In [ ]:
```
# NOTE: THIS PART REUSES THE MIDTERM
# CELL 5 IS WHERE THE FINAL PROJECT ANALYSIS BEGINS
```

In [1]:
```
# cell1
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 pixe
head(data,10)
#smoothScatter(pixels)
```

label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	p
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	

In [2]:
```
#cell2
# 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

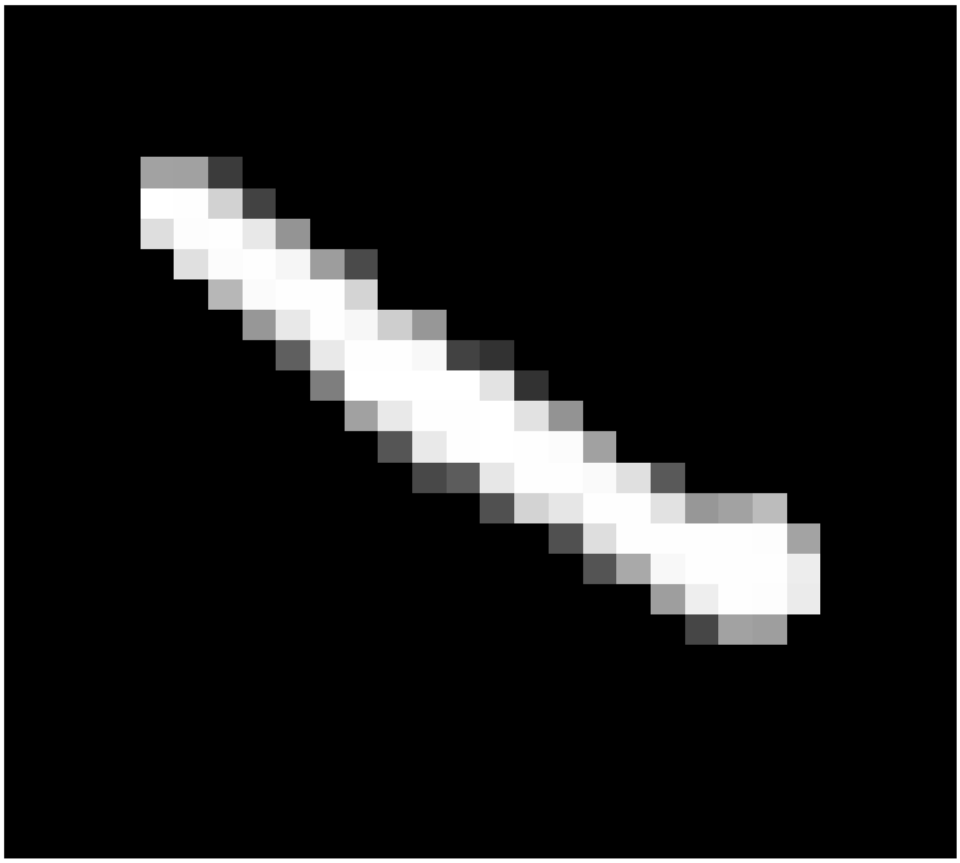
#par(mar= c(5,5,5,5))
image(my_label1, axes =FALSE, col = grey.colors(256, start =0, end=1))
```

```
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")
```



In [3]:

```
#cell3
# 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

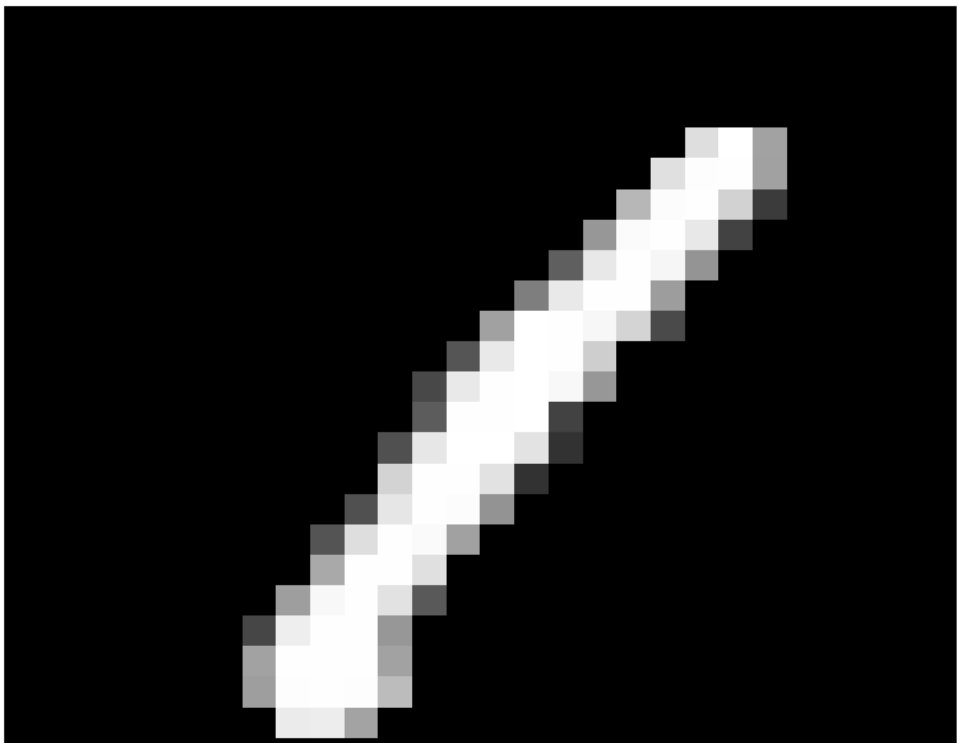
    mode(label1) = "numeric"
    # we need to change the mode of each number in that row to numeric
    # so the image method can work with it

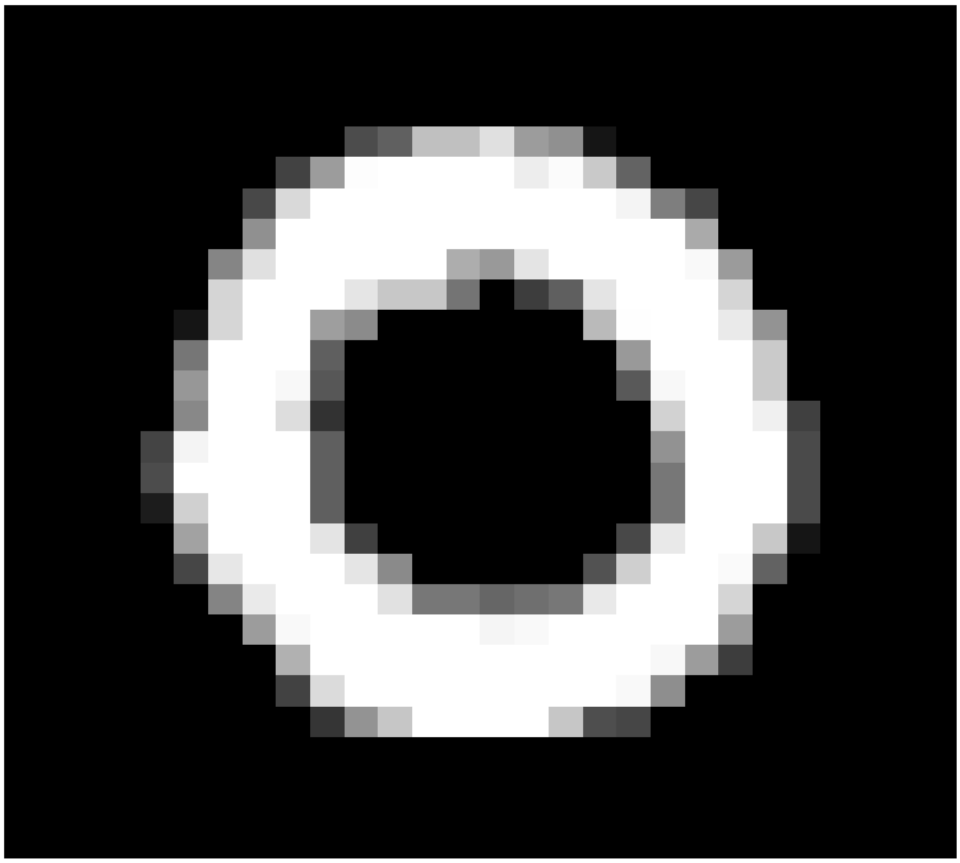
    my_label1 <- matrix(rev(label1), nrow=28, byrow= FALSE)
    # now we make that row into a 28x28 matrix called my_label1

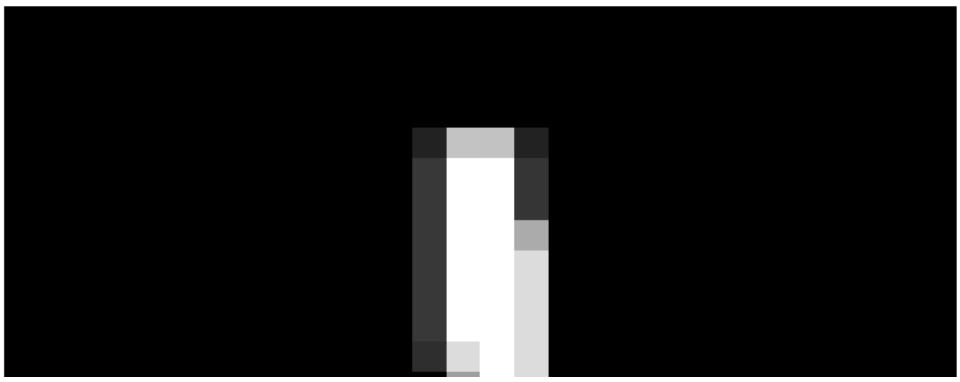

    my_label1 <- apply(my_label1,2,rev)

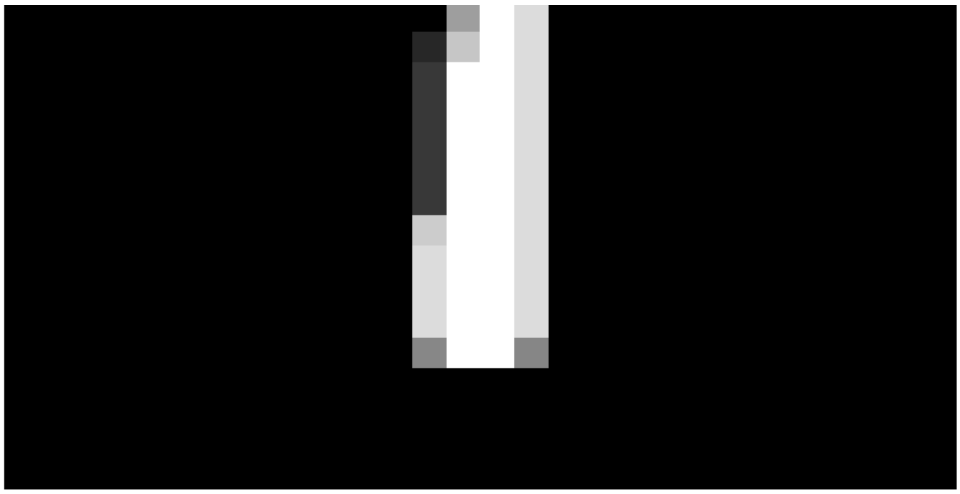
    #par(mar= c(5,5,5,5))
    image(my_label1, axes =FALSE, col = grey.colors(256, start =0, end=1
    # now we use image to plot all the points in the matrix
    # 0 being black and 256 being white, and everything inbetween being
    # shades of grey
    #my_label1
    if (x ==10){ break }
    # since there 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

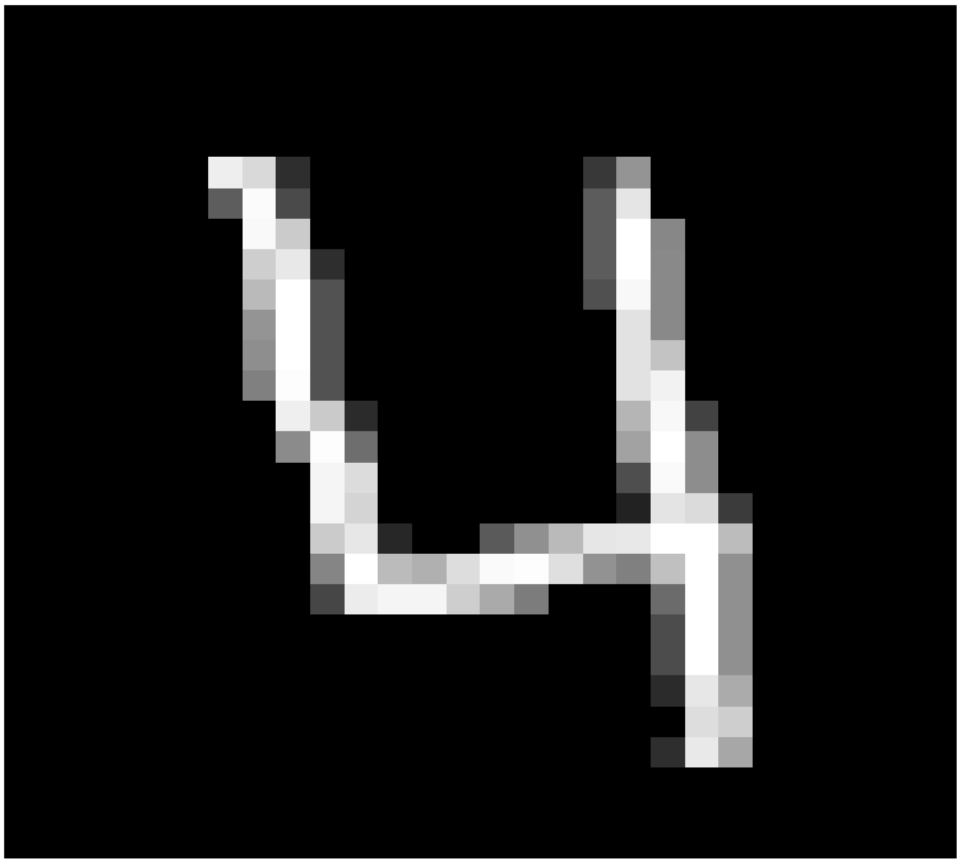
}
```

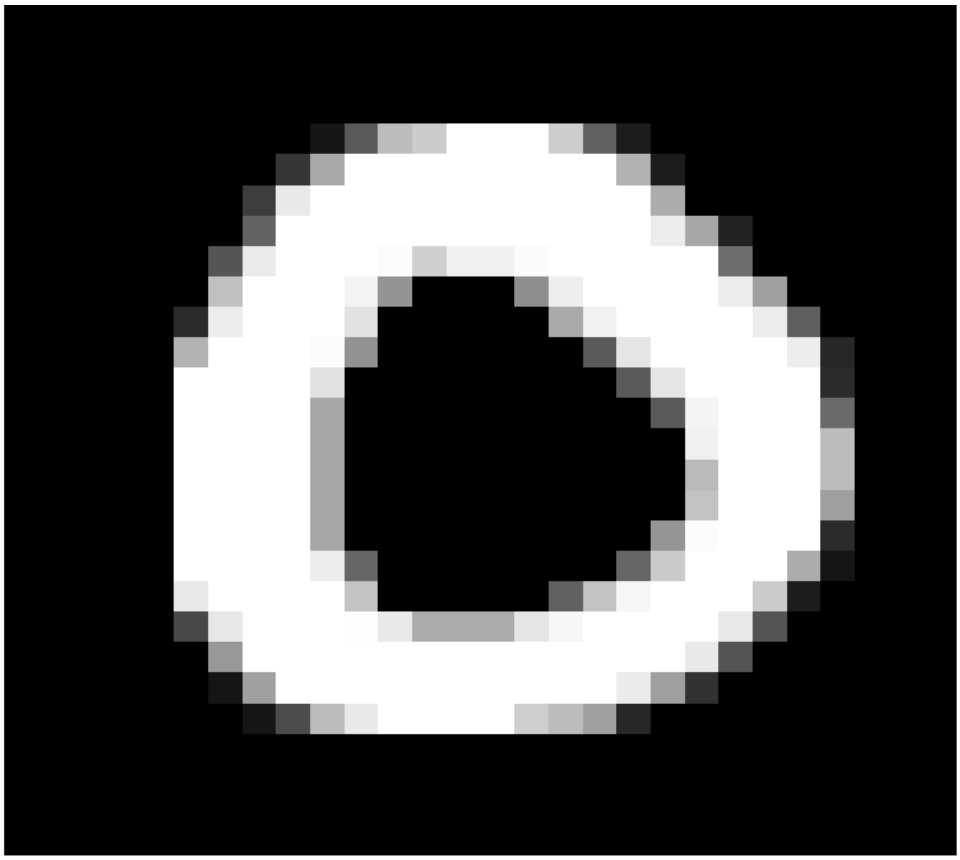


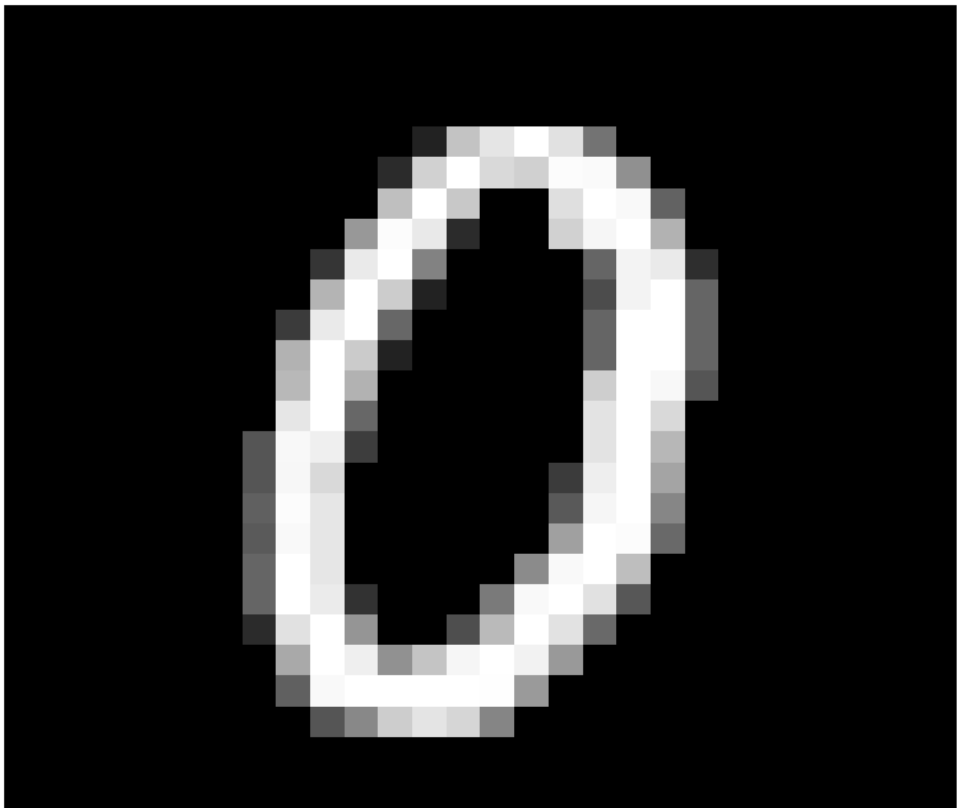


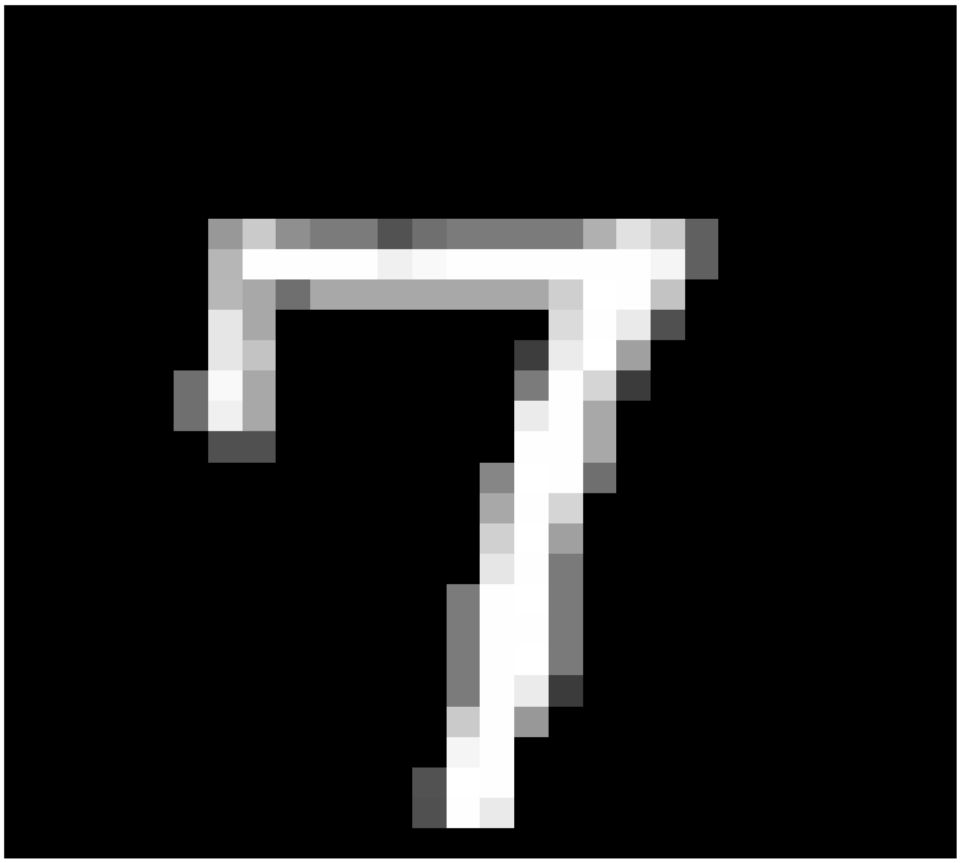


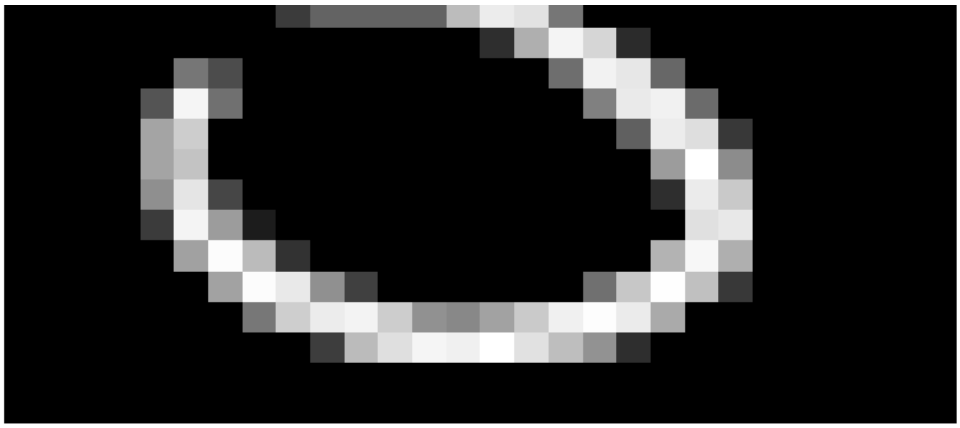


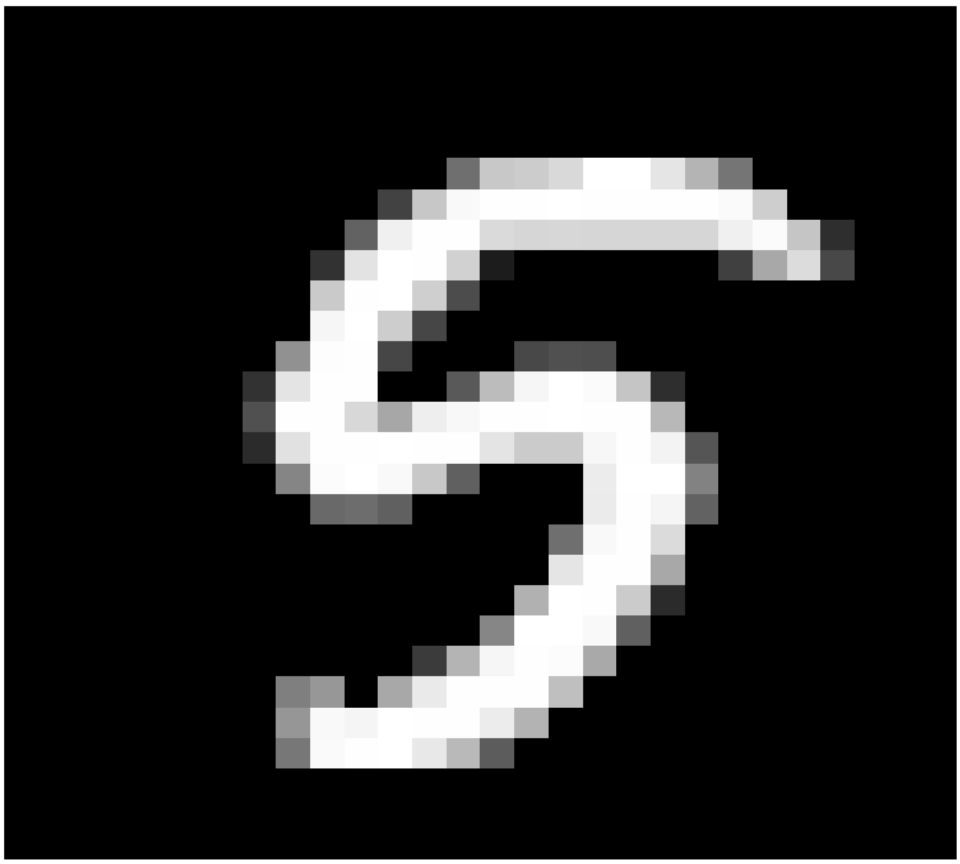


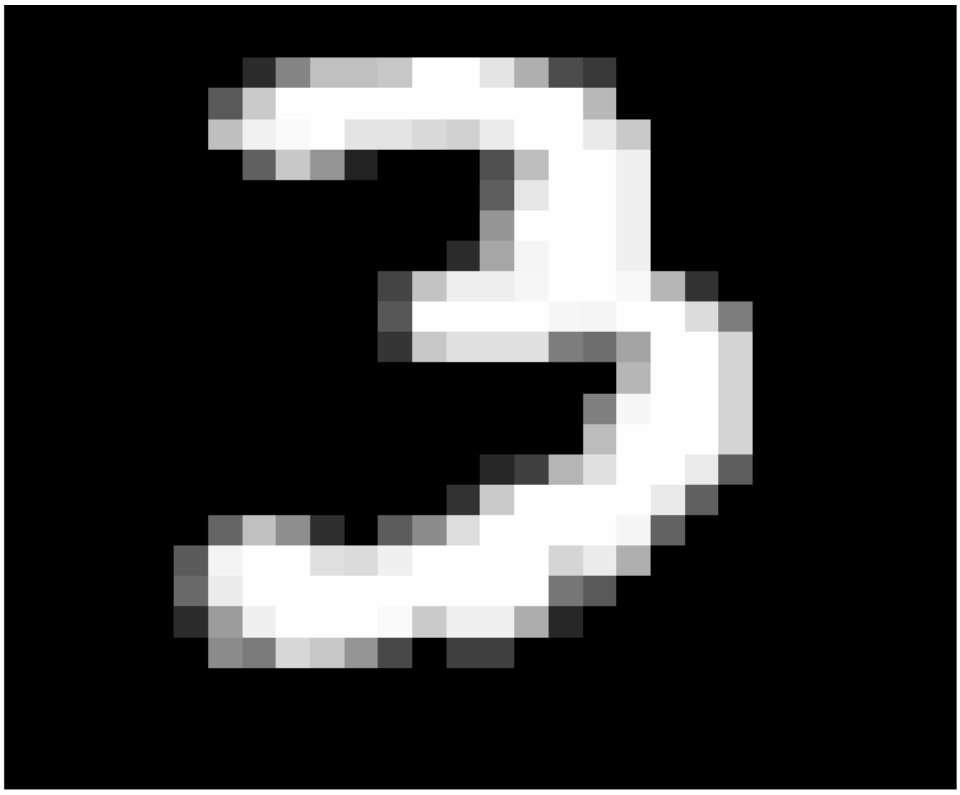












In [4]:

```
#cell4
# 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]),]
```

In [5]:
```
#cell 5

# final project number 1
test <- data.frame(my_label1)
#test
# note: i am using "my_label1" variable from the 3rd cell. this variable
# uses code that gets information from the first 10 rows in the pixel da
# the last pixel data ends up being the digit "3"


# so the digit we are looking to run pca on is 3


list_to_remove <- c()

for (x in c(1:length(test))){

    if (var(test[,x]) == 0) {
        list_to_remove <- append(list_to_remove, x)
    }

}
list_to_remove
# I used this for loop to find all the columns that has a variance of 0
# the column number into the list_to_remove variable. now i will edit th
# matirx test and remove those columns

test_update <- test[,-(list_to_remove)]

test_update
# this is now only the data frame that consists of only columns with vaı
```

1. 1
2. 2
3. 3
4. 4
5. 25
6. 26
7. 27
8. 28

X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	5	36	26	0	0	0	0	0	0	0	0	0	0	0	0	
67	87	215	231	33	0	0	0	0	0	0	0	0	0	0	0	

50	223	253	253	136	0	0	0	0	0	0	0	0	0	0	0	3
176	253	253	253	70	0	0	0	0	0	0	0	0	0	0	0	14
148	253	253	191	6	0	0	0	0	0	0	0	0	0	0	0	7
78	253	253	183	0	0	0	0	0	0	0	0	0	0	0	0	
16	244	253	223	27	0	0	0	0	0	8	24	14	0	0	0	
0	152	253	253	67	0	0	0	0	0	150	253	142	0	0	0	
12	223	253	253	186	7	0	0	0	0	194	253	220	5	0	0	
12	223	253	253	253	153	4	0	0	0	194	253	219	99	78	28	2
0	109	253	253	253	253	12	0	0	0	194	253	236	234	253	206	13
0	4	47	172	253	253	120	0	0	0	53	235	253	253	253	253	25
0	0	25	216	253	253	193	130	55	0	40	233	253	253	253	253	25
0	0	0	112	234	253	253	253	237	122	97	253	240	224	224	224	22
0	0	0	0	31	212	253	253	253	253	253	253	121	0	0	0	
0	0	0	0	0	30	214	253	253	253	253	185	7	0	0	0	
0	0	0	0	0	0	28	170	170	170	170	53	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	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 [6]:

```
length(label_1)

pca_label_1 <- prcomp(test_update[,c(2:length(test_update))], center = T

#summary(pca_label_1)

# from pc1- pc496? the standard deviation goes below 0.05 after that

#plot(pca_label_1x[,1],pca_label_1x[,2])

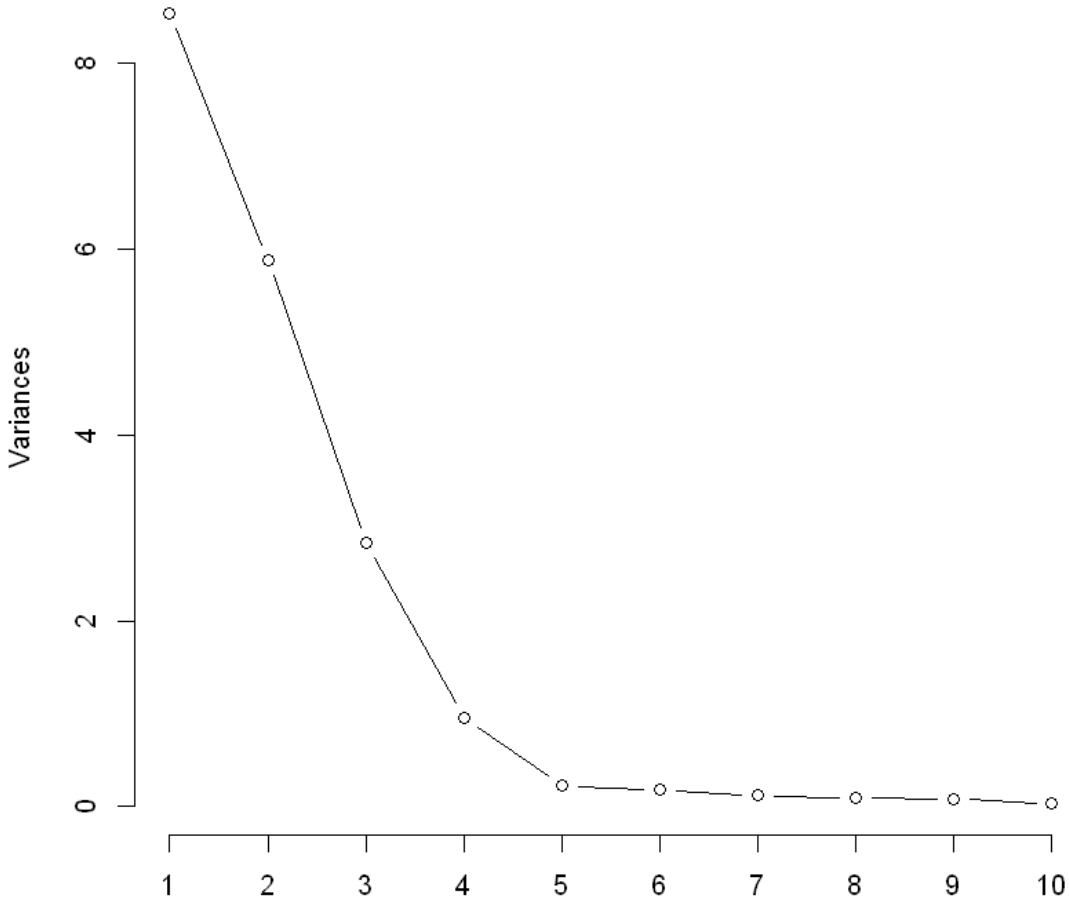
#biplot(pca_label_1, scale =0)
# use screeplot, find the pc cut off that contains 90% of the variance
# and convert it back to the data to see how the digits look

screeplot(pca_label_1, type = "lines")

#new_label_1 <- label_1[,c(2:496)]
#new_label_1
```

785

pca_label_1



In [7]:

```
standard_dev <- pca_label_1$sdev
# this is getting the standard deviation

var_label_1 <- standard_dev^2
# this is to get the variance of the pc
#var_label_1[1:10]

prop_var <- var_label_1/sum(var_label_1)
prop_var[1:3]
# this shows the proportion of variance explained


sum(prop_var[1:3])
# the first 3 pc (principal components) consist of 90% of the variance

# so now i need to take the first 35 pc and convert them back into the c
# data and see how the digit looks.
```

1. 0.449323565287185
2. 0.309497345718064
3. 0.149361985750576

0.908182896755825

In [8]:

```
# number 1

pc.use <- 4

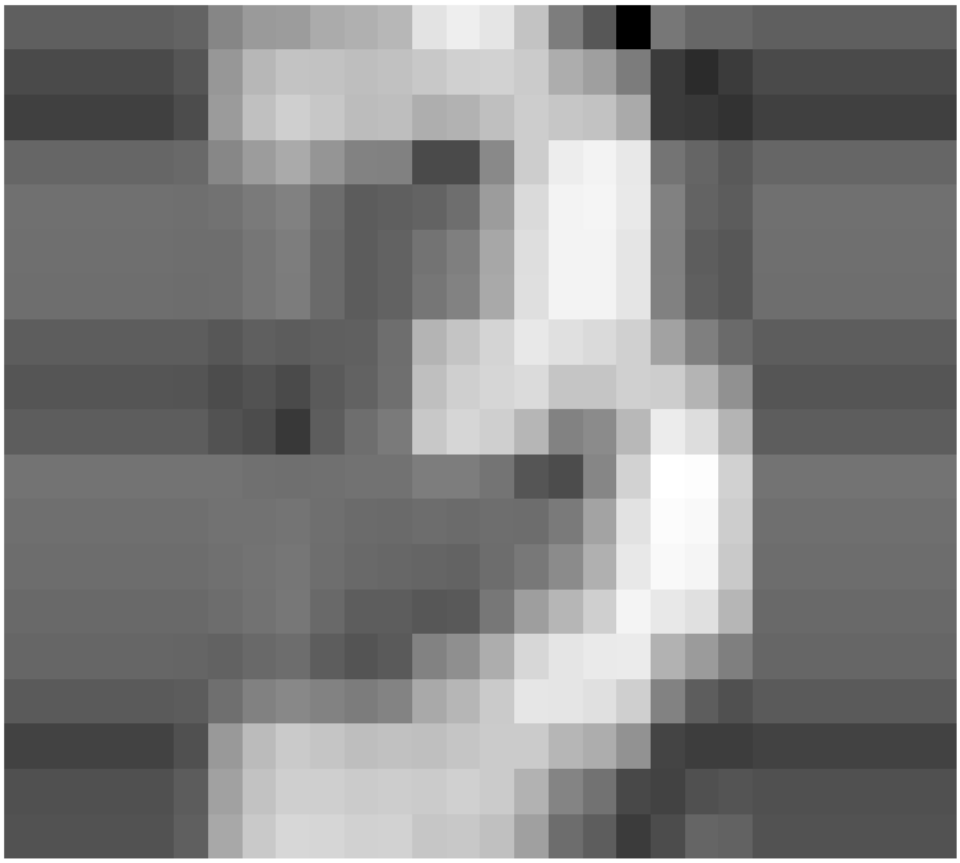
reconstructed <- pca_label_1$x[,1:pc.use] %*% t(pca_label_1$rotation[,1:

reconstructed
# this is the final matrix, within the variable reconstructed


image(reconstructed, axes =FALSE, col = grey.colors(256, start =0, end=1
# even though the above cell shows that 3 principal components account f
# 90% of the variance, i think using 4 is able to reproduce the digit re
# well
```

X6	X7	X8	X9	X10	X11	X1
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.6048609	-0.6352856	-0.7529194	-0.6427501	-0.54699252	-0.4957419	-0.44111247
0.5476336	0.4146480	0.2466798	-0.4225703	-0.57401439	-0.4407446	-0.39299063
1.3405545	1.1564015	0.9922401	-0.1620609	-0.49785690	-0.3792599	-0.37371643
1.7183426	1.4916277	1.3530434	-0.0540572	-0.44703767	-0.2977867	-0.32735642
1.6943396	1.4996481	1.2421715	-0.1505307	-0.61437851	-0.4939950	-0.43300597
1.5558440	1.3890948	1.0626934	-0.2431010	-0.72097520	-0.6195837	-0.50062432
1.5798620	1.4441968	1.1246396	-0.1301549	-0.64839389	-0.6271294	-0.51854009
1.2508881	1.4055468	1.0775944	0.5657874	-0.14822160	-0.6831950	-0.55047231
1.3023923	1.5323816	1.2381569	0.8645526	0.06421055	-0.6632490	-0.56560740
1.0922526	1.3794699	1.3853598	1.3612551	0.65300398	-0.3081666	-0.45154279
0.3905018	0.7791963	1.3940030	2.1718257	1.73478171	0.3470158	-0.28329655
-0.4492012	-0.1154588	0.8718181	2.1325264	2.13816448	0.8704012	-0.00499395
-0.6207145	-0.3757007	0.6651227	1.9782084	2.28846752	1.5023073	0.72436895
-0.9049414	-0.8120450	0.1237109	1.4949165	2.33832781	2.6120066	2.22950665
-0.7747248	-0.8636943	-0.8457336	-0.1472938	0.76679925	2.2288035	2.78199994
-0.5475356	-0.7546981	-0.8914260	-0.6249043	0.27778595	1.9939113	2.64941015
-0.5680083	-0.7144890	-0.8965326	-0.7487676	-0.18840910	0.8811843	1.32831858
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862

-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.44275862



In [9]:

```
# number 2

dist_recon <- dist(reconstructed)

# finding the distances of each row to use in the hclust function

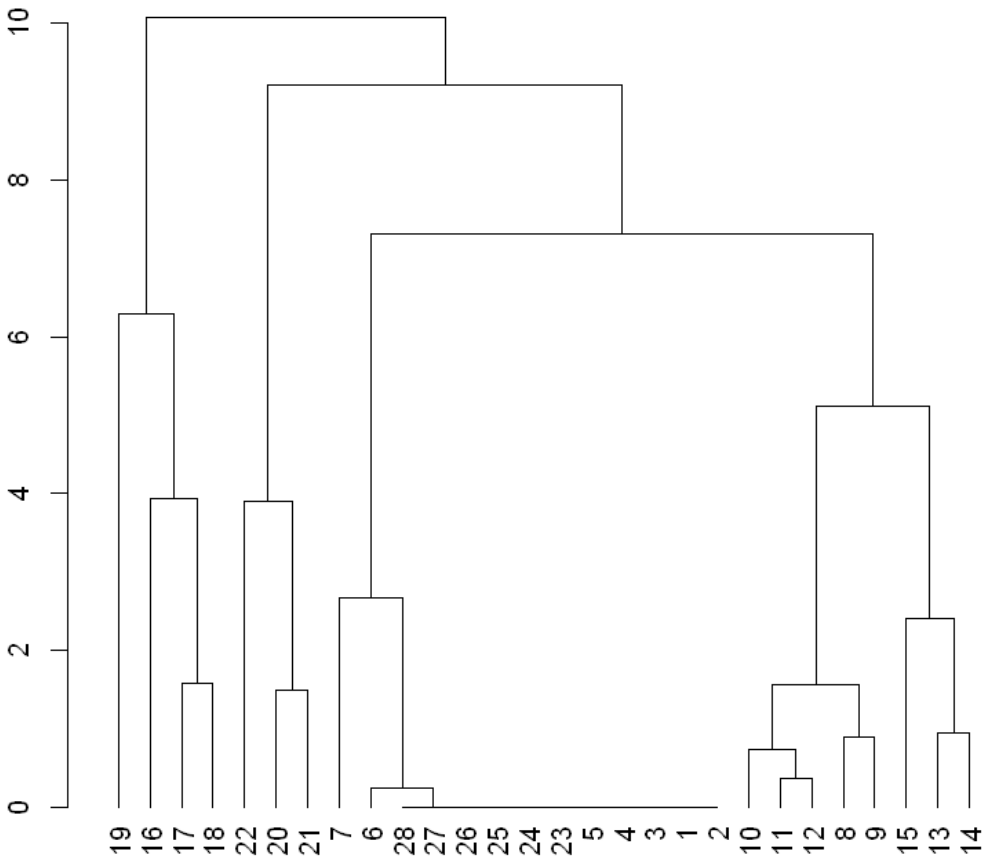
pc_hclust <- hclust(dist_recon, method='complete')


plot(as.dendrogram(pc_hclust))
# next plotting the hclust data as a dendrogram to visualize the relatic
# between each row.

# this clusters based on the rows
# what i can see is that in the center of the dendro gram we have the tc
# rows 1,2,3,4,5,6,7 being clustered together with the bottom rows 23,24
# 27,28 so this shows they have about the same distance values.
# next we see that the bottom middle rows are clustered together on the
```

```
# of the dendro gram which are rows 16,17,18,19,20,21,22.
# lastly, we see on the right of the dendrogram the top middle rows are
# together and the rows are 8,9,10,11,12,13,14,15


# NOTE: maybe i can try finding the distance of the columns and put that
# with the distance of the rows to get a better idea of each pixel someh
```



In [15]:

2066

In [12]:

```
# THOUGHT PROCESS GOING INTO MAKING THE NEXT CELL
# maybe we can use pca results instead
# we can take the average of each digit and find how many pc(principal c
# are needed to recreate 90% of the variance of the average
# then take one single digit and see if that single digit needs to use
# the same amount of pc to recreate 90% of the variance also
label_1[1,1]
```

1

In [10]:

```
# NUMBER 3 complete with written responce at the end of this cell

# this cell is from the midterm. we are using it to get the averages of


# 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 this cell to pass in all the labels_(0-9)




func_make_pc <- function (label){

pixel_label_0  <- label[,-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)
matrix_ave_pix

# UNCOMMENT THE BELOW CODE TO SEE THE IMAGE
#image(matrix_ave_pix, axes = FALSE, col = grey.colors(256, start=0, enc

# here we plot the matrix using image function, and make the colors
# shades of grey
#pixel_label_0


#THIS NEXT PART MAKES THE AVERAGE PIX INTO A MATRIX FOR PCA
```

```
# reused code from making pca of label 3 to make one of label 0
# at the end i found a way to automate the process of finding the least
# of pc's when added up gives 90%


test <- data.frame(matrix_ave_pix)
#test
# note: i am using "my_label1" variable from the 3rd cell. this variable
# uses code that gets information from the first 10 rows in the pixel da
# the last pixel data ends up being the digit "3"


# so the digit we are looking to run pca on is 3


list_to_remove <- c()

for (x in c(1:length(test))){

    if (var(test[,x]) == 0) {
        list_to_remove <- append(list_to_remove, x)
    }

}
list_to_remove
# I used this for loop to find all the columns that has a variance of 0
# the column number into the list_to_remove variable. now i will edit th
# matirx test and remove those columns

test_update <- test[,-(list_to_remove)]
test_update

pca_label_0 <- prcomp(test_update[,c(2:length(test_update))], center = T

# automate the below code to find how many pc equal 90%
standard_dev <- pca_label_0$sdev
# this is getting the standard deviation

var_label_0 <- standard_dev^2
# this is to get the variance of the pc
#var_label_1[1:10]

prop_var <- var_label_0/sum(var_label_0)
#prop_var[1:3]
# this shows the proportion of variance explained

# this while loop automates the proces of finding the least amount of
# pc's variances which add up to equal 90%
x <- 1
while (sum(prop_var[1:x])*100 < 90){
    x=x+1
}

# this keeps track of how many pc's it takes to create 90% of the variar
# for the average label data
#print(length(prop_var[1:x]))
```

```
print(sum(prop_var[1:x]))
assign(paste0("digit_",label[1,1]),  (length(prop_var[1:x])))

#digit_1
# here i am assigning the first column and row as the variable number wh
# creating the variable. that way i can keep track of how many pc's are
# each variable passed through the function
#sum(prop_var[1:3])

# now that i know 4 pc's is the least amount when added gives 90% of the
# variance information for the label 0 when averaged
# i can now look at an individual digit to see whether it will also need
# about 4 pc's to give its variance
# thus this will be my way of quickly classifying which digit i was give
# based on this quick calculation.




}

avg_digit_0_pc <- func_make_pc(label_0)
avg_digit_1_pc <- func_make_pc(label_1)
avg_digit_2_pc <- func_make_pc(label_2)
avg_digit_3_pc <- func_make_pc(label_3)
avg_digit_4_pc <- func_make_pc(label_4)
avg_digit_5_pc <- func_make_pc(label_5)
avg_digit_6_pc <- func_make_pc(label_6)
avg_digit_7_pc <- func_make_pc(label_7)
avg_digit_8_pc <- func_make_pc(label_8)
avg_digit_9_pc <- func_make_pc(label_9)


cat("the pc for 0 is :", avg_digit_0_pc,"\n")
cat("the pc for 1 is :",avg_digit_1_pc,"\n")
cat("the pc for 2 is :",avg_digit_2_pc,"\n")
cat("the pc for 3 is :",avg_digit_3_pc,"\n")
cat("the pc for 4 is :",avg_digit_4_pc,"\n")
cat("the pc for 5 is :",avg_digit_5_pc,"\n")
cat("the pc for 6 is :",avg_digit_6_pc,"\n")
cat("the pc for 7 is :",avg_digit_7_pc,"\n")
cat("the pc for 8 is :",avg_digit_8_pc,"\n")
cat("the pc for 9 is :",avg_digit_9_pc,"\n")

# this is how we will build our classifier. We know that the average of
# digit represents gives us a number of pc's that will add up to 90%
# variance
# so when we use the same code but only plug in one single digit we shou
# that that digit would need the same amount of pc's as its averaged cou
# to make 90% of the variances also.

# additional thing i noticed is that the odd numbers all needed 2 pc's w
# even numbers needed 3 pc's
# so we can definitely at least try to classify wheather the digit is ev
# or odd based on this information
```

```
# and the only digit we can classify is a 0, this is because its the onl
# digit that requires above 3 pc's to get to 90% variance
```

```
[1] 0.9483771
[1] 0.9238257
[1] 0.9710878
[1] 0.9446342
[1] 0.9704346
[1] 0.9395174
[1] 0.967781
[1] 0.9023977
[1] 0.9554669
[1] 0.9493153
the pc for 0 is : 4
the pc for 1 is : 2
the pc for 2 is : 3
the pc for 3 is : 2
the pc for 4 is : 3
the pc for 5 is : 2
the pc for 6 is : 3
the pc for 7 is : 2
the pc for 8 is : 3
the pc for 9 is : 2
```

In [11]:

```
new_pixel <- data.frame(data)
#test_pixel <- new_pixel[9,]
#test_label <- new_pixel[9,1]
#cat("this is the unknown digit: ", test_label, "\n")


#unknown_digit_pc <- func_make_pc(test_pixel)
#cat("this is the pc for the unknown digit: ", unknown_digit_pc, "\n")


# the average pc of the same digit fed into the classifier should be rou
# equal to the average pc we see in the cell above.

# so lets take 10 to 20 digits labeled 1 and see if it pc average ends u
# being 2, as seen as above.

# this code below looks through the entire original data set. then finds
# all of the row numbers that have the digit i will be testing
# next it takes only the first 10 of the rows numbers
new_test_df <- c()
for (x in c(1:length(new_pixel[,1]))){
if (new_pixel[x,1] == 1){
    # here is where we choose which digit to use
    new_test_df <- append(new_test_df, x)

}

}
#length(new_test_df)/2
# i tried to do half of the digit data as tests but it doesnt work
only_20_of_digit <- sample(new_test_df, 20)
#only_20_of_digit <- new_test_df
#only_20_of_digit
```

```
# now i want to loop through the row numbers with the function to get ea
# pc for the digit. note all the digits should be the same
# we will then take the average of all 20 pc and hope it equals the
# above average pc's for that number

take_avg <- c()
for (x in only_20_of_digit){
new_pixel <- data.frame(data)
test_pixel <- new_pixel[x,]
test_label <- new_pixel[x,1]
cat("this is the unknown digit: ", test_label, "\n")


unknown_digit_pc <- func_make_pc(test_pixel)
cat("this is the pc for the unknown digit: ", unknown_digit_pc, "\n\n")
take_avg <- append(take_avg, unknown_digit_pc)
}

# as you can see the unknown digits are 1 like we want
# now to take the average of each pc which is,
new_list_avg <- c()
for (x in c(1:length(take_avg))){
if (take_avg[x]< 5){
    new_list_avg <- append(new_list_avg,take_avg[x])
}
}
new_list_avg
mean(new_list_avg)
# i wanted to remove any pc's that were above 4, because they would be c
# interferring with the classifier.
# none of the average pc's reached above 5.
```

```
this is the unknown digit:  1
[1] 0.90721
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9417889
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9328796
this is the pc for the unknown digit:  3

this is the unknown digit:  1
[1] 0.904581
this is the pc for the unknown digit:  5

this is the unknown digit:  1
[1] 0.903382
this is the pc for the unknown digit:  5

this is the unknown digit:  1
[1] 0.9897542
this is the pc for the unknown digit:  2

this is the unknown digit:  1
[1] 0.9494501
this is the pc for the unknown digit:  2
```

```
this is the unknown digit:  1
[1] 0.9562665
this is the pc for the unknown digit:  2

this is the unknown digit:  1
[1] 0.9641385
this is the pc for the unknown digit:  2

this is the unknown digit:  1
[1] 0.9708957
this is the pc for the unknown digit:  3

this is the unknown digit:  1
[1] 0.9395901
this is the pc for the unknown digit:  3

this is the unknown digit:  1
[1] 0.9382433
this is the pc for the unknown digit:  3

this is the unknown digit:  1
[1] 0.9534331
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9727557
this is the pc for the unknown digit:  2

this is the unknown digit:  1
[1] 0.945469
this is the pc for the unknown digit:  1

this is the unknown digit:  1
[1] 0.939379
this is the pc for the unknown digit:  3

this is the unknown digit:  1
[1] 0.9328765
this is the pc for the unknown digit:  2

this is the unknown digit:  1
[1] 0.9318862
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9265655
this is the pc for the unknown digit:  4

this is the unknown digit:  1
[1] 0.9633652
this is the pc for the unknown digit:  3
```

1. 4
2. 4
3. 3
4. 2
5. 2

6. 2
7. 2
8. 3
9. 3
10. 3
11. 4
12. 2
13. 1
14. 3
15. 2
16. 4
17. 4
18. 3

2.83333333333333

In [ ]:

Tara91 / Final Private

Code Issues Pull requests Actions Projects Security Insights

main

Final / Final.Rmd

Tara91 Add files via upload History

1 contributor

351 lines (240 sloc) | 10.6 KB

```
---
title: "Final"
output: html_notebook
---
Tarabeth Gerrity
TG64

===========
Project 1.
===========
This data is about basketball players from the year 2008 and is
in the file ppg2008.csv. It has various statistics on players in the NBA.
You might not know what each of the metrics means (I don't),
but they are just different dimensions of data.

This is a visualization and data mining exercise.

What can you say about this dataset, use tools that you learned here.
and make a report or a visual that highlights something interesting,
maybe compare players, especially how they have performed since,
based on the data in here. Many of these players have reached
their peak recently and you will be able to find statistics about
their performance in 2019.

Could you have predicted the successes and failures of some of the players, based
on analyses of the data ?
maybe you could be a talent scout for an NBA team ?


```

````
Think of it as your job, as a reporter for NY times, to make a single graphic
that highlights something about this data. Explain the analysis that went
into the graphic and present the code too. This should be done in a notebook so
it is easy to evaluate.



==========
Project 2.
==========

Dataset 2:  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

```{r}
train <- read.csv("/Users/morgenstern/Desktop/BioStats/Final/train.csv")

#column1, label- which digit is drawn
#columns 2 - 785- pixels (784 pixels total)
# digit is represented as pixels in a 28x28 square

for (i in 1:10){
  m <- matrix(as.numeric((train[i,2:785])),nrow=28, ncol=28, byrow = TRUE)#digits drawn up
  m <- m[nrow(m):1,]#flips the matrix to be upside up/flips across the horizontal
  heatmap((m),Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
}
###### 0 = white
###### 255 = black

```



1) Use PCA to reduce dimensions.  How many components do you need to keep
to reproduce the digits reasonably well ? what is your final matrix ?
```{r}
library(fpc)
library(resample)
set.seed(7)
##test program on subset of train
#test <- (train[0:200,2:785])
##test full train dataset
test <- train[,2:785]
##remove all pixels with 0 variance--- all columns with only 0's
##Since we're not scaling in PCA, nozero dataframs uneeded untill pheatmap
````
````

```
i <- (colVars(test, na.rm=T) > 0.5) # T if col variance is not 0, F otherwise
nonzero <- test[i] # all the non-zero columns
nonzero##should have only pixels with variance
zerocols <- test[(colVars(test, na.rm=T) <= 0.5)] #Store the 0 variance columns to re-add
##dataframe in a later step
colVars(nonzero[,1])

### Do no include digit label in analysis!
#initialize the plot

kclusters <- kmeans(nonzero,1)
plot(1,kclusters$tot.withinss,xlim=c(0,10))

for (i in 2:10){
  kclusters <- kmeans(nonzero,i)
  points(i,kclusters$tot.withinss)
}

#Wait, shouldn't we just force 10 clusters, 1 for each digit? No breakpoint needed.

ebp <- kmeans(nonzero,10)
ebp$tot.withinss

ebp$cluster
## Which cluster each sample falls under


##plots clusters
aggregate(nonzero, by=list(cluster=ebp$cluster),mean)
plotcluster(nonzero,ebp$cluster)
#

#create the pca object
test.pca <- prcomp(nonzero,center = FALSE,scale. = FALSE)
print(test.pca)

plot(test.pca)
summary(test.pca)
library(ggbiplot)


ggbiplot(test.pca,obs.scale = 1, var.scale = 1, var.axes=FALSE, labels=(train$label[1:leng
#var.axes=FALSE removes the arrows and pixelnames, to see the grouping better

### Can alter this to label points instead with the actual digit assignment.
##(train[0:50,1])
## Use this to give line # instead
#labels=row.names(test)

```

````
```
```{r}

nComp = 50 ## 200 reproduces original matrix. ~50 the 4 becomes unrecognizable
##other numbers fare better at lower component numbers though
#mu = colMeans(test)
Xhat = test.pca$x[,1:nComp] %*% t(test.pca$rotation[,1:nComp])
#Xhat = scale(Xhat, center = -mu, scale = FALSE)

#loop through original data re-adding any of the missing columns
## $ operator won't work inside function, use [[]] to call variable instead
Phat <- data.frame(Xhat)
Phat <- cbind(Phat, zerocols) ##add back in the zerovar columns

colnames(train)
colnames(Phat)

#install.packages("gtools")
library(gtools)
Phat <- Phat[, mixedsort(colnames(Phat))]
##use this package to sort
##sort orders columns pixel1, pixel11,pixel111 instead of pixel1, pixel2 etc
##Graph the new matrix

##Why does the resulting matrix contain negative values?
Phat[Phat < 1] <- 0


for (i in 1:10){
  m <- matrix(as.numeric((Phat[i,1:784])),nrow=28, ncol=28, byrow = TRUE)#digits drawn ups
  m <- m[nrow(m):1,]#flips the matrix to be upside up/flips across the horizontal
  heatmap((m),Rowv = NA, Colv = NA, col = gray.colors(256, start = 1, end = 0))
}

## Testing the matricies

#pm <- matrix(as.numeric((Phat[9,1:784])),nrow=28, ncol=28, byrow = TRUE)
#heatmap((pm), Rowv=NA, Colv = NA, col = gray.colors(256, start = 1, end=0))

#om <- matrix(as.numeric((train[9,2:785])),nrow=28, ncol=28, byrow = TRUE)
#heatmap((om), Rowv=NA, Colv = NA, col = gray.colors(256, start = 1, end=0))


```


2) Draw a tree of the pixels, and see if you can explain the results based on
geometry of the pixels (how far apart are they in the 2-d space).
````
````

````
Try to Explain the PCA results in light of this.

```{r}
##Note, 2D space. The closest 8 pixels will not be sequential

##Use euclidean correlation?


test
dm <- dist(t(test), method = "euclidean")
test.hclust <- hclust(dm, method = "complete")
#install.packages("dendextend")
library(dendextend)
#par(mar=c(5,5,5,12))
#nPar <- list(lab.cex = 0.6, pch = c(NA, 19),cex = 0.7, col = "blue")
#ePar = list(col = 2:3, lwd = 2:1)
plot(as.dendrogram(test.hclust),horiz=TRUE)

pdf("/Users/morgenstern/Desktop/test.pdf", width=500, height=100)

# Do some plotting
plot(as.dendrogram(test.hclust),horiz=TRUE)
# Close the PDF file's associated graphics device (necessary to finalize the output)
dev.off()

##A tree. Not very useful on its own. Can we do a heatmap for better visualization?

```
```{r}

library(pheatmap)

pdf("/Users/morgenstern/Desktop/testheatmap2.pdf", width=300, height=100)
## Take the original train dataset

trainsort <- train[1:200, 1:785]
trainsort <- trainsort[sort(trainsort$label),]
#sorting the dataframe by label column groups each row by digits.


##This heat map groups samples into digits- pixel correlation patterns emerge
pheatmap(trainsort,
        cellwidth = 20,
        cutree_rows = length(unique(trainsort$label)))
dev.off()

##doing the same but without the zero variance pixels

pdf("/Users/morgenstern/Desktop/nozeropheatmap.pdf", width=200, height=100)
````
````

```
trainsort <- train[1:200, 1:785]
trainsort <- trainsort[sort(trainsort$label),]
trainsort <- trainsort[, mixedsort(colnames(trainsort))]
i <- (colVars(trainsort, na.rm=T) > 0) # T if col variance is not 0, F otherwise
nonzerotrainsort <- trainsort[i] # all the non-zero columns
nonzerotrainsort##should have only pixels with variance

pheatmap(nonzerotrainsort,
         cellwidth = 20)
dev.off()


```

3) Can you use some of the tools you have learnt to build a classifier,
so if you get a new set of pixels you can predict what is in the
picture. This is a the start of a real project, but you don't have all the tools
 (such as neural networks) which might be more suited for this task.
Split your dataset into two (a training set and a test set),
build your classifier and figure out
how well it does with the test data in predicting the digits.
Define the sensitivity and specificity of your classifier.
How well does it recognize your own handwriting
(make sure your handwriting is not in the training set)

```{r}

training <- data.frame(train[0:200,2:785])

testing <- data.frame(train[500,2:785])

myDigit <- data.frame()


testsubject <- data.frame() ## a line of pixel values

nComp = 50 ## Reuse value from previous step
PredictedDigit <- predict(test.pca, newdata = testing)

Prediction <- PredictedDigit[, 1:nComp]

ebp <- kmeans(training,10)
ebp$tot.withinss
ebp$cluster
## Which cluster each sample falls under
##plots clusters
aggregate(training, by=list(cluster=ebp$cluster),mean)
plotcluster(training,ebp$cluster)
```

```
#

#create the pca object
training.pca <- prcomp(training,center = FALSE,scale. = FALSE)

summary(training.pca)

TrainingPlot <- ggbiplot(training.pca,obs.scale = 1, var.scale = 1, var.axes=FALSE, labels

TrainingPlot

PredictionPlot <- geom_point(aes(x=Prediction[1], y=Prediction[2]), colour="Red", size =4)
print(TrainingPlot + PredictionPlot)
print(train[500,1])

```

```
plot_3 <- plot_2 + geom_point(aes(x=new_cus_group_PC1_PC2[1], y=new_cus_group_PC1_PC2[2]),

plot_3 <- plot_3 + labs(title="Plotting new observation against PC1 and PC2")

print(plot_3)
```

```
4) You can try simple things like take average of all data for each number and
then take a "dot" product with your test set, and identify the pixels. This might
work, maybe for some digits, and not others.
```

```
===========
Project 3.
===========

Dataset 3: 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 genes.


1) Build hierarchical trees based on the columns and for the rows (exclude rows
that are "low" expression)

2) Draw a heat map of the expression data

3) Use DESeq2 to analyse this data
   a) which are the most significantly changing genes in this dataset ?
   b) which genes are most consistently highly expressed in these datasets
       they are the "house-keeping" genes
   c) How consistent are these results with the analysis you did
      in the midterm project ?
   d) What else can you say about the data in terms of consistency,
   and the results that you find from your analyses. The question is open-ended,
   think of this as your experiment, you need to write a paper based on this data
   so you have to figure out what kind of "story" you can tell based on this.

 e) what is the most interesting pathway or gene that is responding in this study?
```

```
---
title: "RNAseqHW"
author: "Carolee Nguyen"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, error=TRUE)
```

For dataset 2
1) Build hierarchical trees based on the columns and for the rows (exclude rows
that are "low" expression)
2) Draw a heat map of the expression data
3) Use DESeq2 to analyse this data
   a) which are the most significantly changing genes in this dataset ?
   b) which genes are most consistently highly expressed in these datasets
       they are the "house-keeping" genes
   c) How consistent are these results with the analysis you did
      in the midterm project ?
   d) What else can you say about the data in terms of consistency,
   and the results that you find from your analyses. The question is open-ended,
   think of this as your experiment, you need to write a paper based on this data
   so you have to figure out what kind of "story" you can tell based on this.

   e) what is the most interesting pathway or gene that is responding in this study ?


```{r readdata}
```
```

````
30
31  countdata = read.csv(file = "/Users/carol/Downloads/Mnemiopsis_count_data.csv", header=TRU
32  countdata = as.data.frame(countdata) #Read data file as a data frame
33
34  print(grep("aboral", countdata)) #Use grep function to extract all cases of "aboral"
35  print(grep("oral", countdata)) #Use grep function to extract all cases of "oral"
36
37  dfaboral <- countdata[,c(2,3,4,5)] #Create a dataframe for aboral
38  dforal <- countdata[,c(6,7,8,9)] #Create a dataframe for oral
39
40
41
42  dfcounts <- countdata[,-1]
43  rownames(dfcounts) <- countdata[,1]
44
45
46  print(dfcounts)
47  ```
48
49
50  ### Step 2
51
52
53  ```{r createfactor}
54
55
56
57  expgroup <- data.frame(condition=1:8) #Create a dataframe with 8 values corresponding to t
58  expgroup$condition[c(1:4)] <- 'Aboral' #Assign the first 20 values as healthy using grep i
59  expgroup$condition[c(5:8)] <- 'Oral' #Assign the last 20 values as CF using grep informati
60
61
62  rownames(expgroup) <- colnames(countdata[,c(2:9)]) #Use the column names of read count as
63
64  print(expgroup) #Print the expgroup dataframe
65  ```
66
67  ### Step 3
68
69  ```{r runDESeq2}
70
71  library("DESeq2") #Load DeSeq2 library
72  cds <- DESeqDataSetFromMatrix(countData = dfcounts, #Create a count data set matrix named
73                                colData = expgroup, #Set colData as expgroup
74                                design = ~ condition) #Set design as condition
75  cds #Call the matrix
76
77  ```
78
````

````
### Step 4


```{r estimate}

#Utilize the previous count data set matrix in the functions below:
cds <- estimateSizeFactors(cds) #Function for estimating size factors
cds <- estimateDispersions(cds) #Function for estimating dispersion
plotDispEsts(cds) #Function for plotting dispersion




```

### Step 5

```{r deseqres}

#Utilize the previous count data set matrix in the functions below:
cds <- DESeq(cds) #perform differential gene expression
res <- results(cds) #use results function to obtain differential gene expression results
res #Output DeSeq2 results


```

### Step 6



```{r diffexp}
#Look for genes that have an adjust p-value <0.05 and greater than 1
resSigind = res[ which(res$padj < 0.05 & res$log2FoldChange > 1), ]
#Look for genes that have an adjust p-value <0.05 and less than 1
resSigrep = res[ which(res$padj < 0.05 & res$log2FoldChange < -1), ]
#Combine the two previous results together
diffexpgenes = rbind(resSigind, resSigrep)
#Print the row names, which is the list of differentially expressed genes
rownames(diffexpgenes)

nrow(diffexpgenes)
```


### Step 7
````

````
```{r normvalues}
#Utilize the previous count data set matrix in the functions below:
normvalues = counts(cds, normalized=TRUE)
print(normvalues) #print the normalized values

colnames(normvalues) #Show column names for matching in the step below
```

### Step 8


```{r diffvalues}

###match(rownames(diffexpgenes), rownames(normvalues))
diffexpvalues = normvalues[rownames(normvalues) %in% rownames(diffexpgenes), ]
#Print the dataframe
print(diffexpvalues)
#Print the dimension of the dataframe to see if it matches the list of differentially expr
print(dim(diffexpvalues))


aboral1 = sort(diffexpvalues[,1])
tail(aboral1)
#ML174735a ML034334a ML034337a ML034336a  ML01482a  ML46651a
#17157.06  21911.36  22424.14  23631.30  30581.63 118211.07

aboral2 = sort(diffexpvalues[,2])
tail(aboral2)
#ML174735a ML034337a ML034336a ML034334a  ML01482a  ML46651a
#40418.84  58540.29  65914.50  68219.38  80559.11  93870.29

aboral3 = sort(diffexpvalues[,3])
tail(aboral3)
#ML174735a ML034337a ML034336a ML034334a  ML46651a  ML01482a
#27488.06  47004.61  50998.50  55714.11  56425.56  71249.61

aboral4 = sort(diffexpvalues[,4])
tail(aboral4)
#ML174735a ML034337a ML034336a  ML46651a ML034334a  ML01482a
#37089.43  67521.80  82850.63  91834.25  92639.95 110042.52

oral1 = sort(diffexpvalues[,5])
tail(oral1)
#ML01433a ML087114a ML199832a ML004510a  ML21583a ML306119a
#14912.24  17974.26  18039.35  22064.90  23649.09  31192.30

oral2 = sort(diffexpvalues[,6])
tail(oral2)
#ML11032a ML306119a  ML34341a ML087114a ML199832a ML004510a
````
````

```
#16119.59  16538.89  18610.45  20852.11  21035.21  34514.43

oral3 = sort(diffexpvalues[,7])
tail(oral3)
#ML46651a  ML43587a  ML11032a ML087114a  ML01482a ML004510a
#27853.26  29726.57  30124.16  42894.61  44780.12  66495.13

oral4 = sort(diffexpvalues[,8])
tail(oral4)
#ML01433a  ML11032a  ML09436a ML087114a  ML46651a ML004510a
#26045.23  28540.28  33828.03  39617.47  64936.93  77464.55
```

Step 9

```{r diffgroups}

#Create a hierarchical cluster

hc <- hclust(dist(diffexpgenes), method = "complete")

plot = plot(hc, main = "Dendrogram", horiz = TRUE, cex = 0.5)

print(plot)



```

Step 10

```{r heatmap}

#Load the pheatmap package
library(pheatmap)

#Plot the heatmap using designated variables
pheatmap(diffexpvalues,
         color = colorRampPalette(c("navy", "white", "firebrick3"))(50),
         scale = "row",
         cluster_rows = TRUE,
         cellwidth = 5,
         cellheight = 3,
         fontsize = 5,
         width=2000,
```

```
          height=800
          )


pheatmap(diffexpvalues)
```

Step 11

```{r eval=F}
#Install all Bioconductor packages
BiocManager::install("GOstats")
BiocManager::install("GO.db")
BiocManager::install("Category")
BiocManager::install("org.Hs.eg.db")


```

```{r loadlibraries}
#Load all libraries

library(GOstats)
library(GO.db)
library(Category)
library(org.Hs.eg.db)

```



```{r gostats}
#Create a GOHyperParams object using varaibles created from the previous steps
params=new("GOHyperGParams",
  geneIds=rownames(diffexpgenes),
  universeGeneIds=rownames(readcount),
  annotation="org.Hs.eg",
  ontology="BP",
  pvalueCutoff=0.001,
  conditional=TRUE,
  testDirection="over")

#Run HyperFTest
(overRepresented=hyperGTest(params))
#Print out summary of desired columns
summary(overRepresented)[,c(1,2,5,6,7)]

```

201 lines (184 sloc) | 6.46 KB

```
# Dataset 1
# 1) Use PCA to reduce dimensions.  How many components do you need to keep
# to reproduce the digits reasonably well? what is your final matrix ?
data <- read.csv("/Users/sarahferaidoon/Desktop/train.csv.gz")
data

# Number 2
n <- c(0:9)
# keep track of all digits corresponding to a handwriting from
# 0 to 9 in variable n
for (x in n){
  # loop through each digit
  # assign puts all data we want to grep in a variable
  # paste0 creates variable names dynamically
  # grep takes digit from first column following pattern
  # put entire row matching digit into a variable
  # loop moves onto next digit
  assign(paste0("label_",x),data[grep(x,data[1]),])
}
label_0
label_1
label_2
label_3
label_4
label_5
label_6
label_7
label_8
label_9
data[grep("0",data[,1]),]


length(label_1)
# scale FALSE to take out columns with variance of 0
pca_label_1 <- prcomp(label_1[,c(2:784)],center=TRUE,scale=FALSE)
summary(pca_label_1)
# St dev goes below 0.05 after PC496. It is above 0.05 from PC1-PC496
plot(pca_label_1$x[,1],pca_label_1$x[,2])
biplot(pca_label_1, scale=0)
# using screeplot, find the pc cut off that contains 90% of the variance
```

```r
# and convert it back to the data to see how the digits look
screeplot(pca_label_1,type="lines")
new_label_1 <- label_1[,c(2:496)]

# get standard deviation
standard_dev <- pca_label_1$sdev
# get variance of pc
var_label_1 <- standard_dev^2
var_label_1[1:10]

# explain proportion of variance
prop_var <- var_label_1/sum(var_label_1)
prop_var[1:35]
# the first 35 pc (principal components) consist of 90% of the variance
sum(prop_var[1:35])
#convert the first 35 pc back into data and see how digit looks
pc.use <- 35
???reconstructed <- pca_label_1$x[,1:pc.use] %*% t(pca_label_1$rotation[,1:pc.use])
reconstructed
image(reconstructed, axes=FALSE, col=grey.colors(256, start=0, end=1))
pca_label_1$x[,1:pc.use]


# Dataset 2
# 1) Build hierarchical trees based on the columns and for
# the rows (exclude rows that are "low" expression)
# read data file as variable
countdata <- read.csv(file="/Users/sarahferaidoon/Desktop/Mnemiopsis_count_data.csv", header=TRUE)
# read data file as dataframe
countdata <- as.data.frame(countdata)
countdata
# use grep function to extract all cases of "aboral"
print(grep("aboral",countdata))
# use grep function to extract all cases of "oral"
print(grep("oral",countdata))
# create dataframe for aboral
dfaboral <- countdata[,c(2,3,4,5)]
# create dataframe for oral
dforal <- countdata[,c(6,7,8,9)]
dfcounts <- countdata[,-1]
rownames(dfcounts) <- countdata[,1]
print(dfcounts)
# Create a dataframe called expgroup, with 1 column labeled condition (assigning healthy column). column names of
# Create a df with 8 values (corresponding to the columns in readcount)
expgroup <- data.frame(condition=1:8)
# use grep info and assign the first 20 values as healthy
expgroup$condition[c(1:4)] <- "Aboral"
# use grep info and assign the last 20 values as CF
expgroup$condition[c(5:8)] <- "Oral"
# use column names of read count as row names of expgroup
rownames(expgroup) <- colnames(countdata[,c(2:9)])
# print the df
print(expgroup)

# Step 3
# Load SeSeq2 library
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("DESeq2")
```

```r
library("DESeq2")
cds <- DESeqDataSetFromMatrix(countData = dfcounts, colData=expgroup, design = ~condition)
cds

#Step 4
# Make sure DESeq2 can correct for library size and dispersion estimates
# Use functions estimateSizeFactors and estimateDispersions
# Plot the dispersion using plotDispEsts
# Analyze graph
# Function for estimating size factors
cds <- estimateSizeFactors(cds)
# Function for estimating dispersion
cds <- estimateDispersions((cds))
# Function for plotting dispersion
plotDispEsts(cds)

# Step 5
# Perform differential expression with DeSeq and results functions
# perform differential gene expression
cds <- DESeq(cds)
# Obtain differential gene expression results
res <- results(cds)
# Show output
res

# Step 6
# Find p-values less than 0.05 and log2FoldChange > 1 or less than -1
# How many? save in list: diffexpgenes
# Genes with adjust p-value <0.05 and greater than 1
ressigind <- res[which(res$padj < 0.05 & res$log2FoldChange > 1),]
# Genes with adjust p-value <0.05 and less than -1
ressigrep <- res[which(res$padj < 0.05 & res$log2FoldChange < -1),]
# Combine results
diffexpgenes <- rbind(ressigind, ressigrep)
# print list of differentially expressed genes
rownames(diffexpgenes)
nrow(diffexpgenes)
# ~ 1500 genes

# Step 7
# Normalize values of counts data in cds with counts() function and normalized=Tm call this normvalues
normvalues <- counts(cds, normalized=TRUE)
# print normalized values
print(normvalues)
#Show columns names for matching
colnames(normvalues)

# Step 8
# Create new matrix/dataframe containing expression values from normvalues for only diffexpgenes, call this diffe
diffexpvalues <- normvalues[rownames(normvalues) %in% rownames(diffexpgenes),]
# print the dataframe
print(diffexpvalues)
# see if this matches the list of differentially expressed genes from last step
print(dim(diffexpvalues))

# Step 9
# Create hierarchical cluster
hc <- hclust(dist(diffexpgenes), method="complete")
plot <- plot(hc, main="Dendrogram", horiz=TRUE, cex=0.5)
print(plot)
```

```r

# Step 10
# Create a heat map using pheatmap package
install.packages("pheatmap")
library(pheatmap)
# Plot the heatmap using designated variables
pheatmap(diffexpvalues,
         color=colorRampPalette(c("navy","white","firebrick3"))(50),
         scale="row",
         cluster_rows=TRUE,
         cellwidth=5,
         cellheight=3,
         fontsize=5)

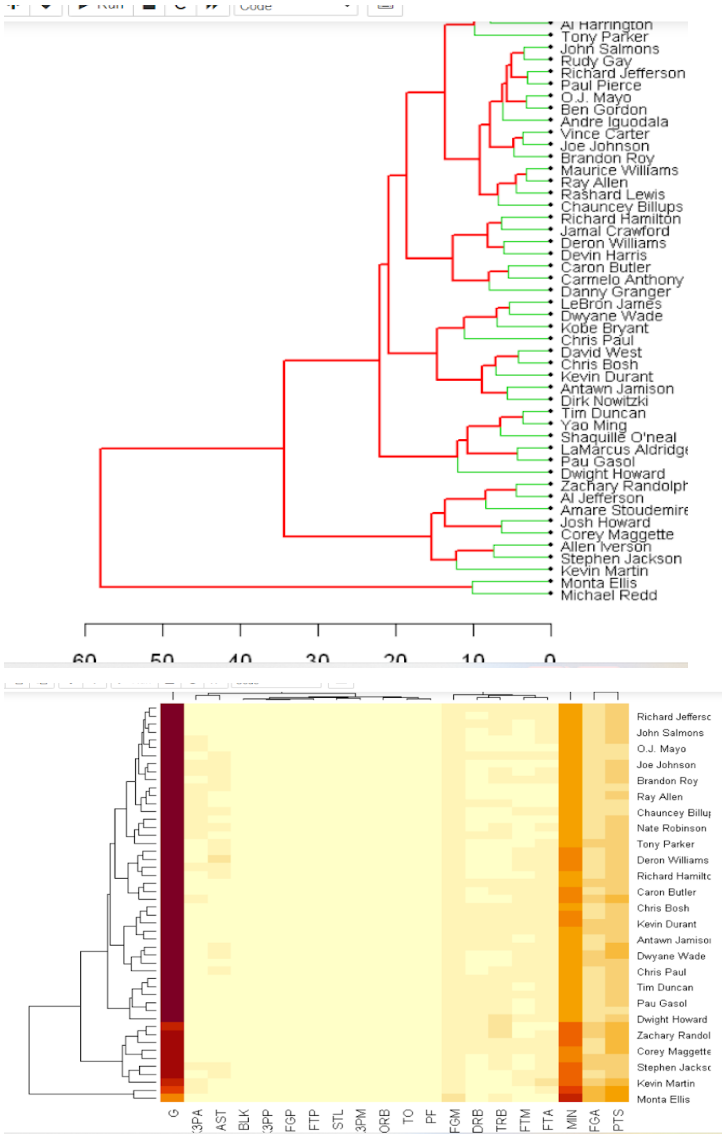
# Step 11
# Use GOStats package to determine which GO-terms are enriched in diffexpgenes
# install the following packages from Bioconductor:
BiocManager::install("GOstats")
BiocManager::install("GO.db")
BiocManager::install("Category")
BiocManager::install("org.Hs.eg.db")
if (!require("BiocManager", quietly = TRUE))
  install.packages("Category")
# Load libraries
library(GOstats)
library(GO.db)
library(Category)
library(org.Hs.eg.db)
# Create a GOHyperParams object using variables created from steps
params <- new("GOHyperGParams",
              geneIds=rownames(diffexpgenes),
              universeGeneIds=rownames(diffexpvalues),
              annotation="org.Hs.eg",
              ontology="BP",
              pvalueCutoff=0.001,
              conditional=TRUE,
              testDirection="over")
# Run HyperFTest
(overRepresented-hyperGTest(params))
# Print summary of desired columns
summary(overRepresented)[,c(1,2,5,6,7)]
```

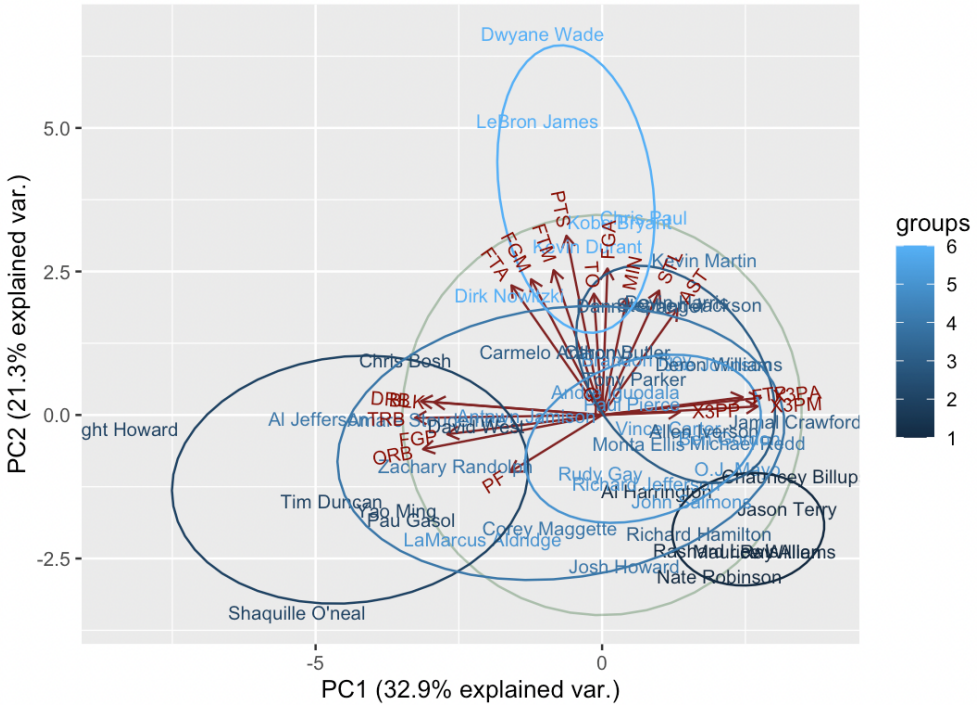
Ryan Johnson
Carolee Nguyen
Sarah Feraidoon
Tarbeth Gerrity

Applied Biostatistics Final:

Project 1.
==========
This data is about basketball players from the year 2008 and is in the file ppg2008.csv. It has various statistics on players in the NBA. You might not know what each of the metrics means (I don't), but they are just different dimensions of data.

This is a visualization and data mining exercise.





What can you say about this dataset, use tools that you learned here and make a report or a visual that highlights something interesting, maybe compare players, especially how they have performed since, based on the data in here. Many of these players have reached their peak recently and you will be able to find statistics about their performance in 2019.

Could you have predicted the successes and failures of some of the players, based on analyses of the data ? Maybe you could be a talent scout for an NBA team ?

It seems like the dimensions, G, MIN, and PTS are very closely related amongst the players when looking at the heatmap. G especially looks like many of the players score a similar number for this parameter, which stands for games played. (we looked it up). So it makes sense that many of the top named players have similar total games played and the outliers in a lighter color must be newer players.

Think of it as your job, as a reporter for NY times, to make a single graphic that highlights something about this data. Explain the analysis that went into the graphic and present the code too. This should be done in a notebook so it is easy to evaluate.

Code to make the heat map and dendrogram:

```
dm <- dist(df_players)
#dm
hclust_data <- hclust(dm,method='complete')
hclust_data
# creating a hclust matrix of the distance data from the players.

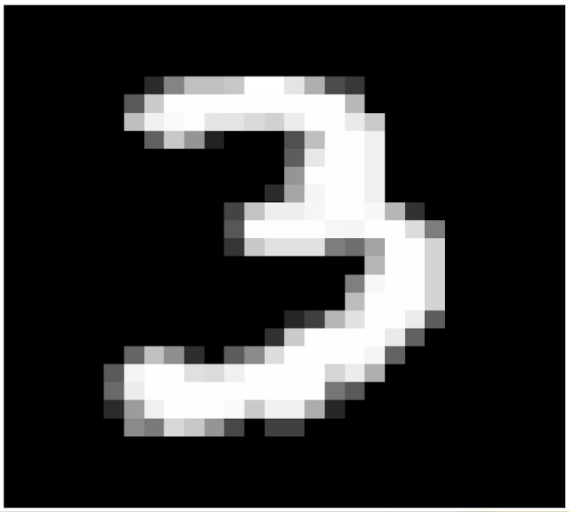
#library(dendextend)
npar  <- list(lab.cex=0.7, pch=c(NA,20),cex=0.7)
par(mar=c(5,5,3,12), cex.axis=1)
epar = list(col=2:3,lwd=2:1)
plot(as.dendrogram(hclust_data),nodePar=npar, edgePar= epar, horiz=TRUE)
# above i plotted a dendrogram of the hclust data generated above to visualize
# the data in a tree and see the clusters of players

df_players_matrix <- as.matrix(df_players)
#df_players_matrix
# i created a matrix of the players stats then made a heatmap to visualize it
heatmap(df_players_matrix, margins=c(5,5), hclustfun=hclust)
```

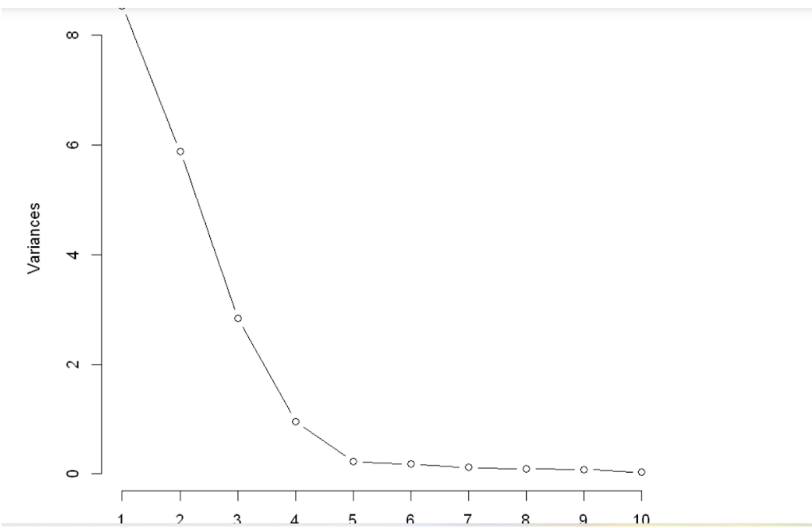
Project 2.
==========

1) Use PCA to reduce dimensions. How many components do you need to keep to reproduce the digits reasonably well ? what is your final matrix ?

We used data from the digit 3. This is how it looks before pca

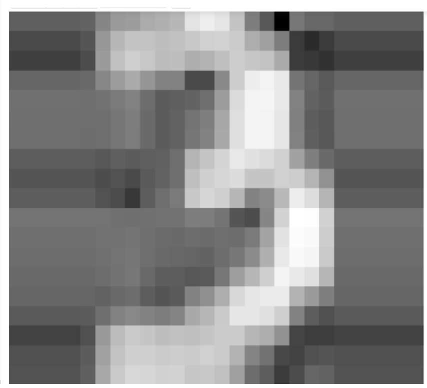


After looking at the screeplot we can see that only 3 principal components are needed to reproduce the digits reasonably well for this digit. Here is the screeplot:



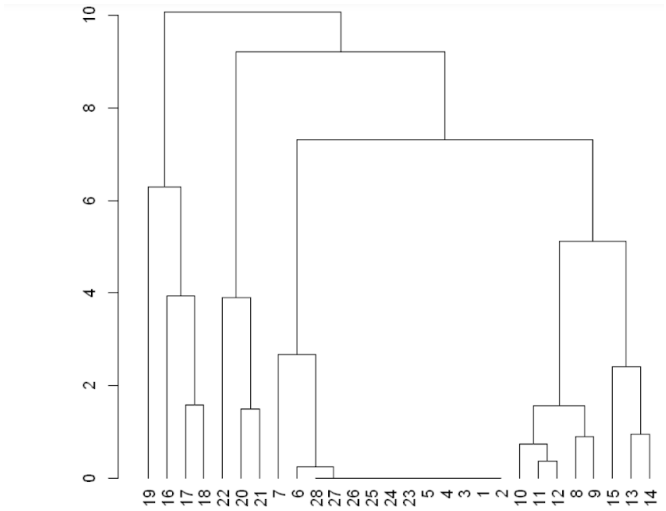
After running a test we saw that only 3 pc's were needed to get 90% of the variances, however we decided to go with using 4 pc's to recreate the image because it ends up looking more distinguishable. Here is a small cutout of how the final matrix looks for digit 3 after using only the 4 pc's and also how the image looks after reconstruction.

X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.442758627	-0.4225221	-0.36429403	-0.623359416	-0.7090114	-0.6157585	-0.4323511
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.442758627	-0.4225221	-0.36429403	-0.623359416	-0.7090114	-0.6157585	-0.4323511
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.442758627	-0.4225221	-0.36429403	-0.623359416	-0.7090114	-0.6157585	-0.4323511
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.442758627	-0.4225221	-0.36429403	-0.623359416	-0.7090114	-0.6157585	-0.4323511
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.442758627	-0.4225221	-0.36429403	-0.623359416	-0.7090114	-0.6157585	-0.4323511
-0.6048609	-0.6352856	-0.7529194	-0.6427501	-0.54699252	-0.4957419	-0.441112470	-0.4201465	-0.36208572	-0.624709674	-0.7134829	-0.6247654	-0.4450416
0.5476336	0.4146480	0.2466798	-0.4225703	-0.57401439	-0.4407446	-0.392990638	-0.3833132	-0.36371630	-0.730569665	-0.7734560	-0.6800830	-0.4386074
1.3405545	1.1564015	0.9922401	-0.1620609	-0.49785690	-0.3792599	-0.373716435	-0.3821489	-0.40652471	-0.781457974	-0.7347916	-0.6096159	-0.3229128
1.7183426	1.4916277	1.3530434	-0.0540572	-0.44703767	-0.2977867	-0.327356425	-0.3540902	-0.42160598	-0.918909389	-0.8006449	-0.6349312	-0.2336473
1.6943396	1.4996481	1.2421715	-0.1505307	-0.61437851	-0.4939950	-0.433005972	-0.4179069	-0.39645565	-0.617568926	-0.6570118	-0.5993732	-0.4774271
1.5558440	1.3890948	1.0626934	-0.2431010	-0.72097520	-0.6195837	-0.500624328	-0.4585487	-0.37856924	-0.431944225	-0.5752465	-0.5844516	-0.6340448
1.5798620	1.4441968	1.1246396	-0.1301549	-0.64839389	-0.6271294	-0.518540091	-0.4743401	-0.38638525	-0.274832889	-0.4076037	-0.4296728	-0.5768250
1.2508881	1.4055468	1.0775944	0.5657874	-0.14822160	-0.6831950	-0.550472316	-0.4494151	-0.21883283	1.325212932	1.0837623	0.8181516	-0.3225394
1.3023923	1.5323816	1.2381569	0.8645526	0.06421055	-0.6632490	-0.565607408	-0.4638937	-0.22385646	1.701572890	1.5020402	1.2116947	-0.1450926
1.0922526	1.3794699	1.3853598	1.3612551	0.65300398	-0.3081666	-0.451542797	-0.4348867	-0.35808869	1.503911873	1.6897694	1.6171456	0.5731011
0.3905018	0.7791963	1.3940030	2.1718257	1.73478171	0.3470158	-0.283296553	-0.4417913	-0.69909936	0.862160380	1.8272265	2.2374705	1.9534559
-0.4492012	-0.1154588	0.8718181	2.1325264	2.13816448	0.8704012	-0.004993958	-0.2790065	-0.77483069	-0.147331281	1.2283987	1.9691670	2.5769208
-0.6207145	-0.3757007	0.6651227	1.9782084	2.28846752	1.5023073	0.724368950	0.4517613	-0.08022675	-0.005403491	1.2245315	1.8057466	2.5763357
-0.9049414	-0.8120450	0.1237109	1.4949165	2.33832781	2.6120066	2.229506659	2.0384536	1.58743988	0.924484113	1.5242096	1.5252731	2.1294163
-0.7747248	-0.8636943	-0.8457336	-0.1472938	0.76679925	2.2288035	2.781999940	2.8970959	2.99817163	2.355380738	1.4402178	0.4240476	-0.1618417
-0.5475356	-0.7546981	-0.8914260	-0.6249043	0.27778595	1.9939113	2.649410157	2.7877895	2.92665955	1.884483031	0.8284913	-0.1947085	-0.6137113
-0.5680083	-0.7144890	-0.8965326	-0.7487676	-0.18840910	0.8811843	1.328318585	1.4321306	1.56524095	0.832475136	0.1127152	-0.4777515	-0.6816771
-0.7275113	-0.7473491	-0.8536929	-0.6584437	-0.53411468	-0.4933435	-0.442758627	-0.4225221	-0.36429403	-0.623359416	-0.7090114	-0.6157585	-0.4323511

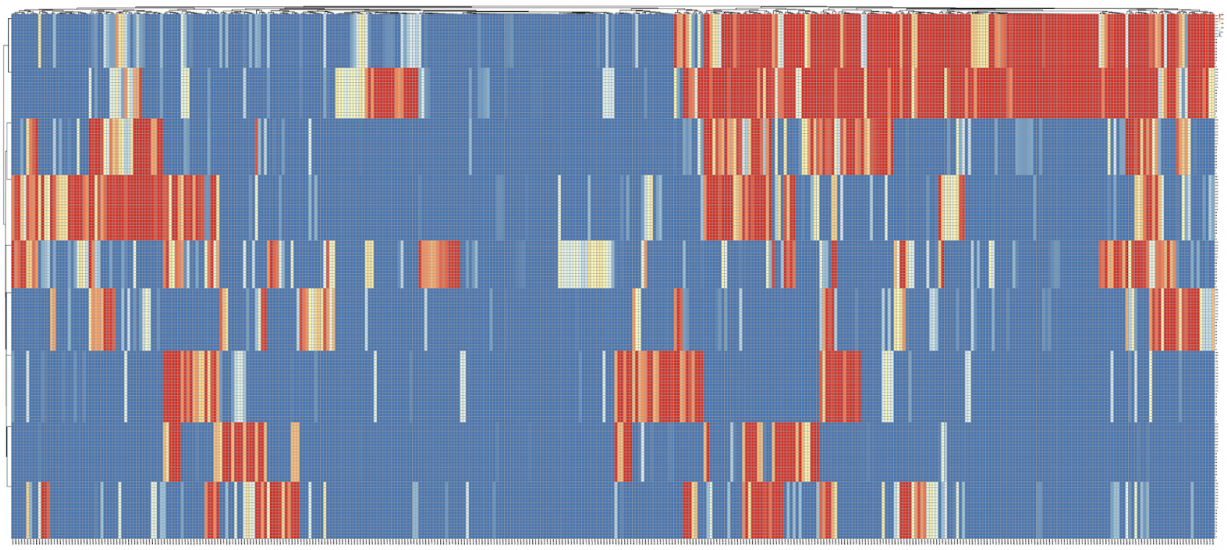


2) Draw a tree of the pixels, and see if you can explain the results based on geometry of the pixels (how far apart are they in the 2-d space).
Try to Explain the PCA results in light of this.

Here is the tree of rows in the image and how they relate to each other.



This tree only represents the rows in image 3 reconstructed.
What I can see is that in the center of the dendrogram we have the top rows 1,2,3,4,5,6,7 being clustered together with the bottom rows 23,24,25,26, 27,28 so this shows they have about the same distance values.
Next we see that the bottom middle rows are clustered together on the left of the dendrogram which are rows 16,17,18,19,20,21,22.
Lastly, we see on the right of the dendrogram the top middle rows are clustered together and the rows are 8,9,10,11,12,13,14,15.
We can then repeat this process by transposing the matrix so the columns become the rows and the rows become the columns. This will then give us a tree of the columns. Combining the data from the rows and columns we would be able to understand how each pixel is clustered in relation to the others. Only based on the row data we can see that there are about 3 to 4 bands that are clustered together, so this represents the 4 pc's we used to reconstruct the image.



3) Can you use some of the tools you have learnt to build a classifier,
so if you get a new set of pixels you can predict what is in the
picture. This is a the start of a real project, but you don't have all the tools
(such as neural networks) which might be more suited for this task.
Split your dataset into two (a training set and a test set),
build your classifier and figure out
how well it does with the test data in predicting the digits.
Define the sensitivity and specificity of your classifier. How well does it recognize your own handwriting (make sure your handwriting is not in the training set)

To build a classifier we thought one way to do so would be using the pca result to see how many pc's are needed to reconstruct each digit. Then we would compare that number of pc's to the number of pc's it takes to reconstruct the average pixels of all of the single digits.

So first, we took all the pixel data for each digit and averaged the rows into one row.

Next, we made a matrix of each of the 10 digits and ran pca to find how many pc's made up 90% of the variance for each.

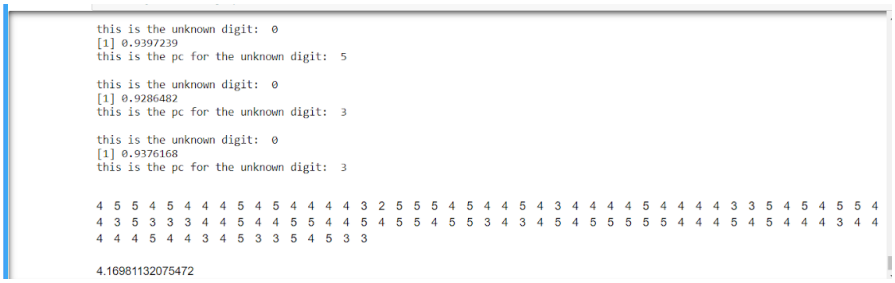
```
[1] 0.9483771
[1] 0.9238257
[1] 0.9710878
[1] 0.9446342
[1] 0.9704346
[1] 0.9395174
[1] 0.967781
[1] 0.9023977
[1] 0.9554669
[1] 0.9493153
the pc for 0 is : 4
the pc for 1 is : 2
the pc for 2 is : 3
the pc for 3 is : 2
the pc for 4 is : 3
the pc for 5 is : 2
the pc for 6 is : 3
the pc for 7 is : 2
the pc for 8 is : 3
the pc for 9 is : 2
```

The decimal numbers represent how much variance each average digit contains after pca and you can see the pc's required to recreate each digit. This is the training data

Next, we thought of finding how to get test data. So we will get a random digit and run pca on that digit, if the number of pc's required to recreate that number is similar to any of the pc's above we can assume the unknown digit corresponds to that digit.

This classifier does not perform well because the digit 0 is the only digit with a unique pc count. One thing that this classifier can do is at least tell us if we have an even or odd unknown number. The odd numbers seem to require 2 pc's while the even numbers require 3 pc's for reconstruction at 90% variance.

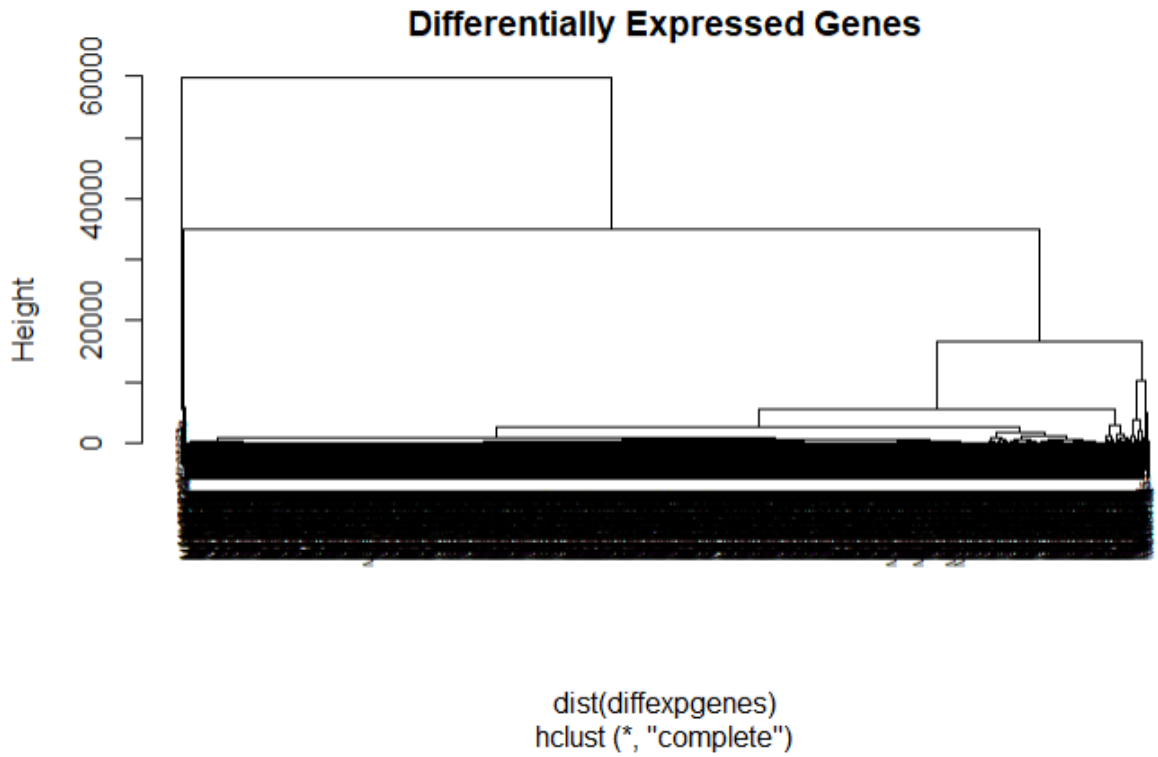
When looking at sensitivity vs specificity, we will have to adjust the sensitivity to require multiple unknown numbers, of the same digit, to be fed into the function so we can take an average of their pc's to compare it to the pc's in the training results.
The specificity would be choosing to not include any test digits that return a pc above 5.



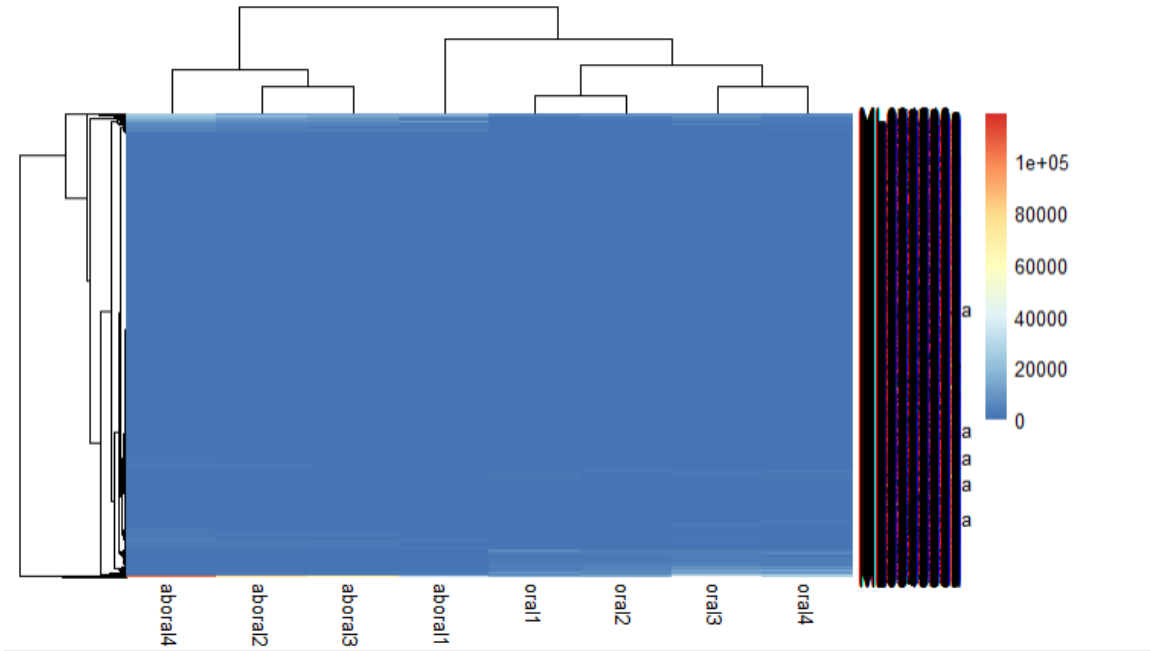
So here you can see we took 120 random samples of digit 0 and found the pc's for each. Next, we used a for loop to remove any pc's that were above 5 then took the average of the remaining pc's which turned out to be 4.1. This means our random number should be 0. This doesn't work too well with other digits, unless we change the sensitivity or specificity.

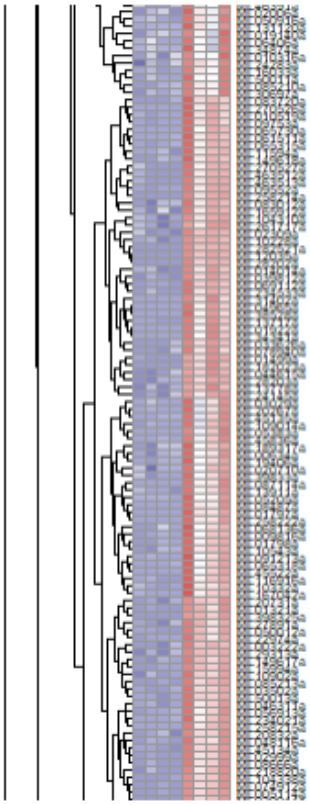
Project 3.
==========

1) Build hierarchical trees based on the columns and for the rows (exclude rows that are "low" expression)



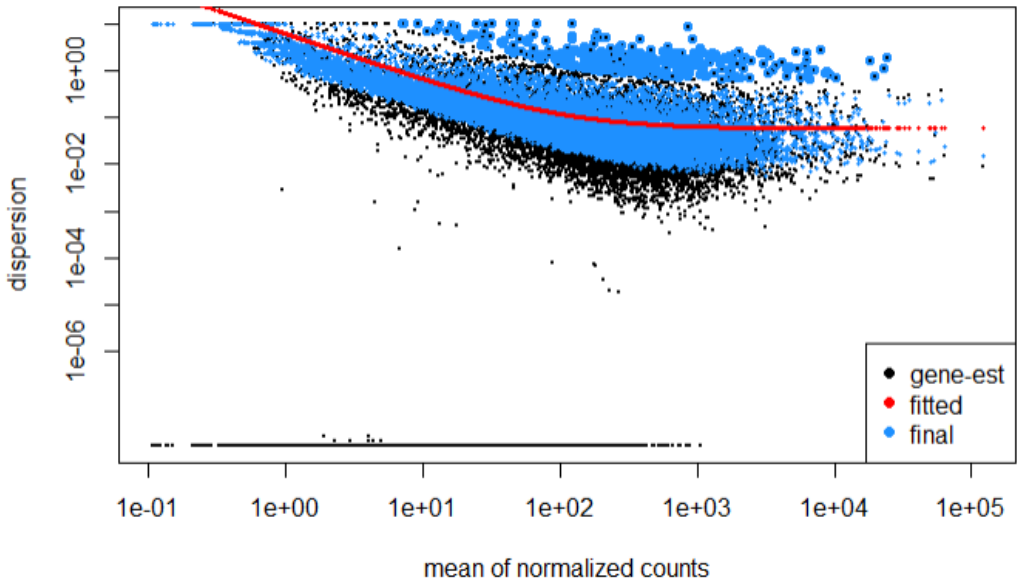
2) Draw a heat map of the expression data





3) Use DESeq2 to analyze this data

a) which are the most significantly changing genes in this dataset?



There are around 1574 genes that are differentially expressed in this dataset. This is determined using the criteria in which genes have an adjusted p-value of less than 0.05 and log2FoldChange value greater than 1 or less than -1 after performing differential expression using DeSeq. They are listed in the R script under Step 7. Several of them are listed here (first 100): "ML000125a" "ML000132a" "ML00016a" "ML000715a" "ML00083a" "ML00085a" "ML001116a" "ML001117a" "ML00114a" "ML001527a" "ML00154a" "ML00173a" "ML00191a" "ML00218a" "ML002221a" "ML002223a" "ML00229a" "ML00283a" "ML003015a" "ML003020a" "ML003022a" "ML003220a" "ML003222a" "ML003248a" "ML00342a" "ML00368a" "ML00372a" "ML00413a" "ML004415a" "ML004421a" "ML004510a" "ML004922a" "ML004923a" "ML00497a" "ML005030a" "ML00504a" "ML005114a" "ML005116a" "ML005119a" "ML00515a" "ML005211a" "ML005212a" "ML005214a" "ML005219a" "ML005324a" "ML005329a" "ML00576a" "ML006114a" "ML00613a" "ML00621a" "ML006313a" "ML00634a" "ML00653a" "ML00679a" "ML006910a" "ML006927a" "ML00692a" "ML006934a" "ML00694a" "ML00697a" "ML00699a" "ML00702a" "ML00704a" "ML00705a" "ML00732a" "ML007335a" "ML007414a" "ML007426a" "ML007430a" "ML00744a" "ML008015a" "ML008017a" "ML008029a" "ML008323a" "ML008325a" "ML008327a" "ML008715a" "ML00892a" "ML00906a" "ML009149a" "ML00918a" "ML009410a" "ML009411a" "ML00948a" "ML00949a" "ML009710a" "ML009811a" "ML009816a" "ML010112a" "ML010316a" "ML010519a" "ML01053a" "ML010623a" "ML01167a" "ML01248a" "ML013112a" "ML01321a" "ML01323a" "ML01348a" "ML01357a" "ML01358a" "ML01359a" "ML014014a" "ML01412a" "ML01433a" "ML01438a" "ML014423a" "ML01466a"

b) which genes are most consistently highly expressed in these datasets they are the "house-keeping" genes

These are the most consistently highly expressed genes in the dataset for each category and their respective expression values:

Aboral1:
#ML174735a ML034334a ML034337a ML034336a ML01482a ML46651a
#17157.06 21911.36 22424.14 23631.30 30581.63 118211.07

Aboral2:
#ML174735a ML034337a ML034336a ML034334a ML01482a ML46651a
#40418.84 58540.29 65914.50 68219.38 80559.11 93870.29

Aboral3:
#ML174735a ML034337a ML034336a ML034334a ML46651a ML01482a
#27488.06 47004.61 50998.50 55714.11 56425.56 71249.61

Aboral4:
#ML174735a ML034337a ML034336a ML46651a ML034334a ML01482a
#37089.43 67521.80 82850.63 91834.25 92639.95 110042.52

Amongst the Aboral category, there are several genes that are consistently highly expressed. These genes include: ML174735a, ML034337a, ML034336a, ML034334a, ML46651a, ML01482a. This was performed by sorting the expression values for each category and extracting the top 5 genes for each category. It appears that each Aboral category (Aboral1-Aboral4) have the same 5 consistent highly expressed genes, but their orders are slightly different for each category.

Oral1:
#ML01433a ML087114a ML199832a ML004510a ML21583a ML306119a
#14912.24 17974.26 18039.35 22064.90 23649.09 31192.30

Oral2:
#ML11032a ML306119a ML34341a ML087114a ML199832a ML004510a
#16119.59 16538.89 18610.45 20852.11 21035.21 34514.43

Oral3:
#ML46651a ML43587a ML11032a ML087114a ML01482a ML004510a
#27853.26 29726.57 30124.16 42894.61 44780.12 66495.13

Oral4:
#ML01433a ML11032a ML09436a ML087114a ML46651a ML004510a

#26045.23 28540.28 33828.03 39617.47 64936.93 77464.55

Amongst the Oral category, there are several genes that are consistently highly expressed. These genes include: ML01433a, ML11032a,ML199832a, ML004510a, ML087114a, ML46651a, and ML306119a. The only genes that appear in every Oral category are genes ML087114a and ML004510a. Unlike the Aboral categories, in which the top five genes are the same for each Aboral category, there is more variety in the top highly expressed genes for the Oral categories.

c) How consistent are these results with the analysis you did in the midterm project ?

The top 5 genes identified in the midterm for the Aboral category using the p-value are ML01482a, ML03658a, ML034334a, ML034336a, ML46651a. On the other hand, the top five genes identified in the final project are: ML174735a, ML034337a, ML034336a, ML034334a, ML46651a, ML01482a. As we can see, the analysis performed does not appear to be consistent with previous results, due to the fact that the highly expressed genes in Aboral differ. The only gene that is consistent within both midterm and final projects is the gene ML034336a. This may be due to differences in the way that we calculated the adjusted p-value for each gene, compared to the method used by DeSeq.

Additionally, the top 5 genes identified for the oral category using the p-value are ML165814a, ML050913a, ML06904a, ML174758a, and ML210018a. On the other hand, the top five genes identified in the final project are: ML01433a, ML11032a,ML199832a, ML004510a, ML087114a, ML46651a, and ML306119a. As we can see, the analysis performed does not appear to be consistent with previous results, due to the fact that the highly expressed genes in Oral differ. This may be due to differences in the way that we calculated the adjusted p-value for each gene, compared to the method used by DeSeq.

d) What else can you say about the data in terms of consistency, and the results that you find from your analyses. The question is open-ended, think of this as your experiment, you need to write a paper based on this data so you have to figure out what kind of "story" you can tell based on this.

In terms of consistency, we can see that the most highly expressed genes in each category remain relatively similar across the dataset in the final project. This is observed in the Aboral and Oral dataset, in which the most highly expressed genes are consistent throughout the entire dataset. However, when compared to the midterm results, the top genes do not appear to be consistent with one another. Both the midterm and final projects demonstrated consistency amongst the top five genes expressed in each Aboral and Oral category, but only within each project. In contrast, both the midterm and final results differ from one another. In order to determine the top genes, we calculated the p-value for each of the genes' expression value. However, in the midterm, the p-values were calculated by hand, whereas in the final, the p-values were calculated using DeSeq2. Therefore, to maintain consistent results, it is recommended that a consistent technique be used. We can tell that the genes remain relatively

similar to one another, so when writing a paper, there is some form of consistency, we just need to determine which technique is the most accurate.

e) what is the most interesting pathway or gene that is responding in this study ?

ML46651a is the most interesting pathway responding in this study. This is due to the fact that the ML46651a gene is the most highly expressed in each Aboral and Oral category, therefore making it a consistently expressed gene throughout the entire dataset. This gene is a cellular component that is part of the membrane attack complex.