DATA 605 Assignment 1: Part 1

Questions 1 & 2

Amanda Fox

Load libraries:

```
# load libraries
library(tidyverse)
library(ggplot2)
library(patchwork)
library(jpeg)
library(RCurl)
```

1. Geometric Transformation of Shapes Using Matrix Multiplication

Create a simple shape (like a square or triangle) using point plots in R. Implement R code to apply different transformations (scaling, rotation, reflection) to the shape by left multiplying a transformation matrix by each of the point vectors. Demonstrate these transformations through animated plots.

To create a triangle, we use a 2x4 matrix: the first three columns represent three point vectors (x,y) and the fourth column repeats the first point vector to close the shape.

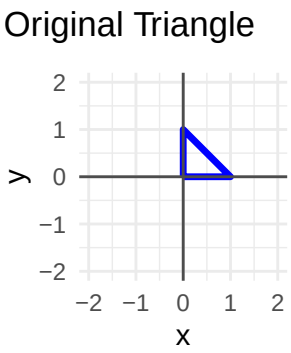
```
# create matrix defining triangle:
my_shape <- matrix(c(0,0,
                     1,0,
                     0,1,
                     0,0),
                   nrow = 2, byrow = TRUE)
my_shape
```

```
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    0
[2,]    0    1    0    0
```

```r
# prepare to plot: convert to df and label columns x & y
df_my_shape <- as.data.frame(t(my_shape))
colnames(df_my_shape) <- c("x","y")

df_my_shape
```

```
  x y
1 0 0
2 0 1
3 1 0
4 0 0
```

```r
# plot
plot_my_shape <- df_my_shape %>%
  ggplot(aes(x = x, y = y)) +
         geom_path(color = 'blue', linewidth = 1.2) +
         geom_hline(yintercept = 0, color = "gray30") +
         geom_vline(xintercept = 0, color = "gray30") +
         coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
         theme_minimal()

plot_my_shape + plot_annotation("Original Triangle")
```



Next, we can scale a shape by multiplying it by a transformation matrix. In this case, I set up a transformation matrix to multiply both x and y by 2:

```r
# create scaling transformation matrix multiplying x and y by 2
matrix_scale <- matrix(c(2,0,
                         0,2),
                       nrow = 2, byrow = TRUE)
matrix_scale
```

```
     [,1] [,2]
[1,]    2    0
[2,]    0    2
```

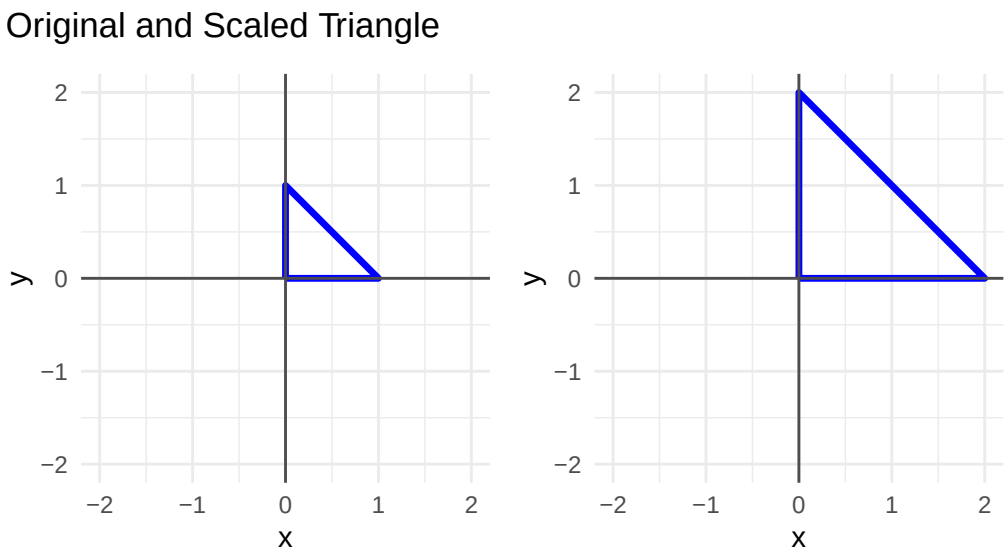
```
# multiply: transpose my_shape matrix so # rows match # columns
my_shape_scaled <- t(my_shape) %*% matrix_scale
my_shape_scaled
```

```
     [,1] [,2]
[1,]    0    0
[2,]    0    2
[3,]    2    0
[4,]    0    0
```

```
# convert to df with column names x,y
df_my_shape_scaled <- as.data.frame(my_shape_scaled)
colnames(df_my_shape_scaled) <- c("x","y")

df_my_shape_scaled
```

```
  x y
1 0 0
2 0 2
3 2 0
4 0 0
```

```
# plot
plot_my_shape_scaled <- df_my_shape_scaled %>%
  ggplot(aes(x = x, y = y)) +
  geom_path(color = 'blue', linewidth = 1.2) +
  geom_hline(yintercept = 0, color = "gray30") +
  geom_vline(xintercept = 0, color = "gray30") +
  coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
  theme_minimal()

plot_my_shape + plot_my_shape_scaled  + plot_annotation("Original and Scaled Triangle")
```



A transformation matrix can also be used to flip our shape over.

In the example below, I created a transformation matrix to flip the shape over the x axis by keeping x values the same while multiplying y values by -1:

```
# create transformation matrix (x, y) → (x, -y)
matrix_flip <- matrix(c(1, 0,
                        0,-1),
                      nrow = 2, byrow = TRUE)
matrix_flip
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0   -1
```

```
# multiply: transpose my_shape matrix so # rows match # columns
my_shape_flip <- t(my_shape) %*% matrix_flip
my_shape_flip
```

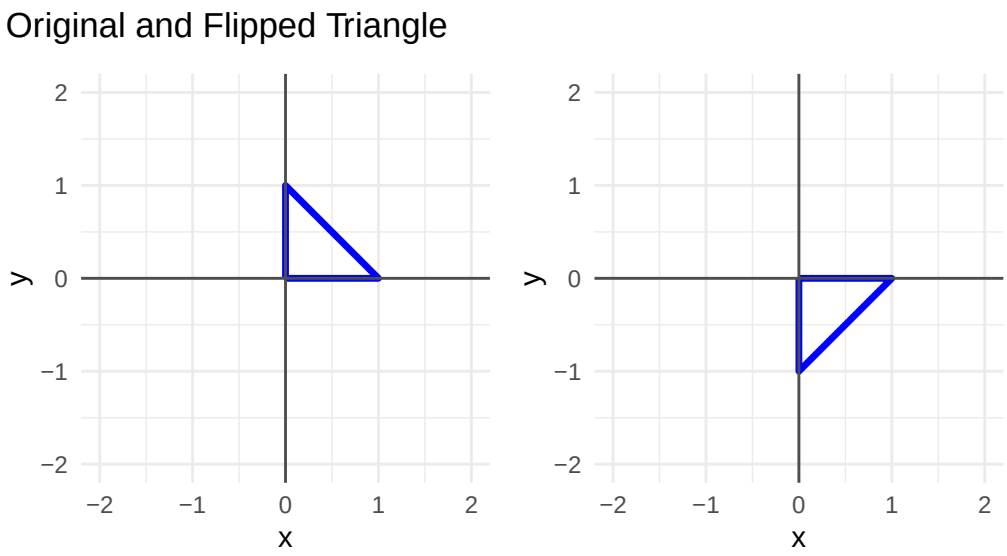
```
     [,1] [,2]
[1,]    0    0
[2,]    0   -1
```

```
[3,]    1    0
[4,]    0    0
```

```
# convert to df with column names x,y
df_my_shape_flip <- as.data.frame(my_shape_flip)
colnames(df_my_shape_flip) <- c("x","y")

df_my_shape_flip
```

```
  x  y
1 0  0
2 0 -1
3 1  0
4 0  0
```

```
# plot
plot_my_shape_flip <- df_my_shape_flip %>%
  ggplot(aes(x = x, y = y)) +
  geom_path(color = 'blue', linewidth = 1.2) +
  geom_hline(yintercept = 0, color = "gray30") +
  geom_vline(xintercept = 0, color = "gray30") +
  coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
  theme_minimal()

plot_my_shape + plot_my_shape_flip + plot_annotation("Original and Flipped Triangle")
```



Finally, transformation matrices can rotate a shape. Below, I created a transformation matrix that rotates a shape 90 degrees clockwise. I repeated it four times and captured images reflecting a full 360 degree rotation:

```
# create transformation matrix (x, y) → (-y, x)
matrix_rotate_90 <- matrix(c(0,-1,
                             1, 0),
                           nrow = 2, byrow = TRUE)
matrix_rotate_90
```

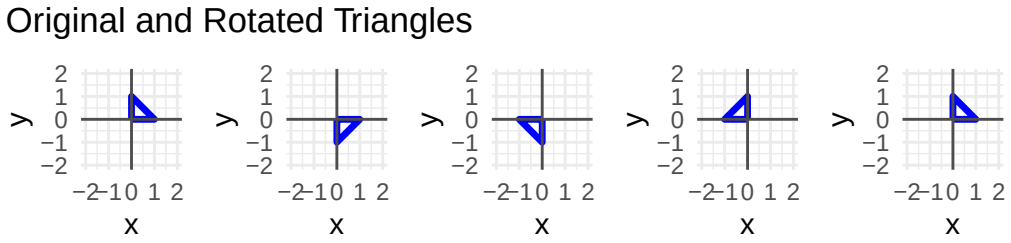
```
     [,1] [,2]
[1,]    0   -1
[2,]    1    0
```

```
# create list to hold plots for display
rotated_shapes <- list()
rotated_shapes[[1]] <- plot_my_shape

current_shape <- t(my_shape)
current_shape
```

```
     [,1] [,2]
```

```
[1,]    0    0
[2,]    0    1
[3,]    1    0
[4,]    0    0
```

```r
# loop: multiply, convert to df, create and store plot

for (i in 2:5) {

  # multiply matrix
  current_shape <- current_shape %*% matrix_rotate_90
  current_shape

  # convert transformed matrix to df with column names x,y
  df_rotate_90 <- as.data.frame(current_shape)
  colnames(df_rotate_90) <- c("x","y")

  df_rotate_90

  # create and store plot
  rotated_shapes[[i]] <- df_rotate_90 %>%
    ggplot(aes(x = x, y = y)) +
    geom_path(color = 'blue', linewidth = 1.2) +
    geom_hline(yintercept = 0, color = "gray30") +
    geom_vline(xintercept = 0, color = "gray30") +
    coord_fixed(xlim = c(-2,2), ylim = c(-2,2)) +
    theme_minimal()
  }

plot_display <- rotated_shapes[[1]] +
                rotated_shapes[[2]] +
                rotated_shapes[[3]] +
                rotated_shapes[[4]] +
                rotated_shapes[[5]]
plot_display +
  plot_layout(ncol = 5) +
  plot_annotation("Original and Rotated Triangles")
```



2. Matrix Properties and Decomposition

Proofs:

Prove that $A \times B \neq B \times A$

```
# --------------
# AB <> BA
# --------------

# define two 3x3 matrices A and B

A <- matrix(c(1,2,3,
              4,5,6,
              7,8,9),
            nrow = 3, byrow = TRUE)

B <- matrix(c(1, 0,-1,
              1, 2, 0,
              0,-1, 1),
            nrow = 3, byrow = TRUE)
```

```
A%*%B
```

```
     [,1] [,2] [,3]
[1,]    3    1    2
[2,]    9    4    2
[3,]   15    7    2
```

```
B%*%A
```

```
     [,1] [,2] [,3]
[1,]   -6   -6   -6
[2,]    9   12   15
[3,]    3    3    3
```

Prove that $A^T * A$ is always symmetric.

The matrix $A^T \times A$ is always symmetric because by transposing any matrix A with dimensions $m \times n$, we get a matrix A^T with dimensions $n \times m$. This has two implications for multiplying $A^T \times A$:

1. The multiplication $(n \times m) \times (m \times n)$ always works because the number of columns in the first matrix (m) match the number of rows(m) in the second, which makes matrix multiplication possible.
2. The resulting matrix has dimensions $n \times n$, which is square and symmetric, because the dimensions are defined by the number of rows in the first matrix (n) and the number of columns in the second matrix (also n).

```
# Matrix A from above:
A
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
# Transpose of A:
t(A)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
# Transpose of A times A:
t(A) %*% A
```

```
     [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

Prove that $det(A^T \times A)$ *is non-negative.*

$A^T \times A$ results in a square matrix as shown above, so we can calculate a determinant.

We also know that $det(A^T \times A) = det(A)^2$, and also that, like any square, $det(A)^2$ must be ≥ 0.

Therefore, if $det(A^T \times A) = det(A)^2$ and $det(A)^2 \geq 0$ then $det(A^T \times A) \geq 0$.

```
# determinant of transpose of A times A:
det(t(A) %*% A)
```

```
[1] -1.534772e-12
```

Singular Value Decomposition (SVD):

Write an R function that performs Singular Value Decomposition (SVD) on a grayscale image (which can be represented as a matrix). Use this decomposition to compress the image by keeping only the top k singular values and their corresponding vectors. Demonstrate the effect of different values of k on the compressed image's quality.

This was a challenging but beneficial exercise demonstrating a practical application of matrix decomposition. I had never manipulated images before so R help and online resources (ChatGPT and Gemini for coding assistance; see Works Cited) were particularly helpful in problem-solving this exercise as I worked my way through understanding the data behind images and manipulating that data in R as matrices.

First, I used a black and white image that turned out to be encoded as RGB. I learned this is common for compatibility reasons, but all channels will have the same values for a

grayscale image: this meant I could extract any channel in order to apply SVD with a simple R function.

Then I extracted the components U, S, and V and created a function to reconstruct the image matrices for various values of k (ChatGPT). The first iteration resulted in appropriately compressed images but in shades of red and rotated -90 degrees. Changing to gray was easy but I found multiple explanations for rotating the images; a matrix approach was most helpful to my understanding in the context of this assignment. While I could not multiply by a transformation matrix directly, the earlier rotation exercise was helpful in understanding how to manipulate the x,y points as rows/columns in an image, while the image matrix expressed the intensity.

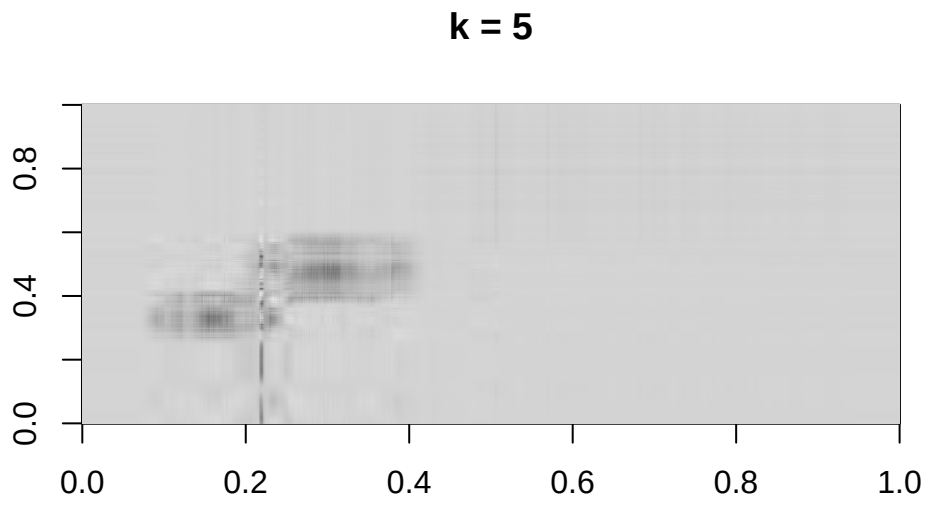
The resulting transformed images were compressed relative to the k value: low values for k (5, 40) showed significant loss of detail, while a higher value (80) was much closer to the original image, but still not as sharp.

```
# import image and convert to matrix

my_url <- "https://raw.githubusercontent.com/AmandaSFox/DATA605_Math/main/OC1Y7T0.jpg"
my_image <- readJPEG(getURLContent(my_url, binary = TRUE))

# check if three channels (RGB) and if so, extract one channel to make it grayscale
if(length(dim(my_image)) == 3) {
  my_image <- my_image[, , 1]
  }

# validate image size and not three dimensions
dim(my_image)
```

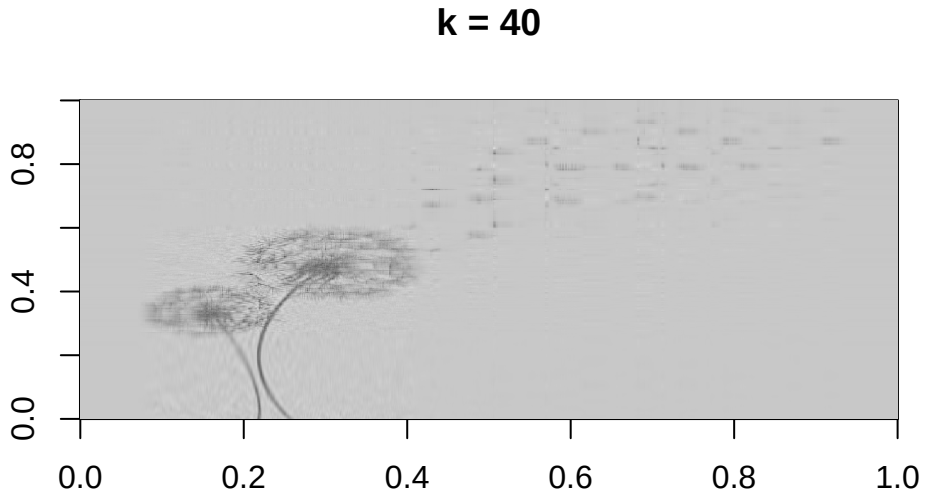
```
[1] 1300 1300
```

```
# apply SVD and display components
my_svd <- svd(my_image)

# Extract components
U <- my_svd$u
S <- diag(my_svd$d)
V <- my_svd$v

# Reconstruct image FUNCTION
reconstruct_image <- function(k) {
  U_k <- U[, 1:k]  # First k columns of U
  S_k <- S[1:k, 1:k]  # Top k singular values
```

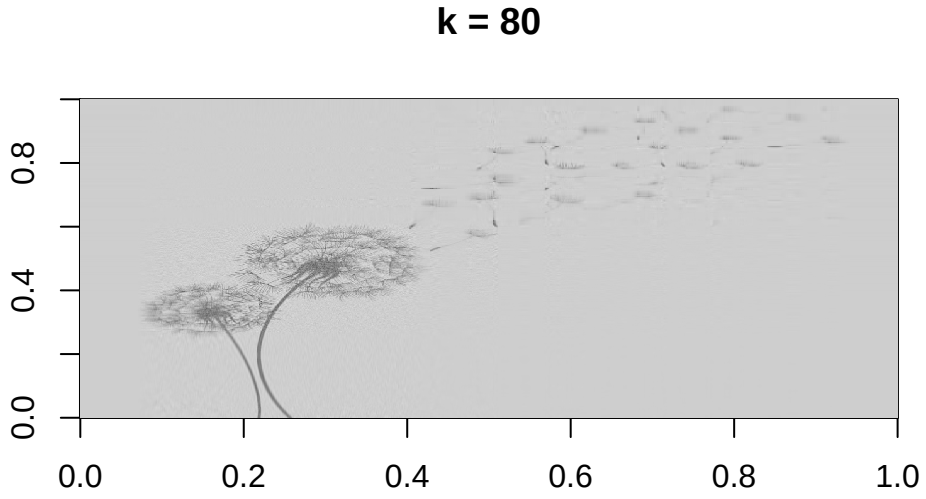
```r
  V_k <- V[, 1:k]  # First k columns of V
  img_k <- U_k %*% S_k %*% t(V_k)
  return(img_k)
}

# Substitute different k values and create three new matrices:
img_k5 <- reconstruct_image(5)
img_k40 <- reconstruct_image(30)
img_k80 <- reconstruct_image(80)

# create function to correct rotation:
rotate_90 <- function(image_matrix) {
  rotated_matrix <- t(image_matrix)[, ncol(image_matrix):1]
  return(rotated_matrix)
}

# setting to display all images in one row
#par(mfrow = c(1,4))

# create four images, applying above rotation function, gray instead of red
my_image_k5 <- image(rotate_90(img_k5), col = gray.colors(256), main="k = 5")
```



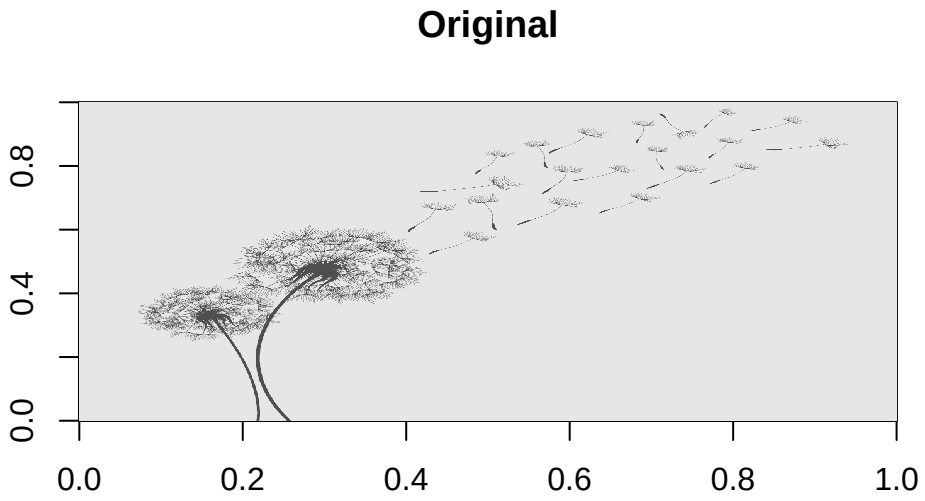
```
my_image_k40 <- image(rotate_90(img_k40), col = gray.colors(256), main="k = 40")
```




```
my_image_k80 <- image(rotate_90(img_k80), col = gray.colors(256), main="k = 80")
```



```
my_image_orig <- image(rotate_90(my_image), col = gray.colors(256), main="Original")
```



Works Cited

Beezer, Robert. "A First Course in Linear Algebra." Open Textbook Library, 2015, open.umn.edu/opentextbooks/textbooks/5. Accessed Feb. 2025.

ChatGPT. "ChatGPT." Chatgpt.com, 2024, chatgpt.com. Accessed Feb. 2025. R coding and LaTeX assistance.

Google. "Gemini." Gemini.google.com, Google, 2024, gemini.google.com/app. Accessed Feb. 2025. R coding assistance.

Layerace. Image by Layerace on Freepik. www.freepik.com/free-vector/dandelion-background-design_921206.htm. Accessed Feb. 2025.

DATA 605 Assignment 1: Part 2

Questions 3 & 4

Amanda Fox

```
library(tidyverse)
library(imager)
library(FactoMineR)
library(reticulate)
```

3. Matrix Rank, Properties, and Eigenspace

Determine the Rank of the Given Matrix.

Find the rank of the matrix A (below). Explain what the rank tells us about the linear independence of the rows and columns of matrix A. Identify if there are any linear dependencies among the rows or columns.

$$A = \begin{bmatrix} 2 & 4 & 1 & 3 \\ -2 & -3 & 4 & 1 \\ 5 & 6 & 2 & 8 \\ -1 & -2 & 3 & 7 \end{bmatrix}$$

In row echelon form, we get:

$$A = \begin{bmatrix} 1 & 2 & 0.5 & 1.5 \\ 0 & 1 & 5 & 4 \\ 0 & 0 & 1 & 16.5/19.5 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

The rank is the number of nonzero rows in row echelon form. This 4x4 matrix has rank = 4 or full rank, which means no free variables or linear dependencies. All rows and columns are linearly independent.

In R:

```r
library(Matrix)
```

```
Attaching package: 'Matrix'

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

    expand, pack, unpack
```

```r
A <- matrix(c(2,4,1,3,
              -2,-3,4,1,
              5,6,2,8,
              -1,-2,3,7), byrow = TRUE, nrow = 4)
rank_A <- as.numeric(rankMatrix(A))
rank_A
```

```
[1] 4
```

Matrix Rank Boundaries

Given an m x n matrix where $m > n$, determine the maximum and minimum possible rank, assuming that the matrix is non-zero. Prove that the rank of a matrix equals the dimension of its row space (or column space). Provide an example to illustrate the concept.

1. Maximum rank = n because the number of linearly independent columns can't exceed n
2. Minimum rank = 1 because there must be at least one independent row or column since the matrix is non-zero
3. The rank of a matrix is the same as the dimension of its row space, which is the number of linearly independent rows. Performing row operations on a matrix to get to row echelon form does not change the row space but it allows us to see it clearly by reducing dependent rows to zeroes:

Example matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

Row echelon form with one linearly independent variable (non-zero row) showing the row space and rank of this 3x3 matrix is actually 1:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In R:

```
A <- matrix(c(1,2,3,
              2,4,6,
              3,6,9), byrow = TRUE, nrow = 3)
rank_A <- as.numeric(rankMatrix(A))
rank_A
```

```
[1] 1
```

Rank and Row Reduction:

Determine the rank of matrix B. Perform a row reduction on matrix B and describe how it helps in finding the rank. Discuss any special properties of matrix B (e.g., is it a rank-deficient matrix?)

$$B = \begin{bmatrix} 2 & 5 & 7 \\ 4 & 10 & 14 \\ 1 & 2.5 & 3.5 \end{bmatrix}$$

Row-reduced to row echelon form:

Swap R3 and R1, subtract 2xR1 from R3 and subtract 2xR1 from R2:

$$B = \begin{bmatrix} 1 & 2.5 & 3.5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The rank is the number of linearly independent rows, which are the non-zero rows in row echelon format, or rank = 1.

This is *rank deficient* because the maximum rank of a 3x3 matrix would be 3, which means this matrix had linearly dependent rows and columns (has free variables). This matrix collapses a 3D space to a 1D vector and is singular or not invertible (can't be reversed).

Check in R:

```r
B <- matrix(c(2,5,7,
              4,10,14,
              1,2.5,3.5), byrow = TRUE, nrow = 3)
rank_B <- as.numeric(rankMatrix(A))
rank_B
```

```
[1] 1
```

Compute the eigenvalues and eigenvectors

Find the eigenvalues and eigenvectors of the matrix A. Write out the characteristic polynomial and show your solution step by step. After finding the eigenvalues and eigenvectors, verify that the eigenvectors are linearly independent. If they are not, explain why.

$$A = \begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix}$$

First we find eigenvalues by solving the characteristic equation $det(A - \lambda I)$:

$$det\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)$$

Multiply by λ:

$$det\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \right)$$

Subtract:

$$det\left(\begin{bmatrix} 3-\lambda & 1 & 2 \\ 0 & 5-\lambda & 4 \\ 0 & 0 & 2-\lambda \end{bmatrix} \right)$$

Find determinant using $det = a(ei - fh) - b(di - fg) + c(dh - eg)$:

From above:
a $= 3 - \lambda$
b $= 1$
c $= 2$
d $= 0$
e $= 5 - \lambda$
f $= 4$
g $= 0$

h = 0
i = $2 - \lambda$

ei = $(5 - \lambda) * (2 - \lambda) = \lambda^2 - 7\lambda + 10$
fh = 0
di = 0
fg = 0
dh = 0
eg = 0

$det = a(ei - fh) - b(di - fg) + c(dh - eg)$
$det = (3 - \lambda) * (\lambda^2 - 7\lambda + 10)$
$det = -\lambda^3 + 10\lambda^2 - 31\lambda + 30$

Set equal to zero to find null space:
$-\lambda^3 + 10\lambda^2 - 31\lambda + 30 = 0$

Try $\lambda = 2$ (Rational Root Theorem: roots must be factor of 30)
$-8 + 40 - 62 + 30 = 0$ is True

So $\lambda = 2$, therefore $(\lambda - 2)$ is a factor and can be factored out:
$-\lambda^2 - 8\lambda - 15 = 0$

Flip signs to make it easier:
$\lambda^2 + 8\lambda + 15 = 0$
$(\lambda - 3)(\lambda - 5) = 0$

Eigenvalues are {2,3,5}

Eigenvectors: Now we find eigenvectors by substituting each eigenvalue into the formula $(A - \lambda I)v = 0$:

1. $\lambda = 2$:
   $(A - 2I)v = 0$

$$\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - 2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \right) \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 3 & 4 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$v_1 + v_2 + 2v_3 = 0$$
$$0 + 3v_2 + 4v_3 = 0$$
$$0 + 0 + 0 = 0$$

So $v_2 = -4/3v_3$ (from second equation) and substituting that into the first equation:

$$v_1 - 4/3v_3 + 2v_3 = 0$$
$$v_1 + 2/3v_3 = 0$$
$$v_1 = -2/3v_3$$

v_3 is free variable; set to t:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} -2/3t \\ -4/3t \\ t \end{bmatrix} = t \begin{bmatrix} -2/3 \\ -4/3 \\ 1 \end{bmatrix} = t \begin{bmatrix} -2 \\ -4 \\ 3 \end{bmatrix}$$

For $\lambda = 2$, the eigenvector is $\begin{bmatrix} -2 \\ -4 \\ 3 \end{bmatrix}$

2. $\lambda = 3$:

$$(A - 3I)v = 0$$

$$\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - 3 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \right) \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$v_2 + 2v_3 = 0$$
$$2v_2 + 4v_3 = 0$$
$$-v_3 = 0$$

v_1 does not appear in the equations and is free variable; set to t:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} t \\ 0t \\ 0t \end{bmatrix} = t \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

For $\lambda = 3$, the eigenvector is

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

3. $\lambda = 5$:

$$(A - 5I)v = 0$$

$$\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - 5 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left(\begin{bmatrix} 3 & 1 & 2 \\ 0 & 5 & 4 \\ 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix} \right) \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} -2 & 1 & 2 \\ 0 & 0 & 4 \\ 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$-2v_1 + v_2 + 2v_3 = 0$$
$$4v_3 = 0$$
$$-3v_3 = 0$$

So $v_3 = 0$ and we can substitute:
$-2v_1 + v_2 = 0$
$v_2 = 2v_1$

Because there are more variables than equations, one must be free. We can assume v_1 is free and set it to t:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} t \\ 2t \\ 0 \end{bmatrix} = t \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

For $\lambda = 5$, the eigenvector is

$$\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

Validation in R:

```
A <- matrix(c(3,1,2,
              0,5,4,
              0,0,2),
            nrow=3,
            byrow=TRUE)

A
```

```
     [,1] [,2] [,3]
[1,]    3    1    2
[2,]    0    5    4
[3,]    0    0    2
```

```
my_eigen <- eigen(A)
my_eigen$values
```

```
[1] 5 3 2
```

```
my_eigen$vectors
```

```
          [,1] [,2]       [,3]
[1,] 0.4472136    1 -0.3713907
[2,] 0.8944272    0 -0.7427814
[3,] 0.0000000    0  0.5570860
```

Linear independence: We can determine if the eigenvectors are linearly independent by setting up an matrix and finding the determinant.

The below R code quickly calculates the determinant is 6. Because it is greater than zero, the vectors are **linearly independent**:

```
B <- matrix(c(-2,1,1,
              -4,0,2,
              3,0,0), byrow = TRUE, nrow = 3)
B
```

```
     [,1] [,2] [,3]
[1,]   -2    1    1
[2,]   -4    0    2
[3,]    3    0    0
```

```
det_B <- det(B)
det_B
```

```
[1] 6
```

Diagonalization of Matrix:

Determine if matrix A can be diagonalized. If it can, find the diagonal matrix and the matrix of eigenvectors that diagonalizes. Discuss the geometric interpretation of the eigenvectors and eigenvalues in the context of transformations. For instance, how does matrix A stretch, shrink, or rotate vectors

We know the matrix A can be diagnolized because it has three linearly independent eigenvectors.

The matrix of eigenvectors from above:

$$\begin{bmatrix} -2 & 1 & 1 \\ -4 & 0 & 2 \\ 3 & 0 & 0 \end{bmatrix}$$

The diagonal matrix D contains the eigenvalues for each vector, on the diagonal:

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Matrix A scales its eigenvectors by the corresponding eigenvalues shown in D.

Conclusion: The matrix A is an upper triangular matrix (non-symmetric) with a **set of eigenvalues = {2,3,5}**.

It has a full set of eigenvectors, which are **linearly independent**:

For $\lambda = 2$, the eigenvector is

$$\begin{bmatrix} -2 \\ -4 \\ 3 \end{bmatrix}$$

For $\lambda = 3$, the eigenvector is

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

For $\lambda = 5$, the eigenvector is

$$\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

Matrix A can be diagonalized as shown above.

4. Eigenfaces from the LFW (Labeled Faces in the Wild) Dataset

Task: Using the LFW (Labeled Faces in the Wild) dataset, build and visualize eigenfaces that account for 80% of the variability in the dataset. The LFW dataset is a well-known dataset containing thousands of labeled facial images, available for academic research.

Download the data

I used the suggested approach of a Python script (lfw module in sklearn library) to download the dataset and extracted the images component to a numpy file. I used no resizing and color=True to set up for our preprocessing step below.

```
##NOTE: This is the code I ran to download the data, which I then stored in an Azure data st

from sklearn.datasets import fetch_lfw_people
import numpy as np

# download full dataset
lfw_people = fetch_lfw_people(min_faces_per_person=1, resize=None, color=True)

# save images file as numpy array
images = lfw_people.images
np.save("lfw_image_arrays.npy", images)
```

Preprocess the images

Convert the images to grayscale and resize them to a smaller size (e.g., 64x64) to reduce computational complexity. Flatten each image into a vector.

In this challenging step, I used R help and online resources (Gemini) to learn about the **imager** package, as well as how and why we need to move between images objects, arrays, and finally vectors to accomplish all of our required processing.

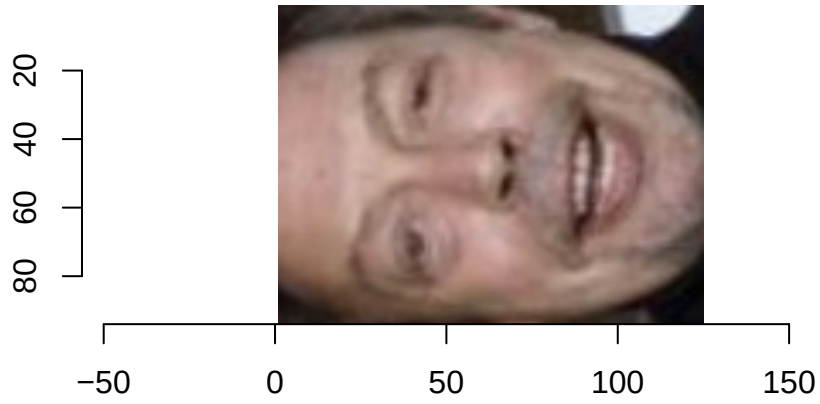
After successfully using an images package function to grayscale the whole array of images at once, I had some trouble in trying to replicate that logic to resize the whole array at once: that definitely did not work as it resized the **array** itself and I needed to resize each image via looping. Frequently checking the dimensions with the **dim** function was key to keeping everything straight.

After that, flattening the array into vectors and storing in a matrix was straightforward with more looping.

```
#Load data with numpy. Verify 13233 images

np <- import("numpy")

# Note - I stored a copy in Azure to make the code shareable but
# it got corrupted: the below references my local C:\

lfw_data <- np$load("lfw_image_arrays.npy")
dim(lfw_data)
```

```
[1] 13233   125    94     3
```

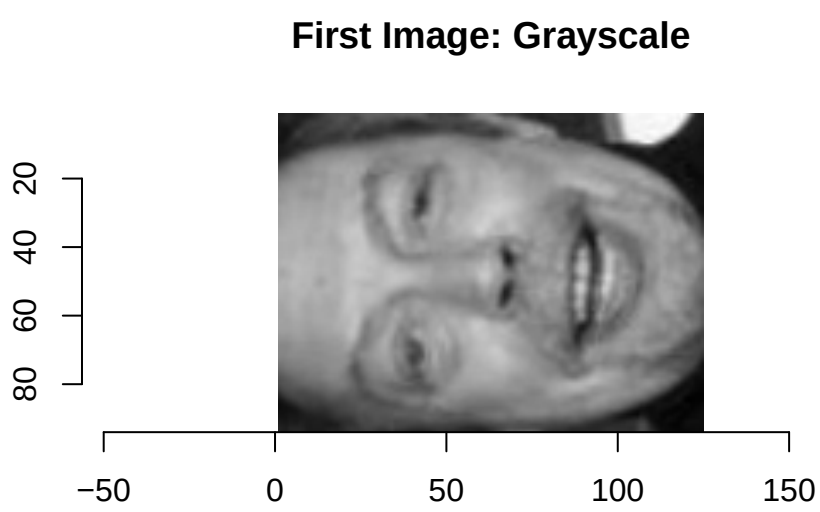
```
# Show the first image in original form
first_image <- lfw_data[1,,,]
first_image_cimg <- as.cimg(first_image)
plot(first_image_cimg)
```




```
# Convert to an imager object and grayscale
lfw_cimg <- as.cimg(lfw_data)
grayscale_lfw_cimg <- grayscale(lfw_cimg)

# Convert back to an array
grayscale_lfw <- as.array(grayscale_lfw_cimg[,,,1]) #select first channel
dim(grayscale_lfw)
```

```
[1] 13233   125    94
```

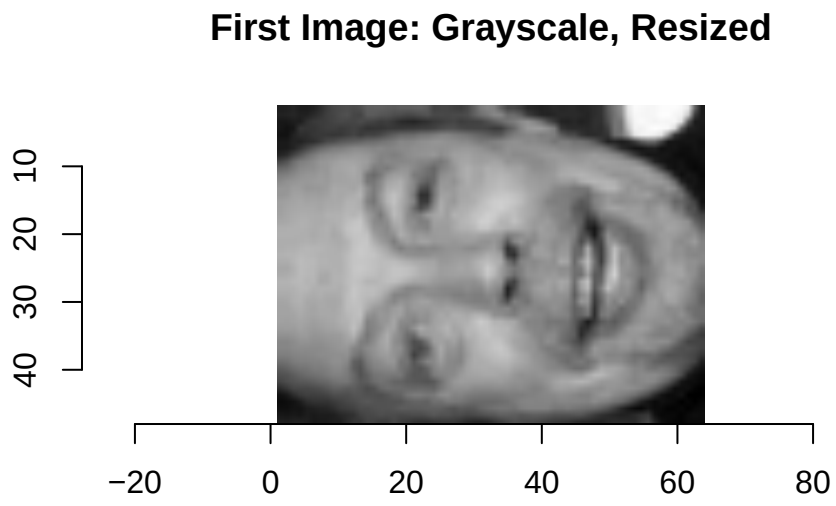
```
# View first grayscaled image
grayscale_image <- grayscale_lfw[1,,]
grayscale_image_cimg <- as.cimg(grayscale_image)
plot(grayscale_image_cimg, main = "First Image: Grayscale")
```




```
# Create array to store resized images
count_images <- dim(grayscale_lfw)[1]
resized_grayscale_lfw <- array(0, dim = c(count_images, 64, 48))

# Loop
for (i in 1:count_images) {
  # Convert to cimg
  image_cimg <- as.cimg(grayscale_lfw[i,,])
  # Resize
  resized_image_cimg <- resize(image_cimg, 64, 48)  # Resize to 64x48
  # Convert back to array
  resized_grayscale_lfw[i,,] <- as.array(resized_image_cimg[,,,1])
}

# Check the dimensions
dim(resized_grayscale_lfw)
```

```
[1] 13233    64    48
```

```
# View first resized image
resized_grayscale_image <- resized_grayscale_lfw[1,,]
resized_grayscale_image_cimg <- as.cimg(resized_grayscale_image)
plot(resized_grayscale_image_cimg, main = "First Image: Grayscale, Resized")
```




```
# Flatten each image into vector and store as matrix

matrix_lfw <- matrix(NA,
                         nrow = dim(resized_grayscale_lfw)[1],
                         ncol = 64*48)

for (i in 1:dim(resized_grayscale_lfw)[1]) {
  matrix_lfw[i, ] <- as.vector(resized_grayscale_lfw[i,,])
}

# Check the new dimensions
dim(matrix_lfw)
```

```
[1] 13233  3072
```

Apply PCA

Compute the PCA on the flattened images. Determine the number of principal components required to account for 80% of the variability.

In this step, I learned about the outputs of the simple base R **prcomp** function and used the sdev component to find the % variation explained by each principal component.

40 principal components account for 80% of the variability:

```
# Compute PCA using base R function
pca_result <- prcomp(matrix_lfw, center = TRUE, scale. = TRUE)

# Compute cumulative variance
explained_variance <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumulative_variance <- cumsum(explained_variance)
components_80 <- which(cumulative_variance >= 0.80)[1]
components_80
```

```
[1] 40
```

Visualize Eigenfaces

Visualize the first few eigenfaces (principal components) and discuss their significance. Reconstruct some images using the computed eigenfaces and compare them with the original images.

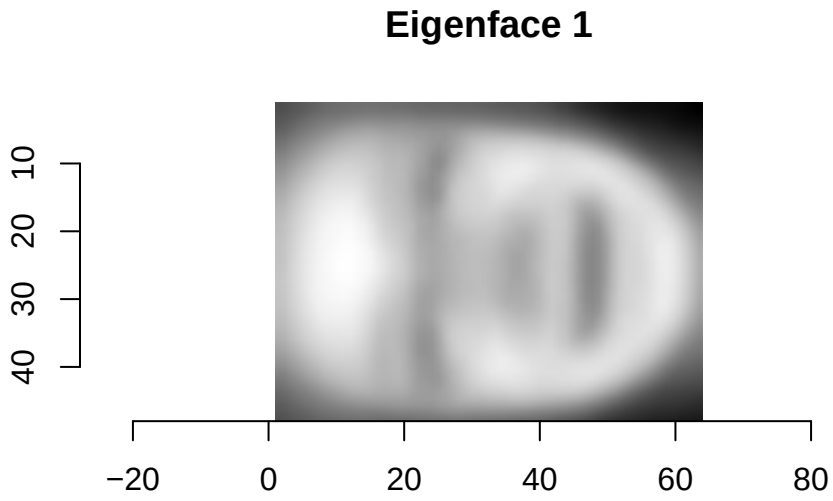
The **rotation** component of the PCA output is a matrix of eigenvectors. Each column is a PCA, arranged in order of highest to lowest variation, and each row is a pixel.

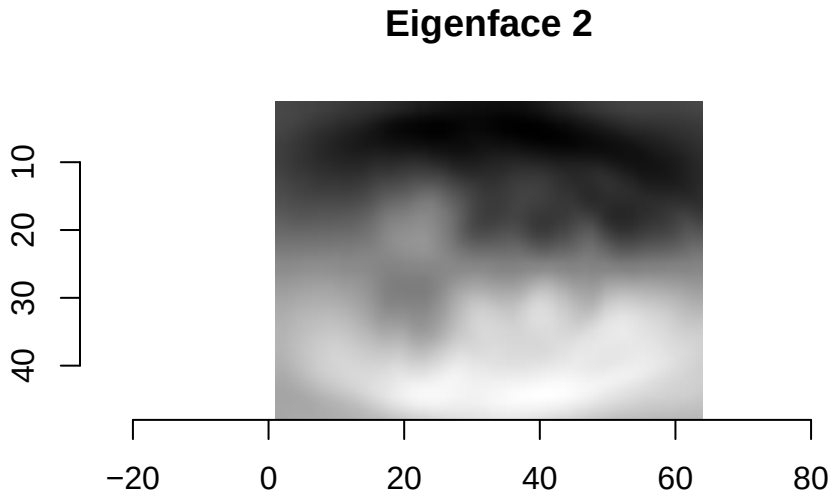
In this facial recognition exercise, a PCA is an eigenface, or a component that makes up face images.

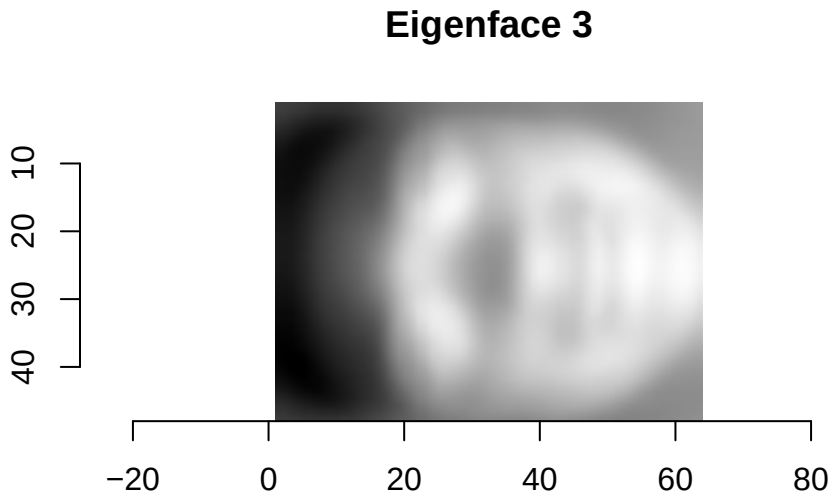
```
# top four pcas
num_eigenfaces <- 4
eigenfaces <- pca_result$rotation[, 1:num_eigenfaces]
dim(eigenfaces)
```

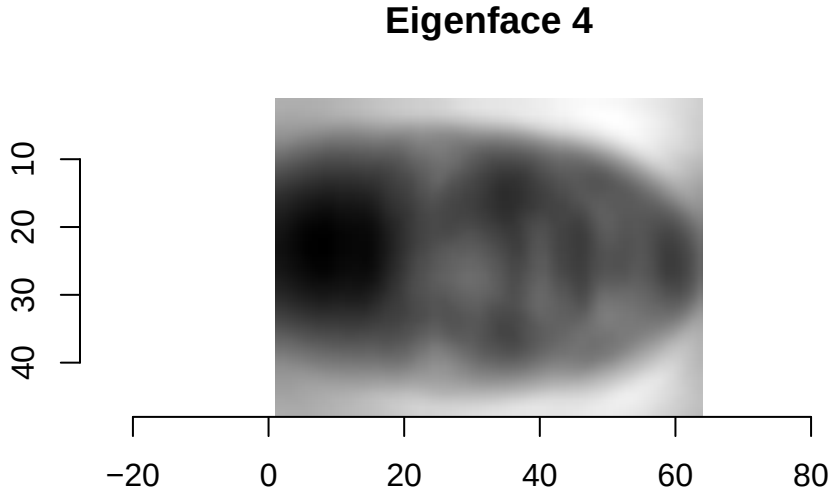
```
[1] 3072    4
```

```
# Loop through the first few principal components
for (i in 1:num_eigenfaces) {
  eigenface_matrix <- matrix(pca_result$rotation[, i],
                             nrow = 64,
                             ncol = 48)  # Reshape vector
  eigenface_cimg <- as.cimg(eigenface_matrix)  # Convert to cimg object
  plot(eigenface_cimg, main = paste("Eigenface", i))
}
```


Eigenface 1
10
20
30
40
−20
0
20
40
60
80

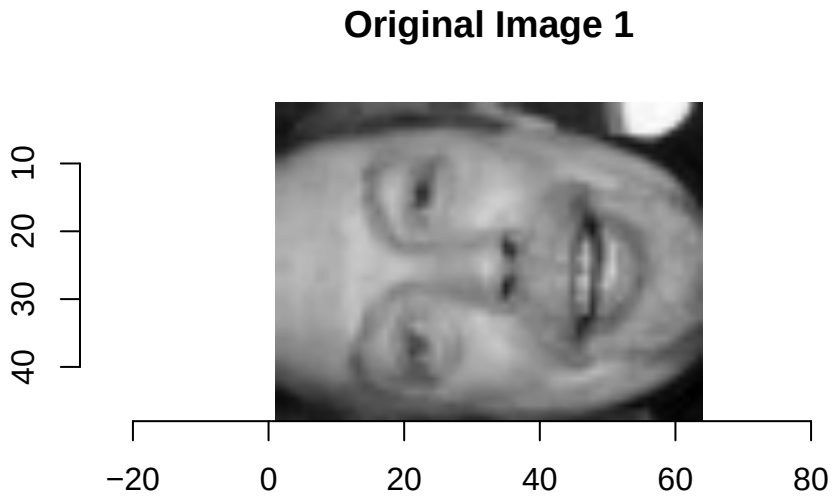

Eigenface 2
10
20
30
40
−20
0
20
40
60
80

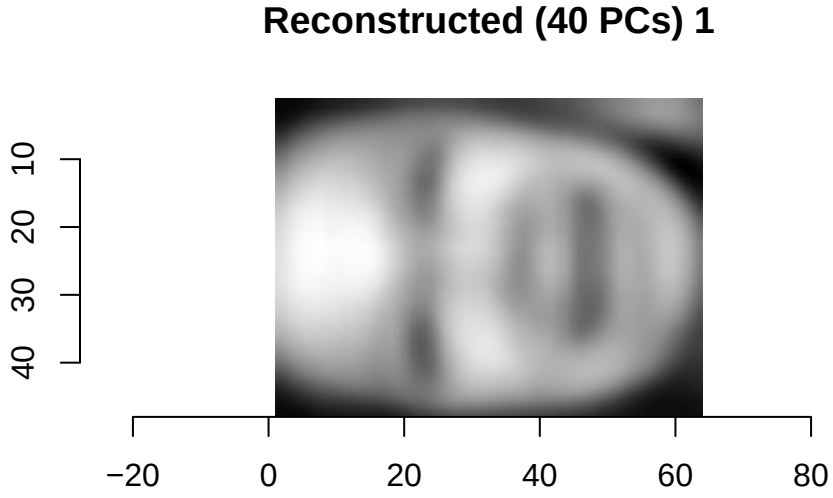


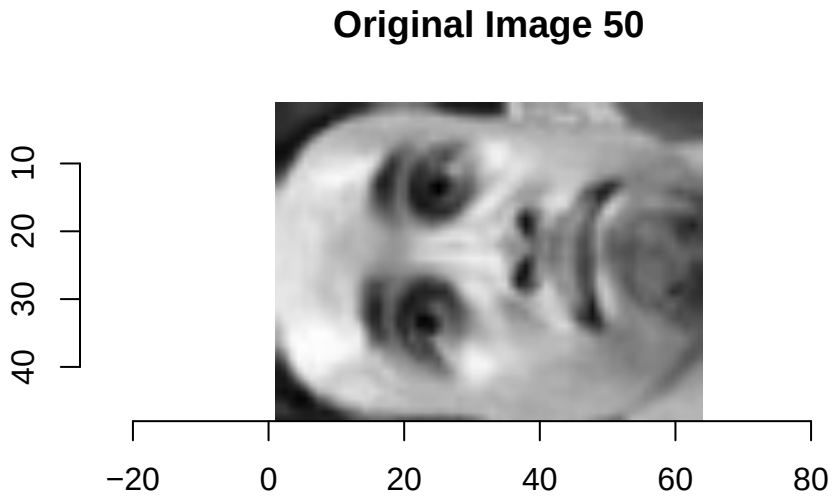


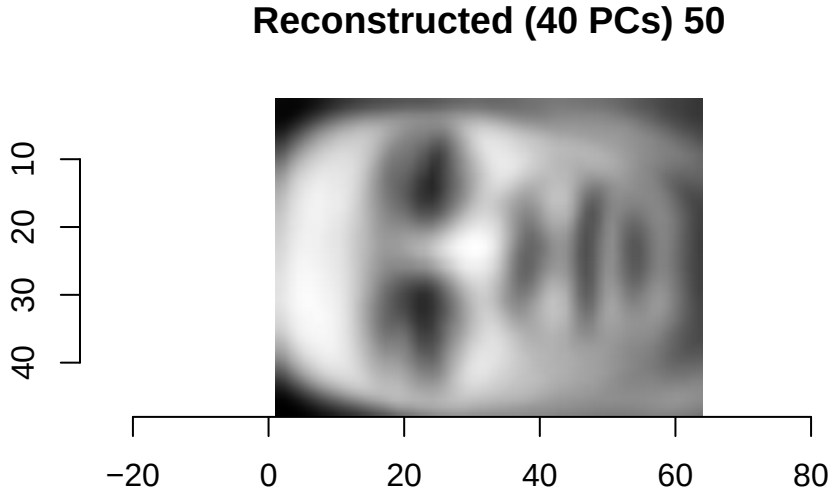
For the final reconstruction, I picked images # 1, 500, and 1000 from the grayscale/resized data and used the top 40 PCAs (which accounted for 80% of the variation in the dataset). Online resources were again very helpful in this step and I learned a lot about this process through this exercise.

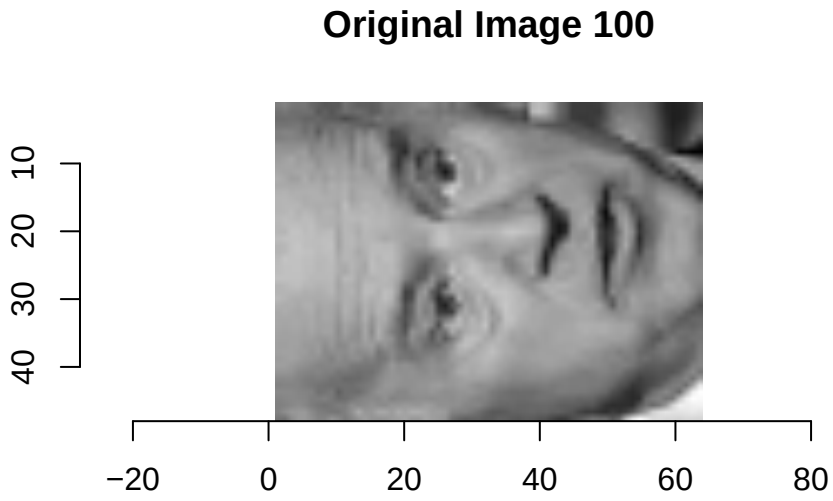
```r
#---------
# Reconstructing with Eigenfaces
#---------

# Images to reconstruct
image_indices <- c(1, 50, 100)

# Initialize a list to store reconstructed images
reconstructed_images <- list()

for (i in seq_along(image_indices)) {
  img_idx <- image_indices[i]
  test_image <- matrix_lfw[img_idx, ]
  # get principal components
  projected_coefficients <- test_image %*% pca_result$rotation[, 1:40]
  # Reconstruct the image
  reconstruction <- projected_coefficients %*% t(pca_result$rotation[, 1:40])

  # Add the mean image back
  reconstructed_image <- reconstruction + pca_result$center

  # Reshape into 64x48 image format and store it
  reconstructed_images[[i]] <- matrix(reconstructed_image, nrow = 64, ncol = 48)
}

#---------
# Plots
#---------

for (i in seq_along(image_indices)) {
  img_idx <- image_indices[i]
  # Plot Original Image
  original_matrix <- matrix(matrix_lfw[img_idx, ], nrow = 64, ncol = 48)
  original_cimg <- as.cimg(original_matrix)
  plot(original_cimg, main = paste("Original Image", img_idx))
  # Plot Reconstructed Image (40 PCs)
  reconstructed_cimg <- as.cimg(reconstructed_images[[i]])
  plot(reconstructed_cimg, main = paste("Reconstructed (40 PCs)", img_idx))
}
```

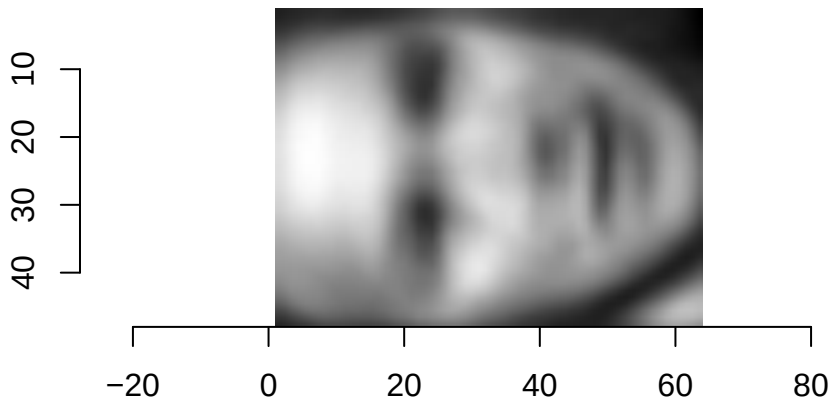

Original Image 1
10
20
30
40
−20
0
20
40
60
80


Reconstructed (40 PCs) 1
10
20
30
40
−20
0
20
40
60
80


Original Image 50
10
20
30
40
−20
0
20
40
60
80


Reconstructed (40 PCs) 50
10
20
30
40
−20
0
20
40
60
80


Original Image 100
10
20
30
40
−20
0
20
40
60
80


Reconstructed (40 PCs) 100
10
20
30
40
−20
0
20
40
60
80

Works Cited

Beezer, Robert. "A First Course in Linear Algebra." Open Textbook Library, 2015, open.umn.edu/opentextbooks/textbooks/5. Accessed Feb. 2025.

ChatGPT. "ChatGPT." Chatgpt.com, 2024, chatgpt.com. Accessed Feb. 2025. R coding and LaTeX assistance.

Google. "Gemini." Gemini.google.com, Google, 2024, gemini.google.com/app. Accessed Feb. 2025. R coding assistance.

Huang, G. B., et al. "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments." University of Massachusetts, Amherst, Technical Report 07-49, 2007.