Midterm Project

dm5153

Applied Biostatistics for Bioinformatics

Devin McAvoy (dm5153), Mudra Patel (mp6092), Jessica Schilter (js12845), Brandon Thong (bt1194)

November 14 2022

Dataset 1

```
#install packages if needed
#install.packages('ggplot2')
#install.packages('reshape2')
#install.packages('png')

#set working directory to location of file to avoid hunting down filepaths
#library(rstudioapi)
#current_path = rstudioapi::getActiveDocumentContext()$path
#setwd(dirname(current_path ))
```

```
library(readxl)
train <- read.csv("train.csv")
```

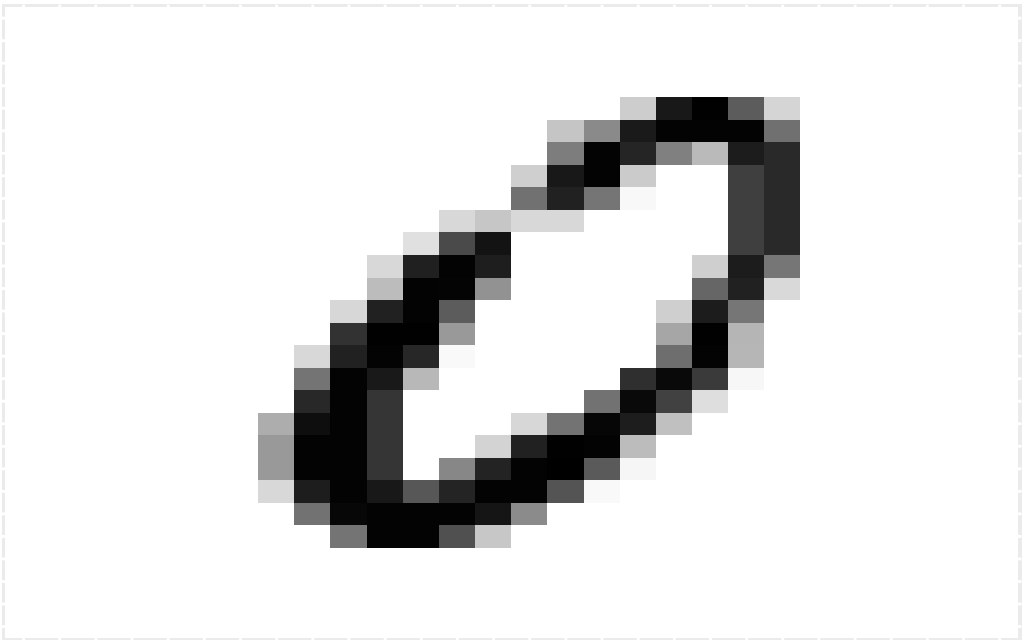
Part 1

```
#Dataset 1 Part 1
#function to draw the digit represented by each row
#input the train.csv data set and the row you want to draw
draw_digit<-function(data,row){
  #import the relevant libraries
```

```
  library(ggplot2)
  library(reshape2)

  #intialize the matrix with the first 28 pixels
  pixel_grid<-data[row,2:29]
  #rename the columns
  colnames(pixel_grid) <- paste("Col", 1:28)


  #put every 28 entries into a new row, starting at second row
  for(x in 1:27){
    #define first pixel in the row
    start<-x*28+2
    #define last pixel in the row
    end<-start+27
    #hold the data from those pixels temporarily
    temp_row<-data[row,start:end]
    #make the column names match the full matrix
    colnames(temp_row) <- paste("Col", 1:28)
    #add the temp row to the full matrix
    pixel_grid<-rbind(pixel_grid,temp_row)
  }
  #flip the matrix
  pixel_grid<-pixel_grid[nrow(pixel_grid):1,]
  #name the rows
  rownames(pixel_grid) <- paste("Row", 1:28)
  #melt the data so ggplot can interpret it
  #also transpose at this point
  m<-melt(as.matrix(t(pixel_grid)))
  #give column names to the melted data
  colnames(m) <- c("x", "y", "value")
  #define the theme for the heatmap - remove axis etc
  theme<-theme(legend.position="none",axis.title.x=element_blank(),axis.text.x=element_bla
  #plot the data as a greyscale heatmap
  ggplot(m, aes(x=x,y=y,fill=value))+scale_fill_gradient(limits = c(0, 255), low = 'white'
}
```

```
#call the function on a row of your choice
draw_digit(train, 897)
```

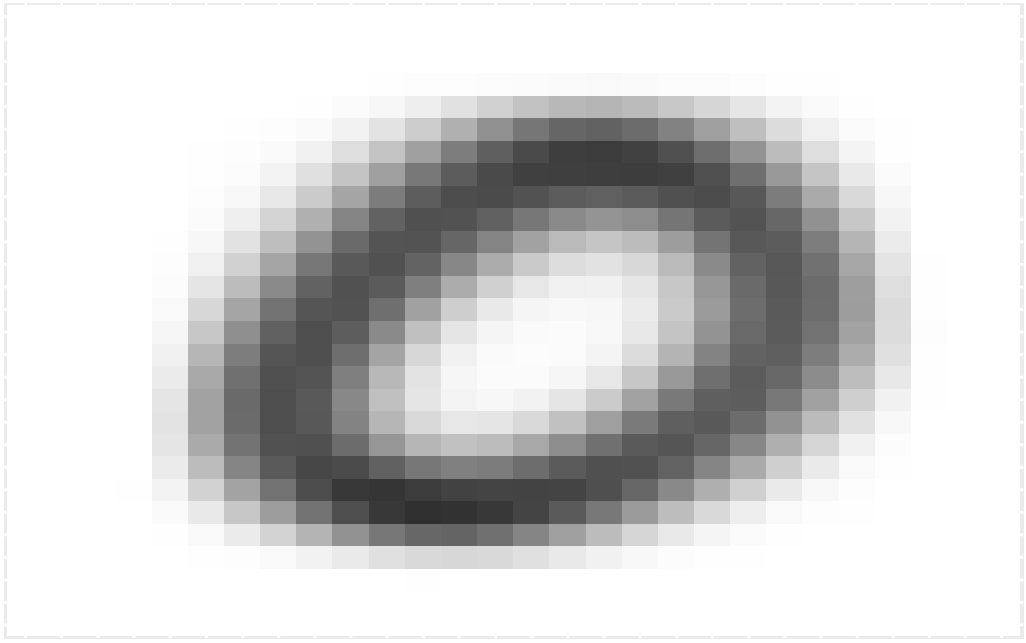
```
Warning: package 'reshape2' was built under R version 4.2.2
```



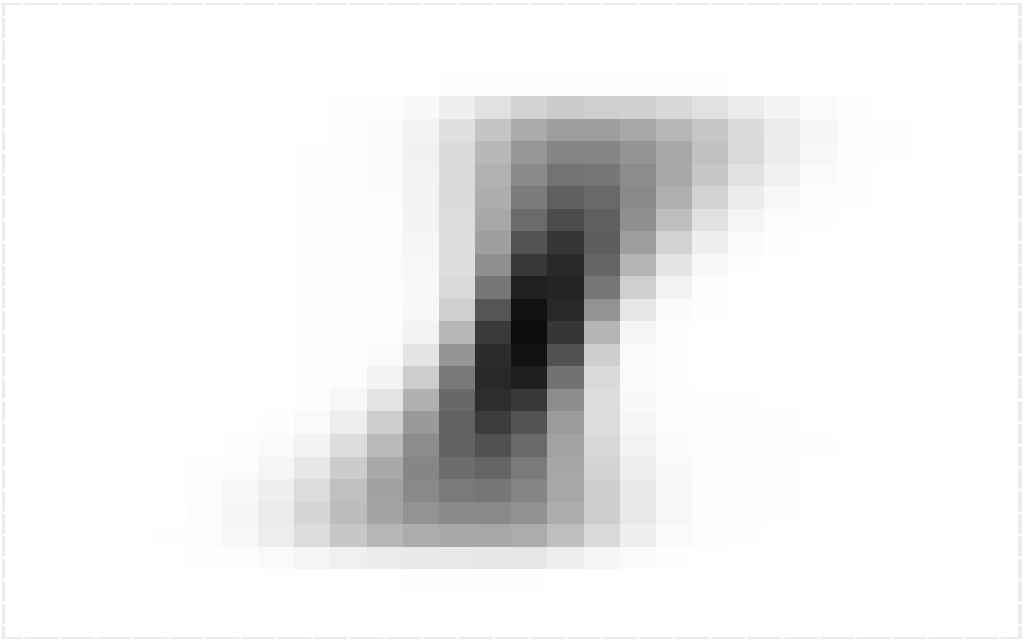
Part 2

```
#Dataset 1 Part 2
#create empty dataframe for the averages
digit_averages<-train[FALSE,]
#loop to get the averages for each digit 0-9
for(x in 0:9){
  #subset the data for the digit
  digit_subset<- train[which(train[,1]==x),]
  #average the columns
  digit_subset<-colMeans(digit_subset)
  #add it to the dataset of averages
  digit_averages<-rbind(digit_averages,digit_subset)
}
#rename the columns to the digit they represent, otherwise the labels start at 1 instead o
row.names(digit_averages)<-0:9
#call the function on the average data for the digit of your choice
draw_digit(digit_averages,"5")
```

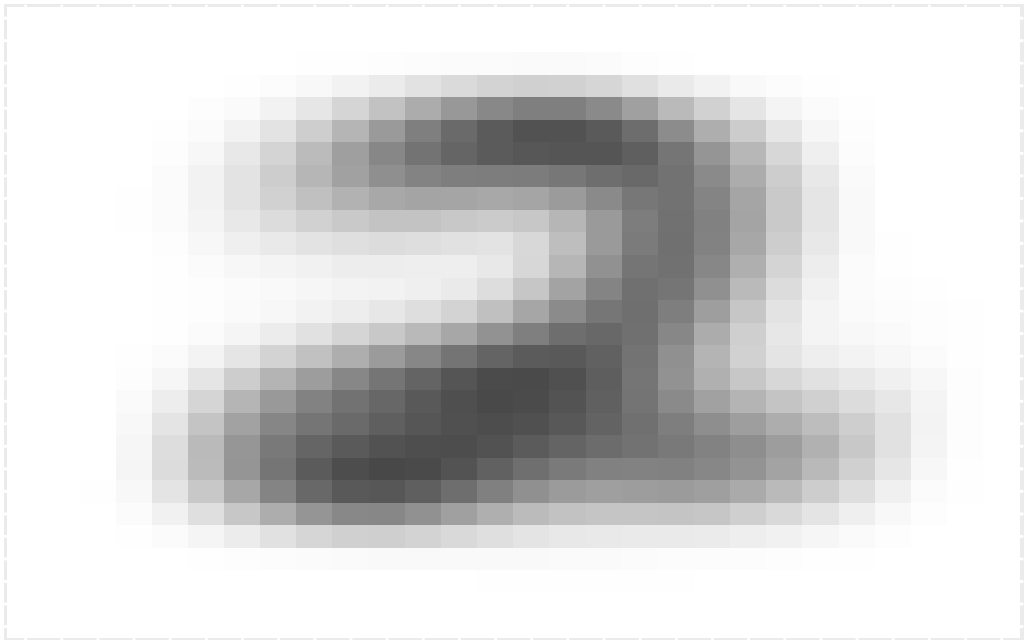
```
#draw all digit averages
draw_digit(digit_averages,"0")
```



```
draw_digit(digit_averages,"1")
```

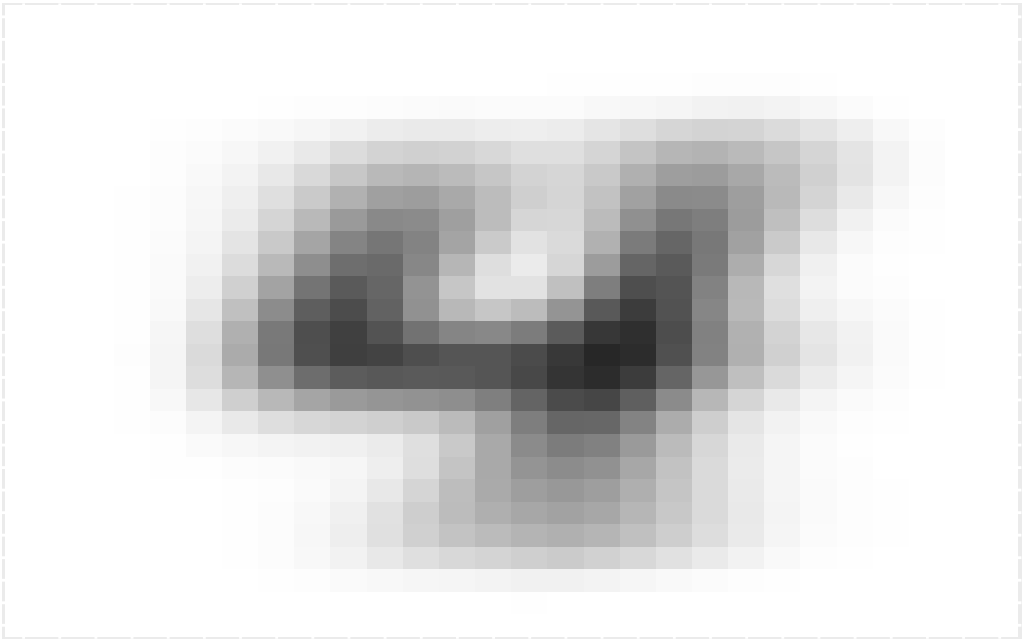


```
draw_digit(digit_averages,"2")
```



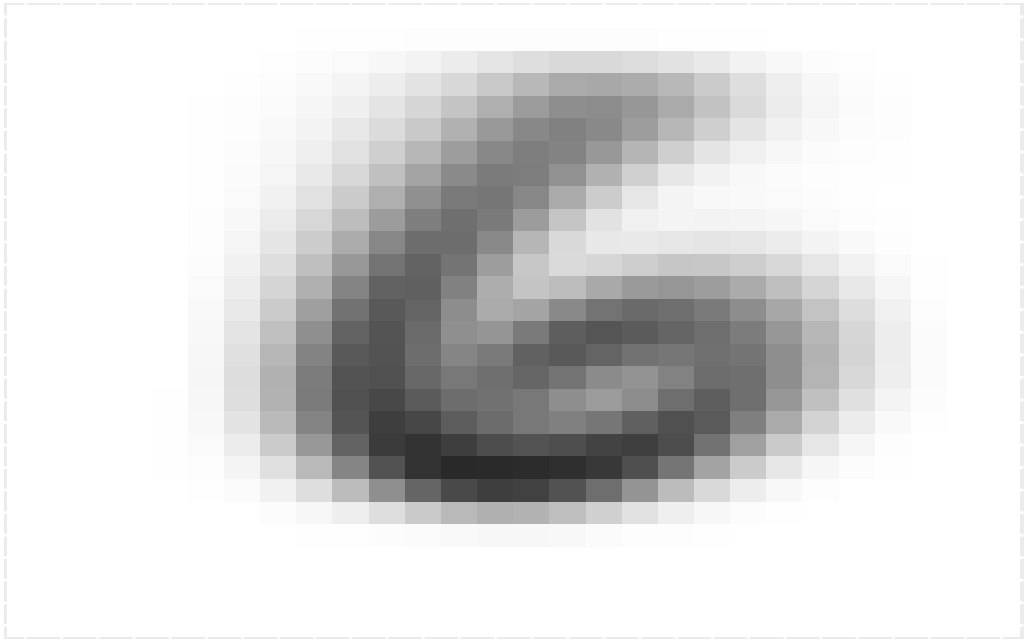
```
draw_digit(digit_averages,"3")
```

```
draw_digit(digit_averages,"4")
```

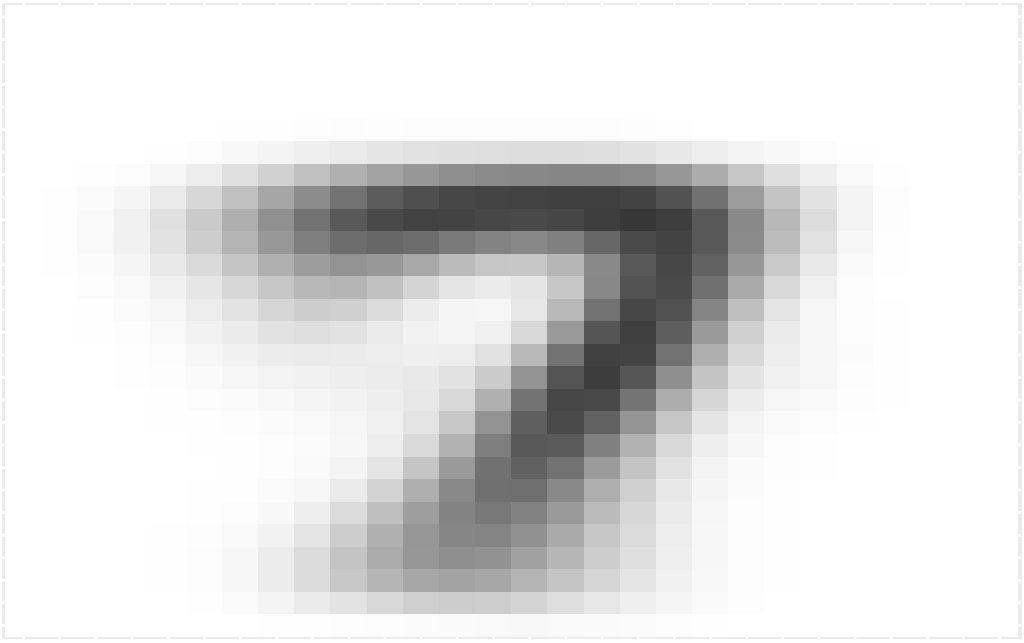


```
draw_digit(digit_averages,"5")
```

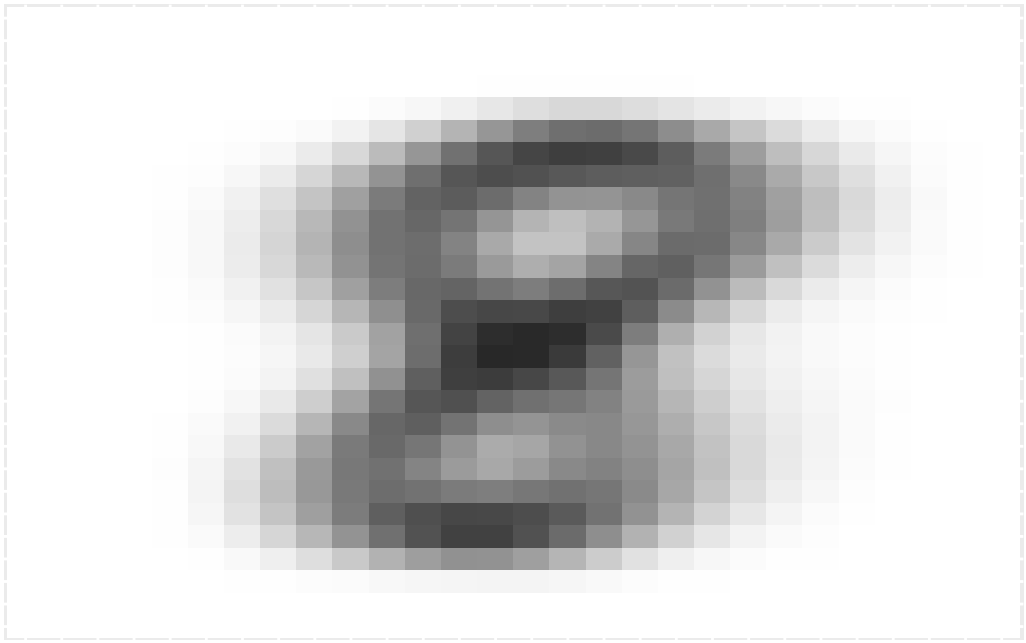
```
draw_digit(digit_averages,"6")
```



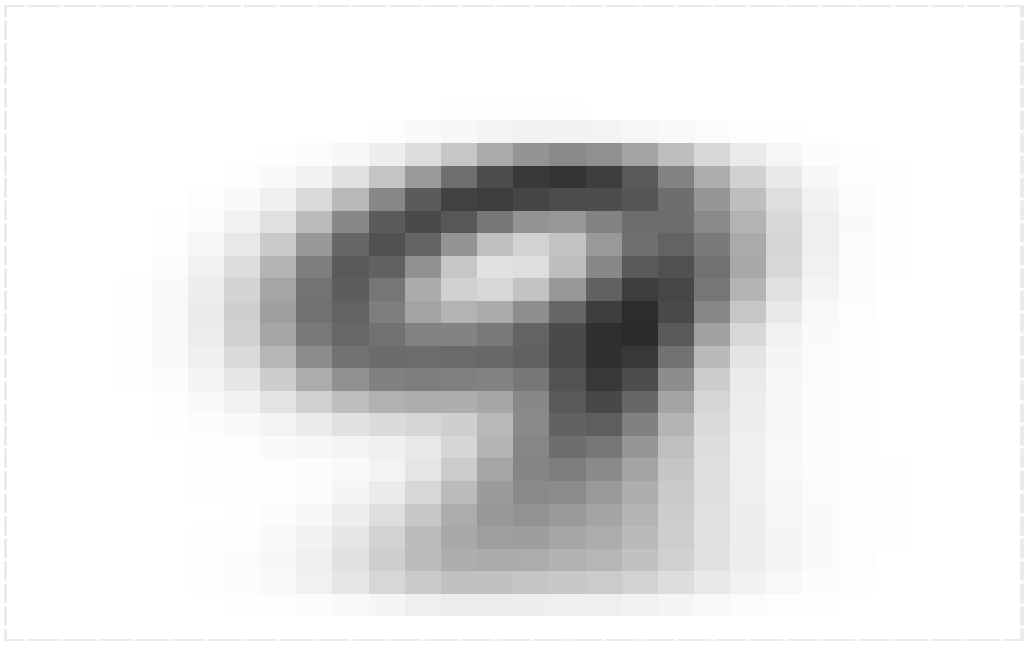
```
draw_digit(digit_averages,"7")
```



```
draw_digit(digit_averages,"8")
```



```
draw_digit(digit_averages,"9")
```

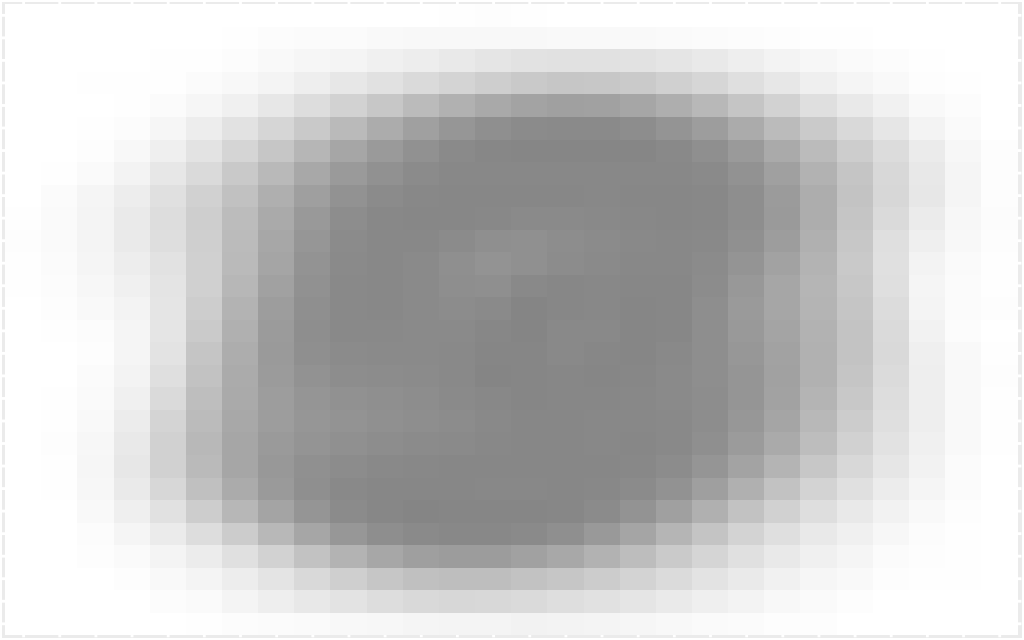


Part 2 b.

Visually, it appears that the digit zero maintains its appearance the best when averaged

Part 3 a.

```
#Dataset 1 Part 3

#create empty dataframe
col_vars<-train[FALSE,]
#calculate standard deviation of every row
temp<-sapply(train, sd)
#add sd to dataframe
col_vars<-rbind(col_vars,temp)
#rename columns
colnames(col_vars)<-colnames(train)
#visualize pixels with highest sd
draw_digit(col_vars,1)
```



```r
#sort the variances
sorted_var<- temp[order(temp, decreasing = TRUE)]
#round the data
sorted_var<-round(sorted_var,2)
#view the top 50 pixels by highest variance
head(data.frame(sorted_var), 10)
```

```
         sorted_var
pixel406     113.85
pixel378     113.71
pixel627     113.00
pixel461     112.92
pixel434     112.75
pixel433     112.75
pixel462     112.69
pixel437     112.58
pixel628     112.52
pixel409     112.44
```

```r
#create empty dataframe for the variances
digit_variance<-train[FALSE,]

#loop to get the averages for each digit 0-9
for(x in 0:9){
  #subset the data for the digit
  digit_subset<- train[which(train[,1]==x),]
  #average the columns
  digit_subset<-sapply(digit_subset, sd)
  #add it to the dataset of averages
  digit_variance<-rbind(digit_variance,digit_subset)
}
#rename the columns to the digit they represent, otherwise the labels start at 1 instead o
row.names(digit_variance)<-0:9
colnames(digit_variance)<-colnames(train)
```

```r
#function to output the top 10 pixels by variance for the selected digit
sort_digit_variance<-function(data, digit){
  data<-t(data)
  data<-data[,digit]
  data<-data[order(data, decreasing = TRUE)]
```

```
  data<-data.frame(data)
  colnames(data)<-digit-1
  head(data,10)
}
```

Using the function above, we can determine the 10 pixels with the highest variance for each digit.

```
#call the function for each digit
for(x in 1:10){
  print(sort_digit_variance(digit_variance,x))
}
```

```
               0
pixel266 113.3277
pixel538 112.9421
pixel454 112.6389
pixel293 112.6150
pixel426 112.5556
pixel347 112.5233
pixel494 112.3957
pixel566 112.2343
pixel320 112.2313
pixel482 112.0948
               1
pixel571 114.0052
pixel543 113.8370
pixel602 113.7203
pixel574 113.3381
pixel599 112.9476
pixel241 112.2655
pixel269 112.1574
pixel238 112.0400
pixel630 111.9536
pixel210 111.6802
               2
pixel465 114.2430
pixel493 113.3505
pixel539 112.4153
pixel567 112.2181
pixel511 111.9299
pixel437 111.9000
```

```
pixel410 111.5553
pixel178 111.5458
pixel210 111.5385
pixel494 111.5150
               3
pixel186 111.9796
pixel243 111.6128
pixel215 111.3124
pixel596 111.1593
pixel298 111.0652
pixel270 110.9701
pixel624 110.8321
pixel403 110.6496
pixel575 110.5384
pixel271 110.5236
               4
pixel427 112.3797
pixel399 111.7711
pixel455 110.7034
pixel456 110.4689
pixel575 110.4127
pixel327 110.1315
pixel547 109.9929
pixel546 109.9909
pixel371 109.9161
pixel291 109.7480
               5
pixel182 112.0824
pixel208 112.0426
pixel183 111.8436
pixel632 111.7946
pixel186 111.6860
pixel184 111.6497
pixel658 111.5736
pixel187 111.4100
pixel181 111.3150
pixel185 111.2709
               6
pixel238 111.7826
pixel348 111.7222
pixel265 111.4131
pixel183 111.2268
pixel210 111.2058
```

```
pixel518 111.0582
pixel383 110.9986
pixel156 110.9319
pixel512 110.8930
pixel484 110.8449
               7
pixel600 112.3203
pixel272 112.0919
pixel628 112.0283
pixel545 111.7212
pixel630 111.6314
pixel300 111.6163
pixel656 111.6080
pixel657 111.4368
pixel288 111.0707
pixel573 111.0473
               8
pixel625 110.7727
pixel654 110.2433
pixel234 110.1027
pixel346 109.9646
pixel289 109.9295
pixel541 109.9064
pixel633 109.7198
pixel653 109.6218
pixel605 109.5590
pixel513 109.5211
               9
pixel574 111.5601
pixel546 111.3095
pixel602 111.0305
pixel629 109.9468
pixel355 109.7818
pixel458 109.7436
pixel630 109.6380
pixel601 109.2579
pixel575 109.1652
pixel400 108.9816
```

Using the digit variance data, we can also visualize how much variance there is per column in each digit.

```
#visualize the variance on any arbitrarily selected digit
draw_digit(digit_variance,"6")
```

Part 3 b.

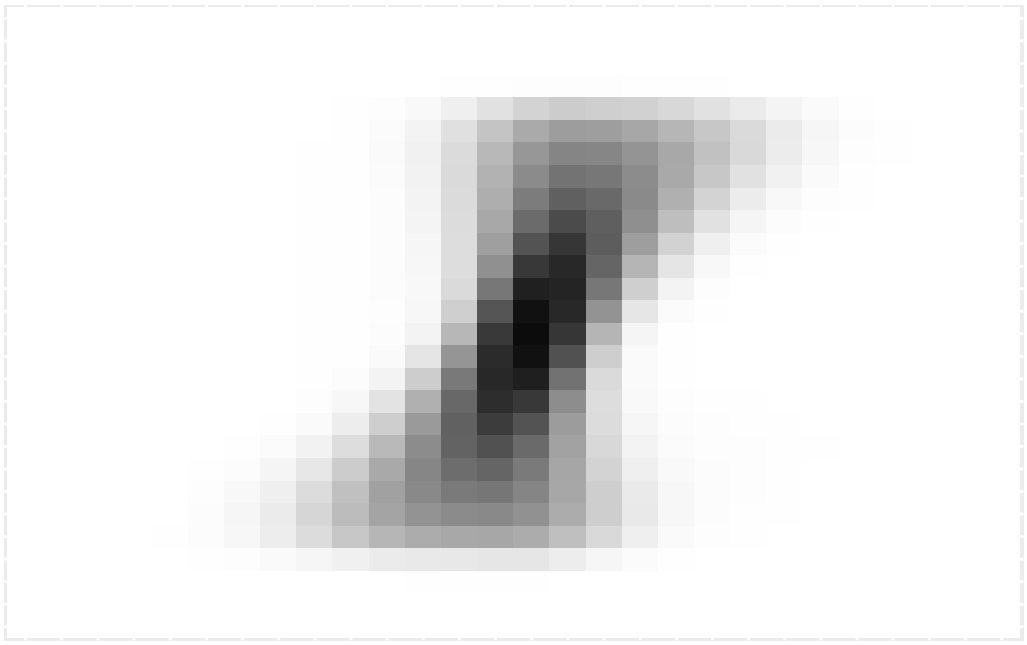
We selected 0 as the digit that looks the best when each pixel is averaged. However, the digit 1 is the digit with the lowest average variance. Comparatively, the visual representation of the average 1 is fairly blurry.

```
#create new object holding variance data
digit_variance_nonzero<-digit_variance
#replace zero variance pixels with NA
digit_variance_nonzero[digit_variance_nonzero==0]<-NA
#calculate average variance of each digit across all pixels, excluding zero variance pixel
average_digit_var<-rowMeans(digit_variance_nonzero,na.rm=TRUE)
#print the average variance for each digit
print(data.frame(average_digit_var))
```

```
  average_digit_var
0          63.87465
1          34.44707
```

```
2          60.73058
3          59.06498
4          54.55053
5          60.69049
6          59.79107
7          52.14659
8          60.85276
9          55.08298
```

```r
#visualize the average digit 1 for comparison
draw_digit(digit_averages,"1")
```

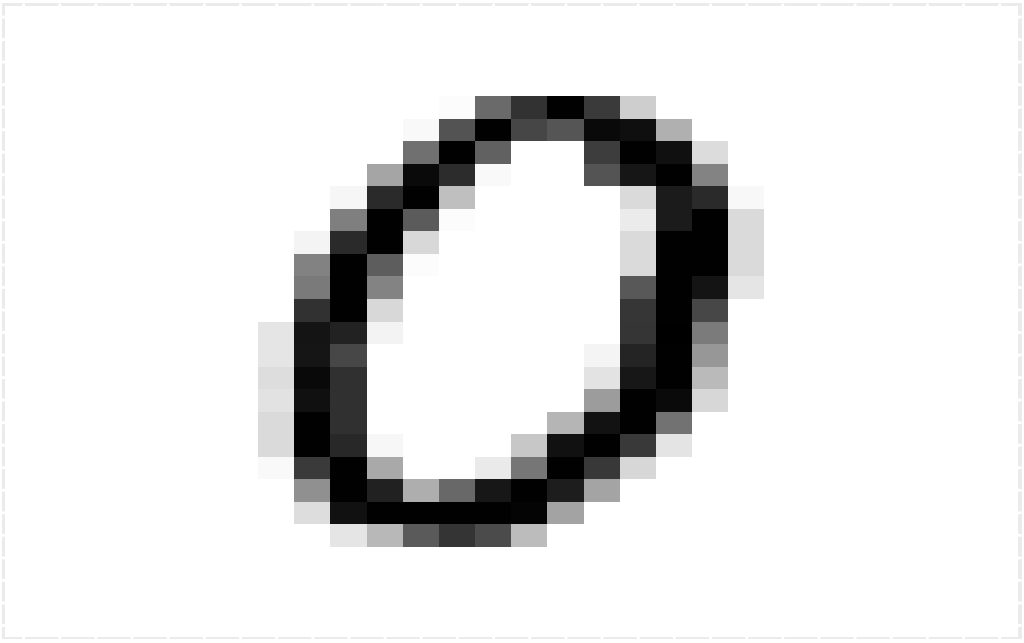


Part 3 c.

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

Replacing the lowest variability pixels with their average value has a minimal impact on the visualization, because columns with low variability had values all close to the average anyway, so the change is fairly negligible.

```r
col_means<-colMeans(train)
train_replace<-train

for(x in 2:785){
  if(col_vars[1,x]<5){
    train_replace[x]<-col_means[x]
  }
}

draw_digit(train_replace,6)
```



Part 3 d.

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

Columns with averages close to 0 tend to be near the edges of the image and are white because there is usually nothing drawn in that space. Columns with average closer to 255 are closer to the center, and where the average digit almost always has some part of the digit included in that pixel. However, the maximum column average is only about 140, so none of the averages are particularly close to 255.

```r
#number of columns with near zero average
near_zero<-sum(col_means[]<5)
print(near_zero)
```

```
[1] 369
```

```r
#number of columns with near 255 average
near_top<-sum(col_means[]>250)
print(near_top)
```

```
[1] 0
```

```r
#highest average column value
round(max(col_means),1)
```

```
[1] 139.8
```

Part 4

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.

```r
#install relevant packages (if not done above) and declare functions

#install.packages('ggplot2')
#install.packages('reshape2')
#install.packages('png')
## average over a small square (fac x fac)
ave_by_fac <- function(i1,fac,ii,jj){
  ave=0;
  cnt=0;
  for(i in c(1:fac)){
    for(j in c(1:fac)){
      cnt = cnt +1;
      x = (ii-1)*fac+i;
      y = (jj-1)*fac+j;
      ##         cat("i,j,ii,jj,x,y=",i,j,ii,jj,x,y,"\n");
```

```
      ave = ave+     i1[x,y];
    }}
  ave = ave/cnt;
  return(ave);
}

## function I wrote to scale down a square image to a 28 x 28 image
## uses the averaging function above
scale_down_image <- function(img_in) {
  ## fac is the factor by which you have to scale the image to become a
  ## 28 x 28 square
  fac <- as.integer(dim(img_in)[1]/28);
  im_out <- matrix(0,nrow=28,ncol=28);
  for(i in c(1:28)){
    for(j in c(1:28)){
      im_out[i,j] = ave_by_fac(img_in,fac,i,j);
    }}
  return(im_out);
}
```

```
#Get data
library(png)
library(vctrs)
```

```
Warning: package 'vctrs' was built under R version 4.2.2
```

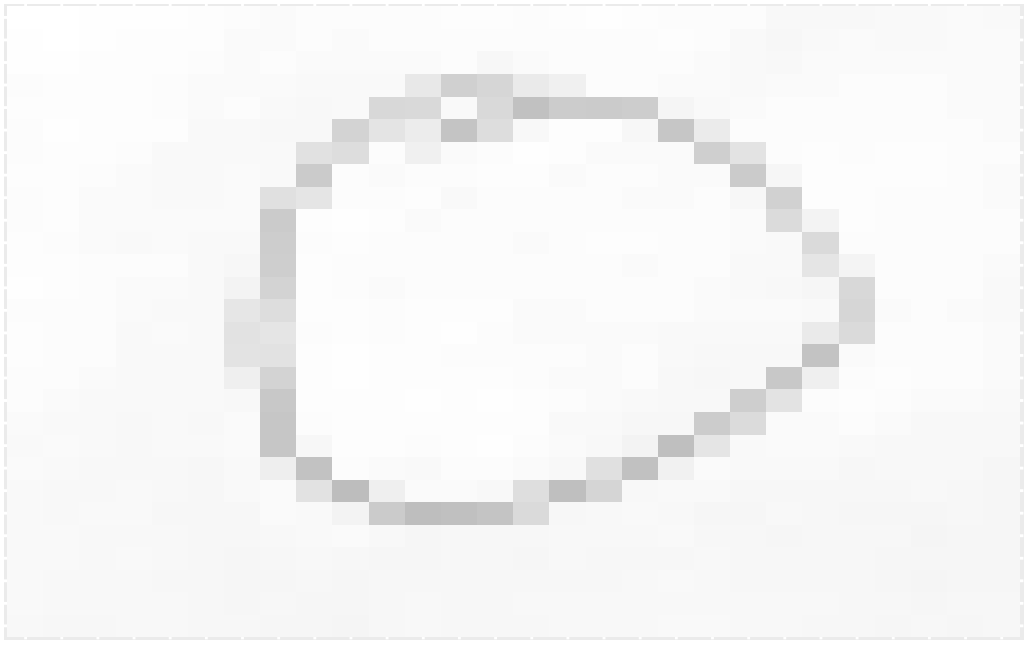
```
library(ggplot2)
library(reshape2)

#function to take png image and convert it to same format as train.csv data
print_HW_digit<-function(img, label){

  #apply image scaling function
  img_scaled<-scale_down_image(img[,,2])

  #rescale values in the data to match given data, 0=white, 255=black
  img_scaled<-abs(img_scaled-1)
  img_scaled<-img_scaled-min(img_scaled)
  img_scaled<-img_scaled*255
  img_scaled<-round(img_scaled,0)
```

```
  #transpose data into correct orientation
  img_scaled<-t(img_scaled)

  #create the label as a dataframe
  label<-data.frame(label)

  #melt the image data so it is in long format
  img_m<-melt(img_scaled)
  #select only the values, excluding the x y coordinates
  img_m<-img_m$value
  #convert the linearized data into a data frame and transpose it so it is a row not a col
  img_lin<-data.frame(img_m)
  img_lin<-t(img_lin)
  #put the label in the first column
  img_lab<-cbind(label, img_lin)
  #label the columns and the row
  colnames(img_lab)<-colnames(train)
  rownames(img_lab)<-label
  #return the transformed data
  return(img_lab)
}
```

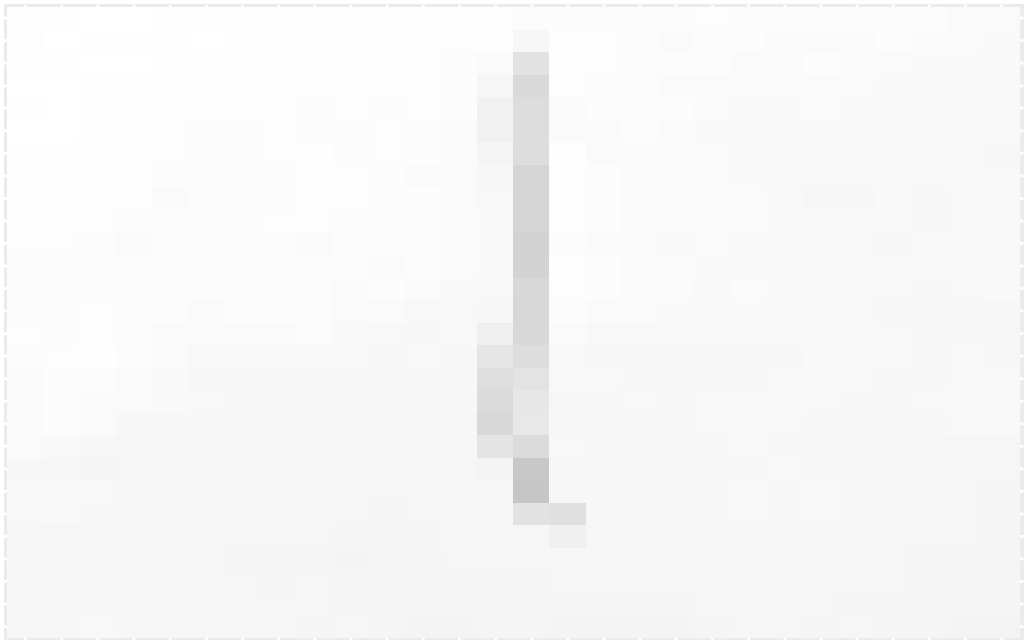
```
#create empty dataframe to store results
HW_digits<-train[FALSE,]
#call the function for each digit and store the results
HW0<-print_HW_digit(readPNG("zero.png"),"HW0")
HW_digits<-rbind(HW_digits,HW0)
HW1<-print_HW_digit(readPNG("one.png"),"HW1")
HW_digits<-rbind(HW_digits,HW1)
HW2<-print_HW_digit(readPNG("two.png"),"HW2")
HW_digits<-rbind(HW_digits,HW2)
HW3<-print_HW_digit(readPNG("three.png"),"HW3")
HW_digits<-rbind(HW_digits,HW3)
HW4<-print_HW_digit(readPNG("four.png"),"HW4")
HW_digits<-rbind(HW_digits,HW4)
HW5<-print_HW_digit(readPNG("five.png"),"HW5")
HW_digits<-rbind(HW_digits,HW5)
HW6<-print_HW_digit(readPNG("six.png"),"HW6")
HW_digits<-rbind(HW_digits,HW6)
HW7<-print_HW_digit(readPNG("seven.png"),"HW7")
HW_digits<-rbind(HW_digits,HW7)
```

```
HW8<-print_HW_digit(readPNG("eight.png"),"HW8")
HW_digits<-rbind(HW_digits,HW8)
HW9<-print_HW_digit(readPNG("nine.png"),"HW9")
HW_digits<-rbind(HW_digits,HW9)
```

```
#call the function on the handwritten digit of your choice
draw_digit(HW_digits,"HW0")
```

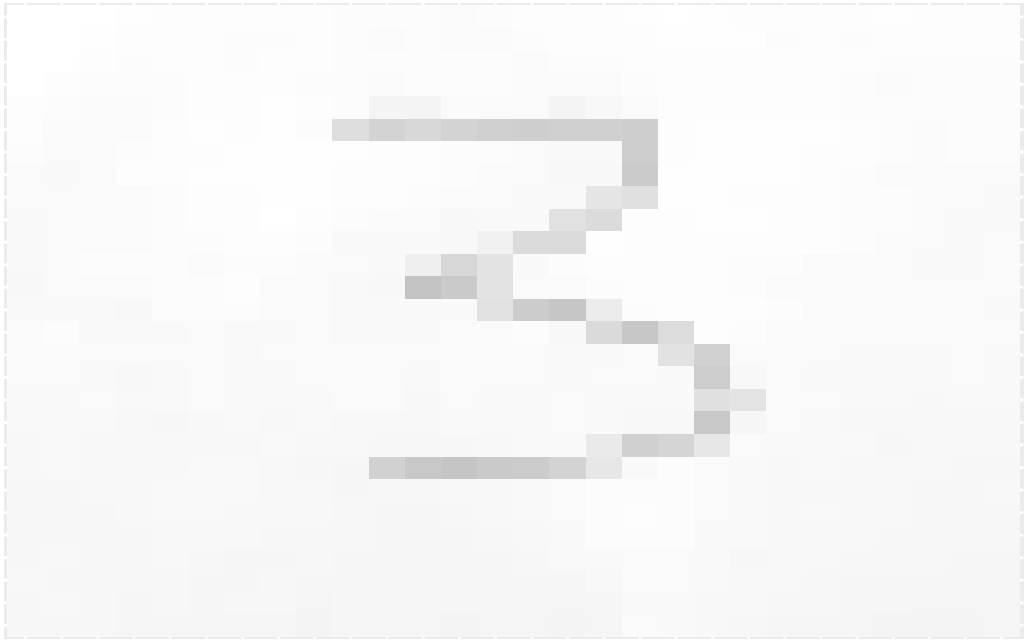


```
draw_digit(HW_digits,"HW1")
```

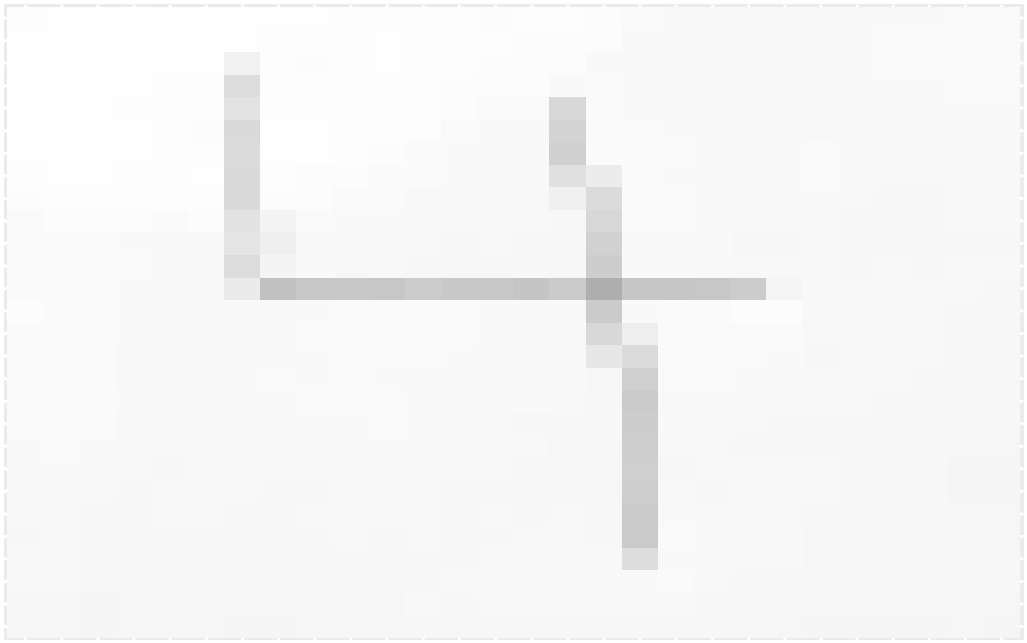


```
draw_digit(HW_digits,"HW2")
```

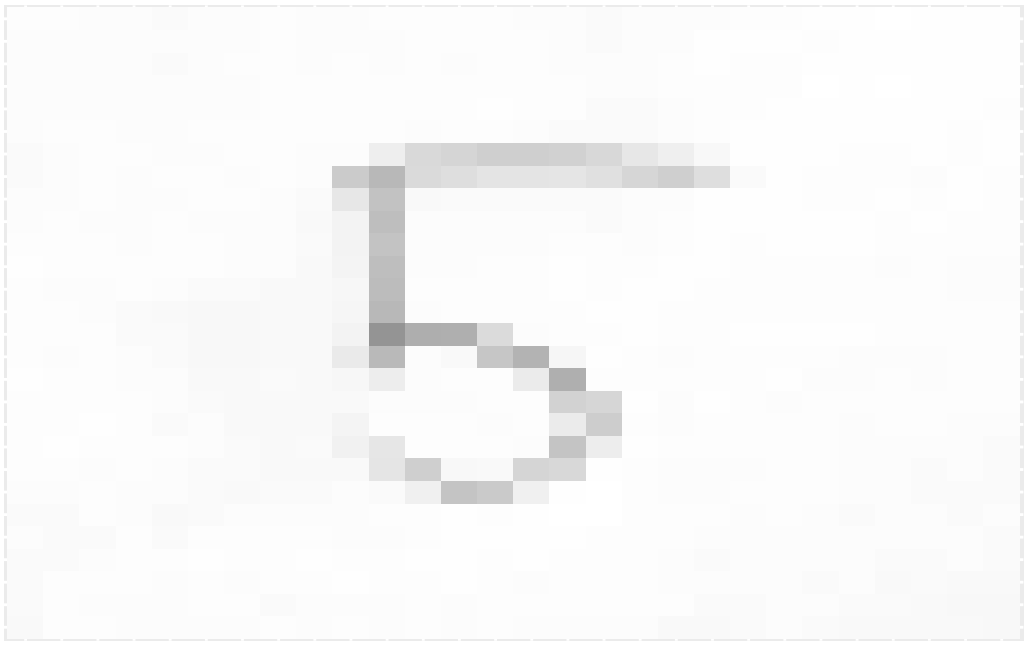
```
draw_digit(HW_digits,"HW3")
```



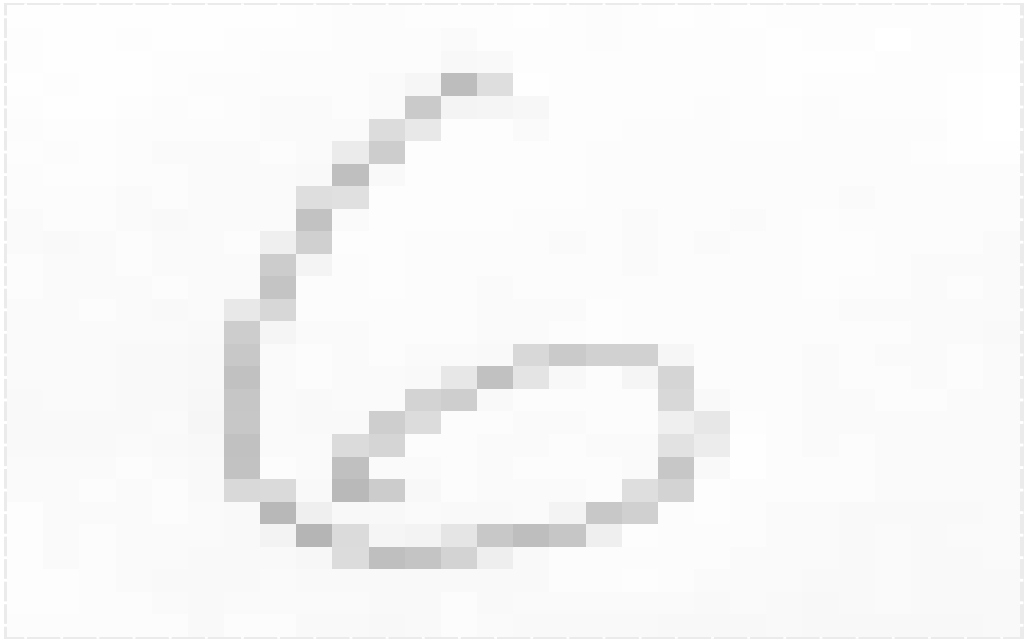
```
draw_digit(HW_digits,"HW4")
```



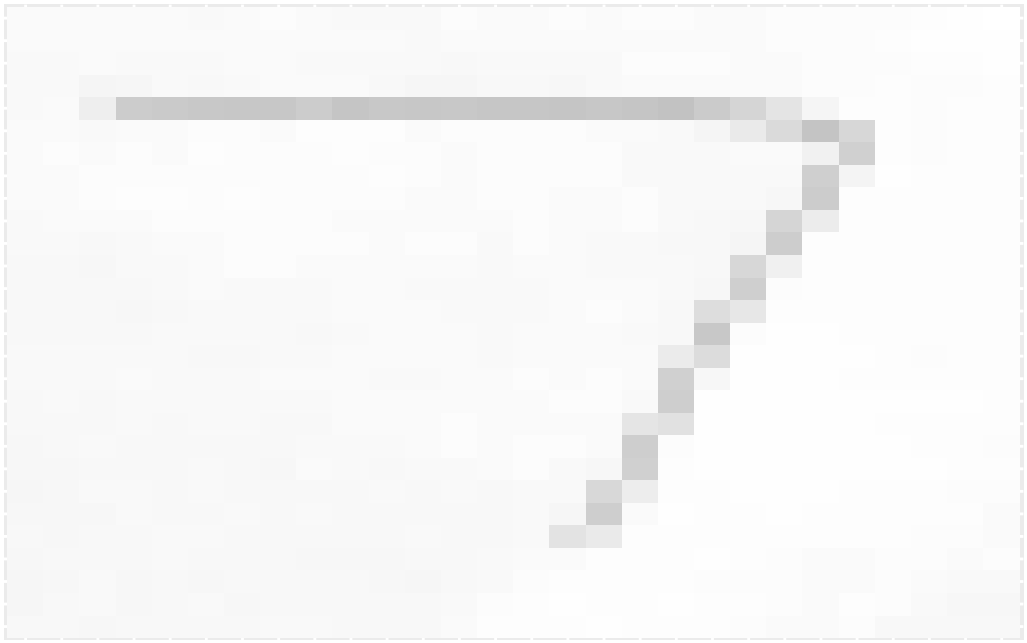
```
draw_digit(HW_digits,"HW5")
```



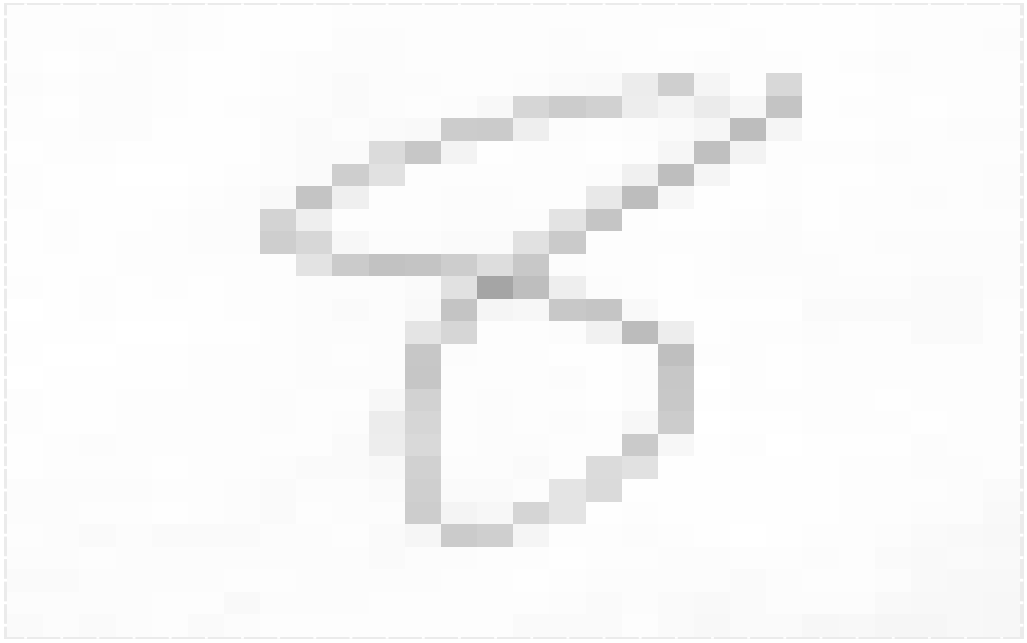
```
draw_digit(HW_digits,"HW6")
```



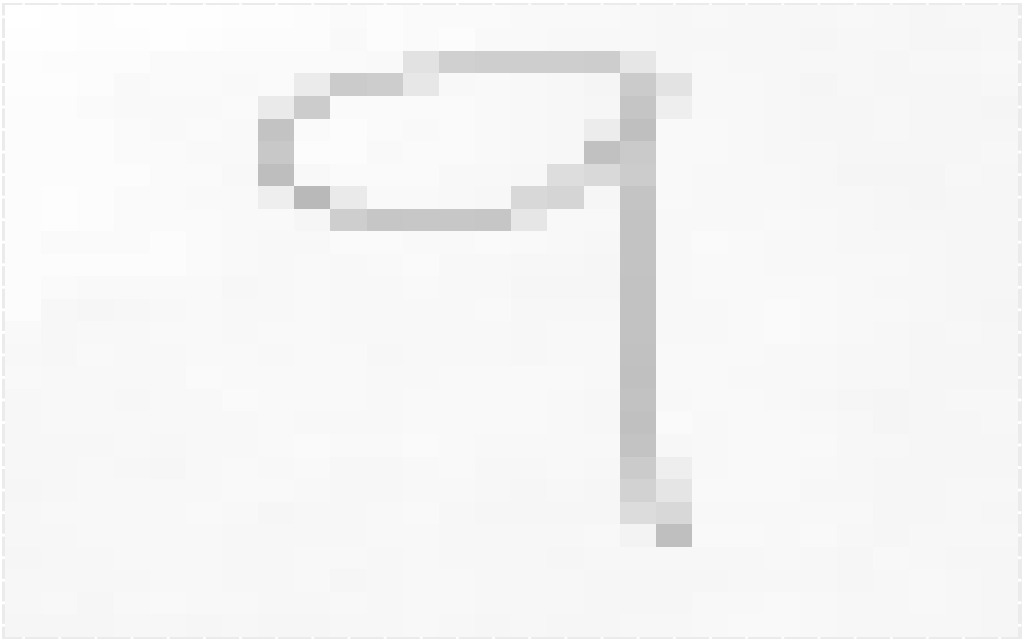
```
draw_digit(HW_digits,"HW7")
```



```
draw_digit(HW_digits,"HW8")
```



```
draw_digit(HW_digits,"HW9")
```



Dataset 2

```
#Install/Load libraries needed
#install.packages("reshape2")
#install.packages("dplyr")
#install.packages("Hmisc")
```

Question 1

```
library("dplyr")
```

```
Warning: package 'dplyr' was built under R version 4.2.2
```

```
Attaching package: 'dplyr'
```

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

    data_frame

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

    filter, lag

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

    intersect, setdiff, setequal, union
```

```
library("Hmisc")
```

```
Warning: package 'Hmisc' was built under R version 4.2.2

Loading required package: lattice

Loading required package: survival

Loading required package: Formula


Attaching package: 'Hmisc'

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

    src, summarize

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

    format.pval, units
```

```
library("reshape2")

#Read both files, set header to true
mcoldata<-read.csv("Mnemiopsis_col_data.csv",header=TRUE)
```

```r
mcountdata<-read.csv("Mnemiopsis_count_data.csv",header=TRUE)
```

```r
#make new column with mean expression for all experiments with a for loop
for(i in 1:nrow(mcountdata))
{
  mcountdata$expmean <- ((mcountdata$aboral1)+(mcountdata$aboral2)+(mcountdata$aboral3)+(m
}
```

```r
#make new column with mean expression for all experiments with a for loop
for(i in 1:nrow(mcountdata))
{
  mcountdata$expmean <- ((mcountdata$aboral1)+(mcountdata$aboral2)+(mcountdata$aboral3)+(m
}
```

```r
#Sort the dataframe by expmean and check the top 5
sortedexpmean<-mcountdata[order(-mcountdata$expmean),]
head(sortedexpmean)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
16420  ML46651a  125638  105808   65907   93351  16236  10449  22838  58247
2612  ML020045a   80445   48643   60380   45170  65580  54406  35861  48147
30     ML00017a   52713   57824   59132   60254  59242  47001  48346  47841
4249   ML04011a   49536   55951   56601   47869  64225  50041  44420  41929
        expmean
12714 122241.38
14235  64737.00
16420  62309.25
2612   54829.00
30     54044.12
4249   51321.50
```

Q1. What are the top 5 genes with the highest average expression (across experiments) in the set? What is their function?

- The top 5 genes with the highest average expression across experiments are: ML20395a, ML26358a, ML46651a, ML020045a, and ML00017a.
- Their functions are:
- ML20395a: Elongation factor 1-alpha (translation)

- ML26358a: Actin (major protein constituent of cytoskeleton-->microfilaments, and for thin filaments in muscle fibrils)
- ML46651a: Membrane attack complex? (according to Argot2: no other results)
- ML020045a: Tubulin beta chain (second protein component of microtubule)
- ML00017a: Elongation factor 2 (translation)

Question 2

```
#Create new variables that hold descending values for each column
sortedaboral1<-mcountdata[order(-mcountdata$aboral1),]
head(sortedaboral1)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
16420  ML46651a  125638  105808   65907   93351  16236  10449  22838  58247
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
2612  ML020045a   80445   48643   60380   45170  65580  54406  35861  48147
11879 ML174731a   70893    3135   22080     185  40422  32876   3125  27576
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
30      ML00017a   52713   57824   59132   60254  59242  47001  48346  47841
        expmean
16420  62309.25
12714 122241.38
2612   54829.00
11879  25036.50
14235  64737.00
30     54044.12
```

```
sortedaboral2<-mcountdata[order(-mcountdata$aboral2),]
head(sortedaboral2)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
16420  ML46651a  125638  105808   65907   93351  16236  10449  22838  58247
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
1908   ML01482a   32503   90804   83222  111860  15018  11845  36717  22066
3788  ML034334a   23288   76895   65076   94170   4216   6801  14845  10235
3790  ML034336a   25116   74297   59568   84219   5130   6048  14005   9833
        expmean
```

```
12714 122241.38
16420  62309.25
14235  64737.00
1908   50504.38
3788   36940.75
3790   34777.00
```

```
sortedaboral3<-mcountdata[order(-mcountdata$aboral3),]
head(sortedaboral3)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
1908   ML01482a   32503   90804   83222  111860  15018  11845  36717  22066
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
16420  ML46651a  125638  105808   65907   93351  16236  10449  22838  58247
3788  ML034334a   23288   76895   65076   94170   4216   6801  14845  10235
2612   ML020045a   80445   48643   60380   45170  65580  54406  35861  48147
        expmean
12714 122241.38
1908   50504.38
14235  64737.00
16420  62309.25
3788   36940.75
2612   54829.00
```

```
sortedaboral4<-mcountdata[order(-mcountdata$aboral4),]
head(sortedaboral4)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
1908   ML01482a   32503   90804   83222  111860  15018  11845  36717  22066
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
3788  ML034334a   23288   76895   65076   94170   4216   6801  14845  10235
16420  ML46651a  125638  105808   65907   93351  16236  10449  22838  58247
3790  ML034336a   25116   74297   59568   84219   5130   6048  14005   9833
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
        expmean
1908   50504.38
12714 122241.38
3788   36940.75
16420  62309.25
```

```
3790   34777.00
14235  64737.00
```

```
sortedoral1<-mcountdata[order(-mcountdata$oral1),]
head(sortedoral1)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
2612  ML020045a   80445   48643   60380   45170  65580  54406  35861  48147
4249   ML04011a   49536   55951   56601   47869  64225  50041  44420  41929
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
30     ML00017a   52713   57824   59132   60254  59242  47001  48346  47841
12239  ML18558a   42189   48687   45877   43513  45029  33798  41173  34251
        expmean
12714 122241.38
2612   54829.00
4249   51321.50
14235  64737.00
30     54044.12
12239  41814.62
```

```
sortedoral2<-mcountdata[order(-mcountdata$oral2),]
head(sortedoral2)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
2612  ML020045a   80445   48643   60380   45170  65580  54406  35861  48147
4249   ML04011a   49536   55951   56601   47869  64225  50041  44420  41929
30     ML00017a   52713   57824   59132   60254  59242  47001  48346  47841
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
12239  ML18558a   42189   48687   45877   43513  45029  33798  41173  34251
        expmean
12714 122241.38
2612   54829.00
4249   51321.50
30     54044.12
14235  64737.00
12239  41814.62
```

```
sortedoral3<-mcountdata[order(-mcountdata$oral3),]
head(sortedoral3)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
588   ML004510a     212      81      38     201  24068  31856  54522  69484
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
30     ML00017a   52713   57824   59132   60254  59242  47001  48346  47841
4249   ML04011a   49536   55951   56601   47869  64225  50041  44420  41929
12239  ML18558a   42189   48687   45877   43513  45029  33798  41173  34251
        expmean
12714 122241.38
588    22557.75
14235  64737.00
30     54044.12
4249   51321.50
12239  41814.62
```

```
sortedoral4<-mcountdata[order(-mcountdata$oral4),]
head(sortedoral4)
```

```
          Gene aboral1 aboral2 aboral3 aboral4  oral1  oral2  oral3  oral4
12714  ML20395a  122707  131017  136282  111388 163380 101792 101421 109944
588   ML004510a     212      81      38     201  24068  31856  54522  69484
16420  ML46651a  125638  105808   65907   93351  16236  10449  22838  58247
2612  ML020045a   80445   48643   60380   45170  65580  54406  35861  48147
30     ML00017a   52713   57824   59132   60254  59242  47001  48346  47841
14235  ML26358a   61229   93272   78693   78310  62893  46232  49534  47733
        expmean
12714 122241.38
588    22557.75
16420  62309.25
2612   54829.00
30     54044.12
14235  64737.00
```

Q2. Are the top 5 genes different if done on a per-column basis?

Top 5 genes in full set are: ML20395a, ML26358a, ML46651a, ML020045a, and ML00017a

(S) = same; (D) = different

- When sorted on a per-column basis, the top 5 genes differ as follows:
- aboral1: ML46651a(S), ML20395a(S), ML020045a(S), ML174731a(D),ML26358a(S)
- aboral2: ML20395a(S),ML46651a(S),ML26358a(S),ML01482a(D),ML034334a(D)
- aboral3: ML20395a(S),ML01482a(D),ML26358a(S),ML46651a(S),ML034334a(D)
- aboral4: ML01482a(D),ML20395a(S),ML034334a(D),ML46651a(S),ML034336a(D)
- oral1: ML20395a(S),ML020045a(S),ML04011a(D),ML26358a(S),ML00017a(S)
- oral2: ML20395a(S),ML020045a(S),ML04011a(D),ML00017a(S),ML26358a(S)
- oral3: ML20395a(S),ML004510a(D),ML26358a(S),ML00017a(S),ML04011a(D)
- oral4: ML20395a(S),ML004510a(D),ML46651a(S),ML020045a(S),ML00017a(S)

-- Yes, the top 5 genes vary depending if it is done on a per-column basis. Many of the original top 5 genes reappear in these newly generated "top 5" gene sets, but each column has 1-3 different genes in its "top 5" listing.

Question 3

```
#Calculate mean and standard deviation for each column
#First for aboral1 column
aboral1vec<-mcountdata$aboral1
aboral1mean<-mean(aboral1vec)
aboral1sd<-sd(aboral1vec)

#Now repeat for the rest
aboral2vec<-mcountdata$aboral2
aboral2mean<-mean(aboral2vec)
aboral2sd<-sd(aboral2vec)
#aboral3
aboral3vec<-mcountdata$aboral3
aboral3mean<-mean(aboral3vec)
aboral3sd<-sd(aboral3vec)
#aboral4
aboral4vec<-mcountdata$aboral4
aboral4mean<-mean(aboral4vec)
aboral4sd<-sd(aboral4vec)
#oral1
```

```
oral1vec<-mcountdata$oral1
oral1mean<-mean(oral1vec)
oral1sd<-sd(oral1vec)
#oral2
oral2vec<-mcountdata$oral2
oral2mean<-mean(oral2vec)
oral2sd<-sd(oral2vec)
#oral3
oral3vec<-mcountdata$oral3
oral3mean<-mean(oral3vec)
oral3sd<-sd(oral3vec)
#oral4
oral4vec<-mcountdata$oral4
oral4mean<-mean(oral4vec)
oral4sd<-sd(oral4vec)


#Display mean for each column
aboral1mean
```

```
[1] 524.0979
```

```
aboral2mean
```

```
[1] 580.5219
```

```
aboral3mean
```

```
[1] 581.2736
```

```
aboral4mean
```

```
[1] 560.0897
```

```
oral1mean
```

```
[1] 551.6403
```

```
  oral2mean
```

```
[1] 428.9934
```

```
  oral3mean
```

```
[1] 419.6067
```

```
  oral4mean
```

```
[1] 457.4317
```

```
  aboral1sd
```

```
[1] 2281.937
```

```
  aboral2sd
```

```
[1] 2665.179
```

```
  aboral3sd
```

```
[1] 2451.04
```

```
  aboral4sd
```

```
[1] 2687.429
```

```
  oral1sd
```

```
[1] 2362.584
```

```
  oral2sd
```

```
[1] 1631.392
```

```
  oral3sd
```

```
[1] 1726.889
```

```
  oral4sd
```

```
[1] 1912.523
```

```
  #now scale each column such that the mean is equal to the first column

  #Make a copy of this data frame to put scaled values in
  sc.mcountdata<-mcountdata

  #Scale all values within each column by the conversion factor determined by the column mea
  sc.mcountdata$aboral2<-(sc.mcountdata$aboral1)*(524.1/580.5)
  sc.mcountdata$aboral3<-(sc.mcountdata$aboral1)*(524.1/581.3)
  sc.mcountdata$aboral4<-(sc.mcountdata$aboral1)*(524.1/560.1)
  sc.mcountdata$oral1<-(sc.mcountdata$aboral1)*(524.1/551.6)
  sc.mcountdata$oral2<-(sc.mcountdata$aboral1)*(524.1/429.0)
  sc.mcountdata$oral3<-(sc.mcountdata$aboral1)*(524.1/419.6)
  sc.mcountdata$oral4<-(sc.mcountdata$aboral1)*(524.1/457.4)

  head(sc.mcountdata)
```

```
       Gene aboral1     aboral2     aboral3     aboral4      oral1      oral2
1 ML000110a      69  62.2961240  62.2103905  64.5650777  65.560007  84.295804
2 ML000111a       0   0.0000000   0.0000000   0.0000000   0.000000   0.000000
3 ML000112a       1   0.9028424   0.9015999   0.9357258   0.950145   1.221678
4 ML000113a     383 345.7886305 345.3127473 358.3829673 363.905547 467.902797
5 ML000114a     188 169.7343669 169.5007741 175.9164435 178.627266 229.675524
6 ML000115a     493 445.1012920 444.4887322 461.3128013 468.421501 602.287413
       oral3      oral4 expmean
1  86.184223  79.061871 121.750
2   0.000000   0.000000   0.125
```

```
3   1.249047   1.145824   5.500
4 478.384890 438.850678 360.125
5 234.820782 215.414954 210.375
6 615.780029 564.891342 459.500
```

```
#IGNORE expmean column in sc.mcountdata data frame; just a holdover from copying the orign
```

```
#Create a correlation matrix for the new data frame
#corr.sc.mcountdata<-cor(sc.mcountdata[2:9],sc.mcountdata[2:9])
#corr.sc.mcountdata
#right now just using the unscaled data
corr.mcountdata<-cor(mcountdata[2:9],mcountdata[2:9])
corr.mcountdata
```

```
          aboral1   aboral2   aboral3   aboral4     oral1     oral2     oral3
aboral1 1.0000000 0.8471946 0.8873340 0.7951286 0.8386773 0.8527215 0.7762130
aboral2 0.8471946 1.0000000 0.9720700 0.9747975 0.7403459 0.7430881 0.8011097
aboral3 0.8873340 0.9720700 1.0000000 0.9491527 0.8257897 0.8260390 0.8427193
aboral4 0.7951286 0.9747975 0.9491527 1.0000000 0.6726462 0.6811715 0.7641900
oral1   0.8386773 0.7403459 0.8257897 0.6726462 1.0000000 0.9586231 0.8905611
oral2   0.8527215 0.7430881 0.8260390 0.6811715 0.9586231 1.0000000 0.9308689
oral3   0.7762130 0.8011097 0.8427193 0.7641900 0.8905611 0.9308689 1.0000000
oral4   0.8500432 0.7501215 0.8014047 0.6955056 0.9020024 0.9420304 0.9491639
            oral4
aboral1 0.8500432
aboral2 0.7501215
aboral3 0.8014047
aboral4 0.6955056
oral1   0.9020024
oral2   0.9420304
oral3   0.9491639
oral4   1.0000000
```

```
#unscaled corr
melt.corr.mcountdata<-melt(corr.mcountdata)
head(melt.corr.mcountdata)
```

```
     Var1    Var2     value
1 aboral1 aboral1 1.0000000
2 aboral2 aboral1 0.8471946
```

```
3 aboral3 aboral1 0.8873340
4 aboral4 aboral1 0.7951286
5   oral1 aboral1 0.8386773
6   oral2 aboral1 0.8527215
```

```
sorted.meltcorr<-melt.corr.mcountdata[order(-melt.corr.mcountdata$value),]
head(sorted.meltcorr)
```

```
      Var1    Var2 value
1  aboral1 aboral1     1
19 aboral3 aboral3     1
28 aboral4 aboral4     1
37   oral1   oral1     1
46   oral2   oral2     1
55   oral3   oral3     1
```

```
#remove every other line in the output of sorted.meltcorr to remove the duplicated compari
#We only really need half of the information because its redundant symmetrical around the
```

```
#Create a correlation matrix for the new data frame
#corr.sc.mcountdata<-cor(sc.mcountdata[2:9],sc.mcountdata[2:9])
#corr.sc.mcountdata
#right now just using the unscaled data
corr.mcountdata<-cor(mcountdata[2:9],mcountdata[2:9])
corr.mcountdata
```

```
          aboral1   aboral2   aboral3   aboral4     oral1     oral2     oral3
aboral1 1.0000000 0.8471946 0.8873340 0.7951286 0.8386773 0.8527215 0.7762130
aboral2 0.8471946 1.0000000 0.9720700 0.9747975 0.7403459 0.7430881 0.8011097
aboral3 0.8873340 0.9720700 1.0000000 0.9491527 0.8257897 0.8260390 0.8427193
aboral4 0.7951286 0.9747975 0.9491527 1.0000000 0.6726462 0.6811715 0.7641900
oral1   0.8386773 0.7403459 0.8257897 0.6726462 1.0000000 0.9586231 0.8905611
oral2   0.8527215 0.7430881 0.8260390 0.6811715 0.9586231 1.0000000 0.9308689
oral3   0.7762130 0.8011097 0.8427193 0.7641900 0.8905611 0.9308689 1.0000000
oral4   0.8500432 0.7501215 0.8014047 0.6955056 0.9020024 0.9420304 0.9491639
            oral4
aboral1 0.8500432
aboral2 0.7501215
aboral3 0.8014047
aboral4 0.6955056
```

```
oral1   0.9020024
oral2   0.9420304
oral3   0.9491639
oral4   1.0000000
```

```r
#unscaled corr
melt.corr.mcountdata<-melt(corr.mcountdata)
melt.corr.mcountdata
```

```
      Var1    Var2     value
1  aboral1 aboral1 1.0000000
2  aboral2 aboral1 0.8471946
3  aboral3 aboral1 0.8873340
4  aboral4 aboral1 0.7951286
5    oral1 aboral1 0.8386773
6    oral2 aboral1 0.8527215
7    oral3 aboral1 0.7762130
8    oral4 aboral1 0.8500432
9  aboral1 aboral2 0.8471946
10 aboral2 aboral2 1.0000000
11 aboral3 aboral2 0.9720700
12 aboral4 aboral2 0.9747975
13   oral1 aboral2 0.7403459
14   oral2 aboral2 0.7430881
15   oral3 aboral2 0.8011097
16   oral4 aboral2 0.7501215
17 aboral1 aboral3 0.8873340
18 aboral2 aboral3 0.9720700
19 aboral3 aboral3 1.0000000
20 aboral4 aboral3 0.9491527
21   oral1 aboral3 0.8257897
22   oral2 aboral3 0.8260390
23   oral3 aboral3 0.8427193
24   oral4 aboral3 0.8014047
25 aboral1 aboral4 0.7951286
26 aboral2 aboral4 0.9747975
27 aboral3 aboral4 0.9491527
28 aboral4 aboral4 1.0000000
29   oral1 aboral4 0.6726462
30   oral2 aboral4 0.6811715
31   oral3 aboral4 0.7641900
32   oral4 aboral4 0.6955056
```

```
33 aboral1   oral1 0.8386773
34 aboral2   oral1 0.7403459
35 aboral3   oral1 0.8257897
36 aboral4   oral1 0.6726462
37   oral1   oral1 1.0000000
38   oral2   oral1 0.9586231
39   oral3   oral1 0.8905611
40   oral4   oral1 0.9020024
41 aboral1   oral2 0.8527215
42 aboral2   oral2 0.7430881
43 aboral3   oral2 0.8260390
44 aboral4   oral2 0.6811715
45   oral1   oral2 0.9586231
46   oral2   oral2 1.0000000
47   oral3   oral2 0.9308689
48   oral4   oral2 0.9420304
49 aboral1   oral3 0.7762130
50 aboral2   oral3 0.8011097
51 aboral3   oral3 0.8427193
52 aboral4   oral3 0.7641900
53   oral1   oral3 0.8905611
54   oral2   oral3 0.9308689
55   oral3   oral3 1.0000000
56   oral4   oral3 0.9491639
57 aboral1   oral4 0.8500432
58 aboral2   oral4 0.7501215
59 aboral3   oral4 0.8014047
60 aboral4   oral4 0.6955056
61   oral1   oral4 0.9020024
62   oral2   oral4 0.9420304
63   oral3   oral4 0.9491639
64   oral4   oral4 1.0000000
```

```
sorted.meltcorr<-melt.corr.mcountdata[order(-melt.corr.mcountdata$value),]
sorted.meltcorr
```

```
     Var1    Var2     value
1  aboral1 aboral1 1.0000000
19 aboral3 aboral3 1.0000000
28 aboral4 aboral4 1.0000000
37   oral1   oral1 1.0000000
46   oral2   oral2 1.0000000
```

```
55   oral3   oral3 1.0000000
64   oral4   oral4 1.0000000
10 aboral2 aboral2 1.0000000
12 aboral4 aboral2 0.9747975
26 aboral2 aboral4 0.9747975
11 aboral3 aboral2 0.9720700
18 aboral2 aboral3 0.9720700
38   oral2   oral1 0.9586231
45   oral1   oral2 0.9586231
56   oral4   oral3 0.9491639
63   oral3   oral4 0.9491639
20 aboral4 aboral3 0.9491527
27 aboral3 aboral4 0.9491527
48   oral4   oral2 0.9420304
62   oral2   oral4 0.9420304
47   oral3   oral2 0.9308689
54   oral2   oral3 0.9308689
40   oral4   oral1 0.9020024
61   oral1   oral4 0.9020024
39   oral3   oral1 0.8905611
53   oral1   oral3 0.8905611
3  aboral3 aboral1 0.8873340
17 aboral1 aboral3 0.8873340
6    oral2 aboral1 0.8527215
41 aboral1   oral2 0.8527215
8    oral4 aboral1 0.8500432
57 aboral1   oral4 0.8500432
2  aboral2 aboral1 0.8471946
9  aboral1 aboral2 0.8471946
23   oral3 aboral3 0.8427193
51 aboral3   oral3 0.8427193
5    oral1 aboral1 0.8386773
33 aboral1   oral1 0.8386773
22   oral2 aboral3 0.8260390
43 aboral3   oral2 0.8260390
21   oral1 aboral3 0.8257897
35 aboral3   oral1 0.8257897
24   oral4 aboral3 0.8014047
59 aboral3   oral4 0.8014047
15   oral3 aboral2 0.8011097
50 aboral2   oral3 0.8011097
4  aboral4 aboral1 0.7951286
25 aboral1 aboral4 0.7951286
```

```
7    oral3 aboral1 0.7762130
49 aboral1   oral3 0.7762130
31   oral3 aboral4 0.7641900
52 aboral4   oral3 0.7641900
16   oral4 aboral2 0.7501215
58 aboral2   oral4 0.7501215
14   oral2 aboral2 0.7430881
42 aboral2   oral2 0.7430881
13   oral1 aboral2 0.7403459
34 aboral2   oral1 0.7403459
32   oral4 aboral4 0.6955056
60 aboral4   oral4 0.6955056
30   oral2 aboral4 0.6811715
44 aboral4   oral2 0.6811715
29   oral1 aboral4 0.6726462
36 aboral4   oral1 0.6726462
```

```
#remove every other line in the output of sorted.meltcorr to remove the duplicated compari
#We only really need half of the information because its redundant symmetrical around the
```

```
#remove every other line in the output of sorted.meltcorr to remove the duplicated compari
#We only really need half of the information because its redundant symmetrical around the
sorted.meltcorr2<-sorted.meltcorr[-c(1:8),]
sorted.meltcorr2 #all 1.00 values remove
```

```
     Var1    Var2     value
12 aboral4 aboral2 0.9747975
26 aboral2 aboral4 0.9747975
11 aboral3 aboral2 0.9720700
18 aboral2 aboral3 0.9720700
38   oral2   oral1 0.9586231
45   oral1   oral2 0.9586231
56   oral4   oral3 0.9491639
63   oral3   oral4 0.9491639
20 aboral4 aboral3 0.9491527
27 aboral3 aboral4 0.9491527
48   oral4   oral2 0.9420304
62   oral2   oral4 0.9420304
47   oral3   oral2 0.9308689
54   oral2   oral3 0.9308689
40   oral4   oral1 0.9020024
```

```
61   oral1   oral4 0.9020024
39   oral3   oral1 0.8905611
53   oral1   oral3 0.8905611
3  aboral3 aboral1 0.8873340
17 aboral1 aboral3 0.8873340
6    oral2 aboral1 0.8527215
41 aboral1   oral2 0.8527215
8    oral4 aboral1 0.8500432
57 aboral1   oral4 0.8500432
2  aboral2 aboral1 0.8471946
9  aboral1 aboral2 0.8471946
23   oral3 aboral3 0.8427193
51 aboral3   oral3 0.8427193
5    oral1 aboral1 0.8386773
33 aboral1   oral1 0.8386773
22   oral2 aboral3 0.8260390
43 aboral3   oral2 0.8260390
21   oral1 aboral3 0.8257897
35 aboral3   oral1 0.8257897
24   oral4 aboral3 0.8014047
59 aboral3   oral4 0.8014047
15   oral3 aboral2 0.8011097
50 aboral2   oral3 0.8011097
4  aboral4 aboral1 0.7951286
25 aboral1 aboral4 0.7951286
7    oral3 aboral1 0.7762130
49 aboral1   oral3 0.7762130
31   oral3 aboral4 0.7641900
52 aboral4   oral3 0.7641900
16   oral4 aboral2 0.7501215
58 aboral2   oral4 0.7501215
14   oral2 aboral2 0.7430881
42 aboral2   oral2 0.7430881
13   oral1 aboral2 0.7403459
34 aboral2   oral1 0.7403459
32   oral4 aboral4 0.6955056
60 aboral4   oral4 0.6955056
30   oral2 aboral4 0.6811715
44 aboral4   oral2 0.6811715
29   oral1 aboral4 0.6726462
36 aboral4   oral1 0.6726462
```

```r
#now remove duplicates be deleting every other entry
row_odd<-seq_len(nrow(sorted.meltcorr2))%%2
sorted.meltcorr2.ev<-sorted.meltcorr2[row_odd == 0,]
sorted.meltcorr2.ev
```

```
      Var1    Var2     value
26 aboral2 aboral4 0.9747975
18 aboral2 aboral3 0.9720700
45   oral1   oral2 0.9586231
63   oral3   oral4 0.9491639
27 aboral3 aboral4 0.9491527
62   oral2   oral4 0.9420304
54   oral2   oral3 0.9308689
61   oral1   oral4 0.9020024
53   oral1   oral3 0.8905611
17 aboral1 aboral3 0.8873340
41 aboral1   oral2 0.8527215
57 aboral1   oral4 0.8500432
9  aboral1 aboral2 0.8471946
51 aboral3   oral3 0.8427193
33 aboral1   oral1 0.8386773
43 aboral3   oral2 0.8260390
35 aboral3   oral1 0.8257897
59 aboral3   oral4 0.8014047
50 aboral2   oral3 0.8011097
25 aboral1 aboral4 0.7951286
49 aboral1   oral3 0.7762130
52 aboral4   oral3 0.7641900
58 aboral2   oral4 0.7501215
42 aboral2   oral2 0.7430881
34 aboral2   oral1 0.7403459
60 aboral4   oral4 0.6955056
44 aboral4   oral2 0.6811715
36 aboral4   oral1 0.6726462
```

For correlation values above 0.9, these samples that are closely correlated with each other are concordant with the column labels. However, we also do see high aboral v. oral correlation values at 0.85 and below.

Question 4.

Sorting by PCA

The problem with computing row-wise correlations for every gene is that the output would be 16548^2 calculations. To reduce this, we attempted to only calculate correlations for rows which would have a high correlation. To estimate the correlation strength, we used a method which first calculated a PCA value for each row.

First, we calculated the PCA value for each row, forcing the number of principal components to 1. This should provide an approximation of similarity, so that rows likely to have high correlations have similar PCA values. By sorting by this value we place rows likely to be highly correlated near each other in the data frame. After this, we test the correlation of each row versus the five following rows and record the correlation. This sorting after dimensionality reduction is performed to reduce the computational load for calculating n^2 correlations. The method is not perfect, and may omit some highly correlated pairs, but should provide decent coverage of highly correlated genes.

The histogram below shows the distribution of correlations, which does skew towards higher correlations, though maybe not as strongly as we would have liked. Increasing the number of PC parameters may improve this model.

The genes with high correlation have very similar expression patterns across groups.

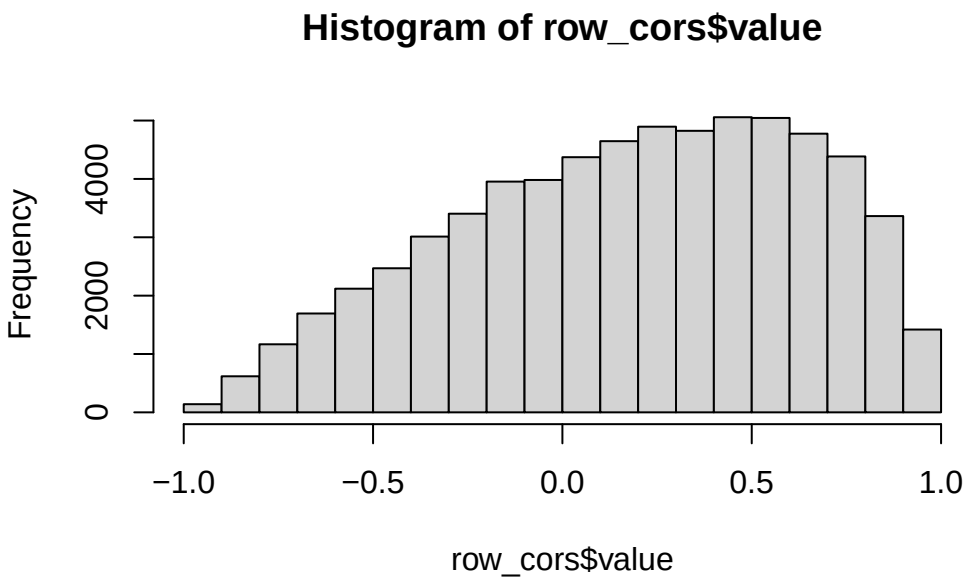
```
library(reshape2)

#create pca values for each row
pca_data<-prcomp(mcountdata[,2:9],rank. = 1)
#add the pca values to the full dataset
mcount_pca<-cbind(mcountdata,pca_data$x)
#drop zeroes
mcount_pca[mcount_pca==0]<-NA
mcount_pca<-na.omit(mcount_pca)
#sort by pca value
mcount_pca_sort<-mcount_pca[order(mcount_pca$PC1),]
#transpose and drop non-numeric data
mcount_pca_sort_t<-t(mcount_pca_sort[2:9])
#colnames
colnames(mcount_pca_sort_t)<-mcount_pca_sort$Gene
#create a dataframe to store correlations
row_cors<-data.frame(cors=as.numeric())
#define range to correlate across
range<-5
```

```
#loop through each row and correlate against nearby rows
#(in its own code cell to perform calculation separate from related code)
for(x in 1:(ncol(mcount_pca_sort_t)-range)){
  #upper bound is x+5 unless outside of range
  upper<-x+range
  if(upper>ncol(mcount_pca_sort_t)){upper<-ncol(mcount_pca_sort_t)}
  #lower bound is x+1
  lower<-x+1
  #if(lower<1){lower<-1}
  #store correlations
  temp<-cor(mcount_pca_sort_t[1:8,x],mcount_pca_sort_t[1:8,lower:upper])
  rownames(temp)<-colnames(mcount_pca_sort_t)[x]
  temp_melt<-melt(temp)
  row_cors<-rbind(row_cors,temp_melt)
}
```

```
#sort by correlation
row_cors<-row_cors[order(row_cors$value, decreasing = TRUE),]
#print top 10 pairs of genes with strongest correlation
head(row_cors,5)
```

```
          Var1      Var2     value
64464  ML45843a ML073030a 0.9993858
64913  ML00365a ML193210a 0.9985403
65316 ML034336a ML034334a 0.9981401
65311 ML034337a ML034336a 0.9969169
65204 ML148538a ML148534a 0.9960324
```

```
#examine the distribution of the correlations
hist(row_cors$value)
```




```
#check the top pair to see why they are so closely correlated
mcountdata[which(mcountdata[,"Gene"]=="ML45843a"),]
```

```
         Gene aboral1 aboral2 aboral3 aboral4 oral1 oral2 oral3 oral4  expmean
16330 ML45843a    4182   10115    9354   12679   799  1155  3322  2453 5507.375
```

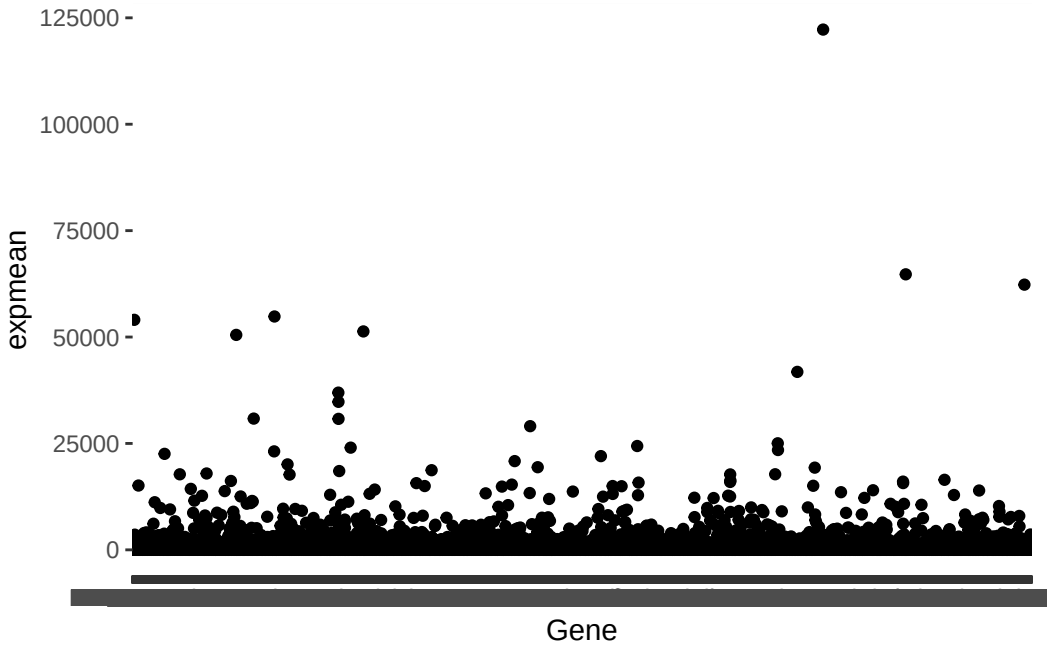
```
mcountdata[which(mcountdata[,"Gene"]=="ML073030a"),]
```

```
         Gene aboral1 aboral2 aboral3 aboral4 oral1 oral2 oral3 oral4  expmean
6866 ML073030a    4256   10042    9666   12496  1069  1335  3394  2379 5579.625
```

Question 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?

#first plot the data to see the distribution of gene expression
library(ggplot2)
ggplot(mcountdata, aes(x=Gene,y= expmean)) +
```

```
  geom_point()
```



Question 6.

```
# if (!require("BiocManager", quietly = TRUE))
#     install.packages("BiocManager")
#
# BiocManager::install("DESeq2")
#
# if (!require("BiocManager", quietly = TRUE))
#     install.packages("BiocManager")
#
# BiocManager::install("airway")
```

```
#6) make a list of the top 5 genes with most variability and top 5 genes with least variab


#perform differential expression before filtering by variance so you are using normalized
#must do between-sample normalization, which is needed to account for technical effects (d
#for this we will do DESeq differential expression
```

```
#read in with no header
coldata<-read.csv("Mnemiopsis_col_data.csv")
countdata<-read.csv("Mnemiopsis_count_data.csv")



#be sure all colnames in count data are in col data
all(colnames(countdata))%in%rownames(coldata)
```

```
Warning in all(colnames(countdata)): coercing argument of type 'character' to
logical
```

```
[1] FALSE
```

```
#make gene column in countdata into the rownames instead of it's own column
#do the same with the sample column in count data
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.2.2
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v tibble  3.1.8     v purrr   0.3.5
v tidyr   1.2.1     v stringr 1.4.1
v readr   2.1.3     v forcats 0.5.2
```

```
Warning: package 'tidyr' was built under R version 4.2.2
```

```
Warning: package 'readr' was built under R version 4.2.2
```

```
Warning: package 'purrr' was built under R version 4.2.2
```

```
Warning: package 'forcats' was built under R version 4.2.2
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x tibble::data_frame() masks dplyr::data_frame(), vctrs::data_frame()
x dplyr::filter()      masks stats::filter()
x dplyr::lag()         masks stats::lag()
x Hmisc::src()         masks dplyr::src()
x Hmisc::summarize()   masks dplyr::summarize()
```

```
coldata <- data.frame(coldata, row.names = 1)#set the first column to the row names
countdata <- data.frame(countdata, row.names = 1)

#rename columns in count data to be the rownames in coldata
colnames(countdata)=rownames(coldata)

print(rownames(coldata))
```

```
[1] "aboral-1" "aboral-2" "aboral-3" "aboral-4" "oral-1"   "oral-2"   "oral-3"
[8] "oral-4"
```

```
print(colnames(countdata))
```

```
[1] "aboral-1" "aboral-2" "aboral-3" "aboral-4" "oral-1"   "oral-2"   "oral-3"
[8] "oral-4"
```

```
#the rownames in col data are the same as the colnames in countdata

#now make sure they are in the same order
all(colnames(countdata)==rownames(coldata))
```

```
[1] TRUE
```

```
library(DESeq2)
```

```
Loading required package: S4Vectors
Loading required package: stats4
Loading required package: BiocGenerics

Attaching package: 'BiocGenerics'

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

    combine, intersect, setdiff, union

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

```
    IQR, mad, sd, var, xtabs

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

    anyDuplicated, aperm, append, as.data.frame, basename, cbind,
    colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
    get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
    match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
    Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
    table, tapply, union, unique, unsplit, which.max, which.min


Attaching package: 'S4Vectors'

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

    expand

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

    first, rename

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

    expand.grid, I, unname

Loading required package: IRanges

Attaching package: 'IRanges'

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

    reduce

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

    collapse, desc, slice

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

    windows

Loading required package: GenomicRanges
```

```
Loading required package: GenomeInfoDb

Warning: package 'GenomeInfoDb' was built under R version 4.2.2

Loading required package: SummarizedExperiment
Loading required package: MatrixGenerics
Loading required package: matrixStats

Warning: package 'matrixStats' was built under R version 4.2.2


Attaching package: 'matrixStats'

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

    count


Attaching package: 'MatrixGenerics'

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

    colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
    colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
    colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
    colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
    colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
    colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
    colWeightedMeans, colWeightedMedians, colWeightedSds,
    colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
    rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
    rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
    rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
    rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
    rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
    rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
    rowWeightedSds, rowWeightedVars

Loading required package: Biobase
Welcome to Bioconductor
```

```
    Vignettes contain introductory material; view with
    'browseVignettes()'. To cite Bioconductor, see
    'citation("Biobase")', and for packages 'citation("pkgname")'.


Attaching package: 'Biobase'

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

    rowMedians

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

    anyMissing, rowMedians

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

    contents
```

```
library(tidyverse)
library(airway)

#construct DESeq2 data set
dds<-DESeqDataSetFromMatrix(countData=countdata,
                        colData = coldata,
                        design = ~condition)
```

```
Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
design formula are characters, converting to factors
```

```
dds
```

```
class: DESeqDataSet
dim: 16548 8
metadata(1): version
assays(1): counts
rownames(16548): ML000110a ML000111a ... ML50792a ML50851a
rowData names(0):
colnames(8): aboral-1 aboral-2 ... oral-3 oral-4
colData names(2): type condition
```

```
#design is the factor in mcoldata that specifies the condition of the samples. as in if th
```

```
#set a factor level. compare between aboral and oral samples. we need to tell deseq to use
dds$condition<- relevel(dds$condition, ref = "aboral")
#this would have been the case either way because it assigns a reference level alphabetica
```

```
#run DESeq
#save it back to the same object
dds<-DESeq(dds)
```

```
estimating size factors

estimating dispersions

gene-wise dispersion estimates

mean-dispersion relationship

final dispersion estimates

fitting model and testing
```

```
res<-results(dds)
res
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 16548 rows and 6 columns
            baseMean log2FoldChange     lfcSE      stat    pvalue      padj
           <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML000110a 120.909051      0.0341593  0.369949  0.092335 0.9264319  0.979483
ML000111a   0.135431      0.9567868  3.533812  0.270752 0.7865817        NA
ML000112a   5.495487      0.2981197  1.022371  0.291596 0.7705953  0.931240
ML000113a 352.666162     -0.4999978  0.233415 -2.142096 0.0321857  0.160530
ML000114a 206.399933      0.0460193  0.232538  0.197900 0.8431230  0.956648
...              ...            ...       ...       ...       ...       ...
ML50721a    13.37004      1.6975583  0.724000 2.3446945 0.0190427  0.107926
ML50771a    10.81640      0.1903091  0.633715 0.3003070 0.7639430  0.929187
```

```
ML50791a     0.00000             NA        NA        NA        NA        NA
ML50792a     1.85463      0.0661972  1.392121 0.0475513 0.9620738  0.989589
ML50851a     3.71506      0.4076515  1.052232 0.3874159 0.6984483  0.904537
```

log2fold change column: positive values are up regulated genes, negative values are down regulated.

```
summary(res)
```

```
out of 15112 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 1355, 9%
LFC < 0 (down)     : 1149, 7.6%
outliers [1]       : 35, 0.23%
low counts [2]     : 583, 3.9%
(mean count < 1)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

```
#we can adjust the pvalue so as not to detect false potives
re0.01<-results(dds, alpha = 0.01)
summary(re0.01)
```

```
out of 15112 with nonzero total read count
adjusted p-value < 0.01
LFC > 0 (up)       : 955, 6.3%
LFC < 0 (down)     : 668, 4.4%
outliers [1]       : 35, 0.23%
low counts [2]     : 583, 3.9%
(mean count < 1)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

summary shows how many genes are up and down regulated, how many are outliers, etc.

```
resultsNames(dds)
```

```
[1] "Intercept"                 "condition_oral_vs_aboral"
```

this plot tells us the genes that are differentially expressed. significantly differentially expressed genes,(in blue) the blue has adjusted p value of less than 0.05.

the triangles indicate the genes have higher fold changes. direction of the triangles tells you the direction of the fold change.

we want to see genes in the upper right or lower right quadrant because this means the genes have high means of normalized counts and high log fold changes. these are interesting genes to be looked in to.

most of the data is between an expmean of 0 and 25000. therefore to divide the data into three groups, you must decide on a cutoff for low medium and high. the genes cannot be equally divided into three groups.

```
#variability, top 5 genes with the highest variability
sel_high = order(apply(re0.01, 1, var), decreasing=TRUE)[1:5]
sel_high
```

```
[1] 12714 14235 16420  2612    30
```

```
#top 5 genes with the lowest variability
sel_low = order(apply(re0.01, 1, var), decreasing=FALSE)[1:5]
sel_low
```

```
[1] 15128 14811  9412 11633 12414
```

these numbers are the indicies to re0.01

```
#print the indicies from re0.01 to get the genes with the top 5 highest and lowest variabi

#highest variability indicies
print(re0.01[11025,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML14874a   14.1514      -0.718393   0.56208   -1.2781  0.201216  0.536501
```

```
print(re0.01[12343,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
           baseMean log2FoldChange     lfcSE      stat    pvalue      padj
          <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML190411a   7.98496       0.459132  0.704762  0.651471  0.514743  0.811649
```

```
print(re0.01[14204,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML26174a    1396.9      0.0493111  0.106318  0.463809  0.642785   0.87995
```

```
print(re0.01[2298,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
           baseMean log2FoldChange     lfcSE      stat    pvalue      padj
          <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML017910a   17.6566       -0.10098  0.462108  -0.21852  0.827024  0.951727
```

```
print(re0.01[27,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML00014a         0             NA        NA        NA        NA        NA
```

```
#lowest variability indices
print(re0.01[13108,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML21531a    12.852      -0.463118  0.664998  -0.69642  0.486166  0.799375
```

```
print(re0.01[12839,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
           baseMean log2FoldChange     lfcSE      stat    pvalue      padj
          <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML206417a    123.18      -0.168239  0.456829 -0.368276  0.712667   0.90768
```

```
print(re0.01[10103,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
           baseMean log2FoldChange     lfcSE      stat    pvalue      padj
          <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML129321a   74.7509       0.329977   0.29434   1.12107  0.262257  0.615577
```

```
print(re0.01[8197,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML08924a   786.053        1.12767  0.443732   2.54133 0.0110432 0.0697524
```

```
print(re0.01[12160,])
```

```
log2 fold change (MLE): condition oral vs aboral
Wald test p-value: condition oral vs aboral
DataFrame with 1 row and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ML18259a   329.239       0.295764  0.310954  0.951151  0.341528  0.693389
```

highest variability genes from greatest variability to least: ML20395a, ML26358a, ML46651a, ML020045a, ML00017a

lowest variability genes from least variability to greatest: ML32095a, ML29351a, ML16594a, ML11345a, ML25222a

Question 7.

```
#Take the mean of all aboral expressions and all oral expression for each gene
#Calculate a ratio of aboral vs oral
#take the log of this ratio, most positive and negative 5 values are the most up and down

#Loop through and create a new column for aboral means
for(i in 1:nrow(mcountdata))
{
  mcountdata$aboralmean <- ((mcountdata$aboral1)+(mcountdata$aboral2)+(mcountdata$aboral3)
}

#And for oral means
for(i in 1:nrow(mcountdata))
{
  mcountdata$oralmean <- ((mcountdata$oral1)+(mcountdata$oral2)+(mcountdata$oral3)+(mcount
}

#Column containing fold-change ratio of aboral vs oral means
for(i in 1:nrow(mcountdata))
{
  mcountdata$avoratio <- ((mcountdata$aboralmean)/(mcountdata$oralmean))
}

#Take the log of aboral vs oral ratio
for(i in 1:nrow(mcountdata))
```

```
{
  mcountdata$logavoratio <- log((mcountdata$aboralmean)/(mcountdata$oralmean))
}
```

- We can check the generated data frame and check the most positive and negative non-zero values. The most positive are the most upregulated, and the most negative are the most downregulated (for a compared to b). Using simply the log method, the genes we are interested in are as follows. (Excluding positive and negative infinity log values). format: gene name(log of aboral vs oral ratio)
- Most upregulated aboral vs oral: ML327424a(6.169369), ML343422a(5.351331), ML14971a(5.258369), ML27982a(4.941642), and ML311627a(4.862107)
- Most downregulated aboral vs oral: ML34341a(-9.785023), ML090812a(-9.394743), ML087114a(-8.896168), ML034332a(-8.767921), and ML319815a(-8.266678)
- second part of Q7 found below

```
#Remove cells that have non numerical value in the logavoratio column
#Make a new df for this
validmcountdata<-mcountdata

validmcountdata<-validmcountdata[- grep("NaN", validmcountdata$logavoratio),]
validmcountdata<-validmcountdata[- grep("Inf", validmcountdata$logavoratio),]
```

```
#Also create a column for p-values from t-test results between aboralmean v oralmean
pcounter<-1
for(i in 1:nrow(validmcountdata))
{
  ab<-c(validmcountdata$aboral1[pcounter],validmcountdata$aboral2[pcounter],validmcountdat
  or<-c(validmcountdata$oral1[pcounter],validmcountdata$oral2[pcounter],validmcountdata$or
  validmcountdata$ttpval[pcounter] <- t.test(ab,or)$p.value
  pcounter<- pcounter+1
}
```

```
head(validmcountdata) #show that it worked properly
```

```
      Gene aboral1 aboral2 aboral3 aboral4 oral1 oral2 oral3 oral4 expmean
1 ML000110a      69     175     141     139   108   146   133    63 121.750
3 ML000112a       1      10       8       3     2    13     6     1   5.500
4 ML000113a     383     546     402     471   290   190   282   317 360.125
5 ML000114a     188     214     257     230   289   215   162   128 210.375
```

```
6 ML000115a     493     455     540     501   413   403   419   452 459.500
7 ML000116a     404     462     464     362   516   336   285   336 395.625
  aboralmean oralmean avoratio logavoratio      ttpval
1     131.00   112.50 1.164444   0.1522441 0.545318914
3       5.50     5.50 1.000000   0.0000000 1.000000000
4     450.50   269.75 1.670065   0.5128625 0.009173149
5     222.25   198.50 1.119647   0.1130138 0.565346780
6     497.25   421.75 1.179016   0.1646802 0.014222061
7     423.00   368.25 1.148676   0.1386101 0.382324084
```

We can also rank by p-value of the t-test, which will tell us which genes have the most highly differential gene expression between the aboral vs oral values.

The top 10 genes with the lowest t-test p-values are shown below:

ML050913a, ML263524a, ML01833a, ML329912a, ML070258a, ML005114a, ML204423a, ML282521a, ML15096a, ML102911a

```
sortedpval<-validmcountdata[order(validmcountdata$ttpval),]
head(sortedpval,10)
```

```
           Gene aboral1 aboral2 aboral3 aboral4 oral1 oral2 oral3 oral4
5203  ML050913a    8169    8532    8195    7853     3   212    14   266
14228 ML263524a     537     551     580     585   267   266   220   285
2425   ML01833a     546     560     548     584   373   342   325   332
15283 ML329912a    2614    2287    2601    2437   998  1175   817   996
6628  ML070258a    1180    1265    1354    1339   496   652   508   459
714   ML005114a     352     414     408     347   712   675   680   731
12735 ML204423a     231     227     220     220   175   164   159   166
14699 ML282521a      10       7      11       9   147   140   150   135
11230  ML15096a     382     421     393     413   221   236   241   265
8980  ML102911a      49      42      47      47    19    18    25    20
       expmean aboralmean oralmean    avoratio logavoratio       ttpval
5203  4155.500    8187.25   123.75 66.15959596   4.1920699 3.006244e-07
14228  411.375     563.25   259.50  2.17052023   0.7749669 3.773984e-06
2425   451.250     559.50   343.00  1.63119534   0.4893131 5.611578e-06
15283 1740.625    2484.75   996.50  2.49347717   0.9136782 8.505305e-06
6628   906.625    1284.50   528.75  2.42931442   0.8876091 1.321939e-05
714    539.875     380.25   699.50  0.54360257  -0.6095369 1.334813e-05
12735  195.250     224.50   166.00  1.35240964   0.3018879 1.357152e-05
14699   76.125       9.25   143.00  0.06468531  -2.7382211 1.413726e-05
11230  321.500     402.25   240.75  1.67082035   0.5133147 1.516682e-05
8980    33.375      46.25    20.50  2.25609756   0.8136366 2.108346e-05
```