Assignment 2: Feature Engineering, Statistical Analysis and Machine Learning

Matteo Melis (40324932)

Introduction

The objective of this assignment was to create a dataset of hand-drawn images depicting living and non-living objects, extract significant features for each image, and employ machine learning models to classify the dataset. By analyzing the data, the main aim was to determine which features will produce an effective classification model of images.

Section 1

To begin, I used GIMP to manually create a 45x45 pixel square and drew all 112 images using a mouse. These images were then placed in an "images" folder and imported into R. I looped through the folder and used the code below to convert each image into a 45x45 TSV matrix consisting of 0's and 1's (representing white and black pixels, respectively). These matrices were then written into a subfolder.

```r
require("rstudioapi")
require("stringr")
require("raster")
require("rmarkdown")
require("MASS")

# Set the path to the folder containing the PGM files
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
tsv_dir <- paste(getwd(), "/images/", sep = "")
setwd(file.path(getwd(), "/images"))

# Get a list of all the PGM files in the folder
all_files <- list.files(path = getwd())
pgm_files <- list()

# filter out so its just pgms
for (file in all_files) {
    if (str_detect(file, ".pgm")) {
        pgm_files <- append(pgm_files, file)
    }
}

# Initialize an empty list to store the matrices
pgm_matrices <- list()

# Loop through the PGM files
for (pgm in pgm_files) {
    # Read the PGM file into a data frame
    pgm_data <- read.table(pgm, skip = 4, sep = "", header = F)

    # Convert the data frame into a matrix
    tsv_matrix <- matrix(as.matrix(pgm_data), nrow = 45, byrow = TRUE)

    # Change all values from 255 to 0 and 0 to 1

    tsv_matrix[tsv_matrix < 128] <- 1  #black pixels
    tsv_matrix[tsv_matrix > 128] <- 0  #white pixels

    # Add the matrix to the list
    pgm_matrices[[pgm]] <- tsv_matrix

    filename <- str_replace(pgm, "pgm", "tsv")

    write.matrix(tsv_matrix, file = paste(tsv_dir, filename,
        sep = ""))

}
```

Section 2

In this section, I was tasked with developing code to calculate 16 image features and to then display this information in a .csv file. The first feature, nr_pix (number of black pixels), involved looping through each row and column to check for an entry of 1 (black pixel). If a black pixel was present, I added it to the total count.

The following six features required calculating the number of rows/columns with 1, 2, or 3+ black pixels. I used similar logic for all six features, looping through each row to tally up the black pixels present. If a row had 1, 2, or 3+ black pixels, I added 1 to rows_with_1/2/3. I repeated this process for columns, but instead, checking for black pixels in each column.

Features 8, 9, and 10 involved calculating the height, width, and aspect ratio of each image. In order to calculate the height of each image, I looped through the rows until a black pixel was found. Once a black pixel was found, I checked if the current row was less than the current top row and updated the top row if necessary. I did the same for the bottom row, checking if the current row was greater than the current bottom row. After looping through the entire matrix, the image height was set to the bottom row minus the top row. I used a similar approach to calculate image width, but looped through the columns instead of the rows. The aspect ratio was then calculated simply by dividing the image width by its height.

Features 11 and 12 required identifying the row and column with the highest number of black pixels in each image. To accomplish this, I iterated through each row (and column, respectively), and added up the number of black pixels to a temporary total. At the end of each row (or column), I checked if this total was greater than the current maximum, and if it was, I updated the maximum accordingly.

For feature 13, the task was to calculate the number of connected areas in each image, which was achieved using the R raster library. Firstly, the matrix was converted into a raster format. Then using the built in feature clumps, the number of clumps in the raster (i.e the number of areas where black pixels touch each other in any of the 8 possible directions) was calculated.

To calculate Feature 14, I implemented a method similar to calculating the number of connected areas, but I had to alter my approach slightly to detect eyes in the images. In order to use the clumps function, I first inverted the matrix by swapping 0's and 1's. This was necessary because the clumps function only recognizes 1's as connected areas. I then applied the clumps function to the matrix, except this time only considering the orthogonal directions. The resulting number of clumps represented the number of eyes in the image.

For Feature 15, the objective was to calculate the hollowness of the image, which is defined as the ratio of the number of white pixels in the eyes to the total number of black pixels in the image. To begin, I checked if there was only one cluster of white pixels (the white background), in which case hollowness was set to 0 (since we have no eyes). Next, I looped through each eye and calculated the number of white pixels in each. Finally, I divided this sum by the total number of black pixels in the image, which was calculated in Feature 1.

The last feature was a custom one, I opted for the curvature of the image. This choice was made because all four living objects exhibit some degree of curvature, whereas only the wine glass among non-living objects has any noticeable curvature.

As defining a fully complete curvature feature can be challenging, the approach I decided to take was to consider 3x3 submatrices of the 45x45 matrix and loop through all of them, moving along 1-column at a time. Two predefined functions checked if the submatrices contained a diagonal or straight line pattern and I then calculated curvature as diagonals divided by straights. While this method worked well for curved shapes, it was found to have one major issue: an entirely diagonal shape would be considered extremely curved. To address this, a check was added to see if there was a very high number of diagonals (90+) and if there was, curvature was set to 0. This proved to be a simple and effective solution.

```r
# setwd
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
source("section1_code.r")

# part2 feature lists
labels = c()
indexs = c()
nr_pix = c()
rows_with_1 = c()
cols_with_1 = c()
rows_with_2 = c()
cols_with_2 = c()
rows_with_3p = c()
cols_with_3p = c()
height = c()
width = c()
aspect_ratio = c()
maxrow = c()
maxcol = c()
connected_areas = c()
eyes = c()
hollowness = c()
custom = c()

# the following functions are used for the custom feature
# curvature they identify diagonal and straight patterns in
# 3x3 submatrices
diagonals <- function(mat, num_diag) {
    # check for top left to bottom right and top right to
    # bottom left patterns
    if (mat[1, 1] == 1 && mat[2, 2] == 1 && mat[3, 3] == 1) {
        num_diag <- num_diag + 1
    } else if (mat[1, 3] == 1 && mat[2, 2] == 1 && mat[3, 1] ==
        1) {
        num_diag <- num_diag + 1
    } else {
    }
    return(num_diag)
}

straights <- function(mat, num_straights) {
    # check for vertical straight and horizontal straight
    # patterns
    if (mat[1, 2] == 1 && mat[2, 2] == 1 && mat[3, 2] == 1) {
        num_straights <- num_straights + 1
    } else if (mat[2, 1] == 1 && mat[2, 2] == 1 && mat[2, 3] ==
        1) {
        num_straights <- num_straights + 1
    } else {
    }
    return(num_straights)
}

# first lets get the labels and indexs of each file
for (pgm in pgm_files) {
    # str_detect checks if a string has an exact substr
    if (str_detect(pgm, "cherry")) {
        label <- "cherry"
    } else if (str_detect(pgm, "banana")) {
        label <- "banana"
    } else if (str_detect(pgm, "lemon")) {
        label <- "lemon"
    } else if (str_detect(pgm, "tree")) {
        label <- "tree"
    } else if (str_detect(pgm, "envelope")) {
        label <- "envelope"
    } else if (str_detect(pgm, "golfclub")) {
        label <- "golfclub"
    } else if (str_detect(pgm, "pencil")) {
        label <- "pencil"
    } else if (str_detect(pgm, "wineglass")) {
        label <- "wineglass"
    }
    labels <- append(labels, label)

    # index
    index = 0
    # takes the number before the .pgm part of string
    index <- str_sub(pgm, -6, -5)
    indexs <- append(indexs, index)
}


# Loop through the tsv matrices
for (tsv_matrix in pgm_matrices) {

    # 01 nr_pix
    nr_p = 0
    # loop through entire matrix and if an entry is 1 add
```

```r
    # it to a counter
    for (row in 1:nrow(tsv_matrix)) {
        for (col in 1:ncol(tsv_matrix)) {
            if (tsv_matrix[row, col] == 1) {
                nr_p <- nr_p + 1
            }
        }

    }
    nr_pix <- append(nr_pix, nr_p)

    # 02 rows_with_1
    black_pix = 0
    row_with_1 = 0
    for (row in 1:nrow(tsv_matrix)) {
        for (col in 1:ncol(tsv_matrix)) {
            # if theres a 1 add it to a temp variable
            if (tsv_matrix[row, col] == 1) {
                black_pix <- black_pix + 1
            }
        }
        # if there was only one, row only had 1 black pixel
        if (black_pix == 1) {
            row_with_1 <- row_with_1 + 1
        }
        # reset when row finishes
        black_pix = 0
    }
    rows_with_1 <- append(rows_with_1, row_with_1)

    # 03 cols_with_1
    black_pix = 0
    col_with_1 = 0
    for (col in 1:ncol(tsv_matrix)) {
        for (row in 1:nrow(tsv_matrix)) {
            # if theres a 1 add it to a temp variable
            if (tsv_matrix[row, col] == 1) {
                black_pix <- black_pix + 1
            }
        }
        # if there was only one, column only had 1 black
        # pixel
        if (black_pix == 1) {
            col_with_1 <- col_with_1 + 1
        }
        # reset when column finishes
        black_pix = 0
    }
    cols_with_1 <- append(cols_with_1, col_with_1)

    # next four features are identical logic to above just
    # change the if statement so no need to comment
```

```
# 04 rows_with_2
black_pix = 0
row_with_2 = 0
for (row in 1:nrow(tsv_matrix)) {
    for (col in 1:ncol(tsv_matrix)) {

        if (tsv_matrix[row, col] == 1) {
            black_pix <- black_pix + 1
        }
    }
    if (black_pix == 2) {
        row_with_2 <- row_with_2 + 1
    }
    black_pix = 0
}
rows_with_2 <- append(rows_with_2, row_with_2)

# 05 cols_with_2
black_pix = 0
col_with_2 = 0
for (col in 1:ncol(tsv_matrix)) {
    for (row in 1:nrow(tsv_matrix)) {

        if (tsv_matrix[row, col] == 1) {
            black_pix <- black_pix + 1
        }
    }
    if (black_pix == 2) {
        col_with_2 <- col_with_2 + 1
    }
    black_pix = 0
}
cols_with_2 <- append(cols_with_2, col_with_2)

# 06 rows_with_3p
black_pix = 0
row_with_3p = 0
for (row in 1:nrow(tsv_matrix)) {
    for (col in 1:ncol(tsv_matrix)) {

        if (tsv_matrix[row, col] == 1) {
            black_pix <- black_pix + 1
        }
    }
    if (black_pix >= 3) {
        row_with_3p <- row_with_3p + 1
    }
    black_pix = 0
}
rows_with_3p <- append(rows_with_3p, row_with_3p)

# 07 cols_with_3p
black_pix = 0
```

```
    col_with_3p = 0
    for (col in 1:ncol(tsv_matrix)) {
        for (row in 1:nrow(tsv_matrix)) {

            if (tsv_matrix[row, col] == 1) {
                black_pix <- black_pix + 1
            }
        }
        if (black_pix >= 3) {
            col_with_3p <- col_with_3p + 1
        }
        black_pix = 0
    }
    cols_with_3p <- append(cols_with_3p, col_with_3p)

    # 08 height
    top = 46
    bottom = -1
    h = 0
    for (row in 1:nrow(tsv_matrix)) {
        for (col in 1:ncol(tsv_matrix)) {
            # if it has a black pixel, check if its the new
            # highest/lowest row if not, no need to
            # continue, break and go to the next row
            if (tsv_matrix[row, col] == 1) {
                if (row < top) {
                  top <- row
                }
                if (row > bottom) {
                  bottom <- row
                }
                break
            }
        }
    }
    # if it has a top and bottom get the abs difference
    # else keep height as 0
    if (top != 46 && bottom != -1) {
        h <- abs(top - bottom)
    }
    height <- append(height, h)

    # 09 width
    left = 46
    right = -1
    w = 0
    for (col in 1:ncol(tsv_matrix)) {
        for (row in 1:nrow(tsv_matrix)) {
            # if it has a black pixel check if its the new
            # leftmost/rightmost col if not, no need to
            # continue, break and go to the next col
            if (tsv_matrix[row, col] == 1) {
                if (col < left) {
```

```
                left <- col
              }
              if (col > right) {
                right <- col
              }
              break
          }
      }
  }
  # if it has a left and right get the abs difference
  # else keep width as 0
  if (left != 46 && right != -1) {
      w <- abs(right - left)
  }
  width <- append(width, w)

  # 10 aspect_ratio
  ratio = 1
  ratio <- round(w/h, digits = 3)

  aspect_ratio <- append(aspect_ratio, ratio)

  # 11 maxrow
  mrow = 0
  black_pix = 0
  for (row in 1:nrow(tsv_matrix)) {
      for (col in 1:ncol(tsv_matrix)) {
          # count the black pixels in a row
          if (tsv_matrix[row, col] == 1) {
              black_pix <- black_pix + 1
          }
      }
      # check if that row had more than our current max
      # and reset the counter
      if (black_pix > mrow) {
          mrow <- black_pix
      }
      black_pix = 0
  }
  maxrow <- append(maxrow, mrow)

  # 12 maxcol
  mcol = 0
  black_pix = 0
  for (col in 1:ncol(tsv_matrix)) {
      for (row in 1:nrow(tsv_matrix)) {
          # count the black pixels in a col
          if (tsv_matrix[row, col] == 1) {
              black_pix <- black_pix + 1
          }
      }
      # check if that col had more than our current max
      # and reset the counter
```

```
        if (black_pix > mcol) {
            mcol <- black_pix
        }
        black_pix = 0
    }
    maxcol <- append(maxcol, mcol)

    # 13 connected_areas

    # converting the 45x45 matrix to raster
    r <- raster(tsv_matrix, xmn = 0, xmx = 45, ymn = 0, ymx = 45)
    clumps <- clump(r, directions = 8)
    c_areas <- cellStats(clumps, max)  #returns the max clump,
    # they are 1-based indexed so if theres 2 clumps
    # outputs 2 etc

    connected_areas <- append(connected_areas, c_areas)

    # 14 eyes

    # invert matrix, set 1's to 0's and 0's to 1's
    inverted_matrix <- ifelse(tsv_matrix == 0, 1, 0)
    # converting the 45x45 matrix to raster
    r <- raster(inverted_matrix, xmn = 0, xmx = 45, ymn = 0,
        ymx = 45)

    clumps <- clump(r, directions = 4)  #orthogonal directions only

    eye <- cellStats(clumps, max) - 1  #returns the max clump,
    #-1 to get rid of white background case
    eyes <- append(eyes, eye)

    # 15 hollowness
    hollow = 0
    # freq returns information on the number of pixels in
    # clumps
    clumps_info <- freq(clumps)[, 2]

    # if only two entries in info then no eyes hence no
    # hollowness else take white pixels in each eye and sum
    # them up
    if (length(clumps_info) == 2) {
        hollow = 0
    } else {
        index <- clumps_info[3:length(clumps_info) - 1]
        white_pix <- sum(index)
        hollow = round(white_pix/nr_p, digits = 3)
    }

    hollowness <- append(hollowness, hollow)

    # 16 custom - curvature
    col_i <- 1  #keeps track of left col
```

```r
    col_j <- 3  #keeps track of right col
    row_i <- 1  #keeps track of top row
    row_j <- 3  #keeps track of bottom row
    diag <- 0  #diagonals
    strt <- 0  #straights
    for (ii in 1:length(tsv_matrix)) {

        sub_matrix <- tsv_matrix[row_i:row_j, col_i:col_j]  #make the current 3x3 mat
        diag <- diagonals(sub_matrix, diag)  #check if current mat has diagonal
        strt <- straights(sub_matrix, strt)  #check if current mat has straight

        # increment left and right column to move our 3x3
        # matrix over by 1
        col_i <- col_i + 1
        col_j <- col_j + 1

        # this is when we have finished looping through all
        if (row_j == 45 && col_j == 45) {
            # if there is very large number of diag or no
            # straight, set curvature to 0 else calculate
            # the ratio diag/strt and set it to curvature
            if (strt == 0 | diag >= 90) {
                curvature <- 0
            } else {
                curvature <- round(diag/strt, digits = 3)
            }
            custom <- append(custom, curvature)
            break
        }

        # if we reach the right edge of matrix, reset left
        # and right col and move down the top and bottom
        # row by one
        if (col_j == 45) {
            col_i <- 1
            col_j <- 3
            row_i <- row_i + 1
            row_j <- row_j + 1
        }
    }
}
# dataframe for the table
csv_df <- data.frame(labels, indexs, nr_pix, rows_with_1, cols_with_1,
    rows_with_2, cols_with_2, rows_with_3p, cols_with_3p, height,
    width, aspect_ratio, maxrow, maxcol, connected_areas, eyes,
    hollowness, custom)

directory_name = dirname(rstudioapi::getActiveDocumentContext()$path)
file_name = "/40324932_features.csv"

export_path <- paste(directory_name, file_name, sep = "")
# convert our data frame into a csv file
write.table(csv_df, export_path, sep = ",", row.names = FALSE,
    quote = FALSE)
```

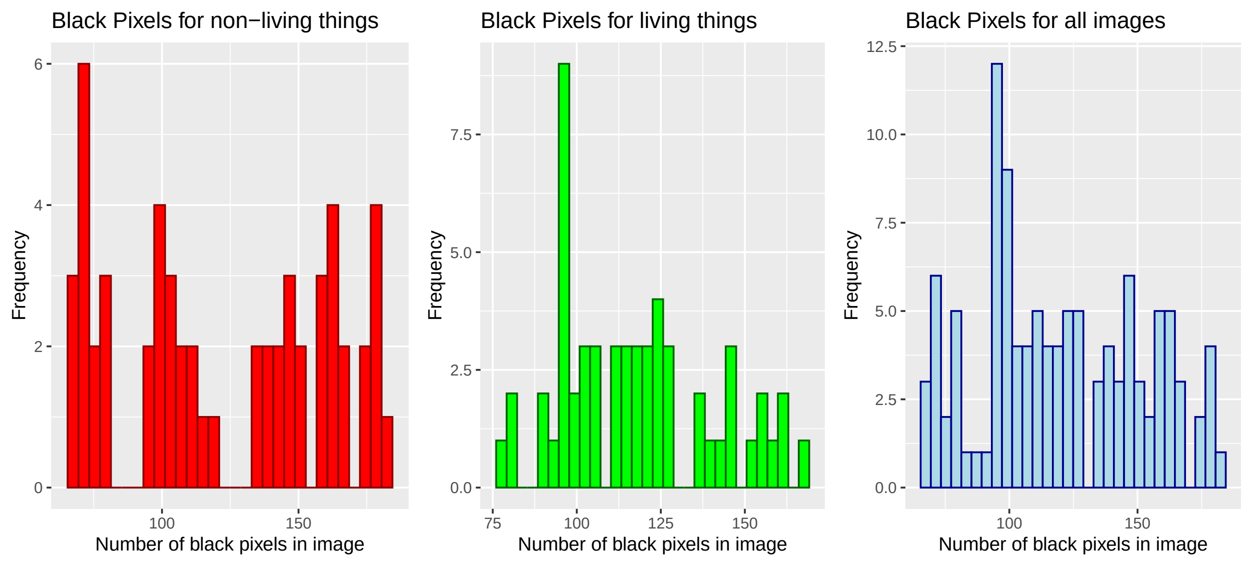
Section 3

This section examines the features that are important in differentiating between the sets of living and non-living images. Statistical analyses of the feature data are used to identify the most relevant and informative features.

Section 3.1

In order to better understand the characteristics of our image dataset, I have constructed histograms representing four features: number of black pixels per image, image height, image width, and image aspect ratio. For each of these features, I have generated separate histograms to represent non-living (red), living (green), and all images (blue).

```
require("ggplot2")
require("gridExtra")
require("dplyr")

source("section2_code.r")

setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# 1 (a), (b), (c)
data <- read.csv(paste(getwd(), "/40324932_features.csv", sep = ""))
living_df <- filter(data, labels == "banana" | labels == "cherry" |
    labels == "lemon" | labels == "tree")
non_living_df <- filter(data, labels == "envelope" | labels ==
    "golfclub" | labels == "pencil" | labels == "wineglass")

ggplot(data = non_living_df, aes(x = nr_pix)) + geom_histogram(fill = "red",
    color = "darkred", bins = 30) + ylab("Frequency") + xlab("Number of black pixels in image") +
    ggtitle("Black Pixels for non-living things") -> p1

# living
ggplot(data = living_df, aes(x = nr_pix)) + geom_histogram(fill = "green",
    color = "darkgreen", bins = 30) + ylab("Frequency") + xlab("Number of black pixels in image") +
    ggtitle("Black Pixels for living things") -> p2

# all
ggplot(data = csv_df, aes(x = nr_pix)) + geom_histogram(fill = "lightblue",
    color = "darkblue", bins = 30) + ylab("Frequency") + xlab("Number of black pixels in image") +
    ggtitle("Black Pixels for all images") -> p3

grid.arrange(p1, p2, p3, ncol = 3)
```

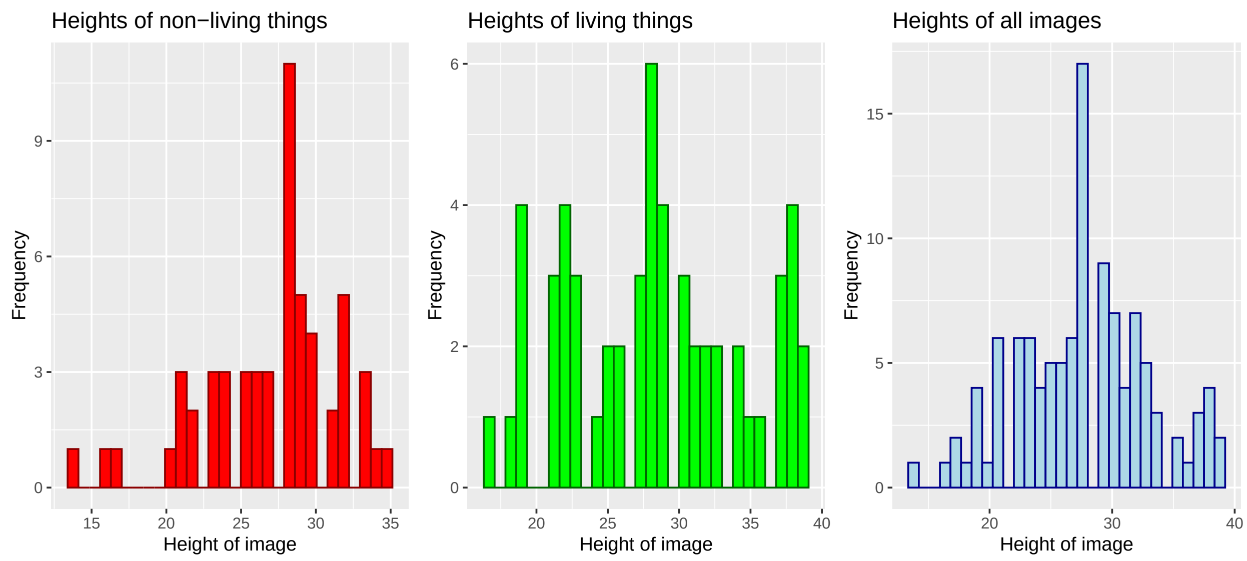


The first histogram representing the number of black pixels in non-living things exhibits a multimodal, non-skewed distribution with several peaks distributed across the range. This suggests that non-living images tend to have a wide range of values for number of black pixels.

Conversely, the histogram for living things displays a unimodal, slightly positively skewed distribution with a peak near the beginning of the range. This indicates that the mode number of black pixels for living things is 100, and that most living images have values falling within the range of 100-125.

Finally, the histogram for all images depicts a unimodal, non-skewed distribution that builds up towards a peak near the beginning of the range before declining and remaining level. As with the other histograms, the mode number of pixels for any image is 100, but the data exhibits a generally wide spread.

```r
# height non-living
ggplot(data = non_living_df, aes(x = height)) + geom_histogram(fill = "red",
    color = "darkred", bins = 30) + ylab("Frequency") + xlab("Height of image") +
    ggtitle("Heights of non-living things") -> p1

# living
ggplot(data = living_df, aes(x = height)) + geom_histogram(fill = "green",
    color = "darkgreen", bins = 30) + ylab("Frequency") + xlab("Height of image") +
    ggtitle("Heights of living things") -> p2

# all
ggplot(data = csv_df, aes(x = height)) + geom_histogram(fill = "lightblue",
    color = "darkblue", bins = 30) + ylab("Frequency") + xlab("Height of image") +
    ggtitle("Heights of all images") -> p3

grid.arrange(p1, p2, p3, ncol = 3)
```

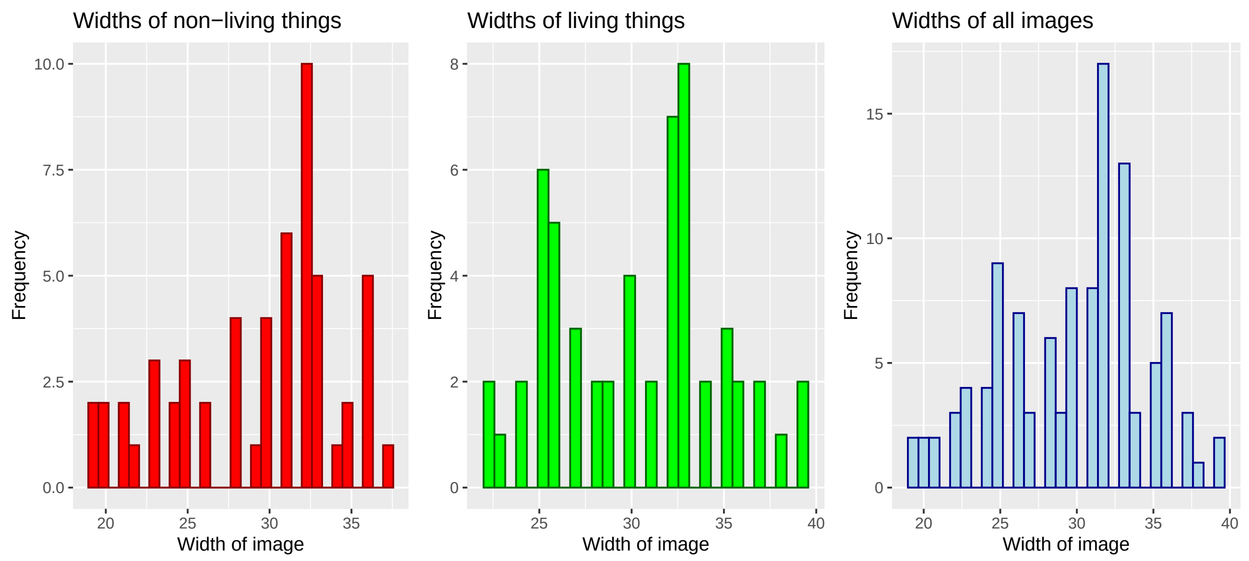


The first histogram, representing the heights of non-living things displays a unimodal, negatively skewed distribution with a significant peak towards the end of the range. This suggests that the majority of non-living images have heights clustering around 27-30.

In contrast, the histogram for living things exhibits a multimodal, non-skewed distribution with multiple significant peaks. Notably, the largest peak occurs at a height of 28, indicating that this is the most common height for living things.

Finally, the histogram for all images depicts a unimodal, non-skewed distribution that roughly resembles a normal distribution. Consistent with the previous two histograms, this distribution shows that the majority of images have a height of 28.

```r
# width non-living
ggplot(data = non_living_df, aes(x = width)) + geom_histogram(fill = "red",
    color = "darkred", bins = 30) + ylab("Frequency") + xlab("Width of image") +
    ggtitle("Widths of non-living things") -> p1

# living
ggplot(data = living_df, aes(x = width)) + geom_histogram(fill = "green",
    color = "darkgreen", bins = 30) + ylab("Frequency") + xlab("Width of image") +
    ggtitle("Widths of living things") -> p2

# all
ggplot(data = csv_df, aes(x = width)) + geom_histogram(fill = "lightblue",
    color = "darkblue", bins = 30) + ylab("Frequency") + xlab("Width of image") +
    ggtitle("Widths of all images") -> p3

grid.arrange(p1, p2, p3, ncol = 3)
```

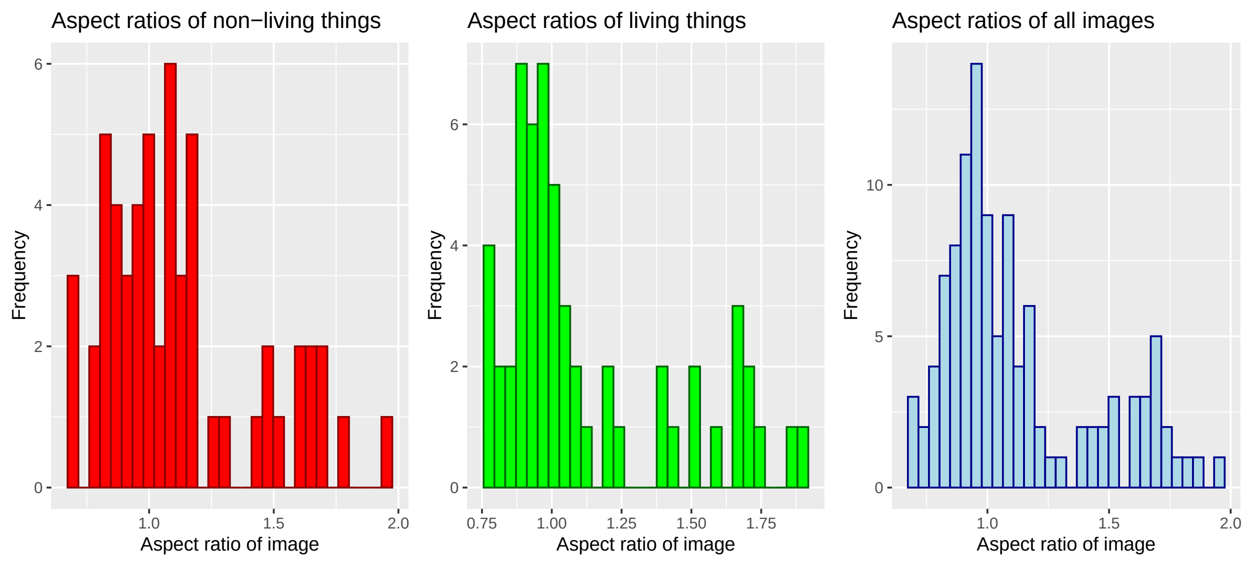


The first histogram, representing the widths of non-living things, shows a unimodal distribution with a negative skew. This suggests that the majority of widths are clustered around a single value, with a tail of lower values. Specifically, the most common width for non-living things is around 32/33.

The second histogram, representing the widths of living things, displays a bimodal distribution with two peaks at 25/26 and 32/33. This indicates that there are two groups of widths that are most common for living things. The distribution is not skewed, as the two peaks are roughly symmetric and the rest of data is quite level.

Finally, the third histogram, representing the widths of all images, shows a multimodal distribution with two smaller peaks near the tails and a larger peak at 32/33. The distribution takes a slight negative skew, indicating that the end tail is slightly more significant than the front. Overall, the majority of data falls in the 25-35 range, with the peak at 32/33 indicating that this is the modal value of widths of all images.

```r
# aspect_ratio non-living
ggplot(data = non_living_df, aes(x = aspect_ratio)) + geom_histogram(fill = "red",
    color = "darkred", bins = 30) + ylab("Frequency") + xlab("Aspect ratio of image") +
    ggtitle("Aspect ratios of non-living things") -> p1

# living
ggplot(data = living_df, aes(x = aspect_ratio)) + geom_histogram(fill = "green",
    color = "darkgreen", bins = 30) + ylab("Frequency") + xlab("Aspect ratio of image") +
    ggtitle("Aspect ratios of living things") -> p2

# all
ggplot(data = csv_df, aes(x = aspect_ratio)) + geom_histogram(fill = "lightblue",
    color = "darkblue", bins = 30) + ylab("Frequency") + xlab("Aspect ratio of image") +
    ggtitle("Aspect ratios of all images") -> p3

grid.arrange(p1, p2, p3, ncol = 3)
```

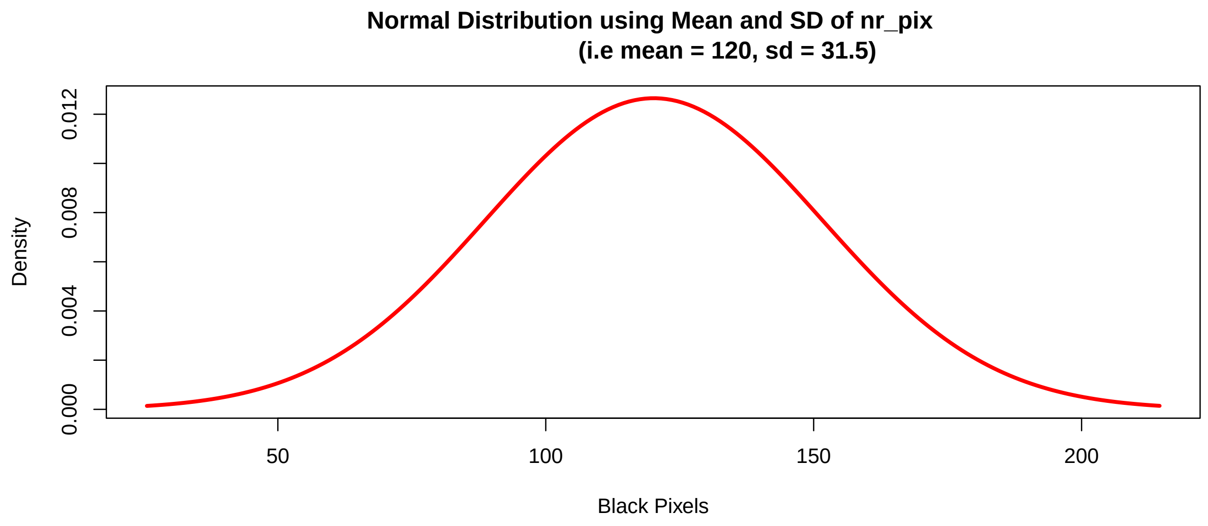


Upon examining the histograms for aspect ratios, we observe a similar shape across all three graphs. Specifically, each histogram exhibits a bimodal distribution with a strong positive skew, as evidenced by a very large peak at the beginning of the data followed by a smaller peak towards the end. Furthermore, the data indicates that the majority of aspect ratios take values of 1, with a decent number around 1.7 as well.

This suggests that the aspect_ratio feature may not be particularly informative for differentiating between living and non-living things in our future analyses.

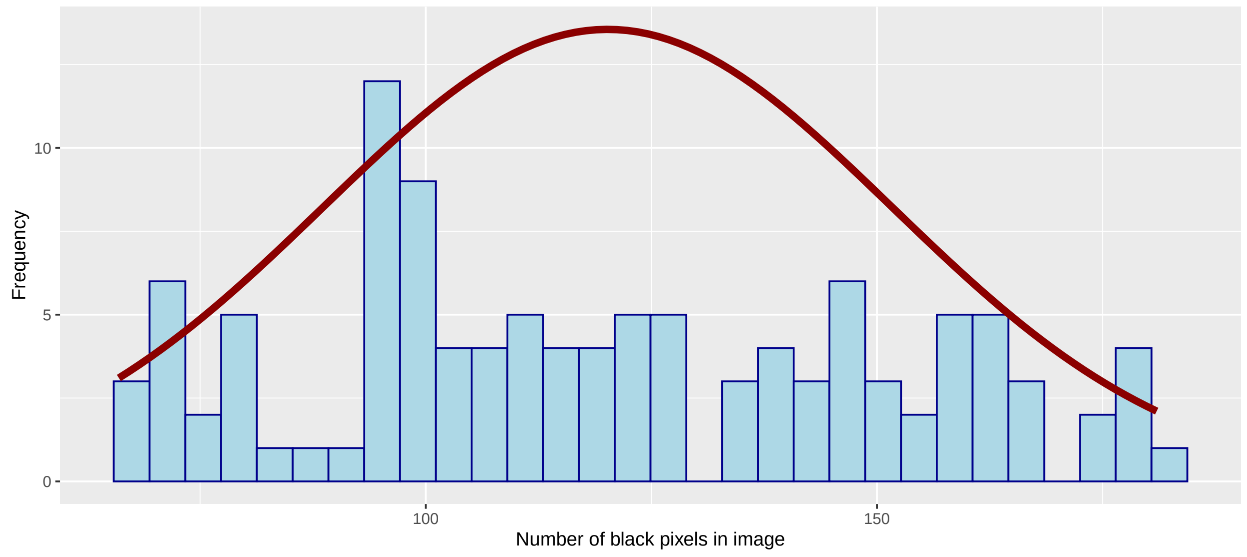
Section 3.2

```
# calculating mean, var and sd
mean_nr_pix <- mean(nr_pix)
var_nr_pix <- var(nr_pix)
sd_nr_pix <- sqrt(var_nr_pix)

# setting range to 3 sd above and below mean
x = (mean_nr_pix - 3 * sd_nr_pix):(mean_nr_pix + 3 * sd_nr_pix)
# normal densities
densities = dnorm(x, mean_nr_pix, sd_nr_pix)

# plotting normal distribution
plot(x, densities, type = "l", col = "red", lty = 1, lwd = 3,
    xlab = "Black Pixels", ylab = "Density", main = "Normal Distribution using Mean and SD of nr_pix
                      (i.e mean = 120, sd = 31.5)")
```



Above is the theoretical normal distribution curve for number of black pixels in all the images. Now let us consider plotting this against our actual data and analyse what we see.

```
ggplot(data = csv_df, aes(x = nr_pix)) + geom_histogram(fill = "lightblue",
    color = "darkblue", bins = 30) + ylab("Frequency") + xlab("Number of black pixels in image") +
    stat_function(fun = function(x) dnorm(x, mean = mean_nr_pix,
        sd = sd_nr_pix) * 300 * 25/7, color = "darkred", linewidth = 2)
```



As described in the previous section, our data does not follow a normal distribution. The peak of our data occurs roughly 1 standard deviation away from the mean of the normal distribution. The tails of the distribution are also much larger than expected for a normal distribution, with less data points around the mean. The graph here and in 3.1 also indicate a slight positive skew in the data, with a peak at the front and tailing off towards the end.

Section 3.3

The following code shows us the 5% cut off value at the top range of our above normal distribution. This tells us the probability of getting a number of pixels greater than this is less than 5%.

```
rtail = qnorm(0.95, mean_nr_pix, sd_nr_pix)
print(rtail)
```

```
## [1] 172.0294
```

A value of 172 falls outside the expected 5% cutoff range for our data. In our sample of 112 images, 7 images fell within this range, which is slightly higher than expected. The presence of several values around 170 also indicates more data than expected around the 5% cutoff. Taking all this in to consideration, in the next section, we will use randomization tests to analyze the data properly since we cannot assume a normal distribution for each feature and carry out t-tests.

Section 3.4

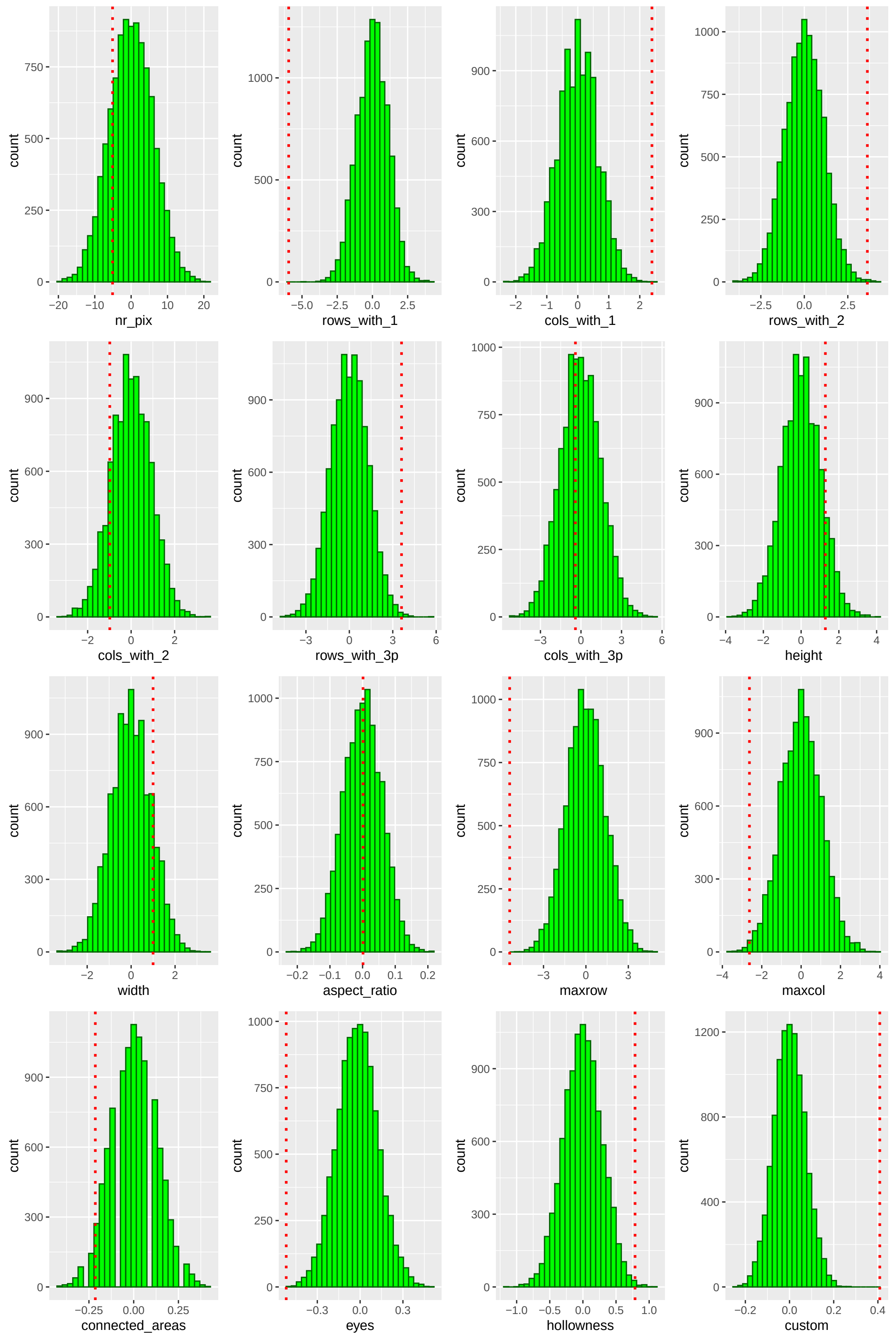
I have included the code below that was used to conduct statistical analyses on our data. The aim was to determine if there is a significant difference between living and non-living items for each of the features we tested. We began with the null hypothesis that there would be no difference in the mean of living and non-living items.

Randomization tests were used to analyze each feature, as most of our data did not meet the normal distribution assumptions of the Central Limit Theorem. I created a histogram (green) for each feature to show the distribution of each feature under the null hypothesis.

A dotted red line is displayed on each graph to show where our observed value was located in relation to the curve. If our observed value had a p-value of less than 0.05, we would reject the null hypothesis and accept that there is a significant difference between living and non-living items for that feature. I have included a table representing the p-value for each feature under the graphs.

```
library(knitr)
# 3.4
all_funcs <- list(nr_pix, rows_with_1, cols_with_1, rows_with_2,
    cols_with_2, rows_with_3p, cols_with_3p, height, width, aspect_ratio,
    maxrow, maxcol, connected_areas, eyes, hollowness, custom)

names(all_funcs) <- c("nr_pix", "rows_with_1", "cols_with_1",
    "rows_with_2", "cols_with_2", "rows_with_3p", "cols_with_3p",
    "height", "width", "aspect_ratio", "maxrow", "maxcol", "connected_areas",
    "eyes", "hollowness", "custom")

# keep track of plots and index
index <- 1
num_of_plot <- 1
plots <- list()
p_vals <- c()
for (f in all_funcs) {

    # Seperate the living things and place them in a list
    l1 <- f[1:28]
    l2 <- f[57:70]
    l3 <- f[85:98]
    living_things <- c(l1, l2, l3)

    # Seperate the nonliving things and place them in a
    # list
    nl1 <- f[29:56]
    nl2 <- f[71:84]
    nl3 <- f[99:112]
```

```r
non_living_things <- c(nl1, nl2, nl3)

# Observed difference in means between living and
# non-living:
obsdiff = mean(living_things) - mean(non_living_things)
nsims = 10000

# Randomization test for difference in means
All <- c(living_things, non_living_things)  # put all values in a single vector

nulldiffs <- rep(0, nsims)
# nulldiffs will contain all our simulated differences
# in means (assuming the null hypothesis is true).
# Initialise with 0s.

for (ni in 1:nsims) {
    # randomly reorder `All`
    randomA = sample(All)
    # create the random group partition
    randomLiving = randomA[1:length(living_things)]
    randomNonLiving = randomA[(length(living_things) + 1):length(All)]
    # calculate the differences in means for the two
    # groups and add it to our result vector
    nulldiffs[ni] <- mean(randomLiving) - mean(randomNonLiving)
}

# What proportion of the differences in means
# calculated with randomization are greater in
# magnitude than than the observed, true difference in
# means?
pvalue = sum(abs(nulldiffs) > abs(obsdiff))/nsims

# 95% of null-distribution differences lie in this
# range:
quantile(nulldiffs, 0.025)
quantile(nulldiffs, 0.975)

# data frame for plotting:
df <- as.data.frame(nulldiffs)
df$group <- "randomization"
colnames(df) <- c("test.statistic", "group")
df$group <- as.factor(df$group)

# get the current variable name we are analysing
name <- names(all_funcs[index])
index <- index + 1

# Generate Histograms of the data, by group
p1 <- ggplot(df, aes(x = test.statistic, fill = group, color = group)) +
    geom_histogram(position = "identity", colour = "darkgreen",
        fill = "green", bins = 30) + geom_vline(xintercept = obsdiff,
    linetype = "dotted", color = "red", linewidth = 1) +
    xlab(name) + theme(legend.position = "none")  # no legend
    if (pvalue < 0.05) {
        p_vals <- append(p_vals, pvalue)
    }
    # placing the plots in a list so we can produce a grid
    # arranged version later on.
    plots[[num_of_plot]] <- p1
    num_of_plot <- num_of_plot + 1
}
```

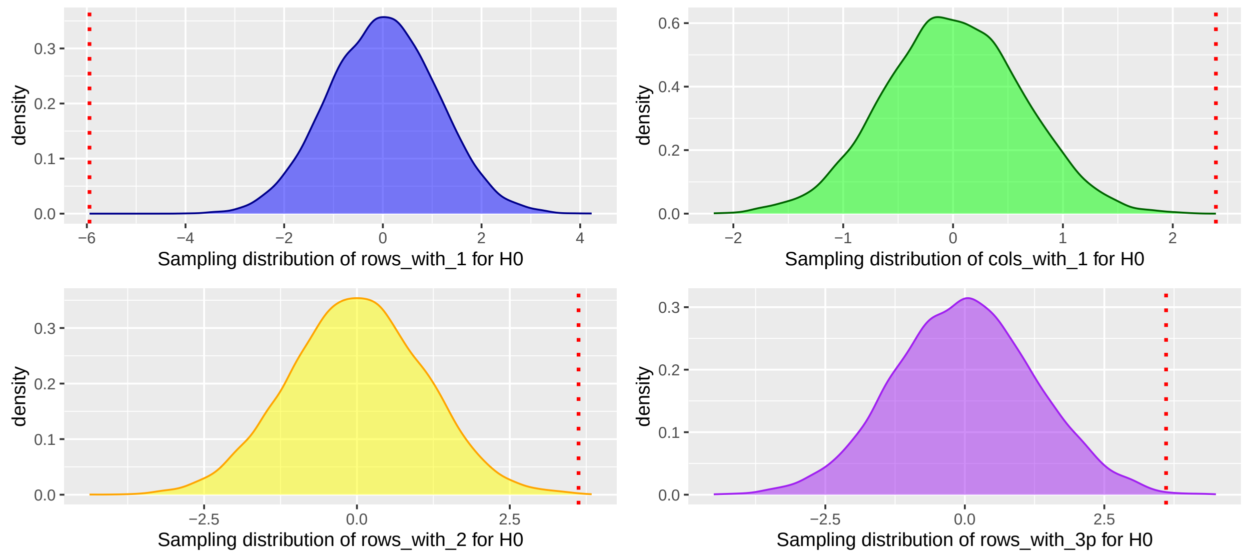

count
nr_pix
rows_with_1
cols_with_1
rows_with_2
cols_with_2
rows_with_3p
cols_with_3p
height
width
aspect_ratio
maxrow
maxcol
connected_areas
eyes
hollowness
custom

```r
# creating a table of all significant features
results <- data.frame(c("rows_with_1", "cols_with_1", "rows_with_2",
    "rows_with_3p", "maxrow", "maxcol", "eyes", "hollowness",
    "custom"), p_vals)

kable(results, col.names = c("feature", "p-value"), caption = "A table describing the p-value in each o
```

Table 1: A table describing the p-value in each of the above graphs for each significant feature:

feature	p-value
rows_with_1	0.0000
cols_with_1	0.0001
rows_with_2	0.0019
rows_with_3p	0.0047
maxrow	0.0000
maxcol	0.0108
eyes	0.0001
hollowness	0.0056
custom	0.0000

As shown by the above table and graphs there are 9 features which have a significant p-value (<0.05). Let us look closer into each of these 9 cases:



In the above four graphs we can see the significance of the four features: rows_with_1, cols_with_1, rows_with_2, and rows_with_3p. Here the red line represented the observed difference in means we had. (i.e living thing mean - non living thing mean)

Our first graph describing the randomization tests carried out on the feature rows_with_1 should maybe not come as a very surprising result since the only two features that have rows_with_1 in most cases is golfclubs and wineglasses. Meaning that we would expect our observed value to be very high compared to a randomly sampled group, which is reflected in the graph. It's worth noting that this feature is actually our most significant of all.

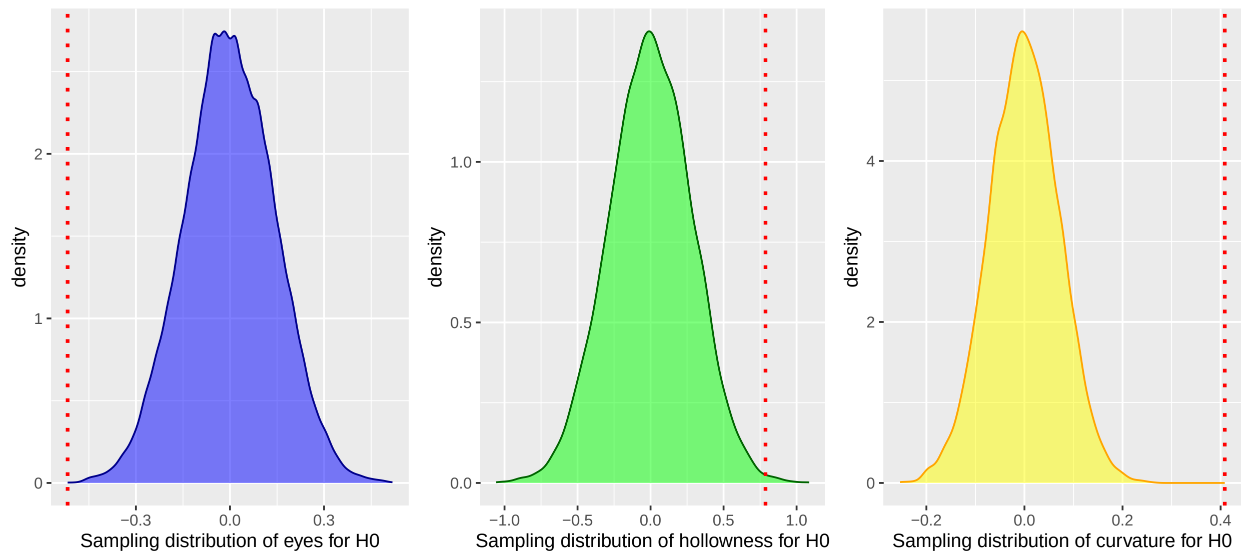
Observing the graph for cols_with_1, this isn't a surprising result since nearly all the significant data from cols_with_1 comes from our trees and some but much less from wineglasses. This means that we should expect our observed value to be very different to a randomly sampled version of our data.

For the feature rows_with_2, again, this isn't surprising since the only significant images here are cherries, bananas, golfclubs and trees. Meaning 3 of our living things have high rows_with_2 as oppose to 1 nonliving.

Our graph for rows_with_3p is very significant again, this is because all of our living things have very high rows_with_3p while only the pencil and envelope have such values for the non-living things.

Since the p-value's for each feature above have a value of significance (<0.05), they will all be worth considering in section 4 when we try to predict whether an item is living or non-living.

Now lets take a look at the next 3 significant features in distinguishing between living and non-living items.



In the above three graphs we can see the significance of the three features: eyes, hollowness and our custom feature, curvature.

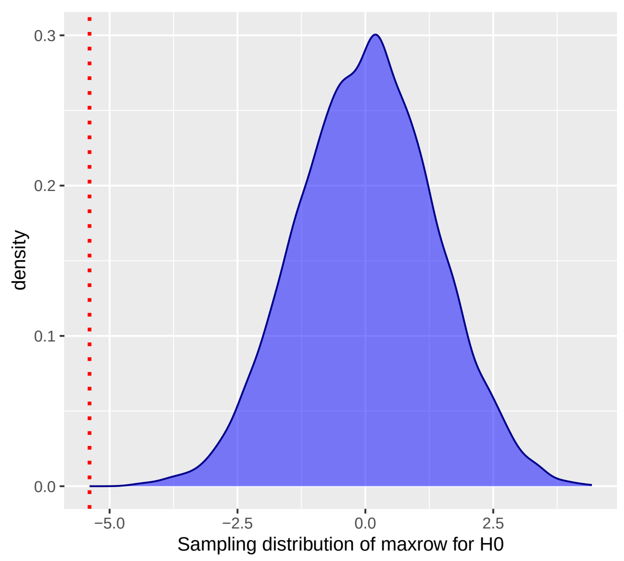
Observing the results for the eyes feature, we can see it is a very significant feature and this is due to the following: Our wineglasses and trees have 0 eyes so can be disregarded. However, two of our non-living items (pencils and envelopes) have 2 eyes each and the golfclubs have 1, while all of our remaining 3 living items only have 1 eye each.

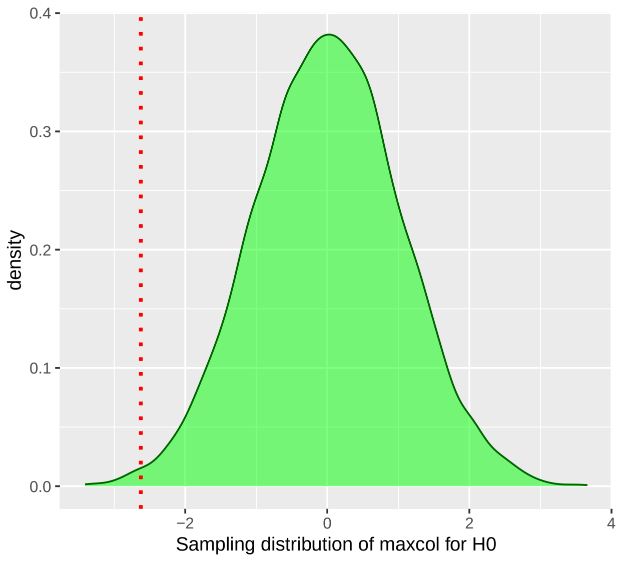
Our hollowness feature is highly significant and this is maybe not immediately obvious to the naked eye why this should be the case. However, observing the hollowness column in the features file we can see that all the high values come from lemons, cherries and bananas (i.e 3 living things). Hence, we should expect a high observed value that's difficult to replicate with a random sample, which is why the red line falls far right.

Our custom feature curvature is no surprise since, my rationale for choosing this feature was on the basis that the living things should have significantly higher curvature than the non-living things.

Once again, since the p-values for each feature above have a value of significance (<0.05), they will all be worth considering in section 4 when we try to predict whether an item is living or non-living.

Now finally let us look at the final 2 significant features:





Upon examining the graph of maxrow, we can see that it has a very significant value. This stems from envelope having significantly higher maxrow values than all other features, all of which have similar values.

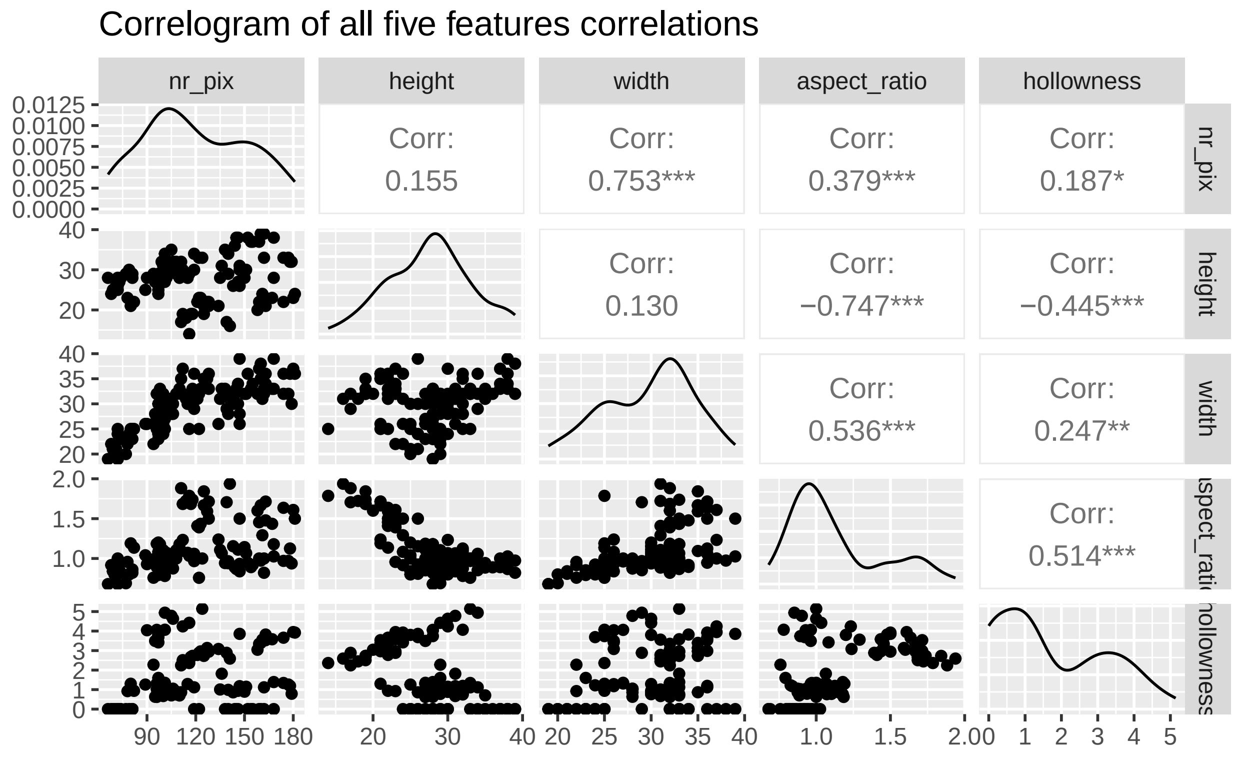
For maxcol, looking at the data again we can see in general the maxcol values of envelopes and wineglasses (two non-living items) are much greater than that of all other images which generally have similar values.

Altogether, we have nine potentially useful features at determining if an item is living or non-living and we now must determine the optimal set that can do this parsimoniously and accurately.

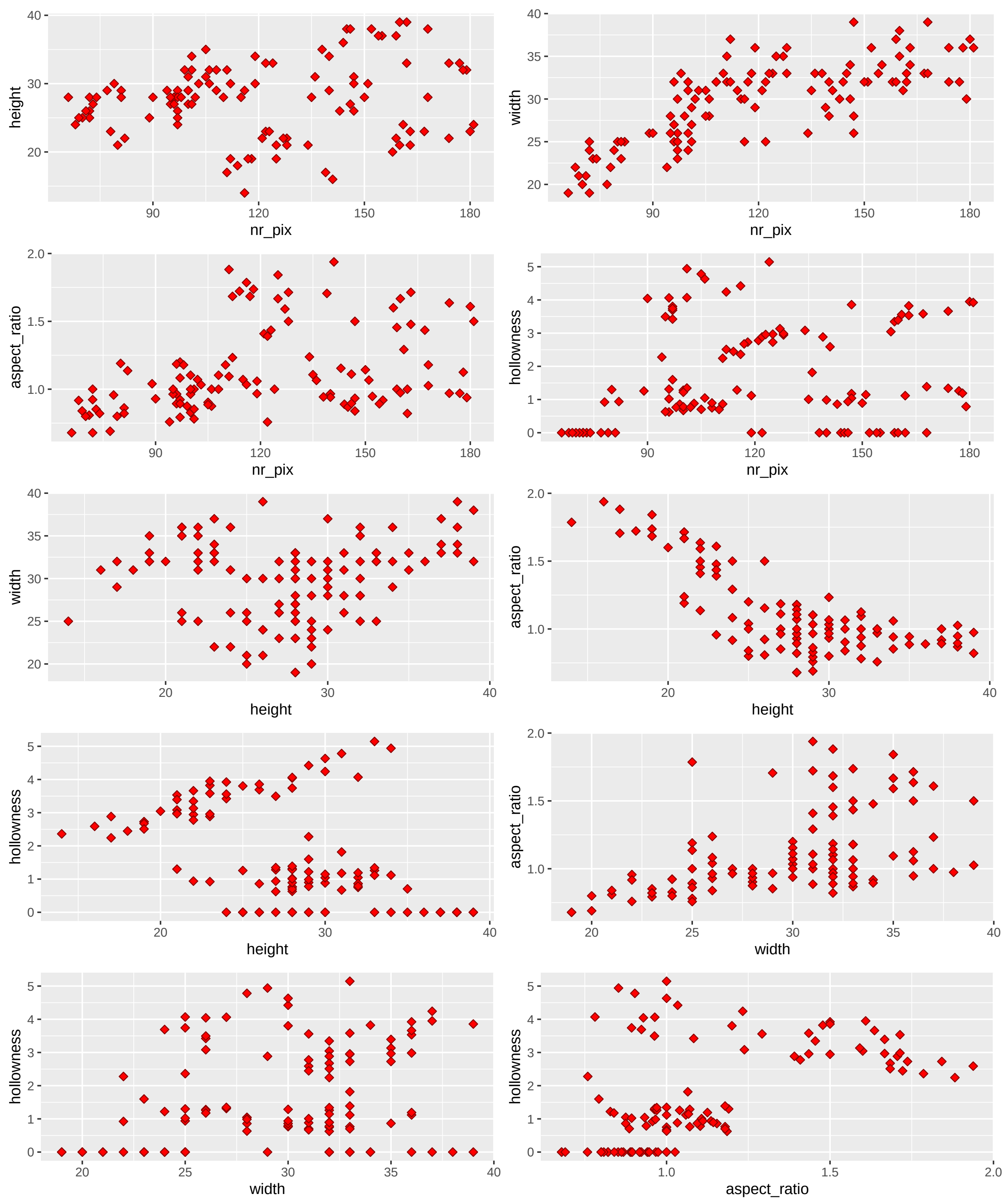
Section 3.5

In this section, we will investigate the correlation between the five features in the set (nr_pix, height, width, aspect_ratio, hollowness) by analyzing appropriate graphs and tables for each pair of features.

```
require("GGally")
five_features <- list(nr_pix, height, width, aspect_ratio, hollowness)

names(five_features) <- c("nr_pix", "height", "width", "aspect_ratio",
    "hollowness")

five_features_df <- data.frame(nr_pix, height, width, aspect_ratio,
    hollowness)

feature1 <- c()
feature2 <- c()
correlation <- c()
p_value <- c()

plots <- list()
index <- 1

for (i in 1:length(five_features) - 1) {

    for (j in (i + 1):(length(five_features))) {

        if (j == 1) {
            break
        }
        nameX <- names(five_features[i])
        nameY <- names(five_features[j])

        feature1 <- append(feature1, nameX)
        feature2 <- append(feature2, nameY)

        cor_test <- cor.test(five_features_df[[nameX]], five_features_df[[nameY]])
        correlation <- append(correlation, cor_test$estimate)
        p_value <- append(p_value, cor_test$p.value)

        p1 <- ggplot(data = five_features_df, aes(x = .data[[nameX]],
            y = .data[[nameY]])) + geom_point(aes(x = .data[[nameX]],
            y = .data[[nameY]]), size = 2, shape = 23, colour = "darkred",
            fill = "red")

        plots[[index]] <- p1
        index <- index + 1

    }

}
ggpairs(five_features_df, title = "Correlogram of all five features correlations")
```



```
grid.arrange(grobs = plots, ncol = 2)
```



```
results <- data.frame(feature1, feature2, correlation, p_value)

kable(results, caption = "A table describing the correlation between each pair of features:")
```

Table 2: A table describing the correlation between each pair of features:

feature1	feature2	correlation	p_value
nr_pix	height	0.1551667	0.1023390
nr_pix	width	0.7529282	0.0000000
nr_pix	aspect_ratio	0.3785212	0.0000387
nr_pix	hollowness	0.1868774	0.0485007
height	width	0.1298389	0.1724246
height	aspect_ratio	-0.7471762	0.0000000
height	hollowness	-0.4451492	0.0000009
width	aspect_ratio	0.5361277	0.0000000
width	hollowness	0.2472191	0.0085915
aspect_ratio	hollowness	0.5138411	0.0000000

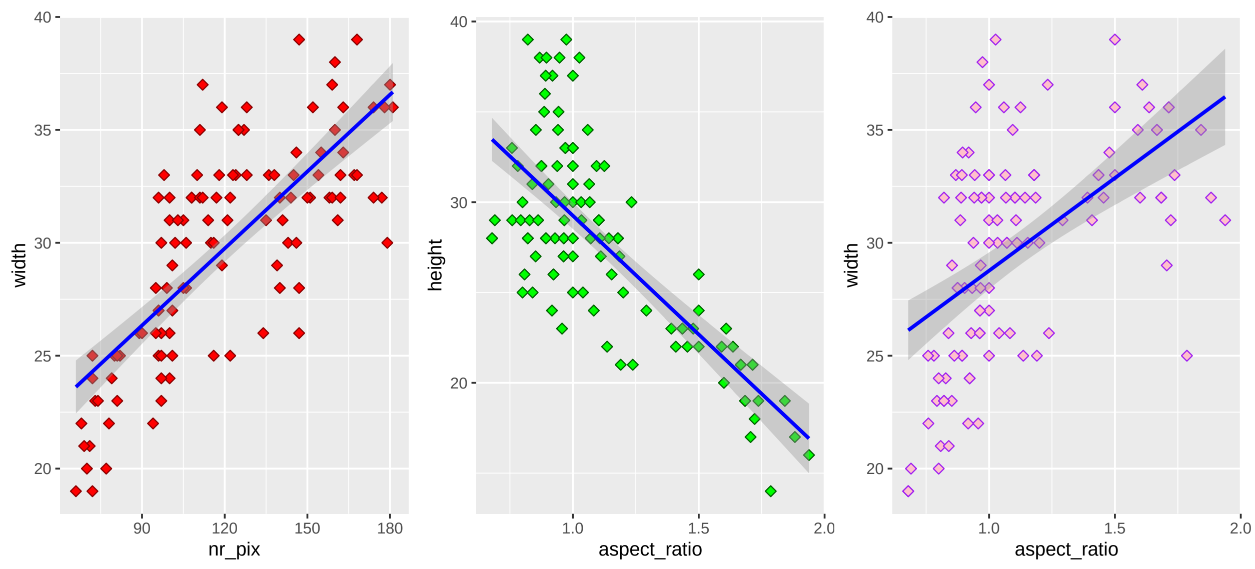
Upon examining the table above, it is evident that seven out of the ten pairs exhibit very significant p-values. Of these pairs, four demonstrate a strong correlation (>|0.5|), while two standout features exhibit a very strong correlation (>|0.7|). Specifically, the four pairs that exhibit a strong correlation are {nr_pix, width}, {height, aspect_ratio}, {width, aspect_ratio} and {aspect_ratio, hollowness}. The first three pairs are intuitive, given the definitions of each feature. Lets take a closer look into these three cases and explain the causes.

Firstly, the pair {nr_pix, width} (cor = 0.75) is logical since width is defined as the difference between the left-most and right-most pixel rows. Therefore, if an image contains a high number of black pixels, we can expect the height to be large (or small, conversely).

Similarly, the high correlation observed in the pair {height, aspect_ratio} (cor = -0.747) is unsurprising when we consider the definition of aspect ratio, which is the width divided by height. As height is used in the aspect ratio calculation, it is reasonable to observe a high correlation between these two features. Additionally, the negative correlation between height and aspect ratio makes sense since height is the divisor in the equation; hence, larger height values typically result in smaller aspect ratios.

Using the same reasoning as above, we can expect the pair {width, aspect_ratio} (cor = 0.53) to exhibit high and positive correlation since width is the multiplier in the aspect ratio equation. However, the correlation is not as strong as that observed in {height, aspect_ratio}, as one might expect to be the case. This discrepancy is most likely caused by the fact that height values tend to be larger than width values in our dataset, hence they have a greater influence on the aspect ratio calculation which results in a stronger correlation.

For a more detailed examination of these three correlations, please refer to the graphs provided below.



Section 4

Section 4.1

In this section, we aim to predict aspect ratio using the other features we have calculated. To achieve this, we want to fit a multiple regression model that is as parsimonious as possible. To determine the most suitable variables for the model, we will try both forward and backward selection methods. The method that yields a better adjusted R squared value will be used to create the final multiple regression model.

We will begin by examining the backward selection method:

```r
# set wd
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))


# 4.1 backwards selection
lmfitted = lm(aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 +
    rows_with_2 + cols_with_2 + rows_with_3p + cols_with_3p +
    height + width + maxrow + maxcol + connected_areas + eyes +
    hollowness + custom, data = csv_df)

backwards_selection <- step(lmfitted, direction = "backward")
```

```
## Start:  AIC=-656.1
## aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_2 +
##     cols_with_2 + rows_with_3p + cols_with_3p + height + width +
##     maxrow + maxcol + connected_areas + eyes + hollowness + custom
##
##
## Step:  AIC=-656.1
## aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_2 +
##     cols_with_2 + rows_with_3p + cols_with_3p + height + maxrow +
##     maxcol + connected_areas + eyes + hollowness + custom
##
##
## Step:  AIC=-656.1
## aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_2 +
##     cols_with_2 + rows_with_3p + cols_with_3p + maxrow + maxcol +
##     connected_areas + eyes + hollowness + custom
##
##                   Df Sum of Sq     RSS     AIC
## - custom           1   0.00005 0.24924 -658.08
## - maxcol           1   0.00063 0.24982 -657.82
## - nr_pix           1   0.00133 0.25053 -657.50
## - hollowness       1   0.00259 0.25178 -656.94
## - eyes             1   0.00379 0.25298 -656.41
## - connected_areas  1   0.00399 0.25319 -656.32
## <none>                         0.24919 -656.10
## - maxrow           1   0.00550 0.25469 -655.65
## - cols_with_3p     1   0.26280 0.51200 -577.45
## - cols_with_2      1   0.35307 0.60227 -559.26
## - cols_with_1      1   0.36765 0.61684 -556.58
## - rows_with_3p     1   0.63529 0.88449 -516.22
## - rows_with_1      1   1.26859 1.51778 -455.74
## - rows_with_2      1   1.72187 1.97107 -426.47
##
## Step:  AIC=-658.08
## aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_2 +
##     cols_with_2 + rows_with_3p + cols_with_3p + maxrow + maxcol +
##     connected_areas + eyes + hollowness
##
##                   Df Sum of Sq     RSS     AIC
## - maxcol           1   0.00064 0.24988 -659.79
## - nr_pix           1   0.00147 0.25071 -659.42
## - hollowness       1   0.00268 0.25192 -658.88
## - eyes             1   0.00410 0.25334 -658.25
## - connected_areas  1   0.00411 0.25335 -658.24
## <none>                         0.24924 -658.08
## - maxrow           1   0.00599 0.25523 -657.42
## - cols_with_3p     1   0.28409 0.53333 -574.88
## - cols_with_1      1   0.36813 0.61738 -558.49
## - cols_with_2      1   0.37424 0.62348 -557.39
## - rows_with_3p     1   0.71824 0.96748 -508.17
## - rows_with_1      1   1.59905 1.84829 -435.67
## - rows_with_2      1   1.84974 2.09899 -421.43
##
## Step:  AIC=-659.79
## aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_2 +
##     cols_with_2 + rows_with_3p + cols_with_3p + maxrow + connected_areas +
##     eyes + hollowness
##
##                   Df Sum of Sq     RSS     AIC
## - nr_pix           1   0.00091 0.25079 -661.38
## - hollowness       1   0.00261 0.25249 -660.63
## - connected_areas  1   0.00348 0.25336 -660.24
## - eyes             1   0.00360 0.25348 -660.19
## <none>                         0.24988 -659.79
## - maxrow           1   0.00782 0.25770 -658.34
## - cols_with_3p     1   0.33273 0.58262 -566.98
## - cols_with_1      1   0.36996 0.61984 -560.04
## - cols_with_2      1   0.42125 0.67113 -551.14
## - rows_with_3p     1   0.78997 1.03985 -502.09
## - rows_with_1      1   1.59991 1.84980 -437.58
## - rows_with_2      1   1.88888 2.13877 -421.33
##
## Step:  AIC=-661.38
## aspect_ratio ~ rows_with_1 + cols_with_1 + rows_with_2 + cols_with_2 +
##     rows_with_3p + cols_with_3p + maxrow + connected_areas +
##     eyes + hollowness
##
##                   Df Sum of Sq    RSS     AIC
## - eyes             1    0.0027 0.2535 -662.19
## - connected_areas  1    0.0043 0.2551 -661.48
## <none>                         0.2508 -661.38
## - hollowness       1    0.0062 0.2570 -660.64
## - maxrow           1    0.0079 0.2587 -659.90
## - cols_with_1      1    0.4923 0.7431 -541.73
## - cols_with_2      1    0.5072 0.7580 -539.50
## - cols_with_3p     1    0.7742 1.0250 -505.70
## - rows_with_3p     1    1.4624 1.7132 -448.18
## - rows_with_1      1    1.9401 2.1909 -420.63
## - rows_with_2      1    3.4285 3.6793 -362.57
##
## Step:  AIC=-662.19
## aspect_ratio ~ rows_with_1 + cols_with_1 + rows_with_2 + cols_with_2 +
##     rows_with_3p + cols_with_3p + maxrow + connected_areas +
##     hollowness
##
##                   Df Sum of Sq    RSS     AIC
## - connected_areas  1    0.0046 0.2580 -662.19
## <none>                         0.2535 -662.19
## - hollowness       1    0.0057 0.2592 -661.71
## - maxrow           1    0.0109 0.2644 -659.47
## - cols_with_2      1    0.5055 0.7590 -541.35
## - cols_with_3p     1    0.7721 1.0256 -507.64
## - cols_with_1      1    0.7901 1.0436 -505.69
## - rows_with_3p     1    1.4903 1.7437 -448.20
## - rows_with_1      1    2.0800 2.3335 -415.57
## - rows_with_2      1    3.4270 3.6805 -364.53
##
## Step:  AIC=-662.19
## aspect_ratio ~ rows_with_1 + cols_with_1 + rows_with_2 + cols_with_2 +
##     rows_with_3p + cols_with_3p + maxrow + hollowness
##
##                Df Sum of Sq    RSS     AIC
## - hollowness    1    0.0027 0.2607 -663.03
## <none>                      0.2580 -662.19
## - maxrow        1    0.0065 0.2646 -661.40
## - cols_with_2   1    0.6805 0.9385 -519.58
## - cols_with_1   1    0.7918 1.0498 -507.03
## - rows_with_1   1    2.0777 2.3357 -417.46
## - cols_with_3p  1    2.5238 2.7818 -397.88
## - rows_with_3p  1    2.9408 3.1989 -382.24
## - rows_with_2   1    3.4254 3.6835 -366.44
```

```
##
## Step:  AIC=-663.03
## aspect_ratio ~ rows_with_1 + cols_with_1 + rows_with_2 + cols_with_2 +
##     rows_with_3p + cols_with_3p + maxrow
##
##                Df Sum of Sq    RSS     AIC
## <none>                      0.2607 -663.03
## - maxrow        1    0.0073 0.2680 -661.95
## - cols_with_2   1    0.6816 0.9424 -521.12
## - cols_with_1   1    0.9221 1.1828 -495.67
## - cols_with_3p  1    2.5941 2.8549 -396.98
## - rows_with_1   1    2.6495 2.9102 -394.83
## - rows_with_3p  1    3.1002 3.3610 -378.70
## - rows_with_2   1    3.4232 3.6839 -368.43
```

As seen from the above call the backwards selection gives us the following formula: aspect_ratio ~ rows_with_1 + cols_with_1 + rows_with_2 + cols_with_2 + rows_with_3p + cols_with_3p + maxrow + eyes + hollowness + custom

Now, let us examine the forward selection method:

```
# forward selection
fitstart = lm(aspect_ratio ~ 1, data = csv_df)

forwards_selection <- step(fitstart, direction = "forward", scope = formula(lmfitted))
```

```
## Start:  AIC=-260.92
## aspect_ratio ~ 1
##
##                   Df Sum of Sq     RSS     AIC
## + height           1    5.9779  4.7300 -350.43
## + cols_with_3p     1    4.6335  6.0744 -322.41
## + width            1    3.0778  7.6301 -296.88
## + eyes             1    2.8333  7.8745 -293.35
## + hollowness       1    2.8272  7.8806 -293.26
## + maxrow           1    1.9032  8.8047 -280.84
## + rows_with_1      1    1.8940  8.8138 -280.72
## + rows_with_2      1    1.8616  8.8462 -280.31
## + nr_pix           1    1.5342  9.1736 -276.24
## + cols_with_2      1    1.3581  9.3497 -274.11
## + cols_with_1      1    0.6897 10.0181 -266.38
## + connected_areas  1    0.5638 10.1441 -264.98
## + custom           1    0.2252 10.4827 -261.30
## <none>                         10.7078 -260.92
## + rows_with_3p     1    0.1448 10.5630 -260.45
## + maxcol           1    0.0727 10.6352 -259.69
##
## Step:  AIC=-350.43
## aspect_ratio ~ height
##
##                   Df Sum of Sq    RSS     AIC
## + width            1    4.3660 0.3639 -635.68
## + cols_with_3p     1    3.0327 1.6973 -463.22
## + nr_pix           1    2.6825 2.0474 -442.22
## + rows_with_1      1    1.1967 3.5332 -381.10
## + cols_with_2      1    1.1722 3.5577 -380.33
## + maxrow           1    0.6155 4.1145 -364.05
## + rows_with_3p     1    0.5598 4.1701 -362.54
## + eyes             1    0.4649 4.2651 -360.02
## + hollowness       1    0.4386 4.2913 -359.33
## + cols_with_1      1    0.3993 4.3306 -358.31
## + connected_areas  1    0.3357 4.3943 -356.68
## + rows_with_2      1    0.0842 4.6457 -350.45
## <none>                         4.7300 -350.43
## + custom           1    0.0483 4.6816 -349.58
## + maxcol           1    0.0106 4.7193 -348.68
##
## Step:  AIC=-635.68
## aspect_ratio ~ height + width
##
##                   Df Sum of Sq     RSS     AIC
## + cols_with_2      1  0.058132 0.30580 -653.17
## + cols_with_1      1  0.030857 0.33307 -643.60
## + maxcol           1  0.018460 0.34547 -639.51
## + rows_with_2      1  0.016813 0.34712 -638.98
## + cols_with_3p     1  0.016107 0.34782 -638.75
## + nr_pix           1  0.012669 0.35126 -637.65
## + maxrow           1  0.009762 0.35417 -636.73
## <none>                         0.36393 -635.68
## + eyes             1  0.006388 0.35754 -635.66
## + rows_with_1      1  0.005850 0.35808 -635.50
## + custom           1  0.004949 0.35898 -635.21
## + connected_areas  1  0.003646 0.36029 -634.81
## + hollowness       1  0.003495 0.36044 -634.76
## + rows_with_3p     1  0.000928 0.36300 -633.97
##
## Step:  AIC=-653.17
## aspect_ratio ~ height + width + cols_with_2
##
##                   Df Sum of Sq     RSS     AIC
## + rows_with_3p     1  0.032337 0.27346 -663.69
## + eyes             1  0.023611 0.28219 -660.17
## + rows_with_2      1  0.023462 0.28234 -660.11
## + nr_pix           1  0.017360 0.28844 -657.72
## + cols_with_1      1  0.015558 0.29024 -657.02
## + cols_with_3p     1  0.015558 0.29024 -657.02
## + hollowness       1  0.006790 0.29901 -653.69
## <none>                         0.30580 -653.17
## + connected_areas  1  0.005298 0.30050 -653.13
## + rows_with_1      1  0.004688 0.30111 -652.90
## + maxcol           1  0.001524 0.30428 -651.73
## + maxrow           1  0.000753 0.30505 -651.45
## + custom           1  0.000283 0.30552 -651.28
##
## Step:  AIC=-663.69
## aspect_ratio ~ height + width + cols_with_2 + rows_with_3p
##
##                   Df Sum of Sq     RSS     AIC
## + maxrow           1 0.0090998 0.26436 -665.48
## + eyes             1 0.0084215 0.26504 -665.19
## <none>                            0.27346 -663.69
## + cols_with_1      1 0.0045525 0.26891 -663.57
## + cols_with_3p     1 0.0045525 0.26891 -663.57
## + nr_pix           1 0.0042057 0.26926 -663.43
## + hollowness       1 0.0033975 0.27007 -663.09
## + rows_with_1      1 0.0023241 0.27114 -662.65
## + rows_with_2      1 0.0023241 0.27114 -662.65
## + custom           1 0.0014709 0.27199 -662.29
## + maxcol           1 0.0013539 0.27211 -662.25
## + connected_areas  1 0.0000042 0.27346 -661.69
##
## Step:  AIC=-665.48
## aspect_ratio ~ height + width + cols_with_2 + rows_with_3p +
##     maxrow
##
##                   Df Sum of Sq     RSS     AIC
## + eyes             1 0.0047330 0.25963 -665.50
## <none>                          0.26436 -665.48
## + connected_areas  1 0.0045556 0.25981 -665.43
## + cols_with_1      1 0.0032712 0.26109 -664.87
## + cols_with_3p     1 0.0032712 0.26109 -664.87
## + hollowness       1 0.0028773 0.26149 -664.71
## + maxcol           1 0.0027583 0.26161 -664.65
## + rows_with_1      1 0.0011525 0.26321 -663.97
## + rows_with_2      1 0.0011525 0.26321 -663.97
## + nr_pix           1 0.0000632 0.26430 -663.51
## + custom           1 0.0000298 0.26433 -663.49
##
## Step:  AIC=-665.5
## aspect_ratio ~ height + width + cols_with_2 + rows_with_3p +
##     maxrow + eyes
##
##                   Df Sum of Sq     RSS     AIC
## <none>                          0.25963 -665.50
## + nr_pix           1 0.0036155 0.25601 -665.07
## + connected_areas  1 0.0021418 0.25749 -664.43
## + hollowness       1 0.0014435 0.25819 -664.13
## + rows_with_1      1 0.0013032 0.25833 -664.07
## + rows_with_2      1 0.0013032 0.25833 -664.07
## + maxcol           1 0.0008431 0.25879 -663.87
## + custom           1 0.0003186 0.25931 -663.64
## + cols_with_1      1 0.0002912 0.25934 -663.63
## + cols_with_3p     1 0.0002912 0.25934 -663.63
```

As seen above, forward selection gives us the following formula: aspect_ratio ~ height + width + cols_with_2 + rows_with_3p + maxrow + eyes

Let us now check which formula yields the better adjusted R squared value:

```
summary(lm(backwards_selection$call$formula, data = csv_df))
```

```
##
## Call:
## lm(formula = backwards_selection$call$formula, data = csv_df)
##
## Residuals:
```

```
##       Min        1Q    Median        3Q       Max
## -0.088764 -0.036613 -0.005221  0.019717  0.167982
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.3605467  0.0508769  26.742   <2e-16 ***
## rows_with_1  -0.0477733  0.0014695 -32.509   <2e-16 ***
## cols_with_1   0.0429207  0.0022380  19.178   <2e-16 ***
## rows_with_2  -0.0472058  0.0012775 -36.952   <2e-16 ***
## cols_with_2   0.0325875  0.0019763  16.489   <2e-16 ***
## rows_with_3p -0.0514328  0.0014626 -35.166   <2e-16 ***
## cols_with_3p  0.0409288  0.0012724  32.168   <2e-16 ***
## maxrow       -0.0016462  0.0009668  -1.703   0.0916 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05007 on 104 degrees of freedom
## Multiple R-squared:  0.9757, Adjusted R-squared:  0.974
## F-statistic: 595.3 on 7 and 104 DF,  p-value: < 2.2e-16
```

```
summary(lm(forwards_selection$call$formula, data = csv_df))
```

```
##
## Call:
## lm(formula = forwards_selection$call$formula, data = csv_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.082267 -0.036179 -0.002148  0.021110  0.184091
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.347674   0.047942  28.110  < 2e-16 ***
## height       -0.047421   0.001073 -44.204  < 2e-16 ***
## width         0.041498   0.001196  34.700  < 2e-16 ***
## cols_with_2  -0.008950   0.001533  -5.839 5.92e-08 ***
## rows_with_3p -0.003642   0.001221  -2.982  0.00356 **
## maxrow       -0.001453   0.000982  -1.479  0.14205
## eyes         -0.012110   0.008753  -1.384  0.16944
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04973 on 105 degrees of freedom
## Multiple R-squared:  0.9758, Adjusted R-squared:  0.9744
## F-statistic: 704.2 on 6 and 105 DF,  p-value: < 2.2e-16
```

As shown above, the forward selection method yielded a better adjusted R squared value and requires fewer variables, making it the clear winner for predicting aspect ratio. Now, let's take a closer look at the variables it selected and why they are useful in predicting aspect ratio.

Height and width feel like an obvious choice if we were to intuitively pick variables that would best calculate aspect ratio, so it is reassuring to see them here. We also know that height and width have a surprisingly low correlation value of 0.129, indicating that we are keeping the prediction as parsimonious as possible. Additionally, height and aspect_ratio have the highest correlation out of all the variables considered in the previous section, making height particularly useful for predicting aspect ratio.

I think cols_with_2, rows_with_3p and maxrow may not seem like obvious choices for predicting aspect ratio at first, however, they all provide unique information about the image, making our model very parsimonious. This is an invaluable aspect of our model and hence why these variables may have been included.

The final feature eyes is also a useful variable for predicting aspect ratio since it provides information that is not covered by any of our previous features. If we had included hollowness, for example, we may not expect to see eyes in our final model since they could be conveying similar information.

Now, let's evaluate how well our model performs at predicting the aspect ratio for 15 images in our dataset.

```
new_aspects <- data.frame(aspect = vector("integer", 15))
predicted <- predict(forwards_selection, newdata = new_aspects)[0:15]

actual <- csv_df$aspect_ratio[0:15]
results <- data.frame(actual, predicted)

kable(results, caption = "A table showing the actual vs predicted aspect ratios:")
```

Table 3: A table showing the actual vs predicted aspect ratios:

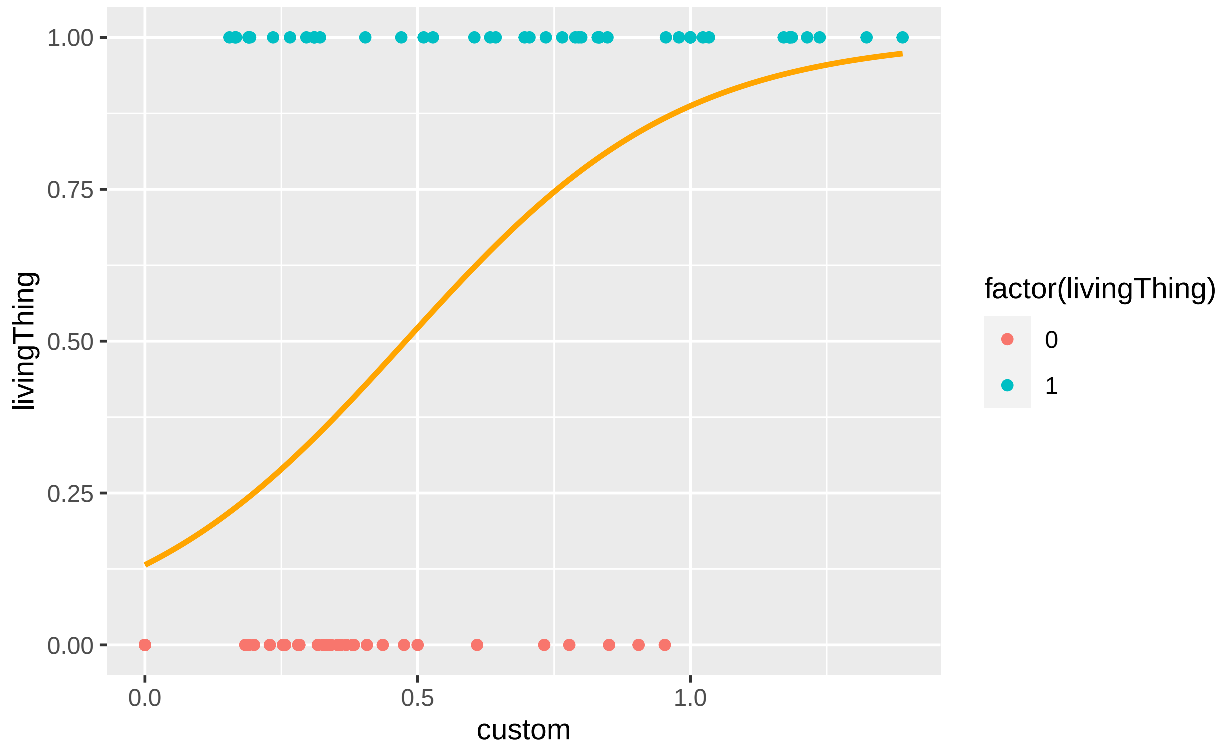
actual	predicted
0.759	0.7440721
0.793	0.7850453
0.875	0.7949047
0.963	0.9833637
0.957	1.0078530
1.040	1.0279581
1.190	1.1701018
1.136	1.1332954
0.964	0.9438299
1.000	0.9876089
1.065	1.0557598
1.071	1.1084026
0.828	0.8057377
0.893	0.8891369
1.000	0.9769392

The model is quite accurate at predicting the aspect ratios and appears to be a good choice.

Section 4.2

In this section we use the most useful discriminatory feature for distinguishing between living thing objects and non-living thing objects based off of the results and visualisations from section 3.4 above. We then fit and visualise a logistic regression model.

```
# Make a livingThing variable that discriminates
# livingThings from the nonLivingThings:
csv_df$livingThing <- 0
csv_df$livingThing[csv_df$labels %in% c("lemon", "cherry", "tree",
    "banana")] <- 1

set.seed(2062)


# make training and test datasets
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# randomly shuffle rows:
csv_shuffled <- csv_df[sample(nrow(csv_df)), ]

# first 80% will be training data:
training_data = csv_shuffled[1:90, ]
test_data = csv_shuffled[91:112, ]


# Logistic regression with our custom feature of curvature
# ========================================================

glmfit <- glm(livingThing ~ custom, data = training_data, family = "binomial")

x.range = range(training_data[["custom"]])

x.values = seq(x.range[1], x.range[2], length.out = 1000)

fitted.curve <- data.frame(custom = x.values)
fitted.curve[["livingThing"]] = predict(glmfit, fitted.curve,
    type = "response")

# Plot the training data and the fitted curve:
plt <- ggplot(training_data, aes(x = custom, y = livingThing)) +
    geom_point(aes(colour = factor(livingThing)), show.legend = T) +
    geom_line(data = fitted.curve, colour = "orange", linewidth = 1)

plt
```



```
summary(glmfit)
```

```
##
## Call:
## glm(formula = livingThing ~ custom, family = "binomial", data = training_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0096  -0.8719  -0.1494   0.7392   1.7443
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.8871     0.4604  -4.099 4.15e-05 ***
## custom        3.9496     0.8635   4.574 4.79e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 124.766  on 89  degrees of freedom
## Residual deviance:  92.687  on 88  degrees of freedom
## AIC: 96.687
##
## Number of Fisher Scoring iterations: 4
```

In general a logistic regression model can be described by: $P(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$. Looking at the above summary and graph we have $\beta_0 = -1.8871, \beta_1 = 3.9496$ giving us:

$$P(x) = \frac{e^{-1.8871+3.9496x}}{1+e^{-1.8871+3.9496x}}$$

Where P(x) is the probability that an image is a living thing, given x (curvature). As seen above all our y-values are bounded between 0 and 1. In general we will take a probability value > 0.5 to be a living thing and ≤ 0.5 to be non-living.

Upon examining the graph of our custom feature, we observe that it takes an S-shaped curve, which tells us it appears to be a good choice for prediction. The rationale behind choosing this feature is as follows. Initially, it might seem that rows_with_1 is the optimal feature as it yields the smallest p-value according to the results in section 3.4. However, after further investigation, we can discover that this result is primarily driven by wineglasses and to some extent, golf clubs, while most other images have zero rows_with_1. Consequently, if our model receives an image with a low rows_with_1 value, it will immediately classify it as a living item, even though it is equally likely to be an envelope or pencil. This outcome is obviously not desirable. Similar observations can be made for most other features with very significant p-values.

On the other hand, the custom curvature feature appears a better choice since it is a more generalized result. We know that nearly all non-living things have a lower curvature value than we would expect from living things. This is accurate to what the graph tells us since if we consider a high curvature, say 1.5, and plug it into our above equation it returns 0.983 (i.e high curvature values almost certainly imply living thing). Altogether, this is the ideal situation for a logistic regression prediction model. Additionally, this feature yields an exceedingly low p-value (the second lowest), as seen in section 3.4, furthering our point that there is a significant difference in the mean curvature of living things and non living things.

Let us check the accuracy of this feature on both the training and test data:

```
# Assuming a p>0.5 cut-off, calculate accuracy on the
# training data
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```r
training_data[["predicted_val"]] = predict(glmfit, training_data,
    type = "response")
training_data[["predicted_class"]] = 0
training_data[["predicted_class"]][training_data[["predicted_val"]] >
    0.5] = 1



correct_items = training_data[["predicted_class"]] == training_data[["livingThing"]]

# proportion correct:
nrow(training_data[correct_items, ])/nrow(training_data)
```

```
## [1] 0.7666667
```

```r
# proportion incorrect:
nrow(training_data[!correct_items, ])/nrow(training_data)
```

```
## [1] 0.2333333
```

```r
# Assuming a p>0.5 cut-off, calculate accuracy on the test
# data
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

test_data[["predicted_val"]] = predict(glmfit, test_data, type = "response")
test_data[["predicted_class"]] = 0
test_data[["predicted_class"]][test_data[["predicted_val"]] >
    0.5] = 1

correct_items = test_data[["predicted_class"]] == test_data[["livingThing"]]

# proportion correct:
nrow(test_data[correct_items, ])/nrow(test_data)
```

```
## [1] 0.7272727
```

```r
# proportion incorrect:
nrow(test_data[!correct_items, ])/nrow(test_data)
```

```
## [1] 0.2727273
```

It appears that the logistic regression model performs quite well in determining whether an image depicts a living or non-living object, with an overall accuracy of approximately 74%. Given our current set of features, it seems unlikely that we can achieve significantly better predictive accuracy, as no individual feature appears to provide consistently high accuracy on its own. For example if all living things had eyes while no non-living things did, it would be the intuitively obvious and highly accurate feature to choose. However, there is no such feature in our current dataset.