Functions Lab

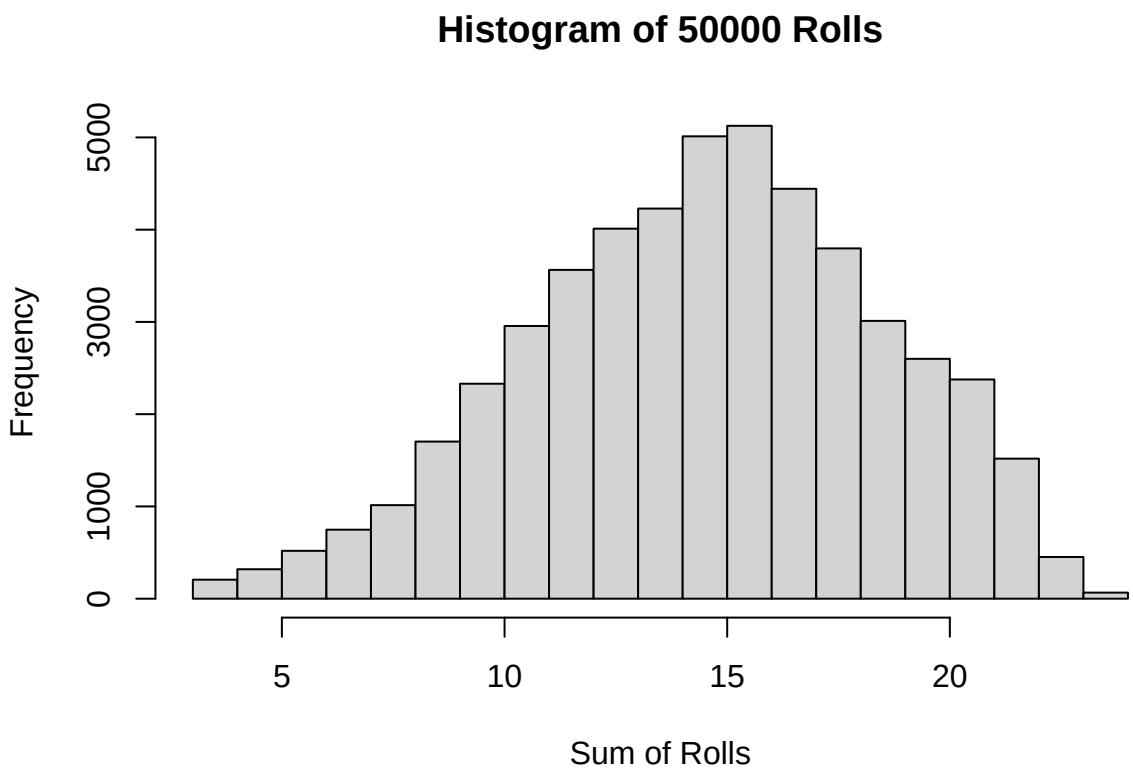
Me Myself

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.



Question 2. Write a function, rescale01(), that receives 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.

```
  # Error_example for knitting in case of error(up), defining function(down)
rescale01 <- function(x) {
  # Verifying the input type
  if (!is.numeric(x)) {
    # Stops to avoid errors for non-numeric values
    stop("inputs must all be numeric")
  }

  # Map -Inf to 0 and Inf to 1 using ifelse() function
  x <- ifelse(x == -Inf, 0, x)
  x <- ifelse(x == Inf, 1, x)

  # Return x
  return(x)
}

# Printing numeric and non-numeric
print(rescale01(8))
```

```
## [1] 8
```

```
print(rescale01("Non-numeric value"))
```

```
## Error in rescale01("Non-numeric value"): inputs must all be numeric
```

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".

```
na_positions <- function(vector1, vector2) {
  # Verifying length of vectors to assure same length
  if (length(vector1) != length(vector2)) {
    stop("vectors must be the same length")
  }

  # Identifying the positions of NA values)
  positions <- sum(is.na(vector1) & is.na(vector2))

  # Return positions of NA values
  return(positions)
}

# Creating vectors for the functions
vector1 <- c(NA, 6, 8, NA, 2, NA)
vector2 <- c(NA, 9, NA, 4, 10, NA)
```

```r
# Print called function
print(na_positions(vector1, vector2))
```

```
## [1] 2
```

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(number) {
  # Creating an if statement for each case returning the required string.
  if (number %% 3 == 0 && number %% 5 == 0) {
    return("fizzbuzz")
  }
  if (number %% 3 == 0) {
    return("fizz")
  }
  if (number %% 5 == 0) {
    return("buzz")
  }
  return (number)
}

#Testing the function
fizzbuzz(1)
```

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

```r
fizzbuzz(15)
```

```
## [1] "fizzbuzz"
```

```r
fizzbuzz(27)
```

```
## [1] "fizz"
```

```r
fizzbuzz(40)
```

```
## [1] "buzz"
```

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

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
# same function with cut(), the first argument is input temperature, the second argument is the break p
get_temp_desc <- function(temp) {
  cut(temp, breaks = c(-Inf, 0, 10, 20, 30, Inf), labels = c("freezing", "cold", "cool", "warm", "hot"))
}
```

```r
#Testing the function

get_temp_desc(-10)
```

```
## [1] freezing
## Levels: freezing cold cool warm hot
```

```r
get_temp_desc(10)
```

```
## [1] cold
## Levels: freezing cold cool warm hot
```

```r
get_temp_desc(100)
```

```
## [1] hot
## Levels: freezing cold cool warm hot
```

```r
get_temp_desc(22)
```

```
## [1] warm
## Levels: freezing cold cool warm hot
```

```r
get_temp_desc(35)
```

```
## [1] hot
## Levels: freezing cold cool warm hot
```