GEOG 714 - Assignment 1

Haoran Xu

Sep 15, 2024

```
a <- 10
b <- 35
c <- a + b

a <- c(3, 5, 2)
b <- c(2, 7, 7)
c <- a + b

d <- c(1, 4, 2, 3)
e <- c(5, 6, 4, 3)
f <- d * e
```

Q1. Write your own code that will add the elements in the vector d to the elements in vector f and put the result in a new vector g.

```
g <- d + f
```

```
j <- c(2, 3, 5, 4, 3, 2, 3, 6, 7)
k <- c(1, 4, 5, 2, 6, 7, 8, 9, 8)
m <- cbind(j, k)
m[,1]
```

Q2. Write your own code that will return the entire row 2 of matrix m.

```
m[2,]
```

```
m <- cbind(m,m[,1] + m[,2])
c1 <- '0'
c2 <- '5'
# c1+c2 # Error in c1 + c2 : non-numeric argument to binary operator
```

Q3. In your own words, describe what caused the error (one sentence)

c1 and c2 are not numbers but strings if written inside the single quotation marks.

```
a <- c(2, 5, 3, 2, 1)
b <- a[1:2]
```

Q4. Write code to subset this vector to include the values of 5 and 3 and put this in a new vector called c.

```
c <- a[2:3]
```

```
a <- c(100, 300, 200, 500, 200)
b <- log(a[1] + a[4]) # If you want to add the base: b <- log(a[1] + a[4], base = 10)
a <- runif(100, 0, 1) # Half open interval so you never get 0 and 1
hist(a)
```

```
summary(a)
```

Q5. If you were to compare results for the different students in the class, you would notice that each student would get different answers. Do some looking into the runif() function to see if you can figure out why. How can you change the code so that everyone gets the same answer? Write your reasoning here (up to three sentences).

Since runif() means creating random numbers so every time you do it it will produce different results. A possible solution is control the runif() function by introducing set.seed(). If we put in the same numbers in set.seed(), runif() would produce the same series of numbers every time we run.

```
set.seed(100)
m <- matrix(runif(1000), ncol = 10, nrow = 100) # the dufault value for runif is from 0 to 1
v <- c(3, 9, 1, 8, 0)
order(v) # "5 3 1 4 2" means the order ranked from lowest to highest

m_ordered <- m[order(m[,1]),] # [,1] means column 1 #ascending order
m_ordered <- m[order(-m[,1]),] # descending order
m_ordered <- m[floor(runif(100)*100)+1,]
m_ordered # you can notice row 59 and row 60 are the same.
```

Q6. In no more than 2 sentences, explain what the code to the right is doing.

"floor(run(100)*100)+1" means a random integar between 1 and 100. So the code means extract a random row from m 100 times and thus form a new matrix *m_ordered* of 100 rows.

```
sub1 <- m[m[,1] < 0.5]
sub2 <- m[m[,1] >= 0.5] # the numbers are even!

m[m[,1] > 0.5, 1] <- 1
m[m[,1] <= 0.5, 1] <- 0
```

Q7. Modify the code above to recode all items in all columns so that any value greater than 0.3 is coded as 0 and any other value is coded as 1

```
m <- ifelse(m > 0.3, 0, 1) # did not find better ways
```

```
# install.packages("curl")
library(curl)
```

```
## Using libcurl 8.7.1 with LibreSSL/3.3.6
```

```
# Import data from a website
data_csv <- read.csv(curl(
  "http://www.healthgeomatics.com/wp-content/uploads/2016/05/Canadian-populated-places.csv"
  ))

is.data.frame(data_csv) # verify if it's data frame (row-obersvations, column-variables)
data_csv$name <- as.factor(data_csv$name) # turn all chr to factor (catagorical - levels)
unique_names <- levels(data_csv$name)
# unique_names2 <- unique(data_csv$name) # unique() can deal with chr
length(unique_names)
```

Q8. Write your own code to put the output from the length function above into a new variable called number_unique

```
number_unique <- length(unique_names)
```

```
data_csv$named <- as.character(data_csv$name)
Alvany <- data_csv[data_csv$name == "Albany",]

place_names <- data_csv$name # put the names of places in a new data object
```

```r
place_names_len <- nchar(as.character(place_names)) # count the length of place names
places <- data.frame(place_names, place_names_len) # create new data frames

places <- places[order(-places$place_names_len),] # sort the data frame in descending order
places <- places[order(places$place_names_len),] # sort the data frame in ascending order
# it's different in matrix like m[order(m[,1]),]

## cannot install the packages
# install.packages("data.table")
# library(data.table)
# aplaces <- as.data.frame(table(places$place_names))

places$cnt <- 1
aplaces <- aggregate(places$cnt, by=list(places$place_names), FUN=sum)
# rename the variables in the data and make it consistent with the ones generated by the table()
names(aplaces) <- c("Var1","Freq")

aplaces <- aplaces[order(-aplaces$Freq),]
```

Q9. Write your own code to create a new data frame called most_common that contains only the first row of the aplaces data frame.

```r
most_common <- aplaces[1, ]
```

```r
# random normal data
rand1 <- rnorm(100, 0, 1)  # normal distribution # this is vector
```

Q10. Write your own code to generate another vector of random numbers called rand2 with the same parameters as above.

```r
rand2 <- rnorm(100, 0, 1)
```

```r
the_data <- data.frame(cbind(rand1, rand2))

# reclassify variable
# it creates a new variable called "r1class"
the_data$r1class[the_data$rand1 > 0] <- "Positive"
the_data$r1class[the_data$rand1 < 0] <- "Negative"

# aggregate(): a powerful tool to do calculations in groups
dataagg <- aggregate(the_data$rand2, by=list(the_data$r1class), FUN=mean)
# "by = list" specifies that the data should be grouped by r1class. # "Fun" means function.
names(dataagg) <- c("class", "average_r2")

m <- matrix(ceiling(runif(9) * 10), nrow = 3) # integer between 1-10
```

```r
result1 <- apply(m, 1, sum) # 1 indicates row
result2 <- apply(m, 2, sum) # 2 indicates column
difference_max_min <- function(x){return(max(x) - min(x))}
result3 <- apply(m, 2, difference_max_min)
```

Q11. Modify the difference_max_min function so that it calculates the difference between the mean of a vector and the minimum value. Use the apply to call the function on the rows of matrix m.

```r
difference_mean_min <- function(x){return(mean(x) - min(x))}
result4 <- apply(m, 1, difference_mean_min)
```

Q12. Import data from: http://www.healthgeomatics.com/wp-content/uploads/2016/06/HockeySeasonData.csv

Q13. Keep only records for years after 2005p

Q14. Aggregate the points (PTS) by team (Team), using a sum function, and put result into a data frame called totalpoints

```r
data_csv2 <- read.csv(curl(
  "http://www.healthgeomatics.com/wp-content/uploads/2016/06/HockeySeasonData.csv"
  ))
data_after2005 <- data_csv2[data_csv2$Year >= 2005,]
totalpoints <- aggregate(data_after2005$PTS, by=list(data_after2005$Team), FUN=sum)
```