CIS_HW1

2024-09-05

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.

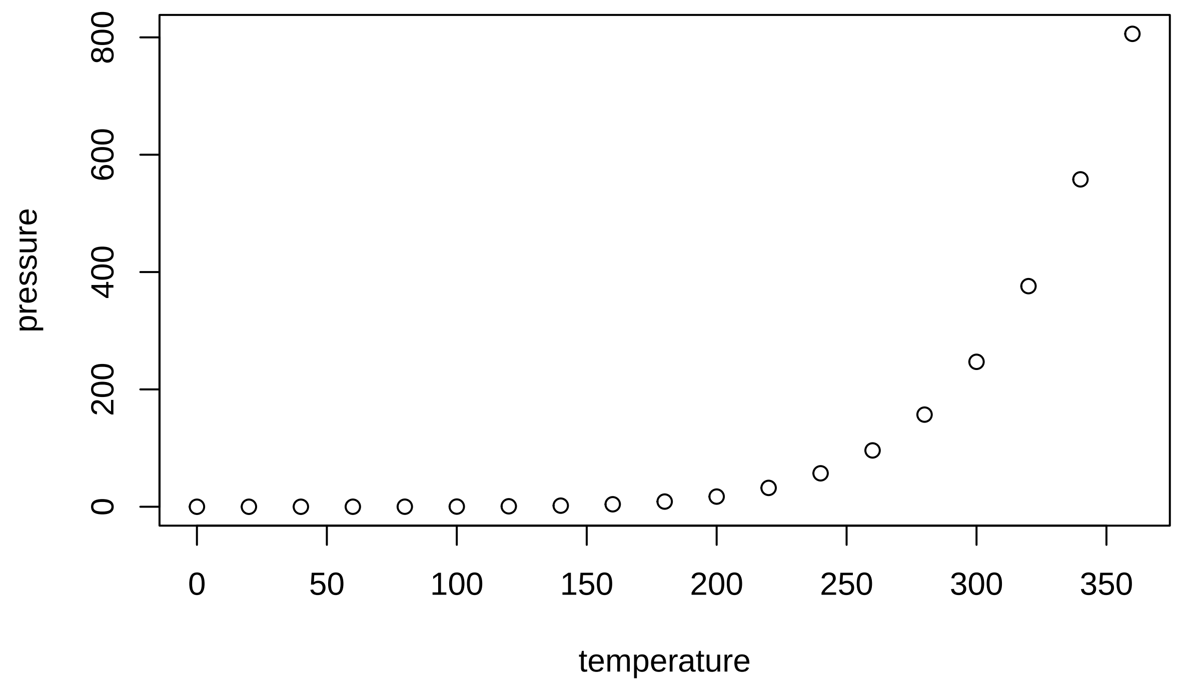
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```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:

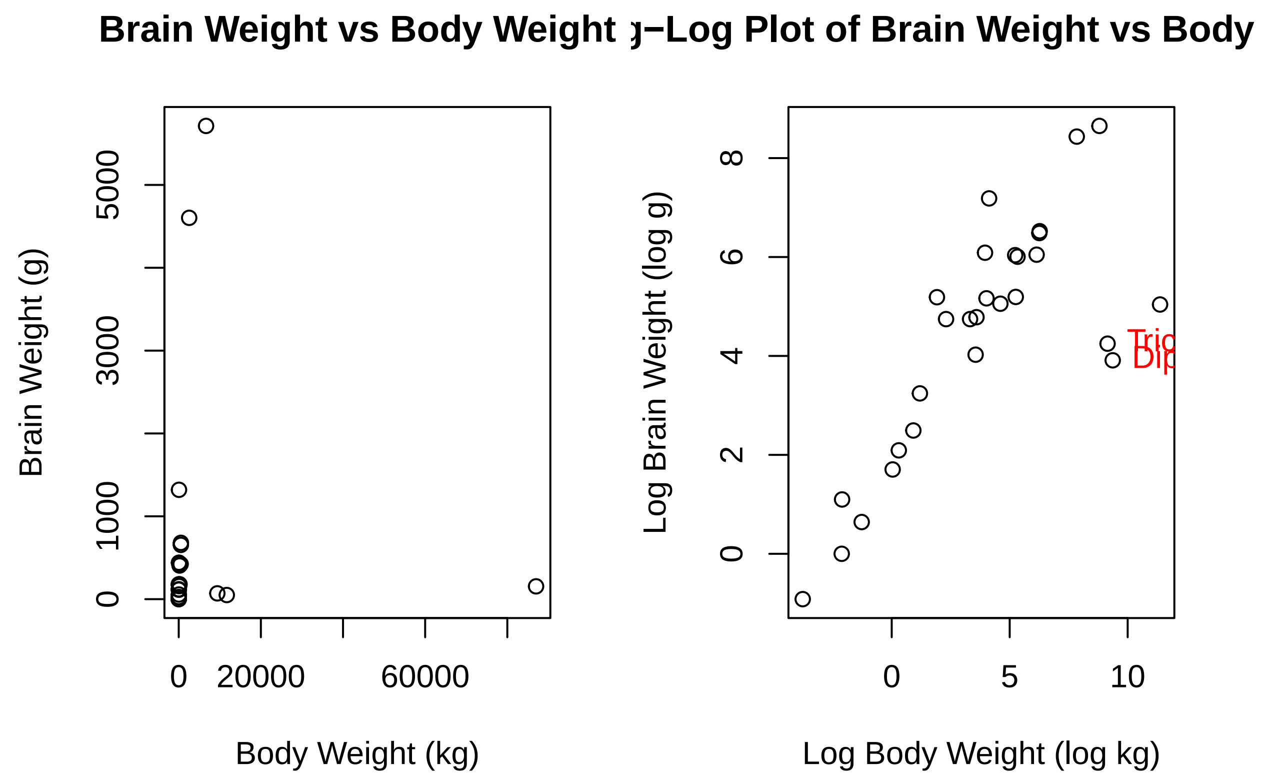


Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```
# Define the range of radii
radius_list <- 3:20

# Calculate the corresponding volumes
volume_list <- (4/3) * radius_list^3 * pi

# Construct the data frame
sphere <- data.frame(Radius=radius_list, Volume=volume_list)

# Print the data frame
print(sphere)
```

```
##    Radius     Volume
## 1       3   113.0973
## 2       4   268.0826
## 3       5   523.5988
## 4       6   904.7787
## 5       7  1436.7550
## 6       8  2144.6606
## 7       9  3053.6281
## 8      10  4188.7902
## 9      11  5575.2798
## 10     12  7238.2295
## 11     13  9202.7721
## 12     14 11494.0403
## 13     15 14137.1669
## 14     16 17157.2847
## 15     17 20579.5263
## 16     18 24429.0245
## 17     19 28730.9120
## 18     20 33510.3216
```

```
# Load the MASS package
library(MASS)

# Access the Animals data set
data(Animals)

# Identify the three largest body weights
largest_bodies <- order(Animals$body, decreasing = TRUE)[1:3]

# Create a two-panel plot layout
par(mfrow = c(1, 2))

# Plot brain weight vs body weight
plot(Animals$body, Animals$brain,
     xlab = "Body Weight (kg)",
     ylab = "Brain Weight (g)",
     main = "Brain Weight vs Body Weight")

# Plot log(brain weight) vs log(body weight)
plot(log(Animals$body), log(Animals$brain),
     xlab = "Log Body Weight (log kg)",
     ylab = "Log Brain Weight (log g)",
     main = "Log-Log Plot of Brain Weight vs Body Weight")

# Label the points with the three largest body weights
text(log(Animals$body)[largest_bodies], log(Animals$brain)[largest_bodies],
     labels = rownames(Animals)[largest_bodies],
     pos = 4, col = "red")
```



```r
# Reset the plot layout to default
par(mfrow = c(1, 1))
```

1. match() The match() function returns the position of the first occurrence of each element of one vector in another vector.

Example: Let's find the positions of elements from one vector in another vector.

```r
# Define two vectors
vec1 <- c("apple", "banana", "cherry")
vec2 <- c("banana", "cherry", "apple", "date")

# Find positions of vec1 elements in vec2
positions <- match(vec1, vec2)

# Print positions
positions
```

```
## [1] 3 1 2
```

Output Explanation: positions: The indices in vec2 where the elements of vec1 are found.

2. rank() The rank() function returns the ranks of the elements in a vector, with ties getting average ranks.

Example: Let's rank a vector of numeric values.

```r
# Numeric vector
values <- c(40, 10, 30, 20, 50)
# Get ranks of the elements
ranks <- rank(values)
# Print ranks
ranks
```

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

Output Explanation: ranks: The rank of each element in values (e.g., 1 for the smallest, 5 for the largest).

3. substring() The substring() function extracts or replaces substrings from a character vector.

Example: Let's extract substrings from a vector of text string

```r
# Character vector
text <- c("HelloWorld", "DataScience", "MachineLearning")

# Extract substrings from position 1 to 5
substrings <- substring(text, 1, 5)

# Print substrings
substrings
```

```
## [1] "Hello" "DataS" "Machi"
```

Output Explanation: substrings: The extracted substrings from each element of text, from the first to the fifth character.

Sure! Let's write our own factorial function in R and then compare its runtime with R's built-in `factorial()` function. We'll also implement a factorial function using `Reduce()` and compare its performance.

1. Custom Factorial Function Using a Loop

```r
# Custom factorial function using a loop
factorial_loop <- function(n) {
  if (n == 0) return(1)
  result <- 1.0
  for (i in 1:n) {
    result <- result * i
  }
  return(result)
}
```

2. Custom Factorial Function Using `Reduce()`

```r
# Custom factorial function using Reduce
factorial_reduce <- function(n) {
  if (n == 0) return(1)
  Reduce(function(x, y) x * y,as.numeric(1:n))
}
```

3. Comparing Runtime

We'll use the `microbenchmark` package to compare the runtimes of these functions with R's built-in `factorial()` function.

First, install and load the `microbenchmark` package if you haven't already:

```
library(microbenchmark)
```

Then, perform the benchmark:

```
# Define the number for testing
n <- 20

# Benchmark the functions
benchmark_results <- microbenchmark(
  factorial_r = factorial(n),
  factorial_loop = factorial_loop(n),
  factorial_reduce = factorial_reduce(n),
  times = 1000
)
```

```
## Warning in microbenchmark(factorial_r = factorial(n), factorial_loop =
## factorial_loop(n), : less accurate nanosecond times to avoid potential integer
## overflows
```

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

```
## Unit: nanoseconds
##              expr  min   lq      mean median   uq     max neval
##       factorial_r   82   82   140.753    123  123    2009  1000
##    factorial_loop  328  369  7370.365    410  492 6888492  1000
##  factorial_reduce 6601 7831 10195.880   8446 9553 1344636  1000
```

```
# Function to find the unique element that appears an odd number of times
find_odd_occurrence <- function(seq) {
  # Create a frequency table of the elements
  counts <- table(seq)

  # Identify the element that has an odd frequency
  odd_element <- as.numeric(names(counts[counts %% 2 != 0]))

  # Return the odd element
  return(odd_element)
}

# Example based on the given code
set.seed(1)
n <- 5
a <- sample(n, n, replace = TRUE)
myseq <- sample(c(a, a))[seq(2 * n - 1)]
```

```r
# Print the sequence
print(myseq)
```

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

```r
# Find the element that appears an odd number of times
find_odd_occurrence(myseq)
```

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