Homework Assignment #2

Matt Kline

February 9, 2017

Part 1 Vectors:

1.) Consider again the data on populations, illiteracy rates, and murder rates for 10 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. Use rev() to print the elements of illit in reverse order.

```
rev(illit)

##  [1] 2.0 1.3 0.9 1.1 0.7 1.1 1.9 1.8 1.5 2.1
```

b. Print the elements of illit that are greater than the median illiteracy rate. Use median() to get the median illiteracy rate.

```
median(illit)

## [1] 1.4

illit[illit[] > median(illit)]

## [1] 2.1 1.5 1.8 1.9 2.0
```

c. Print the elements of murder for which the illiteracy rate is greater than its median.

```
murder[illit[] > median(illit)]

## [1] 15.1 11.3  7.8 10.1 13.9
```

2.) Consider again the data on murder rates from the previous exercise.

a. Write a command involving which() that determines the indices of the murder rates that are greater than 12.

```
which(murder[] > 12, TRUE)

## [1]  1 10
```

b. The following command does the same thing as which(murder > 12):

```
(1:10)[murder >12]

## [1]  1 10
```

Why do you think the parentheses are included in the expression? Experiment a little.

I think that the parentheses are included to show that [murder > 12] needs to be applied to all 10 values.

3.) Consider the data vector:

```
x <- c(-3, -2, 0, 0, 4, 5, 9, 0, -6, 7, -2, 8)
```

Write a command involving sum() and == that counts the number of elements of x that are equal to 0.

```
sum(x == 0)

## [1] 3
```

Part 2 Matrices and Arrays:

4.) Create the following matrix in three different ways:

a. Using matrix()

```
X <- matrix(c(1:5), nrow = 5, ncol = 8)
print(X)

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

b. Using rbind()

```
rbind(rep(1, times = 8),
      rep(2, times = 8),
      rep(3, times = 8),
      rep(4, times = 8),
      rep(5, times = 8))

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

c. Using cbind()

```
cbind(1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5, 1:5)

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

5.)

a. Write a command that uses apply() to find the minimum value in each row of a matrix X.

```
apply(X, MARGIN = 1, min)

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

b. Write a command that uses apply() to sort each column of a matrix X.

```
apply(X, MARGIN = 2, sort)

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

6.) state.x77 is a dataset that is supplied with R and stored in the form of a matrix. It contains information about the population, income, and other factors for each US state. You can see its values by typing its name, just as you would with datasets that you create yourself:

```
#state.x77
```

a. Write a command involving square brackets [ ] that prints the first 5 rows of state.x77.

```
state.x77[1:5,]

##            Population Income Illiteracy Life Exp Murder HS Grad Frost
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20
## Alaska            365   6315        1.5    69.31   11.3    66.7   152
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65
## California      21198   5114        1.1    71.71   10.3    62.6    20
##              Area
## Alabama     50708
## Alaska     566432
## Arizona    113417
## Arkansas    51945
## California 156361
```

b. Write a command that determines how many rows state.x77 has.

```
nrow(state.x77)

## [1] 50
```

c. Write a command involving square brackets [ ] that extracts from state.x77 the rows corresponding to states whose graduation rates are below 50

```
state.x77[state.x77[1:50, 6] < 50,]

##                Population Income Illiteracy Life Exp Murder HS Grad Frost
## Alabama              3615   3624        2.1    69.05   15.1    41.3    20
## Arkansas             2110   3378        1.9    70.66   10.1    39.9    65
## Georgia              4931   4091        2.0    68.54   13.9    40.6    60
## Kentucky             3387   3712        1.6    70.10   10.6    38.5    95
## Louisiana            3806   3545        2.8    68.76   13.2    42.2    12
## Mississippi          2341   3098        2.4    68.09   12.5    41.0    50
## Missouri             4767   4254        0.8    70.69    9.3    48.8   108
## North Carolina       5441   3875        1.8    69.21   11.1    38.5    80
## Rhode Island          931   4558        1.3    71.90    2.4    46.4   127
## South Carolina       2816   3635        2.3    67.96   11.6    37.8    65
## Tennessee            4173   3821        1.7    70.11   11.0    41.8    70
## Texas               12237   4188        2.2    70.90   12.2    47.4    35
## Virginia             4981   4701        1.4    70.08    9.5    47.8    85
## West Virginia        1799   3617        1.4    69.48    6.7    41.6   100
##                  Area
## Alabama         50708
## Arkansas        51945
## Georgia         58073
## Kentucky        39650
## Louisiana       44930
## Mississippi     47296
## Missouri        68995
## North Carolina  48798
## Rhode Island     1049
## South Carolina  30225
## Tennessee       41328
## Texas          262134
## Virginia        39780
## West Virginia   24070
```

d. Find the mean life expectancy for states whose high school graduation rates are below 50

```
mean(state.x77[state.x77[1:50,6] < 50, 4])

## [1] 69.68071
```

e. Find the mean and standard deviation of each column.

```
mean(state.x77[,1])

## [1] 4246.42
```

```r
sd(state.x77[,1])
```

```
## [1] 4464.491
```

```r
mean(state.x77[,2])
```

```
## [1] 4435.8
```

```r
sd(state.x77[,2])
```

```
## [1] 614.4699
```

```r
mean(state.x77[,3])
```

```
## [1] 1.17
```

```r
sd(state.x77[,3])
```

```
## [1] 0.6095331
```

```r
mean(state.x77[,4])
```

```
## [1] 70.8786
```

```r
sd(state.x77[,4])
```

```
## [1] 1.342394
```

```r
mean(state.x77[,5])
```

```
## [1] 7.378
```

```r
sd(state.x77[,5])
```

```
## [1] 3.69154
```

```r
mean(state.x77[,6])
```

```
## [1] 53.108
```

```r
sd(state.x77[,6])
```

```
## [1] 8.076998
```

```r
mean(state.x77[,7])
```

```
## [1] 104.46
```

```r
sd(state.x77[,7])
```

```
## [1] 51.98085
```

```r
mean(state.x77[,8])
```

```
## [1] 70735.88
```

```r
sd(state.x77[,8])
```

```
## [1] 85327.3
```

1.) Write commands that:

a. Locate all rows of a matrix X that are all-zero. Hint: One way to do this is to write a function all.zero() that takes a vector argument x and checks whether all of its elements are equal to 0 or not, returning TRUE if they are and FALSE otherwise. The function all() may come in handy. Then use apply() to apply all.zero() to each row of the matrix.

```
set.seed(1)
X <- matrix(sample(0:1, size = 150, replace = TRUE), nrow = 30)
X
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    0    0    1    0    1
##  [2,]    0    1    0    0    0
##  [3,]    1    0    0    1    0
##  [4,]    1    0    0    1    0
##  [5,]    0    1    1    1    1
##  [6,]    1    1    0    1    0
##  [7,]    1    1    0    0    1
##  [8,]    1    0    1    0    0
##  [9,]    1    1    0    1    0
## [10,]    0    0    1    1    1
## [11,]    0    1    0    1    1
## [12,]    0    1    1    0    0
## [13,]    1    1    0    0    0
## [14,]    0    1    0    1    1
## [15,]    1    1    0    1    1
## [16,]    0    1    1    0    1
## [17,]    1    0    1    0    1
## [18,]    1    0    0    0    1
## [19,]    0    1    1    1    1
## [20,]    1    1    1    1    1
## [21,]    1    0    0    1    1
## [22,]    0    1    1    1    1
## [23,]    1    0    0    0    0
## [24,]    0    0    0    0    0
## [25,]    0    0    1    0    1
## [26,]    0    0    0    0    0
## [27,]    0    0    1    1    0
## [28,]    0    1    0    0    1
## [29,]    1    1    0    0    0
## [30,]    0    0    0    1    1
```

```
all.zero <- function(x) {
  return (all(x == 0))
}

apply(X, MARGIN = 1, all.zero)
```

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

b. Find the first all-zero row of a matrix X.

```
X[(apply(X, MARGIN = 1, all.zero)),]

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
```

c. Compute the mean of the rows in which nonzero elements appear.

```
mean(X[(!apply(X, MARGIN = 1, all.zero))])

## [1] 0.5285714
```

d. Compute the mean of the nonzero elements in each row.

```
split <- X[!apply(X, MARGIN = 1, all.zero),]
for (i in 1:28) {
  print(mean(split[i,]))
}

## [1] 0.4
## [1] 0.2
## [1] 0.4
## [1] 0.4
## [1] 0.8
## [1] 0.6
## [1] 0.6
## [1] 0.4
## [1] 0.6
## [1] 0.6
## [1] 0.6
## [1] 0.4
## [1] 0.4
## [1] 0.6
## [1] 0.8
## [1] 0.6
## [1] 0.6
## [1] 0.4
## [1] 0.8
## [1] 1
## [1] 0.6
## [1] 0.8
## [1] 0.2
## [1] 0.4
## [1] 0.4
## [1] 0.4
## [1] 0.4
## [1] 0.4
```

e. Locate all rows for which the sum of the elements is odd.

```
X[((apply(X, MARGIN = 1, sum)) %% 2) != 0,]

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

```
##  [1,]    0    1    0    0    0
##  [2,]    1    1    0    1    0
##  [3,]    1    1    0    0    1
##  [4,]    1    1    0    1    0
##  [5,]    0    0    1    1    1
##  [6,]    0    1    0    1    1
##  [7,]    0    1    0    1    1
##  [8,]    0    1    1    0    1
##  [9,]    1    0    1    0    1
## [10,]    1    1    1    1    1
## [11,]    1    0    0    1    1
## [12,]    1    0    0    0    0
```

Part 3 Lists:

8.) The function lapply() is designed to execute a function, given as an argument FUN to lapply(), on each element of a list.

a. Use list() to construct a list containing the following three vectors:

```
v1 <- c(23, 18, 34, 14, 19, 22, 67, 37)
v2 <- c(0.1, 0.0, 3.0, 2.4, 4.1, 1.5, 1.2, 2.2)
v3 <- c("h", "b", "d", "c", "a", "f", "e", "g")
mylist <- list(v1, v2, v3)
```

b. Now use lapply() and sort() to sort each component of the list.

```
lapply(mylist, sort)

## [[1]]
## [1] 14 18 19 22 23 34 37 67
##
## [[2]]
## [1] 0.0 0.1 1.2 1.5 2.2 2.4 3.0 4.1
##
## [[3]]
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```