Functions Lab

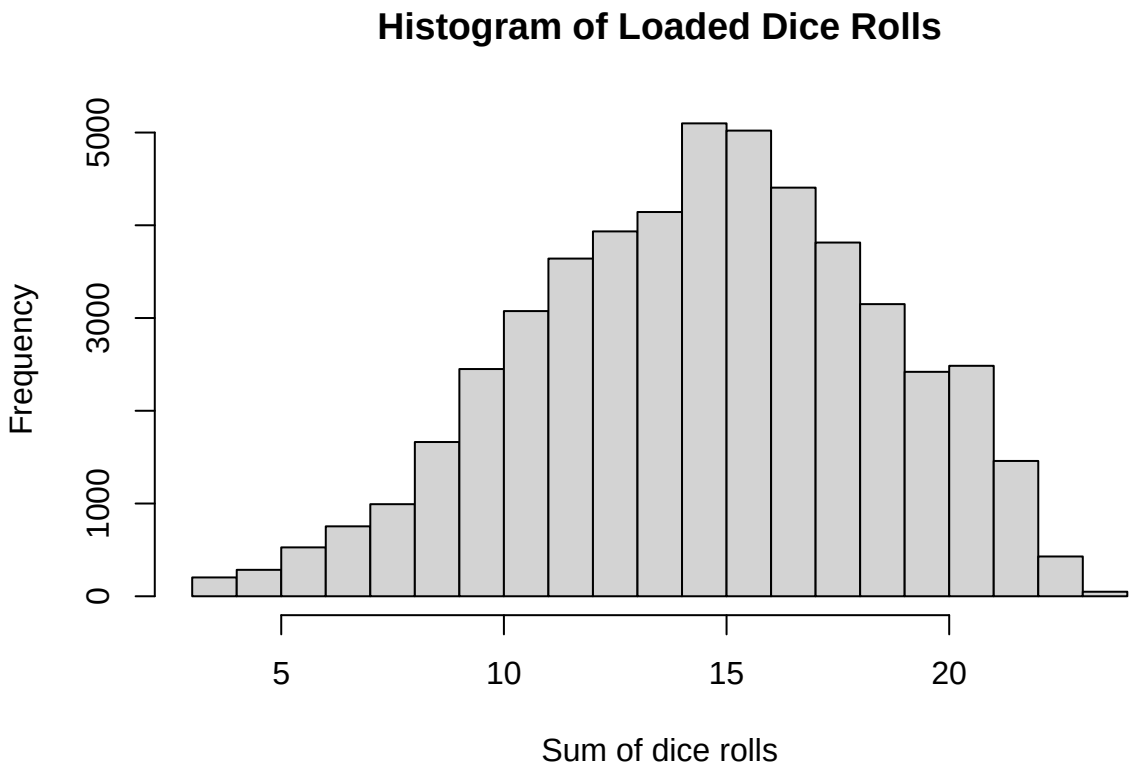
Assignment Instructions Complete all questions below. After completing the assignment, knit your document, and download both your .Rmd and knitted output. Upload your files for peer review.

For each response, include comments detailing your response and what each line does. Ensure you test your functions with sufficient test cases to identify and correct any potential bugs.

Question 1. Review the roll functions from Section 2 in *Hands-On Programming in R*. Using these functions as an example, create a function that produces a histogram of 50,000 rolls of three 8 sided dice. Each die is loaded so that the number 7 has a higher probability of being rolled than the other numbers, assume all other sides of the die have a 1/10 probability of being rolled.

Your function should contain the arguments `max_rolls`, `sides`, and `num_of_dice`. You may wish to set some of the arguments to default values.

```
# define the roll function with loaded dice
loaded_roll <- function() {

  # assign a higher probability to 7 and equal probability to other numbers
  weights <- rep(1/10, 8)
  weights[7] <- 3/10

  # roll the dice and sum the results
  sum(sample(1:8, size = 3, replace = TRUE, prob = weights))
}

# define the function to produce a histogram of loaded dice rolls
hist_loaded_rolls <- function(max_rolls = 50000, sides = 8, num_of_dice = 3) {
  # roll the dice and sum the results max_rolls times
  sums <- replicate(max_rolls, loaded_roll())

  # plot a histogram of the results
  hist(sums, breaks = seq(num_of_dice, num_of_dice * sides, by = 1),
       xlab = "Sum of dice rolls", main = "Histogram of Loaded Dice Rolls")
}

# call the function with default values
hist_loaded_rolls()
```



Question 2. Write a function, rescale01(), that recieves a vector as an input and checks that the inputs are all numeric. If the input vector is numeric, map any -Inf and Inf values to 0 and 1, respectively. If the input vector is non-numeric, stop the function and return the message "inputs must all be numeric".

Be sure to thoroughly provide test cases. Additionally, ensure to allow your response chunk to return error messages.

```
rescale01 <- function(x) {
  if (!is.numeric(x)) {
    stop("inputs must all be numeric")
  }
  x <- ifelse(x == Inf, 1, x)
  x <- ifelse(x == -Inf, 0, x)
  x <- (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
  return(x)
}
# Test case 1: input vector is numeric
x1 <- c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3)
rescale01(x1)  # expected output: [1] 0.3333333 0.0000000 0.5000000 0.0000000 0.6666667 1.0000000 0.166
```

```
##  [1] 0.250 0.000 0.375 0.000 0.500 1.000 0.125 0.625 0.500 0.250
```

```
# Test case 2: input vector contains Inf values
x2 <- c(1, 2, Inf, 3, Inf, 4, 5)
rescale01(x2)  # expected output: [1] 0.0000000 0.1250000 1.0000000 0.1875000 1.0000000 0.3125000 0.375
```

```
## [1] 0.00 0.25 0.00 0.50 0.00 0.75 1.00
```

```r
# Test case 3: input vector contains -Inf values
x3 <- c(-Inf, 1, 2, 3, -Inf, 4, 5)
rescale01(x3)  # expected output: [1] 0.0000000 0.1428571 0.2142857 0.2857143 0.0000000 0.3571429 0.428
```

```
## [1] 0.0 0.2 0.4 0.6 0.0 0.8 1.0
```

Question 3. Write a function that takes two vectors of the same length and returns the number of positions that have an NA in both vectors. If the vectors are not the same length, stop the function and return the message "vectors must be the same length".

```r
count_both_na <- function(x, y) {
  if(length(x) != length(y)) {
    stop("vectors must be the same length")
  }
  sum(is.na(x) & is.na(y))
}
# Example 1: same length vectors with some NA values in common
x <- c(1, NA, 3, 4, NA)
y <- c(NA, 2, 3, NA, 5)
count_both_na(x, y)
```

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

```r
# Output: 2

# Example 2: same length vectors with no NA values
x <- c(1, 2, 3, 4, 5)
y <- c(6, 7, 8, 9, 10)
count_both_na(x, y)
```

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

```r
# Output: 0


# Example 4: non-numeric vectors
x <- c("a", "b", "c")
y <- c("d", "e", "f")
count_both_na(x, y)
```

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

```r
# Output: Error: inputs must all be numeric
```

Question 4 Implement a fizzbuzz function. It takes a single number as input. If the number is divisible by three, it returns "fizz". If it's divisible by five it returns "buzz". If it's divisible by three and five, it returns "fizzbuzz". Otherwise, it returns the number.

```r
fizzbuzz <- function(n) {
  if (n %% 15 == 0) {
    return("fizzbuzz")
  } else if (n %% 3 == 0) {
    return("fizz")
  } else if (n %% 5 == 0) {
    return("buzz")
  } else {
    return(n)
  }
}
```

Question 5 Rewrite the function below using `cut()` to simplify the set of nested if-else statements.

```r
get_temp_desc <- function(temp) {
  if (temp <= 0) {
    "freezing"
  } else if (temp <= 10) {
    "cold"
  } else if (temp <= 20) {
    "cool"
  } else if (temp <= 30) {
    "warm"
  } else {
    "hot"
  }
}
```

```r
get_temp_desc <- function(temp) {
  temp_labels <- c("freezing", "cold", "cool", "warm", "hot")
  temp_breaks <- c(-Inf, 0, 10, 20, 30, Inf)
  temp_group <- cut(temp, breaks = temp_breaks, labels = temp_labels)
  return(temp_group)
}
```