Code

Assignment 2: Feature Engineering, Statistical Analysis and Machine Learning

Pranay Meshram (40280938)

Introduction

This project is about creating a dataset of handwritten letters and symbols including faces to perform different statistical functions on the dataset. We perform feature engineering to allow identifying the handwritten symbols automatically, performing statiscal analysis and implementing machine learning models.

Section 1

I created the images handwritten by drawing them on GIMP and then read the images as text files in R code, changed the pixel values, converted them to matrices and saved as CSV files. I had to loop through all the individual images and remove column and row names individually to ensure they were not added accidentally into the CSV.

Section 2

For this section, I had multiple times ran into the difficulty of my code being unable to read the CSV files. I always tried to set the working directory by RStudio but as this issue occurred again and again, I added the following code so that whenever I ran my file, I wouldn't encounter the issue again.

Hide

```
setwd("./")
```

As needed, I had to loop through all the CSV files to individually calculate the features.

Label feature

For the label feature, the 'grepl' function was used to match the symbol with file name.

Hide

```
if (grepl("40280938_a", fileName, fixed = TRUE)) {
   label <- "a"
 }
```

I had prior used different substring functions to match the filename but it didn't work so I landed on using the 'grepl' function.

Index feature

The same function as label, 'grepl' function was used here.

Hide

```
if (grepl("_01", fileName, fixed = TRUE)) {
    index <- "01"
  }
```

The issue I faced here was when saving index as an integer value, '01' was being saved as '1' which was not the aim. To tackle this issue, I saved the index value as string to be converted into integer value later or used directly as string value, based on the needs.

For subsequent features, I initially used the following code to calculate number of rows/cols with x number of black pixels

Hide

```
nrow(imageMatrix[imageMatrix < 2])
```

(In the above case, it's for calculating number of rows with exactly 1 black pixel)

However after troubleshooting, I found out that was a mistake on coding that I did.

I then used rowSums/colSums as part of 'zoo' library on a logical matrix('imageMatrix == 1') and then created a logical vector '==1'and get the count with 'sum'. I have inlined one of the functions for the code part of it:

Hide

```
  rows_with_1 <- sum(rowSums(imageMatrix == 1) == 1)
  rows_with_1
```

Aspect ratio feature

For the aspect ratio feature, I began with finding the center pixel row and column. I rounded off the value to find the center row and column as of course, the values could be decimal. This could however be a source of error as if we're having odd number of columns or rows, the code will only pick the rounded off value which might not be the center value for some analysts.

With the matrix function, I found the values of the center pixel. For the aspect_ratio function, I calculated the horizontal difference i.e. width instead of height as I found it easier to calculate.

I found the left most and the right most pixel on the same row of the center pixel using this code:

Hide

```
# Right most black pixel column from the center pixel
  right_most_pixel = max(which(imageMatrix == 1, arr.ind=TRUE)[8,])

  # Left most black pixel column from the center pixel
  left_most_pixel = min(which(imageMatrix == 1, arr.ind=TRUE)[8,])
```

I deducted both of them to find the width, and that was the aspect_ratio value.

For subsequent features of finding neighbouring pixels, I looped through each of the values of the matrix.

Before doing that, I ensured I padded the matrix to 20x20 using ‘cbind’ and ‘rbind’ to ensure the extreme rows and columns doesn’t cause error.

As I looped through the matrice, if a black pixel was found, I checked the neighbouring pixels to check for their values. Based on the need of the feature, if any neighbouring pixel was found, it would be summed with all the neighbouring pixels to see how many black pixels are in the neighbour. Depending on the feature, they would be checked if we just need one black pixel in all the neighbour or no black pixel, and the counter was incremented.

I have added a snippet of the code for counting number of black pixels in the neighbour (neigh_1 feature):

Hide

```
# Feature 7
 # neigh_1 - Using counter to count the number of black pixels

 counter = 0

 # Here a new matrx was created out of the original one with paddings added to it to make it

 imageNewMatrix <- imageMatrix

 nrow(imageMatrix)
 ncol(imageMatrix)

 imageNewMatrixa <- cbind(imageNewMatrix, 0)
 imageNewMatrixB <- rbind(imageNewMatrixa, 0)
 imageNewMatrixC <- cbind(imageNewMatrixB, 0)
 imageNewMatrixD <- rbind(imageNewMatrixC, 0)
 nrow(imageNewMatrixD)
 ncol(imageNewMatrixD)

 for (row in 2:19) {
   for (col in 2:19) {
     if (imageNewMatrixD[row,col] == 1) {

       # Get entry of the left pixel

       pixel_to_left = as.numeric(imageNewMatrixD[row, col-1])

       # Get entry of the right pixel

       pixel_to_right = as.numeric(imageNewMatrixD[row, col+1])

       # Get entry of the pixel at top

       pixel_at_top = as.numeric(imageNewMatrixD[row-1, col])

       # Get entry of the pixel at top-left

       pixel_at_top_left = as.numeric(imageNewMatrixD[row-1, col-1])

       # Get entry of the pixel at top right

       pixel_at_top_right = as.numeric(imageNewMatrixD[row-1, col+1])

       # Get entry of the pixel at bottom

       pixel_at_bottom = as.numeric(imageNewMatrixD[row+1, col])

       # Get entry of the pixel at bottom left

       pixel_at_bottom_left = as.numeric(imageNewMatrixD[row+1, col-1])
```

```
      # Get entry of the pixel at bottom right

      pixel_at_bottom_right = as.numeric(imageNewMatrixD[row+1, col+1])


      pixel_sum = pixel_to_left + pixel_to_right + pixel_at_top +
        pixel_at_top_left + pixel_at_top_right + pixel_at_bottom +
        pixel_at_bottom_left + pixel_at_bottom_right

      # Using isTrue as one of the pixels is empty and isTrue converts logical(0) to FALSE

      if (isTRUE(pixel_sum == 1)) {
        counter = counter + 1
      }



    }
  }

}

neigh_1 = counter
```

connected_areas and eyes feature

I unfortunately couldn't calculate the two features and used a random number between 1 to 100 to be used for this feature.

custom feature

My custom feature was calculating number of rows with no black pixels. I used this feature as I think it is crucial to see how much empty space we have in a image.

In future, let's say we have images of greater size, we can simply remove the rows with blank data i.e. rows with no black pixels from the very top and very bottom to save space. This is very ideal when we have millions of images as datasets and we have to save space for computation.

I used the same rowSums function to calculate this feature as:

Hide

```
custom <- sum(rowSums(imageMatrix == 1) == 0)
custom
```

Adding all the features to CSV file

To add all the features to CSV file, I first added all the features to a vector.

Then I created a CSV file using 'write.table' and added all the features to the CSV file. Since the code was in a loop for each image, I had to turn the 'append' to TRUE so the previous image values would still be saved in the CSV file.

Following is the code for the above explanation:

Hide

```
feature <- c(label,index, nr_pix, rows_with_1, cols_with_1,
               rows_with_3p, cols_with_3p, aspect_ratio,
               neigh_1, no_neigh_above, no_neigh_below,
               no_neigh_left, no_neigh_right, no_neigh_horiz,
               no_neigh_vert, connected_areas, eyes, custom)

  feature



  write.table(matrix(feature, nrow=1), file = "./40280938_features.csv",
              append = TRUE, sep = ',',
              row.names = FALSE,
              col.names = FALSE)
```

Now here is a catch. If I tried adding the column names in the ‘write.table’ code above, it would add column name to every image matrix. That would mean instead of supposedly rows of 141 rows including column name, it would be 248 rows with column name added for each value.

I had that happen to me and I tried to fix this issue. To fix this issue, I set row and column names as ‘FALSE’ in the above code to only modify them later.

After the loop, I read from the same CSV file, converted the data to a dataframe and this time with column names. Now I sorted the data alphabetically using label only, as it was already sorted by index.

Then I used write.table to overrite the previous CSV storing features with append as ‘FALSE’. And this time I had the features in the CSV with column names and just 141 rows! Below is the code for that:

Hide

```
# Reading from the CSV file with all the data present and then overwriting it to add the colu
mn names to it

final_data <- read.csv("./40280938_features.csv",
                       header = FALSE,
                       col.names = c('label','index', 'nr_pix', 'rows_with_1', 'cols_with_1',
                                     'rows_with_3p', 'cols_with_3p', 'aspect_ratio',
                                     'neigh_1', 'no_neigh_above', 'no_neigh_below',
                                     'no_neigh_left', 'no_neigh_right', 'no_neigh_horiz',
                                     'no_neigh_vert', 'connected_areas', 'eyes', 'custom'))

# Sorting data alphabetically and then by index

sorted_data_by_label <- final_data[order(final_data$label),]


write.table(sorted_data_by_label, file = "./40280938_features.csv",
            append = FALSE, sep = ',',
            row.names = FALSE)
```

Section 3

Initially as the first step, I set the working directory. I installed the following libraries and included the reasoning for the libraries as follows:

Hide

```
library(moments)
```

- for skewness() function

Hide

```
library(robustbase)
```

- for colMedians() function

Hide

```
library(GMCM)
```

- for colSds() function

Hide

```
library(data.table)
```

- for histogram of feature values of the groups

Hide

```
library(gginference)
```

- for plotting t.test values in the graph

Hide

```
library(ggstatsplot)
```

- this library is needed to plot the correlations (linear association)

Hide

```
library(ggcorrplot)
```

- this library is needed for the above library, ggstatsplot to work

I read the data from the features CSV file.

Section 3.1

I preferred to use:

Hide

```
hist()
```

The above function create histogram as it created the bin by itself, allowing for easier readability for the data.

I also had to change all the data to numeric form to be easily plotted in the histogram.

For instance, my code would be like:

Hide

```
nr_pix <- as.numeric(data$nr_pix)
nr_pix
```

```
  [1] 36 40 48 48 47 39 46 45 31 33 29 33 30 25 31 29 20 24 17 16 20 23 21 23 39 38 33 37 33
29 36 35 25
 [34] 26 29 21 23 27 28 24 35 31 23 28 24 27 28 30 35 39 39 38 46 45 41 40 26 30 25 27 32 29
27 28  9 10
 [67] 10 14 22 16 12 22 19 23 17 17 16 20 17 15 25 22 23 24 23 25 27 27 26 21 24 21 25 31 22
31 26 27 21
[100] 26 29 21 23 29 22 27 30 28 23 25 26 23 31 25 22 24 33 25 25 30  9 11 13  8 13 11 11 16
19 25 20 18
[133] 14 18 15 11 22 10 10 16
```

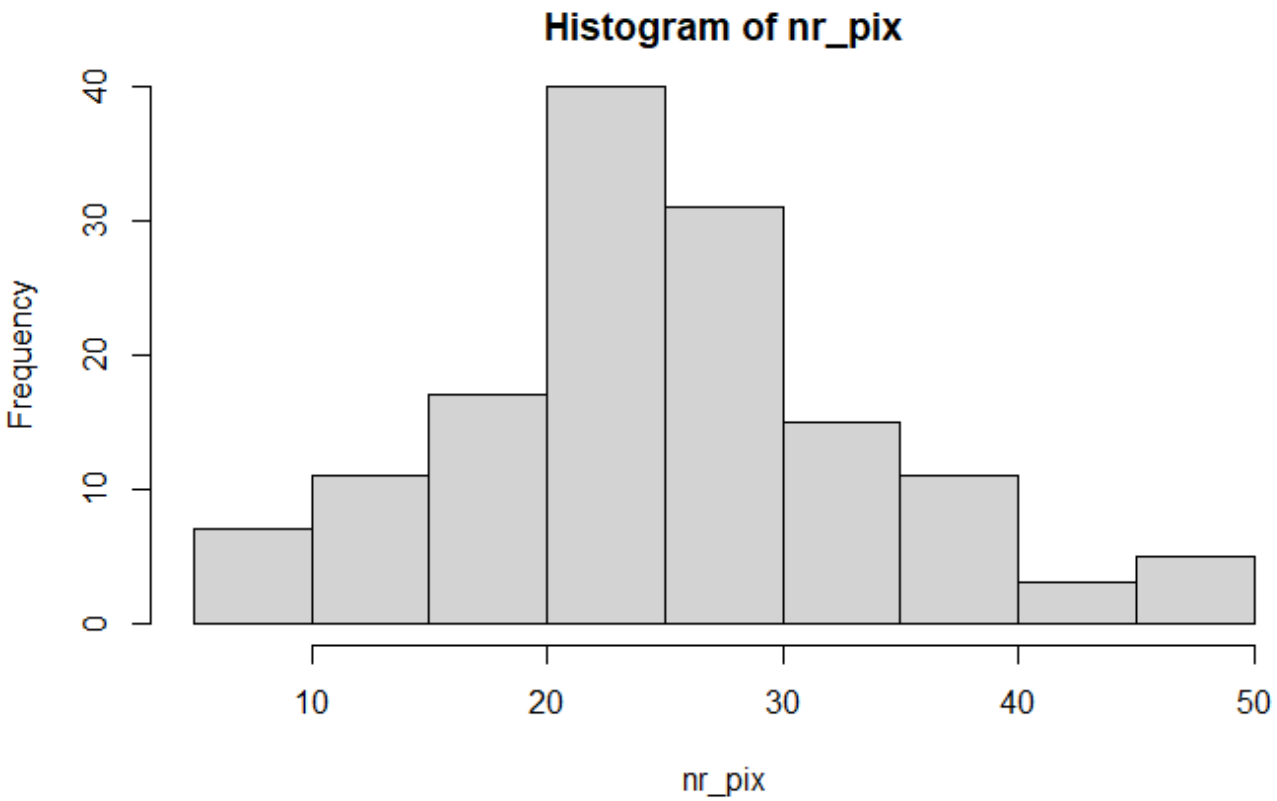
Hide

```
hist(nr_pix)
```

nr_pix (Number of pixels)

Hide

```
hist(nr_pix)
```



(Image may vary in preview in this document)

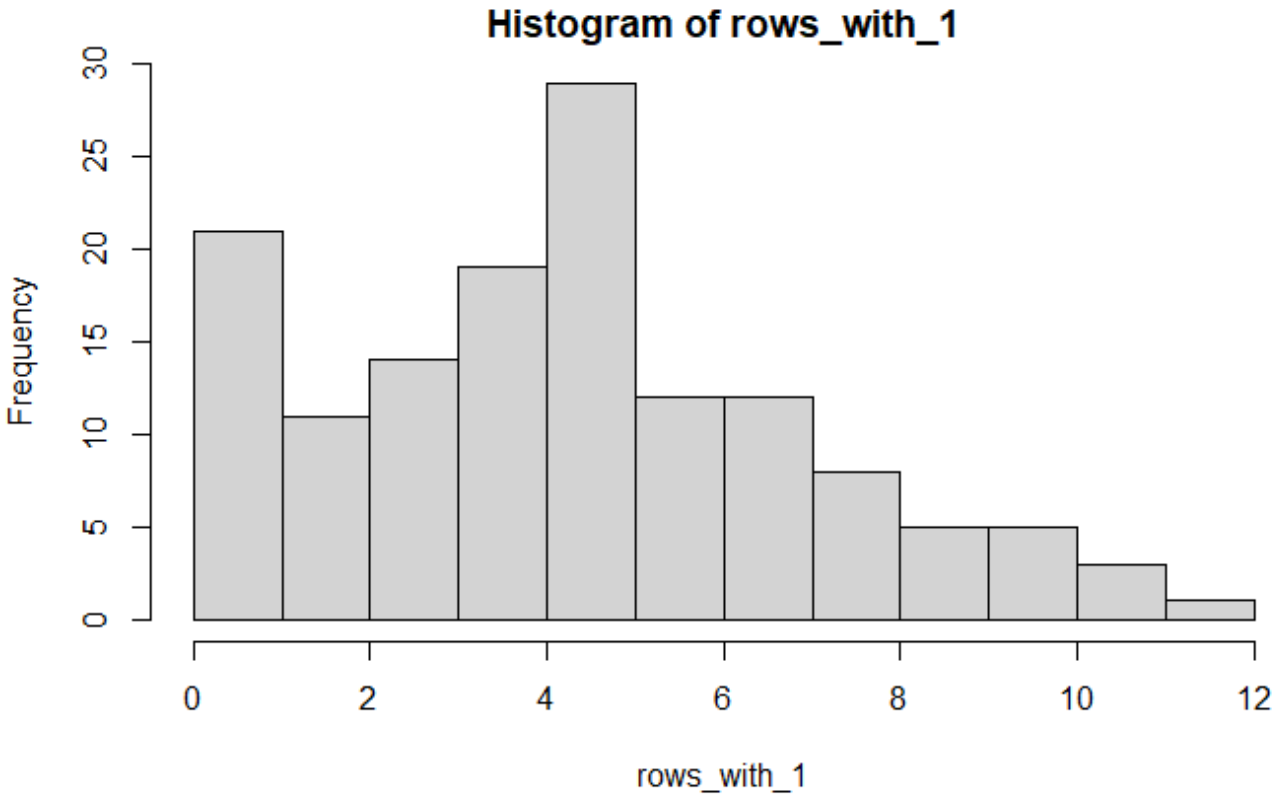
The number of pixels seems like a normal histogram with no skewndness. There are also no outliers in the data. The center of the data seems to be like from 20-25 where most number of black pixels are found. The data are spread out most from 20-30 number of pixels before dropping sharply after. This is a symmetric data. This also shows that for most images, number of black pixels is around twenty to twenty five.

This is a bi-modal histogram.

rows_with_1 (Number of rows with just 1 black pixel)

Hide

```
hist(rows_with_1)
```



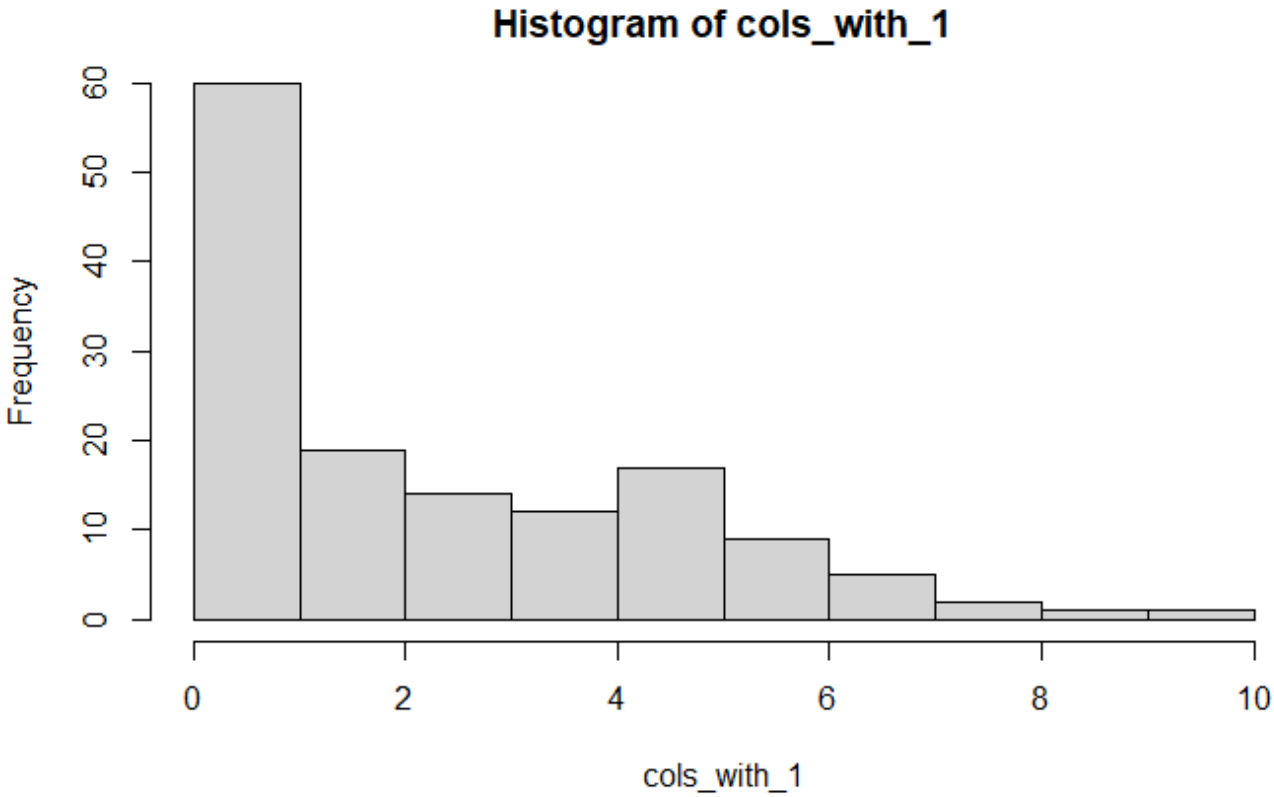
This seems like a right skewed (non-normal) histogram. There are no outliers in the data. In this data, the data seems evenly spread with differences in certain parts (0, 4-5 number of rows). The data seems short-tailed as well and we can see that the majority of data is spread around 0-10 number of rows with data before frequency of 2-5 number of rows the maximum. This data also shows that for most images, their number of rows with just one black pixel ranges from four to five.

This is a unimodal histogram.

cols_with_1 (Number of columns with just 1 black pixel)

Hide

```
hist(cols_with_1)
```



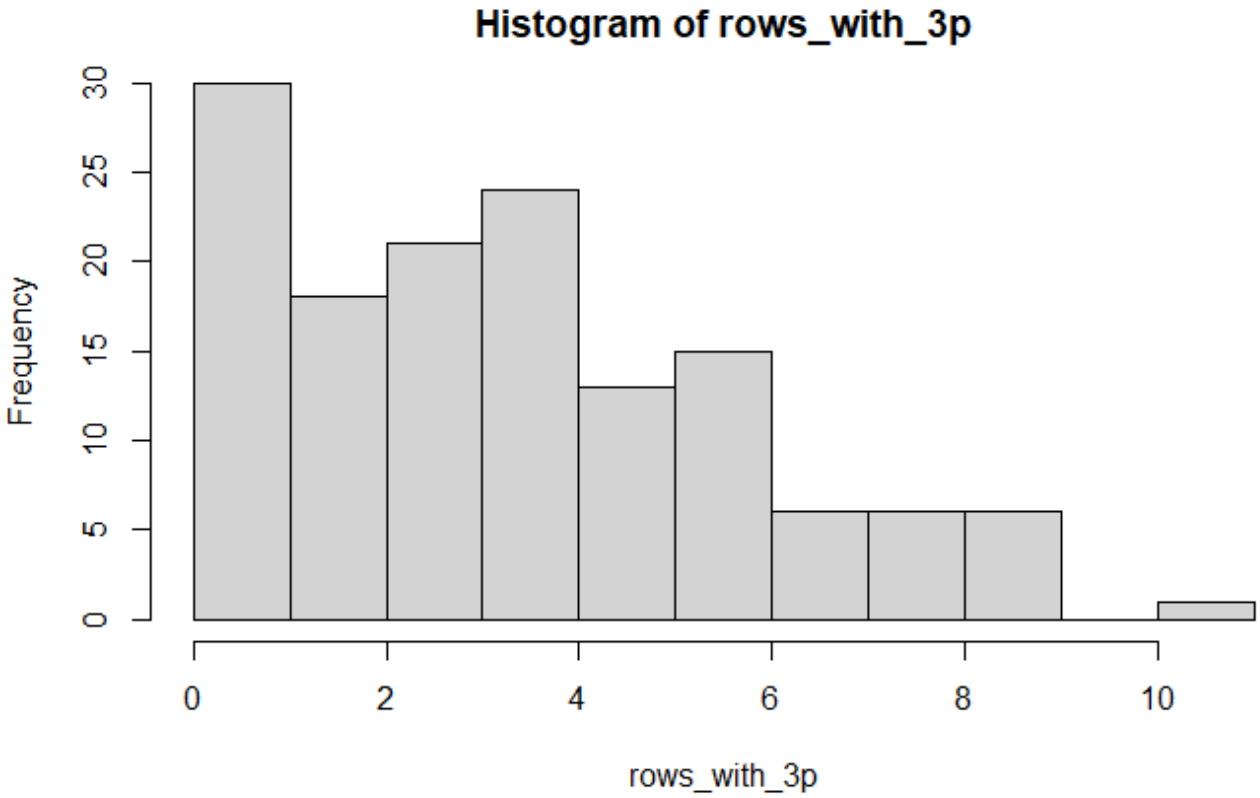
This is an extremely right-skewed data with a steep fall in frequency from 0-1 number of columns. There is no outlier in the data however this histogram seems long-tailed because the tails approach zero very slowly. The data only shows frequency falling down with rise in number of columns. This histogram shows that for most images, number of columns with just one black pixel is zero.

This is a unimodal histogram.

rows_with_3p (Number of rows with three black pixel)

Hide

```
hist(rows_with_3p)
```



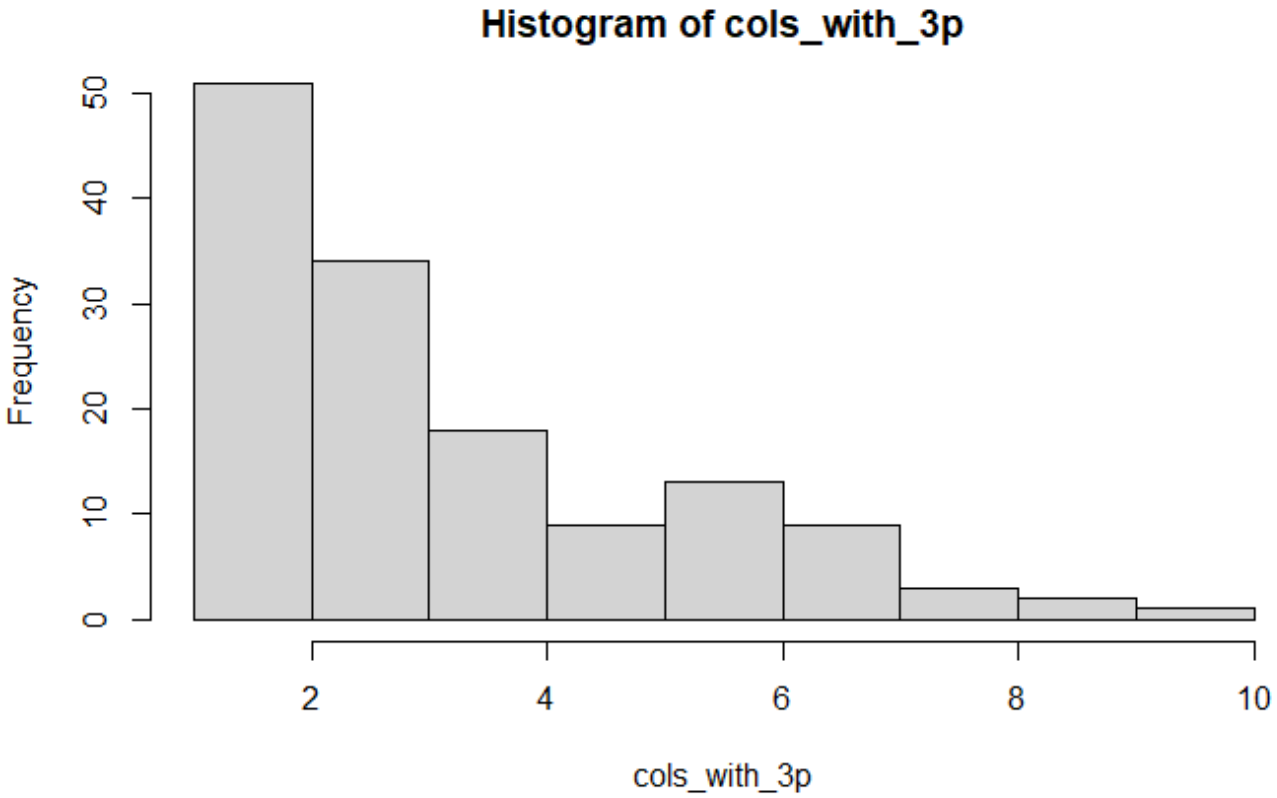
This is a right-skewed (non-normal) histogram as well. However we have an outlier here that is separate from the other values (10-after number of rows). The spread of the data is uneven but there's not a huge difference between the spread. Frequency of data from 6-9 number of rows is consistent and the exact same. The center of the number of rows is at 5 with less than average frequency. The highest frequency shows most rows have less or more than three black pixels.

This is a multimodal histogram.

cols_with_3p (number of columns with three black pixels)

Hide

```
hist(cols_with_3p)
```



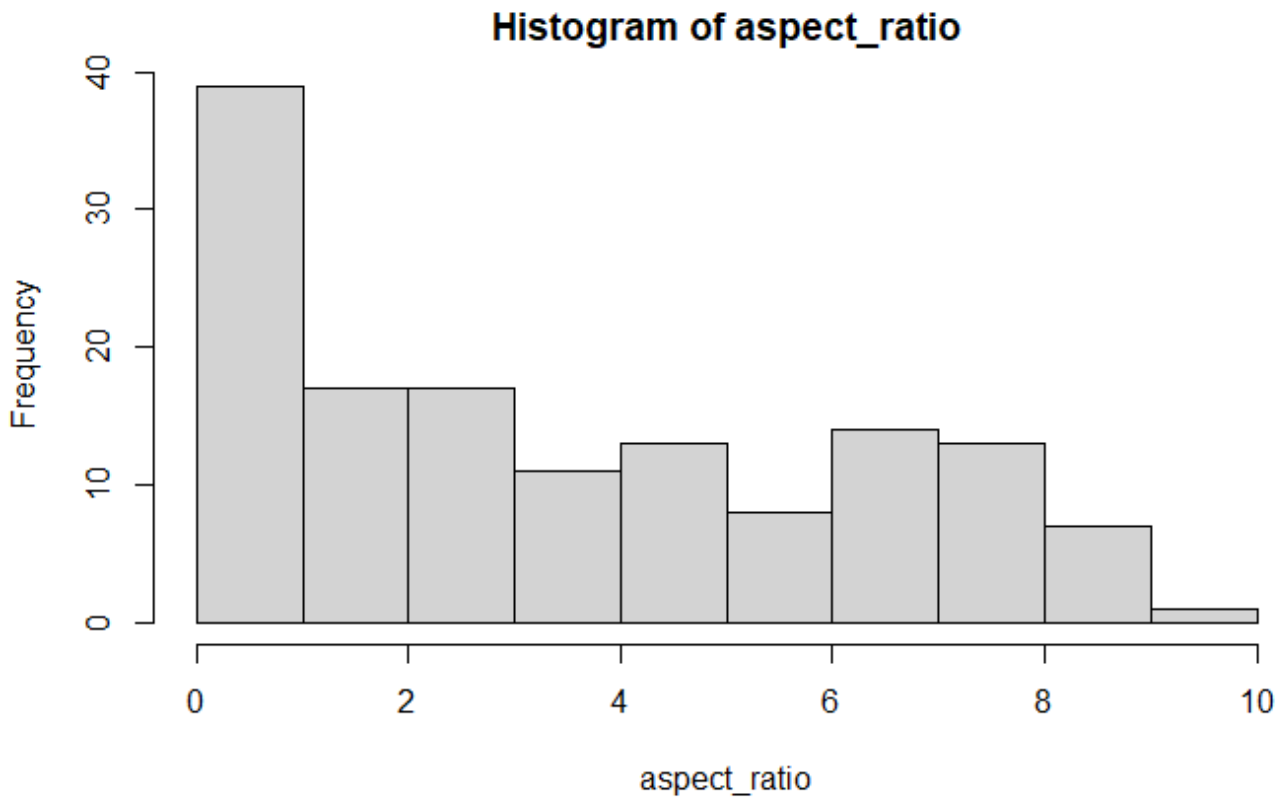
This data is extremely right-skewed with no outliers present. The data is unevenly spread with short-tail as the tail approaches zero very fast. The frequency of columns decline very fastl from the highest frequency. The center of the data seems to be around 5, which is way lower than the highest frequency at 0. The highest frequency shows most columns have less than or more than three black pixels.

This is a bimodal histogram.

aspect_ratio (aspect ratio)

Hide

```
hist(aspect_ratio)
```



This data is right-skewed as well (non-normal) with no outliers present. The data is almost evenly spread in decline from 1 to 8 as the value of aspect ratio with slight exceptions. The center of the data seems to be around 5 with the highest frequency being at 0.Thus it shows that for most images, aspect ratio is zero.

This is a unimodal histogram.

Section 3.2

For statistical analysis, we calculated mean, median and standard deviation for all set of features for both letters and non-letters.

Since we know that letters are from rows one to eighty (not eighty one as the first row is the column name) and non-letters are from row eighty one to one fourty:

Hide

```
only_letters <- data[1:80, ]

only_non_letters <- data[81:140, ]
```

Hide

```
# Finding Mean for full set of letters

mean_of_letters <- colMeans(only_letters[, 2:18])

mean_of_letters


# Finding mean for full set of non-letters

mean_of_non_letters <- colMeans(only_non_letters[, 2:18])

mean_of_non_letters
```

For **median** and **standard deviation**, we used 'data.matrix' as 'colMedians' function only takes vector or matrix as input and our data was dataframe.

Example of that:

Hide

```
median_of_non_letters <- colMedians(data.matrix(only_non_letters[, 2:18]))

median_of_non_letters
```

Then as part of statistical analysis, we calculated **skewness** of each of the feature of letters as well as non-letters.

However, we did not calculate skewness of **connected_areas** and **eyes** as those features had random values and we cannot gain any statistical interference with random values for the rest of the data.

Now out of all the features, we will select features which have the skewdness closes to zero. For features that are closes to zero, that means the data is symmetrical and these maybe be helpful for discriminating between letters and digits.

We calculated the following features having skewdness closest to zero:

Hide

```
only_letter_rows_with_3p_skew = skewness(only_letters$rows_with_3p)
only_letter_rows_with_3p_skew
```

```
[1] 0.06913699
```

Hide

```
only_non_letter_rows_with_3p_skew = skewness(only_non_letters$rows_with_3p)
only_non_letter_rows_with_3p_skew
```

```
[1] 0.009200535
```

Hide

```
only_letter_no_neigh_right_skew = skewness(only_letters$no_neigh_right)
only_letter_no_neigh_right_skew
```

```
[1] -0.06102782
```

Hide

```
only_non_letter_no_neigh_right_skew= skewness(only_non_letters$no_neigh_right)
only_non_letter_no_neigh_right_skew
```

```
[1] 0.1217553
```

Hide

```
only_letter_no_neigh_vert_skew = skewness(only_letters$no_neigh_vert)
only_letter_no_neigh_vert_skew
```

```
[1] -0.1442303
```

Hide

```
only_non_letter_no_neigh_vert_skew = skewness(only_non_letters$no_neigh_vert)
only_non_letter_no_neigh_vert_skew
```

```
[1] -0.5104553
```

Hide

```
only_letter_no_neigh_left_skew = skewness(only_letters$no_neigh_left)
only_letter_no_neigh_left_skew
```

```
[1] 0.306013
```

Hide

```
only_non_letter_no_neigh_left_skew = skewness(only_non_letters$no_neigh_left)
only_non_letter_no_neigh_left_skew
```

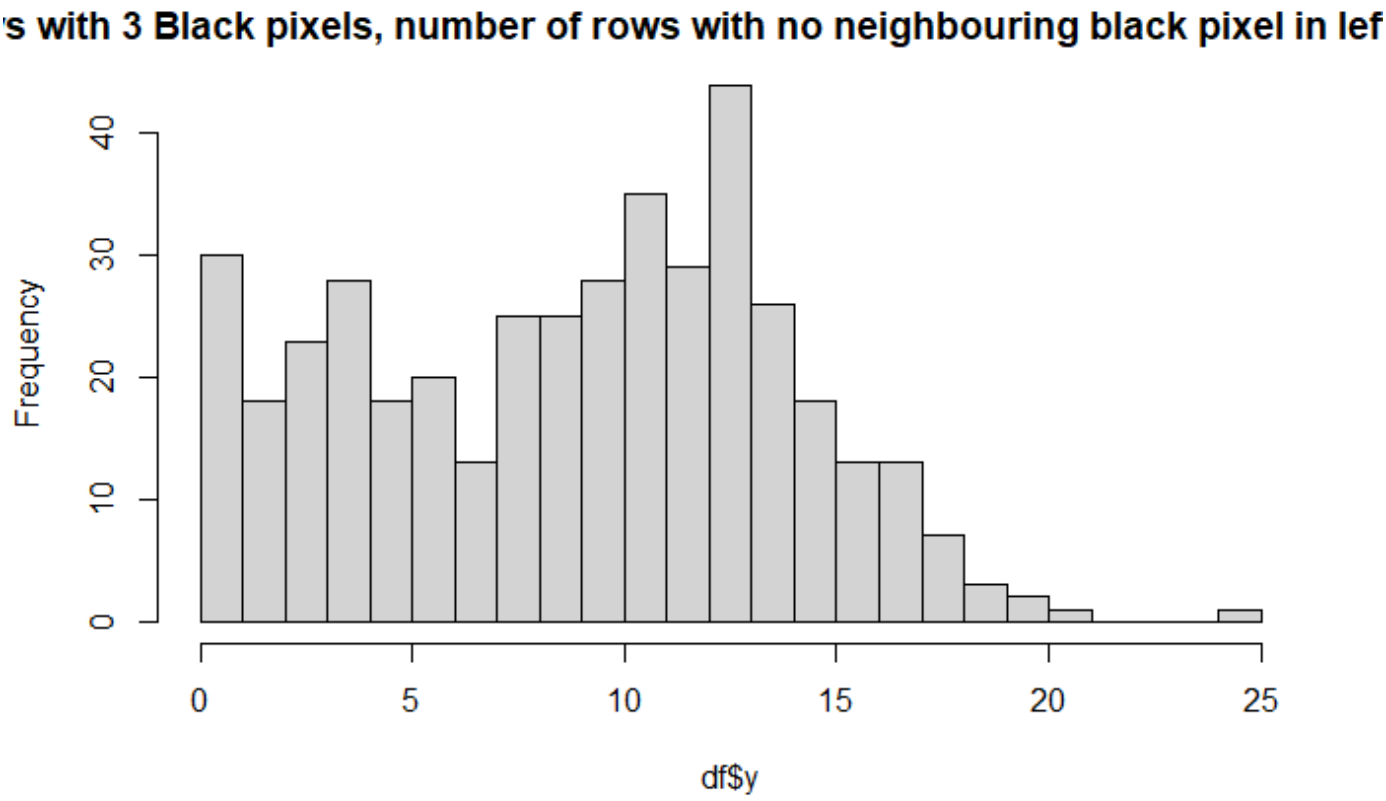
```
[1] -0.02009708
```

Based on the skewdness of all other features, I suggest **no_neigh_left, rows_with_3p, no_neigh_right, no_neigh_vert** features will be most helpful for discriminating between letters and non-letters.

Since out of the above four features, **row_with_3p, no_neigh_right** and **no_neigh_left** have the skewdness, closest to 0, we'll draw visualisation for them.

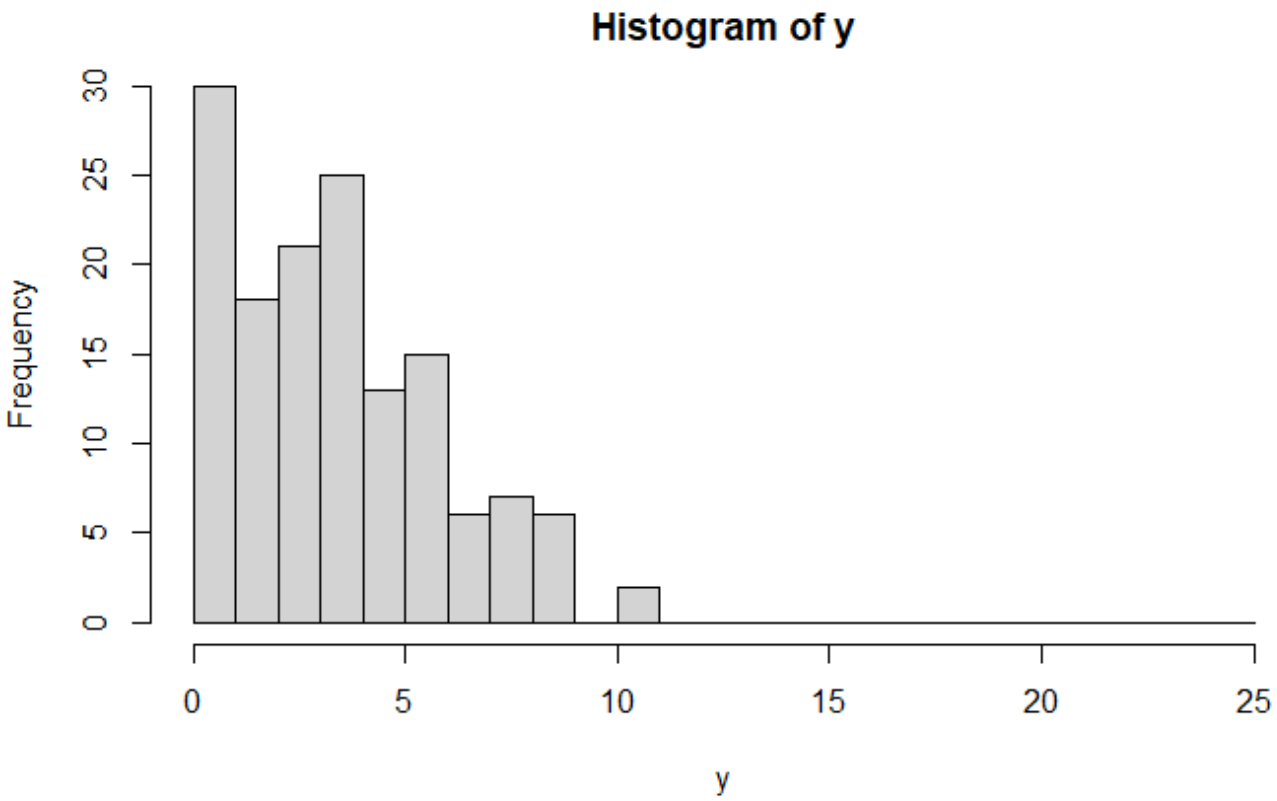
Hide

```
#
# histogram  of  feature
# values for the groups

df = data.frame(y = c(data$rows_with_3p, data$no_neigh_left, data$no_neigh_right),
                x = rep(c(0,100), each = 210))
#full hist
fullhist = hist(df$y, breaks = 20, main = "Histogram of feature values of groups - Number of
rows with 3 Black pixels, number of rows with no neighbouring black pixel in left, number of
rows with no neighbouring black pixel in right") #specify more breaks than probably necessary
```



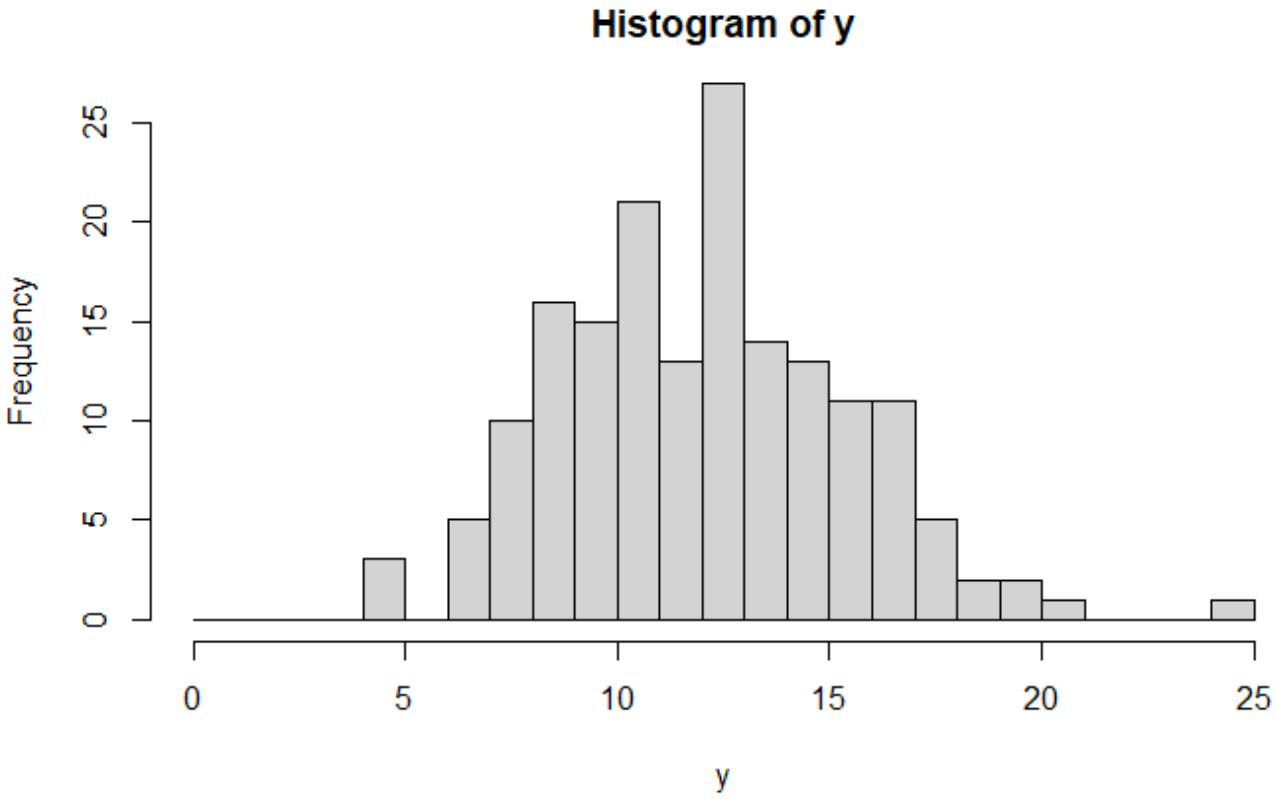
Hide

```
#create histograms for the different features using breaks from full histogram
zerohist = with(subset(df, y == data$rows_with_3p), hist(y, breaks = fullhist$breaks))
```



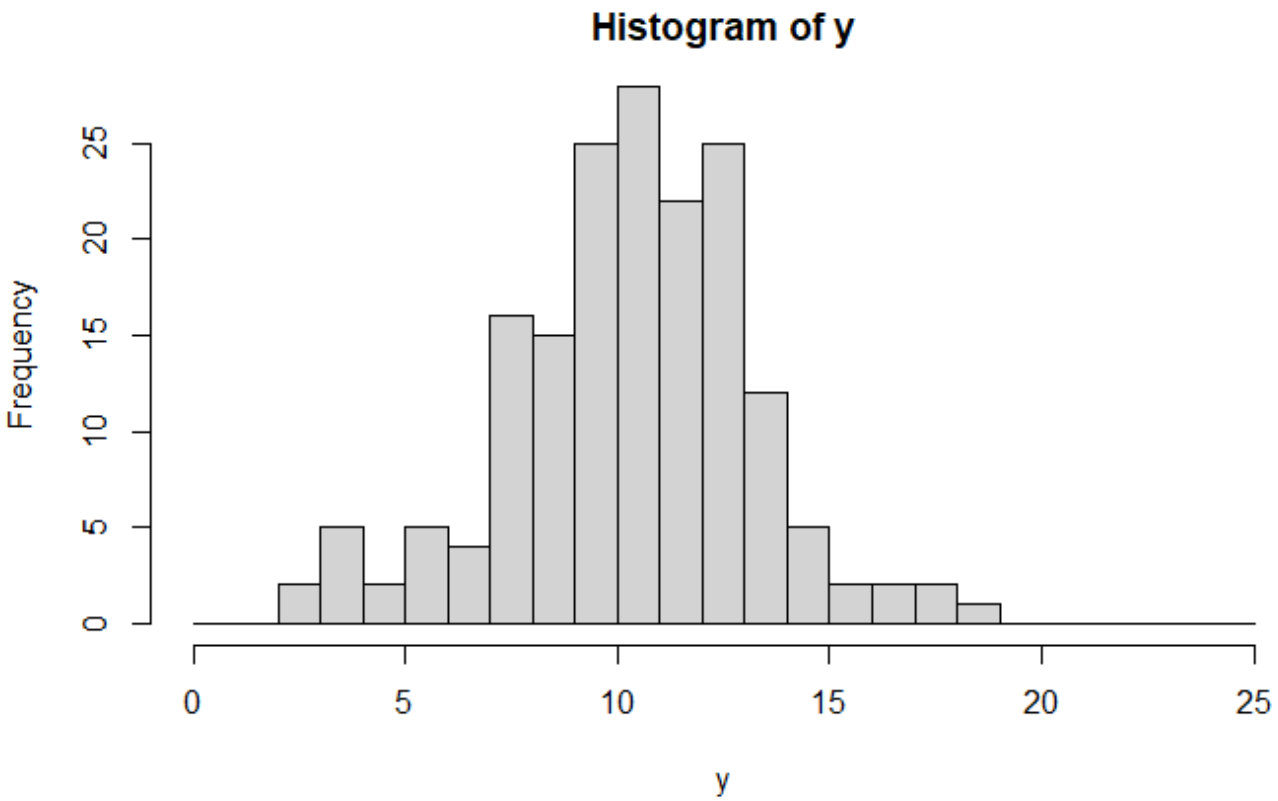
Hide

```
oneshist = with(subset(df, y == data$no_neigh_left), hist(y, breaks = fullhist$breaks))
```



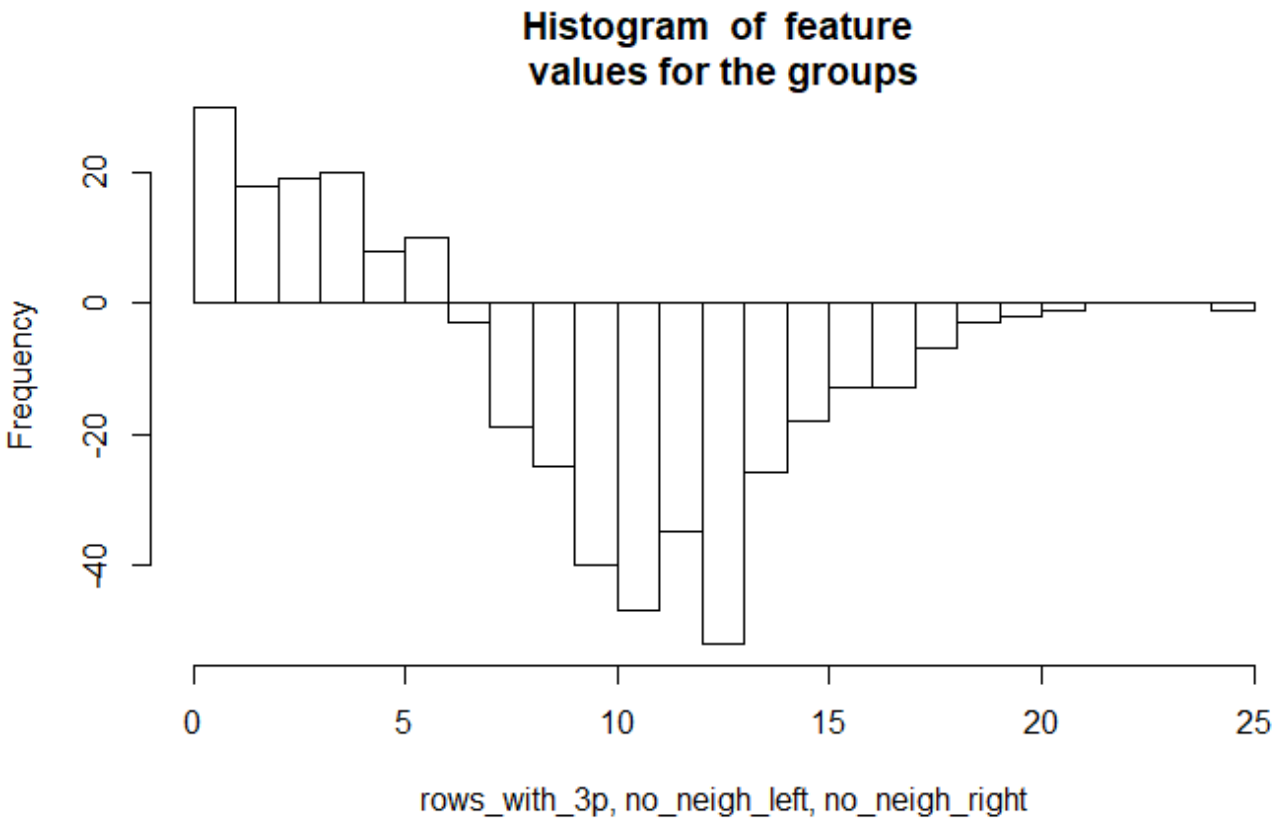
Hide

```
morehist = with(subset(df, y == data$no_neigh_right), hist(y, breaks = fullhist$breaks))
```



Hide

```
#combine the hists
combhist = fullhist
combhist$counts = zerohist$counts - oneshist$counts - morehist$counts
plot(combhist, main="Histogram  of  feature
values for the groups", xlab= "rows_with_3p, no_neigh_left, no_neigh_right")
```



Hide

```
NA
NA
NA
```

Section 3.3

Now, to check which features are most useful to discriminate whether an image is a letter or a non-letter, following are the two statistic analysis we can use:

1. Randomization test
2. T-Test

To check which of the either test to use, we first calculate the skewdness of the data.

If the data is extremely skewed (< -1 or > 1), we use randomization test. However as we calculated in Section 3.2, all of our features have skewdness value of higher than -1 and less than 1. Also, in practice t test is better than randomization test.

Thus we will only use t-test. We will use two sided t-tests since mean of both the groups differ.

We perform the t-test on all the features to calculate the t score and p-values:

Hide

```
t.test(only_letters$nr_pix, only_non_letters$nr_pix)
```

```
    Welch Two Sample t-test

data:  only_letters$nr_pix and only_non_letters$nr_pix
t = 4.8573, df = 136.34, p-value =
3.218e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 3.890747 9.234253
sample estimates:
mean of x mean of y
  28.3625   21.8000
```

Hide

```
t.test(only_letters$rows_with_1, only_non_letters$rows_with_1)
```

```
    Welch Two Sample t-test

data:  only_letters$rows_with_1 and only_non_letters$rows_with_1
t = -0.34319, df = 119.5, p-value =
0.7321
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.156442  0.814775
sample estimates:
mean of x mean of y
 4.562500  4.733333
```

Hide

```
t.test(only_letters$cols_with_1, only_non_letters$cols_with_1)
```

```
    Welch Two Sample t-test

data:  only_letters$cols_with_1 and only_non_letters$cols_with_1
t = -4.2177, df = 88.43, p-value =
5.95e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.5867660 -0.9299007
sample estimates:
mean of x mean of y
 1.775000  3.533333
```

Hide

```
t.test(only_letters$rows_with_3p, only_non_letters$rows_with_3p)
```

```
    Welch Two Sample t-test

data:  only_letters$rows_with_3p and only_non_letters$rows_with_3p
t = 5.3652, df = 137.2, p-value =
3.348e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.318128 2.856872
sample estimates:
mean of x mean of y
   4.5375    2.4500
```

Hide

```
t.test(only_letters$cols_with_3p, only_non_letters$cols_with_3p)
```

```
    Welch Two Sample t-test

data:  only_letters$cols_with_3p and only_non_letters$cols_with_3p
t = 4.318, df = 129.94, p-value =
3.098e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.7224314 1.9442353
sample estimates:
mean of x mean of y
 4.100000  2.766667
```

Hide

```
t.test(only_letters$aspect_ratio, only_non_letters$aspect_ratio)
```

```
    Welch Two Sample t-test

data:  only_letters$aspect_ratio and only_non_letters$aspect_ratio
t = -2.4968, df = 131.29, p-value =
0.01377
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.1059443 -0.2440557
sample estimates:
mean of x mean of y
    3.375     4.550
```

Hide

```
t.test(only_letters$neigh_1, only_non_letters$neigh_1)
```

```
    Welch Two Sample t-test

data:  only_letters$neigh_1 and only_non_letters$neigh_1
t = -7.4769, df = 95.431, p-value =
3.671e-11
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.483547 -1.441453
sample estimates:
mean of x mean of y
   1.5875    3.5500
```

Hide

```
t.test(only_letters$no_neigh_above, only_non_letters$no_neigh_above)
```

```
    Welch Two Sample t-test

data:  only_letters$no_neigh_above and only_non_letters$no_neigh_above
t = -0.64631, df = 123.63, p-value =
0.5193
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.4388086  0.7304753
sample estimates:
mean of x mean of y
 6.912500  7.266667
```

Hide

```
t.test(only_letters$no_neigh_below, only_non_letters$no_neigh_below)
```

```
    Welch Two Sample t-test

data:  only_letters$no_neigh_below and only_non_letters$no_neigh_below
t = -0.10477, df = 127.09, p-value =
0.9167
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.160044  1.043378
sample estimates:
mean of x mean of y
 7.325000  7.383333
```

Hide

```
t.test(only_letters$no_neigh_left, only_non_letters$no_neigh_left)
```

```
    Welch Two Sample t-test

data:  only_letters$no_neigh_left and only_non_letters$no_neigh_left
t = -1.428, df = 136.96, p-value =
0.1556
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.947561  0.314228
sample estimates:
mean of x mean of y
 12.65000  13.46667
```

Hide

```
t.test(only_letters$no_neigh_right, only_non_letters$no_neigh_right)
```

```
    Welch Two Sample t-test

data:  only_letters$no_neigh_right and only_non_letters$no_neigh_right
t = -4.2705, df = 137.23, p-value =
3.62e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.950467 -1.082866
sample estimates:
mean of x mean of y
 10.10000  12.11667
```

Hide

```
t.test(only_letters$no_neigh_horiz, only_non_letters$no_neigh_horiz)
```

```
    Welch Two Sample t-test

data:  only_letters$no_neigh_horiz and only_non_letters$no_neigh_horiz
t = -0.72456, df = 118.2, p-value =
0.4702
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.1153848  0.9820515
sample estimates:
mean of x mean of y
 10.65000  11.21667
```

Hide

```
t.test(only_letters$no_neigh_vert, only_non_letters$no_neigh_vert)
```

```
    Welch Two Sample t-test

data:  only_letters$no_neigh_vert and only_non_letters$no_neigh_vert
t = 3.2365, df = 134.43, p-value =
0.001524
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.234842 5.115158
sample estimates:
mean of x mean of y
   13.675    10.500
```

Hide

```
t.test(only_letters$custom, only_non_letters$custom)
```

```
    Welch Two Sample t-test

data:  only_letters$custom and only_non_letters$custom
t = -2.8812, df = 137.84, p-value =
0.004595
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.3771229 -0.2562105
sample estimates:
mean of x mean of y
 5.350000  6.166667
```

Now we know that if the p-value of any of the t-test is less than 0.05, that means that it is significant and there is indeed a difference. Thus features with p-values < 0.05 will be used to discriminate between letters and non-letters.

Based on the above t-tests,

cols_with_3p, neigh_1, no_neigh_vert, custom have p-values less than 0.05.

Thus, **cols_with_3p, neigh_1, no_neigh_vert, custom** features are most suitable for discriminating whether an image is a letter or a non-letter.

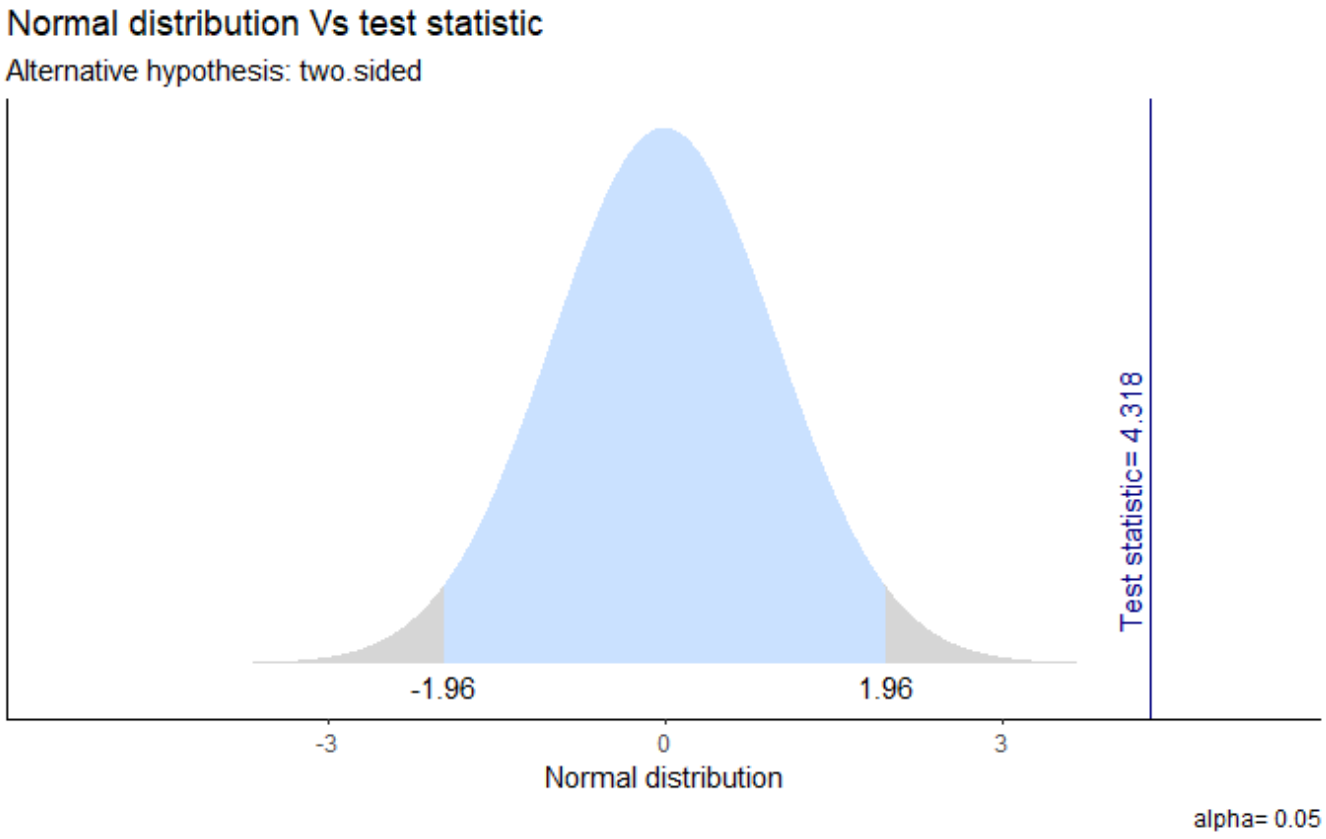
These features will be really helpful for further machine learning task. This gives statistical proof that the letters and non-letters are vastly different.

Plotting **cols_with_3p** t-test:

Hide

```
library(gginference)

ggttest(t.test(only_letters$cols_with_3p, only_non_letters$cols_with_3p))
```

```
Warning: geom_vline(): Ignoring `data` because `xintercept` was provided.
```

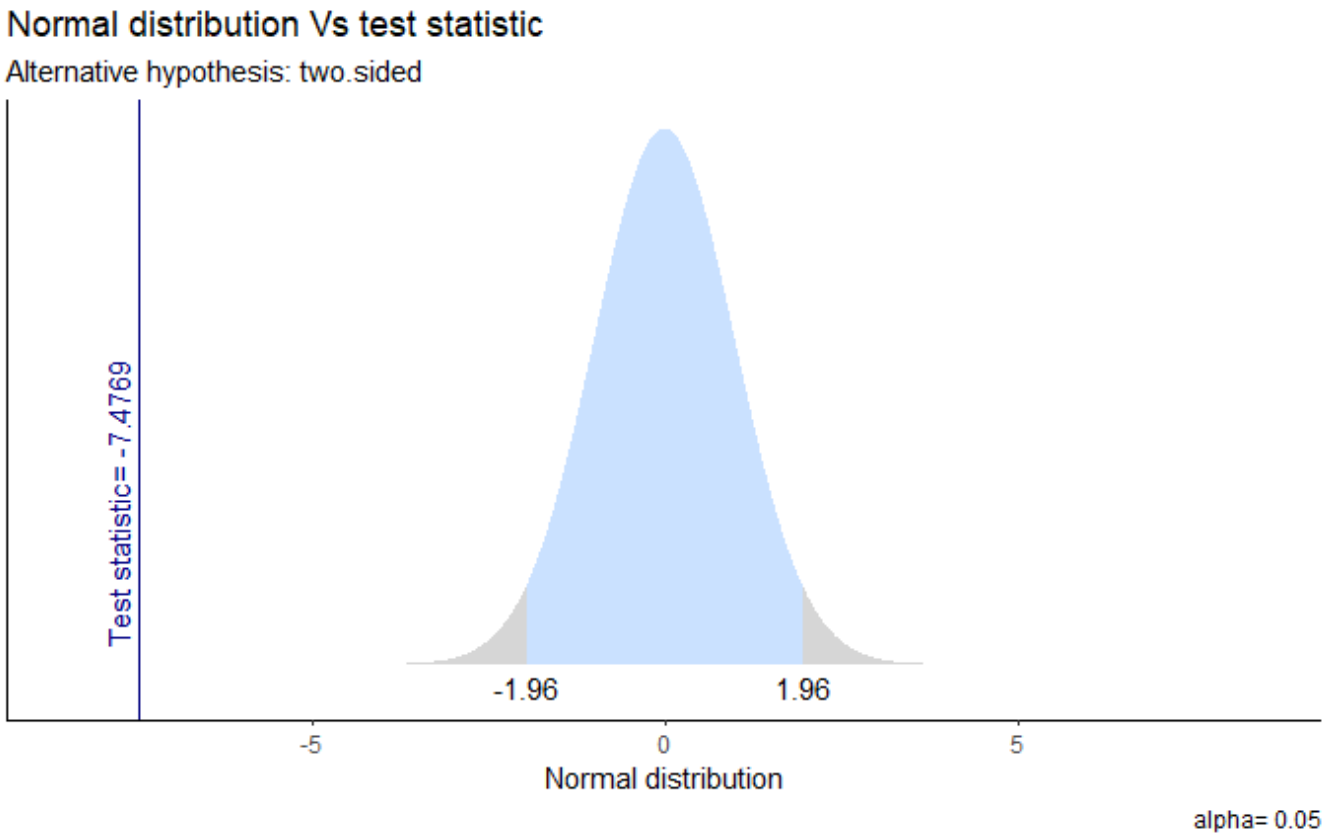


Plotting **neigh_1** t-test:

Hide

```
ggttest(t.test(only_letters$neigh_1, only_non_letters$neigh_1))
```

```
Warning: geom_vline(): Ignoring `data` because `xintercept` was provided.
```

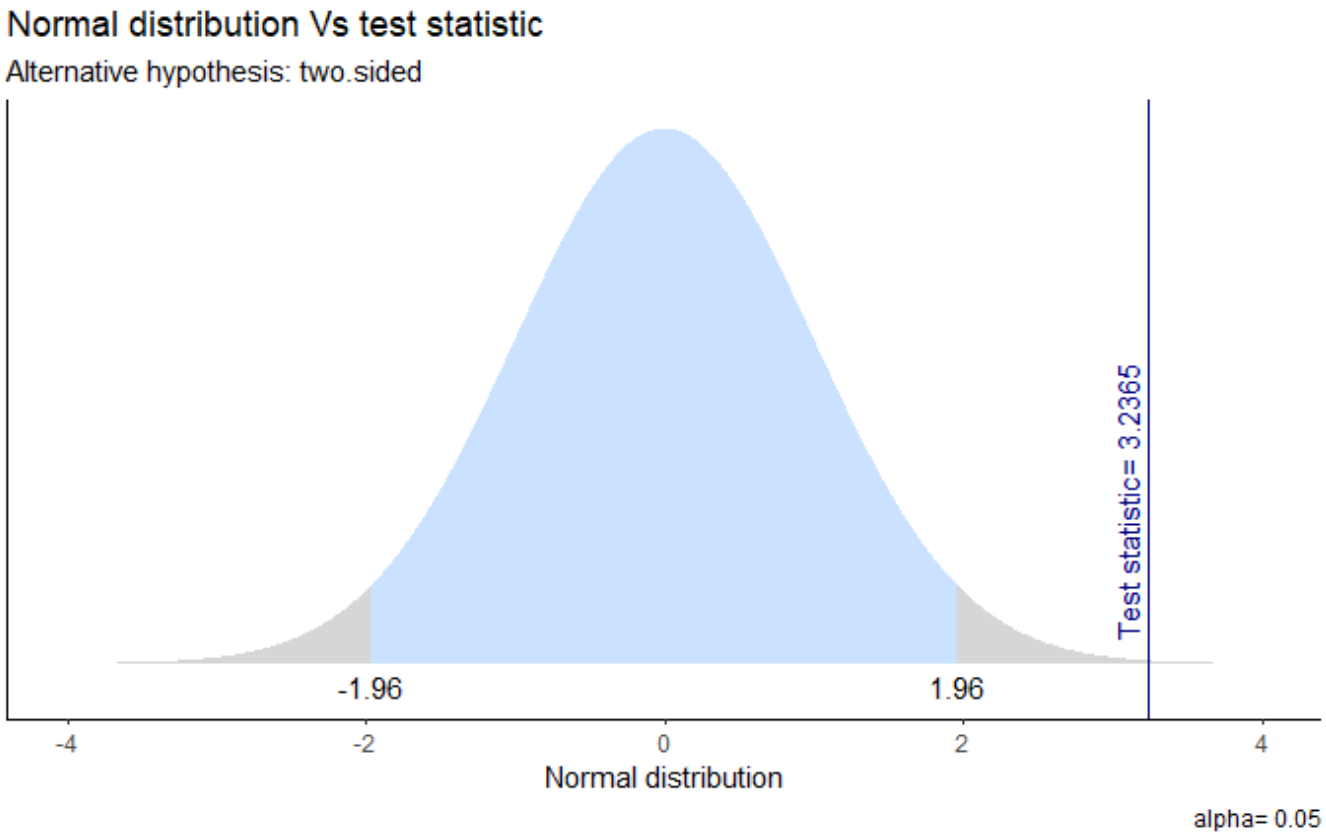


Plotting **no_neigh_vert** t-test:

Hide

```
ggttest(t.test(only_letters$no_neigh_vert, only_non_letters$no_neigh_vert))
```

```
Warning: geom_vline(): Ignoring `data` because `xintercept` was provided.
```

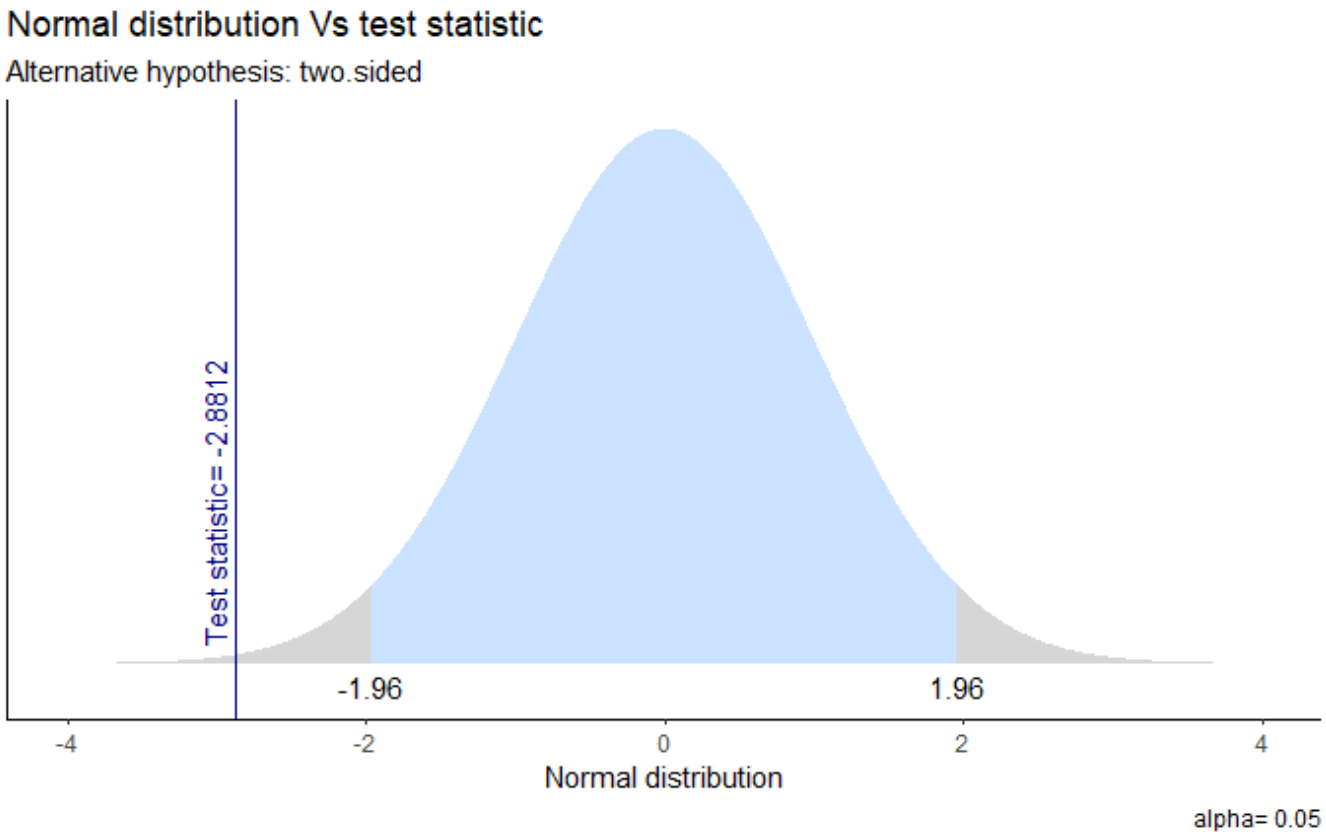


Plotting **custom** t-test:

Hide

```
ggttest(t.test(only_letters$custom, only_non_letters$custom))
```

```
Warning: geom_vline(): Ignoring `data` because `xintercept` was provided.
```



Section 3.4

We perform the 'cor()' function instead of 'cor.test()' for multiple columns. In this case, these columns means features.

Thus we use the following code to measure the degree of linear association between the features:

Hide

```
# Calculating degree of linear association between features

cor(data[, 3:15])
```

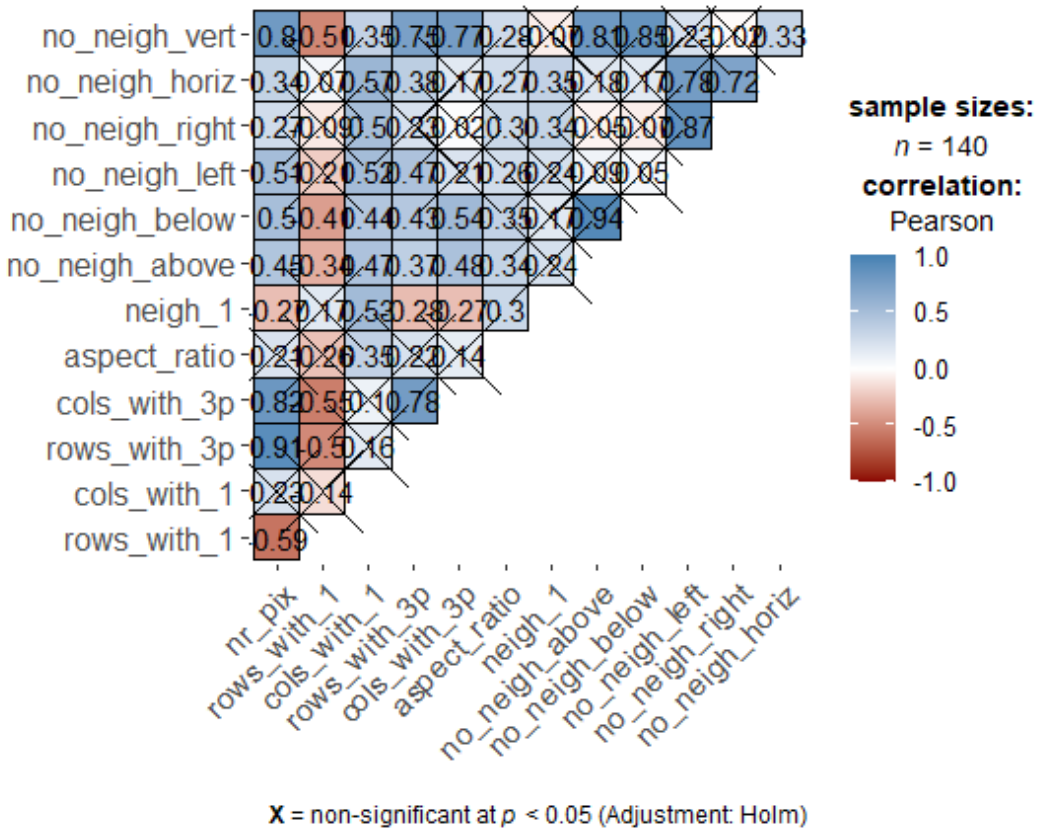
Here we get different columns and rows showing the linear association between different features. All the values seen from the code are correlation coefficient (r).

We know that if r > 0.7, it means there's high association. If r is between 0.5 - 0.7, there is moderate association. And if r is between 0.3 - 0.5, there is low association.

From the above r values, following are the features with high association:

Hide

```
correlation_data <- data

ggstatsplot::ggcorrmat(
  data = correlation_data[, 3:15],
  type = "parametric",
  colors = c("darkred", "white", "steelblue")
)
```



(Image in code is more enlarged)

Following are the top three features with highest linear association (here we are stating features with r > 0.7). Note that there are many features with high linear association but we're only stating the top three. We can see features with high-correlation having blue color in the above plot. Top three features with highest correlation:

no_neigh_below and **no_neigh_above**

- These features might have high association because they're in the opposite side, and these features accept black pixels on the left and right side, which will be very similar. Hence high association.

rows_with_3p and **nr_pix**

- Number of pixels objectively means number of black pixel which will be a high value. The strong linear association just shows that instead of columns, it is rows that have the most number of black pixels and in this case, since this is calculating rows with three black pixels which is higher than rows with just one black pixel, the linear association is high with number of black pixels in total.

no_neigh_right and **no_neigh_left**

- As in the first case, since they're in the opposite side, these features accept black pixels above and

below which will be similar values. That is why they have high association.

Section 4

Section 4.1

To predict the value of aspect_ratio from other features, we have different regression models to pick from. I picked the AIC approach which does step-elimination using 'stepAIC' function. This approach is similar to the adjusted R^2 approach. This model penalizes for adding more variables to the model and does step elimination to remove the variables till the perfect model is found.

As per AIC, the best model for feature selection allows to see the most variation with just lowest independent variables possible that are in the data.

Lower the AIC value, the better the model is.

For this model, we use backward elimination that means that the model will begin with all the variables at first and then step-wise elimination the variables depending on the AIC value to find the best model. This is very helpful in our case as our data have large amount of variables. Starting from one variable to add all the variables will variably take more time and computation than just starting with all at once and gradually reducing.

First step, we add the library that allows us to use the AIC model:

Hide

```
# Adding MASS library

library(MASS)
```

```
Warning: package 'MASS' was built under R version 4.1.3
```

We create a model of different combinations first:

Hide

```
model <- lm(aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_3p +
              cols_with_3p + neigh_1 + no_neigh_above + no_neigh_below + no_neigh_left +
              no_neigh_right + no_neigh_horiz + no_neigh_vert + custom, data = data)

summary(model)
```

As before, we do not include **connected_areas** and **eyes** feature as they were randomly sampled and have no statistical significane.

We then run the stepAIC model with backwards elimination.

Hide

```
stepAIC(model, direction = "backward")
```

```
Start:  AIC=253.94
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_3p +
    cols_with_3p + neigh_1 + no_neigh_above + no_neigh_below +
    no_neigh_left + no_neigh_right + no_neigh_horiz + no_neigh_vert +
    custom

                 Df Sum of Sq    RSS    AIC
- no_neigh_vert   1     0.000 703.08 251.94
- no_neigh_horiz  1     0.006 703.09 251.94
- rows_with_3p    1     0.012 703.09 251.94
- no_neigh_left   1     0.136 703.22 251.96
- no_neigh_above  1     0.294 703.37 251.99
- cols_with_1     1     1.694 704.77 252.27
- no_neigh_below  1     4.762 707.84 252.88
- rows_with_1     1     6.649 709.73 253.25
- neigh_1         1     8.980 712.06 253.71
<none>                       703.08 253.94
- cols_with_3p    1    11.171 714.25 254.14
- nr_pix          1    14.162 717.24 254.73
- no_neigh_right  1    22.911 725.99 256.43
- custom          1    69.386 772.47 265.11

Step:  AIC=251.94
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + rows_with_3p +
    cols_with_3p + neigh_1 + no_neigh_above + no_neigh_below +
    no_neigh_left + no_neigh_right + no_neigh_horiz + custom

                 Df Sum of Sq    RSS    AIC
- rows_with_3p    1     0.011 703.09 249.94
- no_neigh_horiz  1     0.021 703.10 249.94
- no_neigh_left   1     0.160 703.24 249.97
- no_neigh_above  1     0.372 703.45 250.01
- cols_with_1     1     1.710 704.79 250.28
- no_neigh_below  1     5.388 708.47 251.00
- rows_with_1     1     7.374 710.45 251.40
- neigh_1         1     8.987 712.07 251.71
<none>                       703.08 251.94
- cols_with_3p    1    11.181 714.26 252.15
- nr_pix          1    20.578 723.66 253.97
- no_neigh_right  1    38.243 741.32 257.35
- custom          1    69.511 772.59 263.13

Step:  AIC=249.94
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + cols_with_3p +
    neigh_1 + no_neigh_above + no_neigh_below + no_neigh_left +
    no_neigh_right + no_neigh_horiz + custom

                 Df Sum of Sq    RSS    AIC
- no_neigh_horiz  1     0.016 703.11 247.94
- no_neigh_left   1     0.188 703.28 247.98
- no_neigh_above  1     0.360 703.45 248.01
```

```
- cols_with_1     1     1.712 704.80 248.28
- no_neigh_below  1     5.381 708.47 249.01
- rows_with_1     1     8.249 711.34 249.57
- neigh_1         1     9.179 712.27 249.75
<none>                       703.09 249.94
- cols_with_3p    1    11.321 714.41 250.17
- no_neigh_right  1    39.547 742.64 255.60
- nr_pix          1    41.606 744.70 255.99
- custom          1    93.024 796.12 265.33

Step:  AIC=247.94
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + cols_with_3p +
    neigh_1 + no_neigh_above + no_neigh_below + no_neigh_left +
    no_neigh_right + custom

                Df Sum of Sq    RSS    AIC
- no_neigh_left   1     0.182 703.29 245.98
- no_neigh_above  1     0.406 703.51 246.02
- cols_with_1     1     1.845 704.95 246.31
- no_neigh_below  1     5.889 709.00 247.11
- neigh_1         1     9.223 712.33 247.77
<none>                       703.11 247.94
- rows_with_1     1    11.211 714.32 248.16
- cols_with_3p    1    11.511 714.62 248.22
- no_neigh_right  1    40.126 743.23 253.71
- nr_pix          1    41.775 744.88 254.02
- custom          1    96.638 799.74 263.97

Step:  AIC=245.98
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + cols_with_3p +
    neigh_1 + no_neigh_above + no_neigh_below + no_neigh_right +
    custom

                Df Sum of Sq    RSS    AIC
- no_neigh_above  1     0.509 703.80 244.08
- cols_with_1     1     1.717 705.01 244.32
- no_neigh_below  1     6.769 710.06 245.32
- neigh_1         1     9.050 712.34 245.77
<none>                       703.29 245.98
- cols_with_3p    1    11.332 714.62 246.22
- rows_with_1     1    11.462 714.75 246.24
- nr_pix          1    44.962 748.25 252.65
- no_neigh_right  1    77.292 780.58 258.57
- custom          1   101.650 804.94 262.88

Step:  AIC=244.08
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_1 + cols_with_3p +
    neigh_1 + no_neigh_below + no_neigh_right + custom

                Df Sum of Sq    RSS    AIC
- cols_with_1     1     1.571 705.37 242.39
```

```
- neigh_1         1     8.543 712.34 243.77
<none>                       703.80 244.08
- cols_with_3p    1    11.040 714.84 244.26
- rows_with_1     1    11.070 714.87 244.26
- no_neigh_below  1    15.166 718.96 245.06
- nr_pix          1    44.478 748.28 250.66
- no_neigh_right  1    80.060 783.86 257.16
- custom          1   101.273 805.07 260.90

Step:  AIC=242.39
aspect_ratio ~ nr_pix + rows_with_1 + cols_with_3p + neigh_1 +
    no_neigh_below + no_neigh_right + custom

                 Df Sum of Sq    RSS    AIC
- rows_with_1     1    10.073 715.44 242.38
<none>                       705.37 242.39
- cols_with_3p    1    11.874 717.24 242.73
- neigh_1         1    12.380 717.75 242.83
- no_neigh_below  1    25.585 730.95 245.38
- nr_pix          1    43.230 748.60 248.72
- no_neigh_right  1    96.485 801.85 258.34
- custom          1   101.037 806.41 259.13

Step:  AIC=242.38
aspect_ratio ~ nr_pix + cols_with_3p + neigh_1 + no_neigh_below +
    no_neigh_right + custom

                 Df Sum of Sq    RSS    AIC
- cols_with_3p    1    10.271 725.71 242.37
<none>                       715.44 242.38
- neigh_1         1    13.723 729.17 243.04
- no_neigh_below  1    27.568 743.01 245.67
- nr_pix          1    33.864 749.31 246.85
- no_neigh_right  1    86.799 802.24 256.41
- custom          1   115.137 830.58 261.27

Step:  AIC=242.37
aspect_ratio ~ nr_pix + neigh_1 + no_neigh_below + no_neigh_right +
    custom

                 Df Sum of Sq    RSS    AIC
<none>                       725.71 242.37
- neigh_1         1    14.132 739.84 243.07
- no_neigh_below  1    24.230 749.94 244.97
- nr_pix          1    24.827 750.54 245.08
- no_neigh_right  1    99.849 825.56 258.42
- custom          1   105.770 831.48 259.42

Call:
lm(formula = aspect_ratio ~ nr_pix + neigh_1 + no_neigh_below +
    no_neigh_right + custom, data = data)
```

```
Coefficients:
   (Intercept)          nr_pix
      -8.69453         0.08592
       neigh_1  no_neigh_below
       0.25492         0.19174
no_neigh_right          custom
       0.39173         0.71129
```

As we can see the AIC model calculated the AIC values, and with the AIC value of **242.37**, the model showed that **nr_pix, neigh_1, no_neigh_below, no_neigh_right, custom** are the best features to predict **aspect_ratio**.

Once that is done, we fit a regression model with the above features and report the result.

Hide

```
# Regression model

result <- lm(aspect_ratio ~ nr_pix + neigh_1 + no_neigh_below + no_neigh_right + custom, data
= data)

result
```

```

Call:
lm(formula = aspect_ratio ~ nr_pix + neigh_1 + no_neigh_below +
    no_neigh_right + custom, data = data)

Coefficients:
   (Intercept)          nr_pix
      -8.69453         0.08592
       neigh_1  no_neigh_below
       0.25492         0.19174
no_neigh_right          custom
       0.39173         0.71129
```

Hide

```
summary(result)
```

```
Call:
lm(formula = aspect_ratio ~ nr_pix + neigh_1 + no_neigh_below +
    no_neigh_right + custom, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-5.8612 -1.4506 -0.5038  1.1917  6.9637

Coefficients:
               Estimate Std. Error t value
(Intercept)    -8.69453    1.80543  -4.816
nr_pix          0.08592    0.04013   2.141
neigh_1         0.25492    0.15781   1.615
no_neigh_below  0.19174    0.09065   2.115
no_neigh_right  0.39173    0.09123   4.294
custom          0.71129    0.16095   4.419
               Pr(>|t|)
(Intercept)    3.90e-06 ***
nr_pix           0.0341 *
neigh_1          0.1086
no_neigh_below   0.0363 *
no_neigh_right 3.35e-05 ***
custom         2.03e-05 ***
---
Signif. codes:
  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1
  ‘ ’ 1

Residual standard error: 2.327 on 134 degrees of freedom
Multiple R-squared:  0.3491,    Adjusted R-squared:  0.3248
F-statistic: 14.37 on 5 and 134 DF,  p-value: 2.918e-11
```

Section 4.2

From Section 3.3, we figured that the most discriminatory features to differentiate letters and non-letters are **cols_with_3p, neigh_1, no_neigh_vert, custom**.

We will use these to fit a logistic regression model. At first, we create a separate dataframe to use the logistic regression model.

Hide

```
data_for_regression <- data

data_for_regression
```

Since we need to use categorical factors as the logistic regression model only uses them, we need to look out for our data.

Our data raw is not a categorical factor, we need to convert them to categorical features as done below:

Hide

```
data_for_regression$cols_with_3p <- as.factor(data_for_regression$cols_with_3p)

data_for_regression$neigh_1 <- as.factor(data_for_regression$neigh_1)

data_for_regression$no_neigh_vert <- as.factor(data_for_regression$no_neigh_vert)

data_for_regression$aspect_ratio <- as.factor(data_for_regression$aspect_ratio)
```

We were asked to choose only three features so we used **cols_with_3p, neigh_1, no_neigh_vert**.

We found the most discriminating feature to be **cols_with3p** with p-value of -1.902, so we'll use that for the logistic regression model.

Now we fit a logistic regression model and summarize the results:

Hide

```
logistic <- glm(aspect_ratio ~ cols_with_3p, data = data_for_regression, family="binomial")

summary(logistic)
```

```
Call:
glm(formula = aspect_ratio ~ cols_with_3p, family = "binomial",
    data = data_for_regression)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6557   0.2444   0.3381   0.5701   1.1774

Coefficients:
                Estimate Std. Error z value
(Intercept)      1.73460    0.62622   2.770
cols_with_3p2    0.49899    0.87247   0.572
cols_with_3p3    1.76191    1.19267   1.477
cols_with_3p4    1.09861    1.20457   0.912
cols_with_3p5   16.83147 2174.21296   0.008
cols_with_3p6   -0.02985    0.99150  -0.030
cols_with_3p7   -0.48184    1.01736  -0.474
cols_with_3p8   16.83147 3765.84720   0.004
cols_with_3p9   -1.73460    1.54666  -1.122
cols_with_3p10  16.83147 6522.63863   0.003
               Pr(>|z|)
(Intercept)     0.00561 **
cols_with_3p2   0.56737
cols_with_3p3   0.13960
cols_with_3p4   0.36175
cols_with_3p5   0.99382
cols_with_3p6   0.97598
cols_with_3p7   0.63577
cols_with_3p8   0.99643
cols_with_3p9   0.26207
cols_with_3p10  0.99794
---
Signif. codes:
  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1
  ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 86.548  on 139  degrees of freedom
Residual deviance: 76.837  on 130  degrees of freedom
AIC: 96.837

Number of Fisher Scoring iterations: 17
```

Here, the deviance residuals from the summary look good confirming the model is fit as most of them are close to being centered on zero and are roughly symmetrical other than one exception of Min.

We can also see that the Pr value of **cols_with_3p** is greater than 0. This means that as cols_with_3p increases, probability of predicting the aspect_ratio increases.

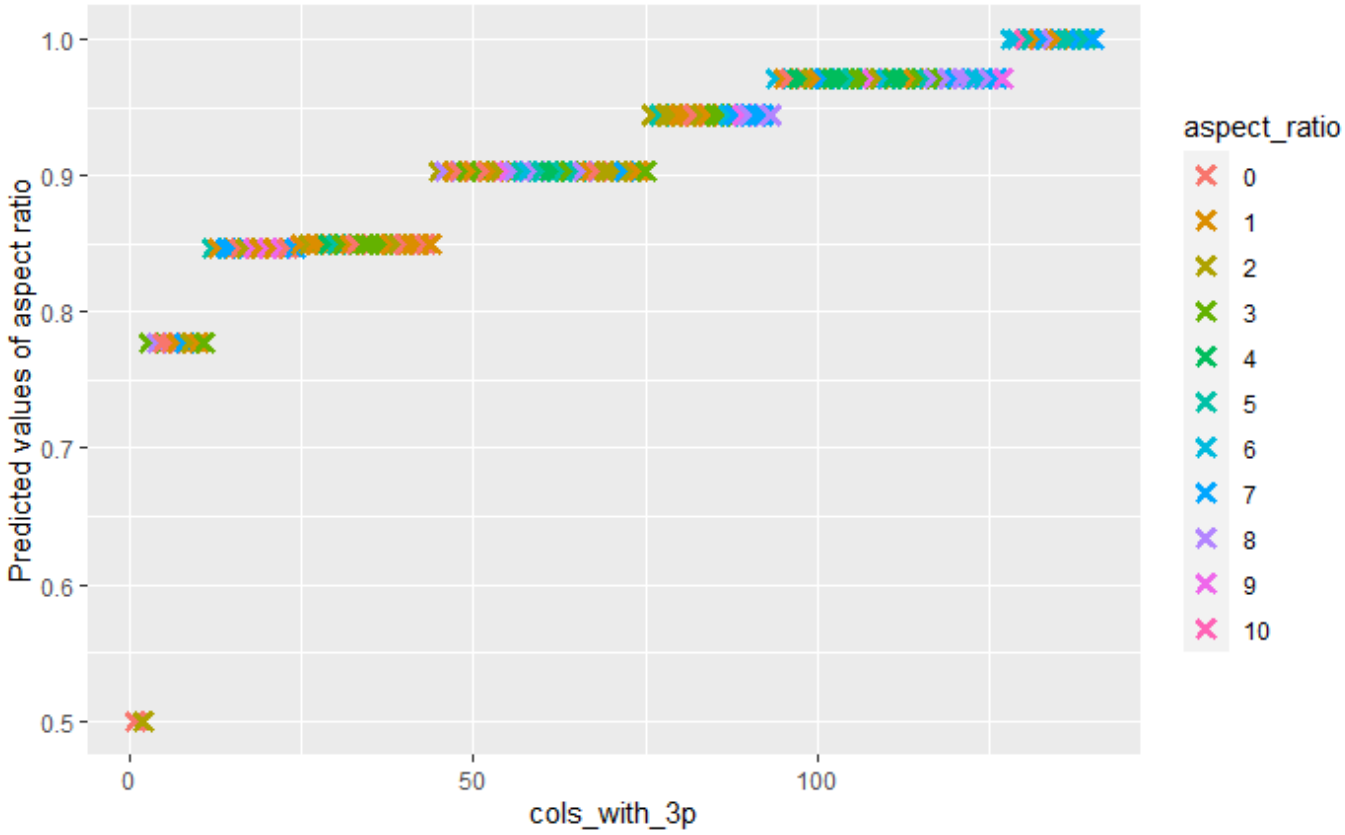
Since the p-value is below 0.05, the log(odds) are statistically significant.

I decided to use the regression model for **aspect_ratio** feature as that is what I deemed to be useful. I unfortunately had difficulties in how to fit a regression model for the entire data, so I used it for aspect_ratio feature.

Then I plotted the visualisation of the regression model:

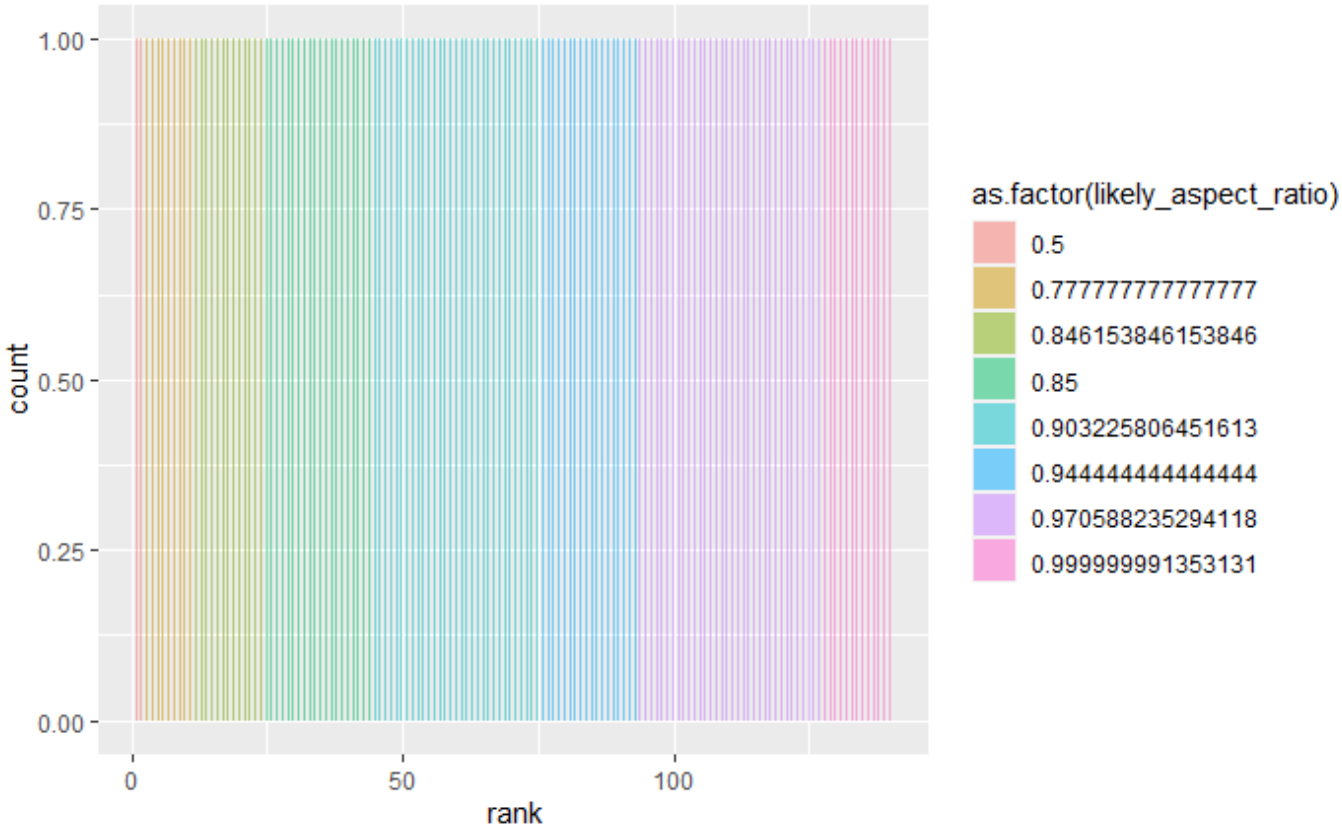
Hide

```
# To draw graph, we start by creating a new data frame that contains the
# probabilities of predicted aspect ration with actual
# aspect _ratio

predicted_data <- data.frame(likely_aspect_ratio = logistic$fitted.values,
                             aspect_ratio = data_for_regression$aspect_ratio)

# then we sort data frame from low probabilities to high probabilities

predicted_data <- predicted_data[order(predicted_data$likely_aspect_ratio, decreasing =FALS
E),]

# Then we add a new column to the data frame that has the rank of each sample, from low proba
bility
# to high probability

predicted_data$rank <- 1:nrow(predicted_data)

ggplot(data=predicted_data, aes(x=rank, y=likely_aspect_ratio)) +
  geom_point(aes(color=aspect_ratio), alpha = 1, shape = 4, stroke = 2) +
  xlab("cols_with_3p") +
  ylab("Predicted values of aspect ratio")
```



Hide

```
ggplot(predicted_data, aes(x=rank, fill=as.factor(likely_aspect_ratio))) +
  geom_histogram(binwidth=.2, alpha=.5, position='identity')
```



I made both a graph and a histogram. The probability beteween 0 and 1 which is good.

As the **cols_with_3p** value increases, the probability of predicting **aspect_ratio** increases.

Section 4.3

For section 4.3, I will explain in steps how I have tackled this issue.

1. At first, I calculated median of **nr_pixel,aspect_ratio, neigh_1** of every image. Then I turn them into categorical variable, naming them as **split1, split2, split3**.

Hide

```
# Calculating medians


median_of_nr_pixels <- colMedians(data.matrix(data[, 3]))
median_of_nr_pixels
```

```
[1] 25
```

Hide

```
median_of_aspect_ratio <- colMedians(data.matrix(data[, 8]))
median_of_aspect_ratio
```

```
[1] 3
```

Hide

```
median_of_neigh_1 <- colMedians(data.matrix(data[, 9]))
median_of_neigh_1
```

```
[1] 2
```

Hide

```
# Categorical variable for each

data$split1 <- as.factor(ifelse(data$nr_pix > median_of_nr_pixels, 1, 0))

data$split2 <- as.factor(ifelse(data$aspect_ratio > median_of_aspect_ratio, 1, 0))

data$split3 <- as.factor(ifelse(data$neigh_1 > median_of_neigh_1, 1, 0))
```

In the above code, in split variables, if the values in **nr_pix, aspect_ratio, neigh_1** are greater than the median of the entire column, the split variable will be 1. Otherwise, it will be 0.

2. After that is done, I count the number of 1 in individual split variables. This will be the total number of 1s in each split variable.

Hide

```
# Counting number of 1s in individual split variables

count_of_split_1 = 0

for (i in data$split1) {
  if (i == 1) {
    count_of_split_1 = count_of_split_1 + 1
  }
}


count_of_split_2 = 0

for (i in data$split2) {
  if (i == 1) {
    count_of_split_2 = count_of_split_2 + 1
  }
}


count_of_split_3 = 0

for (i in data$split3) {
  if (i == 1) {
    count_of_split_3 = count_of_split_3 + 1
  }
}
```

3. After that, we calculate proportions for letters, faces and exclamatory mark images in the files.

I'll explain how I did it for letters and this same algorithm is followed for faces and exclamatory marks.

I first make three count variabnles that store the number of times the letters' value is 1 in the split variables.

Then I go through each values in the **nr_pixel, aspect_ratio, neigh_1** columns of **just letters**. If the value equals to 1, the count value is incremented by one. Following is the code showing that:

Hide

```
# for letters

letters_count_of_split_1 = 0

for (i in data[1:80, 19]) {
  if (i == 1) {
    letters_count_of_split_1 = letters_count_of_split_1 + 1
  }
}

letters_count_of_split_1
```

```
[1] 48
```

Hide

```
letters_count_of_split_2 = 0

for (i in data[1:80, 20]) {
  if (i == 1) {
    letters_count_of_split_2 = letters_count_of_split_2 + 1
  }
}

letters_count_of_split_2
```

```
[1] 32
```

Hide

```
letters_count_of_split_3 = 0

for (i in data[1:80, 21]) {
  if (i == 1) {
    letters_count_of_split_3 = letters_count_of_split_3 + 1
  }
}

letters_count_of_split_3
```

```
[1] 15
```

Now to calculate the proportion that the letters have in each split variable, I divide the count of letters in each split variable with the total number of counts in the split variable.

Since this answer can be in many decimal points and we want a proportion, I round off the value to 2 digits. Following is the code:

Hide

```
letters_split_1_proportion = round(letters_count_of_split_1 / count_of_split_1, digits = 2)
letters_split_1_proportion
```

```
[1] 0.74
```

Hide

```
letters_split_2_proportion = round(letters_count_of_split_2 / count_of_split_2, digits = 2)
letters_split_2_proportion
```

```
[1] 0.48
```

Hide

```
letters_split_3_proportion = round(letters_count_of_split_3 / count_of_split_3, digits = 2)
letters_split_3_proportion
```

```
[1] 0.26
```

This is repeated for face and exclamatory mark.

4.

Then finally, I create a table with all the proportions. For this, I create a 3x3 matrix and then add rownames and columnnames to it. Then I view it as a table.

Hide

```
result <- matrix(c(letters_split_1_proportion, letters_split_2_proportion, letters_split_3_pr
oportion,
                   face_split_1_proportion, face_split_2_proportion, face_split_3_proportion,
                   xclaim_split_1_proportion, xclaim_split_2_proportion, xclaim_split_3_propor
tion),
                 ncol = 3, nrow = 3)
```

```
Error in matrix(c(letters_split_1_proportion, letters_split_2_proportion,  :
  object 'face_split_1_proportion' not found
```

The final result shows as:

Hide

```
final_result
```

```
                  Split1 Split2 Split3
Letters             0.74   0.26   0.00
Faces               0.48   0.49   0.49
Exclamation Marks   0.26   0.68   0.68
```

This table shows the proportion of 1s for each of the three classes, **letters, faces and xclaim**

Section 4.4

Your work for this subsection here.