns contain the 7 vectors we've created, in order of the success probabilities of their underlying binomial distributions (0.2 through 0.8). Hint: use `cbind()`.

```
# YOUR CODE GOES HERE
```

- **4b.** Print the first five rows of `bin.matrix`. Print the element in the 66th row and 5th column. Compute the largest element in first column. Compute the largest element in all but the first column.

```
# YOUR CODE GOES HERE
```

- **4c.** Calculate the column means of `bin.matrix` by using just a single function call.

```
# YOUR CODE GOES HERE
```

- **4d.** Compare the means you computed in the last question to those you computed in Q3b, in two ways. First, using `==`, and second, using `identical()`. What do the two ways report? Are the results compatible? Explain.

```
# YOUR CODE GOES HERE
```

- **4e.** Take the transpose of `bin.matrix` and then take row means. Are these the same as what you just computed? Should they be?

```
# YOUR CODE GOES HERE
```

Q5. Warm up is over, let's go big

- **5a.** R's capacity for data storage and computation is very large compared to what was available 10 years ago. Generate 5 million numbers from Bin$(1 \times 10^6, 0.5)$ distribution and store them in a vector called `big.bin.draws`. Calculate the mean and standard deviation of this vector.

```
# YOUR CODE GOES HERE
```

- **5b.** Create a new vector, called `big.bin.draws.standardized`, which is given by taking `big.bin.draws`, subtracting off its mean, and then dividing by its standard deviation. Calculate the mean and standard deviation of `big.bin.draws.standardized`. (These should be 0 and 1, respectively, or very close to it; if not, you've made a mistake somewhere).

```
# YOUR CODE GOES HERE
```

- **5c.** Plot a histogram of `big.bin.draws.standardized`. To increase the number of histogram bars, set the `breaks` argument in the `hist()` function (e.g., set `breaks=100`). What does the shape of this histogram appear to be? Is this surprising? What could explain this phenomenon? Hint: rhymes with "Mental Gimmick Serum" ...

```
# YOUR CODE GOES HERE
```

- **5d.** Calculate the proportion of times that an element of `big.bin.draws.standardized` exceeds 1.644854. Is this close to 0.05?

```
# YOUR CODE GOES HERE
```

- **5e.** Either by simulation, or via a built-in R function, compute the probability that a standard normal random variable exceeds 1.644854. Is this close to 0.05? Hint: for either approach, it would be helpful to look at the help file for the `rnorm()` function.

```
# YOUR CODE GOES HERE
```

Q6. Now let's go really big

- **6a.** Let's push R's computational engine a little harder. Generate 200 million numbers from Bin($10 \times 10^6, 50 \times 10^{-8}$), and save it in a vector called `huge.bin.draws`.

```
# YOUR CODE GOES HERE
```

- **6b.** Calculate the mean and standard deviation of `huge.bin.draws`. Are they close to what you'd expect? (They should be very close.) Did it longer to compute these, or to generate `huge.bin.draws` in the first place?

```
# YOUR CODE GOES HERE
```

- **6c.** Calculate the median of `huge.bin.draws`. Did this median calculation take longer than the calculating the mean? Is this surprising?

```
# YOUR CODE GOES HERE
```

- **6d.** Calculate the exponential of the median of the logs of `huge.bin.draws`, in one line of code. Did this take longer than the median calculation applied to `huge.bin.draws` directly? Is this surprising?

```
# YOUR CODE GOES HERE
```

- **6e.** Plot a histogram of of `huge.bin.draws`, again with a large setting of the `breaks` argument (e.g., `breaks=100`). Describe what you see; is this different from before, when we had 5 million draws? **Challenge**: Is this surprising? What distribution is this?

```
# YOUR CODE GOES HERE
```

Q7. Going big with lists

- **7a.** Convert `big.bin.draws` into a list using `as.list()` and save the result as `big.bin.draws.list`. Check that you indeed have a list by calling `class()` on the result. Check also that your list has the right length, and that its 1159th element is equal to that of `big.bin.draws`.

```
# YOUR CODE GOES HERE
```

- **7b.** Run the code below, to standardize the binomial draws in the list `big.bin.draws.list`. Note that `lapply()` applies the function supplied in the second argument to every element of the list supplied in the first argument, and then returns a list of the function outputs. (We'll learn much more about the `apply()` family of functions later in the course.) Did this `lapply()` command take longer to evaluate than the code you wrote in Q5b? (It should have; otherwise your previous code could have been improved, so go back and improve it.) Why do you think this is the case?

```
big.bin.draws.mean = mean(big.bin.draws)
big.bin.draws.sd = sd(big.bin.draws)
standardize = function(x) {
  return((x - big.bin.draws.mean) / big.bin.draws.sd)
}
big.bin.draws.list.standardized.slow = lapply(big.bin.draws.list, standardize)
```

- **7c.** Run the code below, which again standardizes the binomial draws in the list `big.bin.draws.list`, using `lapply()`. Why is it so much slower than the code in the last question? (You may stop evaluation if it is taking too long!) Think about what is happening each time the function is called.

```
standardize.slow = function(x) {
  return((x - mean(big.bin.draws)) / sd(big.bin.draws))
}
big.bin.draws.list.standardized.slow = lapply(big.bin.draws.list, standardize.slow)
```

- **7d.** Lastly, let's look at memory useage. The command `object.size(x)` returns the number of bytes used to store the object `x` in your current R session. Find the number of bytes used to store `big.bin.draws` and `big.bin.draws.list`. How many megabytes (MB) is this, for each object? Which object requires more memory, and why do you think this is the case? Remind yourself: why are lists special compared to vectors, and is this property important for the current purpose (storing the binomial draws)?

```
# YOUR CODE GOES HERE
```