Data 605 HW 4

Jean Jimenez

2023-09-21

Homework 4

Question

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

The attached data file contained a folder with .jpg images of shoes.

Work

Preparing the data

For this homework, I am using the imager package which has some useful functions for image processing. After loading the package, and making sure the /jpg folder is in the current working directory, I obtained all of the image file names. The shoes_list line loads each of the images into this list. Then, I get the dimensions of the first image. The dimensions of the first image is 2500 x 1200 x 1 x 3. We have a 4th dimension that we weren't expecting so I got rid of the extra one. Finally, I made an empty matrix where the flattened image is going to be placed. Each column will be an image, while each row is the pixel's value for all images.

```
library(imager)
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'imager'
```

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

```
## The following objects are masked from 'package:stats':
##
##     convolve, spectrum
```

```
## The following object is masked from 'package:graphics':
##
##     frame
```

```
## The following object is masked from 'package:base':
##
##     save.image
```

```
#get the list of image files in the 'jpg' folder

shoe_paths = list.files(path = "./jpg", pattern = "\\.(jpg|jpeg)$", full.names = TRUE)

#read images into list
shoes_list = lapply(shoe_paths, function(file) load.image(file))

#check dims of the first image to set up the data matrix

first_shoe = shoes_list[[1]]

#remove extra dimension

first_shoe = drop(first_shoe)
dims = dim(first_shoe)
n_rows = dims[1] * dims[2] * dims[3]

# make empty matrix to store flattened images

shoe_data = matrix(nrow = n_rows, ncol = length(shoes_list))
```

Transform the Data

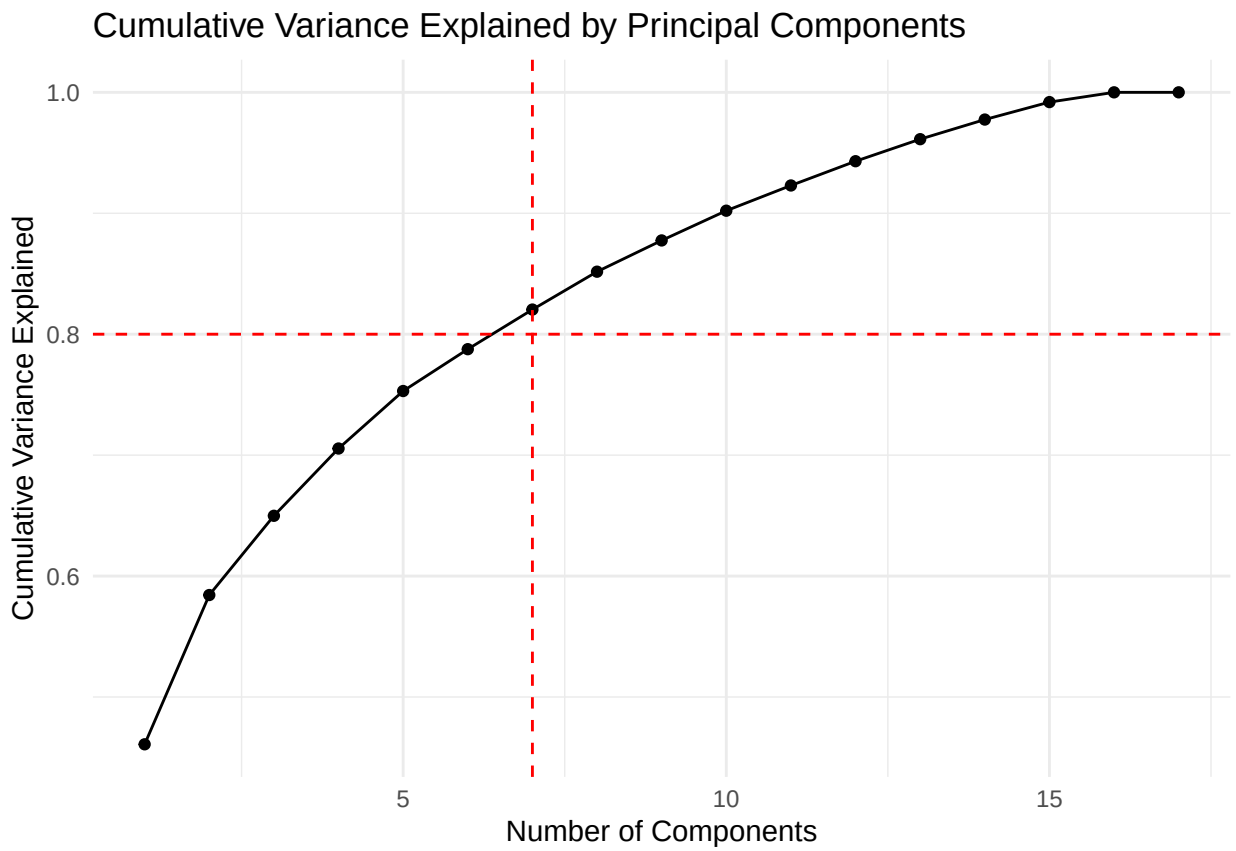
Next I start to flatten the images and placing them in the empty matrix established above. The loop does just that one image at a time. I was getting an error so I had to add a clause to make sure that the dimensions of the images where all the same. Afterwards, I mean centered the data, where the mean of each row (rowMeans function) is subtracted from each value in that row (sweep function).

```
# flatten each image and store it as a column in the matrix

for (i in seq_along(shoes_list)) {
  img = shoes_list[[i]]
  img = drop(img)
  img_dims = dim(img)

  # Check if the image has the same dimensions as the first image

  if (all(img_dims == dims)) {
    shoe_data[, i] = as.vector(img)
  } else {
    warning(paste("Skipping image at index", i, "due to different dimensions."))
  }
}

# mean center the data

mean_shoes = rowMeans(shoe_data, na.rm = TRUE)
centered_shoe_data = sweep(shoe_data, 1, mean_shoes)
```

Dimension Reduction

For the dimension reduction, I decided to use Principal Component Analysis (PCA) instead of SVD. Both PCA and SVD can be used to reduced dimensions. I chose PCA because I am familiar with it and everyone else was using SVD. To conduct the PCA, I used the function 'prcomp'. I set the center value to false because I already mean-centered the data above. PCA automatically mean-centers data unlike SVD. The variance from each principal component is in 'pca_result$sdev^2'. I calculated the cumulative sum of the variance and normalized it. Afterwards, I calculated the number of components needed for 80% variance (num_components) by finding the max in variance when at 80%. Finally, I extracted the eigenvectors associated with each of the principal components to 'eigenimages'. I then plotted number of components vs variance. At for 80% variance I get 7 components.

```
# perform pca

pca_result = prcomp(t(centered_shoe_data), center = FALSE)

# calc cumulative proportion of variance explained

variance_explained = cumsum(pca_result$sdev^2) / sum(pca_result$sdev^2)

# find # of components accounting for 80% of variance

num_components = which.max(variance_explained >= 0.8)

# get top eigenvectors (eigenshoes)
eigenimages = pca_result$rotation[, 1:num_components]




# creat a data frame for plotting num of components vs variance

plot_data = data.frame(
  Num_Components = 1:length(variance_explained),
  Variance_Explained = variance_explained
)

#plot
library(ggplot2)

ggplot(plot_data, aes(x = Num_Components, y = Variance_Explained)) +
  geom_line() +
  geom_point() +
  geom_hline(yintercept = 0.8, linetype = "dashed", color = "red") +
  geom_vline(xintercept = num_components, linetype = "dashed", color = "red") +
  labs(title = "Cumulative Variance Explained by Principal Components",
       x = "Number of Components",
       y = "Cumulative Variance Explained") +
  theme_minimal()
```

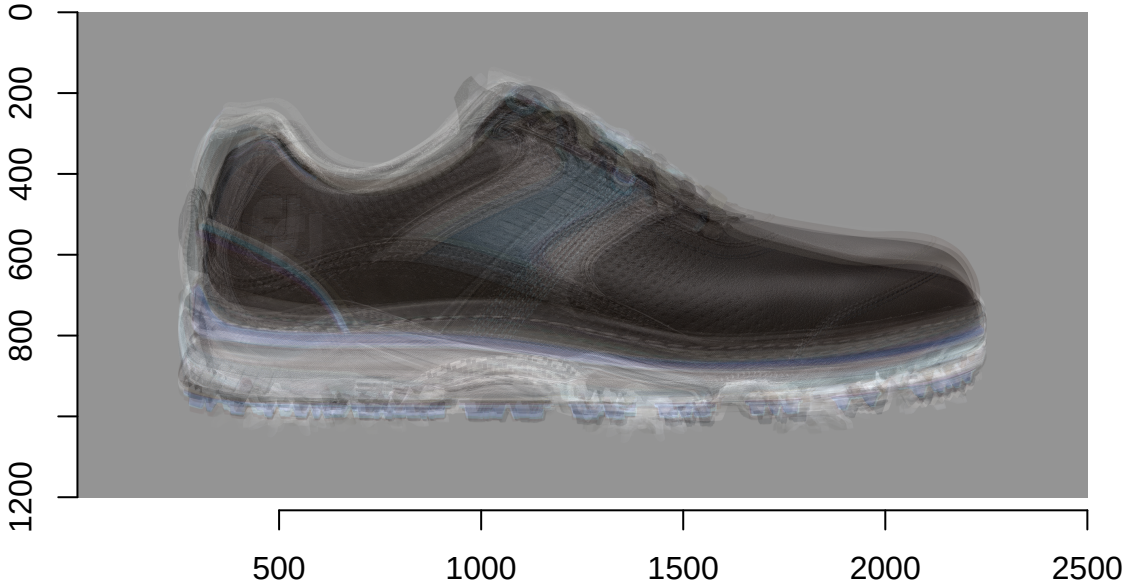


Visualization

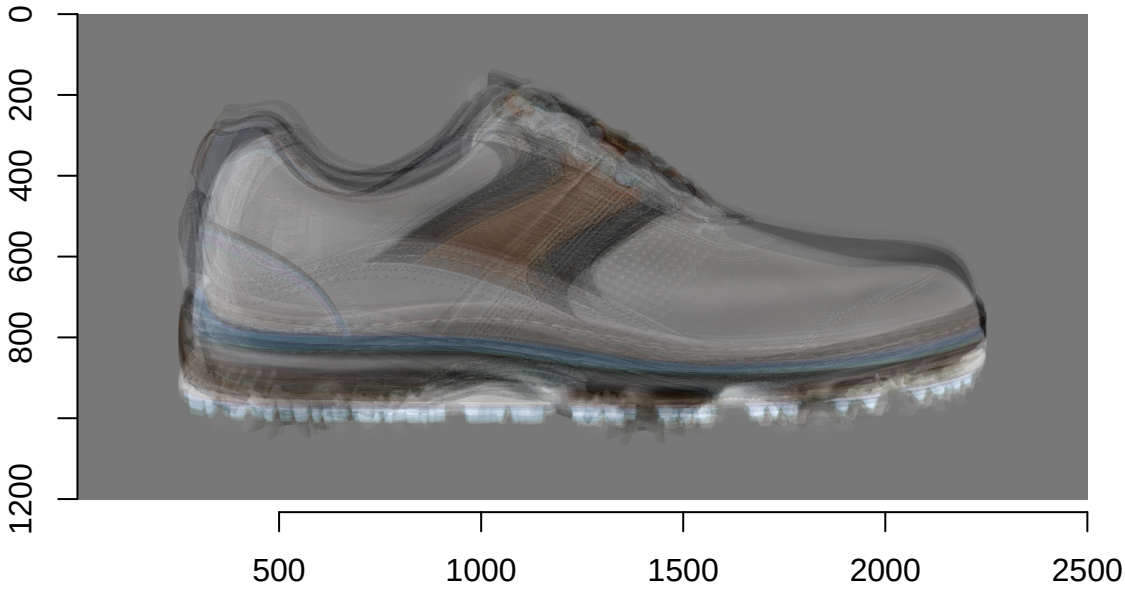
Finally, I create a for loop to plot the first 7 'eigenimages' or eigenshoes (or fewer since at 7 components we have 80% variance. I took each of the eigenimage as a flat vector and reshaped it to the original dimensions. Then, I turn the flat vector back into an image by changing the data type to cimg (C++ img). Afterwards, I plotted all of the eigenshoes.

```
# reshape and visualize first 7 eigenshoes

for (i in 1:min(7, num_components)) {
  eigenimage_vector = eigenimages[, i]

  #reshape back to the original dims

  eigenimage_array = array(eigenimage_vector, dim = dims)

  # make an imager object from the array

  eigenimage_im = as.cimg(eigenimage_array)

  # Visualize the eigenimage

  plot(eigenimage_im)
}
```

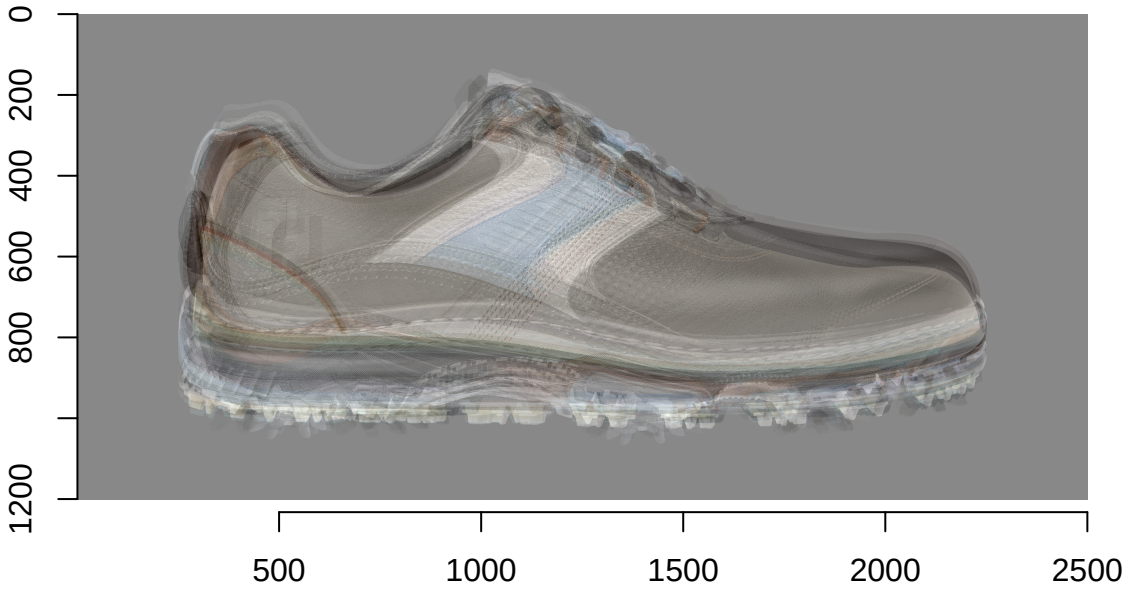
```
## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour

## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour
```



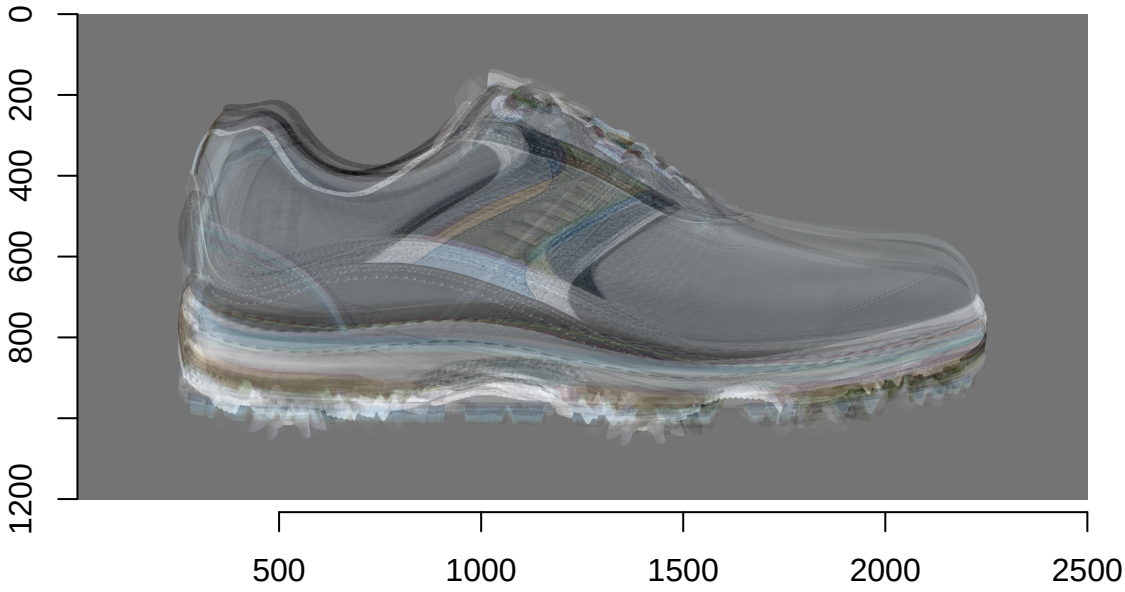

```
## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour
```



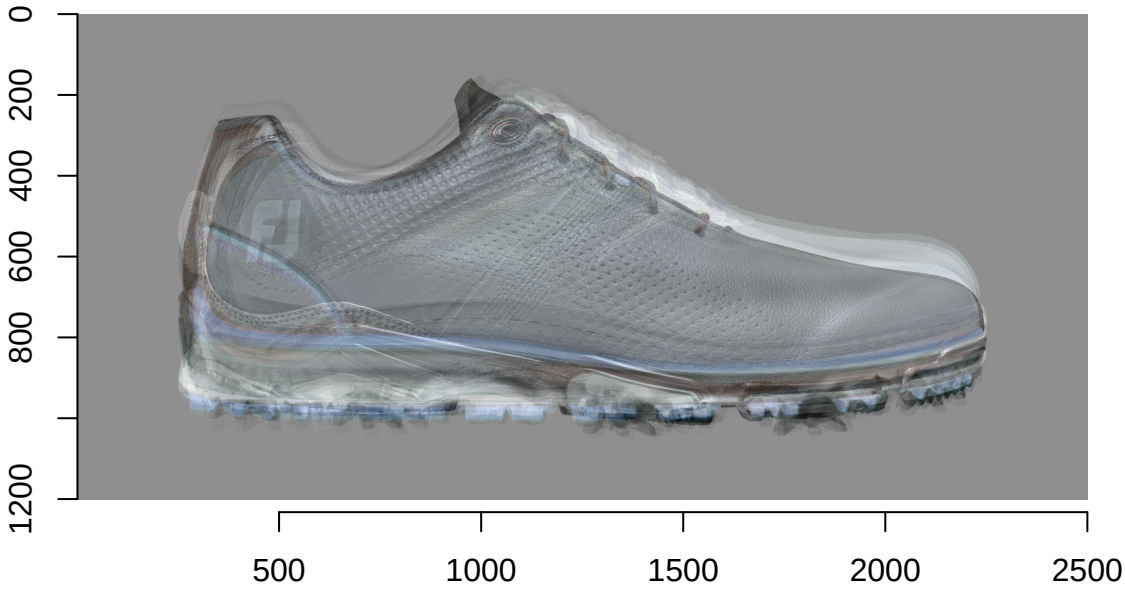

```
## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour
```



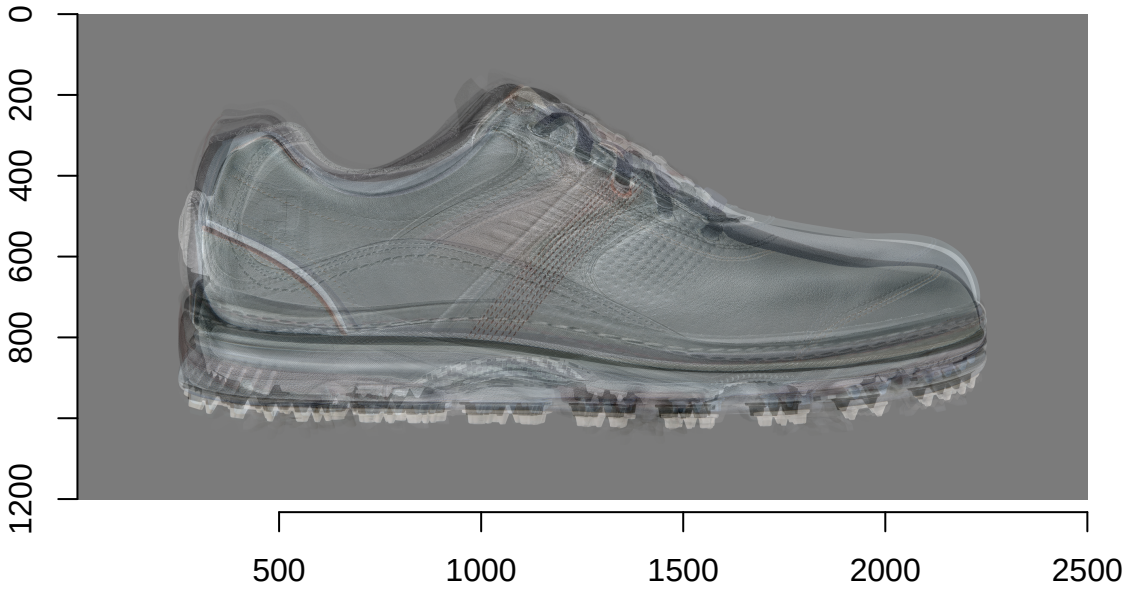

```
## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour
```

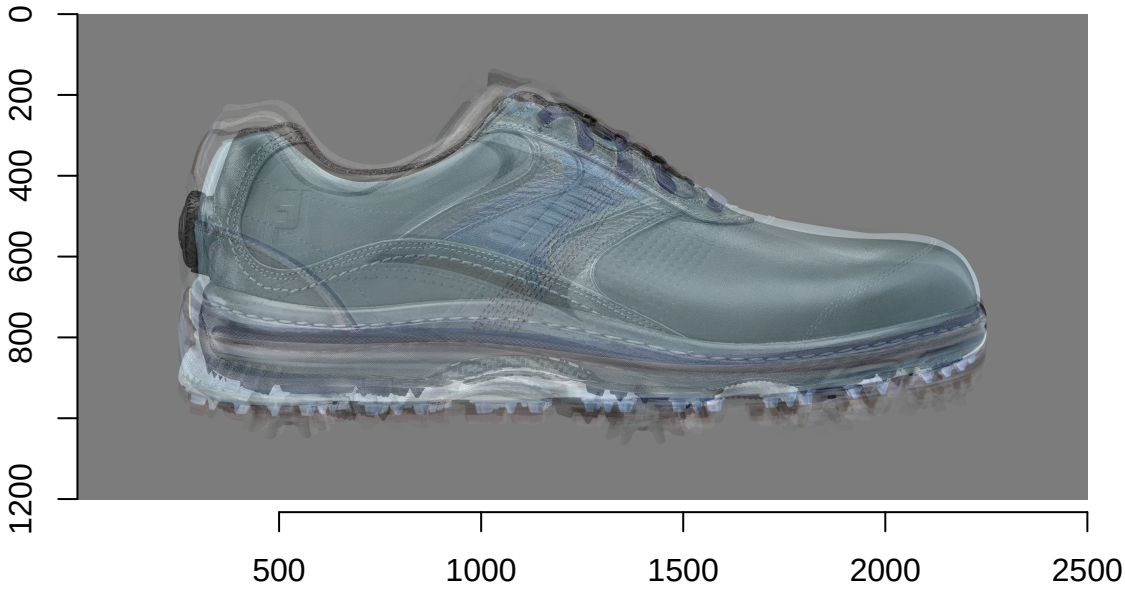



```
## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour
```




```
## Warning in as.cimg.array(eigenimage_array): Assuming third dimension
## corresponds to colour
```


0
200
400
600
800
1200
500
1000
1500
2000
2500



Reflection

In this homework assignment, I loaded a bunch of JPG images of shoes into R, converted each image into a matrix, centered the data, then performed a dimension reduction using PCA. After flattening each image matrix, I calculated the cumulative variance and the number of critical components needed to reach 80% in variance (7 components). Finally, I reshaped each eigenimage of the shoe and plotted it to visualize the 80% in variance of features.

I feel like this excessive was a good practice. As stated in class, understanding linear algebra is really important. As explored in this assignment, one application of linear algebra is image processing. I work processing medical images and have had a lot of parallels at work similar to this assignment. Particularly when it comes to mapping functional data onto anatomical data (or the other way around). All these calculations are engrained into the clinical software used to analyze these images. This unit in linear algebra has helped me both appreciate my work more and enhance my work quality. I am excited to keep learning more computational math!