Homework Assignment #2

Matt Kline

February 5, 2017

Part 1 Getting Started

1.)

A. Use a for() loop to calculate the sum $\sum_{j=1}^{100} j^2 = 1^2 + 2^2 + ... + 100^2$

```
result <- 0
for (j in 1:100){
  result <- j^2 + result
}
print(result)

## [1] 338350
```

B. Calculate the sum without using a for() loop.

```
result <- sum(c(1:100)^2)
print(result)

## [1] 338350
```

C. Calculate $\sum_{j=1}^{100} j^2 = 1^2+2^2+...+100^2$ and compare with n(n+2)(2n+1)/6 for n = 100, 200, 400, 800

```
for (n in 100:800){
  resultSum <- sum(c(1:n)^2)
  resultForm <- (n*((n+1)*(2*n+1)))/6

  if (n == 100 || n == 200 || n == 400 || n == 800) {
    print(resultSum)
    print(resultForm)
  }
}

## [1] 338350
## [1] 338350
## [1] 2686700
## [1] 2686700
## [1] 21413400
## [1] 21413400
## [1] 170986800
## [1] 170986800
```

2.) Let h(x) = $\sum_{i=1}^{n} x^i = 1 + x + x^2 ... + x^n$

A. Write a R function to calculate h(x,n) using a for() loop.

```
result <- 0
h <- function(x,n){
  result <- sum(result + x^(c(1:n)))
  print(result)
}
```

B. The function h(x,n) is the finite sum of a so-called geometric series. it can be shown that h(x,n) can be calculted explicitly by the following formula (for x $\neq$ 1)

```
h(0.3, 55)

## [1] 0.4285714

h(6.6, 8)

## [1] 4243335
```

C. The function h(x,n) is the finite sum of a so-called geometric series. it can be shown that h(x,n) can be calculted explicitly by the following formula (for x $\neq$ 1)

```
h(0.3, 55)

## [1] 0.4285714

h(6.6, 8)

## [1] 4243335
```

Part 2 Vectors

3.) The following are ovsercations of incoming solar radiation at a greenhouse:

$$11.1, 10.6, 6.3, 8.8, 10.7, 11.2, 8.9, 12.2$$

A. Assign the data to a cector called sr

```
sr <- c(11.1, 10.6, 6.3, 8.8, 10.7, 11.2, 8.9, 12.2)
```

B. Find the mean, median, and standard deviation of the radiation observations.

```
mean(sr)

## [1] 9.975

median(sr)

## [1] 10.65

sd(sr)

## [1] 1.877498
```

C. Add 10 to each observation of sr, and assign the results to sr10. Find the mean, median, and standard deviation of sr10. Which statistics change, and by how much?

```
sr10 <- sr + 10

mean(sr10)

## [1] 19.975

median(sr10)

## [1] 20.65

sd(sr10)

## [1] 1.877498
```

The only statistic that did not change was Standard Deviation. The rest of were all changed by 10.

D. Multiply each observation by -2, and assign the results to srm2. Find the mean, median, and standard deviation of sr10. How do the statistics change now?

```
srm2 <- sr * -2

mean(srm2)

## [1] -19.95

median(srm2)

## [1] -21.3

sd(srm2)

## [1] 3.754997
```

All of the statistics were changed. The Mean and Median were changed by the multiple -2, while the Standard Deviation was doubled.

E. There are two formulas commonly used for the standard deviation of a set of data:

$$\sqrt{((1/n)*\sum_{i=1}^{n}(x_i - x)^2)} \text{ and } \sqrt{((1/(n-1))*\sum_{i=1}^{n}(x_i - x)^2)}$$

```
x <- sr
n <- length(sr)

sd(x)

## [1] 1.877498

sqrt(sum((x - mean(x))^2)/(n))
```

```
## [1] 1.756239

sqrt(sum((x - mean(x))^2)/(n-1))

## [1] 1.877498
```

Based on the results from the 3 versions tested I can conclude that R uses the n-1 format to calculate Standard Deviation.

4.) The data below are part of a table of statistical data on the states of the United States: population in thousands, percent illiteracy, and murders per 100,000 population. First create the following vectors:

```
illit <- c(2.1, 1.5, 1.8, 1.9, 1.1, 0.7, 1.1, 0.9, 1.3, 2.0)
murder <- c(15.1, 11.3, 7.8, 10.1, 10.3, 6.8, 3.1, 6.2, 10.7, 13.9)
pop <- c(3615, 365, 2212, 2110, 21198, 2541, 3100, 579, 8277, 4931)
```

A. Compute the mean of the illiteracy data and the derviations $x_i - x$ away from the mean.

```
mean(illit)

## [1] 1.44

illit - mean(illit)

##  [1]  0.66  0.06  0.36  0.46 -0.34 -0.74 -0.34 -0.54 -0.14  0.56
```

B. Compute the medians of murder and illit. Are the medians different thatn the means?

```
mean(illit)

## [1] 1.44

median(illit)

## [1] 1.4

mean(murder)

## [1] 9.53

mean(murder)

## [1] 9.53
```

No the median and the means are the same.

C. The function sort() returns all teh values in a vector sorted smallest to largest. Look at the sorted values for murder and compart those to median(murder).

```
sort(murder)

##  [1]  3.1  6.2  6.8  7.8 10.1 10.3 10.7 11.3 13.9 15.1
```

```
median(murder)

## [1] 10.2
```

By sorting the data you can clearly see that the median falls right inbetween the 2 center values of the sorted list.

D. Convert the murder rate data to number of murders in each of the 10 states.

```
total <- (pop * 1000) * (murder / 100000)
print(total)

##  [1]  545.865   41.245  172.536  213.110 2183.394  172.788   96.100
##  [8]   35.898  885.639  685.409
```

5.) What is the value of the expression

```
1:7 + 1:2

## Warning in 1:7 + 1:2:  longer object length is not a multiple of shorter object
length

## [1] 2 4 4 6 6 8 8
```

This expression is computed as 1+1 2+2 3+1 4+2 5+1 6+2 7+1

6.) Use the following command to create a character vector called queue representing a supermarket queue with Steve first in line.

```
queue <- c("Steve", "Russell", "Alison", "Liam")
print(queue)

## [1] "Steve"   "Russell" "Alison"  "Liam"
```

i Barry arrives

```
queue[length(queue) + 1] <- "Barry"
print(queue)

## [1] "Steve"   "Russell" "Alison"  "Liam"    "Barry"
```

ii Steve is served so he leaves

```
queue <- queue[-1]
print(queue)

## [1] "Russell" "Alison"  "Liam"    "Barry"
```

iii Pam arrives and goes to the front of the line

```
queue <- c("Pam", queue)
print(queue)

## [1] "Pam"     "Russell" "Alison"  "Liam"    "Barry"
```

iv Alison gets impacient and leaves

```
queue <- queue[-which(queue == "Alison")]
print(queue)

## [1] "Pam"     "Russell" "Liam"    "Barry"
```

7.) Replace every third value of a vector x with its negative value

```
x <- c(1:100)
i <- 1
while(i < 100) {
  x[i] <- -x[i]
  i <- i+3
}
```

8.) Create a vector of 123 values (equi-spaced) between 2.1 and 3.75.

```
?seq

## starting httpd help server ...
##  done

seq(2.1, 3.75, length.out = 123)

##   [1] 2.100000 2.113525 2.127049 2.140574 2.154098 2.167623 2.181148
##   [8] 2.194672 2.208197 2.221721 2.235246 2.248770 2.262295 2.275820
##  [15] 2.289344 2.302869 2.316393 2.329918 2.343443 2.356967 2.370492
##  [22] 2.384016 2.397541 2.411066 2.424590 2.438115 2.451639 2.465164
##  [29] 2.478689 2.492213 2.505738 2.519262 2.532787 2.546311 2.559836
##  [36] 2.573361 2.586885 2.600410 2.613934 2.627459 2.640984 2.654508
##  [43] 2.668033 2.681557 2.695082 2.708607 2.722131 2.735656 2.749180
##  [50] 2.762705 2.776230 2.789754 2.803279 2.816803 2.830328 2.843852
##  [57] 2.857377 2.870902 2.884426 2.897951 2.911475 2.925000 2.938525
##  [64] 2.952049 2.965574 2.979098 2.992623 3.006148 3.019672 3.033197
##  [71] 3.046721 3.060246 3.073770 3.087295 3.100820 3.114344 3.127869
##  [78] 3.141393 3.154918 3.168443 3.181967 3.195492 3.209016 3.222541
##  [85] 3.236066 3.249590 3.263115 3.276639 3.290164 3.303689 3.317213
##  [92] 3.330738 3.344262 3.357787 3.371311 3.384836 3.398361 3.411885
##  [99] 3.425410 3.438934 3.452459 3.465984 3.479508 3.493033 3.506557
## [106] 3.520082 3.533607 3.547131 3.560656 3.574180 3.587705 3.601230
## [113] 3.614754 3.628279 3.641803 3.655328 3.668852 3.682377 3.695902
## [120] 3.709426 3.722951 3.736475 3.750000
```

9.) Use rep() to create a vector of elements 1, 7, 1, 7, ... , 1, 7 of length 140.

```
rep(c(1,7), times = 70)

##   [1] 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1
##  [36] 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7
##  [71] 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1
## [106] 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7 1 7
```

10.) Use rep() to create a vector of elements of length 140 where the first 70 elements are 1 and the remaining 70 are 7.

```
rep(c(1,7), times = c(70, 70))

##   [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
##  [36] 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
##  [71] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
## [106] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```

11.) Using rep() and/or seq() as needed, create the following three vectors

```
rep(0:4, times = c(5,5,5,5,5))

##  [1] 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4

rep(1:5, times = 5)

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

c(1:5, 2:6, 3:7, 4:8, 5:9)

##  [1] 1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5 6 7 8 9
```

12.)

A. Write commands that create three vectors:

```
TrueAndMissing <- c(TRUE, NA)
TrueAndMissing <- sample(TrueAndMissing, 25, T)
print(TrueAndMissing)

##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE   NA   NA TRUE   NA TRUE   NA   NA
## [15] TRUE   NA   NA TRUE TRUE TRUE TRUE   NA   NA   NA TRUE

FalseAndMissing <- c(FALSE, NA)
FalseAndMissing <- sample(FalseAndMissing, 25, T)
print(FalseAndMissing)

##  [1]    NA FALSE    NA FALSE    NA    NA FALSE    NA FALSE    NA FALSE
## [12]    NA FALSE    NA FALSE    NA FALSE    NA    NA FALSE FALSE FALSE
## [23] FALSE    NA FALSE

Mixed <- c(TRUE, FALSE, NA)
Mixed <- sample(Mixed, 25, T)
print(Mixed)
```

```
##  [1] FALSE    NA    NA  TRUE    NA    NA FALSE    NA FALSE FALSE  TRUE
## [12]    NA FALSE    NA FALSE FALSE FALSE    NA  TRUE    NA    NA  TRUE
## [23]  TRUE    NA    NA
```

B. Apply the functions any( ) and all() to each of the vectors of part a and report the results

```
any(TrueAndMissing)

## [1] TRUE

all(TrueAndMissing)

## [1] NA

any(FalseAndMissing)

## [1] NA

all(FalseAndMissing)

## [1] FALSE

any(Mixed)

## [1] TRUE

all(Mixed)

## [1] FALSE
```

13.) Look at the help file for median() by tying:

```
?median()
```

A. What's the default value of na.rm?

   The default value for na.rm is FALSE.

B. According to the help file, what's the effect of setting na.rm = TRUE in a call to median()?

   Setting na.rm to TRUE will remove all NA values prior to taking the median.

C. Consider the vector and write a command that will calculate the median of the non NA values.

```
x <- c(4.1, 4.3, 4.7, 3.4, 3.9, 5.6, 4.6, NA, 6.6, 5.1, 4.3, 4.9, NA, 4.4, 6.1)
median(x, T)

## [1] 4.6
```

14.) With the definition

```
x <- c(1, NA, -2)
```

A. What would you expect for the result of the expressions:

```
1 + NA - 2

## [1] NA

length(x)

## [1] 3

sum(x)

## [1] NA

mean(x)

## [1] NA

mean(x, na.rm = T)

## [1] -0.5

median(x)

## [1] NA

median(x, na.rm = T)

## [1] -0.5
```

I would expect length to give 3. The sum, mean, and median without parameters to give NA. The mean with parameters should return -0.5 and the median with parameters should return -0.5 also.

B. How would you compute the sum of the non-NAs in x?

```
sum(x, na.rm = T)

## [1] -1
```

15.) Consider the vector:

```
u <- c(12, 4, 8, NA, 9, NA, 7, 12, 13, NA, 10)
```

Use is.na(), square brackets [ ], and ! (R's version of "not") to write a command (or commands) that extract all the non-missing values (non-NA's) from u.

```
for(i in 1:11) {
  if (is.na(u[i]))
    u <- u[-i]
}
```