Skills Problem Set VI

Earnest Salgado

25/05/2021

```
library(tidyverse)
library(lubridate)
```

This submission is my work alone and complies with the 30535 integrity policy.

Add your initials to indicate your agreement: **ES**

Add your collaborators: **GATW**

Late coins used this pset: 0. Late coins left: 6.

1 Functions (15 points)

1. Write a function that transforms a vector c("a", "b", "c") into a string, a, b, and c (including the and!!). Think carefully about what the function should do if given a vector of length 1, 2, or 3.

```
str_commasep <- function(x, delim = ",") {
  n <- length(x)
  if (n == 0) {
    ""
  } else if (n == 1) {
    x
  } else if (n == 2) {
    str_c(x[[1]], "and", x[[2]], sep = " ")
  } else {
    not_last <- str_c(x[seq_len(n - 1)], delim)
    last <- str_c("and", x[[n]], sep = " ")
    str_c(c(not_last, last), collapse = " ")
  }
}
```

We can test that our function is doing what we want with various vectors of different lengths:

```
str_commasep("")
```

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

```
str_commasep("a")
```

```
## [1] "a"
```

```r
str_commasep(c("a", "b"))
```

```
## [1] "a and b"
```

```r
str_commasep(c("a", "b", "c"))
```

```
## [1] "a, b, and c"
```

```r
str_commasep(c("a", "b", "c", "d"))
```

```
## [1] "a, b, c, and d"
```

```r
str_commasep(c("a", "b", "c", "d", "e"))
```

```
## [1] "a, b, c, d, and e"
```

2. Write a function that given your birthday (as a date), returns how old you are in years

```r
age <- function(bday) {
  (bday %--% today()) %/% years(1)
}
age(ymd("1991-03-17"))
```

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

3. Statistical functions

a. Write a function to calculate the variance of a numeric vector

First I create a numeric vector x and then write my function that serves as the variance formula. Finally, we can check our 'variance' function against the calculated value from using var()

```r
y <- c(3, 6, 9, 17, 19, 25, 6)

variance <- function(x, na.rm = TRUE) {
  n <- length(x)
  m <- mean(x, na.rm = TRUE)
  sq_err <- (x - m)^2
  sum(sq_err) / (n - 1)
}

var(y)
```

```
## [1] 67.47619
```

```r
variance(y)
```

```
## [1] 67.47619
```

b. Write a function to calculate the skewness of a numeric vector

```
skew <- c(1, 6, 19, 263)
skewness <- function(x, na.rm = FALSE) {
  n <- length(x)
  m <- mean(x, na.rm = na.rm)
  v <- var(x, na.rm = na.rm)
  (sum((x - m) ^ 3) / (n - 2)) / v ^ (3 / 2)
}

skewness(skew)
```

```
## [1] 1.484215
```

Since there are multiple definitions for skewness, we could alternatively calculate it with this function:

```
skewness2 <- function(x) {
    n <- length(x)
    mean_x <- mean(x)
    sd_x <- sqrt(sum((x - mean_x)^2) / (n))
    z <- (x - mean_x) / sd_x

skewness2 <- sum(z^3) / n
skewness2
}
print(skewness2(skew))
```

```
## [1] 1.14255
```

c. Use summarize_if() to calculate the mean, variance, and skewness of all numeric columns in the diamond dataset. Then, tidy the table so we have one row for each variable. (Hint: summarize_if() takes a boolean in the first position and a named vector of functions in the second position.)

```
summarizeif <- function(x)  {
      diamond_mean <- summarise_if(x, is.numeric, mean, na.rm=TRUE)
      diamond_var <- summarise_if(x, is.numeric, variance, na.rm=TRUE)
      diamond_skw <- summarise_if(x, is.numeric, skewness, na.rm=TRUE)
      rbind(diamond_mean, diamond_var, diamond_skw)
}
summarizeif(diamonds)
```

```
## # A tibble: 3 x 7
##   carat   depth  table       price     x     y     z
##   <dbl>   <dbl>  <dbl>       <dbl> <dbl> <dbl> <dbl>
## 1 0.798 61.7      57.5     3933.   5.73   5.73 3.54
## 2 0.225    2.05  4.99  15915629.   1.26   1.30 0.498
## 3 1.12  -0.0823  0.797        1.62 0.379  2.43 1.52
```

4. Rename the following functions to better reflect their purpose

The function 'f' identifies whether each element of the character vector `nchar` starts with a common string, in this case it is `prefix`. You could rename this as has_prefix. The function 'g' drops the last element, so a better name could be drop_last.

```r
f <- function(string, prefix){
  str_sub(string, 1, nchar(prefix)) == prefix
}

g <- function(x){
  if (length(x) <= 1)return(NULL)
  x[-length(x)]
}
```

Examples:

```r
has_prefix <- f
drop_last <- g

has_prefix(c("morning", "sun", "moon", "yellow"), "mo")
```

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

```r
drop_last(c(0, 1, 2, 3, 5))
```

```
## [1] 0 1 2 3
```

5. Write a greeting function that says "good morning", "good afternoon", or "good evening", depending on the time of day. (Hint: use a time argument that defaults to lubridate::now(). That will make it easier to test your function.)

```r
greetings=function(time=lubridate::now())
  {
}


greeting <- function(time_now = lubridate::now()) {
  hour_now <- lubridate::hour(time_now)

  if (hour_now < 12) {
    "good morning"
  } else if (hour_now < 18) {
    "good afternoon"
  } else {
    "good night"
  }
}
greeting()
```

```
## [1] "good night"
```

2 For Loops (35 points)

1. Write for loops to

a. Compute the mean of every column in mtcars

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

```
##        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
```

b. Compute the number of unique values in each column of mpg

```
 data("mpg")
    mpg_uniq <- vector("double", ncol(mpg))
    names(mpg_uniq) <- names(mpg)
    for (i in names(mpg)) {
      mpg_uniq[i] <- n_distinct(mpg[[i]])
    }
    mpg_uniq
```

```
## manufacturer        model        displ         year          cyl        trans
##           15           38           35            2            4           10
##          drv          cty          hwy           fl        class
##            3           21           27            5            7
```

c. Generate 10 random points distributed poissons (rpois) for each $\lambda = 1, 3, 10, 30$ and 100. Think about the output, sequence, and body before you start writing the loop.

```
lambda_vector <- c(1, 3, 10, 30, 100)

rpois_fct <- matrix("double", nrow = 10, ncol = 5)
for (i in seq_along(lambda_vector)) {
  rpois_fct[,i] <- rpois(10, lambda = lambda_vector[[i]])
}
rpois_fct
```

```
##       [,1] [,2] [,3] [,4] [,5] 
##  [1,] "1"  "3"  "12" "22" "102"
##  [2,] "0"  "3"  "11" "37" "76" 
##  [3,] "2"  "3"  "9"  "24" "86" 
##  [4,] "0"  "1"  "9"  "26" "93" 
##  [5,] "0"  "5"  "12" "34" "103"
##  [6,] "2"  "3"  "6"  "31" "100"
##  [7,] "0"  "5"  "12" "26" "78" 
##  [8,] "2"  "4"  "12" "29" "112"
##  [9,] "1"  "4"  "5"  "27" "119"
## [10,] "0"  "2"  "14" "30" "101"
```

2. Imagine you have a directory full of CSV files that you want to read in. files <- dir(“data/”, pattern = “\.csv$”, full.names = TRUE), and now want to read each one with read_csv(). Write a for loop that will load them into a single data frame (you do not need to run anything just write code)

```
files <- dir("data/", pattern = "\\.csv$", full.names = TRUE)
files
```

```
## character(0)
```

```
df_list <- vector("list", length(files))
for (i in seq_along(files)) {
  df_list[[i]] <- read_csv(files[[i]])
}

print(df_list)
```

```
## list()
```

```
df <- bind_rows(df_list)
```

3. Write a function that prints the mean of each numeric column in a data frame, along with its name. For example show_mean(iris) would print:

```
# show_mean(iris)
## [1] "Sepal.Length : 5.84333333333333"
## [1] "Sepal.Width : 3.05733333333333"
## [1] "Petal.Length : 3.758"
## [1] "Petal.Width : 1.19933333333333"
```

```
show_mean <- function(df, digits = 2) {

  maxstr <- max(str_length(names(df)))
  for (nm in names(df)) {
    if (is.numeric(df[[nm]])) {
      cat(
        str_c(str_pad(str_c(nm, ":"), maxstr + 1L, side = "right"),
          format(mean(df[[nm]]), digits = digits, nsmall = digits),
          sep = " "
        ),
        "\n"
      )
    }
  }
}

show_mean(iris)
```

```
## Sepal.Length: 5.84
## Sepal.Width:  3.06
## Petal.Length: 3.76
## Petal.Width:  1.20
```

```
names_mean <- function(df) {
  df2 <- (select_if(df, is.numeric))

  output <- vector("double", ncol(df2))

  for (i in seq_along(df2)){
    v_names <- names(df2[,i])
    v_mean <- mean(df2[[i]], na.rm = TRUE)
    output [[i]]<- str_c(v_names, v_mean, sep = " : ")
  }
  output
}

(names_mean(mpg))
```

```
## [1] "displ : 3.47179487179487" "year : 2003.5"
## [3] "cyl : 5.88888888888889"   "cty : 16.8589743589744"
## [5] "hwy : 23.4401709401709"
```

4. Write code that uses one of the map function to:

a. Compute the mean of every column in mtcars

```
map_dbl(mtcars, mean)
```

```
##        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
```

b. Compute the number of unique values in each column in mpg

Out of curiosity I pulled the types of each variable. I believe we can also use map_dbl(iris, n_distinct) to return number of unique values.

```
map_chr(mpg, typeof)
```

```
## manufacturer        model        displ         year          cyl        trans
##  "character"  "character"     "double"    "integer"    "integer"  "character"
##          drv          cty          hwy           fl        class
##  "character"    "integer"    "integer"  "character"  "character"
```

```
map_int(iris, n_distinct)
```

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

c. Generate 10 random poissons (rpois) for each $\lambda = 1$, 3, 10, 30 and 100

```
v_pois <- map(mtcars, ~ rpois(10, lambda = 100))
v_pois
```

```
## $mpg
##  [1] 112 105 100 105  99 105 117  98 110  95
##
## $cyl
##  [1]  94  97  93 101  94  95 118 107 117 110
##
## $disp
##  [1]  77 101 108  94 106 103 103 102 115  95
##
## $hp
##  [1] 111 102 118 105  80 111  99 111  95 104
##
## $drat
##  [1] 107  96  89  97 115 101  94 103 136 128
##
## $wt
##  [1] 107 103 108  98  82  99 113 114 111 115
##
## $qsec
##  [1] 111  87 113  92 109  90  85 118 100  88
##
## $vs
##  [1]  99  96  78  88  85  97 103 106  94 107
##
## $am
##  [1]  93 101  91 100 100  96 114  95 111 102
##
## $gear
##  [1]  86 101 106  90  79  83  98 104  97  90
##
## $carb
##  [1]  95 106 105  94 103  88 110 102  90 106
```

5. Repeat question 2 using the map function. (The csv one)

```
files <- dir("data/", pattern = "\\.csv$", full.names = TRUE)
files
```

```
## character(0)
```

```
read_files <- map_dbl(files, ~read.csv)
```

6. What happens when we use the map functions on vectors that aren't lists. Use the following vector for the next section:

```
five_squares <- (1:5)^2
five_squares
```

```
## [1]  1  4  9 16 25
```

a. Describe the output of using on a list map(list(five_squares),rnorm). Explain why the output turns out this way.

The output is this way because five_squares is no longer being read as a vector, but as a list. This means the function rnorm will only be run 5 times, through the list of objects within five_squares.

```
map(list(five_squares),rnorm)
```

```
## [[1]]
## [1] -0.9360600  0.3754579  0.9440346  1.2040501 -1.1324263
```

b. What does map(five_squares,rnorm) do? Why?

This output computes normal distributions for the vector five_squares. As it generates, it takes each object as its parameter. So at five_squares[[1]], it is only 1 so the list is one value long. It continues until finally at five_squares[[5]], it identifies n = 25 so that particular list is 25 values long.

```
map(five_squares,rnorm)
```

```
## [[1]]
## [1] 0.479778
##
## [[2]]
## [1] -1.5463709  0.2053481  2.4115110  0.9944145
##
## [[3]]
## [1] -0.6293438 -0.4855667 -0.4016000 -1.5007529  0.9002636 -1.0726673 -0.3959066
## [8] -1.0897849 -0.1793170
##
## [[4]]
##  [1]  0.61945407  1.15324207  0.46566991  0.92522463 -1.17900892 -0.93223092
##  [7]  0.76150423 -0.46810149 -1.57377214 -0.02570065 -0.29390585 -0.57311813
## [13] -0.95508679 -0.77931944 -0.03056555  1.44675356
##
## [[5]]
##  [1] -0.30983088 -0.33387857 -0.45200444 -0.51825385 -0.23014334  0.89301074
##  [7]  0.02674961  0.37785759  0.22115265 -0.49013335 -1.09226317  0.37133598
## [13] -0.14947942 -1.49296485 -1.36129355 -0.30661164  1.62641917 -0.24545710
## [19] -0.13274726 -0.36592010  0.22966729  1.54780260  0.90549867  0.08441285
## [25]  1.02123519
```

c. What does map(five_squares, rnorm, n = 5) do? Why?

This output is the list of vectors rnorm(five_squares[[1]], n=5), rnorm(five_squares[[2]], n=5), rnorm(five_squares[[3]], n=5), rnorm(five_squares[[4]], n=5), rnorm(five_squares[[5]], n=5). It will perform the operation rnorm at each object in the vector, in our case 1 through 5, thus why its returning an output n = 5 times at each object.

```r
map(five_squares, rnorm, n = 5)
```

```
## [[1]]
## [1] 1.59598235 3.18299583 0.02430941 1.44055317 1.38595167
##
## [[2]]
## [1] 2.986210 3.882378 3.797465 5.595936 2.489471
##
## [[3]]
## [1] 8.709307 9.590686 8.574969 7.909066 9.206858
##
## [[4]]
## [1] 16.94049 15.45705 16.98997 15.90463 17.10969
##
## [[5]]
## [1] 24.26358 25.68863 26.66159 25.07138 23.19359
```