HW4

FUNDAMENTALS OF COMPUTATIONAL MATHEMATICS

Zachary Palmore

2/12/2021

Directions

With the attached data file, build and visualize eigenimagery that accounts for 80% of the variability. Provide full R code and discussion.

An example is provided in Rpubs titled "Eigen Shoes" by Larry Fulton which can also be accessed via this link: https://rpubs.com/R-Minator/eigenshoes

Preparations

To perform a principal component analysis (PCA) it is essential to have and understand the data in the analysis. In this section, the data, which in this case are images of the file format '.jpg', are read into our R environment. There are several considerations to make, such as how to extract the files and import them, what format we want the files in for the PCA, whether to scale the data to the same size, and more. This analysis follows the example set by the rpubs paper listed in the directions and begins by reading the images into an array.

The following packages were used during this analysis and should be installed prior to replication. Note that, if the package 'EBImage' is to be used, you will need to install 'BiocManager' from the Bioconductor package. Once installed, the function 'BiocManager::install should be used to install the desired package, 'EBImage' and its dependencies. This step is listed in the notes below.

```
library(jpeg)
library(EBImage) # Requires BiocManager::install('EBimage')
library(RCurl)
library(ggpubr)
library(gridExtra)
library(tidyverse)
```

Image Data

The files listed here were provided in a compressed zip folder and were downloaded and unzipped manually prior to use. This step was completed by hand. Once saved, the file could then be uploaded to a repository or other form of online storage and accessed. These steps were not taken here but could replace the "locale" variable shown. Instead, the files are accessed from the folder in which it was downloaded.

There are 17 total images. Each file is a .jpg type image of a shoe. To import the images into R, the 'readJPEG' function in the 'jpeg' package was used as demonstrated in the eigen shoes example. The names of the images within the file are shown.

```
n <- 17 # number of images present in folder
locale <- "C:/data/images/jpg" # location on computer
imgs <- list.files(locale,pattern="\\.jpg")[1:n] # list of image names
imgs
```

```
##  [1] "RC_2500x1200_2014_us_53446.jpg" "RC_2500x1200_2014_us_53455.jpg"
##  [3] "RC_2500x1200_2014_us_53469.jpg" "RC_2500x1200_2014_us_53626.jpg"
##  [5] "RC_2500x1200_2014_us_53632.jpg" "RC_2500x1200_2014_us_53649.jpg"
##  [7] "RC_2500x1200_2014_us_53655.jpg" "RC_2500x1200_2014_us_53663.jpg"
##  [9] "RC_2500x1200_2014_us_53697.jpg" "RC_2500x1200_2014_us_54018.jpg"
## [11] "RC_2500x1200_2014_us_54067.jpg" "RC_2500x1200_2014_us_54106.jpg"
## [13] "RC_2500x1200_2014_us_54130.jpg" "RC_2500x1200_2014_us_54148.jpg"
## [15] "RC_2500x1200_2014_us_54157.jpg" "RC_2500x1200_2014_us_54165.jpg"
## [17] "RC_2500x1200_2014_us_54172.jpg"
```

Notice that each file is named according to a pattern. There are two letters, presumably the image size, a year, the US abbreviation and a sequence of five numbers. For our purposes, the only numbers of interest in the names are the assumed image sizes which appear to all be identical in dimension.

Parameters

Deducing from the name of the photos and the example provided, the dimensions of the images are likely going to be the same. Given that each image is wider than its height, the smaller value of 1200 is listed as each image's height 'h' and the larger value 2500 as its width 'w' in the array. Additionally, these images will each need to be reduced in size. For this, the scale 's' is assigned the value of 20, which is used to create the arrays to fit plotting during visualizations.

Before extracting the images, an empty array with the same dimensions of our resultant array is created using the height, width, and scaling measures determined previously. The array is titled raster array or 'ra' for short. A variable named 'path' is assigned the file path to our local image files unique to this machine.

```
h <- 1200
w <- 2500
s <- 20
ra <- array(rep(0, length(imgs)*h/s*w/s*3), # creates placeholder array elements
            dim = c(length(imgs), h/s, w/s, 3)) # sets array dimensions
path <- "C:/data/images/jpg/"
```

This 'ra' array serves as a placeholder to convert and read in the images as points. Its dimensions must be set prior to conversion and in this case, the length of files present in the imgs list, serves as the quantity of places needed. The height and width variables of each image are scaled down by division of the variable by the set scale. With that, the parameters are set and there is a place for the image files to be stored. Now, there are only a few adjustments to make on the images as they are converted into the array.

Adjustments

The 'EBImage' package is used to fill the raster array with scaled raster image points for each file thereby reading the images into the R environment. This loop begins with the first image in the 'img' list and continues until the last image is processed at 'n' or the total number of files specified. For the directions provided, this is given at 17 images. Note that this 'readJPG' function is specific to .jpg images and will not perform without this format.

```
for (i in 1:n){
  rs <- resize(readJPEG(paste0(path, imgs[i])), h/s, w/s) # locates each file and adjusts h and w by th
  ra[i,,,] <- array(rs, dim = c(1, h/s, w/s, 3)) # Adds the image values to the placeholder array
}
```

For the visualization of the analysis, the example provided in the directions listed a function to plot each image as a separate object in its own space and display it at an appropriate resolution along with the other images in a matrix simultaneously. This is achieved through several steps but in short, it takes the images from their native file path, which in this case is localized to this computer, reads and adjusts the image size and resolution, applies those metrics to create an empty plot then converts the array of points into a raster image which fills the empty array and plots the results within a singular loop. This loop must be run through 'n' times for each image to appear in the plot, however, this would be quite difficult to replicate without changing the visuals. These shoe images are closely related to their original form, only resized and displayed as rasters.

```
# Example function for visuals
plot_jpeg = function(path, add=FALSE)
{ jpg = readJPEG(path, native=T) # read the file
  res = dim(jpg)[2:1] # get the resolution, [x, y]
  if (!add) # initialize an empty plot area if add==FALSE
    plot(1,1,xlim=c(1,res[1]),ylim=c(1,res[2]),asp=1,type='n',xaxs='i',yaxs='i',xaxt='n',yaxt='n',xlab=
  rasterImage(jpg,1,1,res[1],res[2])
}
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:n){  # adjust to number of files present in folder
plot_jpeg(writeJPEG(ra[i,,,])) # view shoes function applied here; plots the raster data
}
```









Analysis

In this section the principal components analysis is performed. Thankfully, R provides the functions and documentation to do so and once the images are in a practical form for computation, the steps of the PCA can begin. No packages are necessary, however, for visualization the 'tidyverse', 'gridExtra', and 'ggpubr' packages are utilized.

Basics

The purpose of a PCA is to explain as much of the variation between variables as possible with fewer variables. Our results should show a dataset that is reduced in size but, according to the directions, also accounts for 80% of the variability in the imagery. This is done to save space while maintaining a level of representation to the original source of data.

Conversion

To actually manipulate the data type as a set of raster points would be quite difficult given that, contrary to what the plot may indicate, there is little definition to the points in each image. Each image currently has within it an associated array of colors. These colors must be extracted and converted into points before further computation. The steps follow the given example excluding the need to set new parameters since this has already been established.

```
vecs <- matrix(0, n, prod(dim(ra)))
for (i in 1:n) {
  r <- as.vector(ra[i,,,1])
  g <- as.vector(ra[i,,,2])
  b <- as.vector(ra[i,,,3])
  vecs[i,] <- t(c(r, g, b))
}
pre <- data.frame(t(vecs))
```

The result returns a data frame that holds a point between zero and one for each column. The columns correspond to the points in a singular image from which the raster shoe was developed. Since the images are different, the set of vectors they hold are also unique to the set. Although, there is quite a bit of empty space, represented by the vector 1 in this array. The first four unique vectors for the first two images are displayed.

```
head(unique(pre[c(1:2)]), 4)
```

```
##            X1        X2
## 1   1.0000000 1.0000000
## 755 1.0000000 0.9990196
## 756 1.0000000 0.9725490
## 757 0.9441176 1.0000000
```

It becomes abundantly clear that these images are largely what the vector conversion considers blank space. This could be due to a small amount of variation throughout the colored point arrays of individual shoes and it will reduce overall variation across the images. In the visuals after the PCA, these will show as overlapping black dots, or the absence of a color in the image.

Looking at this quick view of the data frame, it is reasonable to expect quite a bit of black surrounding the shoes and to have very subtle variations of color in the images. This is determined by the first deviations. Starting at an index of one, variation from the vector of 1 does not occur until row 755 in the second image and even then it is a practically equivalent value of 0.99902.

For PCA purposes, these large blank spaces could be cropped out to reveal as much as possible of the variation in the shoes themselves rather than the lack of variation in the blank spaces surrounding the shoes. It would certainly enhance the ability of this PCA to explain the most variation in the images while also reducing the size of the data. However, for this analysis cropping is ignored in lieu of following the provided example.

PCA Steps

In attempting to explain the most variation between the images it would be cumbersome to evaluate each column of variables in the vector data set. There are 382,500 observations of 17 variables. According to the principle $p(p-1)/2 = x$ where p is the number of variables and x the quantity of analyses, to review them all by hand would require at least 136 different linear regression lines on scatter plots without including a method for combining and applying the information in an array. Thus, R has built-in functions that eliminate the need for these plots.

Since the variables have already been plotted in a two dimensional vector space the next steps are computing the normalized principal components to find where in the variables there is the highest variance. Importantly, this only considers X and the approach is completely unsupervised which means there is no correlation made between the X_1 and X_2. The process will result in the scores for the principal component which can be used to plot the results and is as follows:

```r
pre.cov <- cov(pre) # calculate covariance matrix for each
pre.eigens <- eigen(pre.cov) # find eigenvalues and eigenvectors of each matrix
str(pre.eigens)
```

```
## List of 2
##  $ values : num [1:17] 0.9347 0.1246 0.0458 0.0304 0.0249 ...
##  $ vectors: num [1:17, 1:17] -0.177 -0.326 -0.12 -0.295 -0.307 ...
##  - attr(*, "class")= chr "eigen"
```

```r
phi <- -(pre.eigens$vectors[,1:2]) # Inverse of loadings for PC1 and PC2
colnames(phi) <- c("PC1", "PC2")
PC1 <- as.matrix(pre) %*% phi[,1]
PC2 <- as.matrix(pre) %*% phi[,2]
PCs <- data.frame(row.names = NULL, PC1, PC2)
```

It begins with the calculation of the covariance for each matrix, then the eigenvalue and eigenvectors are found to determine most useful principal components. That is, the components of the vector array that explains the variation best for that particular X_n, which in this case is a set of points converted from an image. The loadings, or the elements that make up the principal component's maximum variance, are described by the eigenvectors. Due to the nature of the function in R to return negative ordered values, they are inverted, to create a display that fits positive principal components. This should assist in visualizing the space. Meanwhile, the loadings are used to find the first two principal component scores just to demonstrate. Using matrix multiplication, the inverted loadings are used to calculate the first and second principal components as shown. A final data frame containing the first two principal components named "PC1" and "PC2" respectively, is created. The first few rows are shown below.

```r
head(PCs)
```

```
##        PC1       PC2
## 1 3.934566 0.9676124
## 2 3.934566 0.9676124
## 3 3.934566 0.9676124
## 4 3.934566 0.9676124
## 5 3.934566 0.9676124
## 6 3.934566 0.9676124
```

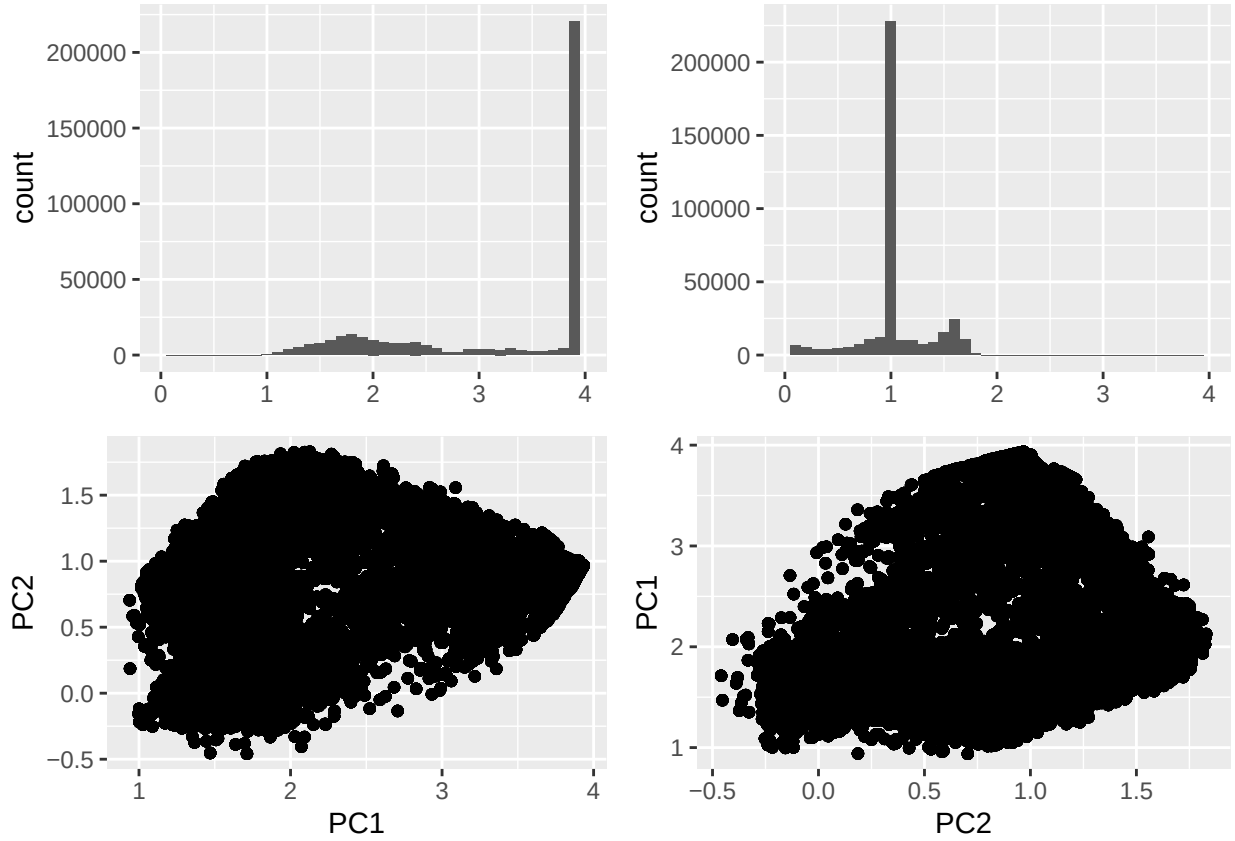
Some may find it useful to review these components before proceeding. Things to consider include the variation of the distribution and layout of two-dimensional points on the x-y plane. One can seek to describe how much variation there is, what stands out in the distribution, whether it is normal and if there are clear anomalies or outliers in the images. It might also be interesting to see how the components compare when plotted side-by-side. This may indicate what to expect in the image visualization and confirm some of the previous observations. For this, two histograms and two scatter plots were created to represent PC1 and PC2.

```r
PC1hist <- ggplot(PCs, aes(PC1)) + geom_histogram(binwidth = .1) + xlim(0, 4) + xlab(NULL)
PC2hist <- ggplot(PCs, aes(PC2)) + geom_histogram(binwidth = .1) + xlim(0, 4) + xlab(NULL)
PC2pnt <- ggplot(PCs, aes(PC2, PC1)) + geom_point()
PC1pnt <- ggplot(PCs, aes(PC1, PC2)) + geom_point()
ggarrange(PC1hist, PC2hist, PC1pnt, PC2pnt)
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```

```
## Warning: Removed 7803 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



Here again, it is quite easy to spot the anomalies. Since these components are the equivalent points that will be rendered as the images of the shoes, we can already begin to picture what it would look like based on this distribution. The first component will contain more color, probably closer to the white end of the spectrum since it has higher component values and most of its distribution is concentrated towards the highest end of the spectrum. Meanwhile, the second image will be darker by comparison since it is concentrated at the lower end of the spectrum.

With the components found, it is time to evaluate the results. This can be done using the eigenvalues of the original vectors to find the proportion of variance explained (PVE). It is given by the equation:

$$PVE = \frac{\sum \frac{n}{i=1} * (\sum \frac{p}{j=1} \phi_{jm} x_{ij})^2}{\sum \frac{p}{j=1} \sum \frac{n}{i=1} x_{ij}^2}$$

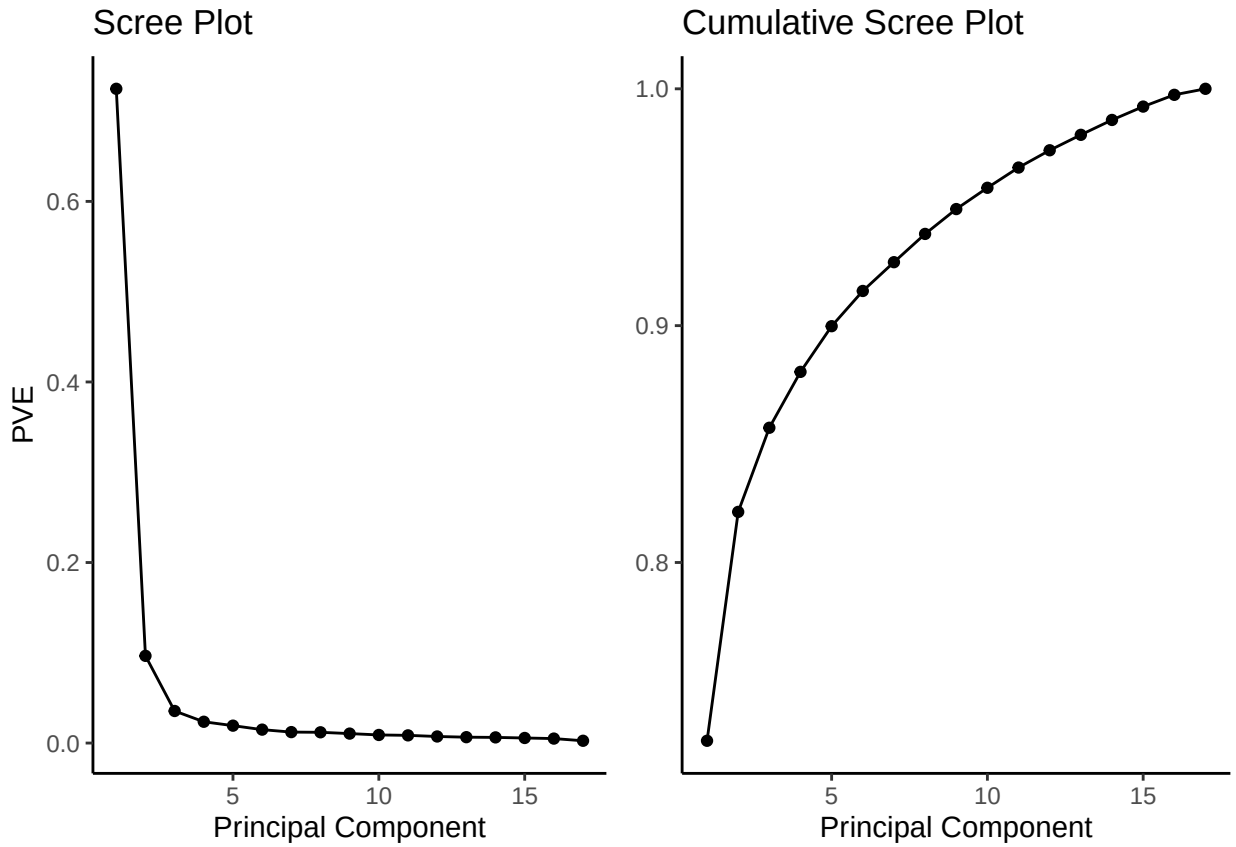
Where the proportion of variance explained by the mth principal component is found by dividing the eigenvalue by the total eigenvalues in the data set. This will produce a fraction that can be expressed as a percentage of the whole in this particular PCA.

```
PVE <- pre.eigens$values / sum(pre.eigens$values)
round(PVE[1:2], 3)*100
```

```
## [1] 72.5  9.7
```

Thus, in this example, the first component (PC1), explains about 69.3% of the variability in the original data. Following the same pattern, the second component (PC2) in this example, explains about 10.1% of the variability in the original data. When considering how many components to use, the same pattern can be applied to chart each PVE value in a manner that is consistent with the values per principal component, and the cumulative sum of those same values.

```
PVEplot <- qplot(PVE, c(1:17)) +
  geom_line() + ggtitle("Scree Plot") + xlab("PVE") +
  ylab("Principal Component") + coord_flip() + theme_classic()
cPVE <- qplot(c(1:17), cumsum(PVE)) +
  geom_line() +
  ggtitle("Cumulative Scree Plot") +
  xlab("Principal Component") +
  ylab(NULL) + theme_classic()
grid.arrange(PVEplot, cPVE, ncol = 2)
```

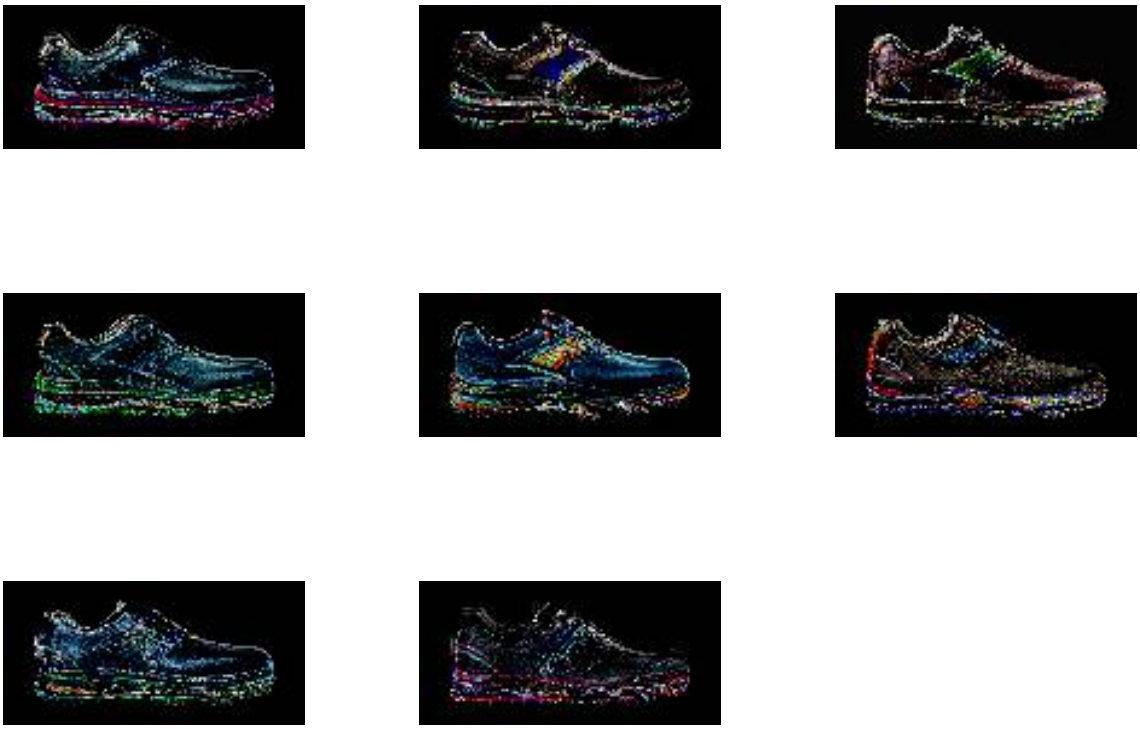


These plots are known as scree and cumulative scree plots. Generally, the best number of principal components to use is dependent upon how much variation you need to have explained and the amount of data you can store. With the purpose of PCA in mind, the point on the graph where there is a large enough significant decrease from the rest of the points on each plot indicates the number of principal components to use. In our case, the first two were used and we can assume based on these plots that it is a reasonable choice to make if the intent is to explain about 80% of the variability in the imagery. This is sometimes called the 'elbow' of the plot. These will reappear in the review as they are quite practical when evaluating how many components to use.

PCA Results

To give an idea of what the result should look like, the example is written below using slight modifications to suit the needs of the function. For reference, this included altering the raster array, the path to the images, the quantity of images, their sizing and scaling. Other than these, the example computation remains as given. For display and comparison purposes, the plot sizes were adjusted to match that of the previous plot.

```
# Example results using princomp in 'stats'
newdata=ra
dim(newdata)=c(length(imgs),h*w*3/s^2)
mypca=princomp(t(as.matrix(newdata)), scores=TRUE, cor=TRUE)
mypca2=t(mypca$scores)
dim(mypca2)=c(length(imgs), h/s, w/s,3)
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:n){#plot the first 25 Eigenshoes only
plot_jpeg(writeJPEG(mypca2[i,,,]))  #complete without reduction
}
```



Given these results we can replicate the process performed in the PCA steps but much quicker since the scores of the principal components were computed in the prcomp function demonstrated above. They reside in the variable 'x' within the array titled 'prcomp_results'. In this example, the matrices are reduced to determine the scores of each vector array and the results are stored in a matrix. As with the example, the resultant matrix is inverted to fit within the example function to provide an array of images in plot form. However, this method differs in its choice of function and fundamental calculation. These slight differences are compared to the function 'princomp' used in the example from the 'stats' package. In this analysis, that package is also necessary.

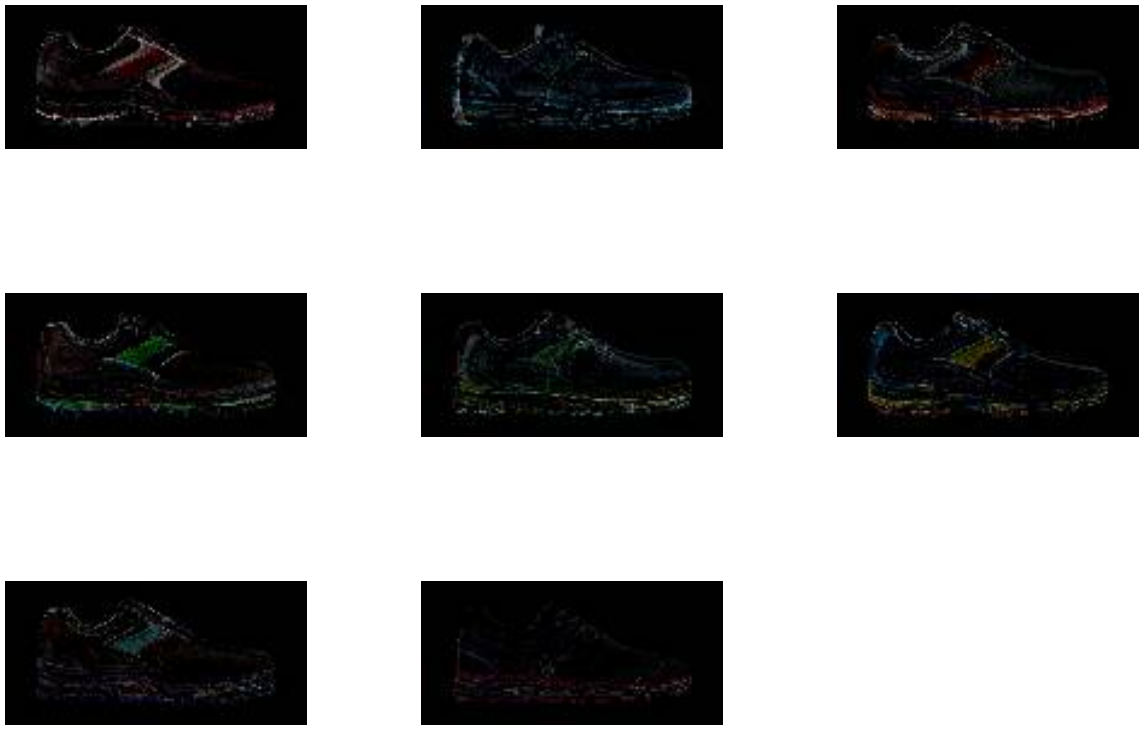
```
va <- ra
dim(va) <- c(n, h*w*3/s^2)
princomp_results <- princomp(t(as.matrix(va)), scores=TRUE, cor=TRUE)
prcomp_results <- prcomp(t(as.matrix(va)))
princomp_results_scores_3 <- head(princomp_results$scores, 3)
princomp_results_scores_3 == head(prcomp_results$x, 3)
```

```
##      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
## [1,]  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
## [2,]  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
## [3,]  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   FALSE
##      Comp.11 Comp.12 Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
## [1,]   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## [2,]   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## [3,]   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
```

Although all results were verified and there are no matches to the example data, only the first three of both function types are shown with each matrix. From there, the results can be visualized using the same

'readJPG' function given in the directions. The same dimensions given in the function must be used, otherwise, a new function is necessary to plot the rasters. The display size is adjusted to match the rest of the results.

```
vispca_results <- t(prcomp_results$x)
dim(vispca_results)=c(n, h/s, w/s,3)
par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:n){#plot the first 25 Eigenshoes only
plot_jpeg(writeJPEG(vispca_results[i,,,]))  #complete without reduction
}
```



This is one way to visualize the shoes using raster arrays. As expected, the image is similar to the example provided in the directions. However, the focus in this analysis was to attempt to have the first shoe show with a bit more detail. Unfortunately, this detracted a bit of detail from the other shoes since, this first shoe has one of the largest variations from the other shoes within the data set.

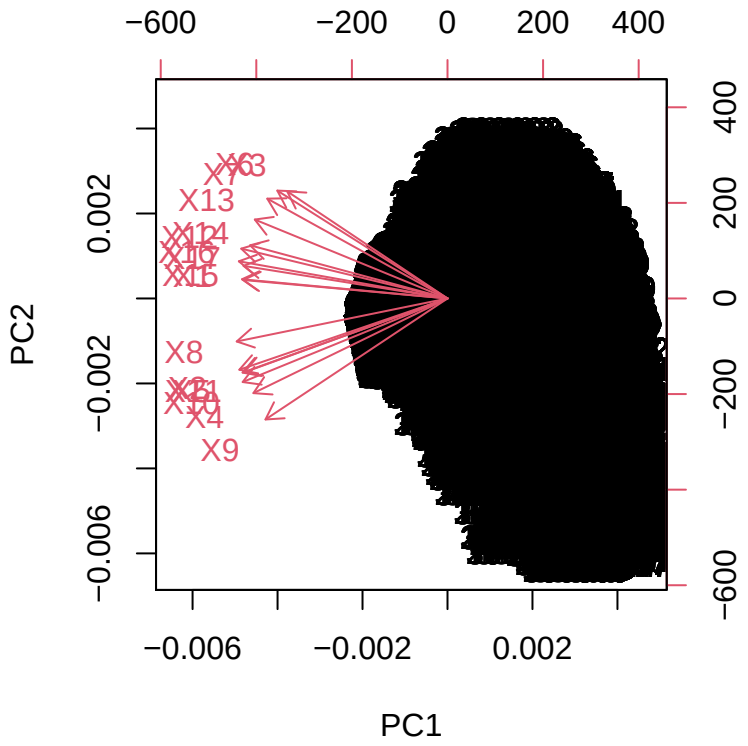
Our expectations were confirmed when the data displayed mostly black since from image to image, there is little to no variation in the vectors that make up this color. It was also reasonable to justify from the distributions what the first two shoes would look like. This method could be applied to estimating specific values in larger data sets, such as those with image recognition algorithms, by computing the counts of known colors represented by raster points which may save time in estimations.

Another angle to this image recognition through reduction might be to assign a pixelated array to the image where there are noticeable variations in the image data. For this, it may work best to assign the shoe images (where variation occurs) a unique array of colors that correspond to the image file or vector data itself. Although it would not capture the full image, PCA does not either. When storage of data, image size, scale, and quality are significant issues, and only the relationship between the data and its counterparts in a system of components are necessary, such a method could be convenient.

To demonstrate what this might look like, one might repeat the process of the PCA. This is because such a method would require an understanding of the variation between arrays of image data. The same steps are recompiled and reapplied to a biplot to assess where the principal components influence the results.

```
post <- prcomp(pre, scale = TRUE)
post$rotation <- -post$rotation
post$x <- t(as.matrix(post$x))
postx <- (as.matrix(post$x))
postx <- post$x
```

```
pcabi <- prcomp(pre, scale. = TRUE)
biplot(pcabi)
```



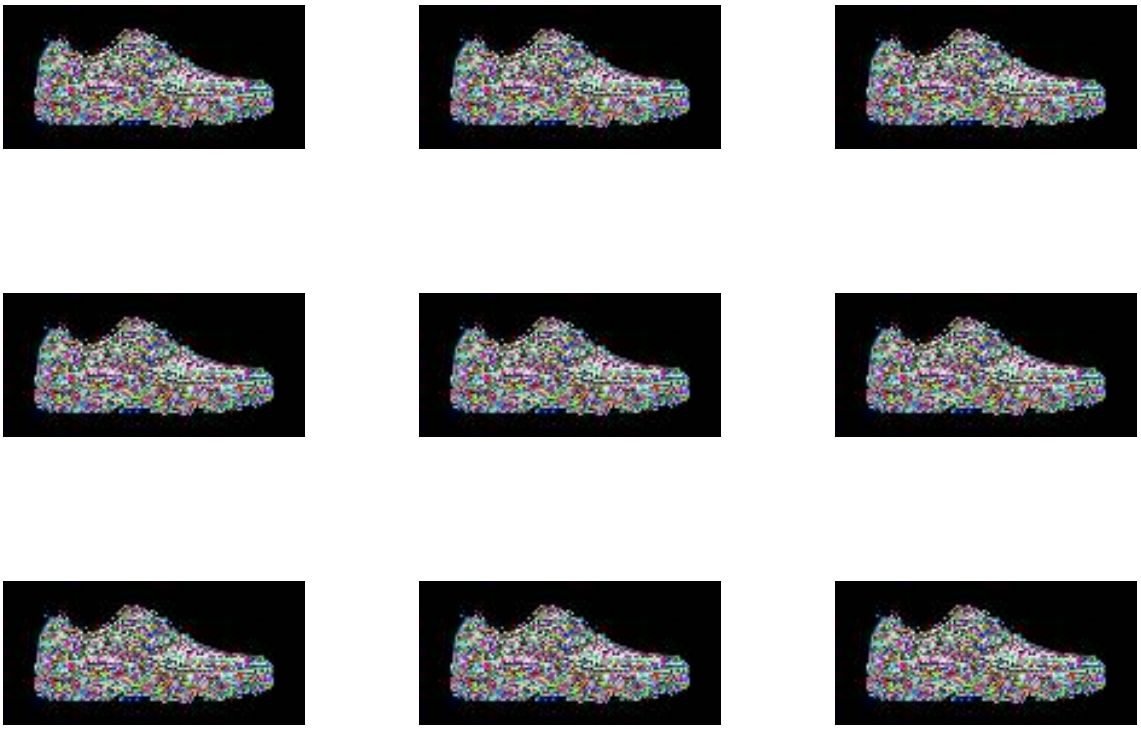
This plot provides the direction and magnitude of influence each variable exerts on the surrounding variables. The red arrows are error lines represented in scale by loadings. It assesses only the first two principal components because our assessment determined the first two were best for this PCA based on our goal. The direction is clearly left and the magnitude of PC2's variables on PC1's are at about the same strength.

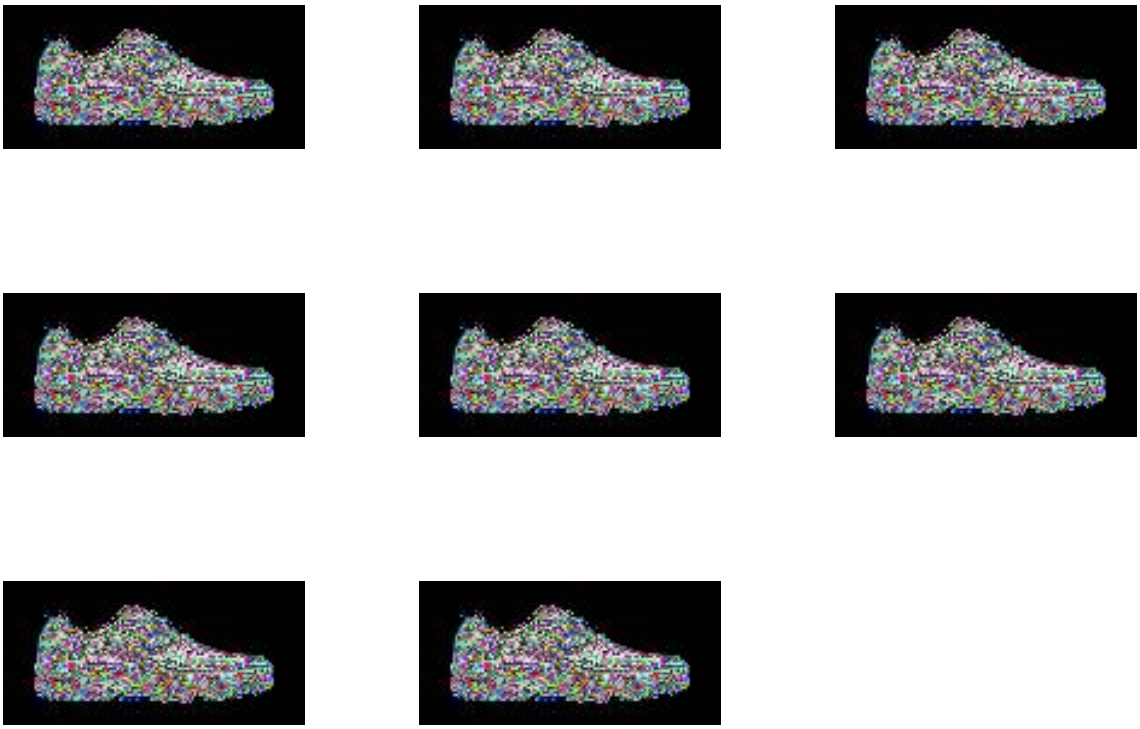
Rather than delving further into the review of the relationship between these principal components, these values in the 'postx' which represent the principal component scores from the 'prcomp' function, are plotted. The same plotting function is used and the process repeated except this time, all the points will simply be represented as raster colors.

```
pca <- array(rep(0, n*h/s*w/s*3), # creates placeholder array elements
             dim = c(n, h/s, w/s, 3)) # sets array dimensions

for (i in 1:n){
  vs <- resize(readJPEG(paste0(path, imgs[i])), h/s, w/s) # locates each file and adjusts h and w by th
  pca[i,,,] <- array(t(as.matrix(postx)), dim = c(1, h/s, w/s, 3)) # Adds the image values to the place
}

par(mfrow=c(3,3))
par(mai=c(.3,.3,.3,.3))
for (i in 1:n){  # adjust to number of files present in folder
plot_jpeg(writeJPEG(pca[i,,,], bg = "white")) # view shoes function applied here; plots the raster data
}
```





It would be interesting to see how raster points like this could be used to identify image characteristics or be used to translate between databases unique features to the images or dataset. For example, in wildlife biology there are far too few individuals to accurately identify individuals of a species. Yet, understanding whether the individual is the same fauna previously seen, is often at the forefront of assessments about the entire species and their behavior. However, this is now beyond the scope of this assignment. Perhaps this could occur in the future.

Review

Lastly, this section contains a brief review of what decisions were made for the PCA and why. It considers how the conclusions were made and confirms some basic assumptions made. At this point, the principal components are a new data set that could be exported and used for other purposes.

Evaluate PCs

As previously mentioned, there is no default method of deciding how many principal components to use and doing so requires a deeper understanding of the problem. For this assignment, the directions stated to "build and visualize eigenimagery that accounts for 80% of the variability." To find out what proportion of the variance was explained by this analysis the PVE was found.

```
VE <- post$sdev^2
PVE <- VE / sum(VE)
PVEstat <- round(PVE, 3)[1:2]*100
VE2 <- prcomp_results$sdev^2
PVE2 <- VE2 / sum(VE2)
```

```
PVE2stat <- round(PVE2, 3)[1:2]*100
PVEstat
```

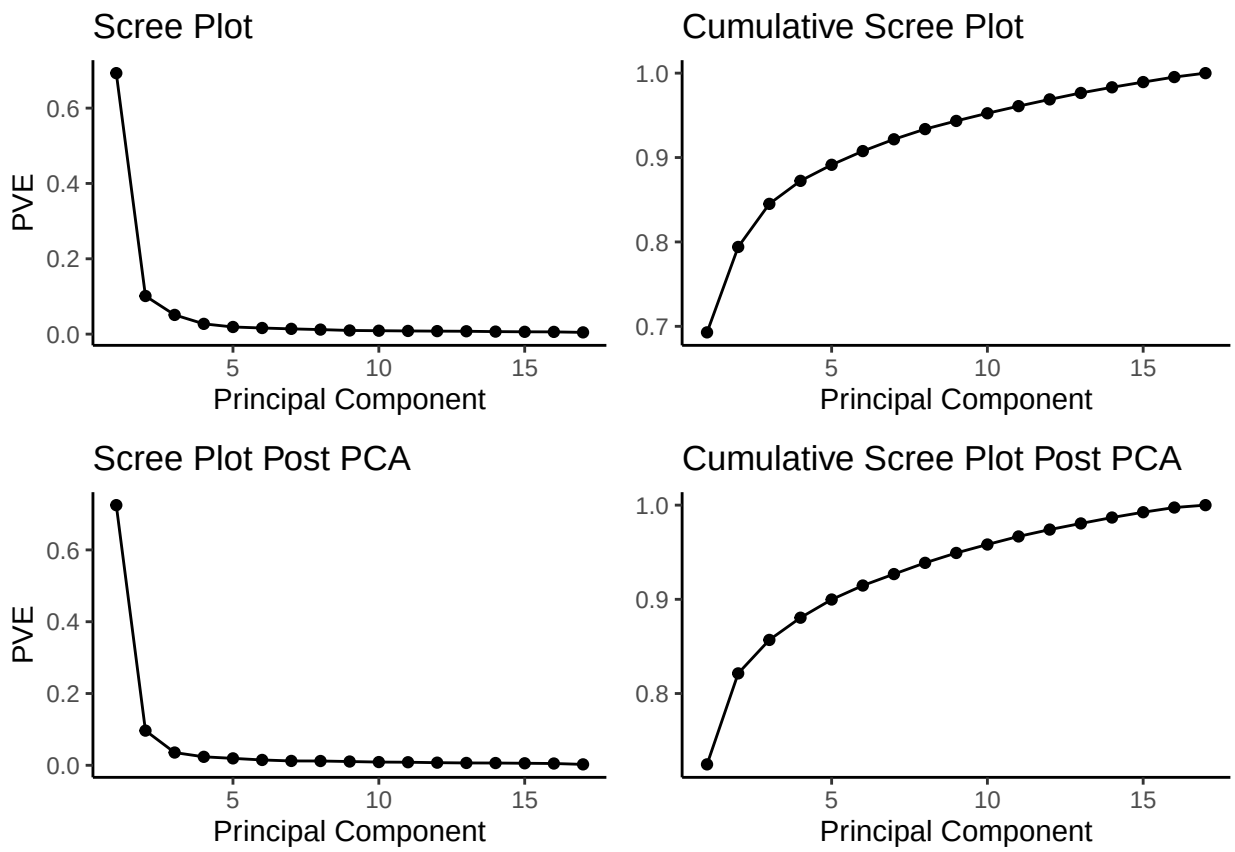
```
## [1] 69.3 10.1
```

```
PVE2stat
```

```
## [1] 72.5  9.7
```

Based on this calculation, the imagery created from our 'prcomp' function accounts for 82.2% of the variation. This goal is satisfied. It is also possible to review a scree plot and cumulative screen plot as shown previously to confirm this. These are shown below. Since the largest change in variation occurs at the second principal component only the first two were chosen to represent the data.

```
PVEplot2 <- qplot(PVE2, c(1:17)) +
  geom_line() + ggtitle("Scree Plot Post PCA") + xlab("PVE") +
  ylab("Principal Component") + coord_flip() + theme_classic()
cPVE2 <- qplot(c(1:17), cumsum(PVE2)) +
  geom_line() +
  ggtitle("Cumulative Scree Plot Post PCA") +
  xlab("Principal Component") +
  ylab(NULL) + theme_classic()
grid.arrange(PVEplot, cPVE, PVEplot2, cPVE2, ncol = 2)
```



Confirmation

As shown above, the 'elbow' of the scree plots could be seen at the second principal component. The values of PVE were calculated to be 72.5% and 9.7% for PC1 and PC2. Thus, our results capture approximately 82.2% of the original image data. Results in imagery using the same function to display the images used in the example, appear similar. This is a good sign that expectations were achieved. In future PCA of images, it might be best to first crop the image space to show only the shoes allowing the analysis to focus more on the variation between shoes rather than the blank space surrounding it. This may assist in reducing the amount of data while simultaneously providing a better explanation for it.