Introduction to Computer Programming with R (FOR 6934)

Qing Zhao (School of Forest Resources & Conservation, qing.zhao@ufl.edu)
Daijiang Li (Department of Wildlife Ecology & Conservation, dli1@ufl.edu)
Denis Valle (School of Forest Resources & Conservation, drvalle@ufl.edu)

Class twelve

- Use colMeans() and several other functions on numeric matrices and data frames
- Use apply() function on numeric matrices and data frames
- Use tapply() function on data frames with multiple types of data

Functions for numeric matrics and data frames

- colMeans()
- rowMeans()
- colSums()
- rowSums()

Sample code

Mean of each column of a matrix

```
a <- matrix(c(1:12), nrow=3, ncol=4)
a
```

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

```
colMeans(a)
```

```
## [1]  2  5  8 11
```

Sample code

Mean of each row of a matrix

```
a
```

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

```
rowMeans(a)
```

```
## [1] 5.5 6.5 7.5
```

Sample code

Sums

```
a
```

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

```
colSums(a)
```

```
## [1]  6 15 24 33
```

```
rowSums(a)
```

```
## [1] 22 26 30
```

Sample code

These functions can be used on data frames of numeric values

```
dat <- data.frame(a)
class(dat)
```

```
## [1] "data.frame"
```

```
dat
```

```
##   X1 X2 X3 X4
## 1  1  4  7 10
## 2  2  5  8 11
## 3  3  6  9 12
```

```
colMeans(dat)
```

```
## X1 X2 X3 X4
##  2  5  8 11
```

```
rowSums(dat)
```

```
## [1] 22 26 30
```

Sample code

These functions can also be used on arrays with more than two dimensions

```
arr <- array(1:24, dim=c(3,4,2))
arr
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
```

```
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

Sample code

However the results are not always straightforward to interpret

```
colSums(arr)
```

```
##      [,1] [,2]
## [1,]    6   42
## [2,]   15   51
## [3,]   24   60
## [4,]   33   69
```

```
rowSums(arr)
```

```
## [1]  92 100 108
```

Sample code

When there are missing values

```
b <- a
b[2,c(2,4)] <- NA
b
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2   NA    8   NA
## [3,]    3    6    9   12
```

```
colMeans(b)
```

```
## [1]  2 NA  8 NA
```

Sample code

Use na.rm option to deal with missing values

```
b
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2   NA    8   NA
## [3,]    3    6    9   12
```

```
colMeans(b, na.rm=T)
```

```
## [1]  2  5  8 11
```

This concludes Class 12, Section 1

Please continue on to the next video

Situations when we need to use apply() function

- Use other maths functions (e.g. median())
- Deal with arrays

Sample code

Use apply() to calculate means of rows

```
a
```

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

```
apply(a, MARGIN=1, FUN=mean)
```

```
## [1] 5.5 6.5 7.5
```

```
rowMeans(a)
```

```
## [1] 5.5 6.5 7.5
```

Sample code

Use apply() to calculate means of columns

```
a
```

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

```
apply(a, MARGIN=2, FUN=mean)
```

```
## [1]  2  5  8 11
```

```
colMeans(a)
```

```
## [1]  2  5  8 11
```

Sample code

Use apply() to calculate medians of rows and variances of columns

```
a
```

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

```r
apply(a, MARGIN=1, FUN=median)
```

```
## [1] 5.5 6.5 7.5
```

```r
apply(a, MARGIN=2, FUN=var)
```

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

Sample code

Deal with missing values in apply()

```r
b
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2   NA    8   NA
## [3,]    3    6    9   12
```

```r
apply(b, MARGIN=1, FUN=sd)
```

```
## [1] 3.872983       NA 3.872983
```

```r
apply(b, MARGIN=1, FUN=sd, na.rm=T)
```

```
## [1] 3.872983 4.242641 3.872983
```

Sample code

Add options for complex functions

```r
a
```

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

```r
apply(a, MARGIN=2, FUN=quantile)
```

```
##      [,1] [,2] [,3] [,4]
## 0%    1.0  4.0  7.0 10.0
## 25%   1.5  4.5  7.5 10.5
## 50%   2.0  5.0  8.0 11.0
## 75%   2.5  5.5  8.5 11.5
## 100%  3.0  6.0  9.0 12.0
```

Sample code

Add options for complex functions, cont'd

```r
a
```

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

```r
apply(a, MARGIN=2, FUN=quantile, probs=c(.5, .05, .95))
```

```
##     [,1] [,2] [,3] [,4]
## 50%  2.0  5.0  8.0 11.0
## 5%   1.1  4.1  7.1 10.1
## 95%  2.9  5.9  8.9 11.9
```

Sample code

apply() can be used on numeric data frames

```r
dat
```

```
##   X1 X2 X3 X4
## 1  1  4  7 10
## 2  2  5  8 11
## 3  3  6  9 12
```

```r
apply(dat, MARGIN=2, FUN=range)
```

```
##      X1 X2 X3 X4
## [1,]  1  4  7 10
## [2,]  3  6  9 12
```

Sample code

apply() is more flexible for arrays

```r
arr
```

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

```r
apply(arr, MARGIN=1:2, FUN=sum)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   14   20   26   32
## [2,]   16   22   28   34
## [3,]   18   24   30   36
```

```
apply(arr, MARGIN=3, FUN=max)
```

```
## [1] 12 24
```

This concludes Class 12, Section 2

Please continue on to the next video

Calculate summary statistics from data frames with multiple types of data

- tapply()

Sample code

Read in NBA data

```
nba <- read.csv('c:/data/nba.csv')
head(nba, n=4)
```

```
##                  Team Win Lose Rank Conference
## 1      Boston Celtics  53   29    1    Eastern
## 2 Cleveland Cavaliers  51   31    2    Eastern
## 3     Toronto Raptors  51   31    3    Eastern
## 4  Washington Wizards  49   33    4    Eastern
```

Sample code

Calculate the number of teams for each conference

```
tab1 <- tapply(nba$Conference, INDEX=nba$Conference, FUN=length)
tab1
```

```
## Eastern Western
##      15      15
```

```
sum1 <- data.frame(Conference=names(tab1), Number_of_Teams=tab1)
sum1
```

```
##         Conference Number_of_Teams
## Eastern    Eastern              15
## Western    Western              15
```

Sample code

Calculate the mean and standard deviation of wins for each conference

```
tab2 <- tapply(nba$Win, INDEX=nba$Conference, FUN=mean)
sum2 <- data.frame(Conference=names(tab2), Mean_of_Wins=tab2)
sum2
```

```
##         Conference Mean_of_Wins
## Eastern    Eastern         39.6
## Western    Western         42.4
```

```
tab3 <- tapply(nba$Win, INDEX=nba$Conference, FUN=sd)
sum3 <- data.frame(Conference=names(tab3), SD_of_Wins=tab3)
sum3
```

```
##         Conference SD_of_Wins
## Eastern    Eastern   9.560335
## Western    Western  12.793972
```

Sample code

Put the results together to create a summary table

```
sum.out <- merge(sum1, sum2)
sum.out <- merge(sum.out, sum3)
sum.out
```

```
##   Conference Number_of_Teams Mean_of_Wins SD_of_Wins
## 1    Eastern              15         39.6   9.560335
## 2    Western              15         42.4  12.793972
```

Sample code

More effort to distinguish play-off and non-play-off teams

```
nba$Playoff <- ifelse(nba$Rank <= 8, 'play-off', 'non-play-off')
nba$Category <- paste(nba$Conference, nba$Playoff, sep=' / ')
head(nba, n=4)
```

```
##                  Team Win Lose Rank Conference  Playoff           Category
## 1      Boston Celtics  53   29    1    Eastern play-off Eastern / play-off
## 2 Cleveland Cavaliers  51   31    2    Eastern play-off Eastern / play-off
## 3     Toronto Raptors  51   31    3    Eastern play-off Eastern / play-off
## 4  Washington Wizards  49   33    4    Eastern play-off Eastern / play-off
```

Sample code

Calculate summary statistics

```
tab1 <- tapply(nba$Category, INDEX=nba$Category, FUN=length)
sum1 <- data.frame(Category=names(tab1), Number_of_Teams=tab1)
tab2 <- tapply(nba$Win, INDEX=nba$Category, FUN=mean)
sum2 <- data.frame(Category=names(tab2), Mean_of_Wins=tab2)
tab3 <- tapply(nba$Win, INDEX=nba$Category, FUN=sd)
sum3 <- data.frame(Category=names(tab3), SD_of_Wins=tab3)
sum.mat <- cbind(tab1, tab2, tab3)
sum.out <- data.frame(Category=row.names(sum.mat), sum.mat, row.names=NULL)
names(sum.out)[-1] <- c('Number_of_Teams', 'Mean_of_Wins', 'SD_of_Wins')
```

```
sum.out
```

```
##                 Category Number_of_Teams Mean_of_Wins SD_of_Wins
## 1 Eastern / non-play-off               7     31.71429   6.969321
## 2     Eastern / play-off               8     46.50000   4.956958
## 3 Western / non-play-off               7     31.42857   5.287001
## 4     Western / play-off               8     52.00000   8.815571
```

Sample code

Use loops and if else statement to do the same thing

```
variables <- c('Category', 'Win', 'Win')
functions <- c('length', 'mean', 'sd')
var.names <- c('Number_of_Teams', 'Mean_of_Wins', 'SD_of_Wins')

for (i in 1:length(functions)) {
  tab.temp <- tapply(nba[,variables[i]], INDEX=nba$Category, FUN=functions[i])
  sum.temp <- data.frame(Category=names(tab.temp), tab.temp)
  names(sum.temp)[2] <- var.names[i]
  if (i == 1) {
    sum.out <- sum.temp
  } else {
    sum.out <- merge(sum.out, sum.temp)
  }
}
```

```
sum.out
```

```
##                 Category Number_of_Teams Mean_of_Wins SD_of_Wins
## 1 Eastern / non-play-off               7     31.71429   6.969321
## 2     Eastern / play-off               8     46.50000   4.956958
## 3 Western / non-play-off               7     31.42857   5.287001
## 4     Western / play-off               8     52.00000   8.815571
```

Sample code

Use loops and if else statement to do the same thing, but with more statistics

```
variables <- c('Category', 'Win', 'Win', 'Win', 'Win', 'Win')
functions <- c('length', 'mean', 'sd', 'median', 'max', 'min')
var.names <- c('Number_of_Teams', 'Mean_of_Wins', 'SD_of_Wins',
               'Median_of_Wins', 'Maximum_of_Wins', 'Minimum_of_Wins')

for (i in 1:length(functions)) {
  tab.temp <- tapply(nba[,variables[i]], INDEX=nba$Category, FUN=functions[i])
  sum.temp <- data.frame(Category=names(tab.temp), tab.temp)
  names(sum.temp)[2] <- var.names[i]
  if (i == 1) {
    sum.out <- sum.temp
  } else {
    sum.out <- merge(sum.out, sum.temp)
```

```
  }
}
```

```
sum.out
```

```
##                 Category Number_of_Teams Mean_of_Wins SD_of_Wins
## 1 Eastern / non-play-off               7     31.71429   6.969321
## 2     Eastern / play-off               8     46.50000   4.956958
## 3 Western / non-play-off               7     31.42857   5.287001
## 4     Western / play-off               8     52.00000   8.815571
##   Median_of_Wins Maximum_of_Wins Minimum_of_Wins
## 1             31              41              20
## 2             46              53              41
## 3             32              40              24
## 4             51              67              41
```

Summary

- colMeans, rowMeans, colSums, and rowSums can be considered as convenient forms of apply()
- apply() can be used to apply maths functions other than mean() and sum() on numeric matrices and data frames
- tapply() can be used to apply maths functions on data frames with multiple types of data

Thank you and see you next class