#Apply Sapply Lapply Tapply Vapply

#Dr.Trilok Nath Pandey

#SCOPE,VIT

rm(list=ls())

#apply() function apply() takes Data frame or matrix as an input and gives output in vector, list or array. #Apply function in R is primarily used to avoid explicit uses of loop constructs. It is the most basic of all #collections that can be used over a matrices.

#This function takes 3 arguments:

#apply(X, MARGIN, FUN)

Here:

-x: an array or matrix

-MARGIN: take a value or

#range between 1 and 2 to

#define where to apply the

#function:

-MARGIN=1`: the

#manipulation is performed

#on rows

-MARGIN=2`: the manipulation

#is performed on columns

-MARGIN=c(1,2)`

#the manipulation is

#performed on rows and columns

-FUN: tells which function

#to apply. Built functions

#like mean,

#median, sum, min, max and

#even user-defined functions

#can be applied>

```
m1 <- matrix(C<-(1:10),
          nrow=5, ncol=6)
m1
a_m1 <- apply(m1, 1, sum)
a_m1
a_m2 <- apply(m1, 2, sum)
a_m2

#############################################################################

# lapply() function
# lapply() function is useful
#for performing operations on
#list objects
#and returns a list
#object of same length of
#original set.
#lappy() returns a list of
#the similar length as input
#list object,
#each element of which is
#the result of applying
#FUN to the corresponding
#element of list. Lapply
#in R takes list,
#vector or data frame as
#input and gives output
#in list.
# lapply(X, FUN)
```

```
# Arguments:
#   -X: A vector or an object
# -FUN: Function applied to
#each element of x
# l in lapply()
#stands for list.
#The difference between lapply()
#and apply() lies between the output return. The output of lapply()
#is a list. lapply() can be used for other objects like data frames and
#lists.
# lapply() function does not
#need MARGIN.
movies <- c("SPYDERMAN",
          "BATMAN","VERTIGO",
          "CHINATOWN")
movies
movies_lower <-lapply(movies,
                  tolower)
movies_lower
str(movies_lower)
#We can use unlist() to convert the list into a vector.
movies_lower <-unlist(lapply
                   (movies,
                     tolower))
movies_lower
str(movies_lower)



##############################################################################
```

```
# sapply() function
# sapply() function takes list,
#vector or data frame as
#input and
#gives output in vector
#or matrix.
#It is useful for
#operations on
#list objects and returns a
#list object of same length of
#original set.
#Sapply function in R does
#the same job as lapply()
#function but returns
#a vector.
# sapply(X, FUN)
# Arguments:
#   -X: A vector or an object
# -FUN: Function applied
#to each element of x
movies_lower <-sapply(movies,
                      tolower)
movies_lower
###################
?cars
View(cars)
dt <- cars
class(dt)
```

```
View(dt)
lmn_cars <- lapply(dt, min)
lmn_cars
smn_cars <- sapply(dt, min)
#lmn_cars
smn_cars
lmxcars <- lapply(dt, max)
smxcars <- sapply(dt, max)
#lmxcars
smxcars
#We can use a user built-in
#function into lapply() or
#sapply().
#We create a function named
#avg to compute the average of the minimum and
#maximum of the vector.
avg <- function(x)
 {
   (min(x) + max(x))/2
 }
fcars <- sapply(dt, avg)
fcars

#############################################################################


#tapply() function
#tapply() computes a measure (mean, median,
#min, max, etc..) or a function
```

```
#for each factor variable in a vector.
#It is a very useful function that
#lets you create a subset of a vector
#and then apply some functions to
#each of the subset.
# tapply(X, INDEX, FUN = NULL)
# Arguments:
#   -X: An object, usually a vector
# -INDEX: A list containing factor
# -FUN: Function applied to each element of x

my.matrx <- matrix(c(1:10, 11:20, 21:30),
                   nrow = 10, ncol = 3)
my.matrx
tdata <- as.data.frame(cbind(c(1,1,1,1,1,2,2,2,
                          2,2), my.matrx))
tdata
class(tdata)
colnames(tdata)
#Now let's use column 1 as the index and
#find the mean of column 2
tapply(tdata$V2, tdata$V1, mean)
##############################################################################

#mapply
#the arguments for mapply are
#mapply(FUN, …, MoreArgs = NULL,
#SIMPLIFY = TRUE, USE.NAMES = TRUE).
#First you list the function,
```

```
#using the rest of the arguments have
#default values so they don't need to be
#changed for now.
#When you have a function that takes 2
#arguments,
#the first vector goes into the first
#argument and the second vector
#goes into the second argument.
#In this example, 1:9 is specifying the value
#to repeat,
#and 9:1 is specifying how many times to
#repeat.
#This order is based on the order of arguments in the rep function itself.
mapply(rep, 1:9, 9:1)

#########################################################################
data(iris)
?iris
View(iris)
tapply(iris$Sepal.Width, iris$Species, median)
#vapply
#vapply is similar to sapply, but it requires you to specify what type of
#data you are expecting the arguments for vapply are
#vapply(X, FUN, FUN.VALUE). FUN.VALUE is where you specify the type of data
#you are expecting.
#I am expecting each item in the list to return a single numeric value,
#so FUN.VALUE = numeric(1).
vec <- c(1:10)
class(vec)
```

```
vec1=vapply(vec,sum,numeric(1))
vec1
class(vec1)
# Example
library(MASS)
data(state)
?state
head(state.x77)
str(state.x77)
#using apply to get summary data
apply(state.x77, 2, mean)
apply(state.x77, 2, median)
apply(state.x77, 2, sd)
#In this, I created one function that gives the mean and SD,
#and another that give min, median, and max.
#Then I saved them as objects that could be used later.
state.summary<- apply(state.x77, 2, function(x)
  c(mean(x), sd(x)))
state.summary
state.range <- apply(state.x77, 2, function(x)
  c(min(x), median(x), max(x)))
state.range
#Using mapply to compute a new variable
#In this example, I want to find the population density for each state.
#In order to do this, I want to divide population by area.
#state.area and state.x77 are not from the same dataset,
#but that is fine as long as the vectors are the same length
#and the data is in the same order. Both vectors are alphabetically
#by state, so mapply can be used.
```

```
population <- state.x77[1:50]
area <- state.area
pop.dens <- mapply(function(x, y) x/y,
                   population, area)
pop.dens
#Using tapply to explore population by region
#In this example, I want to find out some information about
#the population of states split by region. state.region is a
#factor with four levels: Northeast, South, North Central, and West.
#For each region, I want the minimum, median, and maximum populations.
```

```
region.info <- tapply(population, state.region,
                      function(x) c(min(x),
                                    median(x),
                                    max(x)))
region.info
```