CIS_HW_2

2024-09-12

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

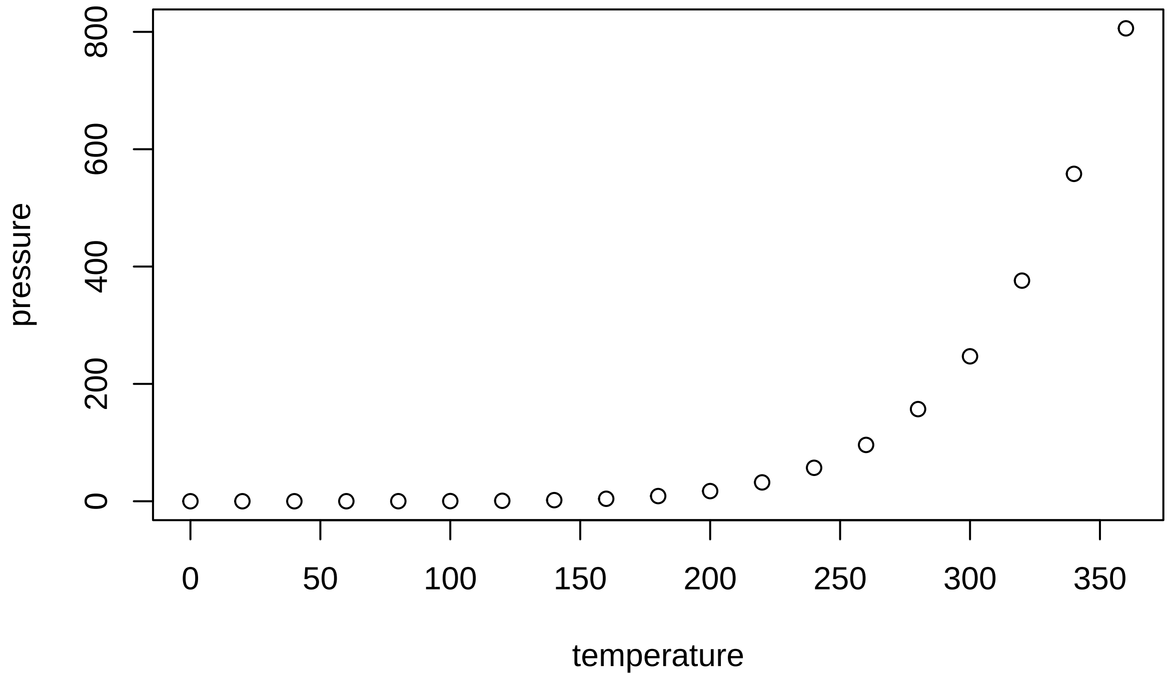
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```r
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Problem 1: Timing two methods of summing 500,000 random variates

```
# Method 1: Generating all random numbers at once and summing
system.time({
  x <- rnorm(500000, mean = 0, sd = 1)
  sum_x <- sum(x)
})
```

```
##    user  system elapsed
##   0.011   0.001   0.011
```

```
# Method 2: Using a for loop to sum random numbers iteratively
system.time({
  answer <- 0
  for (i in 1:500000) {
    answer <- answer + rnorm(1)
  }
})
```

```
##    user  system elapsed
##   0.188   0.003   0.192
```

Comparison of timings: Method 1, where we generate all the random numbers at once and then calculate their sum, is generally more efficient compared to using a for loop to generate and sum each value iteratively. This is because vectorized operations in R are optimized for performance, whereas loops tend to be slower due to the overhead of repeated operations in R.

Problem 2: Matrix Operations

```r
# Define the matrix M
M <- matrix(c(1, 3, 5, 2, 4, 6, 3, 6, 9), nrow = 3, byrow = TRUE)

# Define the vector v
v <- c(17, 46, 181)

# Matrix multiplication
M_v_product <- M %*% v
M_v_product
```

```
##      [,1]
## [1,] 1060
## [2,] 1304
## [3,] 1956
```

```r
# Transpose of matrix M
M_transpose <- t(M)
M_transpose
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    3    4    6
## [3,]    5    6    9
```

```r
# Display elements of v less than 50
v[v < 50]
```

```
## [1] 17 46
```

Problem 3: Skewness and Custom Function

Skewness and Five-Number Summary

```r
options(repos = c(CRAN = "https://cran.rstudio.com/"))
install.packages("e1071")  # Install the package (if not installed)
```

```
##
## The downloaded binary packages are in
##  /var/folders/z0/tj5pgt615dj6crxp6hf_vmg00000gn/T//RtmpuIAIBo/downloaded_packages
```

```r
library(e1071)  # Load the package

# Example usage
x <- rnorm(100)
skewness(x)
```

```
## [1] -0.2023952
```

```r
# Example vector
y <- rnorm(100)
fivenum(y)
```

```
## [1] -2.40937037 -0.76333545  0.04545567  0.55064349  2.03917498
```

Custom R Function

```r
# Custom function to check the vector and compute skewness and summary stats
my_skewness_function <- function(vec) {
  # Check if the vector is numeric
  if (!is.numeric(vec)) {
    return("Vector must be numeric and exist.")
  }

  # Remove NA values
  vec <- na.omit(vec)

  # Compute skewness
  skew_val <- skewness(vec)

  # If the absolute value of skewness is less than 1, return skewness, mean, and SD
  if (abs(skew_val) < 1) {
    descstats <- c(mean = mean(vec), sd = sd(vec))
    return(list(skewness = skew_val, descstats = descstats))
  } else {
    # If skewness is greater than or equal to 1, return skewness and five-number summary
    descstats <- fivenum(vec)
    return(list(skewness = skew_val, descstats = descstats))
  }
}

# Test the function with the provided vectors
test1 <- my_skewness_function(c("stat", "actuarial", "2022"))
test2 <- my_skewness_function(rnorm(100))  # Normal distribution
test3 <- my_skewness_function(rexp(5))  # Exponential distribution

# Print results
test1
```

```
## [1] "Vector must be numeric and exist."
```

```
test2
```

```
## $skewness
## [1] -0.1160488
##
## $descstats
##       mean         sd
## 0.09306911 1.09812929
```

```
test3
```

```
## $skewness
## [1] -0.3690366
##
## $descstats
##     mean       sd
## 3.150035 1.885082
```

Problem 4

Generate the Data:

```
set.seed(1)  # For reproducibility
m <- 1000    # Number of experiments (rows)
n <- 50      # Number of individuals (columns)
X <- matrix(rnorm(m * n, mean = 10, sd = 3), nrow = m)  # 1000 rows, 50 columns
grp <- rep(1:2, each = n/2)  # Group assignment: first 25 are group 1, next 25 are group 2
```

Compute the t-statistic manually for the first sample (row)

```
# For the first sample (row 1)
X1 <- X[1, grp == 1]
X2 <- X[1, grp == 2]

# Manual calculation of the t-statistic
mean1 <- mean(X1)
mean2 <- mean(X2)
var1 <- var(X1)
var2 <- var(X2)
n1 <- length(X1)
n2 <- length(X2)

# Calculate t-statistic
t_stat_manual <- (mean1 - mean2) / sqrt((var1 / n1) + (var2 / n2))
t_stat_manual  # Manual result
```

```
## [1] -0.5284632
```

```r
# Using t.test() for the first row
t_stat_test <- t.test(X[1, grp == 1], X[1, grp == 2])$statistic
t_stat_test  # Should be approximately the same as the manual result
```

```
##          t
## -0.5284632
```

Compute t-statistics for all 1000 samples:

```r
# Compute t-statistics for all 1000 samples using t.test()
t_stats_all <- apply(X, 1, function(row) {
  t.test(row[grp == 1], row[grp == 2])$statistic
})

head(t_stats_all)  # Display the first few t-statistics
```

```
## [1] -0.5284632 -1.9921718 -0.9203112 -0.3869880 -3.1064628  0.5531318
```

Optimize the Code:

```r
# Optimized computation of t-statistics for all rows
system.time({
  means_grp1 <- rowMeans(X[, grp == 1])  # Mean of group 1 for each row
  means_grp2 <- rowMeans(X[, grp == 2])  # Mean of group 2 for each row
  vars_grp1 <- apply(X[, grp == 1], 1, var)  # Variance of group 1 for each row
  vars_grp2 <- apply(X[, grp == 2], 1, var)  # Variance of group 2 for each row

  t_stats_optimized <- (means_grp1 - means_grp2) / sqrt((vars_grp1 / 25) + (vars_grp2 / 25))
})
```

```
##    user  system elapsed
##   0.005   0.000   0.006
```

```r
head(t_stats_optimized)  # Display the first few t-statistics
```

```
## [1] -0.5284632 -1.9921718 -0.9203112 -0.3869880 -3.1064628  0.5531318
```

Compare with the adv-r Approach

```r
# Compare timing for both approaches
system.time({
  t_stats_test <- apply(X, 1, function(row) {
    t.test(row[grp == 1], row[grp == 2])$statistic
  })
})
```

```
##    user  system elapsed
##   0.044   0.000   0.044
```

```r
system.time({
  means_grp1 <- rowMeans(X[, grp == 1])
  means_grp2 <- rowMeans(X[, grp == 2])
  vars_grp1 <- apply(X[, grp == 1], 1, var)
  vars_grp2 <- apply(X[, grp == 2], 1, var)
  t_stats_optimized <- (means_grp1 - means_grp2) / sqrt((vars_grp1 / 25) + (vars_grp2 / 25))
})
```

```
##    user  system elapsed
##   0.006   0.000   0.005
```

Comments on Performance

The optimized code using rowMeans() and apply() for variance should be faster than calling t.test() for each row individually. This is because t.test() involves some overhead due to hypothesis testing logic and checking input data. On the other hand, directly calculating means and variances focuses only on the essential operations, making it faster.