PSTAT 10 Homework/Worksheet 1 Solutions

Sou Hamura

Problem 1: Exploring the ecosystem

read.fortunes <- function(file = NULL) {

it takes one argument, file = NULL

Problem 2: More ecosystem exploration

#BUG FIXES: #as.numeric(), scan(), type.convert() and other places which use the internal C function R_strtod now require a non-empty digit sequence in a decimal or binary exponent. This aligns with the C/POSIX standard for strtod and with ?NumericConstants.

#NEW FEATURES: #The confint() methods for "glm" and "nls" objects have been copied to the stats package. Previously, they were stubs which called versions in package MASS. The MASS namespace is no longer loaded if you invoke (say) confint(glmfit). Further, the "glm" method for profile() and the plot() and pairs() methods for class "profile" have been copied from MASS to stats. (profile.nls() and plot.profile.nls() were already in stats.)

##Problem 3: State areas

```
us_state_name <- data.frame(state.name)
us_state_area <- data.frame(state.area)
```

#1. find the mean area of all states

```
mean_area <- mean(us_state_area$state.area)
mean_area
```

```
## [1] 72367.98
```

#2. Find the median area of all states.

```
median_area <- median(us_state_area$state.area)
median_area
```

```
## [1] 56222
```

#3. Find the name and the area of the smallest state.

```r
smallest_area <- which.min(us_state_area$state.area)
smallest_state <- us_state_name$state.name[smallest_area]
smallest_area
```

```
## [1] 39
```

```r
smallest_state
```

```
## [1] "Rhode Island"
```

#4. Find the name and the area of the largest state

```r
largest_area <- which.max(us_state_area$state.area)
largest_state <- us_state_name$state.name[largest_area]
largest_area
```

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

```r
largest_state
```

```
## [1] "Alaska"
```

Prolem 4: A function without paremeters

```r
my_time <- function(){
  time <- Sys.time()
  hello <- "Hello, the time is"
  time_only <- format(time, "%H:%M:%S")
  return(paste(hello, time_only))
}

my_time()
```

```
## [1] "Hello, the time is 12:05:15"
```

Problem 4: Length of a vector

```r
vector_length <- function(vec){
  if(is.numeric(vec) != TRUE){
    return(print("The argument must be a numeric vector"))
  }
  else{
    nijo <- vec * vec
    sum <- sum(nijo)
    root <- sum ** (1/2)
    return(root)
```

```
  }
}

vector_length(c(5, 9, 11, 15))
```

```
## [1] 21.26029
```

Problem 4: Dot product

```
dot_product <- function(vec1, vec2){
  if(is.numeric(vec1) != TRUE || is.numeric(vec2) != TRUE){
    print("Both argumets must be numeric!")
  }
  else{
    pro <- vec1 * vec2
    sum <- sum(pro)
    return(sum)
  }
}

dot_product(1:3, c(0, 1, 5))
```

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

```
dot_product(2, 4)
```

```
## [1] 8
```

```
dot_product(c(1, 1), c("dog", "cat"))
```

```
## [1] "Both argumets must be numeric!"
```

Problem 5: Frobenius norm

```
frobenius_norm <- function(mat){
  if(is.matrix(mat) != TRUE){
    print("Argument must be a matrix!")
  }
  else{
    value <- c(mat)
    if(is.numeric(value) != TRUE){
      print("Argument must be numeric!")
    }
    else{
      prod <- value ** 2
      sum <- sum(prod)
      root <- sum ** (1/2)
```

```
      return(root)
    }
  }
}


frobenius_norm(matrix(1:4, nrow = 2, ncol = 2))
```

```
## [1] 5.477226
```

```
frobenius_norm(c(3,5,7,10,15,21))
```

```
## [1] "Argument must be a matrix!"
```

```
frobenius_norm(matrix(c(3,5,7,10,15,21), nrow = 2, ncol = 3))
```

```
## [1] 29.1376
```

```
frobenius_norm(matrix(c(3,"fish",7,10,15,21), nrow = 2, ncol = 3))
```

```
## [1] "Argument must be numeric!"
```

Problem 6: Compare Count

```
compare_count <- function(x, y, comp = ">"){
  if(comp != ">" & comp != "<" & comp != "="){
    print("Unrecognized compare operator!")
    return(NULL)
  }
  else if(length(x) != length(y)){
    print("Both vectors must have the same length!")
    return(NULL)
  }
  else if(is.numeric(x) != TRUE | is.numeric(y) != TRUE){
    print("Both vectors must be numeric!")
    return(NULL)
  }
  else{
    if(comp == "="){
      result <- x == y
    }
    else if(comp == "<"){
      result <- x < y
    }
    else if(comp == ">"){
      result <- x > y
    }
  }
  num <- length(result[result == TRUE])
```

```
  return(num)
}


compare_count(rep(1, 5), rep(2, 5))
```

```
## [1] 0
```

```
compare_count(rep(1, 5), rep(2, 5), ">")
```

```
## [1] 0
```

```
compare_count(c(1, 2, 1, 2, 1), rep(2, 5), "<")
```

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

```
compare_count(c(1, 2, 1, 2, 1), rep(2, 5), "=")
```

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

```
compare_count(c(1, 2, 1, 2, 1), rep(2, 5), ">=")
```

```
## [1] "Unrecognized compare operator!"

## NULL
```

```
compare_count(c(1, 2, 1, 2, 1), rep(2, 6), "=")
```

```
## [1] "Both vectors must have the same length!"

## NULL
```

```
compare_count(c(1, 2, 1, 2, "owl"), rep(2, 6))
```

```
## [1] "Both vectors must have the same length!"

## NULL
```