#Name: Hui Lyu

Main topic: Using the "apply" family function

#Q1 (5 pts)

Given a function below,

myfunc <- function(z) return(c(z,z^2, z^3%/%2))

#(1) Examine the following code, and briefly explain what it is doing.

y = 2:8

myfunc(y)

matrix(myfunc(y),ncol=3)

Your explanation

First generate a vector named y of seven consecutive integers from 2 to 8

Then pass the value of y to the established function myfunc to compute corresponding return values

Finally reformat the values into a matrix of 3 columns

The first column represents the value of y, the second column represents y^2, the third column represents y^3%/%2.

Each row of the matrix contains three return values based on myfunc of each element in y.

```
> y = 2:8
> myfunc(y)
 [1]   2   3   4   5   6   7   8   4   9  16  25  36  49  64   4  13  32
62 108 171 256
> matrix(myfunc(y),ncol=3)
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    3    9   13
[3,]    4   16   32
[4,]    5   25   62
[5,]    6   36  108
[6,]    7   49  171
[7,]    8   64  256
```

#(2) Simplify the code in (1) using one of the "apply" functions and save the result as m.

###code & result

m = t(sapply(2:8, myfunc))

```
> m = t(sapply(2:8, myfunc))
> m
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    3    9   13
[3,]    4   16   32
[4,]    5   25   62
[5,]    6   36  108
[6,]    7   49  171
```

```
[7,]    8   64  256
```

#(3) Find the row product of m.

###code & result

apply(m,1,prod)

```
> apply(m,1,prod)
[1]      32     351    2048    7750   23328   58653 131072
```

#(4) Find the column sum of m in two ways.

###code & result

apply(m,2,sum)

colSums(m)

```
> apply(m,2,sum)
[1]  35 203 646
> colSums(m)
[1]  35 203 646
```

#(5) Could you divide all the values by 2 in two ways?

code & result

m/2

apply(m, 1:2, function(x) x/2)

```
> m/2
     [,1] [,2]  [,3]
[1,]  1.0  2.0   2.0
[2,]  1.5  4.5   6.5
[3,]  2.0  8.0  16.0
[4,]  2.5 12.5  31.0
[5,]  3.0 18.0  54.0
[6,]  3.5 24.5  85.5
[7,]  4.0 32.0 128.0
> apply(m, 1:2, function(x) x/2)
     [,1] [,2]  [,3]
[1,]  1.0  2.0   2.0
[2,]  1.5  4.5   6.5
[3,]  2.0  8.0  16.0
[4,]  2.5 12.5  31.0
[5,]  3.0 18.0  54.0
[6,]  3.5 24.5  85.5
[7,]  4.0 32.0 128.0
```

#Q2 (8 pts)

#Create a list with 2 elements as follows:

l <- list(a = 1:10, b = 11:20)

#(1) What is the product of the values in each element?

lapply(l,prod)

```
> lapply(l,prod)
```

```
$a
[1] 3628800

$b
[1] 670442572800
```

#(2) What is the (sample) variance of the values in each element?

lapply(l,var)

```
> lapply(l,var)
$a
[1] 9.166667

$b
[1] 9.166667
```

#(3) What type of object is returned if you use lapply? sapply? Show your R code that finds these answers.

```
> class(lapply(l,var))
[1] "list"
> class(sapply(l,var))
[1] "numeric"
```

lapply returns a list, while sapply returns a vector whose type is numeric for this variable

Now create the following list:

l.2 <- list(c = c(21:30), d = c(31:40))

#(4) What is the sum of the corresponding elements of l and l.2, using one function call?

mapply(sum, la, lb, l.2$c, l.2$d)

```
> mapply(sum, l$a, l$b, l.2$c, l.2$d)
 [1]  64  68  72  76  80  84  88  92  96 100
```

#(5) Take the log of each element in the list l:

sapply(l, log)

```
> sapply(l, log)
              a        b
 [1,] 0.0000000 2.397895
 [2,] 0.6931472 2.484907
 [3,] 1.0986123 2.564949
 [4,] 1.3862944 2.639057
 [5,] 1.6094379 2.708050
 [6,] 1.7917595 2.772589
 [7,] 1.9459101 2.833213
 [8,] 2.0794415 2.890372
 [9,] 2.1972246 2.944439
[10,] 2.3025851 2.995732
```

#(6) First change l and l.2 into matrixes, make each element in the list as column,

your code here

l = matrix(unlist(l), ncol = 2)

l.2 = matrix(unlist(l.2), ncol = 2)

```
> l = matrix(unlist(l), ncol = 2)
> l.2 = matrix(unlist(l.2), ncol = 2)
> l
      [,1] [,2]
 [1,]    1   11
 [2,]    2   12
 [3,]    3   13
 [4,]    4   14
 [5,]    5   15
 [6,]    6   16
 [7,]    7   17
 [8,]    8   18
 [9,]    9   19
[10,]   10   20
> l.2
      [,1] [,2]
 [1,]   21   31
 [2,]   22   32
 [3,]   23   33
 [4,]   24   34
 [5,]   25   35
 [6,]   26   36
 [7,]   27   37
 [8,]   28   38
 [9,]   29   39
[10,]   30   40
```

#Then, form a list named mylist using l,l.2 and m (from Q1) (in this order).

your code here

mylist = list(l=l,l.2=l.2,m=m)

```
> mylist = list(l=l,l.2=l.2,m=m)
> mylist
$l
      [,1] [,2]
 [1,]    1   11
 [2,]    2   12
 [3,]    3   13
 [4,]    4   14
 [5,]    5   15
 [6,]    6   16
 [7,]    7   17
 [8,]    8   18
 [9,]    9   19
[10,]   10   20

$l.2
      [,1] [,2]
 [1,]   21   31
 [2,]   22   32
 [3,]   23   33
 [4,]   24   34
 [5,]   25   35
 [6,]   26   36
 [7,]   27   37
 [8,]   28   38
 [9,]   29   39
[10,]   30   40

$m
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    3    9   13
```

```
[3,]    4   16   32
[4,]    5   25   62
[5,]    6   36  108
[6,]    7   49  171
[7,]    8   64  256
```

#Then, select the first column of each elements in mylist in one function call (hint '[' is the select operator).

your code here

lapply(mylist, function(l) l[,1])

```
> lapply(mylist, function(l) l[,1])
$l
 [1]  1  2  3  4  5  6  7  8  9 10

$l.2
 [1] 21 22 23 24 25 26 27 28 29 30

$m
[1] 2 3 4 5 6 7 8
```

#Q3 (3 pts)

Let's load our friend family data again.

load(url("http://courseweb.lis.illinois.edu/~jguo24/family.rda"))

#(1) Find the mean bmi by gender in one function call.

tapply(family$bmi,family$gender,mean)

```
> tapply(family$bmi,family$gender,mean)
       m        f
25.73898 23.02564
```

#(2) Could you get a vector of what the type of variables the dataset is made of?

sapply(family, class)

```
> sapply(family, class)
firstName    gender       age    height    weight       bmi    overWt
 "factor"  "factor" "integer" "numeric" "integer" "numeric" "logical"
```

#(3) Could you sort the firstName in height descending order?

family$firstName[order(family$height, decreasing = TRUE)]

```
> family$firstName[order(family$height, decreasing = TRUE)]
 [1] Joe Tom Tom Liz Jon Tim Bob Ann Dan Art Sal May Sue Zoe
Levels: Ann Art Bob Dan Joe Jon Liz May Sal Sue Tim Tom Zoe
```

#Q4 (2 pts)

There is a famous dataset in R called "iris." It should already be loaded

in R for you. If you type in ?iris you can see some documentation. Familiarize

yourself with this dataset.

#(1) Find the mean petal length by species.

code & result

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

```
> tapply(iris$Petal.Length, iris$Species, mean)
    setosa versicolor  virginica
     1.462      4.260      5.552
```

#(2) Now obtain the sum of the first 4 variables, by species, but using only one function call.

code & result

by(iris[, 1:4], iris$Species, colSums)

```
> by(iris[, 1:4], iris$Species, colSums)
iris$Species: setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       250.3        171.4         73.1         12.3
------------------------------------------------------------------------
iris$Species: versicolor
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       296.8        138.5        213.0         66.3
------------------------------------------------------------------------
iris$Species: virginica
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       329.4        148.7        277.6        101.3
```

#Q5 (2 pts)

#Below are two statements, their results have different structure,

lapply(1:4, function(x) x^3)

sapply(1:4, function(x) x^3)

Could you change one of them to make the two statements return the same results (type of object)?

unlist(lapply(1:4, function(x) x^3))

```
> lapply(1:4, function(x) x^3)
[[1]]
[1] 1

[[2]]
[1] 8

[[3]]
[1] 27

[[4]]
[1] 64

> sapply(1:4, function(x) x^3)
[1]  1  8 27 64
> unlist(lapply(1:4, function(x) x^3))
[1]  1  8 27 64
> class(unlist(lapply(1:4, function(x) x^3)))
[1] "numeric"
> class(sapply(1:4, function(x) x^3))
[1] "numeric"
```

#Q6. (5 pts) Using the family data, fit a linear regression model to predict

weight from height. Place your code and output (the model) below.

lm(family$weight ~ family$height, data = family)

```
> lm(family$weight ~ family$height, data = family)

Call:
lm(formula = family$weight ~ family$height, data = family)

Coefficients:
  (Intercept)  family$height  
     -455.666          9.154
```

The model is: weight = -455.666 + 9.154*height

How do you interpret this model?

The weight has a positive correlation with height since the slope 9.154>0. As height goes up,

the corresponding weight also goes up. This conforms to our common sense. The intercept is below 0,

which means the value of weight is below 0 when height equals 0. This is a sort of parallel movement

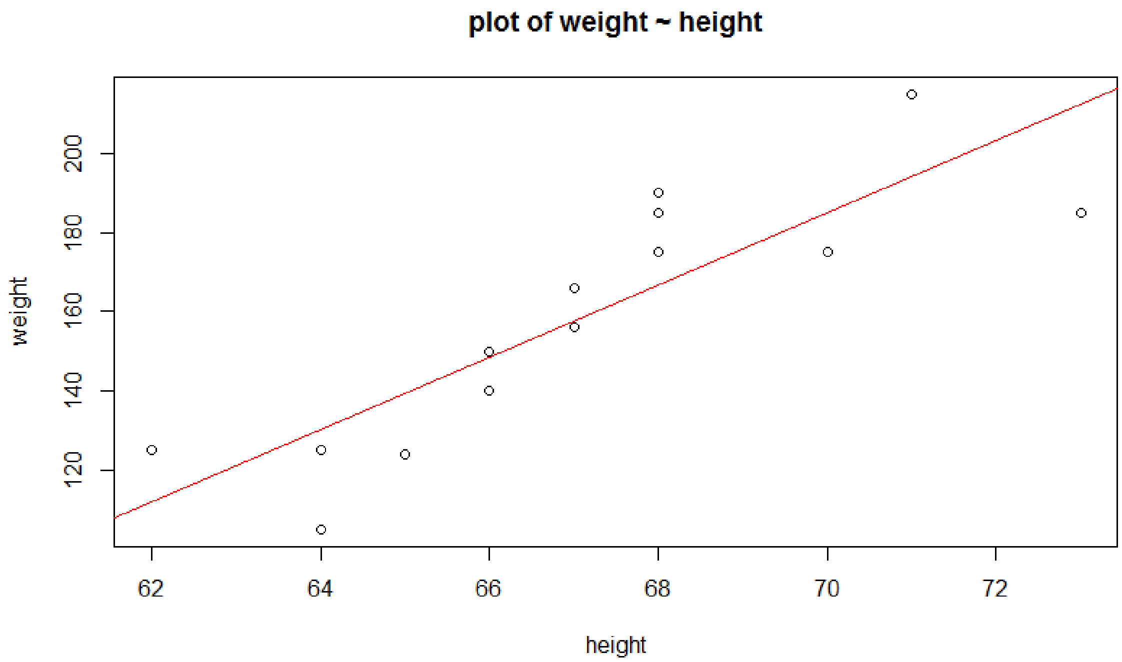
to make it adjust to the data.

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

plot(family$height, family$weight, xlab = "height", ylab = "weight", main = "plot of weight ~ height")

abline(lm(family$weight ~ family$height, data = family), col='red')

```
> plot(family$height, family$weight, xlab = "height", ylab = "weight", mai
n = "plot of weight ~ height")
> abline(lm(family$weight ~ family$height, data = family), col='red')
```



Provide an interpretation for your plot.

In the plot, the scattered spots are true values of 14 objects in the family dataset. The x

axis is height, and the y axis is weight. Basically, as height goes up, his or her weight

also goes up. The red line is the linear regression line based on the 14 objects. As can be

seen, the number of spots under the line is equal to the number of spots above the line. All

the spots are basically close to the line, which means the residuals are not too large. There

is no outlier in the plot. In general, it is a good linear model based on true observations.

```
> summary(lm(family$weight ~ family$height, data = family))

Call:
lm(formula = family$weight ~ family$height, data = family)

Residuals:
    Min      1Q  Median      3Q     Max 
-27.554  -9.689  -0.055  11.944  23.214 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)   -455.666    107.029  -4.257  0.00111 ** 
family$height    9.154      1.594   5.741 9.29e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16.94 on 12 degrees of freedom
Multiple R-squared:  0.7331,	Adjusted R-squared:  0.7109 
F-statistic: 32.96 on 1 and 12 DF,  p-value: 9.287e-05
```

P-value is also very small, which means a good linear model.