problem_set_4a.R

Aaryan Agarwal

2023-12-08

```r
# PROBLEM SET 4a
# This assignment is done in a group of three
# Prajwal Kaushal, Sumukha Sharma, Aaryan Agarwal

#Part1
#Question 1
#1
calculate_na_proportion <- function(x) {
  mean(is.na(x))
}
# This function calculated the proportion of NA values in x

calculate_na_proportion(c(10,20,NA,40,50,NA,60,70))
```

```
## [1] 0.25
```

```r
standardize_vector <- function(x,na.rm = TRUE ) {
  x / sum(x, na.rm = na.rm)
}
#This function standardize the vector. It returns the normalize version where each element
#is divided by the sum of all the elements in the vector so that it the sums to one.
#The na.rm is used to handle the NA elements in the vector.
#If na.rm = FALSE, it returns NA and if na.rm = TRUE,ir drops NA

#when na.rm = TRUE
standardize_vector(c(10,20,NA,40,50,NA,60,70))
```

```
## [1] 0.04 0.08   NA 0.16 0.20   NA 0.24 0.28
```

```r
#when na.rm = FALSE
standardize_vector(c(10,20,NA,40,50,NA,60,70),na.rm = FALSE )
```

```
## [1] NA NA NA NA NA NA NA NA
```

```r
calculate_coefficient_of_variation <- function(x) {
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
}
#This funvtion calculates the coefficient of variation by dividing standard deviation with mean.
#Here both mean and standard deviation are calculated by dropping NA values since na.rm = TRUE

calculate_coefficient_of_variation(c(10,20,NA,40,50,NA,60,70))
```

```
## [1] 0.5559856
```

```r
########################################################################################################
#2
both_na <- function(x,y) {
  if(length(x) == length(y)) {
    result <- is.na(x) & is.na(y)
  }
  return(result)
}

both_na(c(10,20,NA,40), c(NA,60,NA,NA))
```

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

```r
########################################################################################################
is_directory <- function(x) file.info(x)$isdir
is_readable <- function(x) file.access(x, 4) == 0

#The function is_directory checks whether the path in x is a directory.
#t uses the file.info function to retrieve file information and then extracts the "isdir" field from th

#The function is_readqble checks whether the file is readable which means
#that the file in the given path exists  and the user has permission to open it.

#These two funvtions are useful while accessing the files, to check for the correct directory and
#user permisions to access the file.
########################################################################################################
#4
f1 <- function(string, prefix) {
  substr(string, 1, nchar(prefix)) == prefix
}
#This function checks if each element in the vector statrs with the given prefix.
#the fucntion can be renamed as starts_with_prefix
starts_with_prefix <- function(string, prefix) {
  substr(string, 1, nchar(prefix)) == prefix
}
starts_with_prefix(c("abort","adapt","absorb","abroad","append"),"ab")
```

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

```r
f2 <- function(x) {
  if (length(x) <= 1) return(NULL)
  x[-length(x)]
}
#This function drops the last element of the vector. If the vector has only one element or if the vecto
#empty, it returns NULL.the fucntion can be renamed as remove_last_element
remove_last_element <- function(x) {
  if (length(x) <= 1) return(NULL)
  x[-length(x)]
}
remove_last_element(c(1,2,3,4,5))
```

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

```
remove_last_element(c(1))
```

```
## NULL
```

```
remove_last_element(c())
```

```
## NULL
```

```
f3 <- function(x, y) {
  rep(y, length.out = length(x))
}
#This function repeats the element y element in the vector of lenght x.
#the fucntion can be renamed as repeat_element
repeat_element <- function(x, y) {
  rep(y, length.out = length(x))
}
repeat_element(c(1,2,3,4,5),3)
```

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

```
########################################################################################################
#5
fizzbuzz <- function(x) {
  if((x %% 3 == 0) && (x %% 5 == 0 )){
    "fizzbuzz"
  }else if (x %% 3 == 0){
    "fizz"
  }else if (x %% 5 == 0){
    "buzz"
  }else {
    x
  }
}
fizzbuzz(15)
```

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

```
fizzbuzz(3)
```

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

```
fizzbuzz(10)
```

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

```r
fizzbuzz(11)
```

```
## [1] 11
```

```r
#######################################################################################################
#6
temp <- 40
cut(temp, breaks = c(-Inf, 5, 10, 20, 30,Inf),
                          labels = c("freezing", "cold", "cool", "warm", "hot"),
                          right = TRUE)
```

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

```r
#To use '<' instead of '<=' while working with cut, we can change right = FALSE
#chief advantage of using cut is that it can handle multiple values which means that it can work with v
#which cannot happen while using if statement which takes single value. Also the handling the intervals
#temperature values, we need to change the operators in case of if statement whereas with cut, we just
#change right = False.
#using cut also makes the code simple and readable.
#######################################################################################################
#7
x <-'e'
switch_out <- switch(x,
                     a = ,
                     b = "ab",
                     c = ,
                     d = "cd"
)
switch_out
```

```
## NULL
```

```r
#The switch function returns the first non missing values it encounters while matching the arguments.
#According to the above code, it will return 'ab' for 'a', 'ab' for 'b', 'cd' for 'c', 'cd' for 'd.'
#If given 'e', it returns NULL since 'e' is not there in the given arguments.
#######################################################################################################
#Question 2
#1
#is.vector() checks if it is a specific type of vector with no attributes other than names. That is
#is.vector() checks for a specific type of vector as defined by the specified mode and imposes
#the constraint that the vector should have no attributes other than names.
#is.atomic() specifically checks if an object belongs to the atomic modes: "logical", "integer",
#"numeric" (synonym "double"), "complex", "character", or "raw".If an object is of any of these
#modes and has no attributes other than names, is.atomic() returns TRUE.
#######################################################################################################
#2
x <- c(5,10,15,20,25,30)

#returns last element
last_ele <- function(x) {
```

```r
  x[length(x)]
}
last_ele(x)
```

```
## [1] 30
```

```r
#elements at even numbered positions
even_positions <- function(x){
  x[seq(from = 2, length(x),by = 2)]
}
even_positions(x)
```

```
## [1] 10 20 30
```

```r
#Every element except the last value.
except_last_value <- function(x){
  x[1:length(x)-1]
}
except_last_value(x)
```

```
## [1]  5 10 15 20 25
```

```r
#Only even numbers
even_numbers <- function(x){
  x[x %% 2 == 0]
}
even_numbers(x)
```

```
## [1] 10 20 30
```

```r
########################################################################################################
#3
#x[-which(x > 0)] and x[x <= 0] gives similar results. However, the difference comes while handling nul
#x[-which(x > 0)] will ignore `NA`'s and leave them as it is and
# x[x <= 0]  returns any value that cannot be comparable as NA
x <- c(1,2,3,4,5,NaN,NA)
x[-which(x > 0)]
```

```
## [1] NaN  NA
```

```r
x[x <= 0]
```

```
## [1] NA NA
```

```r
########################################################################################################
#4
#when we subset with a positive integer that's bigger than the length of the vector, It returns NA
x <- c(2,4,6,8)
x[5]
```

```
## [1] NA
```

```r
#when we subset with a name that doesn't exist, still it returns NA
x <- c(a = 1, b = 2, c = 3)
x["d"]
```

```
## <NA>
##   NA
```

```r
########################################################################################################
#5
res = list('a', 'b', list('c', 'd'), list('e', 'f'))
res1 = list(list(list(list(list(list('a'))))))
########################################################################################################
#Question 3
#1
#he mean of every column in mtcars
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2

## Warning: package 'ggplot2' was built under R version 4.2.2

## Warning: package 'stringr' was built under R version 4.2.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
mtcars_means <- vector("double", ncol(mtcars))
names(mtcars_means) = names(mtcars)
for (i in names(mtcars)) {
  mtcars_means[i] <- mean(mtcars[, i])
}
mtcars_means
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

```r
#type of each column in nycflights13::flights
library(nycflights13)
flights_types <-  vector("list", ncol(flights))
names(flights_types) <- names(flights)
for (i in names(flights)){
  flights_types[[i]] <-  class(flights[[i]])
}
flights_types
```

```
## $year
## [1] "integer"
##
## $month
## [1] "integer"
##
## $day
## [1] "integer"
##
## $dep_time
## [1] "integer"
##
## $sched_dep_time
## [1] "integer"
##
## $dep_delay
## [1] "numeric"
##
## $arr_time
## [1] "integer"
##
## $sched_arr_time
## [1] "integer"
##
## $arr_delay
## [1] "numeric"
##
## $carrier
## [1] "character"
##
## $flight
## [1] "integer"
##
## $tailnum
## [1] "character"
##
## $origin
## [1] "character"
##
## $dest
## [1] "character"
##
## $air_time
## [1] "numeric"
##
## $distance
## [1] "numeric"
##
## $hour
## [1] "numeric"
##
## $minute
## [1] "numeric"
##
```

```
## $time_hour
## [1] "POSIXct" "POSIXt"
```

```
#number of unique values in each column of iris
data(iris)
iris_unique_counts <- vector("double", ncol(iris))
names(iris_unique_counts) <- names(iris)
for (i in names(iris)) {
  iris_unique_counts[i] <- length(unique(iris[[i]]))
}
iris_unique_counts
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##           35           23           43           22            3
```

```
#10 random normals for each of mu = -10, 0, 10, and 100
mu_values <- c(-10, 0, 10, 100)
random_normals <- vector("list", length(mu_values))
for (i in seq_along(random_normals)) {
  random_normals[[i]] <- rnorm(10, mean = mu_values[i])
}
random_normals
```

```
## [[1]]
##  [1] -11.924662 -10.755108  -8.877296 -10.201195  -8.625040 -10.027645
##  [7]  -9.088611  -9.266832 -10.476637 -10.988767
##
## [[2]]
##  [1] -0.8192203  1.7567165 -0.7367987  0.2884779  0.5098887  0.6157295
##  [7]  0.1437723  0.6244447  0.9215035  0.3620429
##
## [[3]]
##  [1] 10.426648 13.674756  8.071447  9.062279 10.391058 10.824349  9.531927
##  [8]  9.741673  9.575444  9.475149
##
## [[4]]
##  [1] 101.87441 100.79778 100.43538 100.44907  99.49232  98.61299  99.07646
##  [8] 100.86896 100.80077 100.24302
```

```
###########################################################################################################
#2
out <- ""
for (x in letters) {
  out <- str_c(out, x)
}
out
```

```
## [1] "abcdefghijklmnopqrstuvwxyz"
```

```
#str_c works with vectors. Hence we can use str_c() with the collapse to return single argument
str_c(letters, collapse = "")
```

```
## [1] "abcdefghijklmnopqrstuvwxyz"
```

```
x <- sample(100)
sd <- 0
for (i in seq_along(x)) {
  sd <- sd + (x[i] - mean(x)) ^ 2
}
sd <- sqrt(sd / (length(x) - 1))
sd
```

```
## [1] 29.01149
```

```
#we have inbuilt function for standard deviation and we can use that
sd(x)
```

```
## [1] 29.01149
```

```
x <- runif(100)
out <- vector("numeric", length(x))
out[1] <- x[1]
for (i in 2:length(x)) {
  out[i] <- out[i - 1] + x[i]
}
out
```

```
##   [1]  0.6356822  0.6516984  1.0968228  1.7936585  2.5832804  2.8784087
##   [7]  3.3058126  4.0942250  4.5260372  5.4884799  6.3560634  7.2616213
##  [13]  7.9767410  8.2520153  8.6657439  8.7889136  9.5864055 10.0293412
##  [19] 10.4938538 11.3910198 12.3187792 12.3375007 12.7834097 13.7641795
##  [25] 14.4773917 14.6695184 15.6598150 16.4199068 16.8991088 17.6548045
##  [31] 18.2766297 18.3009445 18.7838471 19.7821069 20.7311701 21.4798825
##  [37] 21.9937426 22.5850064 23.4521036 23.8622832 24.1271329 25.1165413
##  [43] 26.0940669 26.4197936 26.5224378 26.7833610 27.0767097 27.5342914
##  [49] 28.5001404 29.0139957 30.0099129 30.9274816 31.1272090 31.2412186
##  [55] 32.1326147 32.8127133 33.4492759 33.9806792 34.4748673 35.3848277
##  [61] 36.0794068 36.5693499 36.6946817 36.7092654 36.7213446 36.7372823
##  [67] 37.1091941 37.6289677 38.2858735 38.8395104 39.2138672 39.4605764
##  [73] 39.8628801 40.7758210 41.7616143 42.4265713 42.9014504 43.7740011
##  [79] 44.2919272 44.7406753 45.0949241 45.7056694 46.4610271 46.8067474
##  [85] 47.7445954 48.0877819 48.7200890 48.8653614 49.3034205 49.9143285
##  [91] 49.9645368 50.3482669 50.7922232 50.8350444 51.3113823 51.6528141
##  [97] 52.6344343 52.8019273 53.0151463 53.5625808
```

```
#The code is calculating cumulative sum which can be done using cumsum()
all.equal(cumsum(x), out)
```

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

```
##########################################################################################################
#3
output <- vector("integer", 0)
```

```r
for (i in seq_along(x)) {
  output <- c(output, lengths(x[[i]]))
}
output
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```r
#To discuss the performance effect, we are definig two functions and use the microbenchmark package
#to compare the time
add_to_vector <- function(n) {
  output <- vector("integer", 0)
  for (i in seq_len(n)) {
    output <- c(output, i)
  }
  output
}
add_to_vector_2 <- function(n) {
  output <- vector("integer", n)
  for (i in seq_len(n)) {
    output[[i]] <- i
  }
  output
}
library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.2.3
```

```r
timings <- microbenchmark(add_to_vector(10000), add_to_vector_2(10000), times = 3)
timings
```

```
## Unit: microseconds
##                    expr     min        lq     mean   median       uq      max
##    add_to_vector(10000) 99724.5 103697.55 105769.4 107670.6 108791.9 109913.2
##  add_to_vector_2(10000)   207.6    212.85    986.8    218.1   1376.4   2534.7
##  neval
##      3
##      3
```

```r
#From the results we can say that the pre-allocated vector performs much faster. However, the longer th
#and the bigger the objects, the more that pre-allocation will outperform appending

###################################################################################################
#Part2
#RMarkdown is widely appreciated for its ability to integrate code, text, and visualizations in a singl
#I find it valuable for creating reproducible and dynamic reports, especially in statistical modeling p
#I also appreciate the ability to render the document in various formats (e.g., HTML, Word, PDF)
#allowing for flexibility and customization.
#Once missing feature which I came across is that the feacture facilitating collaborative writing and e
#which makes beneficial while working in teams.
```