HW 3 More on R

STAT 5400

Due: Sep 20, 2024 9:30 AM

Submit your solutions as an .Rmd file and accompanying .pdf file. Include all the **relevant** R code and output. With the `echo` option in R markdown, you opt to hide some R code that is not very relevant (but still run it to generate the document).

Always comment on your result whenever it is necessary.

Problems

1. The map function in `tidyverse` Redo Question 4 in HW 1, say, write your own factorial function. At this time, explore `map_dbl` function in the R package `tidyverse`.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
fact <- function(a) {
  return (ifelse(a==1, 1, a * (a-1)))
}

# print(map_dbl(5, 'fact'))
```

2. The skewness function. Download and install the R package `moments`. Find the following `skewness` function.

```
"skewness" <-
function (x, na.rm = FALSE)
{
    if (is.matrix(x))
        apply(x, 2, skewness, na.rm = na.rm)
    else if (is.vector(x)) {
```

```
    if (na.rm) x <- x[!is.na(x)]
    n <- length(x)
     (sum((x-mean(x))^3)/n)/(sum((x-mean(x))^2)/n)^(3/2)
    }
    else if (is.data.frame(x))
        sapply(x, skewness, na.rm = na.rm)
    else skewness(as.vector(x), na.rm = na.rm)
}
```

Write your own skewness function and use `microbenchmark` library to compare the speed with the `skewness` function in `moment`. Can your function be faster than that? Also make sure your function handles both vectors, matrices, and dataframes, as well as NA values in as the `moment` package.

```
# Custom skewness calculation function
calculate_skewness_manual <- function(data) {
  # Helper function to compute skewness for a numeric vector
  skewness_vector <- function(x) {
    n <- length(x)
    mean_x <- mean(x, na.rm = TRUE)
    sd_x <- sd(x, na.rm = TRUE)

    # Calculate skewness using the formula: (n / ((n-1) * (n-2))) * sum((x_i - mean)^3 / sd^3)
    if (sd_x == 0) return(NA)  # Handle case with zero standard deviation
    skewness <- sum(((x - mean_x) / sd_x)^3, na.rm = TRUE) / n
    return(skewness)
  }

  if (is.vector(data)) {
    # For vectors
    return(skewness_vector(data))
  } else if (is.matrix(data) || is.data.frame(data)) {
    # For matrices and data frames
    result <- apply(data, 2, function(column) {
      if (is.numeric(column)) {
        return(skewness_vector(column))
      } else {
        return(NA)  # Return NA for non-numeric columns
      }
    })
    return(result)
  } else {
    stop("Unsupported data type. Please provide a vector, matrix, or data frame.")
  }
}


library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.3.3
```

```
library(moments)  # For built-in skewness function
```

```
##
```

```
## Attaching package: 'moments'

## The following object is masked _by_ '.GlobalEnv':
##
##     skewness
```

```
# Test data
set.seed(123)
test_vector <- rnorm(1000)  # A large vector of random numbers

# Benchmarking the performance
benchmark_results <- microbenchmark(
  Manual_Skewness = calculate_skewness_manual(test_vector),
  Moments_Skewness = skewness(test_vector)
)

# Print the benchmark results
print(benchmark_results)
```

```
## Unit: microseconds
##              expr  min    lq    mean median    uq     max neval
##   Manual_Skewness 75.9 80.00 235.396  82.25 112.5 13256.1   100
##  Moments_Skewness 63.9 67.95 197.987  69.45  76.3 12026.3   100
```

We can simplify how we handle vectorized operations and avoid unnecessary overhead.

3. Monty Hall problem Back in 1970, a game show called "Let's Make a Deal" was hosted by Monty Hall. Suppose there are three doors on the stage. Behind one door there is a nice prize, while behind the other two there are worthless prizes. A contestant selects one door at random, and then Monty Hall opens one of the other two door to reveal a worthless prize. Monty Hall then expresses the willingness to trade the curtain that the contestant has chosen for the other door that has not been opened. Should the contestant switch curtains or stick with the one that she has?

- Determine the probability that she wins the prize if she switches.

P(Win) = P(original choice is correct) + P(original choice is incorrect)

```
print(1/3 + (1/3))
```

```
## [1] 0.6666667
```

- Use a simulation to verify your results.

```
montydoorSwitch <- function() {
  # Choose door behind which prize is hidden
  prize <- sample(1:3, 1)

  # Contestant chooses original door
  original_choice <- sample(1:3, 1)
```

```
  # If original choice is the prize then switching would always result in a win
  if(prize == original_choice) {
    return (FALSE)
  } else {
    return (TRUE)
  }

}

results <- replicate(1000, montydoorSwitch())

# Calculate the ratio of TRUE values (winning by switching)
win_ratio <- mean(results)
print(paste("Win ratio by switching:", win_ratio))
```

```
## [1] "Win ratio by switching: 0.67"
```

- What if there are two prizes and four doors? What about m prizes and n doors, where $m < n$?

$$P(\text{win} \mid \text{switch}) = \frac{4-2}{4} \times \frac{2}{4-1} = \frac{2}{4} \times \frac{2}{3} = \frac{4}{12} = \frac{1}{3}$$

$$P(\text{win} \mid \text{switch}) = \frac{n-m}{n} \times \frac{m}{n-1}$$

4. Coding practice I Check if a positive integer is power of 4.

```
foo <- function (n) {
  # input: foo(n)
  # output: TRUE or FALSE
  # example foo(16) outputs TRUE and foo(31) outputs FALSE
  tol=1e-5


  return(abs( sqrt(sqrt(n)) %% 1 ) < tol)
}

print(foo(16))
```

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

```
print(foo(31))
```

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

5. Coding practice II Write your own function to merge and sort two sorted vectors without using any sorting function such as `sort` or `order`.

```r
set.seed(5400)

a1 <- sort(sample(8, 8, replace=TRUE))
a2 <- sort(sample(10, 8, replace=TRUE))

foo <- function(a1, a2) {
  # Initialize an empty vector to hold the merged result
  a3 <- numeric(length(a1) + length(a2))

  # Initialize pointers for both vectors
  pointer1 <- 1
  pointer2 <- 1
  index <- 1

  # Loop until we reach the end of either vector
  while (pointer1 <= length(a1) && pointer2 <= length(a2)) {
    if (a1[pointer1] < a2[pointer2]) {
      a3[index] <- a1[pointer1]
      pointer1 <- pointer1 + 1
    } else {
      a3[index] <- a2[pointer2]
      pointer2 <- pointer2 + 1
    }
    index <- index + 1
  }

  # If there are remaining elements in a1, add them to a3
  while (pointer1 <= length(a1)) {
    a3[index] <- a1[pointer1]
    pointer1 <- pointer1 + 1
    index <- index + 1
  }

  # If there are remaining elements in a2, add them to a3
  while (pointer2 <= length(a2)) {
    a3[index] <- a2[pointer2]
    pointer2 <- pointer2 + 1
    index <- index + 1
  }

  return(a3)
}

# Example usage
print(foo(a1, a2))
```

```
##  [1]  1  2  2  3  3  3  4  5  5  5  6  8  8  9 10 10
```