# Main topic: Using the âapplyâ family function.

# Part 1: let’s start with “apply”. (2 pts each)

# create a 10-by-10 matrix for manipulation.

set.seed(1120)  
some\_matrix = matrix(rnorm(100), ncol = 10) # standard normal dist  
  
some\_matrix[1, 3]= some\_matrix[5,5]=some\_matrix[10, 2]= some\_matrix[6,8] = NA

# think about how to deal with NA.

#NA values become a hinderance when using certain functions, so they can be removed by specifying values of arguments #in your R code that remove them before making calculations eg na.rm = TRUE

# 1a, what’s the variance of the observations in each row of the matrix?

row\_var = apply(some\_matrix, 1, var,na.rm=TRUE)  
 row\_var

## [1] 0.5789088 0.1880476 1.2409336 0.5984562 0.7451429 1.2965789 0.6087033  
## [8] 1.6622512 0.4732179 1.0681420

# 1b, sort (all) the columns of the above matrix

some\_matrix[,1]= sort(some\_matrix[,1])  
 some\_matrix[,2]=sort(some\_matrix[,2], na.last = TRUE)  
 some\_matrix[,3]=sort(some\_matrix[,3], na.last = TRUE)  
 some\_matrix[,4]=sort(some\_matrix[,4])  
 some\_matrix[,5]=sort(some\_matrix[,5],na.last = TRUE)  
 some\_matrix[,6]=sort(some\_matrix[,6])  
 some\_matrix[,7]=sort(some\_matrix[,7])  
 some\_matrix[,8]=sort(some\_matrix[,8],na.last=TRUE)  
 some\_matrix[,9]=sort(some\_matrix[,9])  
 some\_matrix[,10]=sort(some\_matrix[,10])

# 1c, find the total number of negative values by column in some\_matrix.

# your answer should be a vector of integer of length 10.

new\_vec = c(sum(some\_matrix[,1] < 0),sum(some\_matrix[,2] < 0, na.rm= TRUE),  
 sum(some\_matrix[,3] < 0, na.rm= TRUE ),sum(some\_matrix[,4] < 0),  
 sum(some\_matrix[,5] < 0, na.rm= TRUE),sum(some\_matrix[,6] < 0),  
 sum(some\_matrix[,7] < 0),sum(some\_matrix[,8] < 0, na.rm= TRUE),  
 sum(some\_matrix[,9] < 0),sum(some\_matrix[,10] < 0))  
 new\_vec

## [1] 5 6 5 6 6 7 5 2 8 4

# Part 2: let’s play with “sapply”. (2 pts each)

# 2a, find the square root, the square and the cube of the numbers 1 to 10.

# (hint: think about how to define the function first; your answer should be a 3-by-10 matrix.)

numbers = c(1,2,3,4,5,6,7,8,9,10)  
squareroot\_vector= sapply(numbers, sqrt)  
 square\_vector =sapply(numbers, function (x) x^2)  
 cube\_vector = sapply(numbers, function (x) x^3)  
 solution\_matrix = rbind(squareroot\_vector,square\_vector,cube\_vector)  
 solution\_matrix

## [,1] [,2] [,3] [,4] [,5] [,6]  
## squareroot\_vector 1 1.414214 1.732051 2 2.236068 2.44949  
## square\_vector 1 4.000000 9.000000 16 25.000000 36.00000  
## cube\_vector 1 8.000000 27.000000 64 125.000000 216.00000  
## [,7] [,8] [,9] [,10]  
## squareroot\_vector 2.645751 2.828427 3 3.162278  
## square\_vector 49.000000 64.000000 81 100.000000  
## cube\_vector 343.000000 512.000000 729 1000.000000

# 2b, we played with the data set mtcars previously.

# now find the class of each variable(column)

data(mtcars)  
sapply(mtcars, class)

## mpg cyl disp hp drat wt qsec   
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"   
## vs am gear carb   
## "numeric" "numeric" "numeric" "numeric"

# Part 3: the next is “lapply”! (2 pts each)

# 3a, same as 2b), but using “lapply”, and get your function output as a vector as well.

# (hint: what does “lapply” return? how to convert it to a vector?)

unlist(lapply(mtcars, class))

## mpg cyl disp hp drat wt qsec   
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"   
## vs am gear carb   
## "numeric" "numeric" "numeric" "numeric"

# 3b, from 3a), what do you observe about the relationship, the difference

# between “sapply” and “lapply”? (hint: think about their inputs and outputs)

# sapply tries to simplify output by returning it in a vector form, while lapply returns output as a list

# 3c, here I create some random data

set.seed(1120)  
rndm <- replicate(15, runif(sample(1:10, 1)), simplify = FALSE)

# first, what did the above code generate? name the data structure, the length,

# and the content as well as giving a description.

#the above code generated a random distribution of samples in a list format #it is a list

length(rndm)

## [1] 15

str(rndm)

## List of 15  
## $ : num [1:7] 0.1359 0.1727 0.5719 0.0383 0.628 ...  
## $ : num [1:3] 0.0997 0.1031 0.4102  
## $ : num [1:2] 0.711 0.996  
## $ : num [1:2] 0.839 0.344  
## $ : num [1:8] 0.727 0.784 0.414 0.597 0.489 ...  
## $ : num [1:9] 0.094 0.649 0.442 0.583 0.912 ...  
## $ : num [1:5] 0.0547 0.3792 0.6228 0.2838 0.141  
## $ : num [1:8] 0.5109 0.112 0.6394 0.0277 0.9371 ...  
## $ : num [1:5] 0.7878 0.7276 0.122 0.1758 0.0278  
## $ : num [1:3] 0.183 0.664 0.427  
## $ : num [1:3] 0.997 0.663 0.582  
## $ : num [1:3] 0.582 0.463 0.996  
## $ : num [1:7] 0.966 0.296 0.622 0.119 0.587 ...  
## $ : num 0.657  
## $ : num [1:8] 0.0685 0.4317 0.0478 0.258 0.4222 ...

# 3d, using “lapply”, return the lengths of each component in rndm.

lapply(rndm, length)

## [[1]]  
## [1] 7  
##   
## [[2]]  
## [1] 3  
##   
## [[3]]  
## [1] 2  
##   
## [[4]]  
## [1] 2  
##   
## [[5]]  
## [1] 8  
##   
## [[6]]  
## [1] 9  
##   
## [[7]]  
## [1] 5  
##   
## [[8]]  
## [1] 8  
##   
## [[9]]  
## [1] 5  
##   
## [[10]]  
## [1] 3  
##   
## [[11]]  
## [1] 3  
##   
## [[12]]  
## [1] 3  
##   
## [[13]]  
## [1] 7  
##   
## [[14]]  
## [1] 1  
##   
## [[15]]  
## [1] 8

# 3e, now create the following list:

list\_a <- list(c=c(11:20), d=c(31:40))

# take the (natural) log of EACH element in list\_a

log(list\_a$c)

## [1] 2.397895 2.484907 2.564949 2.639057 2.708050 2.772589 2.833213  
## [8] 2.890372 2.944439 2.995732

log(list\_a$d)

## [1] 3.433987 3.465736 3.496508 3.526361 3.555348 3.583519 3.610918  
## [8] 3.637586 3.663562 3.688879

# explain why log2(list\_a) doesn’t work. (hint: help manual, look at the function input format)

#log2(x) works on numeric arguments, it does not view list\_a as a numeric argument because list\_a is a list

# Part 4: the “tapply” and its equivalent. (2 pts each)

# we will use the data set “iris”; familiarize yourself with it first:

data(iris)

# 4a, find the mean petal length by species.

tapply(iris$Petal.Length, iris$Species, mean )

## setosa versicolor virginica   
## 1.462 4.260 5.552

# 4b, look up the function “by”

# obtain the mean of the 4 features, by species, but using only one function call.

for( i in 1:4){  
   
   
 print(by(iris[,i], iris$Species, mean))  
   
 }

## iris$Species: setosa  
## [1] 5.006  
## --------------------------------------------------------   
## iris$Species: versicolor  
## [1] 5.936  
## --------------------------------------------------------   
## iris$Species: virginica  
## [1] 6.588  
## iris$Species: setosa  
## [1] 3.428  
## --------------------------------------------------------   
## iris$Species: versicolor  
## [1] 2.77  
## --------------------------------------------------------   
## iris$Species: virginica  
## [1] 2.974  
## iris$Species: setosa  
## [1] 1.462  
## --------------------------------------------------------   
## iris$Species: versicolor  
## [1] 4.26  
## --------------------------------------------------------   
## iris$Species: virginica  
## [1] 5.552  
## iris$Species: setosa  
## [1] 0.246  
## --------------------------------------------------------   
## iris$Species: versicolor  
## [1] 1.326  
## --------------------------------------------------------   
## iris$Species: virginica  
## [1] 2.026

# 4c, same as 4b), but using “aggregate” function.

aggregate(iris[,1:4], list(iris$Species), mean)

## Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Width  
## 1 setosa 5.006 3.428 1.462 0.246  
## 2 versicolor 5.936 2.770 4.260 1.326  
## 3 virginica 6.588 2.974 5.552 2.026

# 4d, same as 4b), but using the combination of “apply” and “tapply”.

# (hint: nested function)

# Part 5: other apply functions. (2 pts each)

# 5a, look up the function “mapply”

# we create list\_a in 3e).

list\_b <- list(a=1:10, b=21:30)

# What is the sum of the corresponding elements of list\_a and list\_b,in one function call?

# Your result should be a vector of length 10.

mapply(sum, list\_a$c,list\_a$d,list\_b$a, list\_b$b)

## [1] 64 68 72 76 80 84 88 92 96 100

# 5b, look up the function “rapply”

# same as 3e), but using rapply function, get your function output as a list as well.

# (hint: take a look at the function parameters)

log\_function\_output= as.list(rapply(list\_a, log))  
 log\_function\_output

## $c1  
## [1] 2.397895  
##   
## $c2  
## [1] 2.484907  
##   
## $c3  
## [1] 2.564949  
##   
## $c4  
## [1] 2.639057  
##   
## $c5  
## [1] 2.70805  
##   
## $c6  
## [1] 2.772589  
##   
## $c7  
## [1] 2.833213  
##   
## $c8  
## [1] 2.890372  
##   
## $c9  
## [1] 2.944439  
##   
## $c10  
## [1] 2.995732  
##   
## $d1  
## [1] 3.433987  
##   
## $d2  
## [1] 3.465736  
##   
## $d3  
## [1] 3.496508  
##   
## $d4  
## [1] 3.526361  
##   
## $d5  
## [1] 3.555348  
##   
## $d6  
## [1] 3.583519  
##   
## $d7  
## [1] 3.610918  
##   
## $d8  
## [1] 3.637586  
##   
## $d9  
## [1] 3.663562  
##   
## $d10  
## [1] 3.688879

# Part 6: to sum up… (2 pts each)

# 6a,

# give an example or detailed explanation of and example of when the “apply” family function

# doesn’t work, we have to use for loop(s).

#some members of the apply family functions only work on arguments of the same length as its index per time, #in such cases a loop has to be introduced. # For example a for loop is necessary when using the tapply function to find the mean of all the columns of the iris dataset with index “Species”

# 6b, to make sure you really know how to apply “apply”,

# give an example of, two nested “apply” family functions, explain what you did.

# e.g., apply(apply(some\_matrix,2,is.na), 2, sum), this gives you the number

# of NA’s in each column. It’s just for illustration, you can do this with one “apply” too.

# anything similar to the example above doesn’t count.

# Part 7: linear regression. (5pts)

# Using the family data, fit a linear regression model to predict weight from height.

# comment on the output; how do you interpret this model? (hint: lm() function)

load("C:/Users/Kosi/Downloads/family.rda")  
   
 lm(weight ~ height, data = family)

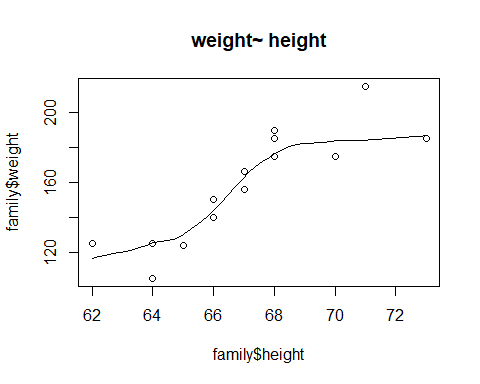
##   
## Call:  
## lm(formula = weight ~ height, data = family)  
##   
## Coefficients:  
## (Intercept) height   
## -455.666 9.154

#-455.666 andd 9.154 are the beta coefficients. #we can therefore say weight = -455.666 + 9.154height

# Create a scatterplot of height vs weight. Add the linear regression line you found above.

# Provide an interpretation for your plot.

scatter.smooth (x = family$height, y = family$weight, main= "weight~ height")



# weight and height have a positive linear relationship meaning that as height increases , weight tends to incease too.