PreLab 6: Eigenfaces

Your Name Here

3/17/2021

Completing Lab

If you are unable to connect to the Rstudio server please contact Prof. Bennett bennek@rpi.edu and Dr. Erickson erickj4@rpi.edu immediately.

Overview

This prelab will help you learn to work with image data and prepare you to make a facial recognition system in Lab 6.

Examine Data

First familiarize yourself with the dataset, `faces.csv`. The following code reads the data into the dataframe, `F.df`.

- Each image vector appears as rows in `F.matrix`. Each image is assigned a row name which is the order in which the image occurs in the original database.
- The `labels` vector contains the ID of the person in the picture. If the labels match, then the pictures are of the same person.

```
# Read in data
F.df <- read.csv('~/MATP-4400/data/faces.csv')
# Save first column as labels
labels <- as.numeric(F.df[,1])
F.matrix<-as.matrix(F.df[,-1])
# Assign the row names
row.names(F.matrix)<-1:(nrow(F.matrix))
```

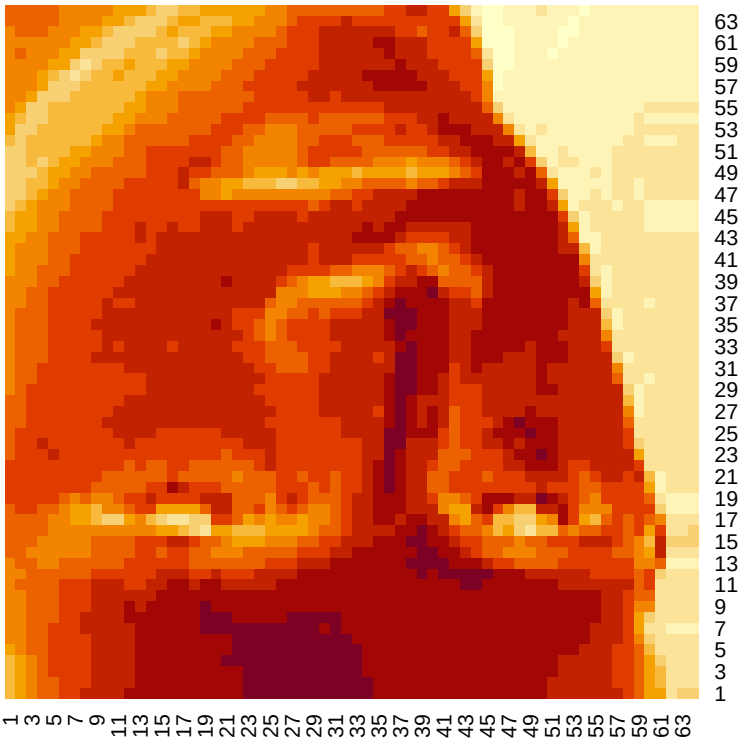
*Exercise*1

Each image vector represents a 64x64 matrix. First we "convert" the 10th image vector into a 64x64 matrix and draw it as a heatmap with no scaling and no dendrogram. Then we display the matrix using `heatmap()`. Play with the matrix until it makes a properly oriented picture.

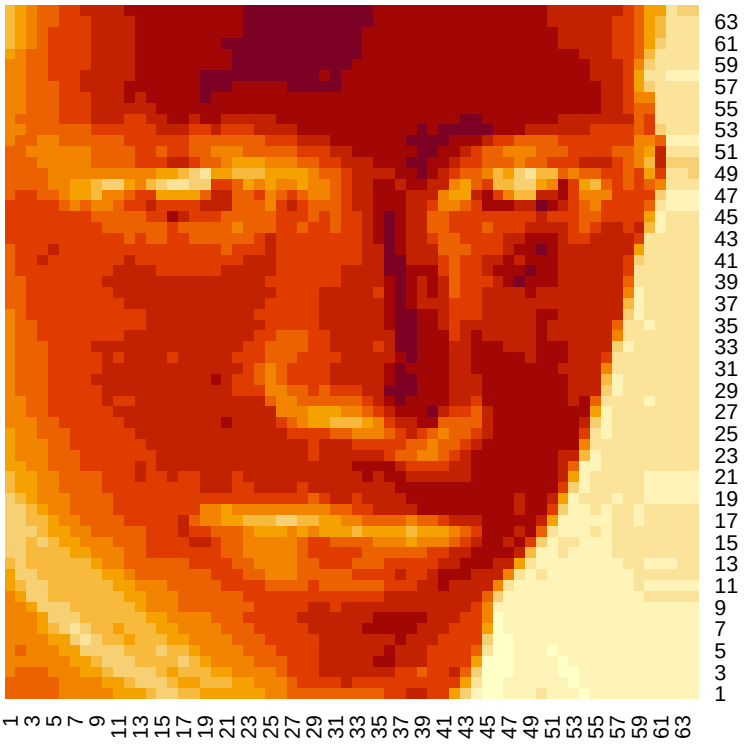
HINTS:

- You may need to reorder the rows and/or columns.
- The R command `64:1` gives a sequence of integers 64 down to 1.

```
# Converting the 10th (row) image into its own 64x64 matrix
vi<-matrix(F.matrix[10,],ncol=64)
heatmap(vi,scale="none", Rowv=NA, Colv=NA )
```




```
# Reverse the rows of the matrics (so first becomes the last)
rev_vi = vi[nrow(vi):1,]
heatmap(rev_vi,scale="none", Rowv=NA, Colv=NA)
```



View the Data

In the following code we've provided the function, `faceplot()`, which plots faces in greyscale based on their vectors and arranges them in a grid:

```
# A function to help plot faces
faceplot <- function(xx,width=64,midcolor="grey10",gcols=2,labels=row.names(xx)) {
# Note require(ggplot2); require(reshape2); require(gridExtra)
 if (is.vector(xx)) {
   xx <- matrix(xx,nrow=1)
}
  pl <- vector("list",nrow(xx))
  for(i in 1:nrow(xx)){
    face <- matrix(xx[i,], nrow=width)
    face.m <- melt(apply(face, 2, function(x) as.numeric(rev(x))))
    pl[[i]] <- ggplot(data=face.m) + geom_raster(aes(x=Var2, y=Var1, fill=value)) +
        theme(axis.text=element_blank(), axis.title=element_blank()) + guides(fill=FALSE) +
        scale_fill_gradient2(low="black", mid=midcolor, high="white") +ggtitle(labels[i])
  }
  grid.arrange(grobs=pl,ncol=gcols)
}
```

`faceplot()` takes the following arguments:

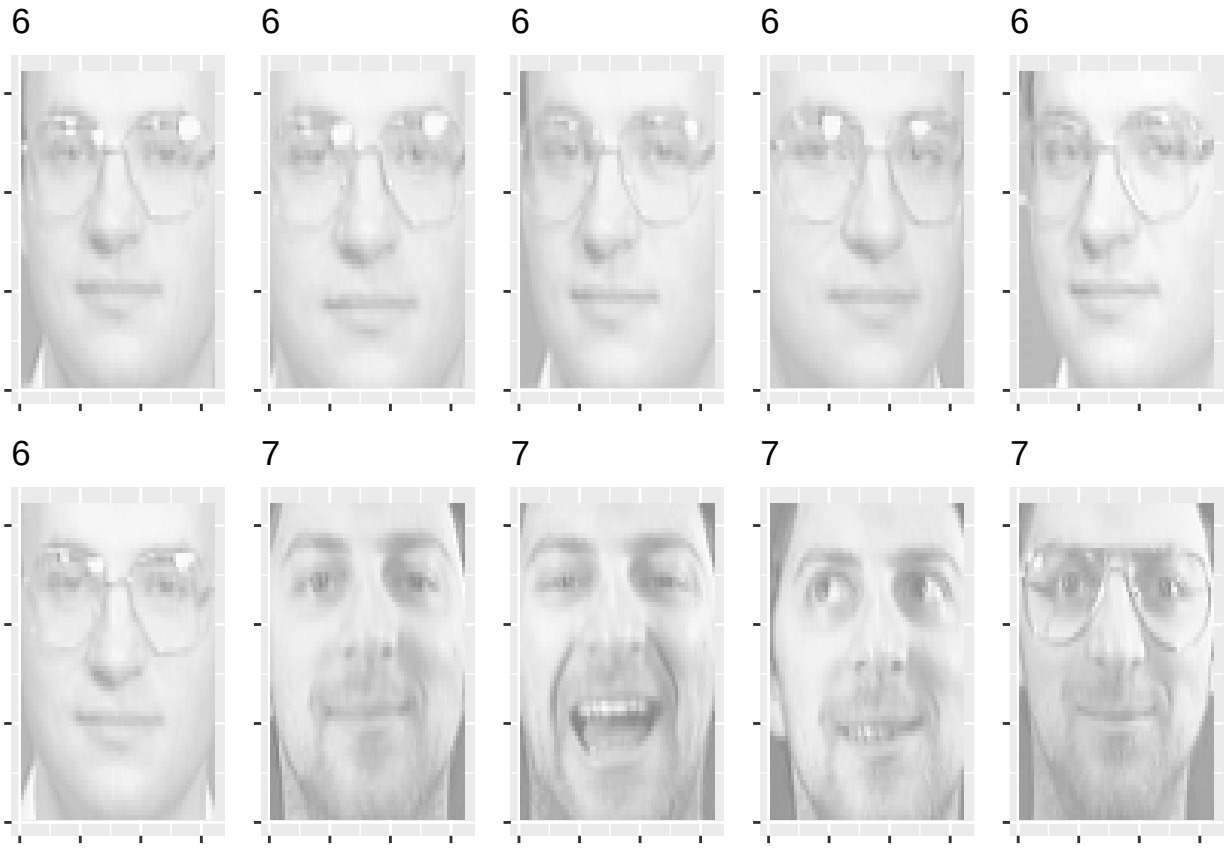
- `xx` is a matrix containing images as row vectors
- `width` of the image (64 pixels for this lab)
- `gcols` is the number of columns
- `midcolor` adjusts the brightness of the image
- `labels` the labels for the images

NOTES:

- Don't plot more than ten faces at a time because the function can be very slow.
- `faceplot()` uses `ggplot` and `geom_raster` to plot the heatmap; this requires the use of `melt` to change format of the data prior to ploting.
- The function `grid.arrange()` arranges `ggplot` objects in a grid.
- High values appear light and low values appear dark in the images.

The following code plots faces 55-64 in a grid with five columns. Each image is labeled with the Subject ID.

```
# Plot faces 55 though 64 with 5 images per row
faceplot(F.matrix[c(55:64),],
         gcols=5,
         midcolor="grey50",
         labels=labels[c(55:64)])
```

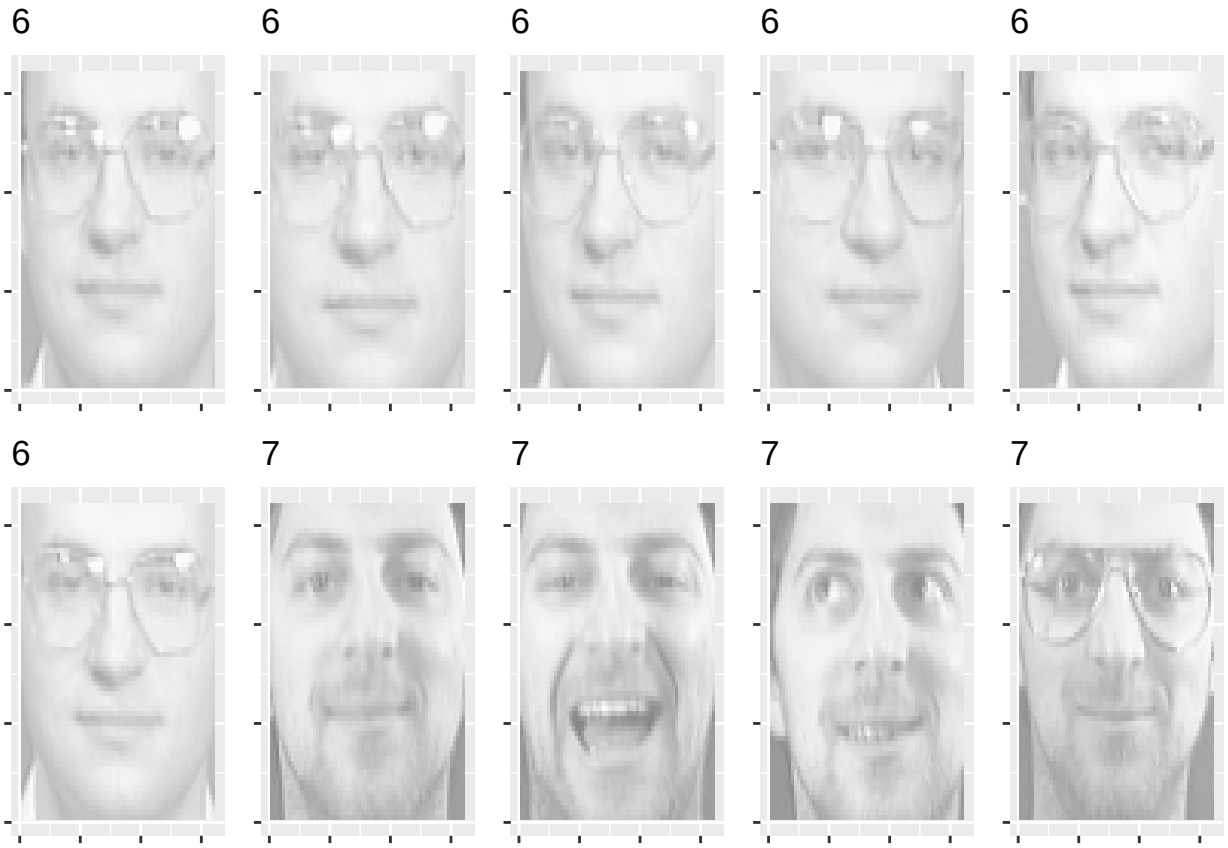


We can remove unwanted variations in lighting by scaling images so that they have unit length. The matrix Fbar.matrix contains the scaled images represented as vectors.

```
# Insert your answer here
rownorms <- apply(F.matrix, 1, function(x){sqrt(sum(x^2))})
Fbar.matrix <- diag(1/rownorms) %*% F.matrix
```

*** Try it: Check out how the faces change after scaling ***

```
# Plot scaled faces 55 though 64 with 5 images per row
faceplot(Fbar.matrix[c(55:64),],
         gcols=5,
         midcolor="grey50",
         labels=labels[c(55:64)])
```



Preparing images for analysis

In data analytics, frequently the goal is to develop a model to be applied on future data. For example, in this lab we are creating a PCA to compress data and do facial recognition. Thus we would like to evaluate how effective the model would be on future data. To do this, we divide the data into two disjoint sets called the *training set* and the *testing set*. We create the data analytics model based on the training set and then apply the model to the testing set.

In the following example, the training set consists of six images for each person and the testing set consists of four images for each person.

```
# Divide into `trainface` and `testface`
# Since images occur in blocks of 10, work mod 10 (using %%10) and get images 4-9 as train and 0-3 as t
# Create trainface
gg <- c(1:400)
h <- gg[ gg%%10 > 3]
# h is the list of numbers of faces to extract
trainface.uncentered<- Fbar.matrix[h,]
trainfacelabels<-labels[h]
# Create testface
gg <- c(1:400)
h <- gg[ gg%%10 <= 3  ]
# h is the list of numbers of faces to extract
testface.uncentered<- Fbar.matrix[h,]
testfacelabels<-labels[h]
```

Find the mean trainface and plot it

We will be using PCA, so we will center the data about the mean. In this case the mean is also a face!

Let's take a look at the mean face...

```
train.mean <- colMeans(trainface.uncentered)
faceplot(train.mean)
```

Center the training data

We center the training data using the mean of the training data. Write down the mathematics that is being done here. The command 'matrix(1,ncol=1,nrow=nrow(trainface.uncentered)) is making a column vector of ones.

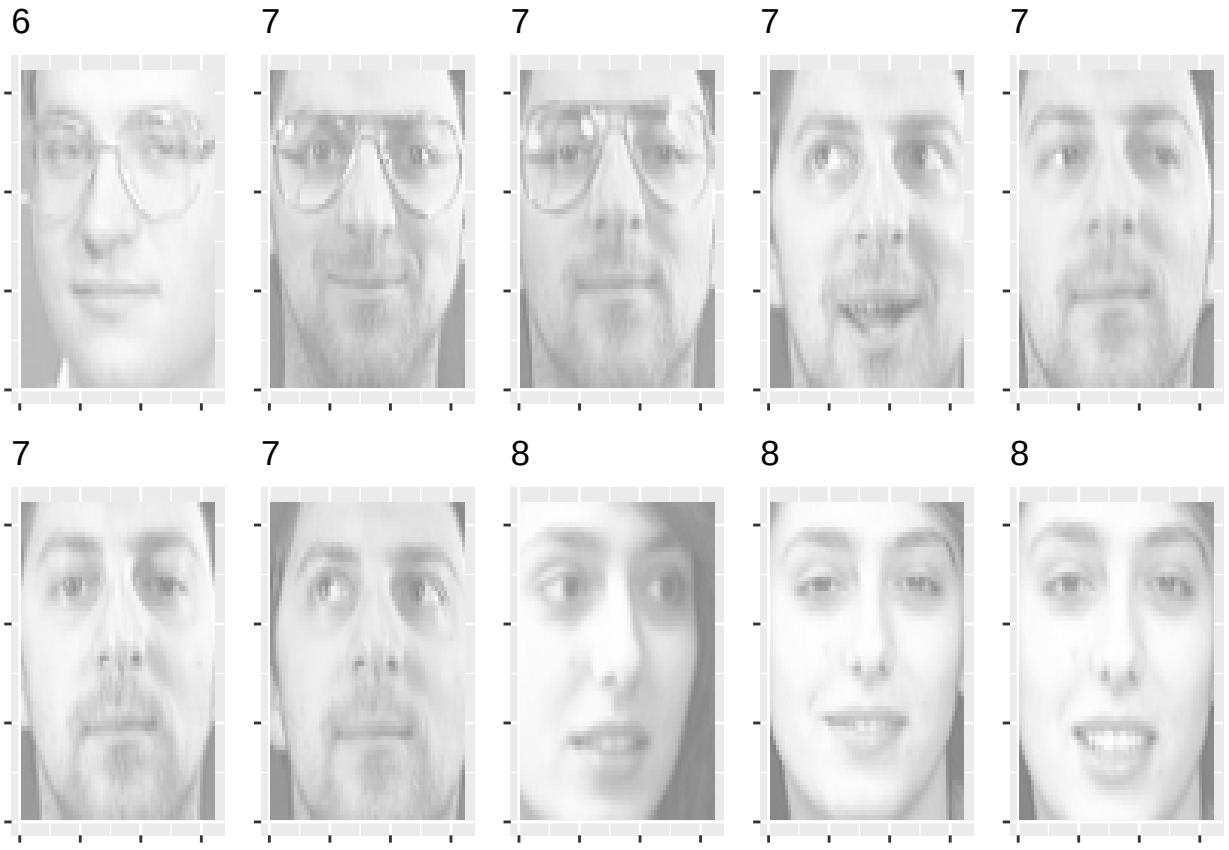
```
# Mean center the data and save in trainface.
trainface.centered <- trainface.uncentered-matrix(1,ncol=1,nrow=nrow(trainface.uncentered)) %*% train.m
```

Exercise2

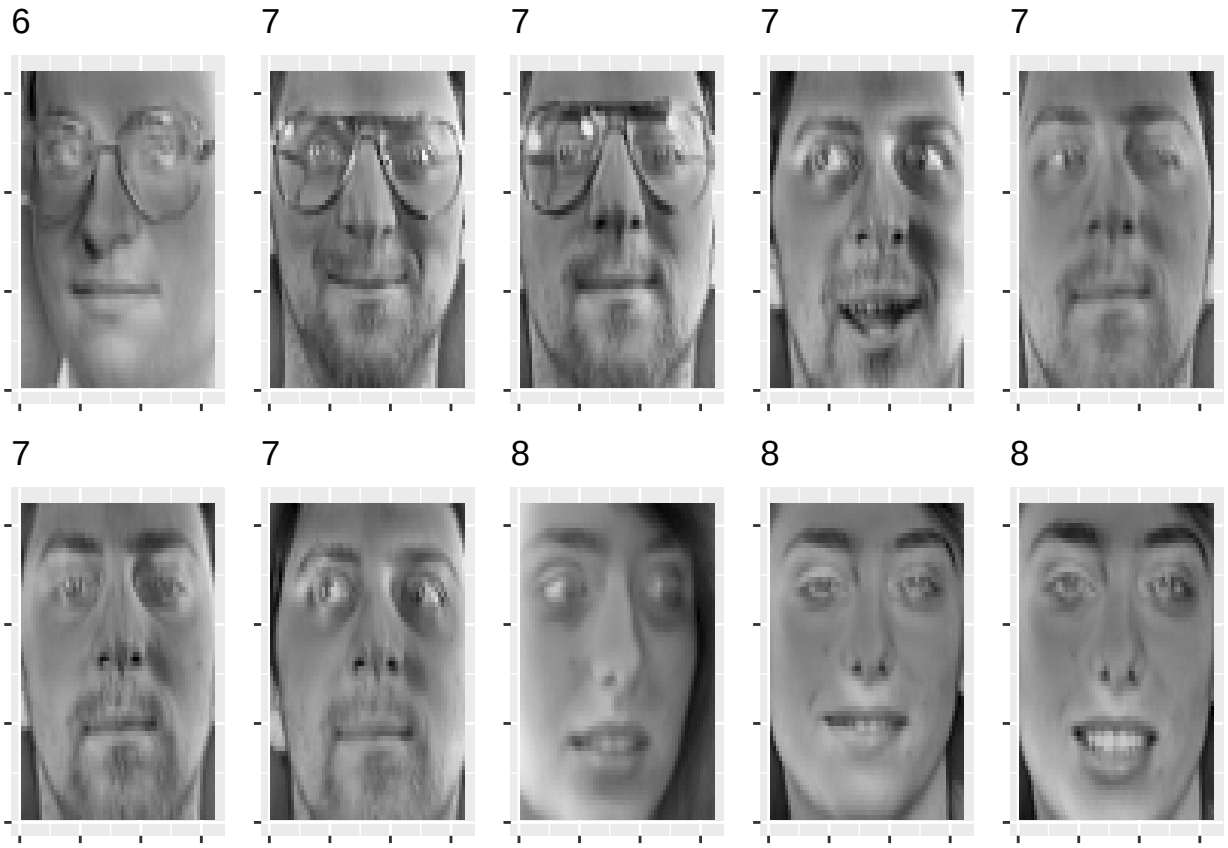
Plot training images 36-45 using `trainface.uncentered` and then again using `trainface.centered`.

```
# Insert your answer here
# Plot faces 55 though 64 with 5 images per row
imagenum<-c(36:45)
faceplot(trainface.uncentered[imagenum,],
         gcols=5,
         midcolor="grey50",
```

```
        labels=trainfacelabels[imagenum])
```




```
faceplot(trainface.centered[imagenum,],
         gcols=5,
         midcolor="grey50",
         labels=trainfacelabels[imagenum])
```



Describe how we should interpret the mean centered data. What does it mean when a region of pixels is light? What does it mean when a region is dark?

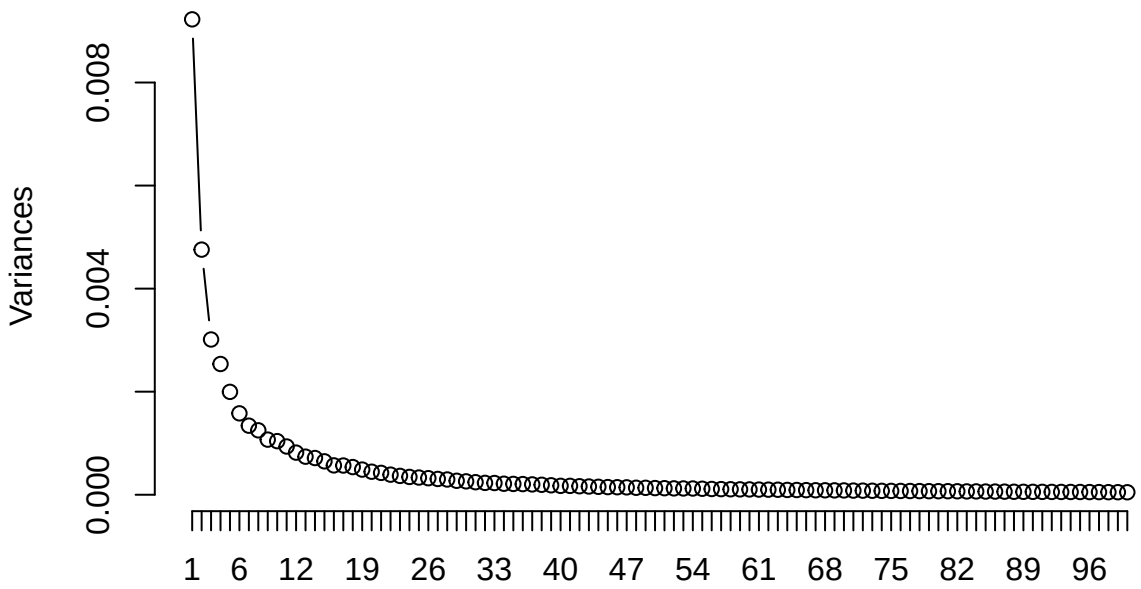
The mean centered data is tracking the differentiations from the mean, the mean image was removed. So it is missing a lot of the lighter skin around the faces, that is why they appear darker. When region is darker is means that it differs more from the mean.

Exercise3

Run PCA on `Fbar.matrix`. Plot a screeplot using the `npcs = 100` to set the number of components to show. *What is the minimum number of components necessary to capture 90% of the variance?*

Note: We used `head(t(summary(my.pca)$importance), n=100)` to see the summary of variance explained for first 100 principal components.

```
# Do the PCA on the centered training data with no additional scaling or centering
my.pca<- prcomp(trainface.centered,retx=TRUE, scale=FALSE,center=FALSE)
#print first 100 components
screeplot(my.pca,type="lines",main="Face Screeplot",npcs=100)
```




```
head(t(summary(my.pca)$importance),n=100)
```

```
##       Standard deviation Proportion of Variance Cumulative Proportion
## PC1          0.096050806                0.19291                0.19291
## PC2          0.068972112                0.09947                0.29239
## PC3          0.054892136                0.06301                0.35539
## PC4          0.050371741                0.05306                0.40845
## PC5          0.044685036                0.04175                0.45020
## PC6          0.039741842                0.03303                0.48323
## PC7          0.036606847                0.02802                0.51125
## PC8          0.035355763                0.02614                0.53739
## PC9          0.032702869                0.02236                0.55975
## PC10         0.032249193                0.02175                0.58150
## PC11         0.030596229                0.01957                0.60107
## PC12         0.028594502                0.01710                0.61817
## PC13         0.027177981                0.01545                0.63362
## PC14         0.026697470                0.01490                0.64852
## PC15         0.025466954                0.01356                0.66208
## PC16         0.023869094                0.01191                0.67400
## PC17         0.023813512                0.01186                0.68585
## PC18         0.023172569                0.01123                0.69708
## PC19         0.022102062                0.01021                0.70730
## PC20         0.021144767                0.00935                0.71665
## PC21         0.020599478                0.00887                0.72552
## PC22         0.019753499                0.00816                0.73368
## PC23         0.019119842                0.00764                0.74132
## PC24         0.018583844                0.00722                0.74854
## PC25         0.018262907                0.00697                0.75552
## PC26         0.017919733                0.00671                0.76223
## PC27         0.017499932                0.00640                0.76864
## PC28         0.017201794                0.00619                0.77482
## PC29         0.016466461                0.00567                0.78049
```

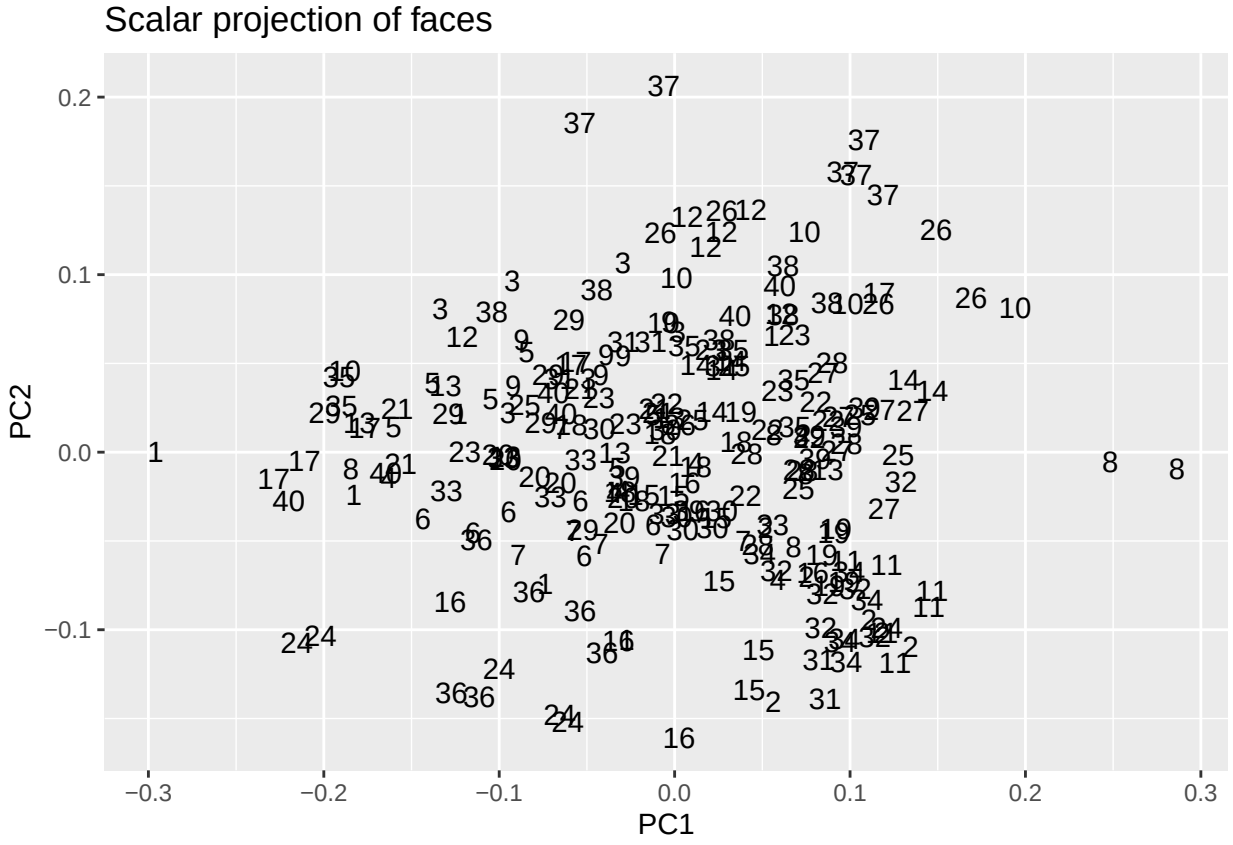
```
## PC30      0.016115214          0.00543          0.78592
## PC31      0.015591785          0.00508          0.79101
## PC32      0.015173757          0.00481          0.79582
## PC33      0.015088528          0.00476          0.80058
## PC34      0.014620030          0.00447          0.80505
## PC35      0.014493859          0.00439          0.80945
## PC36      0.014391390          0.00433          0.81378
## PC37      0.014169426          0.00420          0.81797
## PC38      0.013907532          0.00404          0.82202
## PC39      0.013558475          0.00384          0.82586
## PC40      0.013147796          0.00361          0.82948
## PC41      0.013122954          0.00360          0.83308
## PC42      0.012851871          0.00345          0.83653
## PC43      0.012631608          0.00334          0.83987
## PC44      0.012419992          0.00323          0.84309
## PC45      0.012181928          0.00310          0.84620
## PC46      0.012072458          0.00305          0.84924
## PC47      0.011972493          0.00300          0.85224
## PC48      0.011630375          0.00283          0.85507
## PC49      0.011595612          0.00281          0.85788
## PC50      0.011373179          0.00270          0.86059
## PC51      0.011268038          0.00265          0.86324
## PC52      0.011111854          0.00258          0.86582
## PC53      0.011044268          0.00255          0.86837
## PC54      0.010993283          0.00253          0.87090
## PC55      0.010781513          0.00243          0.87333
## PC56      0.010559841          0.00233          0.87566
## PC57      0.010438940          0.00228          0.87794
## PC58      0.010306238          0.00222          0.88016
## PC59      0.010217094          0.00218          0.88235
## PC60      0.010154684          0.00216          0.88450
## PC61      0.010008413          0.00209          0.88660
## PC62      0.009933039          0.00206          0.88866
## PC63      0.009853224          0.00203          0.89069
## PC64      0.009655081          0.00195          0.89264
## PC65      0.009598238          0.00193          0.89457
## PC66      0.009444001          0.00186          0.89643
## PC67      0.009330496          0.00182          0.89825
## PC68      0.009276366          0.00180          0.90005
## PC69      0.009211877          0.00177          0.90183
## PC70      0.009161566          0.00176          0.90358
## PC71      0.008990532          0.00169          0.90527
## PC72      0.008925981          0.00167          0.90694
## PC73      0.008794851          0.00162          0.90855
## PC74      0.008695619          0.00158          0.91013
## PC75      0.008632042          0.00156          0.91169
## PC76      0.008552721          0.00153          0.91322
## PC77      0.008512078          0.00152          0.91474
## PC78      0.008412194          0.00148          0.91622
## PC79      0.008316242          0.00145          0.91766
## PC80      0.008298020          0.00144          0.91910
## PC81      0.008287694          0.00144          0.92054
## PC82      0.008197079          0.00141          0.92194
## PC83      0.008039921          0.00135          0.92330
```

```
## PC84        0.008022113             0.00135             0.92464
## PC85        0.007886950             0.00130             0.92594
## PC86        0.007842964             0.00129             0.92723
## PC87        0.007730048             0.00125             0.92848
## PC88        0.007616297             0.00121             0.92969
## PC89        0.007547605             0.00119             0.93088
## PC90        0.007499953             0.00118             0.93206
## PC91        0.007462330             0.00116             0.93322
## PC92        0.007388917             0.00114             0.93436
## PC93        0.007350497             0.00113             0.93549
## PC94        0.007312236             0.00112             0.93661
## PC95        0.007294755             0.00111             0.93773
## PC96        0.007182916             0.00108             0.93880
## PC97        0.007135940             0.00106             0.93987
## PC98        0.007048236             0.00104             0.94091
## PC99        0.007007601             0.00103             0.94193
## PC100       0.006939980             0.00101             0.94294
```

Exercise4

Here we make a plot of the scalar projections of the data onto the first two principal components the label number representing each image.

```
# Insert your answer here
#Plot the scalar projections
ggplot(data=as.data.frame(my.pca$x),aes(x=PC1,y=PC2)) +
       geom_text(label=trainfacelabels) +
  ggtitle("Scalar projection of faces")
```



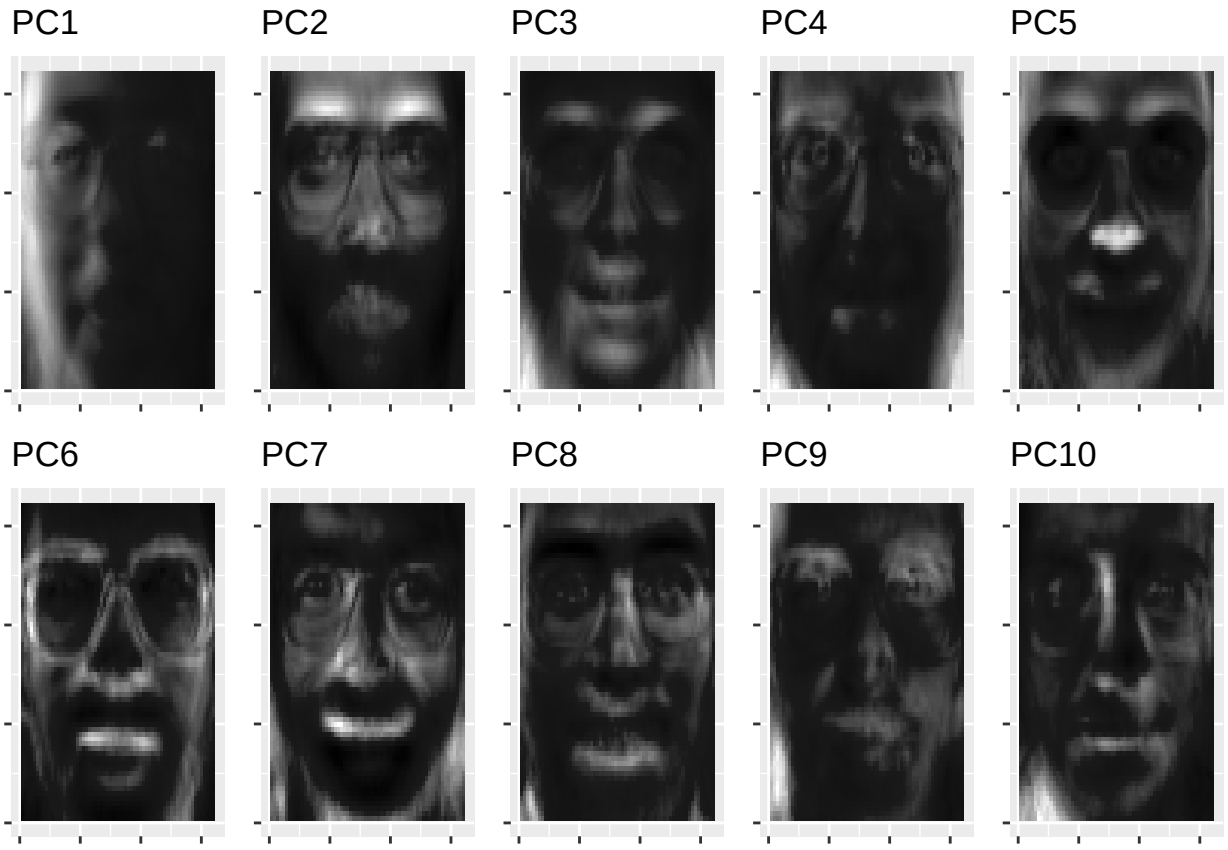
Do you see any relation between faces of the same people and the location of these points?

Yes specific people are grouped in the same region, this is becuase of similar facial features but also differences in lighting can throw it off.

Exercise5

Plot the first ten principal components as images. You will see why the principal components are known as *eigenfaces*. We also plot the centered image with the largest scalar projection on PC1 and the centered image with scalar projectiong on PC1. NOTE: which.max(v) finds the index of the largest element in vector v. which.min(v) finds the index of the smallest element in vector v.

```
# The code is provided.   Just look.
faceplot(t(my.pca$rotation[,1:10]),gcols=5)
```




```
# Plot the centered image with highest projection on PC1
maxpca<-which.max(my.pca$x[,'PC1'])
faceplot(trainface.centered[maxpca,],gcols=1,labels= trainfacelabels[maxpca])
```

8

```
# Plot the centered image with smallest projection on PC1
minpca<-which.min(my.pca$x[,'PC1'])
faceplot(trainface.centered[minpca,],gcols=1,labels= trainfacelabels[minpca])
```

What parts of the images should be light or dark to have a high scalar projection on PC1? Discuss what PC1 checks for in the image? The right part of the image should be darkened to get a higher value for PC1. It checks where the lighting is coming from, left of right.

Image recognition

WeCU's has a system for facial recognition that uses the "Nearest Neighbor" Algorithm (NN). NN works by taking an image, computing the closest point in the faces dataset, and then returning that closest point as the predicted label of the image. If the predicted label matches the true label of the person, then the system has correctly identified the person. You have been hired to improve their current system.

Here is the code of the current NN system, which uses the function `Matchimage()`. Examine it carefully.

```
#Function to compute 2-norm
norm2 <- function(x) sqrt(as.vector(x) %*% as.vector(x))
#Function to find closest match
Matchimage<-function(image.matrix,refimage)
  # image.matrix contains the imaages
  # refimage contains the image
  # returns the index of the closest imaige in image.matrix
  {
  dist2myimage<-apply(image.matrix,1,function(x)norm2(x-refimage))
  q <- which.min(dist2myimage)
}
```

Find the cousins

Let's pick a person in images 121-240. We will call the closest matching person in images 1-120 their "cousin." We'll find the match and then plot the original image on the left and the closest matching original image on the right.

Note that we have to add in the mean image to recover the original image. Let's try it for Person 145:

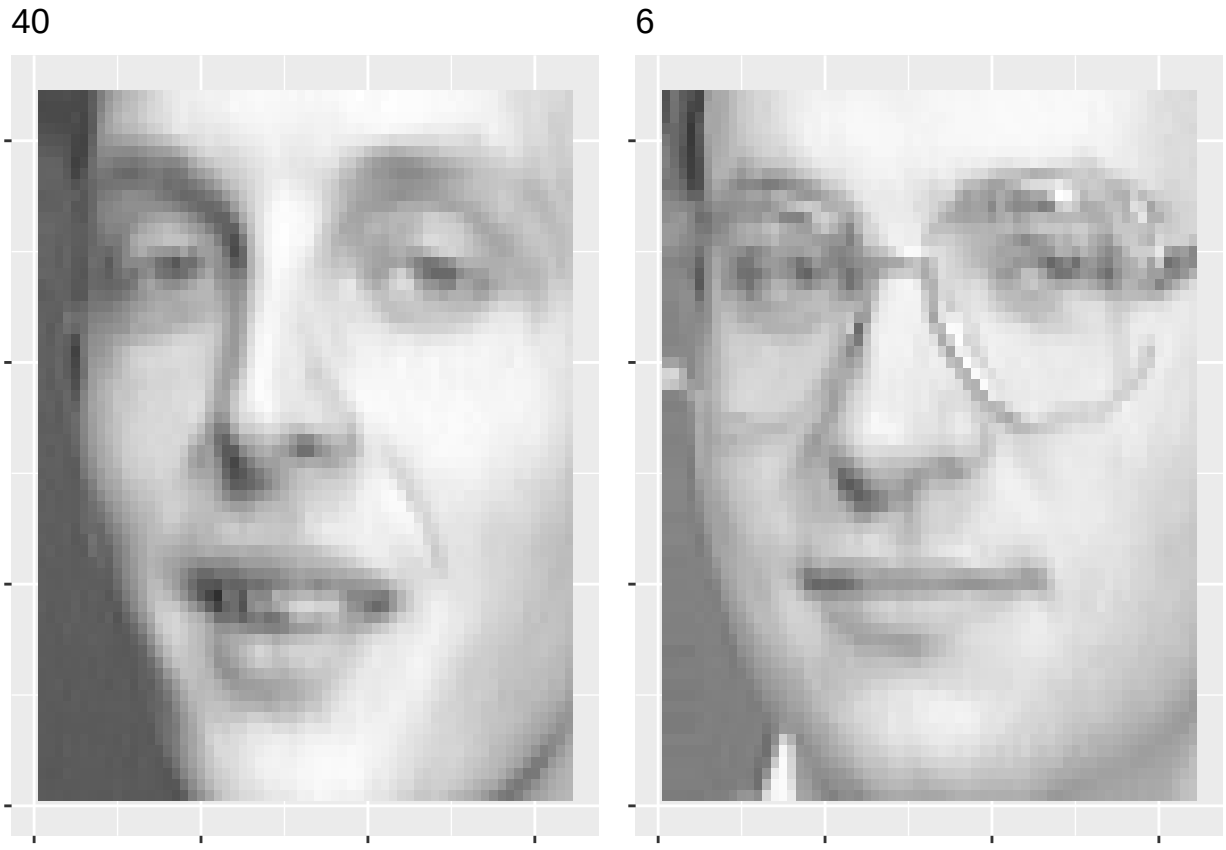
```
#Plot original and closest images
refimageind<-145
refimage<-trainface.centered[refimageind,]
matchind<-Matchimage(trainface.centered[1:120,],refimage)
faceplot(rbind(refimage+train.mean,trainface.centered[matchind,]+train.mean),gcols=2,
         labels=trainfacelabels[c(refimageind,matchind)])
```

*Exercise*6

- What is the label of the "cousin"" of Person 240?
- What happens if you try to find the cousin of Person 43? Why do you think this happens?

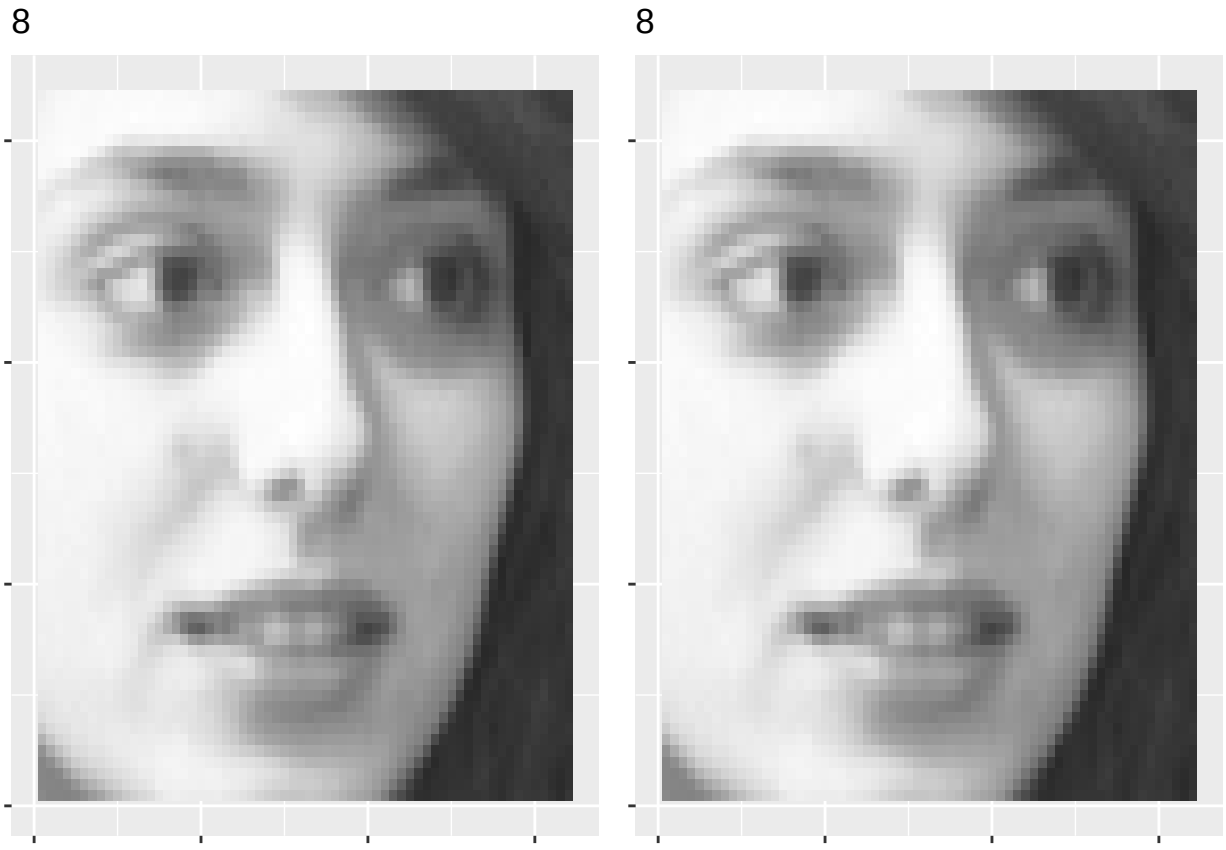
```
#Plot original and closest images for 240

refimageind<-240
refimage<-trainface.centered[refimageind,]
matchind<-Matchimage(trainface.centered[1:120,],refimage)
faceplot(rbind(refimage+train.mean,trainface.centered[matchind,]+train.mean),gcols=2,
         labels=trainfacelabels[c(refimageind,matchind)])
```




```
#Plot original and closest images for 43

refimageind<-43
refimage<-trainface.centered[refimageind,]
matchind<-Matchimage(trainface.centered[1:120,],refimage)
faceplot(rbind(refimage+train.mean,trainface.centered[matchind,]+train.mean),gcols=2,
         labels=trainfacelabels[c(refimageind,matchind)])
```



You've now completed Prelab6! Go to LMS and complete the online quiz.

‘‘