Lecture 6
Factor and Table

GEOG 489 SPRING 2020

Matrix

2-d looping

The goal is to write a loop that will go through all possible combinations of a matrix in 2d.

```
myMatrix <- matrix(seq(10,60,by=10),nrow=3,ncol=2)
for(current_col in seq(2))
{
    for(current_row in seq(3))
     {
        print(paste("the center cell value is:",
                            myMatrix[current_row,current_col]))
    }
}
myMatrix
```

Quiz 2

Please create a matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, and use the 2-d loop to change the elements of the first row to 10, and to change the elements of the second row to 20. So you will get a matrix $\begin{bmatrix} 10 & 10 & 10 \\ 20 & 20 & 20 \end{bmatrix}$

```
myMatrix <- matrix(c(1,2,3,4,5,6),nrow=2,byrow = TRUE)
for(col in c(1,2,3))
{
  for(row in c(1,2))
  {
            ifelse(row == 1, myMatrix[row,col] <- 10, myMatrix[row, col] <- 20)
  }
}
myMatrix
```

Factors and tables

Factors are the data objects which are used to categorize the data and store it as levels.

1) Factors and levels

x <- c(5,12,13,12) # Right now, this is continuous data.

xf <- factor(x) # convert this numerical vector into a categorical vector. This creates a new set of information which are the "levels" (unique factors)

Factors and tables

2) tapply() function: tapply, used with vectors, allows us to apply functions on a per-level basis

tapply(X,INDEX,FUN)

For instance, say we want to calculate the mean age of people from different political parties:

ages <- c(25,26,55,37,21,42)

affils <- c("R","D","D","R","U","D")

(D)emocrat, (R)epublican, (U)naffiliated

tapply(X=ages,INDEX=affils,FUN=mean)

Factors and tables

2) tapply() function: tapply, used with vectors, allows us to apply functions on a per-level basis

```
d <- data.frame(gender=c("M","M","F","M","F","F"),
                age=c(47,59,21,32,33,24),
                income=c(55000,88000,32450,76500,123000,45650))
```

Now, we want to see the mean income for four groups: Males under 25, males over 25, females under 25, females over 25:

```
# Let's make a categorical variable if someone is over 25:
d$over25 <- ifelse(d$age > 25, 1, 0)
```

Factors and tables

2) tapply() function: tapply, used with vectors, allows us to apply functions on a per-level basis

```
# Now, we want to see the mean income for four groups:

#  Males under 25, males over 25, females under 25, females over 25:

tapply(X=d$income,INDEX=list(d$gender,d$over25),mean)

# Notice that tapply figured out there were 4 groups, given the 2 factors (gender and over25).
```

Factors and tables

3) by() function: tapply() is designed for use with vectors, but what if we want to do something similar with a dataframe?

myData <- warpbreaks # a built in dataset

by(data=myData,INDICES=warpbreaks$wool,FUN=summary)

```
> myData
   breaks wool tension
1      26    A       L
2      30    A       L
3      54    A       L
4      25    A       L
5      70    A       L
6      52    A       L
7      51    A       L
8      26    A       L
9      67    A       L
10     18    A       M
```

```
> str(myData)
'data.frame':   54 obs. of  3 variables:
 $ breaks : num  26 30 54 25 70 52 51 26 67 18 ...
 $ wool   : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
 $ tension: Factor w/ 3 levels "L","M","H": 1 1 1 1 1 1 1 1 1 2 ...
```

Factors and tables

Tables (count data for factors): represent the number of occurrences of every unique factor

```
mytable <- table(data.frame(warpbreaks$wool,warpbreaks$tension))

# This counts all combinations of wool and tension


# Tables are a lot like matrices, so you can subset them as such:

mytable[1,1]

# and do arithmetic:

mytable+1
```

R Programming Structure

1) Loop over non-vector sets: We can coerce non vector sets into a list, then use lapply to loop through them

```
# Say we have two matrices:

u <- matrix(runif(6),ncol=2)

v <- matrix(runif(6),ncol=2)

# We want to perform a linear regression between columns 1 and 2 of each matrix,

lm(u[,2] ~ u[,1])

lm(v[,2] ~ v[,1])
```

R Programming Structure

1) Loop over non-vector sets: We can coerce non vector sets into a list, then use lapply to loop through them

```
# Say we have two matrices:

u <- matrix(runif(6),ncol=2)

v <- matrix(runif(6),ncol=2)

# So how do we do loop through each matrix?  With lapply, we'd do:

newlist=list(u=u,v=v)

lapply(X=newlist,FUN=function(x) {lm(x[,2] ~ x[,1])  })
```

R Programming Structure

2) if-else:

```
if(TRUE)
{
    # Do something.
} else
{
    # Do something else.
}
```

R Programming Structure

2) if-else:

```
r <- 3
if(r == 4)
{ # Execute if TRUE
    x <- 1
} else
{ # Execute if FALSE
    x <- 3
}
```

R Programming Structure

3) Boolean operators

```
# Element-wise boolean operators:
x & y    # Element-wise "AND" statement
x | y    # Element-wise "OR" statement
xor(x,y)   # Element-wise "XOR" (exclusive OR) statement
!x         # Element-wise NOT statement
```

R Programming Structure

3) Boolean operators

Notice that none of these could be used with an if() statement, because an if() statement must collapse down to a SINGLE TRUE.

```
x && y      # AND, only uses the first element of x and y
x || y      # OR, only uses the first element of x and y
```

R Programming Structure

4) Return() with > one object (or a complex object)

return() in a function can only be used once to return a single object.

If you want to return MULTIPLE objects, you need to store them in a list object.

R Programming Structure

4) Return() with > one object (or a complex object)

we have a function that takes an input x, and we want it to return two things: the square of x, and a matrix of all ones of dimension x by x.

```
myfunc <- function(x)
{
    x_squared <- x^2
    x_matrix <- matrix(1,nrow=x,ncol=x)
    # How do we return both of these?
    output <- list(x_squared=x_squared,x_matrix=x_matrix)
    return(output)
}
```

Assignment 1

Your goal is to write a function that takes two inputs:

x = a vector of numbers

d = a single value

The function should return a vector of numeric indices of all vector locations in which the element of x divided by d has no remainder.

Assignment 1 is due today, Feb. 11 at midnight.

Please submit your assignment on Compass 2g