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.

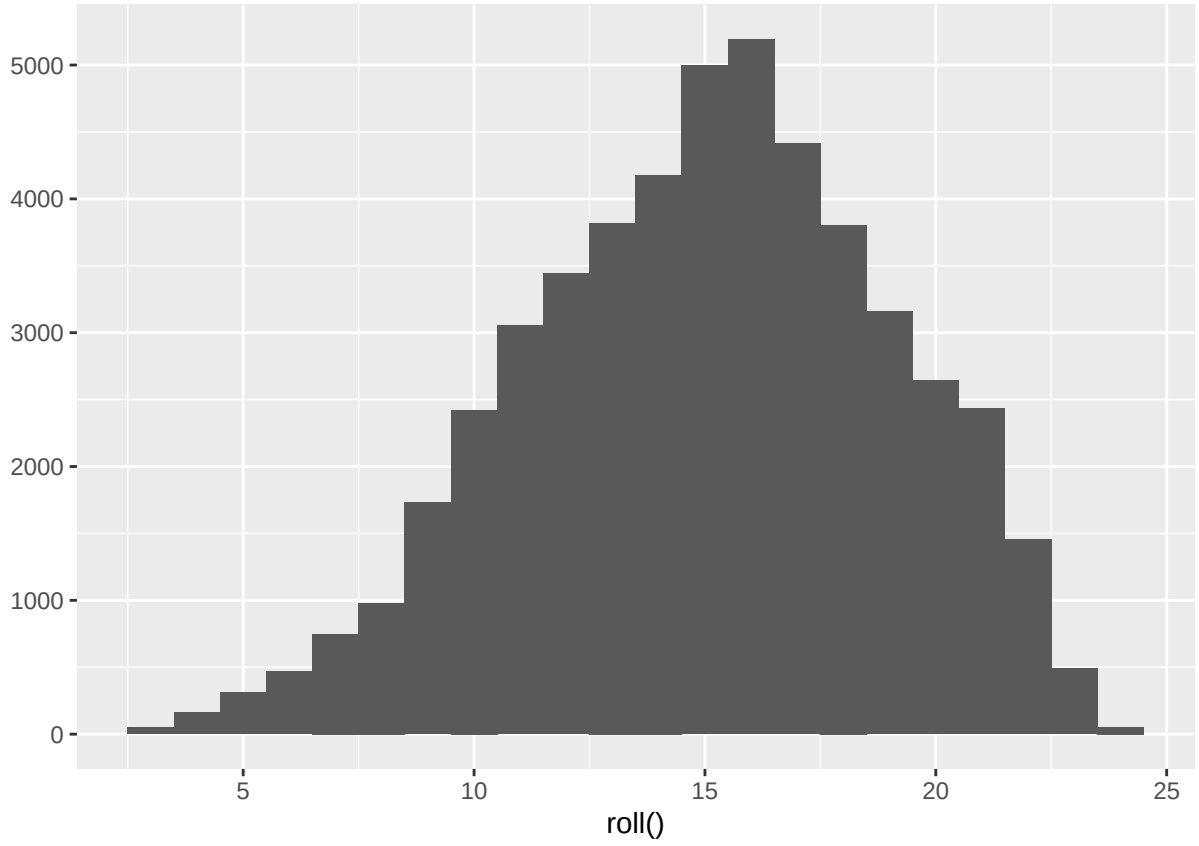
For each response, include comments detailing your response and what each line does. Ensure you test your functions with sufficient test cases to identify and correct any potential bugs.

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.

```
# load ggplot library for histogramm
library(ggplot2)

# roll function
roll <- function(max_rolls = 50000, sides = 1:8, num_of_dice = 3) {

  # function to roll three 8 sided dice one time
  roll_once <- function(){
    dice <- sample(sides, num_of_dice, replace = TRUE,
                   prob = c(1/10, 1/10, 1/10, 1/10, 1/10, 1/10, 3/10, 1/10))
    sum(dice)
  }

  # replicate the dice roll 50000 times and store the result in the object rolls
  rolls <- replicate(max_rolls, roll_once())

  # return the value of rolls
  return(rolls)
}

# produce a histogramm of the results
qplot(roll(), binwidth =1)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



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.

```
# function, that rescales vector
rescale01 <- function(x) {

  # check if all inputs are numeric
  if (!is.numeric(x)) {
    stop("inputs must all be numeric")
  }

  # map -Inf to 0 und Inf to 1
  x[x == -Inf] <- 0
  x[x == Inf] <- 1

  # return rescaled vector
  return(x)
}

# test cases
  # numeric vector
  a <- c(1, 2, 3, 4, 5)
```

```
  # vector with -Inf and Inf
  b <- c(-Inf, 0, 2, 4, Inf)

  # vector with NA value
  c <- c(NA, 3, 5, 6, 9)

  # non numeric vector
  d <- c("a", "b", "c")

# test runs
rescale01(a)
```

```
## [1] 1 2 3 4 5
```

```
rescale01(b)
```

```
## [1] 0 0 2 4 1
```

```
rescale01(c)
```

```
## [1] NA  3  5  6  9
```

```
rescale01(d)
```

```
## Error in rescale01(d): 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".

```
count_na_positions <- function(vec1, vec2) {
  # check if the vectors are of the same length
  if (length(vec1) != length(vec2)) {
    return("vectors must be the same length")
  }

  # return the number of positions that have an NA in both vectors
  na_count <- sum(is.na(vec1) & is.na(vec2))

  return(na_count)
}

#test run
vec1 <- c(1, 2, 3, NA, 5)
vec2 <- c(NA, 4, 7, NA, 9)

count_na_positions(vec1, vec2)
```

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

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.

```
fizzbuzz <- function(x) {

  # check if the number x is divisible by three and five
  if (x %% 3 == 0 && x %% 5 == 0) {
    return("fizzbuzz")
  # check if x is divisible by three
  } else if (x %% 3 == 0) {
    return("fizz")
  # check if x is divisible by five
  } else if (x %% 5 == 0) {
    return("buzz")
  # otherwise return the number
  } else {
    return(x)
  }
}

#test run
fizzbuzz(3)
```

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

```
fizzbuzz(5)
```

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

```
fizzbuzz(15)
```

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

```
fizzbuzz(7)
```

```
## [1] 7
```

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
get_temp_desc <- function(temp) {
  cut(temp,
      # definition of breakpoints
      breaks = c(-Inf, 0, 10, 20, 30, Inf),
      # name of levels
      labels = c("freezing", "cold", "cool", "warm", "hot"))
}

#test run
get_temp_desc(-5)
```

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

```r
get_temp_desc(5)
```

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

```r
get_temp_desc(15)
```

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

```r
get_temp_desc(25)
```

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

```r
get_temp_desc(35)
```

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