Functions Assignment

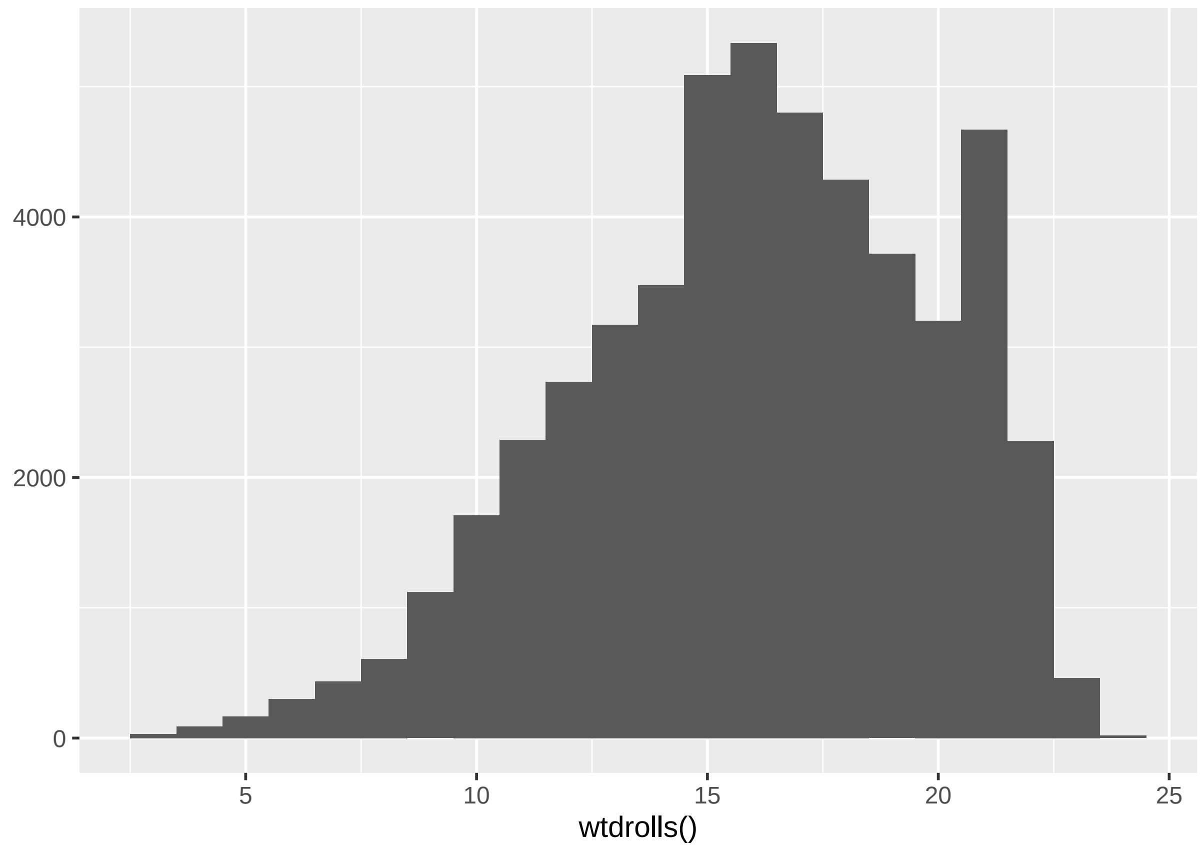
Anthony Tetreault

2025-02-26

1. 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.

```r
wtdrolls <- function(sides=1:8, max_rolls = 50000, num_of_dice = 3) {
  replicate(max_rolls, {
    dice <- sample(sides, size = num_of_dice, replace = TRUE,
                   prob = c(0.1,0.1,0.1,0.1,0.1,0.1,0.5,0.1))
    sum(dice)
  })
}
qplot(wtdrolls(), binwidth = 1)
```



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.")
  }
  rng <- range(x, na.rm = TRUE, finite =TRUE)
  y <- (x- rng[1]) / (rng[2] - rng[1])
  y[y == -Inf] <- 0
  y[y == Inf] <- 1
  y
}
z <- c(-Inf, 24, 17, 45, 6, 42, Inf)
rescale01(z)
```

```
## [1] 0.0000000 0.4615385 0.2820513 1.0000000 0.0000000 0.9230769 1.0000000
```

```
y <- c(36, 7, 42, 495, 344, 76)
rescale01(y)
```

```
## [1] 0.05942623 0.00000000 0.07172131 1.00000000 0.69057377 0.14139344
```

```
x <- c(45, 23, "5k67", "7k7kkkk", 53)
rescale01(x)
```

```
## Error in rescale01(x): Inputs must all be numeric.
```

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".

```
both_na <- function(x, y) {
  if (length(x) != length(y)) {
    stop("Vectors must be the same length.")
  }
  sum(is.na(x) & is.na(y))
}
x <- c(NA, 1, 3, NA, NA, NA, 4)
y <- c(NA, 3, 2, 4, NA, NA, NA)
both_na(x, y)
```

```
## [1] 3
```

```
red <- c(NA, 1, 3, 5, NA)
blue <- c(NA, 1, 3, 2)
both_na(red, blue)
```

```
## Error in both_na(red, blue): Vectors must be the same length.
```

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(x) {
  if (!(x %% 3)) {
    if (!(x %% 5)) {
    print("fizzbuzz")
    } else {
      print("fizz")
    }
  } else if (!(x %% 5)) {
    print("buzz")
  } else {
    print(x)
  }
}
fizzbuzz(36)
```

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

```r
fizzbuzz(55)
```

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

```r
fizzbuzz(120)
```

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

```r
fizzbuzz(347)
```

```
## [1] 347
```

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"
  }
}
```

Rewrite

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

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

```r
get_temp_desc(8)
```

```
## [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(42)
```

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