# Analytics Programming Assessment

Ryan Greenup - 17805315

## Contents



# Preamble

## Load Packages

```
## (01) Clean up the Iris Data

# Preamble
## Install Pacman
load.pac <- function() {
```

```r
  if(require("pacman")){
    library(pacman)
  }else{
    install.packages("pacman")
    library(pacman)
  }

  pacman::p_load(xts, sp, gstat, ggplot2, rmarkdown, reshape2, ggmap,
                 parallel, dplyr, plotly, tidyverse, reticulate, UsingR, Rmpfr,
                 swirl, corrplot, gridExtra, mise, latex2exp, tree, rpart, knitr,
                 bookdown, tidyverse, tidyr, jpeg, grid)

}

load.pac()
```

```
## Loading required package: pacman
```

### Knitr Configuration

#### Make Chunks verbose

```r
knitr::opts_chunk$set(echo = TRUE, eval = TRUE)
```

#### Set Figure Locations

```r
knitr::opts_chunk$set(
  fig.path = "figure/"
)
```

# Question 1

## d. Submission Receipt

The submission receipt for the preliminary assessment is provided as a screenshot at figure 1 and the recipt number is:

```
95d652b5-e2e7-4f43-b7e5-3290b9e4ef0b
```

```r
include_graphics("./Receipt.png")
```

## e. Set Variables

```r
studentname <- "Ryan Greenup"
studentno   <- 17805315
```

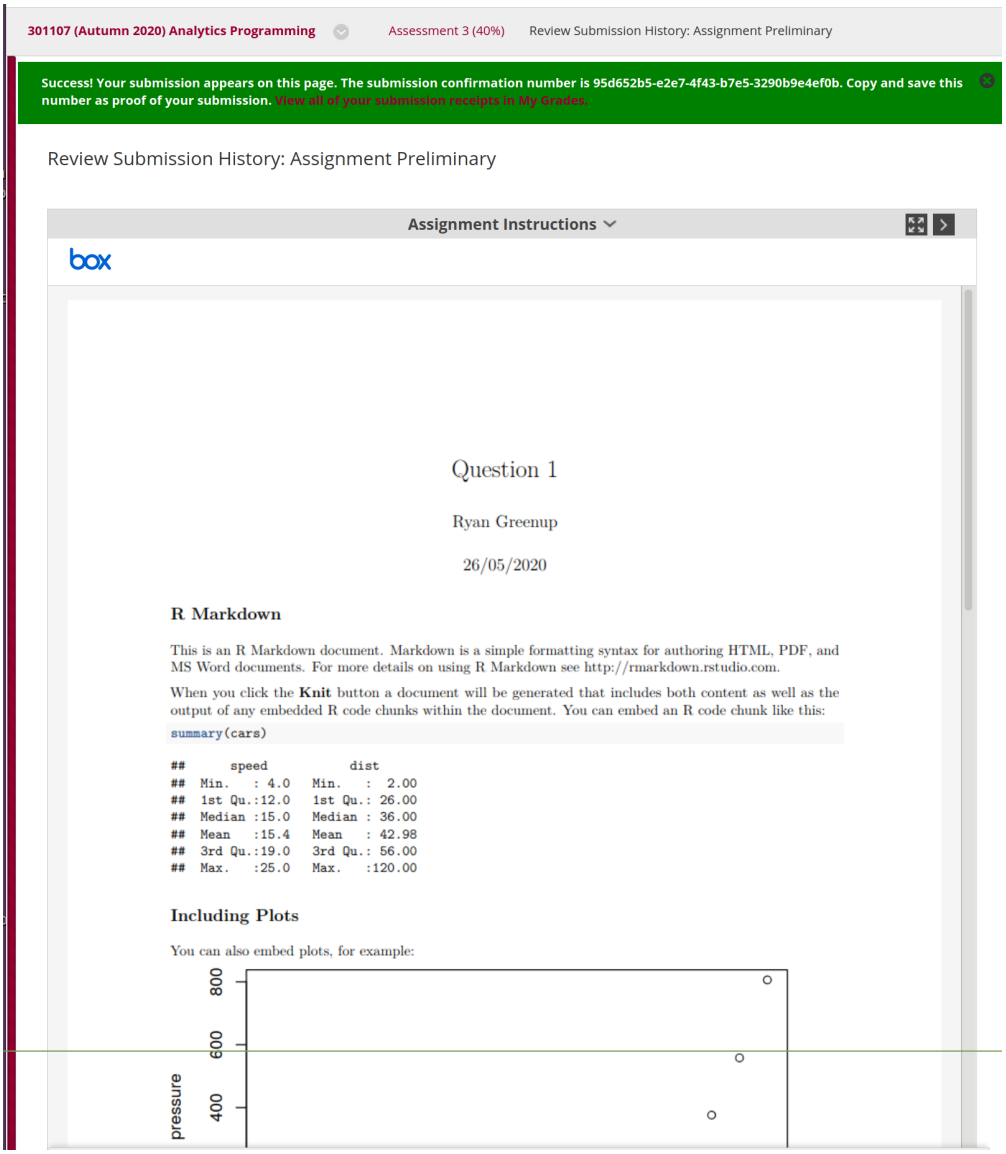



Figure 1: Receipt from preliminary submission

### f. Print Variables

#### i. Show the Code Without the Results

```
print(studentname)
print(studentno)
```

#### ii. Show the Evaluated Output in the Document

```
## [1] "Ryan Greenup"
```

```
## [1] 17805315
```

#### ii. Show the Evaluated Output and corresponding code in the Document

```
print(studentname)
```

```
## [1] "Ryan Greenup"
```

```
print(studentno)
```

```
## [1] 17805315
```

## Question 2

### 1. Write a function that takes a vector of numbers as its input data and randomly picks 10% of thenumbers, then writes that subsample into a new vector

the `sample` function can be used to take a sample from a vector:

```
subsample <- function(vec) {
  n <- length(vec)
  sample(vec, size = 0.1*n, replace = FALSE)
                            ## For a bootstrap replace should be TRUE
}
```

### 2. Test your function by creating a vector of 1000 random samples from a normal distribution withmean 1 and standard deviation 2, and then sampling 100 numbers from it.

The `rnorm` function can be used to generate normally distributed values and passing this to the previously defined `subsample` function will extract a random sample of 10% of the values:

```
norm_values <- rnorm(n = 1000, mean = 1, sd = 2)
(norm_ss <- subsample(norm_values))
```

```
##   [1] -3.97031786  1.26635354 -4.11372896  3.19083763 -1.02809205  2.06752821
##   [7]  3.11429614  0.82482273 -3.06094291  5.03570699 -1.31491438  0.25458026
##  [13]  3.22720593 -1.55049085  3.12186203  0.72283727  0.74540822  4.34461416
##  [19]  1.19549662  0.53874623  1.95078623  0.14022707  4.34638160 -0.57797454
##  [25]  2.75808314  2.53580974  0.84505101  2.69796030 -1.31244829  4.65419940
```

```
##  [31] -1.57135070  0.01481950  0.15028311  1.24008117  2.92860927  1.37894029
##  [37]  0.61057331  3.54730997  0.64820344  3.46194831  3.25182723 -3.32058572
##  [43] -2.86128282 -2.09482301 -1.65230411  3.81164406  3.44951880 -0.21232512
##  [49] -3.96110102 -1.55970190  0.43073007  1.91166993  0.88845352  1.89847895
##  [55]  2.97806393  2.47337065  2.03140049  3.77255575 -0.77425958  3.66678775
##  [61]  3.48379756  0.64680484 -0.12532606  3.85811847  1.56221206  3.55644227
##  [67]  0.03458989  3.81678581  4.12454592 -1.74671009  1.76042038 -1.07550937
##  [73]  5.15078177 -1.57291774 -0.07204647 -0.97497744 -0.40644119  3.72296585
##  [79]  3.32275464 -0.31130984 -0.17952010  1.87415206  0.90572660 -0.73715864
##  [85]  4.21556527 -1.08639667  5.63156604 -0.45563704 -1.67536013  2.19211160
##  [91]  2.00143105  2.84406534 -2.96814102  0.86356713 -0.85801310 -0.99550632
##  [97]  0.88559360 -0.30979502  1.45103324  6.48665561
```

### 3. Output the mean and standard deviation of the 100-number subsample

```r
rbind(c(
  "Mean"   = mean(norm_ss),
  "StdDev" = sd(norm_ss))
  )
```

```
##          Mean   StdDev
## [1,] 1.080283 2.331363
```

### 4. Call the function 100 times to get 100 different 100-number subsamples of the same 1000-numbervector as used above, calculate their means and standard deviations, and then plot histograms of themeans and standard deviations of the subsamples.

First resample the vector of values 100 times and produce a matrix using `replicate`[1]:

```r
## Resample the Values
boot_vals <- replicate(100, {
  subsample(norm_values)
})
```

Each Column of the output matrix corresponds to a seperate sample, so in order to get the mean and sd of each sample the `apply` function can be used, specifying the `MARGIN` argument as `2` in order to denote function operating column-wise.

```r
## Collumns correspond to the sample number and the rows to obsevations
##
## Calculate the mean values
sample_means <- apply(boot_vals, 2, mean)

## Calculate the STD Dev
sample_sd    <- apply(boot_vals, 2, sd)
```

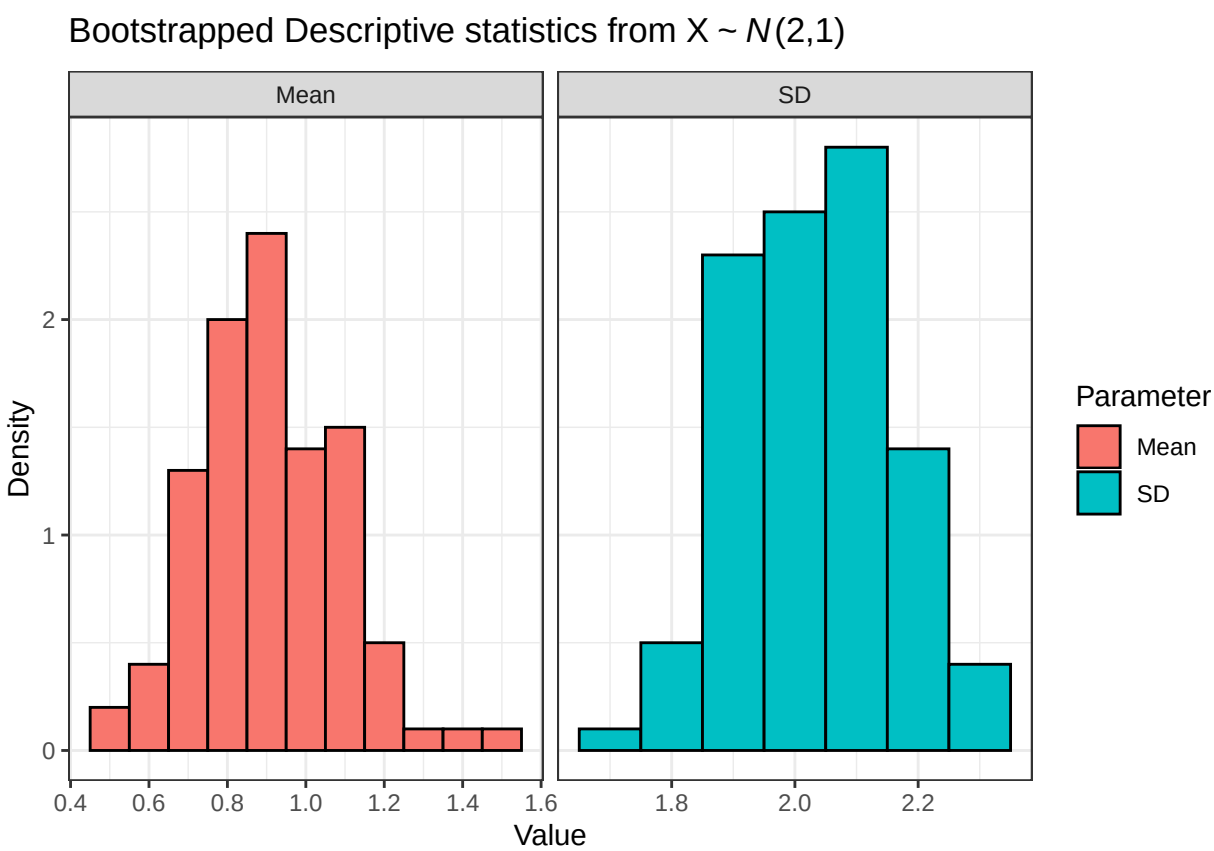
In order to plot the histograms it is first necessary to manipulate the data such that it is in a `tidy` format, this can be done by using the `tidyr` package, then the histograms can be plotted using `ggplot2`.

[1] `replicate` is a wrapper for `sapply` and is usually quicker than using a loop, loops using a dynamic vector are often slow, `replicate` has the added benefit that it is not necessary to define a static vector either.

```r
data <- data.frame(means = sample_means, sds = sample_sd)
tidy_data <- tidyr::pivot_longer(cols = c(means, sds), data = data, names_to = "Parameter")
tidy_data$Parameter <- factor(tidy_data$Parameter, labels = c("Mean", "SD"), ordered = FALSE)
tidy_data
```

```
## # A tibble: 200 x 2
##    Parameter value
##    <fct>     <dbl>
##  1 Mean      0.742
##  2 SD        2.13
##  3 Mean      0.586
##  4 SD        2.02
##  5 Mean      1.13
##  6 SD        1.70
##  7 Mean      0.912
##  8 SD        2.14
##  9 Mean      0.800
## 10 SD        2.12
## # ... with 190 more rows
```

```r
ggplot(tidy_data, aes(x = value, fill = Parameter, y = ..density..)) +
  geom_histogram(col = "black", binwidth = 0.1) +
  facet_grid(. ~ Parameter, scales = "free_x") +
  theme_bw()  +
  labs(x = "Value", y = "Density",
       title = TeX('Bootstrapped Descriptive statistics from $X \\sim \\textit{N} (2,1)$'))
```



```
##  stat_function(fun = dnorm, args = list(mean = mean(data$means),
##                                         sd = sd(data$means)))
```

# Question 3

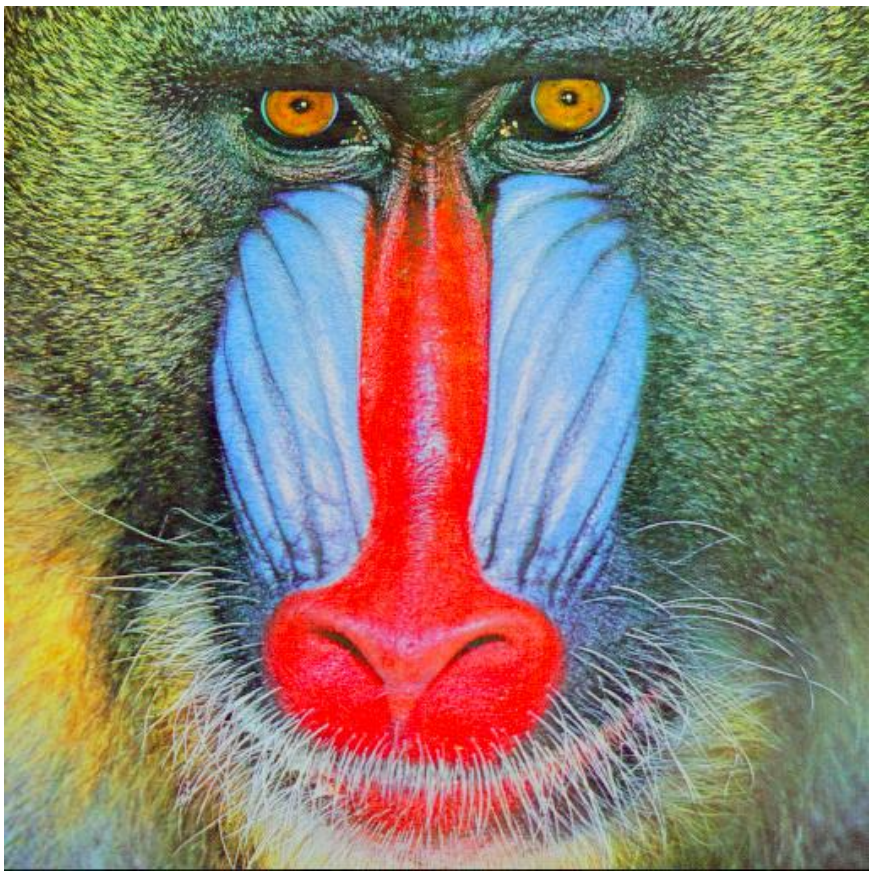
## 1. Display the image in a plot.

First load the JPEG file into ***R*** using the `readJPEG` function:

```
## read in the JPEG File
img <- readJPEG("./figure/mandrill.jpg")
### could also use native format
img.n <- readJPEG("./figure/mandrill.jpg", native = TRUE)
```

Now the image can be plotted by using the `grid.raster` function:

```
## Without the plot/axis
grid.raster(img)
```

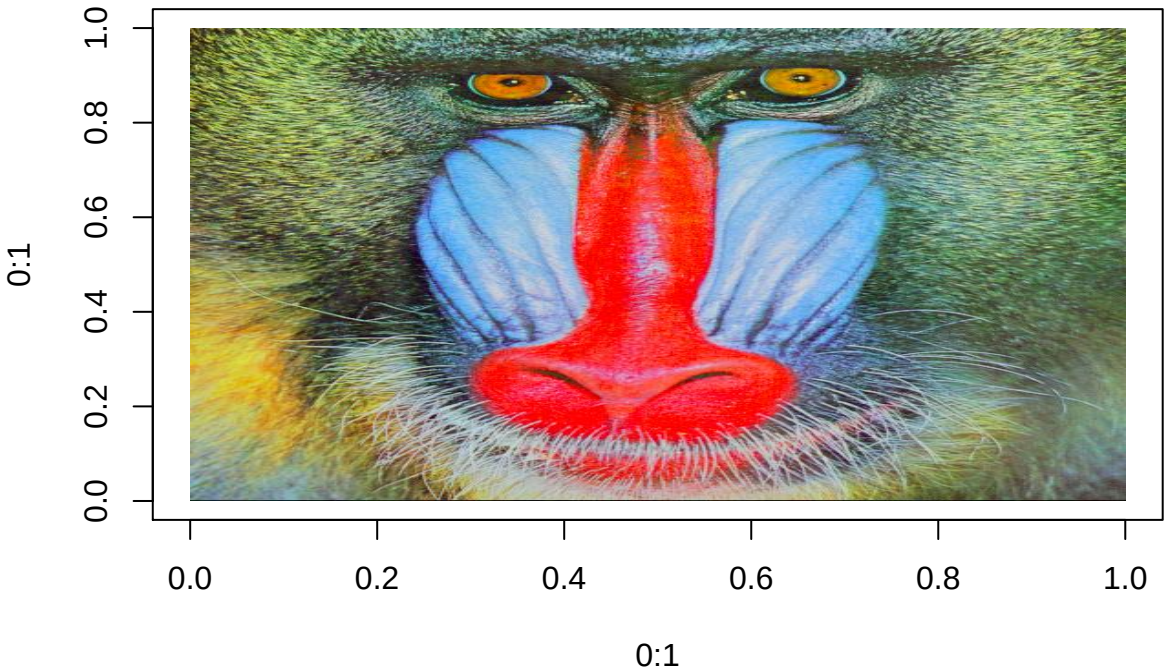


```
## grid.raster(img.n)
```

If however we desired to use a plot window we could first create one using `plot` and instead use the `rasterImage` function:

```
## With the axis
plot(x = 0:1, y = 0:1, type = "n")
```

```r
## grid.raster(img)
rasterImage(img, 0, 0, 1, 1)
```

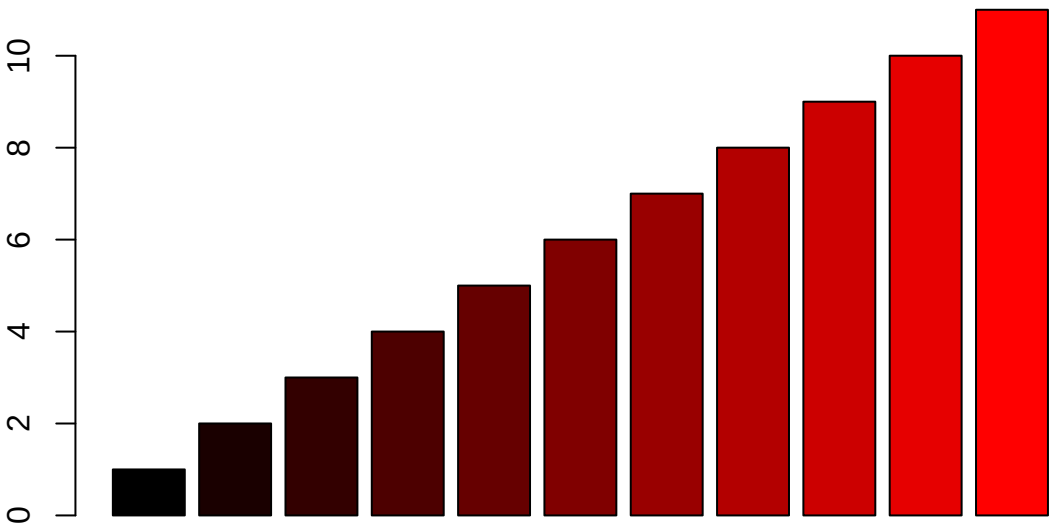



```r
## grid.raster(img.n)
```

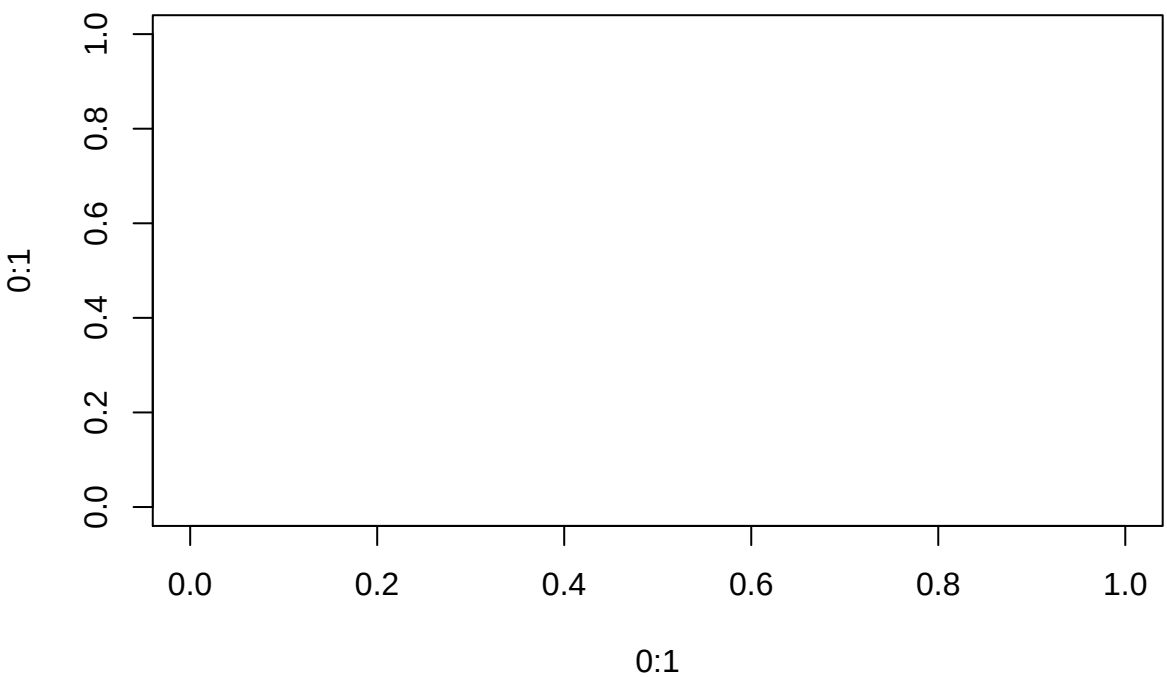
## 2. Plot only the red colour plane.

The JPEG image comprises separate images (or "planes") in the three primary colours, Red, Greenand Blue. Plot only the Red colour plane.

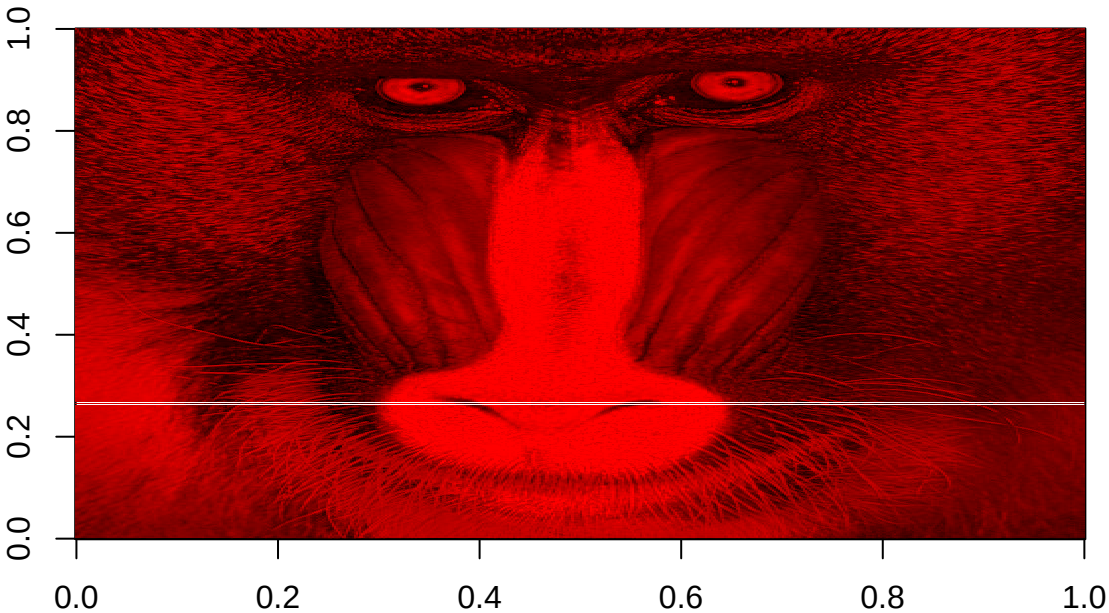
```r
library(RColorBrewer)
reds   <- img[,,1]
  reds <- t(reds[rev(1:nrow(reds)),])
greens <- img[,,2]
  greens <- t(greens[rev(1:nrow(greens)),])
blues  <- img[,,3]
  blues <- t(blues[rev(1:nrow(blues)),])

## Make a Red Pallete
cols <- rgb(red = 0:10, green = 0, blue = 0, maxColorValue = 10)

## Preview the Pallete
barplot(seq(1, length.out = length(cols)), col = cols)
```




```r
# cols <- rgb(1:255, 1:255, 1:255, maxColorValue = 255)

## Make a broader spectrum
cols <- rgb(red = 0:255, green = 0, blue = 0, maxColorValue = 255)

## Plot the red layer using the pallete
plot(x = 0:1, y = 0:1, type = "n")
```

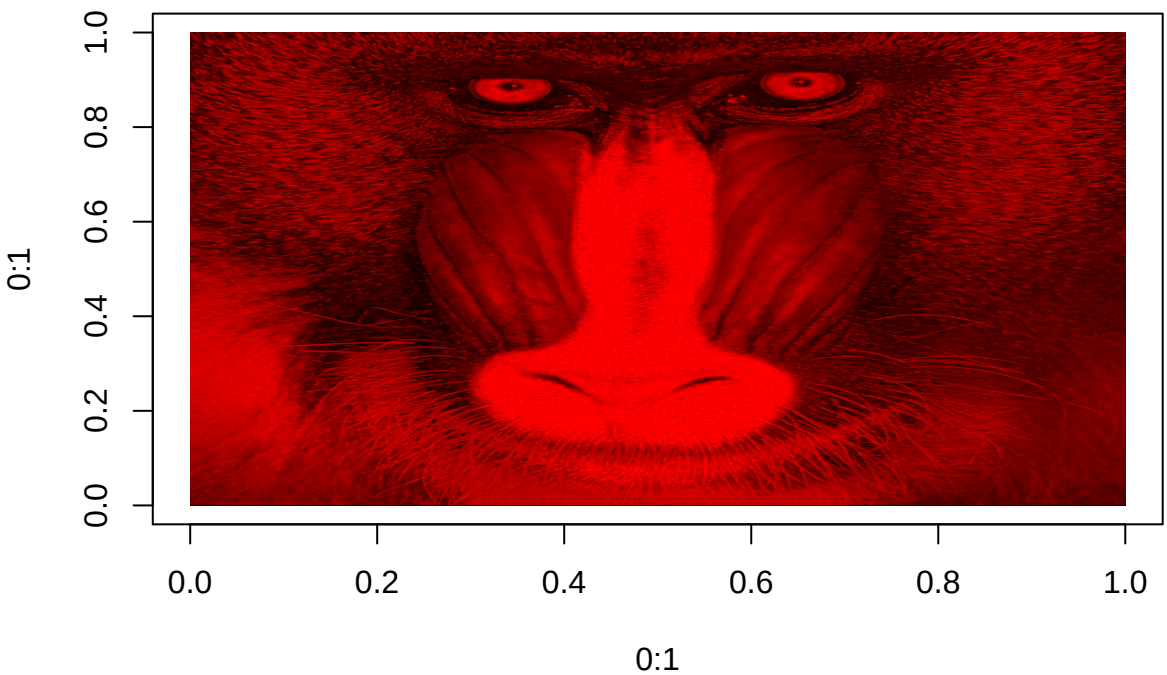



```r
image(reds, col = cols)
```




This unfourtunately produces artifacts when exporting the plot, so a seuperior option is to use `raster.image` and modify the channels green and blue channels to be zero:

```
plot(x = 0:1, y = 0:1, type = "n")
img_r <- img
img_r[,,2:3] <- 0
rasterImage(img_r, 0, 0, 1, 1)
```

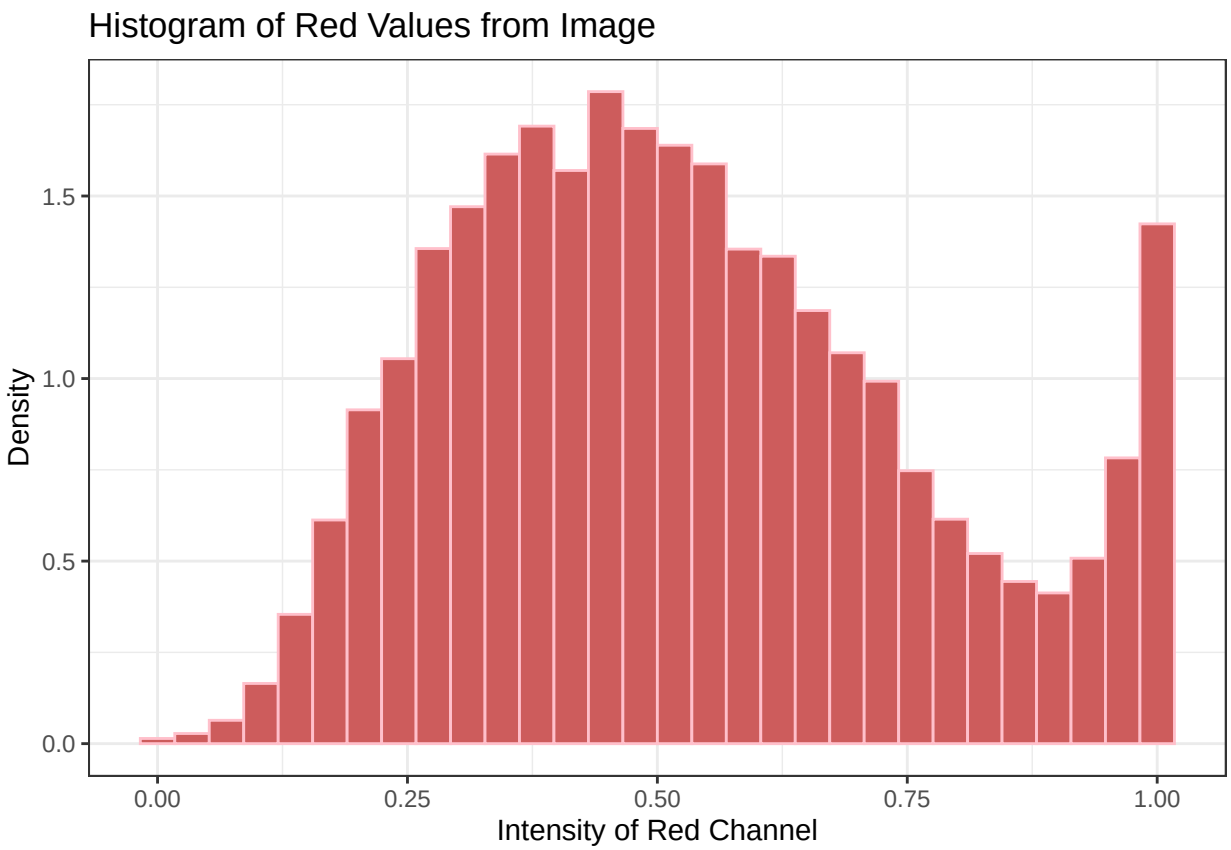



### 3. Plot a histogram of the Red values of all the pixels in the image.

A histogram of the values may be plotted by passing them to `ggplot` (`barplot` or `hist` could also have been used):

```r
## hist(reds)

ggplot(data = data.frame(reds = as.vector(reds)), aes(x = reds, y = ..density..)) +
  geom_histogram(fill = "indianred", col = "pink")+
  theme_bw() +
  labs(x = "Intensity of Red Channel", y = "Density", title = "Histogram of Red Values from Image")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




**4. Plot the red, green and blue pixel values of all the pixels in a horizontal line across the middle of theimage (i.e. all pixels with a vertical index of 256).**

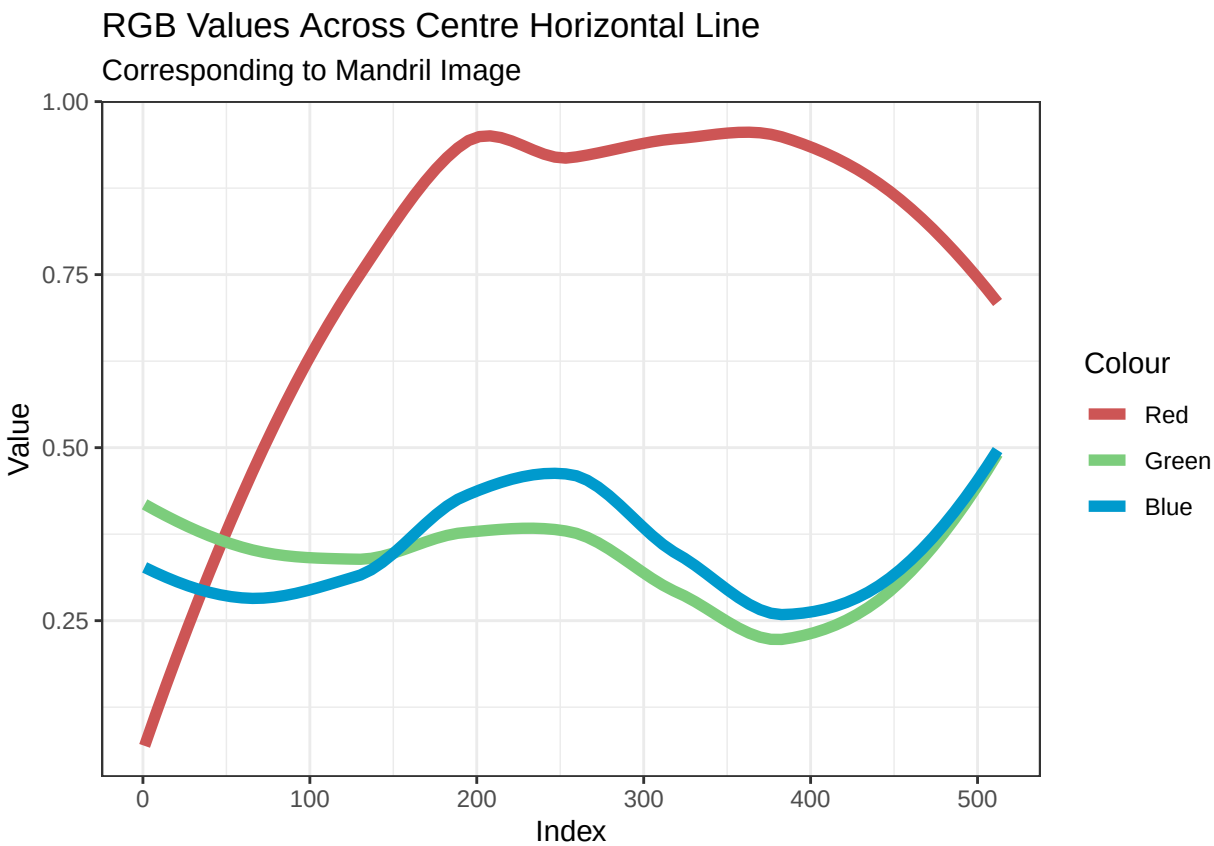
```r
## Extract the values from centre horizontal line
rel_vals <- img[, 256, ]

## Make a Tibble from the DAta
rel_vals_tb <- as_tibble(rel_vals)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have column names if `.name_repair` is omitte
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
names(rel_vals_tb) <- c("Red", "Green", "Blue")
rel_vals_tb$index <- 1:nrow(rel_vals)
```

```r
## Make the Data Frame Tidy
rel_vals_tb_tidy <-
  pivot_longer(data = rel_vals_tb,
               cols = c(Red, Green, Blue),
               names_to = "Colour")

## Preserve the order of RGB
rel_vals_tb_tidy$Colour <-
  factor(
    rel_vals_tb_tidy$Colour,
    levels = c("Red", "Green", "Blue"),
    ordered = TRUE
  )


## Remember that ggplot2 uses alpha
ggplot(data = rel_vals_tb_tidy, aes(x = index, y = value, col = Colour)) +
  geom_smooth(se = FALSE, size = 2) +
  scale_color_manual(values = c("Indianred3", "Palegreen3", "Deepskyblue3")) +
  theme_bw() +
  labs(
    x = "Index",
    y = "Value",
    title = "RGB Values Across Centre Horizontal Line",
    subtitle = "Corresponding to Mandril Image "
  )
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




# Question 4

## a.Imimtate a sequence of Dice Rolls

Try to imitate a sequence of dice rolls. Create a vector of numbers (1, 2, 3, 4, 5, 6), of length 51. Beas random as you can. Do this before you read the rest of the question. (Or, get someone else to doit.) Print your sequence.

```
(my_seq <- c(1,2,2,2,1,2,3,4,1,4,4,5,1,4,6,3,4,5,6,1,1,1,1,2,
             3,4,2,1,3,6,1,3,4,1,3,3,3,5,6,6,6,5,3,2,6,4,3,1,6,1,3))
```

```
##  [1] 1 2 2 2 1 2 3 4 1 4 4 5 1 4 6 3 4 5 6 1 1 1 1 2 3 4 2 1 3 6 1 3 4 1 3 3 3 5
## [39] 6 6 6 5 3 2 6 4 3 1 6 1 3
```

```
length(my_seq)
```

```
## [1] 51
```

## b. Devise a test to real dice-rolling,

Can we devise a test to distinguish between the fake dice-rolling and real dice-rolling, i.e. a trulyrandom process? One method is based on the observation that people tend to put fewer runs ofnumbers in their sequences than you find in random sequences. Write code to count the number ofruns of 1 (i.e. each number that is not followed by the same number), 2 (i.e. 2 consecutive numbersthe same), 3 and 4.

```r
runs <- rle(my_seq)
tab <- table(runs$lengths)

## The question asks for lengths 1 to 4, so to be precice
tab <- table(runs$lengths) %>% append(values = seq(0, 0, length.out = 4))
tab <- tab[1:4]
```

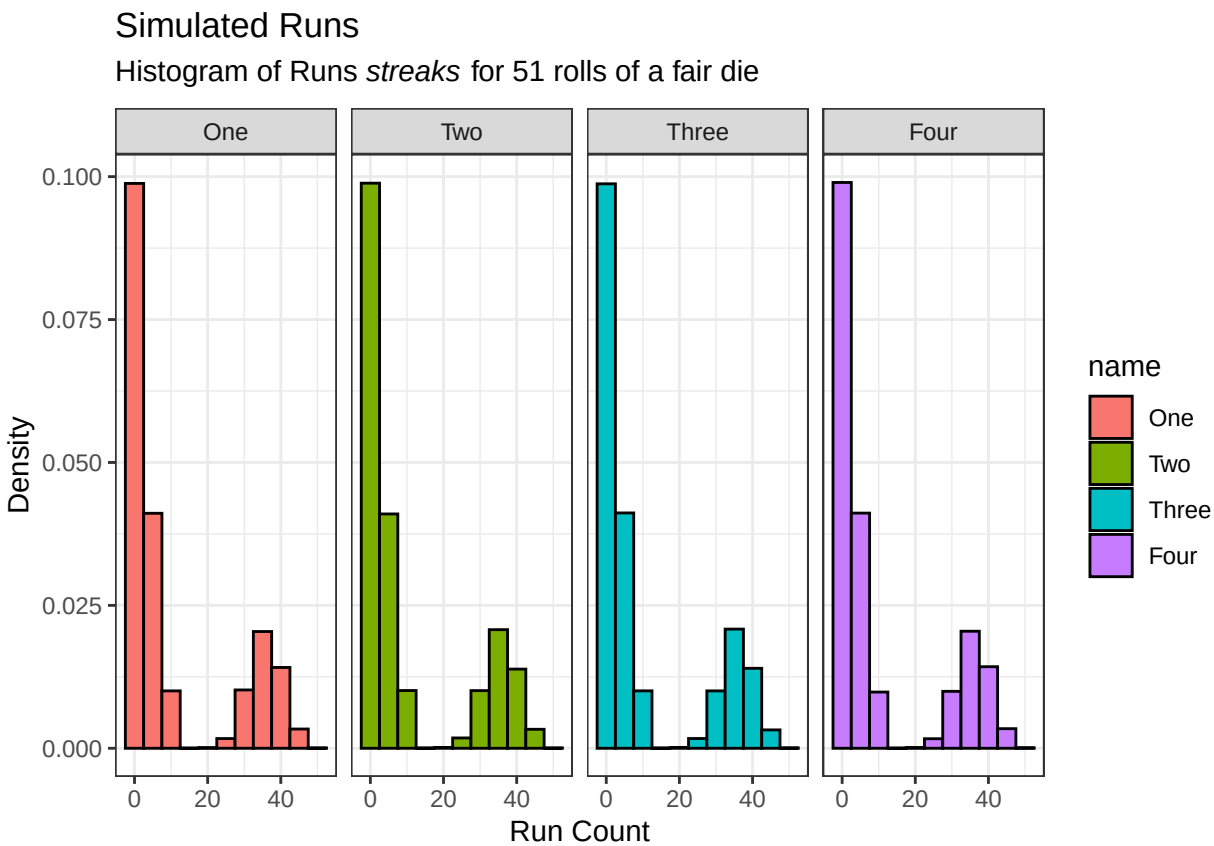
## c.How many runs of each length would we get if the data were truly random?

> How many runs of each length would we get if the data were truly random? Write a loop to simulate105sets of 51 tosses, storing the number of runs of each length (1, 2, 3, 4). Plot the histogram of runs(i.e. total number of runs of each length added up over al

```r
n <- 10^5
sim_seq <- replicate(n, {
  ## Sample a Dice throw
  s_dice <- sample(1:6, size = 51, replace = TRUE)

  ## Count the Runs
  s_runs <- rle(s_dice)

  ## Tabulate the Counts but pad the end with zeroe's so length=4
  s_runs_tb <- table(s_runs$lengths) %>% append(values = seq(0, 0, length.out = 4))

  ## Make it a Matrix
  s_runs_tb[1:4] %>% unlist() %>% matrix(ncol = 4)
}) %>% unlist() %>% matrix(nrow = n, ncol = 4)
```

### Plot the Histogram of Runs

```r
col_names_run_count <- c("One", "Two", "Three", "Four")
sim_seq_tib <- as_tibble(sim_seq)
names(sim_seq_tib) <- col_names_run_count

sim_seq_tib %>%
  pivot_longer(cols = c(One, Two, Three, Four)) %>%
  mutate(name = factor(name, labels = col_names_run_count), ordered = TRUE) %>%
  ggplot(aes(fill = name, x = value, y = ..density..)) +
    geom_histogram(binwidth = 5, col = "black") +
    facet_grid(. ~ name) +
    theme_bw() +
    labs(x = "Run Count",
         title = "Simulated Runs",
         subtitle = TeX("Histogram of Runs \\textit{streaks} for 51 rolls of a fair die"),
         y = "Density")
```




**Print the Totals**

The total number of runs that occured over the simulations can be returned by using `apply`:

```
apply(sim_seq_tib, 2, sum)
```

```
##     One     Two   Three    Four
## 1066271 1067592 1066568 1066918
```

It may however be more instructive to inspect the average number of runs over the simulations compared to the human generated sequence:

```
data.frame("Simulated" = apply(sim_seq_tib, 2, mean),
           "Human" = as.vector(tab)) %>%
  kable()
```

| | Simulated | Human |
|---|---|---|
| One | 10.66271 | 36 |
| Two | 10.67592 | 1 |
| Three | 10.66568 | 3 |
| Four | 10.66918 | 1 |