sarahdoon / Midterm Public

Code Issues 1 Pull requests Actions Projects Wiki Security

main

Midterm / Midterm Dataset 1.R

sarahdoon Add files via upload History

1 contributor

171 lines (158 sloc) | 5 KB

```r
# Number 1
# put train file in variable
data <- read.csv("/Users/sarahferaidoon/Desktop/train.csv.gz")
data
# Length(data[,1]) = 42000
length(data[,1])

# make data frame
pixels <- data.frame(data[,-1])
pixels
# use smoothScatter
smoothScatter(pixels)
pixels

# make a matrix and remove the label, first column, so we only have pixels
label1<-pixels[1,]
mode(label1)
mode(label1) = "numeric"
label1
# make a matrix
my_label1 <- matrix(label1, nrow=28, byrow=TRUE)
par(mar = c(5,5,5,5))
# use image to plot it out
image(my_label1, axes = FALSE, col = grey.colors(256, start=0, end=1))
my_label1
matplot((my_label1),type="l")

# make a for-loop
r <- c(1:length(pixels[,1]))
```

```
for (x in r){
  label1 <- pixels[x,]
  mode(label1) = "numeric"
  my_label1 <- matrix(label1, nrow=28, byrow=TRUE)
  par(mar=c(5,5,5,5))
  image(my_label1, axes = FALSE, col = grey.colors(256, start=0, end=1))
  my_label1
}

# Number 2
# everyone who writes a certain number will be put into a certain variable
# 0 into label_0
# take the average, after we do this and see if it still looks like its number
n <- c(0:9)

for (x in n){
  assign(paste0("label_",x),data[grep(x,data[,1]),])
}
# label is the number and the pixels of where the data is drawn
# 0 is black all the way to 256 (white) and shades of grey in between
label_0
label_1
label_2
label_3
label_4
label_5
label_6
label_7
label_8
label_9

# Problem 2a
# take all the data that has label handwriting as 0 and remove the first column
# so there are only pixels that represent 0
pixel_label_0 <- label_0[-1]
length(pixel_label_0) # -> 784
# make a variable to keep track of every number from 1 to max length of columns
c <- c(1:length(pixel_label_0))
# make an empty list to append the average values for each column to
ave_pix <- c()
# make a for loop
# loop through c with x
# make x the column in pixel_label_0 variable
for (x in c){
  # look at column, take mean of entire column, append the mean to the ave_pix variable
  ave_pix <- append(ave_pix, mean(pixel_label_0[,x]))
}
# make a matric with means from ave-pix
# each entry represents the mean number for each column in the dataset labeled 0
```

```r
matrix_ave_pix <- matrix(ave_pix, nrow=28, ncol=28)
# plot the matrix with image function
# make the colors shades of grey
image(matrix_ave_pix, axes=FALSE, col=grey.colors(256,start=0,end=1))

# Repeat for "1" digit
pixel_label_1 <- label_1[-1]
length(pixel_label_1) # -> 784
c <- c(1:length(pixel_label_1))
ave_pix_1 <- c()
for (x in c){
  ave_pix_1 <- append(ave_pix_1, mean(pixel_label_1[,x]))
}
matrix_ave_pix_1 <- matrix(ave_pix_1, nrow=28, ncol=28)
image(matrix_ave_pix_1, axes=FALSE, col=grey.colors(256,start=0,end=1))

# Make a Function
myfunction <- function(r){
  pixel_label_0 <- r[,-1]
  c <- c(1:length(pixel_label_0))
  ave_pix <- c()
  for (x in c) {
    ave_pix <- append(ave_pix, mean(pixel_label_0[,x]))
  }
  matrix_ave_pix <- matrix(ave_pix, nrow=28, ncol=28)
  image(matrix_ave_pix, axes=FALSE, col=grey.colors(256,start=0,end=1))
}
myfunction(label_0)
myfunction(label_1)
myfunction(label_2)
myfunction(label_3)
myfunction(label_4)
myfunction(label_5)
myfunction(label_6)
myfunction(label_7)
myfunction(label_8)
myfunction(label_9)
# 0, 3, and 8 look the best because they are both distinguishable and in the right orienta
# 1 and 9 look the most blurry

# Correct the orientation
for (x in r){ # r represents the current row in the dataframe pixels
  label1 <- pixels[x,] # pass entire row into variable
  mode(label1) = "numeric" # change mode of number so it can be imaged
  my_label1 <- matrix(rev(label1), nrow=28, byrow=FALSE) # make that row into a 28x28 matr
  my_label1 <- apply(my_label1,2,rev) # reverse the matrix with apply
  par(mar=c(5,5,5,5))
  image(my_label1,axes=FALSE, col=grey.colors(256,start=0,end=1)) # plot points in matrix
  if (x==10){break} # just see the first 10 rows
```

```r
}

install.packages("raster")

# Problem 3
# 3a
# make a function
# c will keep track of the column numbered 2-783
# make empty variable var_of_col to pass all of the variance results to
func_each_label <- function(L,k){
  length(L)
  c <- c(2:length(L))
  var_of_col <- c()
  for (x in c){
    # in each column calculate variance of entire column and append that
    var_of_col <- append(var_of_col, var(L[,x]))
  }
  max(var_of_col)
  cat("the column number for ", k, "that has the highest variance is: ", which(var_of_col=
  # didn't include first column so +1
}
func_each_label(label_0, "label_0")
func_each_label(label_1, "label_1")
func_each_label(label_2, "label_2")
func_each_label(label_3, "label_3")
func_each_label(label_4, "label_4")
func_each_label(label_5, "label_5")
func_each_label(label_6, "label_6")
func_each_label(label_7, "label_7")
func_each_label(label_8, "label_8")
func_each_label(label_9, "label_9")


# Problem 4
library(raster)
for (n in x){
  k <- paste0("hand_writing_",n,".png")
  im <- load.image(k)
  # im <- load.image(file)
  im.r <- as.raster(im,interpolate=F)
  str(im.r)
  plot(im.r)
}
dev.off()
```

sarahdoon / Midterm Public

Code Issues 1 Pull requests Actions Projects Wiki Security

main

Midterm / Midterm Dataset 2.R

sarahdoon Add files via upload History

1 contributor

96 lines (79 sloc) | 3.68 KB

```
1  # Problem 1
2  # put file into variable
3  Mcountdata <- read.csv("/Users/sarahferaidoon/Desktop/Mnemiopsis_count_data.csv")
4  Mcountdata
5
6  Mcoldata <- read.csv("/Users/sarahferaidoon/Desktop/Mnemiopsis_col_data.csv")
7  Mcoldata
8
9  Mcountdata$avg <- rowMeans(Mcountdata[,2:9])
10 topgenes <- Mcountdata[order(Mcountdata$avg,decreasing = TRUE), 1]
11 head(topgenes,5)
12 # >[1] "ML20395a"  "ML26358a"  "ML46651a"  "ML020045a" "ML00017a"
13 # ML20395a has molecular function: nucleotide binding, translation elongation factor activ
14 # ML26358a has molecular function : nucleotide binding, ATPbinding; Cellular Componend: cy
15 # ML46651a has cellular component: membrane attack complex
16 # ML020045a molecular function: nucleotide binding, GTPase activity, GTP binding; Cellular
17 # ML00017a
18
19 # Problem 2
20 # Yes, the top 5 genes will be different if done on a per column basis
21 # aboral1
22 head(Mcountdata[order(Mcountdata$aboral1,decreasing=TRUE),1],5)
23 # [1] "ML46651a"  "ML20395a"  "ML020045a" "ML174731a" "ML26358a"
24 # ML174731a is new, missing ML00017a
25
26 # aboral2
27 head(Mcountdata[order(Mcountdata$aboral2,decreasing=TRUE),1],5)
28 # [1] "ML20395a"  "ML46651a"  "ML26358a"  "ML01482a"  "ML034334a"
29 # "ML01482a" & "ML034334a" are new, missing ML00017a and ML020045a
```

```
30
31  # aboral3
32  head(Mcountdata[order(Mcountdata$aboral3,decreasing=TRUE),1],5)
33  # [1] "ML20395a"  "ML01482a"  "ML26358a"  "ML46651a"  "ML034334a"
34  # "ML01482a" & "ML034334a" are new
35
36  # aboral4
37  head(Mcountdata[order(Mcountdata$aboral4,decreasing=TRUE),1],5)
38  # [1] "ML01482a"  "ML20395a"  "ML034334a" "ML46651a"  "ML034336a"
39  # "ML01482a" & "ML034334a" & "ML034336a" are new
40
41  # oral1
42  head(Mcountdata[order(Mcountdata$oral1,decreasing=TRUE),1],5)
43  # [1] "ML20395a"  "ML020045a" "ML04011a"  "ML26358a"  "ML00017a"
44  # ML04011a is new
45
46  # oral2
47  head(Mcountdata[order(Mcountdata$oral2,decreasing=TRUE),1],5)
48  # [1] "ML20395a"  "ML020045a" "ML04011a"  "ML00017a"  "ML26358a"
49  # ML04011a is new
50
51  # oral3
52  head(Mcountdata[order(Mcountdata$oral3,decreasing=TRUE),1],5)
53  # [1] "ML20395a"  "ML004510a" "ML26358a"  "ML00017a"  "ML04011a"
54  # "ML004510a" is new
55
56  # oral4
57  head(Mcountdata[order(Mcountdata$oral4,decreasing=TRUE),1],5)
58  # [1] "ML20395a"  "ML004510a" "ML46651a"  "ML020045a" "ML00017a"
59  # "ML004510a" is new
60
61  # Problem 3
62  # from columns 2-9 in Mcountdata, put the colmeans into a list
63  MCDmean <- c(colMeans(Mcountdata[,2:9]))
64  MCDmean
65  # apply sd method to columns 2 - 9 of Mcountdata
66  MCDsd <- sapply(Mcountdata[,2:9],sd)
67  MCDsd
68
69  # put all columns in Mcountdata in variable
70  scale <- Mcountdata[,1:9]
71  for (i in 2:9){# for every item in the column
72    c <- (MCDmean[1])/(MCDmean[i-1]) # create scaling factor of meancol/meancolumn-1
73    scale[,i]<- c * scale[i] # multiply column by this scaling factor
74  }
75  scale
76  colMeans(scale[,2:9])
77
78  # Problem 4
```

```
79  cor(scale[,2:9])
80  # aboral-aboral and oral-oral correlations are closer than aboral-oral
81
82
83  # Problem 5
84  # I would break the data into thirds somehow
85
86  # Problem 6
87  # create a dataframe with gene names
88  vardf <- data.frame()
89  for (i in 1:length(scale$Gene)){ # for each Gene item in the length of the scaled data
90    if(sum(scale[i,2:9])>=0.5){
91      rbind(vardf,c(scale[i,1],var(as.numeric(scale[i,2:9]))))
92    }
93  }
94  vardf
95
96  # Problem 7
```

508 lines (384 sloc) | 14.1 KB

```
---
title: "Midterm"
output:
  pdf_document: default
  html_notebook: default
---
Tarabeth Gerrity
tg64@nyu.edu

Midterm
Dataset 1:  train.csv.gz
This is hand-written digits (0-9) from many people.
It contains 785 columns, first columns is digit and the 784 remaining columns
  are pixels 0-783. These are essentially 28x28 squares, so you can map the 784
  columns to entries in the squares. Each entry is 0-255, 0 is for black and 255
  stands for white, numbers in between are shades of gray
1) Read in the data and convert the pixel data into pictures using plot
   show a few examples in your report along with the code


```{r}

#import train data

train <- read.csv("/Users/morgenstern/Desktop/BioStats/train.csv")

# first column is the digit


##create a 28x28 matrix for the rest of the columns
```
```

```
30
31  train1<- train[1,]
32
33  train1
34  #create the matrix as numeric, default sets it to a character matrix instead
35  matrix1 <- matrix(as.numeric(train[1,785:2]),nrow=28, ncol=28)
36  heatmap(matrix1,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
37
38
39  ##Set range values to whichever rows of the train data you want to graph
40  for (i in 1:10){
41    m <- matrix(as.numeric((train[i,785:2])),nrow=28, ncol=28, byrow = FALSE)
42    heatmap(t(m),Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
43  }
44
45
46
47  ```
48  2)  Separate the data by digits, and calculate average value of each pixel,
49     a)  Plot the average values, does it still resemble the digit label ?
50     Yes.
51     b) which digits fare the best under this operation ?
52     2,5 and 7
53
54  ```{r}
55  library(dplyr)
56  ##create 10 different vars for each number
57
58  dig0 <- train[which(train$label == 0), ]
59  mat0 <-matrix(as.numeric(colMeans(dig0[,2:785])),nrow=28, ncol=28)
60  heatmap(mat0,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
61
62  dig1<- train[which(train$label == 1), ]
63  mat1 <-matrix(as.numeric(colMeans(dig1[,2:785])),nrow=28, ncol=28)
64  heatmap(mat1,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
65
66  dig2<- train[which(train$label == 2), ]
67  mat2 <-matrix(as.numeric(colMeans(dig2[,2:785])),nrow=28, ncol=28)
68  heatmap(mat2,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
69
70  dig3<- train[which(train$label == 3), ]
71  mat3 <-matrix(as.numeric(colMeans(dig3[,2:785])),nrow=28, ncol=28)
72  heatmap(mat3,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
73
74  dig4<- train[which(train$label == 4), ]
75  mat4 <-matrix(as.numeric(colMeans(dig4[,2:785])),nrow=28, ncol=28)
76  heatmap(mat4,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
77
78  dig5<- train[which(train$label == 5), ]
```

```
mat5 <-matrix(as.numeric(colMeans(dig5[,2:785])),nrow=28, ncol=28)
heatmap(mat5,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))

dig6<- train[which(train$label == 6), ]
mat6 <-matrix(as.numeric(colMeans(dig6[,2:785])),nrow=28, ncol=28)
heatmap(mat6,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))

dig7<- train[which(train$label == 7), ]
mat7 <-matrix(as.numeric(colMeans(dig7[,2:785])),nrow=28, ncol=28)
heatmap(mat7,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))

dig8<- train[which(train$label == 8), ]
mat8 <-matrix(as.numeric(colMeans(dig8[,2:785])),nrow=28, ncol=28)
heatmap(mat8,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))

dig9<- train[which(train$label == 9), ]
mat9 <-matrix(as.numeric(colMeans(dig9[,2:785])),nrow=28, ncol=28)
heatmap(mat9,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))

##Now map all of the train data- this will give insight into variance over the entire data
digall <- train
matall <-matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
heatmap(matall,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))

```
   3) Find which columns have the most variance and which have the least.
   a) Over the whole dataset and then separately, for each digit (0-9)

```{r}

##Variance over whole data set

v<- data.frame(ncol=2, nrow = 0)
colnames(v) <- c("pixel", "variance")

for(i in 2:785){
  newrow <- c(i, as.numeric(var(train[,i])))
  v <- rbind(v, newrow)
}

v

varmax <- v[which.max(v$variance),]
varmax
#pixel 408, var 12961.86
varmin <- v[which.min(v$variance),]
varmin
#pixel 2, var   0
##For the whole data, the max variance is in pixel 408 with var of 12961.86
```

```
##Min var is in pixel 2 ##Note, there are many pixels with var of zero

##Variance of each digit.
##reuse the variables from the previous question

digvar <- function(digit){
  #create a temporary empty dataframe
  tempdf<- data.frame(ncol=2, nrow = 0)
  #set colnames
colnames(tempdf) <- c("pixel", "variance")

for(i in 2:785){
  newrow <- c(i, as.numeric(var(digit[,i])))
  tempdf <- rbind(tempdf, newrow) }

tempvarmax <- tempdf[which.max(tempdf$variance),]
tempvarmin <- tempdf[which.min(tempdf$variance),]

  return(c(tempvarmax, tempvarmin))
}

digvar(dig0)
#pixel 268 variance 12843.17
#2$variance[1] 0
digvar(dig1)
#573$variance[1] 12997.2
#2$variance[1] 0
digvar(dig2)
#467$variance[1] 13051.45
#$pixel2$variance[1] 0
digvar(dig3)
#188 12539.43
#2 0
digvar(dig4)
# 429 12629.21
#2,0
digvar(dig5)
# 184 12562.46
#2 0
digvar(dig6)
#240, 12495.36
#2,0
digvar(dig7)
#602, 12615.84
#2, 0
digvar(dig8)
#627 12270.6
#2,0
digvar(dig9)
```

```
#627 12445.65
#2,0
```
   b) Can you connect the variance to the results in 2b ?

   Pixels with variance of zero corresponds regions where no one writes in - The extreme c
   Pixels with the highest variance are biased towards the center of the matrix - People w

   c) Does replacing the columns with the lowest variability by their average
        value have an effect on the digits ?

        No effect. The lowest var 0 corresponds to columns with only 0's and a mean of 0.
```{r}
## lowest var in each digit = 0

digrep <- function(digit){
  #input digit dataframe
  tempdf <- (digit)
  for(i in 2:785){
    #conditional, of column var is equal to 0 replace the tempdf column with the mean
  if (var(digit[,i])==0){
    tempdf[,i]<- mean(digit[,i])
    }
  }
  ## Now heatmap the tempdf
  tempmat <- matrix(as.numeric(colMeans(tempdf[,2:785])),nrow=28, ncol=28)
  heatmap(tempmat,Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
}

digrep(dig0)
digrep(dig1)
digrep(dig2)
digrep(dig3)
digrep(dig4)
digrep(dig5)
digrep(dig6)
digrep(dig7)
digrep(dig8)
digrep(dig9)
digrep(train)
```

   d) How many columns have average values close to 255 or 0 and why ?

   Many columns have a mean of close to 0 and very few columns have means close to 255. Nu

```{r}

digmean <- function(digit){
```

```
minmean <- c()
maxmean <- c()
for(i in 2:785){
  if(mean(digit[,i])<= 5){
    minmean <- c(minmean, 1)
  }
  ##made the maxmean capture less sensitive to get a nonzero result
  else if (mean(digit[,i])>= 200){
    maxmean <- c(maxmean, 1)
  }
}
  #length of the two lists represents the number of columns meeting the capture criteria
  #the contents of the lists are unimportant
  return(c(length(minmean), length(maxmean)))
}

digmean(train)
digmean(dig0)
digmean(dig1)
digmean(dig2)
digmean(dig3)
digmean(dig4)
digmean(dig5)
digmean(dig6)
digmean(dig7)
digmean(dig8)
digmean(dig9)

```

4) On a graph paper (you can print your own at https://print-graph-paper.com),
   mark off 28 x 28 squares, write the digits (0-9) in these squares and "digitize"
   them, essentially add lines corresponding to your own handwriting to this set
   You should present a program that prints out digits in your handwriting.
   If you are ambitious, create a program to print your signature (not a
   statistic question, no bonus points for this)

```{r}

library(png)

myimg <- readPNG("/Users/morgenstern/Desktop/sample8.png")

dim(myimg)


image(myimg[1:3024,1:3024,3], col = gray.colors(256, start = 0, end = 1))
##Input image must be a square
```

```
##constant to convert image dimensions to a 28x28 grid
shrink <- 3024/28


#create square "slices".
#start by going across the first row, adding to slice array

shrinkify <- function(img){
slice <- c()
for(j in 1:28){
  ##row shifter
  ##change y and rows in terms of j
  ##initially rows:y is 1:28
  cols <- 1
  x <- shrink
  y <- shrink * j
  rows <- y - 27
for (i in 1:28){
  tempmean <- mean(myimg[rows:y,cols:x,3])
  slice <- c(slice, tempmean)
  #shift to the right, stay in row
  #rows unchanged
  #y unchanged
  cols <- cols +shrink
  x <- x + shrink
}
  #after reaching the right edge, reset the column coords
  cols <- 1
  x<- shrink
}
  p <- matrix(slice, nrow=28, ncol=28, byrow=TRUE)
  heatmap(p, Rowv = NA, Colv = NA, col = gray.colors(256, start = 0, end = 1))

}

shrinkify(myimg)


```

Dataset 2: a) Mnemiopsis_col_data.csv b) Mnemiopsis_count_data.csv
This is gene expression data, The columns represent samples, whose information is
in the col_data file. The count_data file contains counts for each gene (rows).
The file, info_gene.txt contains information about the organism and some links to
look up gene functions. It will be a good experience to learn to use the genome resources,
as this is the kind of struggles most researchers go through when they start looking at ge
1) What are the top 5 genes with the highest average expression
  (across experiments) in the set ? what is their function ?
2) Are the top 5 genes different if they are done on a per column basis ?

````
```{r}

MCD <- read.csv("/Users/morgenstern/Desktop/BioStats/Mnemiopsis_count_data.csv")
MCD

#using rowMeans as is results in error. Need to exclude the Gene names
MCD$avg <- rowMeans(MCD[,2:9])

topgenes <- MCD[order(MCD$avg, decreasing = TRUE), 1]
head(topgenes,5)
#"ML20395a"  "ML26358a"  "ML46651a"  "ML020045a" "ML00017a"


# 2) Are the top 5 genes different if they are done on a per column basis ?
##aboral1
head(MCD[order(MCD$aboral1, decreasing = TRUE), 1],5)
#"ML46651a"  "ML20395a"  "ML020045a" "ML174731a" "ML26358a"

#aboral2
head(MCD[order(MCD$aboral2, decreasing = TRUE), 1],5)
#"ML20395a"  "ML46651a"  "ML26358a"  "ML01482a"  "ML034334a"

#aboral3
head(MCD[order(MCD$aboral3, decreasing = TRUE), 1],5)
#"ML20395a"  "ML01482a"  "ML26358a"  "ML46651a"  "ML034334a"

#aboral4
head(MCD[order(MCD$aboral4, decreasing = TRUE), 1],5)
#"ML01482a"  "ML20395a"  "ML034334a" "ML46651a"  "ML034336a"

#oral1
head(MCD[order(MCD$oral1, decreasing = TRUE), 1],5)
# "ML20395a"  "ML020045a" "ML04011a"  "ML26358a"  "ML00017a"

#oral2
head(MCD[order(MCD$oral2, decreasing = TRUE), 1],5)
#"ML20395a"  "ML020045a" "ML04011a"  "ML00017a"  "ML26358a"

#oral3
head(MCD[order(MCD$oral3, decreasing = TRUE), 1],5)
#"ML20395a"  "ML004510a" "ML26358a"  "ML00017a"  "ML04011a"

#oral4
head(MCD[order(MCD$oral4, decreasing = TRUE), 1],5)
#"ML20395a"  "ML004510a" "ML46651a"  "ML020045a" "ML00017a"

```
````

````
373
374 3) Calculate mean and standard deviation of each column
375     If the mean is different, then scale the columns so that they all have
376       the same mean for the subsequent questions
377
378 ```{r}
379 MCD
380
381 meanMCD <- c(colMeans(MCD[,2:9]))
382 meanMCD
383
384 sdMCD <- sapply(MCD[,2:9], sd)
385 sdMCD
386
387 ##get all means equal to aboral1- 524.0979
388
389 ## newmean = oldmean x columnconstant
390 ## columnconstant = newmean / oldmean
391 ## newmean = colMeans[1,2]
392 ##old means -- colMeans[1,3:9]--- aboral1 column remanins unchanged
393
394 scaledMCD <- MCD[,1:9]
395 for (i in 2:9){
396   const <- (meanMCD[1])/(meanMCD[i-1])
397   scaledMCD[,i] <- const*scaledMCD[i]
398 }
399 scaledMCD
400
401 #test to see if it worked
402 colMeans(scaledMCD[,2:9])
403 #all means are now 524.0979
404
405 ```
406
407 3) Use correlations between columns to find the samples that are closely
408     related. Is this concordant with the column labels ?
409
410     Yes, aboral-aboral and oral-oral correlations are greater than aboral-oral correlatior
411
412 ```{r}
413
414 cor(scaledMCD[,2:9])
415
416
417 ```
418
419 4) Use correlations between rows to find the closest pairs (top 5)
420    Are these close because they vary a lot between the groups or are they close because th
421 ```{r}
````

```
genenames <- c(scaledMCD$Gene)
g<- t(sapply(scaledMCD, as.numeric))
g[-1,]
colnames(g) <- genenames
g<- g[2:9,]
##This is to flip ScaledMCD, as using transpose alone removes the column names
g

#Remove genes with low expression
g <- g[,-(which(colMeans(g)<5))]


##really big don't run this!
library(HiClimR)
library(reshape2)

rowcor <- fastCor(g, upperTri = TRUE)
q <- subset(melt(rowcor),value>.9)
##Top 5 genes
head(q[order(q$value, decreasing=TRUE),],5)
```

5) If you were forced to divide the genes in each column into high, medium and low count genes, how would you do this based on the data that you have ?
medium = mean +/- SD
low < mean -SD
high > mean +SD

6) make a list of the top 5 genes with most variability and top 5 genes with least variability (exclude genes that have low expression values)

```{r}

#create a dataframe
vardf <- data.frame(matrix(ncol=2,nrow=0))

#loop through scaledMCD and put variance results in second column
##of vardf
for(i in 1:length(scaledMCD$Gene)){
  if(sum(scaledMCD[i,2:9]) >= 0.5){
    tempvar <- c(scaledMCD[i,1], var(as.numeric(scaledMCD[i,2:9])))
    vardf <- rbind(vardf, tempvar)
  } }

colnames(vardf)<- c("gene", "var")
vardf
## highest variance

```

```
head(vardf[order(vardf[,2], decreasing = TRUE), ],5)


head(vardf[order(vardf[,2], decreasing = FALSE), ],5)


```


7) Using the labels of columns provided, find the top variable genes between
the two groups using a t-test list the 5 most up regulated and
  5 most down regulated genes. What happens if you rank by p-value of the t-test ?
  would you exclude some of these genes for having low expression ?
```{r}
##add 2 new columns to scaledMCD to store tvalue and pvalue
for(i in 1:length(scaledMCD$Gene)){
  testt <- t.test(scaledMCD[i,2:5], scaledMCD[i,6:9])
  scaledMCD[i,10]<- testt$statistic
  scaledMCD[i,11]<-testt$p.value
}

#5 most up regulated
#return greatest t value

head(scaledMCD[order(scaledMCD[,10], decreasing = TRUE), ],5   )

# 5 most  down regulated genes
#return negative tvalue
head(scaledMCD[order(scaledMCD[,10], decreasing = FALSE), ],5   )

##small pvalue = significant difference

head(scaledMCD[order(scaledMCD[,11], decreasing = FALSE), ],5   )


```

caroleen21 / Biostatistics-Midterm Public

Code Issues Pull requests Actions Projects Security Insights

main

Biostatistics-Midterm / midterm.Rmd

caroleen21 Add files via upload History

1 contributor

261 lines (182 sloc) | 6.37 KB

```
---
title: "Midterm"
author: "Carolee Nguyen"
date: '2022-10-11'
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
#read data
imagedata = read.csv("/Users/Carol/Downloads/train.csv")

digits = imagedata[1,]

#2) Separate the data by digits, and calculate average value of each pixel

num0 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
num1 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
num2 = matrix(as.numeric(colMeans(train,2:785])),nrow=28, ncol=28)
num3 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
num4 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
num5 = matrix(as.numeric(colMeans(train,2:785])),nrow=28, ncol=28)
num6 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
num7 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)
num8 = matrix(as.numeric(colMeans(train[,2:785])),nrow=28, ncol=28)

#3) Find which columns have the most variance and which have the least.

```

```
30
31 #Over column
32 variancewhole = var(train[,2:785])
33
34 #Over whole datasaet
35 for(i in nrwow:(train)){
36   variance = var(train[,i])
37   print(variance)
38 }
39
40
41 #I am not sure how to answer C and D
42
43 #For each digit
44 var(num0)
45 var(num1)
46 var(num2)
47 var(num3)
48 var(num4)
49 var(num5)
50 var(num6)
51 var(num7)
52 var(num8)
53 var(num9)
54
55 #Create an a 28x28 image
56 image1 = digits
57 as.numeric(imgage1)
58 image1 = matrix(as.numeric(imagedata[1,2:785]),nrow=28, ncol=28)
59 heatmap(image1, col = grey.colors(256, start = 0, end = 1))
60
61
62 install.packages("raster")
63 library(raster)
64 upload = readPNG("/Users/Carol/Downloads/image.png")
65
66
67 #I was not able to get this part
68 picture <- matrix(stuff, nrow=28, ncol=28, byrow=TRUE)
69 heatmap(picture, Rowv = NA, Colv = NA, col = gray.colors(256, start = 0, end = 1))
70
71
72
73 #DATASET 2
74
75 #read data
76 data = read.csv("C:/Users/Carol/Downloads/Mnemiopsis_count_data.csv")
77
78 #1) What are the top 5 genes with the highest average expression (across experiments) in t
```

```
79
80  #Separate the count data from the gene names
81  count = data[2:9]
82
83  #Convert the count data to a dataframe
84  as.data.frame(count)
85  is.data.frame(count)
86
87  ##Calculate the mean for each row
88  means = apply(count, 1, mean)
89
90  #Create a new dataframe that has every gene name and its mean
91  new <- data.frame(genename=1:16548)
92
93  new$genename = data[1]
94
95  new$meanvalue = means
96
97  as.data.frame(new)
98  is.data.frame(new)
99
100 #Sort the mean from greatest to lowest mean value
101 df <- new[order(new$meanvalue, decreasing = TRUE),]
102
103 #View the top 5 genes with the greatest mean
104 head(df, 5)
105
106 #ML20395a 122241.38
107 #ML26358a  64737.00
108 #ML46651a  62309.25
109 #ML020045a  54829.00
110 #ML00017a  554044.12
111
112 #2) Are the top 5 genes different if they are done on a per column basis ?
113
114 #Separate oral and aboral data
115 aboral = data[2:5]
116 oral = data[6:9]
117
118 aboralmeans = apply(aboral, 1, mean)
119 oralmeans = apply(oral, 1, mean)
120
121 #Aboral
122 aboraldata <- data.frame(genename=1:16548)
123
124 aboraldata$genename = data[1]
125
126 aboraldata$meanvalue = aboralmeans
127
```

```
as.data.frame(aboraldata)
is.data.frame(aboraldata)

df1 <- aboraldata[order(aboraldata$meanvalue, decreasing = TRUE),]

head(df1, 5)

#ML20395a 125348.50
#ML46651a  97676.00
#ML01482a  79597.25
#ML26358a  77876.00
#ML034334a  64857.25


#Oral

oraldata <- data.frame(genename=1:16548)

oraldata$genename = data[1]

oraldata$meanvalue = oralmeans

as.data.frame(oraldata)
is.data.frame(oraldata)

df2 <- oraldata[order(oraldata$meanvalue, decreasing = TRUE),]

head(df2, 5)

#ML20395a 119134.25
#ML26358a  51598.00
#ML020045a  50998.50
#ML00017a  50607.50
#ML04011a  50153.75


#3) Calculate mean and standard deviation of each column, If the mean is different, then s

meancol = colMeans(count)
print(meancol)

sd = colSds(as.matrix(count[sapply(count, is.numeric)]))
print(sd)

df3 <- data.frame(Mean=1:8)
df3$StandardDev = c(1:8)
df3$StandardDev = sd
df3$Mean[c(1:8)] = meancol

```

```
rownames(df3) <- colnames(count)

print(df3)

```
x = df3[1,1]
print(x)


#Scaling factors
value2 = x/(df3[2,1])
value3 = x/df3[3,1]
value4 = x/df3[4,1]
value5 = x/df3[5,1]
value6 = x/df3[6,1]
value7 = x/df3[7,1]
value8 = x/df3[8,1]
```

```
#Scaled values
new2 = value2*(df3[2,])
new3 = value3*(df3[3,])
new4 = value4*(df3[4,])
new5 = value5*(df3[5,])
new6 = value6*(df3[6,])
new7 = value7*(df3[7,])
new8 = value8*(df3[8,])

```
df4 = data.frame(Mean=1:8)
df4$StandardDev = c(1:8)
df4[1,] = df3[1,]
df4[2,] = new2
df4[3,] = new3
df4[4,] = new4
df4[5,] = new5
df4[6,] = new6
df4[7,] = new7
df4[8,] = new8

rownames(df4) = colnames(count)

print(df4)

as.data.frame(df4)
is.data.frame(df4)

```

```
newdf4 = t(df4)

#3) Use correlations between columns to find the samples that are closely related. Is this
cor(count)
cor(t(count))

Yes, they are concordant with the column labels

#ttest

for (row in 1:nrow(stock)) {
t.test(aboralmeans, oralmeans)$p.value

5) If you were forced to divide the genes in each column into high, medium and
low count genes, how would you do this based on the data that you have ?

I am not sure, either calculate the means for each row and define a threshold for high, me

6) make a list of the top 5 genes with most variability and top 5 genes with
least variability (exclude genes that have low expression values)

#I think we use Ryans method for this
rowcor <- cor(count[,-1],)

7) Using the labels of columns provided, find the top variable genes between
the two groups using a t-test list the 5 most up regulated and
  5 most down regulated genes. What happens if you rank by p-value of the t-test ?
  would you exclude some of these genes for having low expression ?

for(i in 1:nrow(data)){
  testt = t.test(data[,2:5], data[,6:9])
  pvalue = testt$p.value
  print(pvalue)
}


```

Ryan-Oblivion / **Biostat-Midterm** Public

Code Issues Pull requests Actions Projects Security Insights

86cc4c41c9 ···

Biostat-Midterm / Midterm_project_biostatistics_final.ipynb

**Ryan-Oblivion** use this as final code History

1 contributor

6.1 MB ···

In [ ]:
```
names
Carolee Nguyen
Sarah Feraidoon
Tarabeth Gerrity
Ryan Johnson
```

In [1]:
```
1+1
```

In [2]:
```
data <- read.csv("train.csv.gz")
# Taking the train file and giving it the variable name data
data

#length(data[,1])
#42000

pixels <- data.frame(data[,-1])
# we want to remove the first column from the file, so we only have pixel

#smoothScatter(pixels)
```

1	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
5	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
8	0	0	0	0	0	0	0	0	0	...	0
9	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	0	0	0	...	0

5	0	0	0	0	0	0	0	0	0	...	0
8	0	0	0	0	0	0	0	0	0	...	0
6	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
6	0	0	0	0	0	0	0	0	0	...	0
9	0	0	0	0	0	0	0	0	0	...	0
9	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...		...
2	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
9	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	0	0	0	...	27
2	0	0	0	0	0	0	0	0	0	...	0
8	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
5	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
5	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	...	0
9	0	0	0	0	0	0	0	0	0	...	0
6	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0		0

4	0	0	0	0	0	0	0	0	0	...	0
0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	0	0	0	...	0
6	0	0	0	0	0	0	0	0	0	...	0
9	0	0	0	0	0	0	0	0	0	...	0

In [97]:

```
#library(raster)

# This cell is to test out how to turn one row into a matrix and turn it
# into the image
# Then we will make a function and a for loop to do this for every row

label1 <- pixels[1,]
# now getting the first row into a variable called label1
# this is just to visualize the frist row as its own data set

#mode(label1)
mode(label1) = "numeric"
# now we need to make the mode of this label1 data into numeric so it wo
# the image method


my_label1 <- matrix(label1, nrow=28, byrow= TRUE)
# this turns our label into a 28x28 matrix

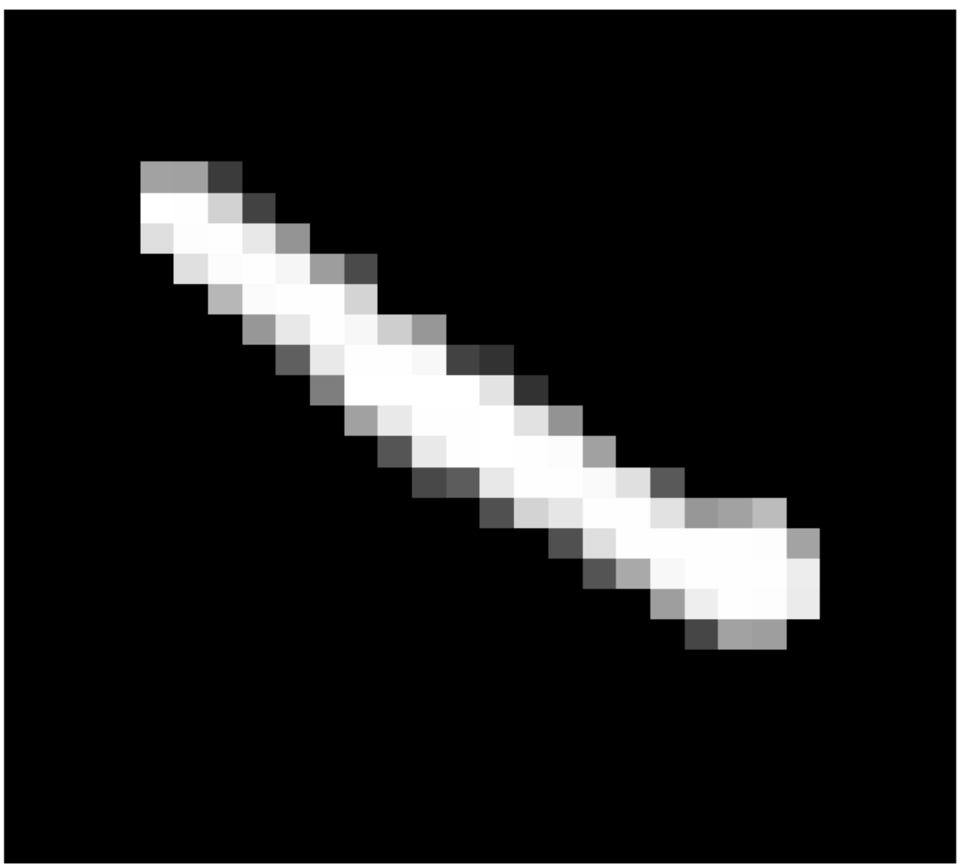
#par(mar= c(5,5,5,5))
image(my_label1, axes =FALSE, col = grey.colors(256, start =0, end=1))
# and now we can print the values in the matrix as an image
# where 0 is black and 256 is white, and anything inbetween as shades of
# this prints the image also

my_label1
# next we print the matrix

#matplot((my_label1), type= "l")
```

0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	188	255	94	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	191	250	253	93	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	123	248	253	167	10	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	247	253	208	13	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	253	235	77	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	253	88	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	170	17	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	22	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	103	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	89	240	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	15	220	253	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	94	253	253	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	89	251	253	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	214	218	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0



In [4]:

```
# number 1

# this cell is where we do the same as the above cell but for every row

r  <- c(1:length(pixels[,1]))
# we want to make a variable that keeps track of how many rows are
# in the pixels dataframe

for (x in r){
    # now we want to loop through each number in r, which represents the
    # current row in the dataframe pixels

    label1 <- pixels[x,]
    # next we want to pass that entire row into the variable
    # called label1

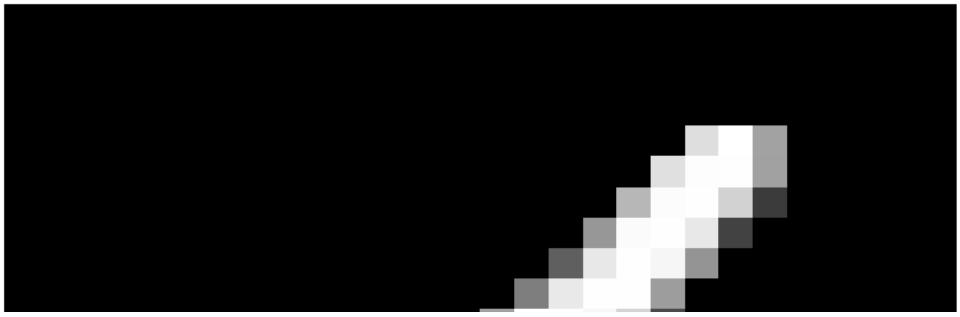
    mode(label1) = "numeric"
    # we need to change the mode of each number in that row to numeric
    # so the image method can work with it

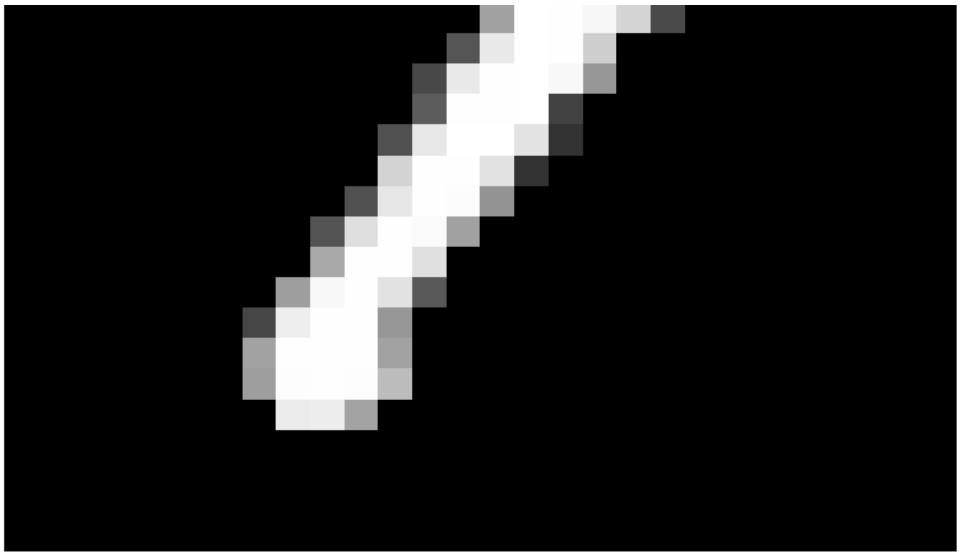
    my_label1 <- matrix(rev(label1), nrow=28, byrow= FALSE)
    # now we make that row into a 28x28 matrix called my_label1

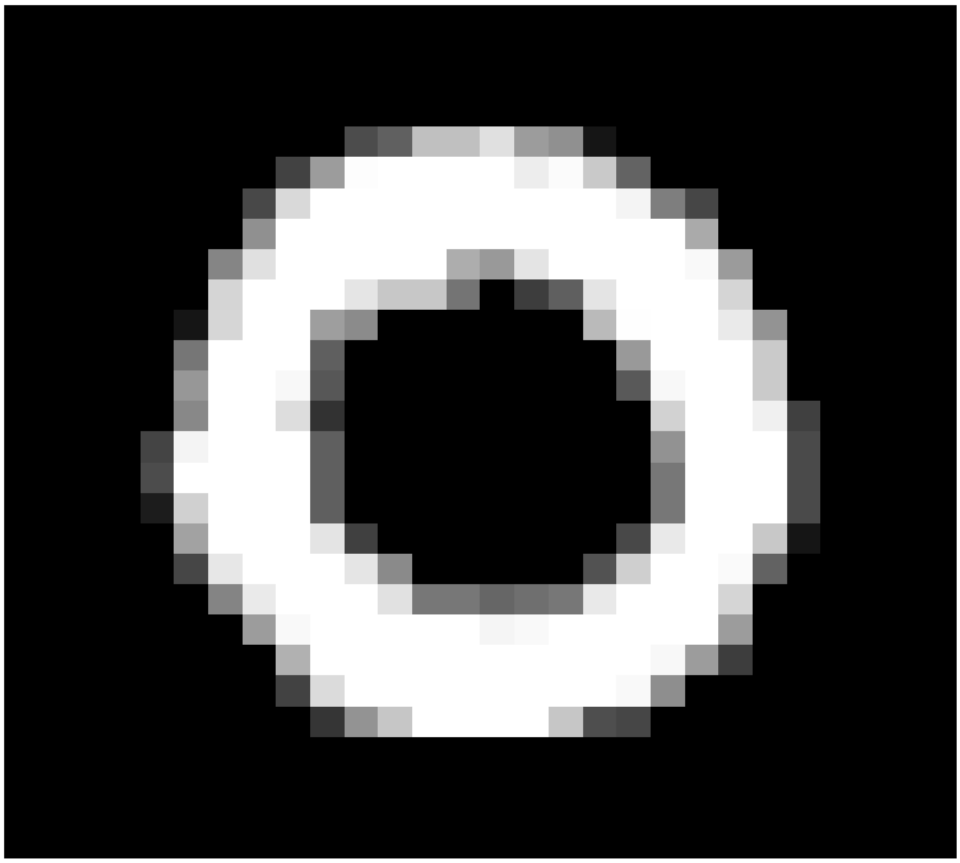

    my_label1 <- apply(my_label1,2,rev)

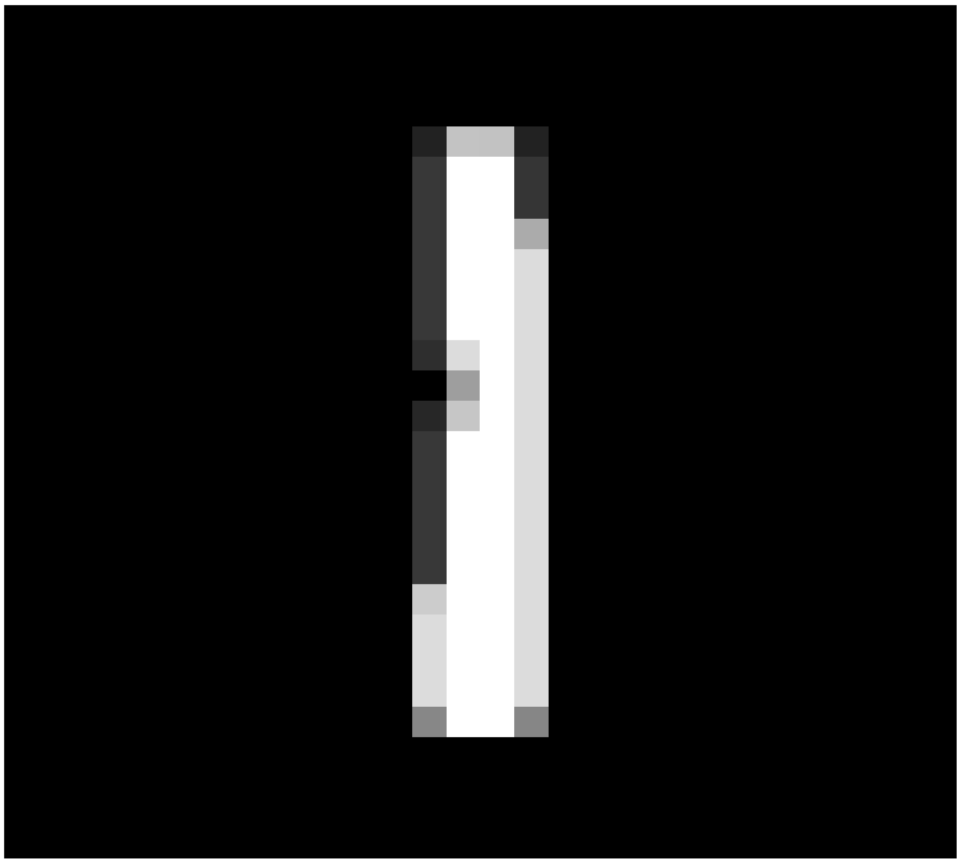
    #par(mar= c(5,5,5,5))
    image(my_label1, axes =FALSE, col = grey.colors(256, start =0, end=1)
    # now we use image to plot all the points in the matrix
    # 0 being black and 256 being white, and everything inbetween being
    # shades of grey
    #my_label1
    if (x ==10){ break }
    # since there are 64k rows, we used a break statement at 10
    # just to see how the first 10 hand written digits looked
    # the code will run much longer if we had not done this
    # in the next few cells we take the average of each number
    # which is much better at combining all of the data into
    # smaller parts

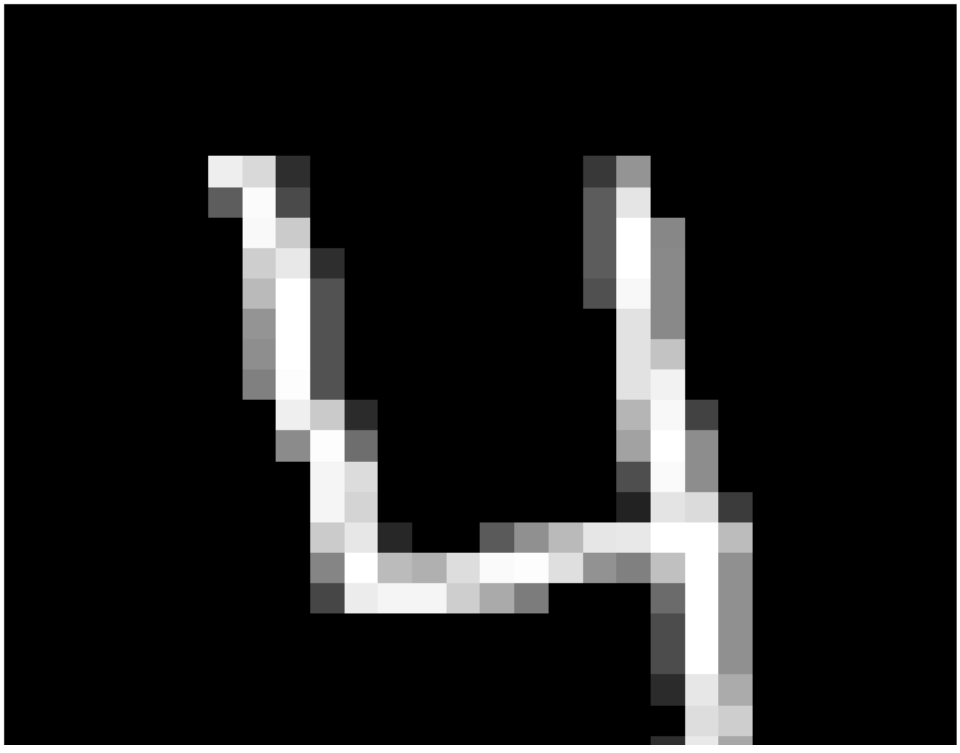
}
```

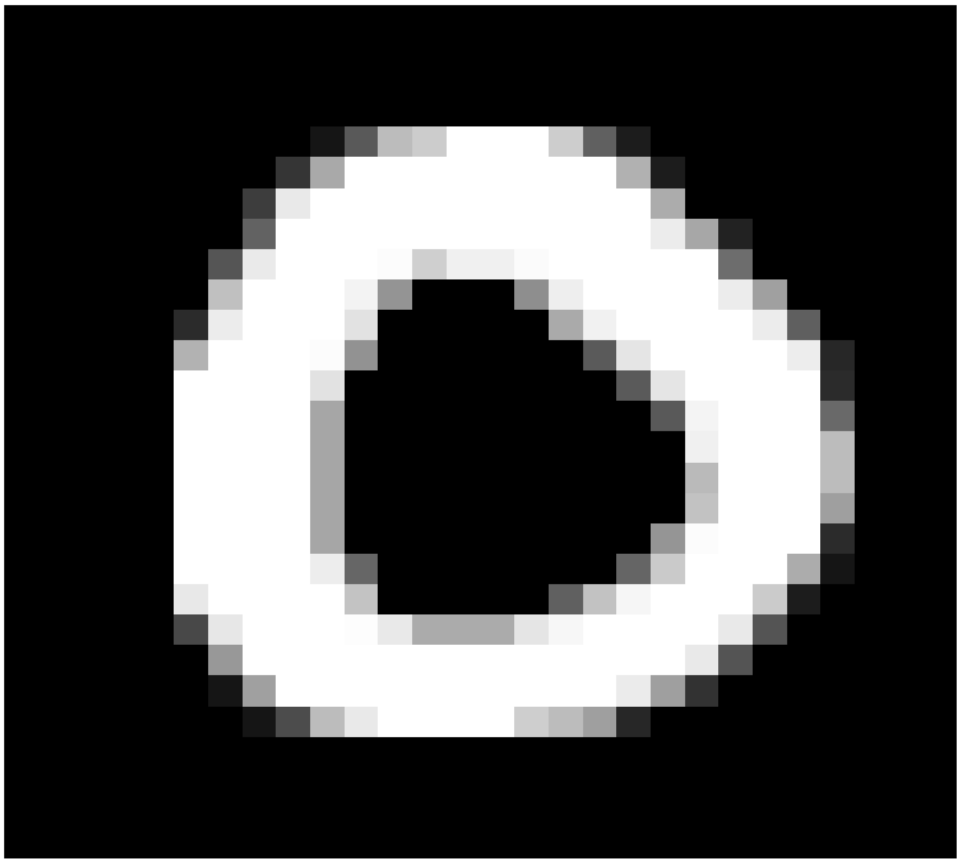


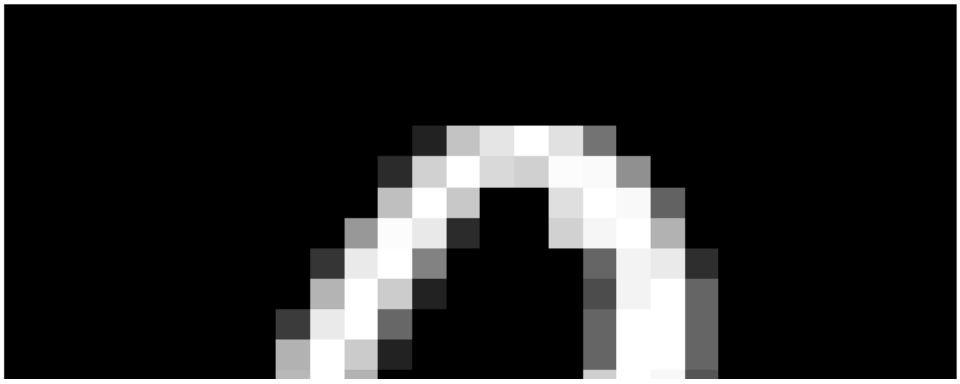


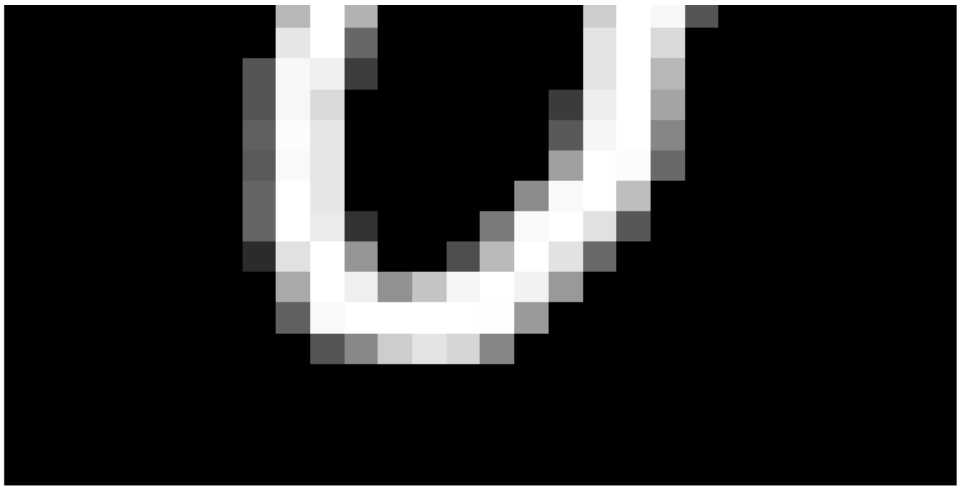


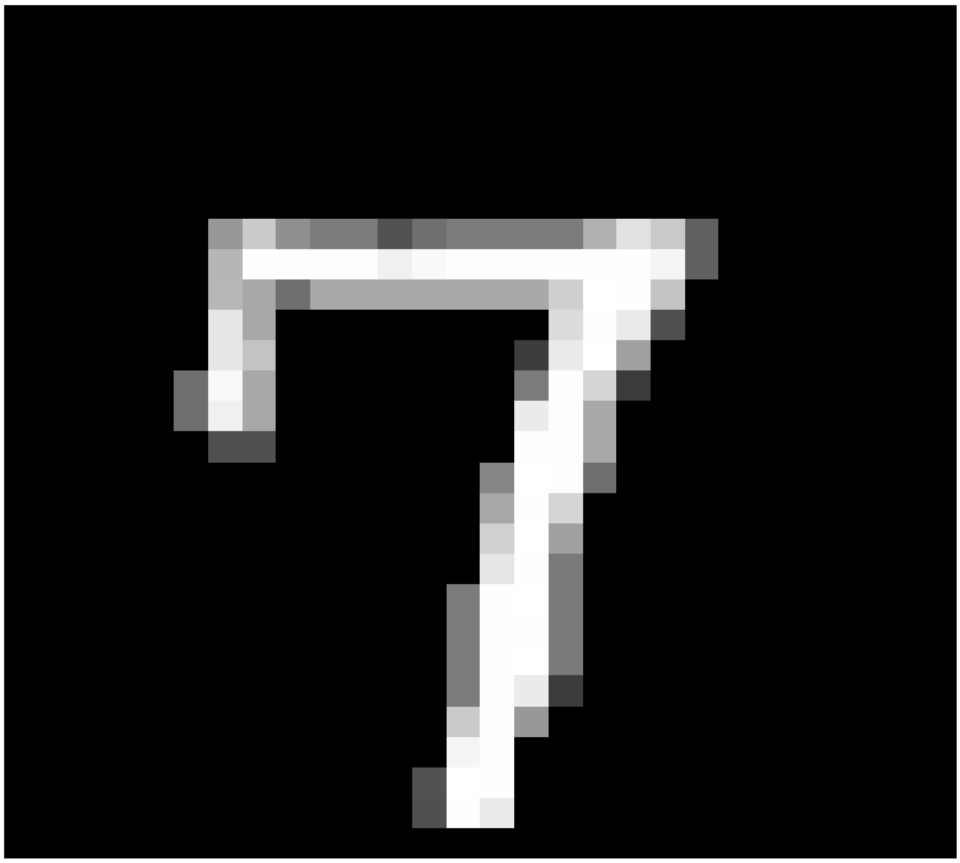


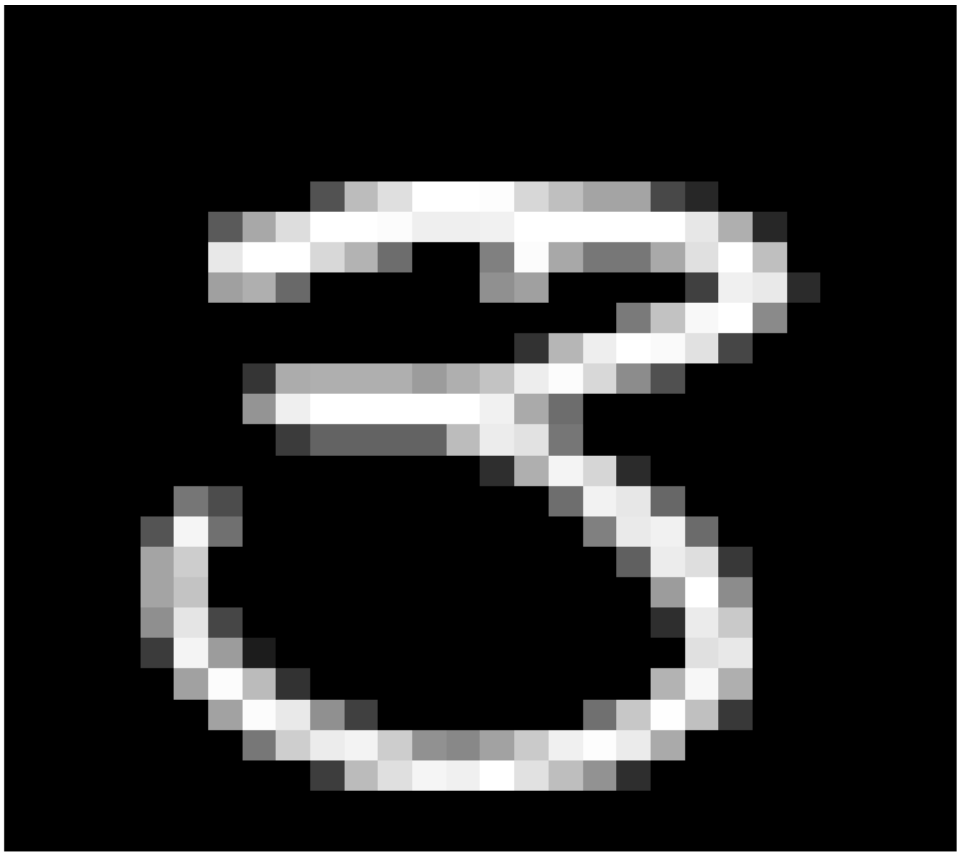


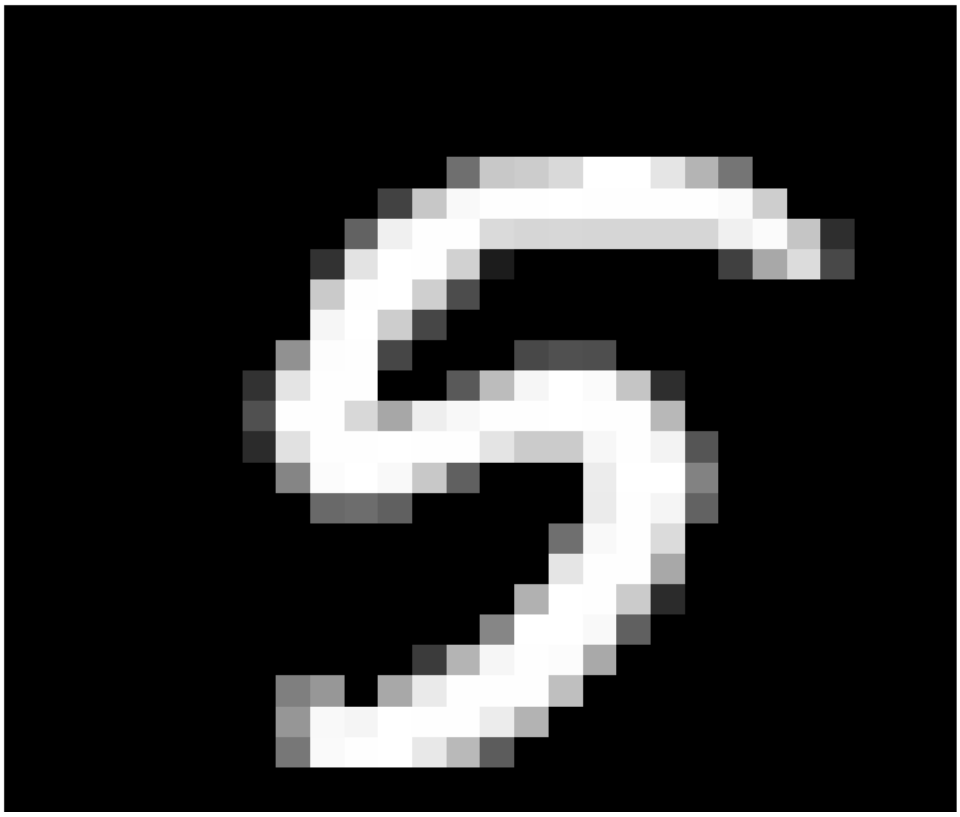


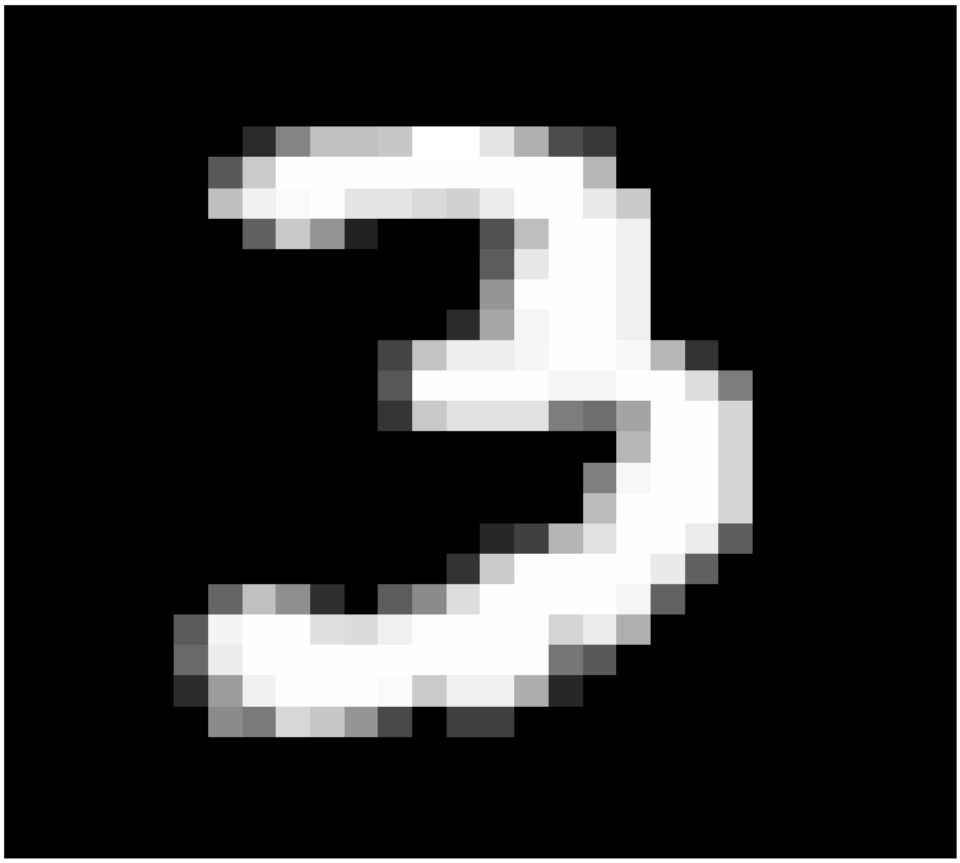












In [99]:

```
# number 2
n <- c(0:9)
# we want to keep track of all the digits that correspond to
# a handwriting. from 0 to 9 as the variable n

for (x in n){
    # next we loop through each digit

    assign(paste0("label_",x), data[grep(x,data[,1]),])
    # we use assign to put all the data that we wish to grep into
    # a variable. we use paste0 to create variable names dynamically
    # then grep to get the digit which is the pattern that
    # the current loop is on from the first column. we then put
    # that entire row that matches the current digit into a variable
    # then the loop moves onto the next digit.
```

```
}
#label_0
label_1
# here we show that it works for label_1 and before that we checked
# to see if it works for label_0 which it does.
# So this code will work for all label_(0-9)


#data[grep("0", data[,1]),]
```

1	1	0	0	0	0	0	0	0	0	0	...
3	1	0	0	0	0	0	0	0	0	0	...
13	1	0	0	0	0	0	0	0	0	0	...
16	1	0	0	0	0	0	0	0	0	0	...
36	1	0	0	0	0	0	0	0	0	0	...
38	1	0	0	0	0	0	0	0	0	0	...
39	1	0	0	0	0	0	0	0	0	0	...
42	1	0	0	0	0	0	0	0	0	0	...
53	1	0	0	0	0	0	0	0	0	0	...
60	1	0	0	0	0	0	0	0	0	0	...
61	1	0	0	0	0	0	0	0	0	0	...
62	1	0	0	0	0	0	0	0	0	0	...
69	1	0	0	0	0	0	0	0	0	0	...
78	1	0	0	0	0	0	0	0	0	0	...
80	1	0	0	0	0	0	0	0	0	0	...
97	1	0	0	0	0	0	0	0	0	0	...
119	1	0	0	0	0	0	0	0	0	0	...
125	1	0	0	0	0	0	0	0	0	0	...
135	1	0	0	0	0	0	0	0	0	0	...
141	1	0	0	0	0	0	0	0	0	0	...
153	1	0	0	0	0	0	0	0	0	0	...
183	1	0	0	0	0	0	0	0	0	0	...
186	1	0	0	0	0	0	0	0	0	0	...
192	1	0	0	0	0	0	0	0	0	0	...
210	1	0	0	0	0	0	0	0	0	0	...
222	1	0	0	0	0	0	0	0	0	0	...
225	1	0	0	0	0	0	0	0	0	0	...
250	1	0	0	0	0	0	0	0	0	0	...
256	1	0	0	0	0	0	0	0	0	0	...

257	1	0	0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	
41821	1	0	0	0	0	0	0	0	0	0	...
41827	1	0	0	0	0	0	0	0	0	0	...
41832	1	0	0	0	0	0	0	0	0	0	...
41852	1	0	0	0	0	0	0	0	0	0	...
41853	1	0	0	0	0	0	0	0	0	0	...
41855	1	0	0	0	0	0	0	0	0	0	...
41862	1	0	0	0	0	0	0	0	0	0	...
41863	1	0	0	0	0	0	0	0	0	0	...
41867	1	0	0	0	0	0	0	0	0	0	...
41879	1	0	0	0	0	0	0	0	0	0	...
41881	1	0	0	0	0	0	0	0	0	0	...
41883	1	0	0	0	0	0	0	0	0	0	...
41890	1	0	0	0	0	0	0	0	0	0	...
41897	1	0	0	0	0	0	0	0	0	0	...
41900	1	0	0	0	0	0	0	0	0	0	...
41905	1	0	0	0	0	0	0	0	0	0	...
41912	1	0	0	0	0	0	0	0	0	0	...
41924	1	0	0	0	0	0	0	0	0	0	...
41927	1	0	0	0	0	0	0	0	0	0	...
41933	1	0	0	0	0	0	0	0	0	0	...
41934	1	0	0	0	0	0	0	0	0	0	...
41937	1	0	0	0	0	0	0	0	0	0	...
41939	1	0	0	0	0	0	0	0	0	0	...
41940	1	0	0	0	0	0	0	0	0	0	...
41952	1	0	0	0	0	0	0	0	0	0	...
41953	1	0	0	0	0	0	0	0	0	0	...
41955	1	0	0	0	0	0	0	0	0	0	...
41961	1	0	0	0	0	0	0	0	0	0	...
41992	1	0	0	0	0	0	0	0	0	0	...
41997	1	0	0	0	0	0	0	0	0	0	...

In [5]:

```
# number 2a

# this code it to just take the average of one single label number which
# label_0. we are doing this just to test on one lable then will make a
# function in the next cell to pass in all the labels_(0-9)
```

```
pixel_label_0  <- label_0[,-1]
# taking all the data that has label hand writting as 0 and removing
# the first column so we only have the pixels that represent 0
# and we put that into a variable called pixel_label_0

#length(pixel_label_0)
c <- c(1:length(pixel_label_0))
# now we want to make a variable that keeps track of every a number from
# 1 to the max length of the columns this data consists

ave_pix <- c()
# we want to have an empty list ready for when we want to append the
# average values of each column

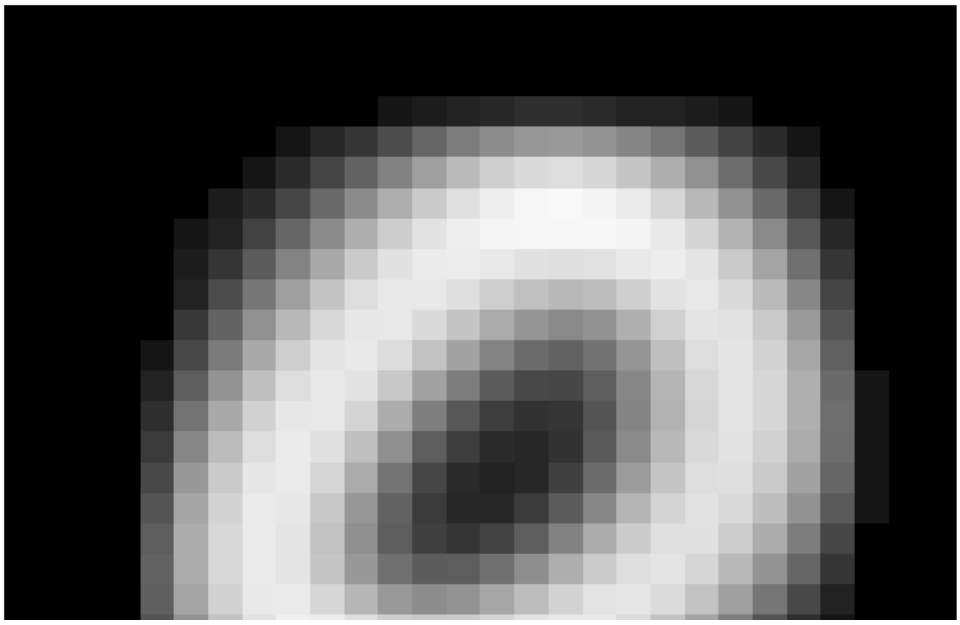
for (x in c){
    # we want to loop through my c variable with x, then we will make
    # x the column in the variable pixel_label_0

    #pixel_label_0[,x]
    ave_pix <- append(ave_pix,mean(pixel_label_0[,x]))
    # with each for loop we will look at the column and take the mean of
    # that entire column, then append the mean to the variable ave_pix
}

matrix_ave_pix <- matrix(rev(ave_pix), nrow=28, ncol=28)
# here we take the list of means from ave_pix to make a matrix, where
# each entry represents the mean number for each column in the
# dataset labeled 0.

matrix_ave_pix <- apply(matrix_ave_pix,2,rev)

image(matrix_ave_pix, axes = FALSE, col = grey.colors(256, start=0, end=1
# here we plot the matrix using image function, and make the colors
# shades of grey
#pixel_label_0
```



In [ ]:

In [7]:

```
# number 2b

# this is the cell where we pass in all the label_(0-9) take the averages
# and look at if the images are still eligible

myfunction <- function(r){
    # here we used the same code from the last cell but put it in a
    # function so we can pass all the labels in without rewriting the
    # code 10 times

    pixel_label_0  <- r[,-1]
# taking all the data that has label hand writting as 0 and removing
# the first column so we only have the pixels that represent 0
# and we put that into a variable called pixel_label_0

#length(pixel_label_0)
    c <- c(1:length(pixel_label_0))
# now we want to make a variable that keeps track of every a number from
# 1 to the max length of the columns this data consists

    ave_pix <- c()
# we want to have an empty list ready for when we want to append the
# average values of each column

    for (x in c){
    # we want to loop through my c variable with x, then we will make
    # x the column in the variable pixel_label_0

    #pixel_label_0[,x]
        ave_pix <- append(ave_pix,mean(pixel_label_0[,x]))
    # with each for loop we will look at the column and take the mean of
    # that entire column, then append the mean to the variable ave_pix
    }

    matrix_ave_pix <- matrix(rev(ave_pix), nrow=28, ncol=28)

# here we take the list of means from ave_pix reversed to make a matrix,
# each entry represents the mean number for each column in the
# dataset labeled 0.

    matrix_ave_pix <- apply(matrix_ave_pix,2,rev)
    # here we needed to reverse the matrix again so it is in the correct
    # orientation
```

```
    image(matrix_ave_pix, axes = FALSE, col = grey.colors(256, start=0, e
# here we plot the matrix using image function, and make the colors
# shades of grey
#pixel_label_0





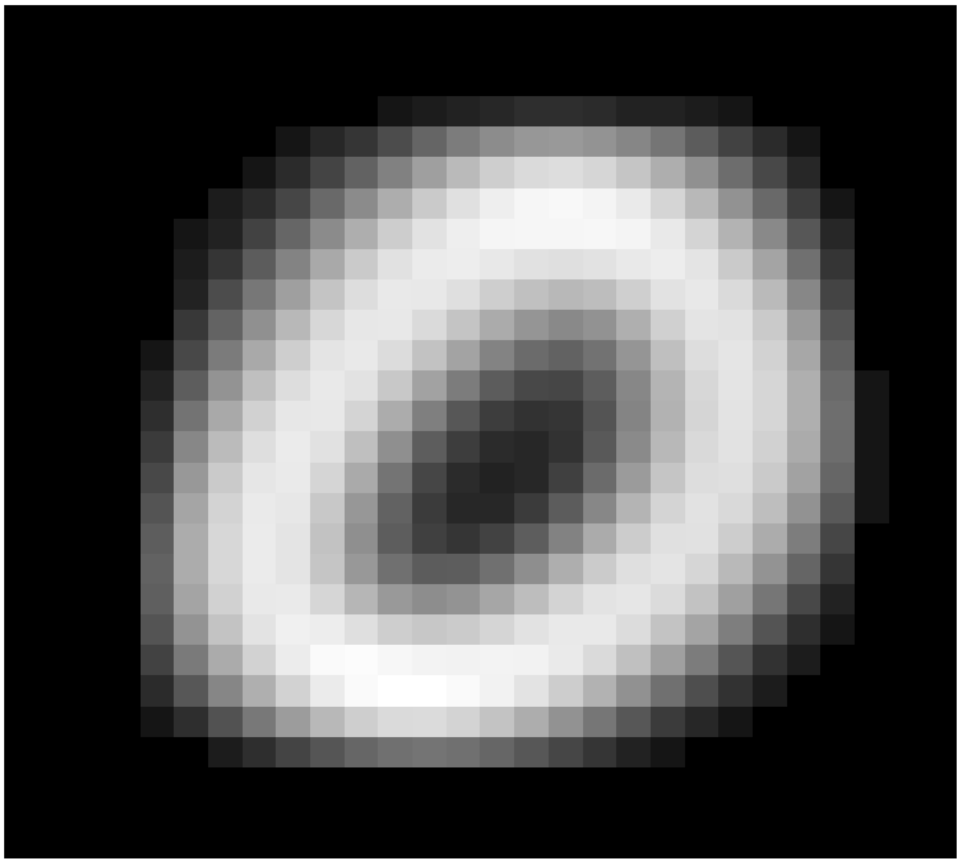


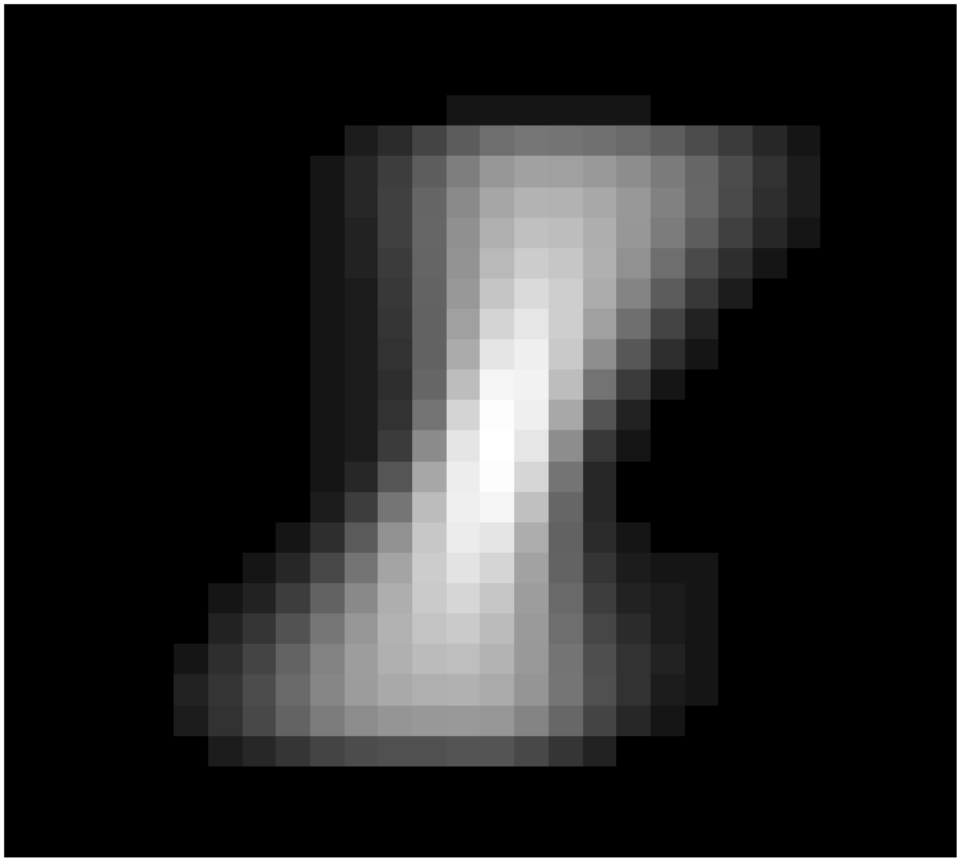


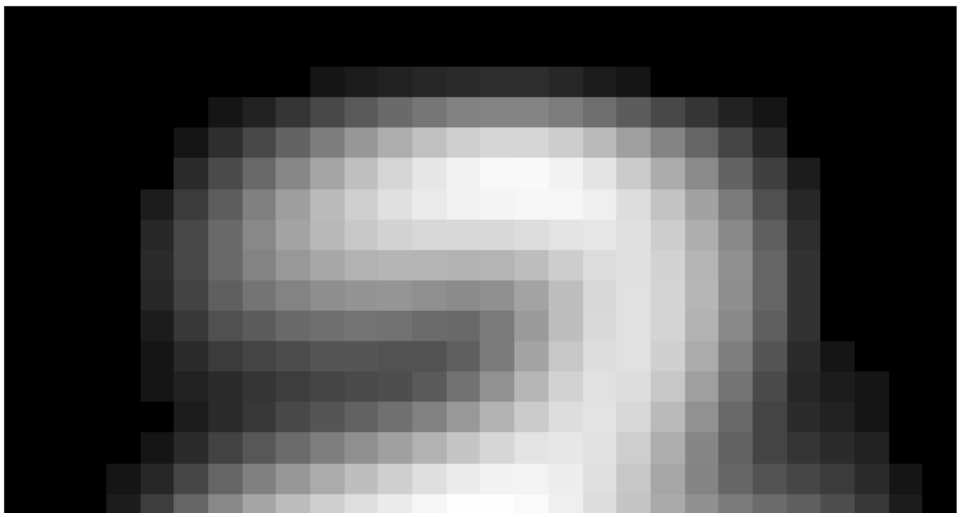
}

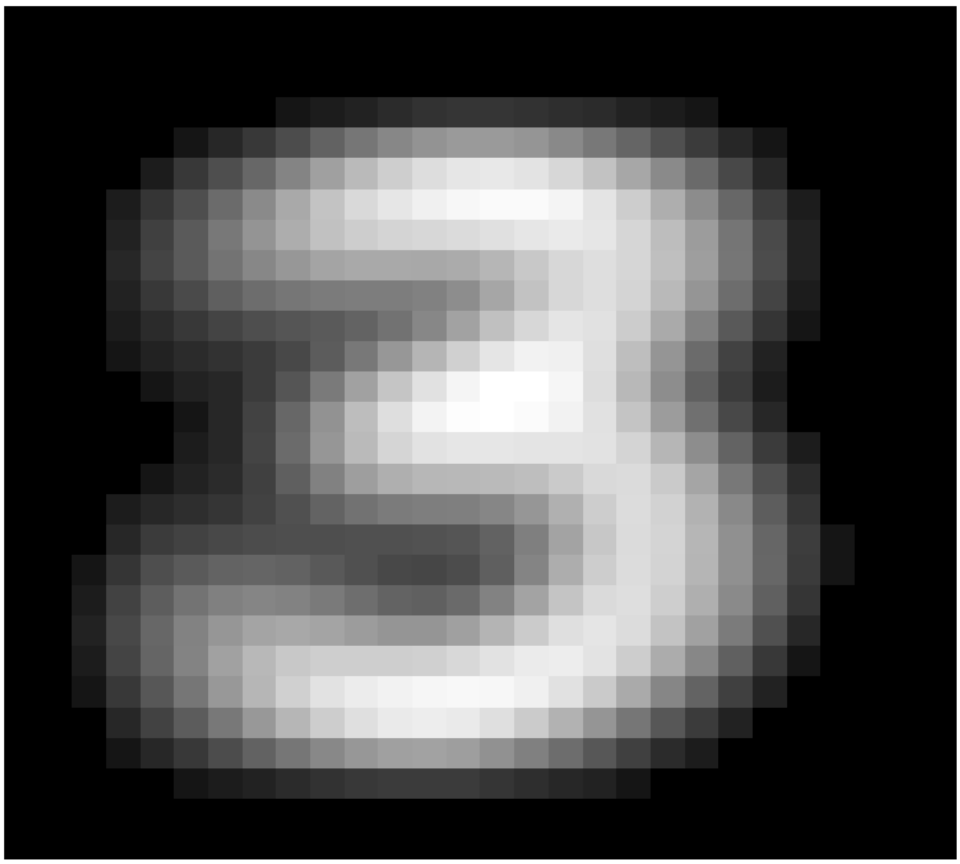
myfunction(label_0)
myfunction(label_1)
myfunction(label_2)
myfunction(label_3)
myfunction(label_4)
myfunction(label_5)
myfunction(label_6)
myfunction(label_7)
myfunction(label_8)
myfunction(label_9)

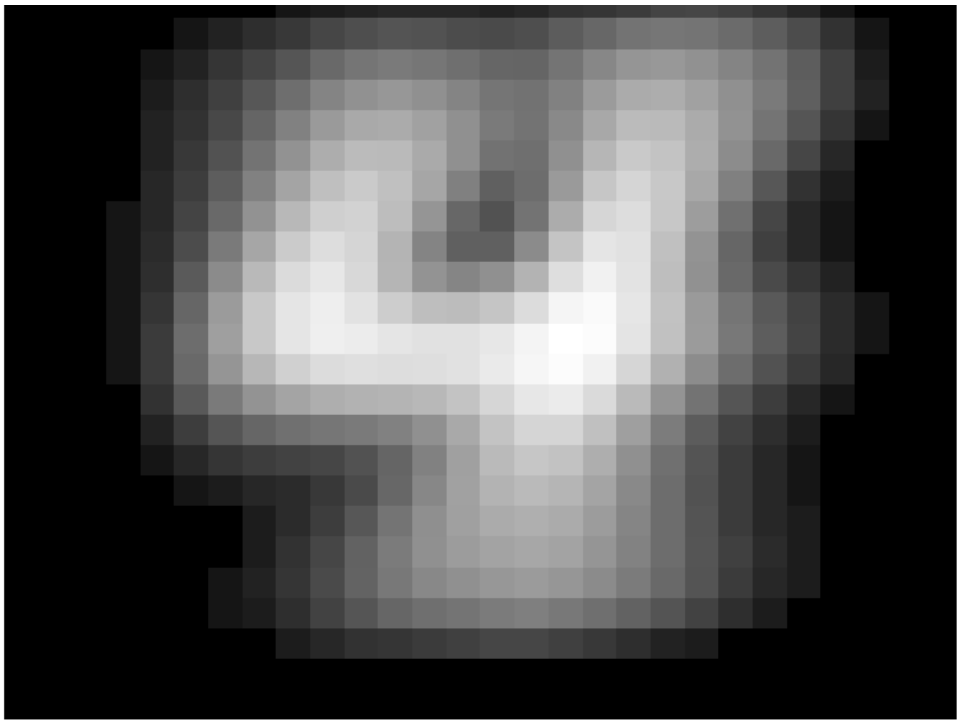
# to answer the question 2a and 2b, yes we think they still do resemble
# digit labels. under this operation we think 0,1,2,3,8 look the best
```

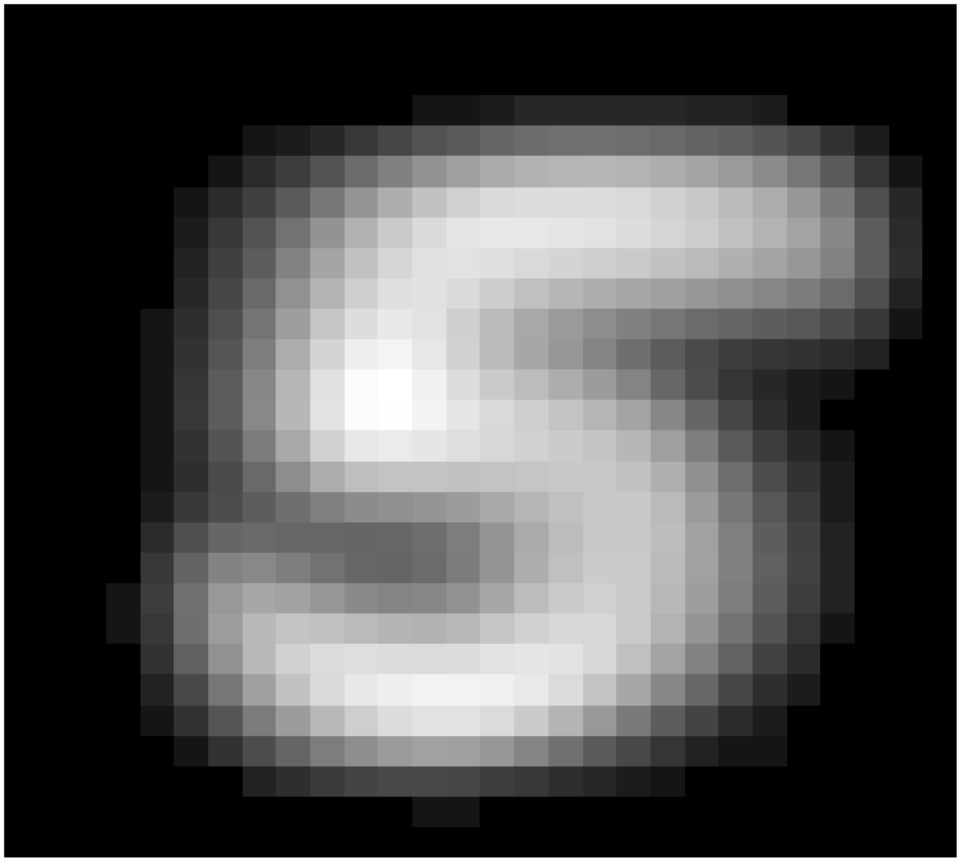


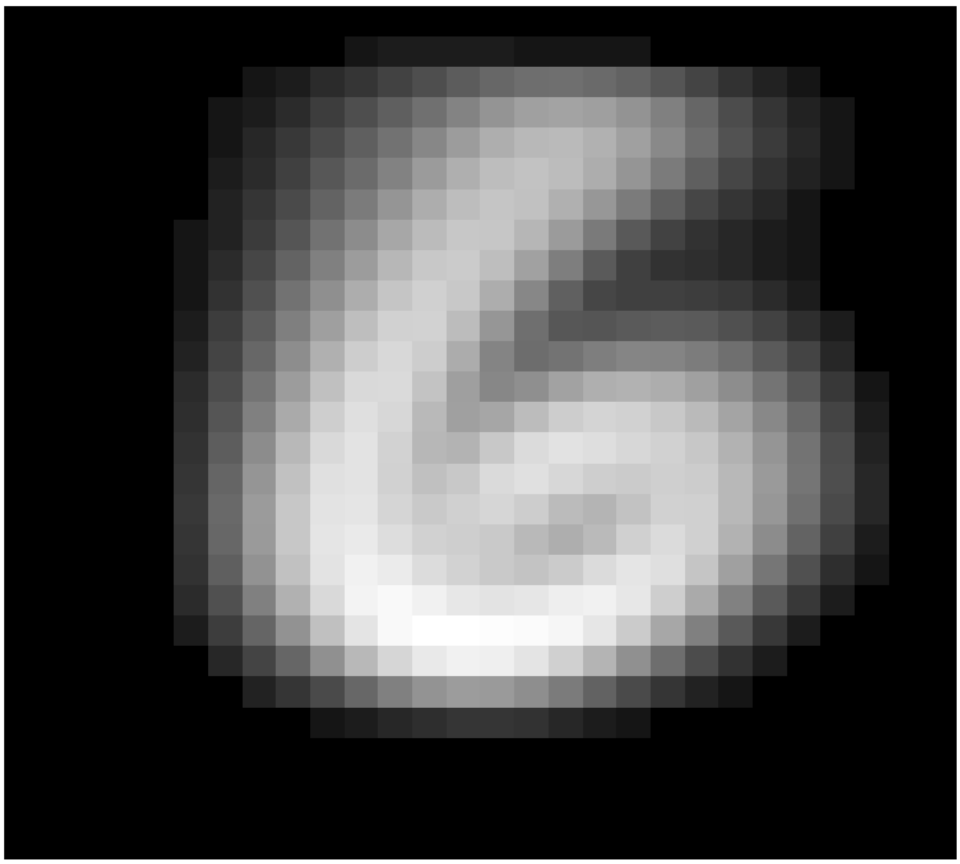


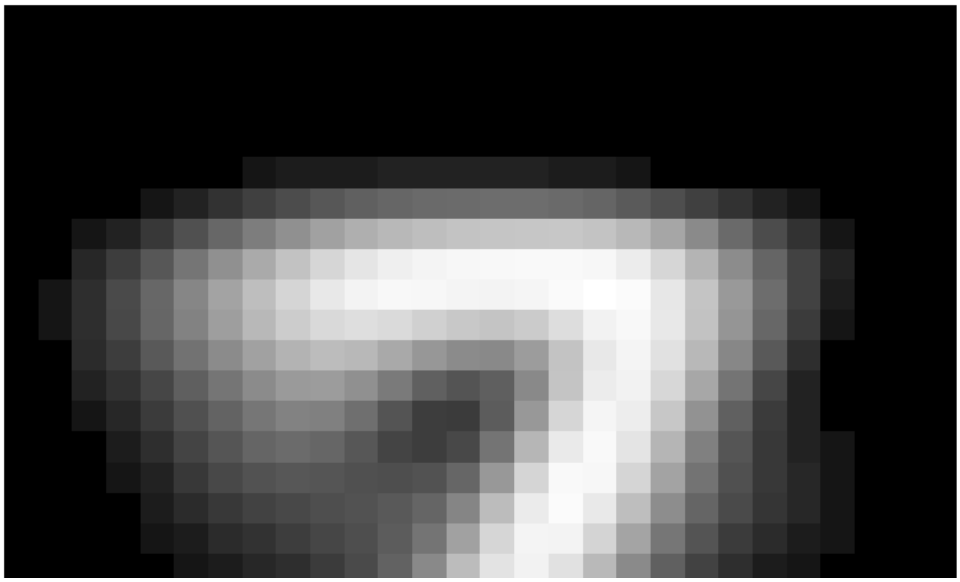


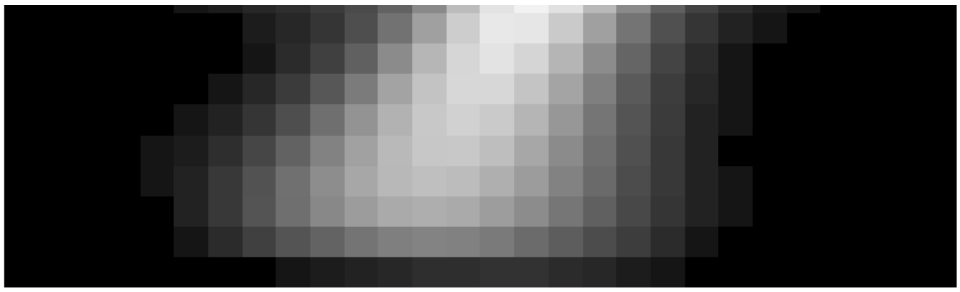


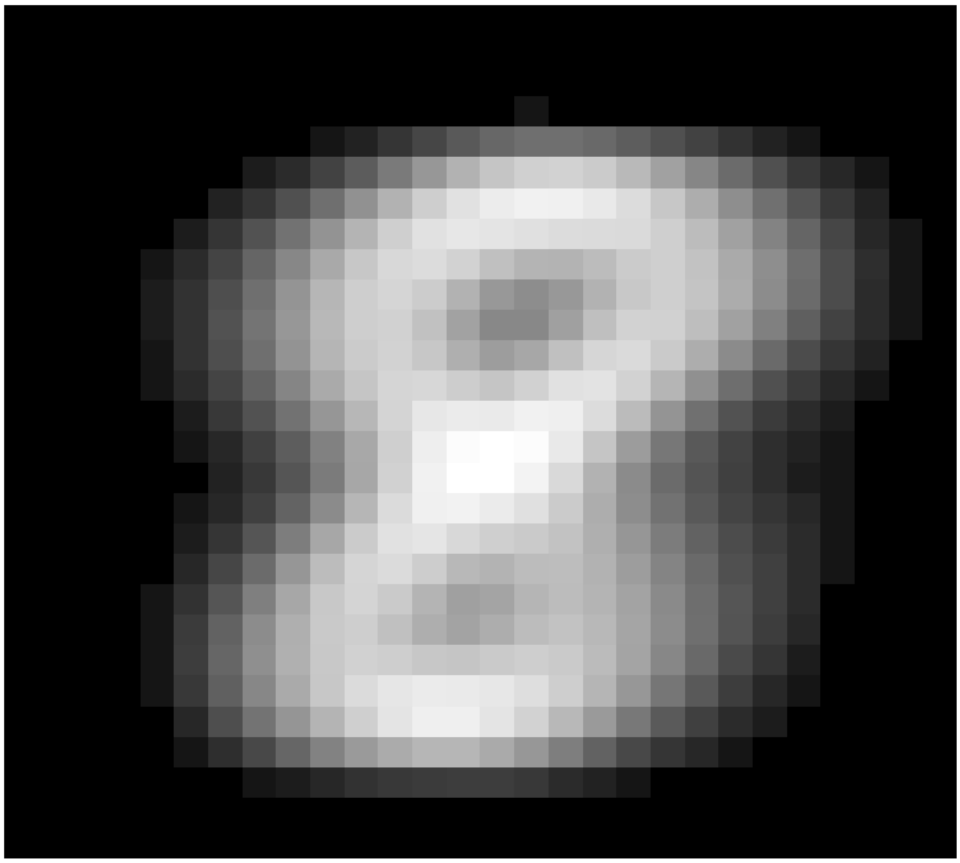


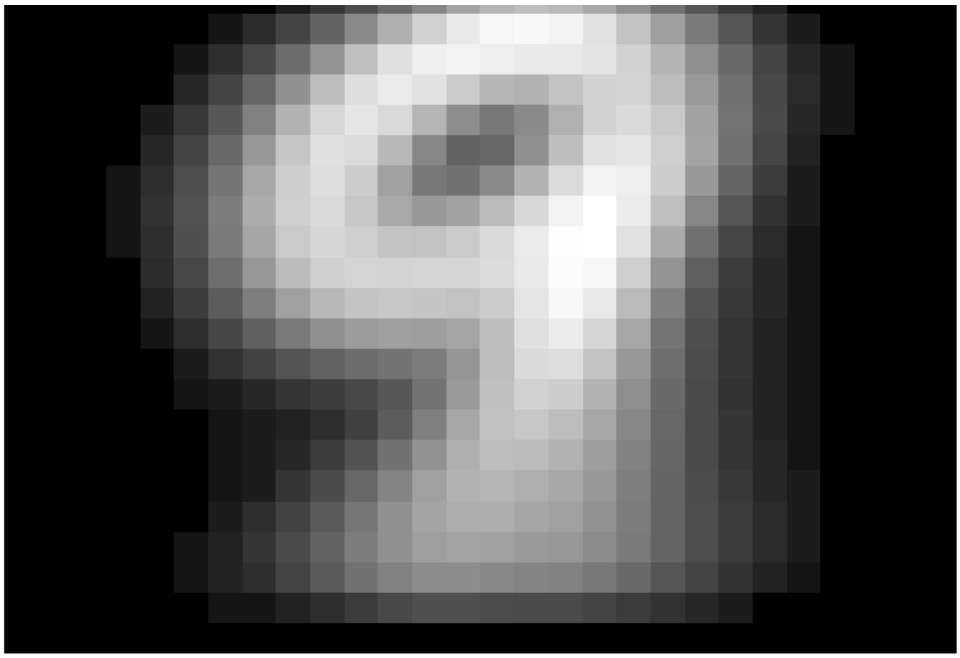












In [104…

```
# number 3a

# this cell found the variance for only the full data set.
# the next cell finds the variance for all labels 0-9

length(data)
c <- c(2:length(data))
# the variable c is to keep track of the column numbered 2 - 785

var_of_col <- c()
# we wanted to make an empty variable to pass all of the variance
# results to

for (x in c){
    #now we loop through the columns

    var_of_col <- append(var_of_col, var(data[,x]))
    # in each column i will calculate the variance of that entrie column
    # then append that to the variable var_of_col
}
# max(var_of_col)
# now we want to find the max number that exists in var_of_col
which(var_of_col == max(var_of_col))
max_vari <- which(var_of_col == max(var_of_col))+ 1

max_vari

# in the full dataset we wanted to see which column had the most
# variance, and it looks like that is col 408
# we didnt include the first column in the list generated so we needed
# to add an extra 1 to it.
var(data[,408])
# so in the full dataset the column that has the greatest variance is
# column 408
```

```
# column 408
```

In [7]:

```
# number 3a
func_each_label  <- function(L,k){
    #length(L)
    c <- c(2:length(L))
# the variable c is to keep track of the column numbered 2 - 785

    var_of_col <- c()
# i wanted to make an empty variable to pass all of the variance
# results to

    for (x in c){
    #now i loop through the columns

        var_of_col <- append(var_of_col, var(L[,x]))
    # in each column i will calculate the variance of that entrie column
    # then append that to the variable var_of_col
    }
    #(max(var_of_col))
# now i want to find the max number that exists in var_of_col

    cat("the column number for ",k," that has the highest variance is: ",
        which(var_of_col == max(var_of_col))+1,
        " and the variance is ",print(max(var_of_col)), "\n")
# in the full dataset i wanted to see which column had the most
# variance, and it looks like that is col 408
# i didnt include the first column in the list generated so i needed
# to add an extra 1 to it.
    #var(L[,which(var_of_col == max(var_of_col))])
# so in the full dataset the columb that has the greatest variance is
# column 408

}

func_each_label(label_0,"label_0")
func_each_label(label_1,"label_1")
func_each_label(label_2,"label_2")
func_each_label(label_3,"label_3")
func_each_label(label_4,"label_4")
func_each_label(label_5,"label_5")
func_each_label(label_6,"label_6")
func_each_label(label_7,"label_7")
func_each_label(label_8,"label_8")
func_each_label(label_9,"label_9")

# how this relates to the answer in 2b is that it seems the greater the
# variance the less visible the digit looks. and in this case we think be
# the label_2 has the highest variance it looks a bit more blurry
```

In [ ]:

```
```

In [105…

```
# number 4
# I didnt complete number 4 so use it from a different group member
library(imager)

x <- c(0:9)

for (n in x){

k <- paste0(n,"_writing.png")

im <- load.image(k)
#im <- load.image(file)
im.r <- as.raster(im, interpolate=F)
str(im.r)
dim(im.r)
plot(im.r)

}
dev.off()
```

```
Warning message:
"package 'imager' was built under R version 3.6.3"Loading required packag
e: magrittr
Warning message:
"package 'magrittr' was built under R version 3.6.3"
Attaching package: 'imager'

The following object is masked from 'package:magrittr':
```

```
    add

The following objects are masked from 'package:stats':

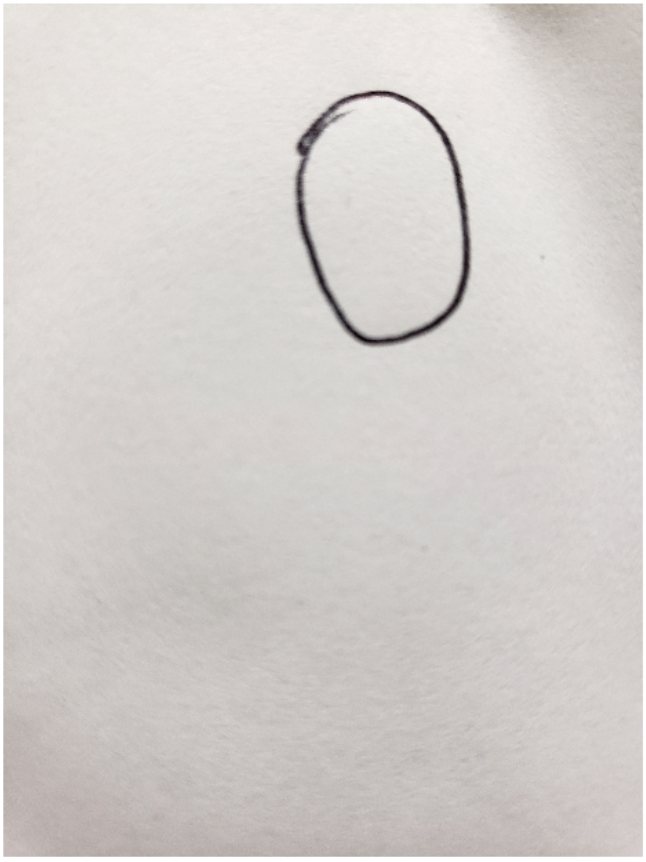
    convolve, spectrum

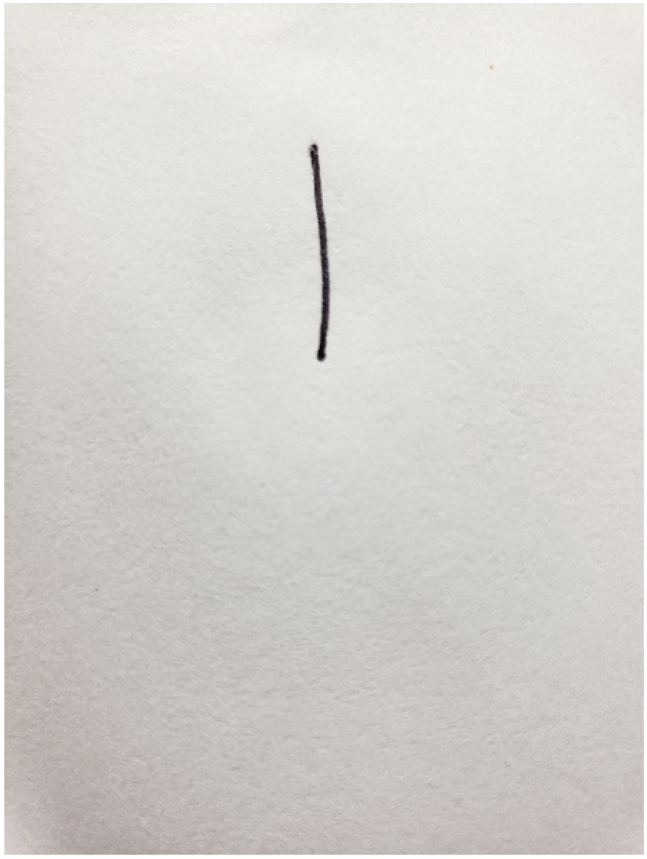
The following object is masked from 'package:graphics':

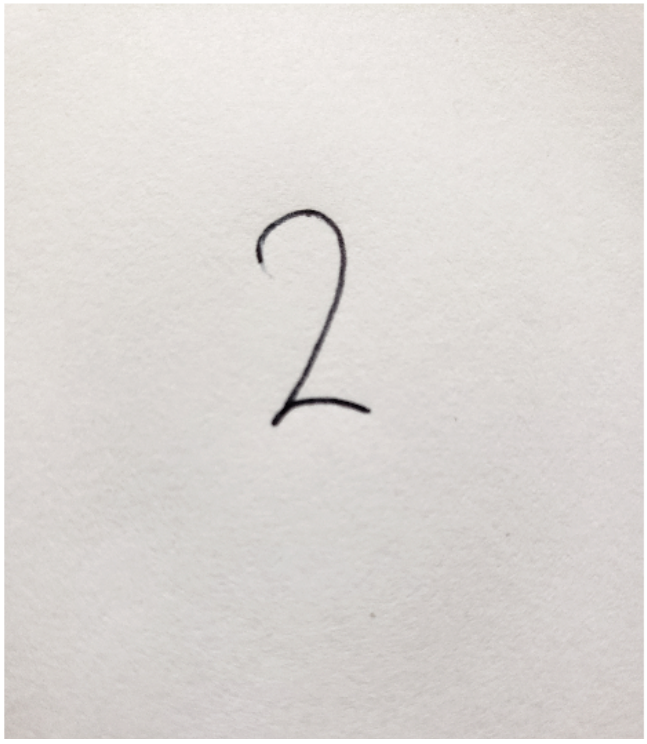
    frame

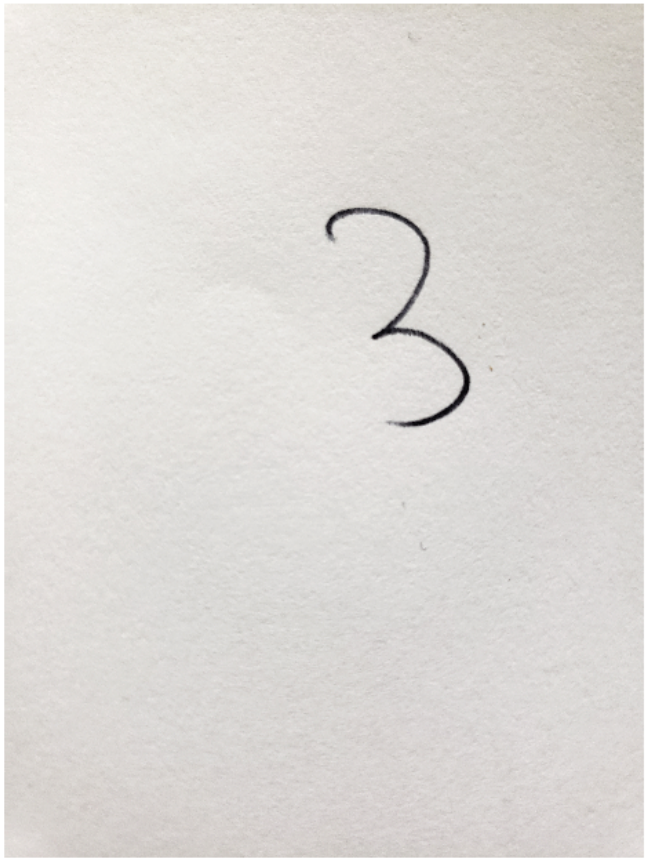
The following object is masked from 'package:base':

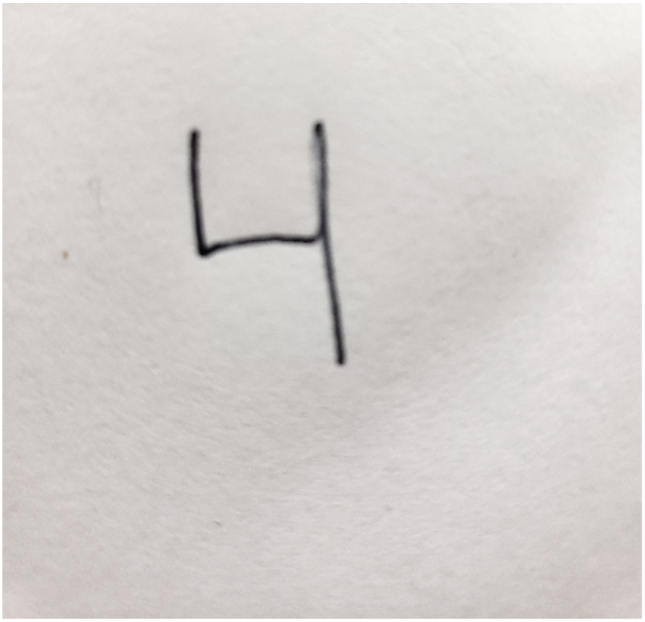
    save.image
```

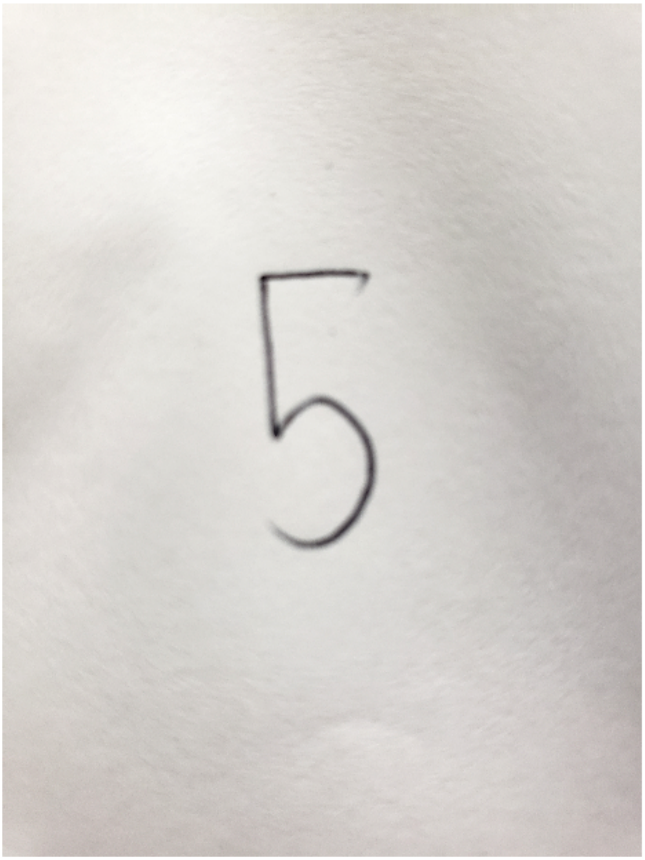


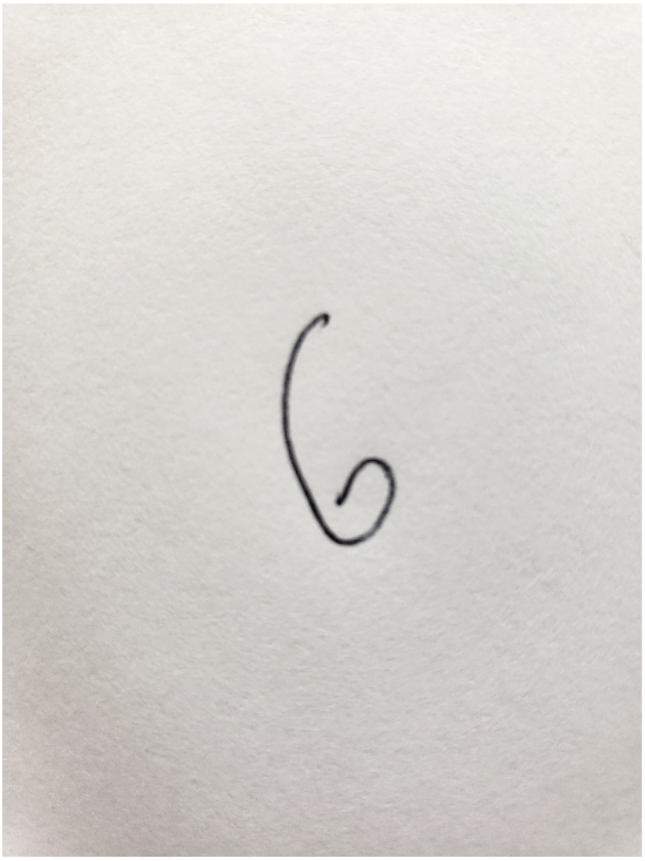

1

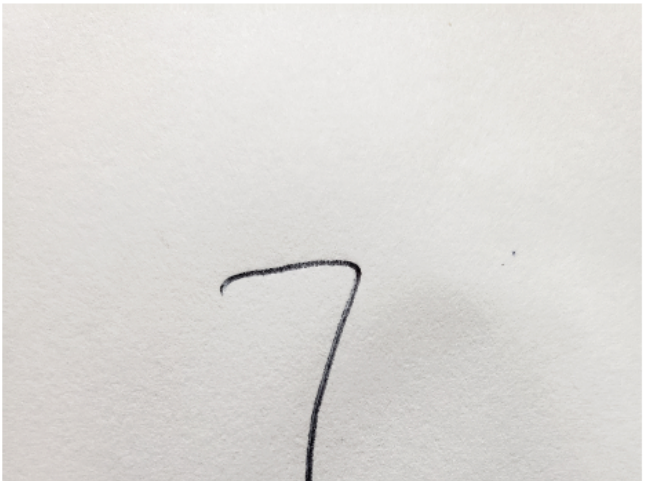

2

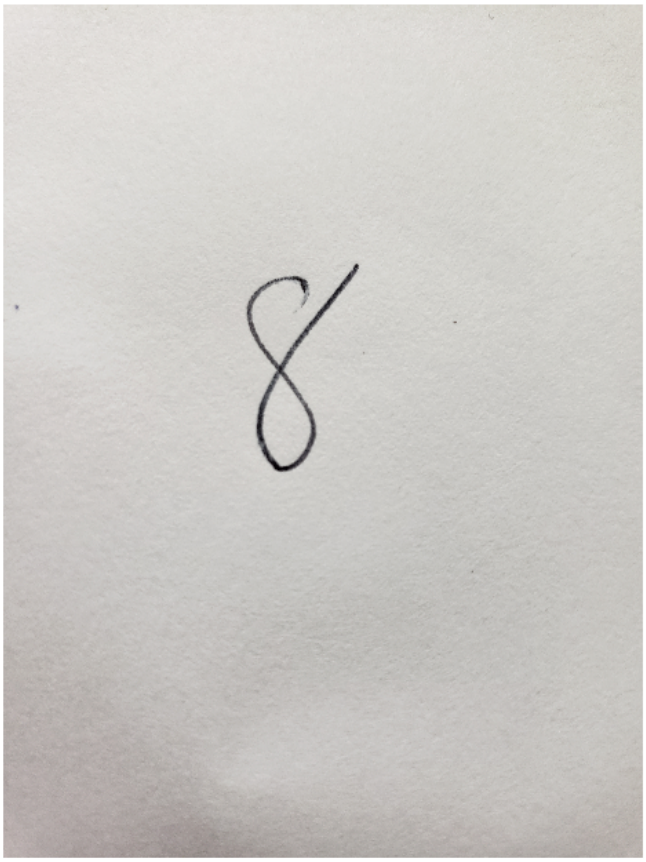

3

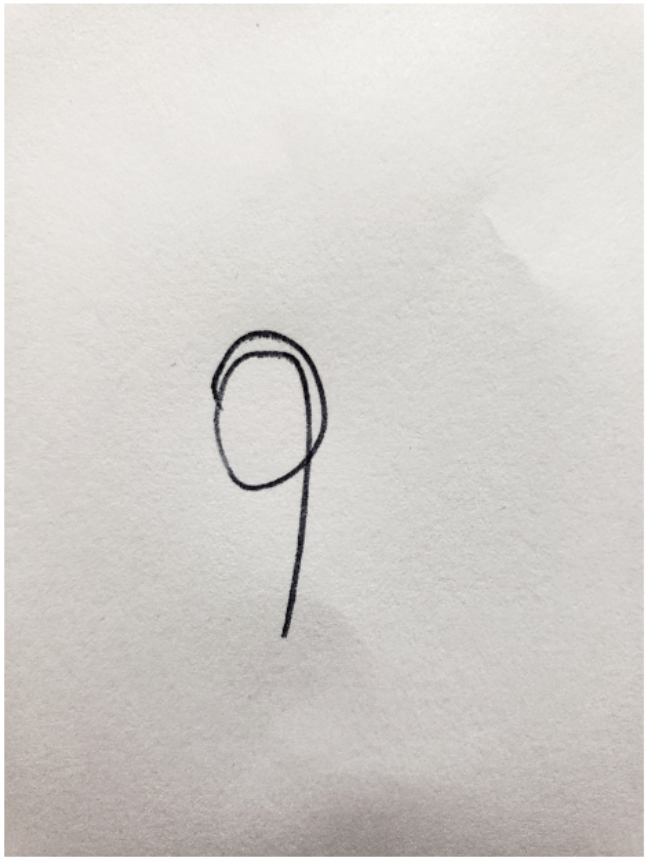












In [ ]:
```
library(png)
im <- readPNG("hand_writing_9.png")
#im <- load.image("hand_writing_9.png")
#im <- load.image(file)
#im.r <- as.raster(im, interpolate=F)
im
#im.r
#matrix(im.r,nrow = 4032, ncol=3024)
#mage(matrix(im.r, nrow = 4032, ncol=3024), axes =FALSE, col =grey.colors
dim(im)
#dim(im.r)
#plot(im.r)
```

In [33]:
```
install.packages("imager")
```

```
also installing the dependencies 'bmp', 'tiff', 'png', 'jpeg', 'readbitma
p', 'downloader', 'igraph'
```

In [ ]:

```
```

In [60]:

```
# midterm part 2
col_data  <- read.csv("Mnemiopsis_col_data.csv")
count_data <- read.csv("Mnemiopsis_count_data.csv")
info_gene <- read.delim("info_gene.txt")

# here we are importing all the data we need and saving it to different v

col_data
count_data
info_gene
```

aboral-1	Mleidyi	aboral
aboral-2	Mleidyi	aboral
aboral-3	Mleidyi	aboral
aboral-4	Mleidyi	aboral
oral-1	Mleidyi	oral
oral-2	Mleidyi	oral
oral-3	Mleidyi	oral
oral-4	Mleidyi	oral

ML000110a	69	175	141	139	108	146	133	63
ML000111a	0	0	0	0	0	1	0	0
ML000112a	1	10	8	3	2	13	6	1
ML000113a	383	546	402	471	290	190	282	317
ML000114a	188	214	257	230	289	215	162	128
ML000115a	493	455	540	501	413	403	419	452
ML000116a	404	462	464	362	516	336	285	336
ML000117a	266	361	301	273	396	277	239	277
ML000118a	177	158	162	153	164	131	107	136

ML000119a	382	339	362	295	254	310	259	308
ML00011a	37	26	33	29	24	46	34	26
ML000120a	227	225	250	141	333	241	130	169
ML000121a	385	294	398	213	385	351	188	270
ML000122a	352	336	336	283	442	300	245	276
ML000123a	1353	1232	1534	1162	1919	1272	976	1130
ML000124a	882	1694	1025	1001	979	834	655	849
ML000125a	5	0	1	0	30	16	9	12
ML000126a	295	306	244	231	237	208	214	253
ML000127a	490	810	854	923	451	619	708	412
ML000128a	0	0	0	0	0	0	0	0
ML000129a	397	207	133	124	373	240	284	367
ML00012a	240	598	418	37	264	468	625	259
ML000130a	32	56	22	42	49	40	34	67
ML000131a	26	62	66	58	69	23	56	50
ML000132a	308	850	686	840	2946	4839	4990	5979
ML00013a	89	154	127	63	80	45	63	103
ML00014a	0	0	0	0	0	0	0	0
ML00015a	1	0	2	0	1	1	4	3
ML00016a	469	619	573	505	1542	775	1094	917
ML00017a	52713	57824	59132	60254	59242	47001	48346	47841
...	...	...	...	...	...	...	...	...
ML49653a	823	798	577	619	431	472	414	693
ML49654a	173	199	162	111	223	167	96	182
ML49655a	143	174	157	165	178	101	120	135
ML49656a	516	495	494	407	641	479	360	416
ML49657a	1291	1700	1600	1515	1568	1178	1016	1137
ML49658a	889	1269	864	1014	543	571	641	632
ML49659a	3405	3372	3581	3442	3795	2948	2290	2776
ML49751a	0	4	2	0	1	0	0	0
ML49851a	64	57	40	68	51	54	67	50
ML50011a	591	991	1143	1848	4447	3881	7955	7989
ML50012a	43	91	86	384	58	24	238	30
ML50013a	7	2	8	7	44	36	51	44
ML50014a	10	50	36	60	1	1	0	0
ML50015a	0	0	1	8	7	4	10	5
ML50181a	5	670	1622	774	1217	546	506	11

ML50361a	28	38	12	25	5	38	30	22
ML50411a	0	0	0	0	0	0	0	0
ML50511a	246	206	170	161	329	220	237	275
ML50512a	17	12	8	7	16	18	20	21
ML50513a	353	406	419	311	292	259	309	305
ML50514a	0	0	0	0	0	0	0	0
ML50515a	667	824	768	621	822	593	672	788
ML50516a	337	393	435	433	359	387	376	380
ML50531a	0	0	0	0	0	0	0	0
ML50541a	26	274	86	97	357	45	122	23
ML50721a	8	9	1	9	9	12	24	28
ML50771a	10	21	3	10	12	10	8	13
ML50791a	0	0	0	0	0	0	0	0
ML50792a	2	2	3	1	3	0	4	0
ML50851a	7	3	3	1	5	1	1	9

https://metazoa.ensembl.org/Mnemiopsis_leidyi/Info/Index

Mnemiopsis leidyi, the warty comb jelly or sea walnut, is a ctenophore, originally native to western Atlantic coasts but now found throughout Europe and Asia, in waters of varying salinity and temperature. M. leidyi has a lobed body and four rows of cilia by which it moves. It can grow up to 12 cm long and 2.5 cm wide, making them the largest animals that move by means of cilia. In the absence of predators (e.g. Beroe ovata, another ctenophore) this comb jelly can drastically deplete fish populations, because it feeds on fish eggs, larvae and plankton.

The phylogenetic position of M. leidyi has been long debated, with recent studies positioning ctenophores as the sister group to porifera, placozoa, cnidaria and bilateria. The study of key biological processes, like regeneration and axial patterning, and its phylogenetic importance make the comb jelly an important model for evolutionary and developmental studies.

ftp://ftp.ensemblgenomes.org/pub/metazoa/release-46/gff3/mnemiopsis_leidyi/Mnemiopsis_leidyi.MneLei_Aug2011.46.gff3.gz

Gene annotations

Mnemiopsis_leidyi.MneLei_Aug2011.46.gff3.gz

Get information on gene function here

https://research.nhgri.nih.gov/mnemiopsis/genes/genewiki.cgi

In [61]:

```
# midterm part 2 continued
# use this cell also
# this cell is to get the data into a format that is useful to us

aboral_genes <- count_data[,-c(6:9)]
# this is to remove the oral data and only keep the aboral data with the
# gene name and save it to a new variable called aboral_genes

oral_genes <- count_data[,-c(2:5)]
# this is to remove the aboral data and keep the oral data with the
```

```
# this is to remove the aboral data and keep the oral data with the
# gene names and save it to a new variable called oral_genes

aboral_genes
oral_genes
```

ML000110a	69	175	141	139
ML000111a	0	0	0	0
ML000112a	1	10	8	3
ML000113a	383	546	402	471
ML000114a	188	214	257	230
ML000115a	493	455	540	501
ML000116a	404	462	464	362
ML000117a	266	361	301	273
ML000118a	177	158	162	153
ML000119a	382	339	362	295
ML00011a	37	26	33	29
ML000120a	227	225	250	141
ML000121a	385	294	398	213
ML000122a	352	336	336	283
ML000123a	1353	1232	1534	1162
ML000124a	882	1694	1025	1001
ML000125a	5	0	1	0
ML000126a	295	306	244	231
ML000127a	490	810	854	923
ML000128a	0	0	0	0
ML000129a	397	207	133	124
ML00012a	240	598	418	37
ML000130a	32	56	22	42
ML000131a	26	62	66	58
ML000132a	308	850	686	840
ML00013a	89	154	127	63
ML00014a	0	0	0	0
ML00015a	1	0	2	0
ML00016a	469	619	573	505
ML00017a	52713	57824	59132	60254
...	...	...	...	...
ML49653a	823	798	577	619

ML49654a	173	199	162	111
ML49655a	143	174	157	165
ML49656a	516	495	494	407
ML49657a	1291	1700	1600	1515
ML49658a	889	1269	864	1014
ML49659a	3405	3372	3581	3442
ML49751a	0	4	2	0
ML49851a	64	57	40	68
ML50011a	591	991	1143	1848
ML50012a	43	91	86	384
ML50013a	7	2	8	7
ML50014a	10	50	36	60
ML50015a	0	0	1	8
ML50181a	5	670	1622	774
ML50361a	28	38	12	25
ML50411a	0	0	0	0
ML50511a	246	206	170	161
ML50512a	17	12	8	7
ML50513a	353	406	419	311
ML50514a	0	0	0	0
ML50515a	667	824	768	621
ML50516a	337	393	435	433
ML50531a	0	0	0	0
ML50541a	26	274	86	97
ML50721a	8	9	1	9
ML50771a	10	21	3	10
ML50791a	0	0	0	0
ML50792a	2	2	3	1
ML50851a	7	3	3	1

ML000110a	108	146	133	63
ML000111a	0	1	0	0
ML000112a	2	13	6	1
ML000113a	290	190	282	317
ML000114a	289	215	162	128
ML000115a	413	403	419	452

ML000116a	516	336	285	336
ML000117a	396	277	239	277
ML000118a	164	131	107	136
ML000119a	254	310	259	308
ML00011a	24	46	34	26
ML000120a	333	241	130	169
ML000121a	385	351	188	270
ML000122a	442	300	245	276
ML000123a	1919	1272	976	1130
ML000124a	979	834	655	849
ML000125a	30	16	9	12
ML000126a	237	208	214	253
ML000127a	451	619	708	412
ML000128a	0	0	0	0
ML000129a	373	240	284	367
ML00012a	264	468	625	259
ML000130a	49	40	34	67
ML000131a	69	23	56	50
ML000132a	2946	4839	4990	5979
ML00013a	80	45	63	103
ML00014a	0	0	0	0
ML00015a	1	1	4	3
ML00016a	1542	775	1094	917
ML00017a	59242	47001	48346	47841
...	...	...	...	...
ML49653a	431	472	414	693
ML49654a	223	167	96	182
ML49655a	178	101	120	135
ML49656a	641	479	360	416
ML49657a	1568	1178	1016	1137
ML49658a	543	571	641	632
ML49659a	3795	2948	2290	2776
ML49751a	1	0	0	0
ML49851a	51	54	67	50
ML50011a	4447	3881	7955	7989
ML50012a	58	24	238	30

ML50013a	44	36	51	44
ML50014a	1	1	0	0
ML50015a	7	4	10	5
ML50181a	1217	546	506	11
ML50361a	5	38	30	22
ML50411a	0	0	0	0
ML50511a	329	220	237	275
ML50512a	16	18	20	21
ML50513a	292	259	309	305
ML50514a	0	0	0	0
ML50515a	822	593	672	788
ML50516a	359	387	376	380
ML50531a	0	0	0	0
ML50541a	357	45	122	23
ML50721a	9	12	24	28
ML50771a	12	10	8	13
ML50791a	0	0	0	0
ML50792a	3	0	4	0
ML50851a	5	1	1	9

In [62]:

```
# number 1
# making the function

library(reshape2)
get_top_5_genes  <- function (data){
# we want to take the average then put it into a new table with the gene

num_rows_ab <- c(1:length(data[,1]))
# making a list of numbers for the for loop to go through

avg <- c()
# making an empty list to store the averages


for (x in num_rows_ab) {
    # this for loop goes through the list of numbers that starts from 1 t
    # the number of rows in the aboral_genes df
    avg <- append(avg, mean(as.numeric(data[x,c(2:5)])))
    # next we append the mean of each row to a list called ab_avg

    #if (x ==1){break}

}
avg_df <- data.frame(avg)
#avg_df
# above we wanted to take the list of appended averages then make it into
# data frame called avg_df
```

```
data_with_avg <- data.frame(data, avg_df)
#new <- melt(data_with_avg)
#aboral_with_avg
# above we wanted to combine the aboral data with the new average data we
# into one data frame called aboral_with_avg


#which(avg_df ==max(avg_df))
#max(avg_df)
#avg_df
#aboral_with_avg[12714,]
# just something we did to find the max, which is not needed rn


#sort(aboral_with_avg, decreasing = TRUE)

top_avg_ab <- data_with_avg[order(-data_with_avg$avg),]
# we took the dataframe and used the order method to only sort the data b
# on the aboral average column in decending order

head(top_avg_ab,5)
# next we used the head method to get the top 5 rows which include the
# top 5 genes that has the highest average expression in the aboral data
}

aboral_top_5 <- get_top_5_genes(aboral_genes)
oral_top_5  <- get_top_5_genes(oral_genes)

aboral_top_5
oral_top_5
#Mleidyi

# ML20395a in the top_avg_ab genes containing aboral data, the gene ML203
# a function in chain elongation during polypeptide synthesis at the ribo

# ML46651a gene is the 2nd highest average in the aboral genes, and its
# function is a protein complex porduced by activated components of the
# complement cascade inserted into a target cell membrane and forming a p
# leading to cell lysis via ion and water flow

# ML01482a for this gene which is the 3rd highest average in aboral,
# there are three biological processes annotated. it function is microtub
# based process, microtubule-based movement, and protein polymerization

# ML26358a is the 4th gene in the highest aboral average, and has 4 funct
# annotated for it. the functions are cytoskeleton organization, actin
# cytoskeleton organization, cytokinesis, and embryo development ending i
# birth or egg hatching

# ML034334a is the 5th gene in the highest aboral average, and has 4 func
# the functions annotated are, negative regulation of biological process,
# regulation of apoptotic process, organ development, cell proliferation


# now for the top_avg_ab genes containing the oral data

# ML20395a is the 1st gene in the oral, and has 3 functions, translation
# elongation factor activity, GTP binding, GTPase activity

# ML26358a is the 2nd gene, has 2 functions, protein binding, and ATP bi
```

```
# ML020045a is the 3rd gene, has 4 functions, GTP binding, protein bindi
# GTPase activity, structural constituent of cytoskeletion

# ML00017a is the 4th gene, has 3 functions, translation elongation facto
# activity, GTPase activity, GTP binding

# ML04011a is the 5th gene, has 5 functions, nitric-oxide synthease
# regulator activity, ATP binding, TPR domain binding, unfolded protein b
# protein homodimerization acitivity
```

12714	ML20395a	122707	131017	136282	111388	125348.50
16420	ML46651a	125638	105808	65907	93351	97676.00
1908	ML01482a	32503	90804	83222	111860	79597.25
14235	ML26358a	61229	93272	78693	78310	77876.00
3788	ML034334a	23288	76895	65076	94170	64857.25

12714	ML20395a	163380	101792	101421	109944	119134.25
14235	ML26358a	62893	46232	49534	47733	51598.00
2612	ML020045a	65580	54406	35861	48147	50998.50
30	ML00017a	59242	47001	48346	47841	50607.50
4249	ML04011a	64225	50041	44420	41929	50153.75

In [ ]:
```
# note to self. make all the code for getting the top 5 genes into
# a function then just pass the aboral_genes then oral_genes variables
# into it. so i dont have to write the code all over again
#DONE
```

In [65]:
```
# number 3a


k <- c(2:(length(aboral_top_5)))

average  <- c("average")
stand_devi <- c("stand_devi")

col_1_mean  <- mean(aboral_top_5[,2])
col_1_mean
for (x in k){
    scale <- col_1_mean/as.numeric(mean(aboral_top_5[,x]))
    # the scale we used is the mean of column 1 divided by the mean of th
    # current column in the loop, column x.

    scaled_df <- (as.numeric(aboral_top_5[,x]))*scale
    # the means of each column were not the same so we decided to use thi
    # to scale each column in aboral data by multiplying each entry by sc
    # this doesnt reflect in each entry, it only reflects in the average
    # the next cell shows each entry scaled
```

```
    average <- append(average,as.numeric(mean(scaled_df)))
    stand_devi <- append(stand_devi,as.numeric(sd(aboral_top_5[,x])))
}

row <- data.frame(average)
row
stand_devi
new_df <- aboral_top_5
#aboral_top_5[nrow(aboral_top_5)+1,]  <- average
#aboral_top_5[nrow(aboral_top_5)+1,]  <- stand_devi
#aboral_top_5

new_df[nrow(new_df)+1,]  <- average
new_df[nrow(new_df)+1,]  <- stand_devi
new_df
# this new_df contains the aboral data with the average and standard devi
# for each column as new rows respectively.
scaled_df
```

average
73073
73073
73073
73073
73073

```
Warning message in `[<-.factor`(`*tmp*`, iseq, value = "average"):
"invalid factor level, NA generated"Warning message in `[<-.factor`(`*tmp
*`, iseq, value = "stand_devi"):
"invalid factor level, NA generated"
```

12714	ML20395a	122707	131017	136282
16420	ML46651a	125638	105808	65907
1908	ML01482a	32503	90804	83222
14235	ML26358a	61229	93272	78693
3788	ML034334a	23288	76895	65076
6	NA	73073	73073	73073

```
7          NA  48711.8174625008  20362.2755039804  29286.5028724838  14099.1728
```

In [191…

```
# just a quick cheak to make sure the normalized numbers are correct
131017*0.733
105808*0.733
90804*0.733
93272*0.733
76895*0.733

(96035.461+77557.264+66559.332+68368.376+56364.035)/5
```

In [67]:

```
# number 3a continued



# this is to normalize the data while reflecting that in each column enti
# not just doing it behind the scene and only keeping the average that is
# normalized
# only for the aboral data so far. later we can go back and make this a f
# then pass both aboral and oral data through to get normailzation for ev
col_1_mean  <- mean(aboral_top_5[,2])
#scale <- col_1_mean/as.numeric(mean(aboral_top_5[,x]))

h  <- c(2:length(aboral_top_5)-1)
for (x in h){
    scale <- col_1_mean/as.numeric(mean(aboral_top_5[,x]))
    assign(paste0("aboral",x-1) ,  (as.numeric(aboral_top_5[,x]))*scale)
}
# apply(paste0("aboral",x-1) ,  (as.numeric(aboral_top_5[,x]))*0.733)
s1 <- data.frame(aboral1)
s2 <- data.frame(aboral2)
s3 <- data.frame(aboral3)
s4 <- data.frame(aboral4)

#new_aboral_norm <- data.frame(gene_names=aboral_top_5[,1],s1,s2,s3,s4)
new_aboral_norm <- data.frame(s1,s2,s3,s4)
rownames(new_aboral_norm)  <- aboral_top_5[,1]
new_aboral_norm

cor_columns <- cor(new_aboral_norm)
cor_columns
# this is giving the correlation between the columns of the normalized
```

```
# this is giving the correlation between the columns of the normalized
# top 5 genes in aboral (or later both aboral and oral if we make a funct

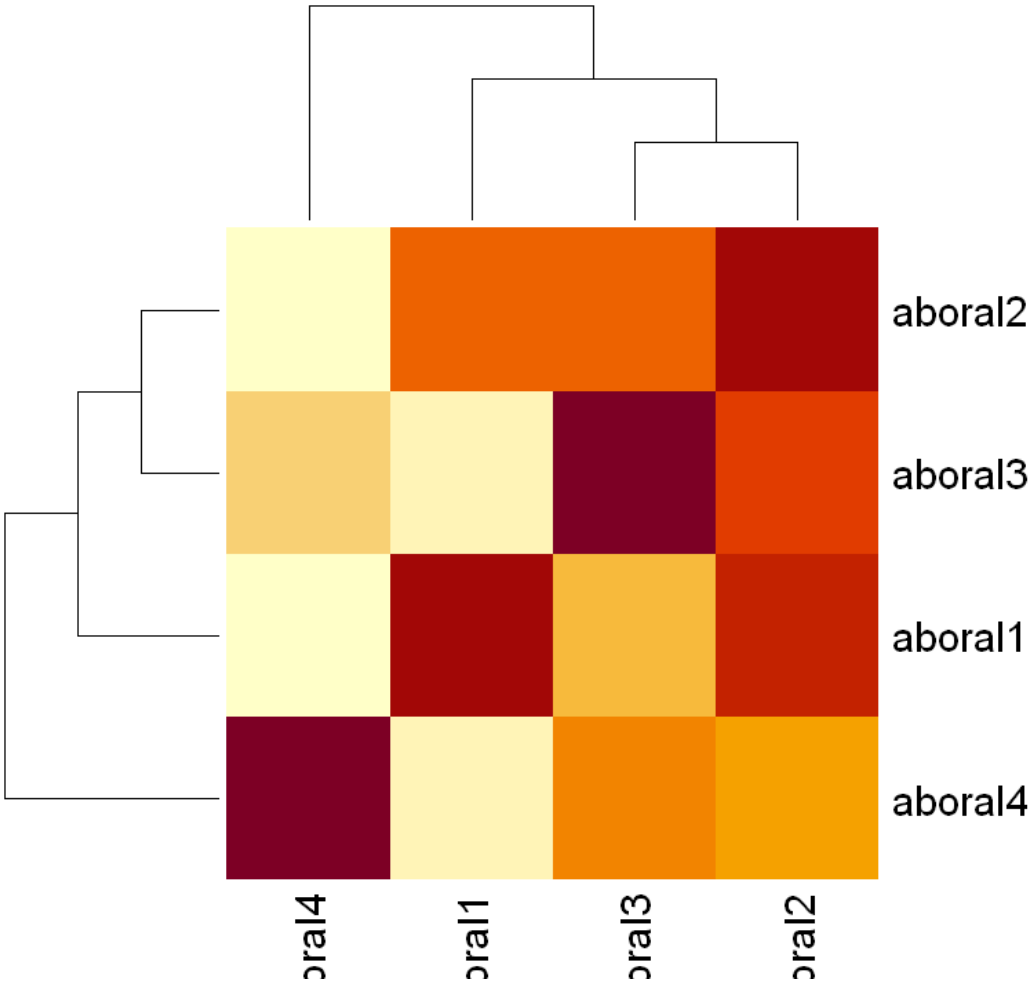
heatmap(cor_columns, margins= c(10,10))

# we think the correlation and the heatmap data both show it is concordar
# the column labels. For example, we expected the aboral4 to have the lea
# correlation with aboral1
```

```
Warning message in mean.default(aboral_top_5[, x]):
"argument is not numeric or logical: returning NA"
```

ML20395a	122707	96161.93	116018.16	83212.07
ML46651a	125638	77659.40	56107.25	69737.58
ML01482a	32503	66646.99	70847.68	83564.68
ML26358a	61229	68458.41	66992.10	58501.25
ML034334a	23288	56438.26	55399.82	70349.42

aboral1	1.0000000	0.8690149	0.4697286	0.1025450
aboral2	0.8690149	1.0000000	0.8387383	0.4191683
aboral3	0.4697286	0.8387383	1.0000000	0.5763573
aboral4	0.1025450	0.4191683	0.5763573	1.0000000



abc abc abc abc

In [69]:

```
# number 4 part 2
cor_matrix <- cor(t(new_aboral_norm))
# we am using the new_aboral_norm data here, which are the top 5 genes wh
# have been normalized

# above we wanted to make a correlation matrix between the top 5 aboral g
# since we are looking at the correlation between the rows we used t to
# transpose the matrix before using the cor function

cor_matrix
heatmap(cor_matrix, margins= c(10,10))
# then made a heat map to visualize that matrix
# the genes that have the best correlations (high) are ML034334a / ML0148
# which have a correlation of 0.994. this shows they are very similar but
# not the exact same which would give us back a 1.0



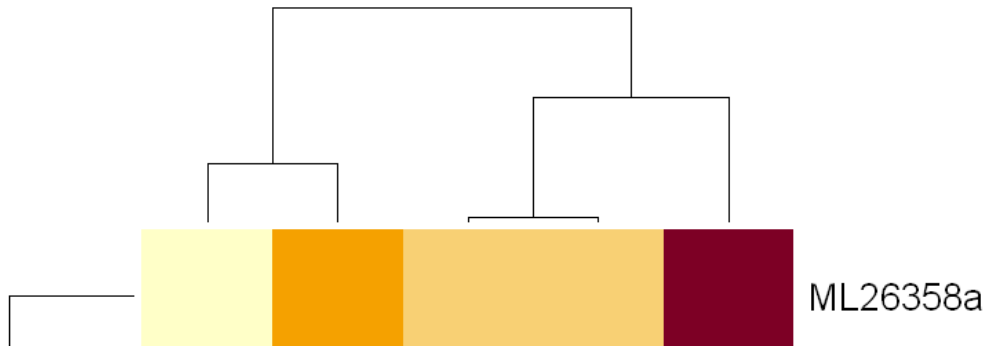
# the other two genes that have a good correlation (high) are ML46651a/ M
# these would be considered the closest pairs between the top 5 genes fro
# the previous part. they are close because they dont vary much.

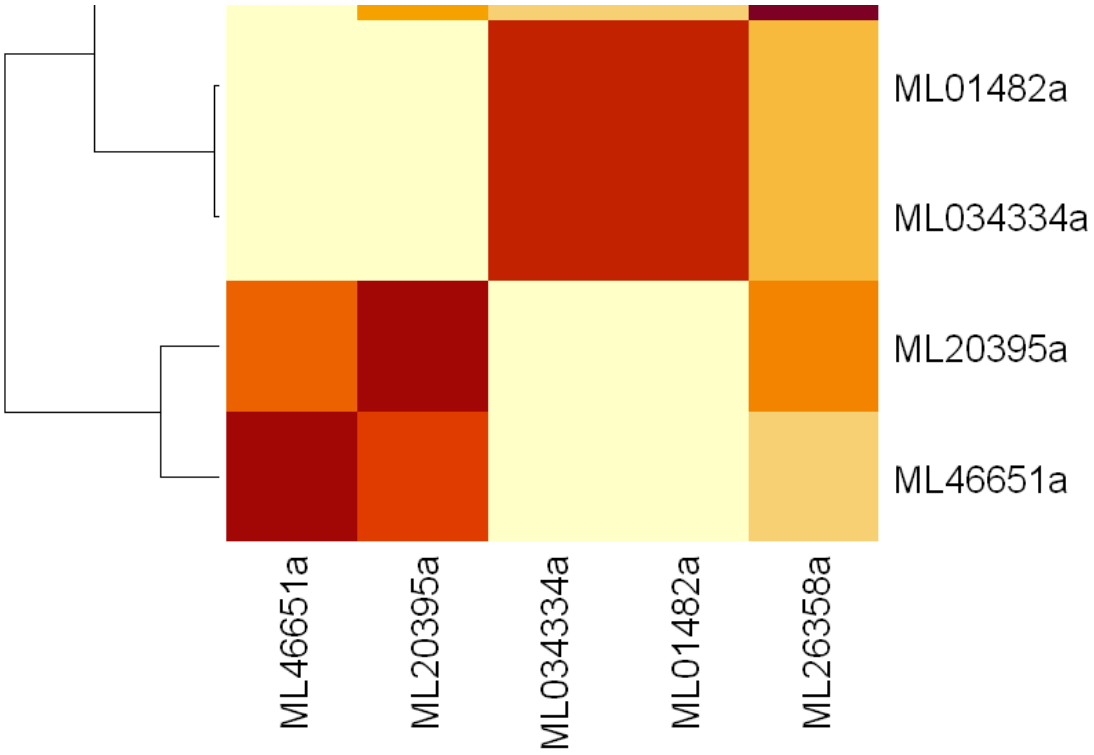

# genes that have a medium correlation are the ones in light orange on th

# we would say looking on the heat map we see a large square thats darker
# color from the other 2 large squares which represents the two genes tha
# are better correlated yet aren't the exact same.

# the lighter in color, large squares represent the genes that are most
# dissimilar, which would be the most negative numbers in the dataframe c
```

ML20395a	1.0000000	0.4822272	-0.7862276	0.24935052	-0.84054332
ML46651a	0.4822272	1.0000000	-0.9108768	-0.35116798	-0.87665364
ML01482a	-0.7862276	-0.9108768	1.0000000	0.03726750	0.99477505
ML26358a	0.2493505	-0.3511680	0.0372675	1.00000000	0.02878694
ML034334a	-0.8405433	-0.8766536	0.9947751	0.02878694	1.00000000





In [ ]:

```
# number 5 part2

# if we wanted to divide the genes into high medium and low count genes w
# look at the heatmap and get the genes that have a darker color and make
# the high count genes. then we would get the genes that have the light c
# color as the medium, lastly the genes that have a color yellow and ligh
# yellow would be the genes that are low count genes.
```

In [95]:

```
# number 6 part 2 and number 7

# this function is a repuropsed function that gets the variances instead
# the mean like in number 3.

library(reshape2)
get_top_5_genes  <- function (data){
# we want to take the variances then put it into a new table with the ger

num_rows_ab <- c(1:length(data[,1]))
# making a list of numbers for the for loop to go through

vari <- c()
# making an empty list to store the variances


for (x in num_rows_ab) {
    # this for loop goes through the list of numbers that starts from 1 t
    # the number of rows in the aboral_genes df (data df)
    vari <- append(vari, var(as.numeric(data[x,c(2:5)])))
    # next we append the variances of each row to a list called vari

    #if (x ==1){break}

}
```

```
}
vari_df <- data.frame(vari)
#avg_df
# above we wanted to take the list of appended variances then make it int
# data frame called vari_df

data_with_vari <- data.frame(data, vari_df)
#new <- melt(data_with_vari)
#aboral_with_vari
# above we wanted to combine the aboral data with the new variance data w
# into one data frame called data_with_vari

list_to_remove <- c()
for (x in num_rows_ab) {
    # this for loop checks to see if the mean of each column is
    # less than 5000 expression value. if so it will append the row numbe
    # to the variable called list_to_remove

    if (mean(as.numeric(data_with_vari[x,c(2:5)]) < 5000)){
        list_to_remove <- append(list_to_remove, x)
    }
}

new_vari_data <- data_with_vari[-list_to_remove,]
# this will remove the list of rows that had a mean variance less than 50

#sort(data_with_vari, decreasing = TRUE)

top_vari_ab <- new_vari_data[order(-new_vari_data$vari),]
# we took the dataframe and used the order method to only sort the data b
# on the aboral(data) variance column in decending order


top_5_vari <- ( head(top_vari_ab,5))
print(top_5_vari)
# next we used the head method to get the top 5 rows which include the
# top 5 genes that has the highest variance expression in the aboral (dat

bottom_5_vari <- ( tail(top_vari_ab,5))
print(bottom_5_vari)
# this gives back the 5 genes with the least variability which would be t
# bottom 5 genes in the dataframe that was ordered in decending order.

cat(" the top 5 p-values for the above data
are \n")
for (x in c(1:length(top_5_vari[1,c(2:5)]))){
    print(t.test(top_5_vari[x,c(2:5)])$p.value)

}

cat(" the bottom 5 p-values for the above data
are \n")
for (x in c(1:length(bottom_5_vari[1,c(2:5)]))){
    print(t.test(bottom_5_vari[x,c(2:5)])$p.value)

}

}

#aboral_top_5 <- get_top_5_genes(aboral_genes)
#oral top 5  <- get top 5 genes(oral genes)
```

```
#aboral_top_5
#oral_top_5
#Mleidyi
get_top_5_genes(aboral_genes)
get_top_5_genes(oral_genes)
```

In [90]:

```
# number 7 part 2
#?t.test

for (x in c(1:length(aboral_top_5[1,c(2:5)]))){
```

```
for (x in c(1:length(aboral_top_5[1,c(2:5)]))){
    print(t.test(aboral_top_5[x,c(2:5)])$p.value)

}
```

In [48]:

```
# test cell to help keep track of which rows are the ones that are less t
# the value i want so i can eliminate them later
top6_aboral <- head(aboral_genes,6)
top6_aboral
test_l <- c()
for (x in c(1:length(top6_aboral[,1]))){
    #cat(mean(as.numeric(top6_aboral[x,c(2:5)])))
    test_l <- append(test_l,mean(as.numeric(top6_aboral[x,c(2:5)])))
}
test_l
#which(mean(as.numeric(top6_aboral[1,c(2:5)]))== 131)
```

ML000110a	69	175	141	139
ML000111a	0	0	0	0
ML000112a	1	10	8	3
ML000113a	383	546	402	471
ML000114a	188	214	257	230
ML000115a	493	455	540	501

In [53]:

```
# test cell to figure out how to remove a row that has a mean less than
# the target mean. which will help eliminate genes that have low
# expression values.

top6_aboral <- head(aboral_genes,6)
num_rows_ab <- c(1:length(top6_aboral[,1]))

list_to_remove <- c()
for (x in num_rows_ab) {
    #list_to_remove <- append(list_to_remove, which(mean(as.numeric(top6_
    #cat(x, "\n")
    if (mean(as.numeric(top6_aboral[x,c(2:5)]) < 240)){list_to_remove <-
}
list_to_remove
```

```
top6_aboral[-list_to_remove,]
```

4	ML000113a	383	546	402	471
6	ML000115a	493	455	540	501

In [248…

```
?heatmap
```