# Image Exercise

*Dasapta Erwin Irawan, Willem Vervoort and Gene Melzack*

*Today's date is 01-02-2018*

## Introduction

To demonstrate the overall process of collecting data, creating reproducible research, and publishing data we have designed an exercise that will allow you to plot the results of the vegetation images that we have collected during our small field excursion. While the current data are arbitrary, they contain several aspects that are useful to demonstrate steps in the open research data process. R markdown is a nice way to actually present your reproducible research, as it allows you to record both the code and the text.

In this exercise, we will do three separate components:

1. Learn how to plot spatial data and convert from latitude and longitude to eastings and northings
2. Understand how to extract the different colour channels from photos and calculate GCC following the paper by Moore et al (2017)
3. Plot the GCC values in space to make a map of the collected data.

Here we will use some example data and do some exercises, after which we can apply this to the field data we will collect on Thursday.

## Plotting spatial data

Spatial data is generally in latitudes and longitudes. This can be both in decimal form, such as in Google maps, or in degree form (which is much of the older data). As latitudes and longitudes are related to the globe, to calculate actual distances and to display correct spatial relationships, the data needs to be projected (or flattened). For projections we need to use the right coordinate system. If anyone of you has worked with GIS, then this is old news.

Here we first use some example data from Australia:

- This file AusLatLongData.txt is a simple comma delimited text file that has decimal latitudes and longitudes in and around Sydney.

We first need to load the data and we can assign a spatial coordinate system to it that indicates that it is in latitude and longitudes. We can then plot the data in these coordinates.

We also need a few libraries: rgdal, sp

```
# library
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------ tidyverse 1.2.1 --
```

```
## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.3.4     v dplyr   0.7.4
## v tidyr   0.7.2     v stringr 1.2.0
## v readr   1.1.1     v forcats 0.2.0
```

```
## -- Conflicts --------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(rgdal)
```

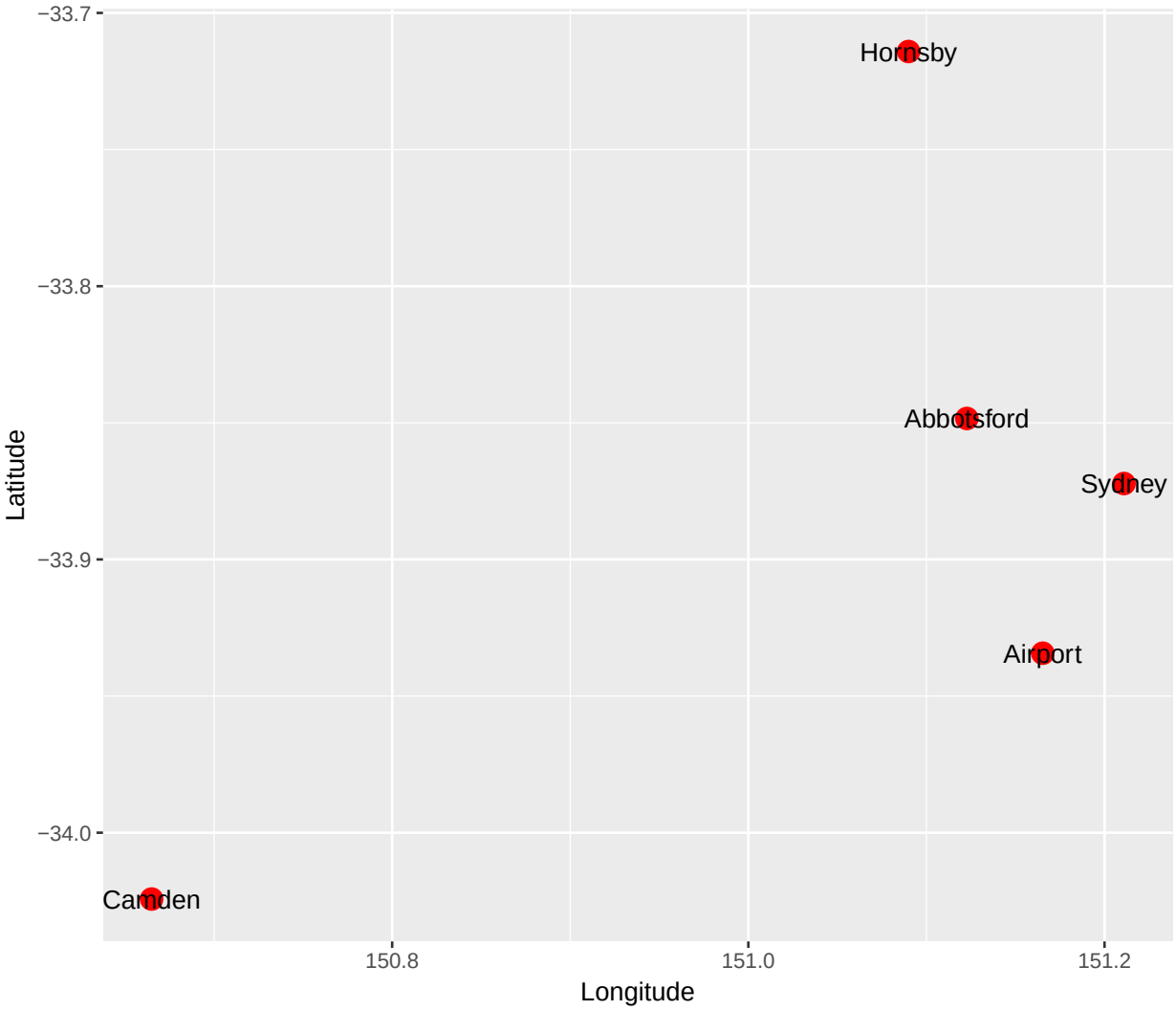
```
## Loading required package: sp
```

```
## rgdal: version: 1.2-16, (SVN revision 701)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.0, released 2017/04/28
##  Path to GDAL shared files: C:/Users/rver4657/Documents/R/win-library/3.4/rgdal/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/Users/rver4657/Documents/R/win-library/3.4/rgdal/proj
##  Linking to sp version: 1.2-5
```

```r
library(sp)

# root dir
root <- "C:/Users/rver4657/Google Drive/ITB_Usyd_Project2017/workshop_opendata/OriginalDataFolder"
# read in the data
Sydney <- read.csv(paste(root,"AusLatLongData.txt",sep="/"))

# create Spatial points data frame using package sp and insert coordinate reference
coord_ref <- CRS("+proj=longlat +datum=WGS84")
Sydney_sp <- SpatialPoints(coords=Sydney[,2:3], proj4string = coord_ref)
# we can plot this
p <- ggplot(aes(x = Longitude, y = Latitude), data = as.data.frame(Sydney_sp)) + geom_point(col="red", s
  geom_text(aes(label = Sydney[,1]))
print(p)
```

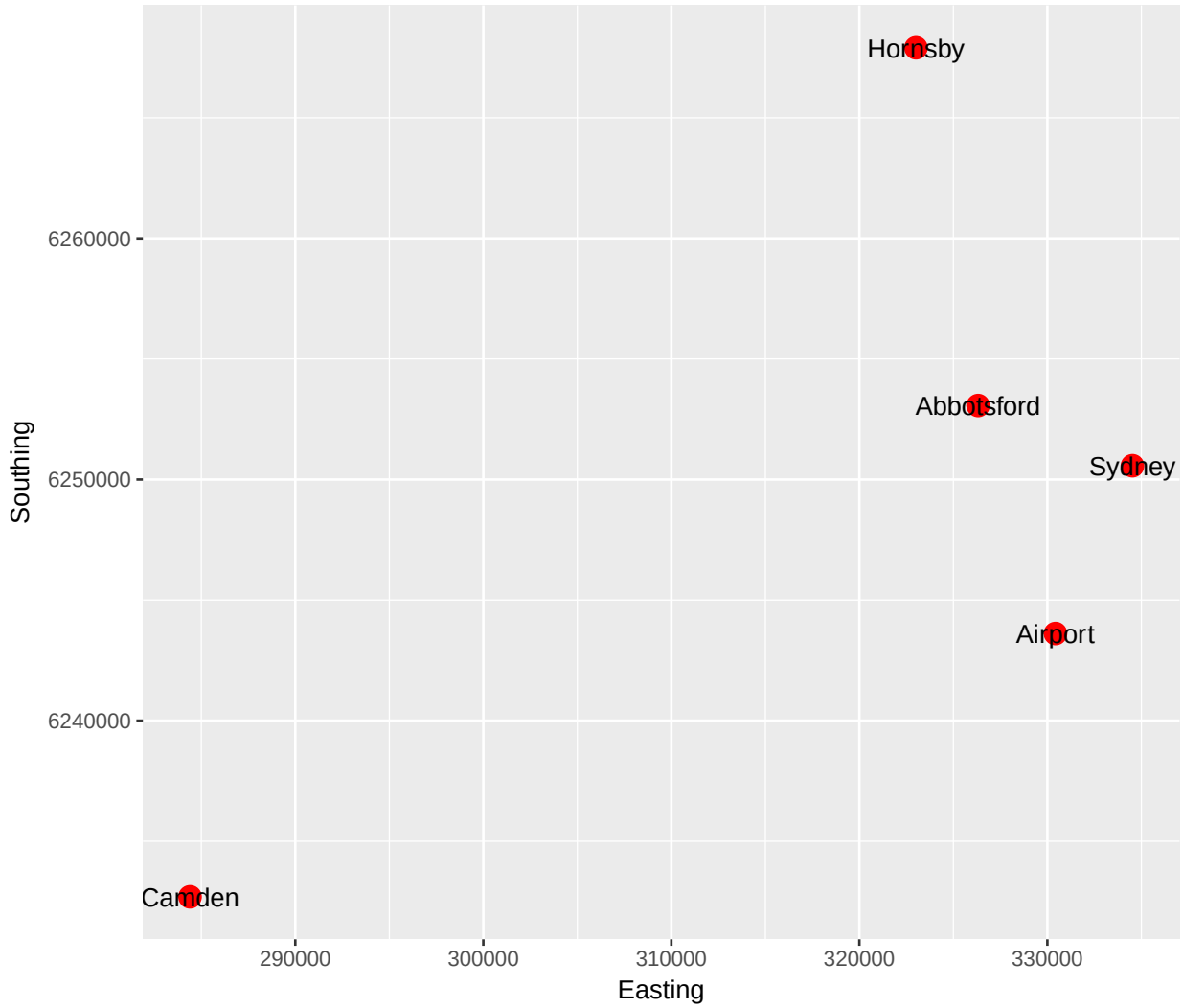



However, this does not reflect the real distances, for this we would like to convert the latitudes and longitudes into eastings and northings. This is a projection and we need to use the correct projection for the Sydney area. For this we simply use the correct epsg code using the spatial reference system, which for Sydney is zone 56 or epsg code 28356.

We can now simply calculate the distance between Abbotsford and the airport in m.

```
# reproject
Sydney_pr <- spTransform(Sydney_sp, CRS("+init=epsg:28356")) # reproject

# plot again
p <- ggplot(aes(x = Longitude, y = Latitude), data = as.data.frame(Sydney_pr)) + geom_point(col="red", s
  geom_text(aes(label = Sydney[,1])) +
  xlab("Easting") + ylab("Southing")
print(p)
```




```
# distance Abbotsford (2nd row) to Airport (3rd row)
Dist <- sqrt((Sydney_pr@coords[2,1]-Sydney_pr@coords[3,1])^2 +
             (Sydney_pr@coords[2,2]-Sydney_pr@coords[3,2])^2)
paste("the distance between Abbotsford and Sydney airpoirt =",round(Dist,2), "m")
```

```
## [1] "the distance between Abbotsford and Sydney airpoirt = 10310.7 m"
```

## Exercise

- Repeat the above analysis using the Bandung data that is also on the drive. Note that this data does not have any named labels, just row numbers.

## Image analysis

In the field excursion we collected some images, which we downloaded into a shared directory. Here we will do a simple analysis to extract the red and blue bands and calulate the average ratio of the red and blue bands, because we don't actually have access to any Near-Infrared to mimic NDVI.

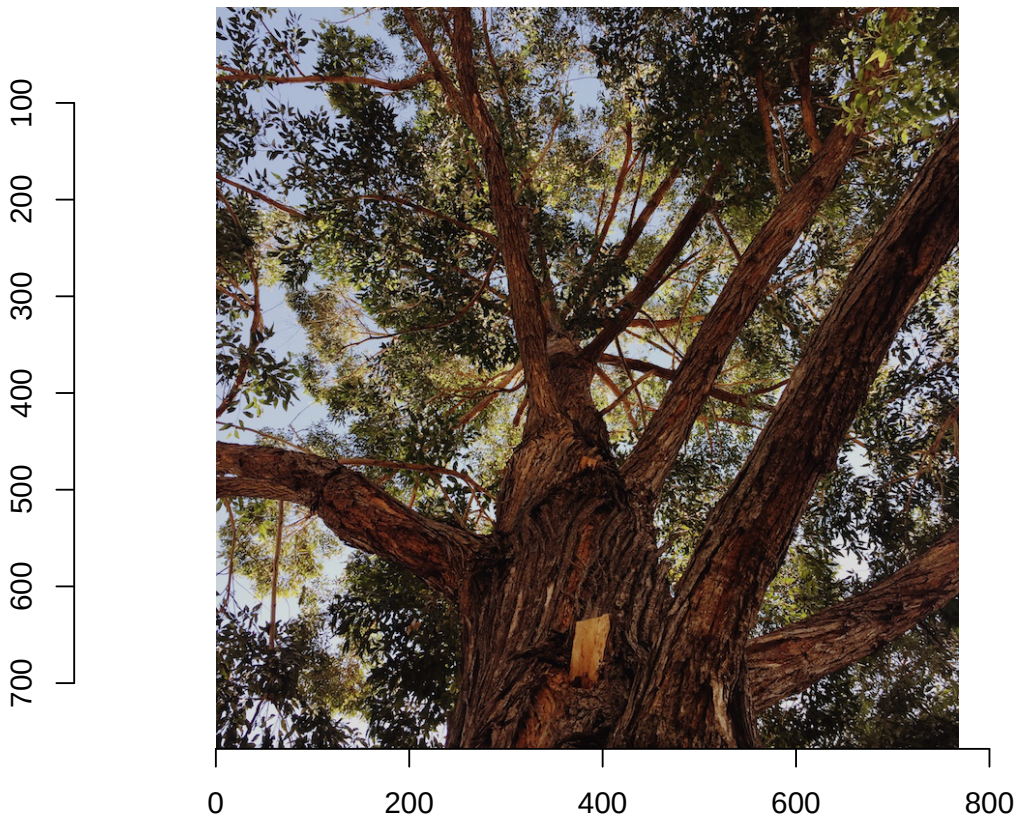



Figure 1: A demonstration picture

The first thing we need to do is to install the package `imager`, or just require if you have this already installed. We will also use the package `tidyverse` which we have seen before.

```
# not installed? run
# install.packages("imager")
# otherwise
library(imager)
library(tidyverse)
```

The next step is to load an image and to convert this to a data.frame

```
our_image <- load.image("imageanalysis/5.jpg")
plot(our_image)
```

```
image_df <- as.data.frame(our_image)
head(image_df,3)
```

```
##   x y cc     value
```

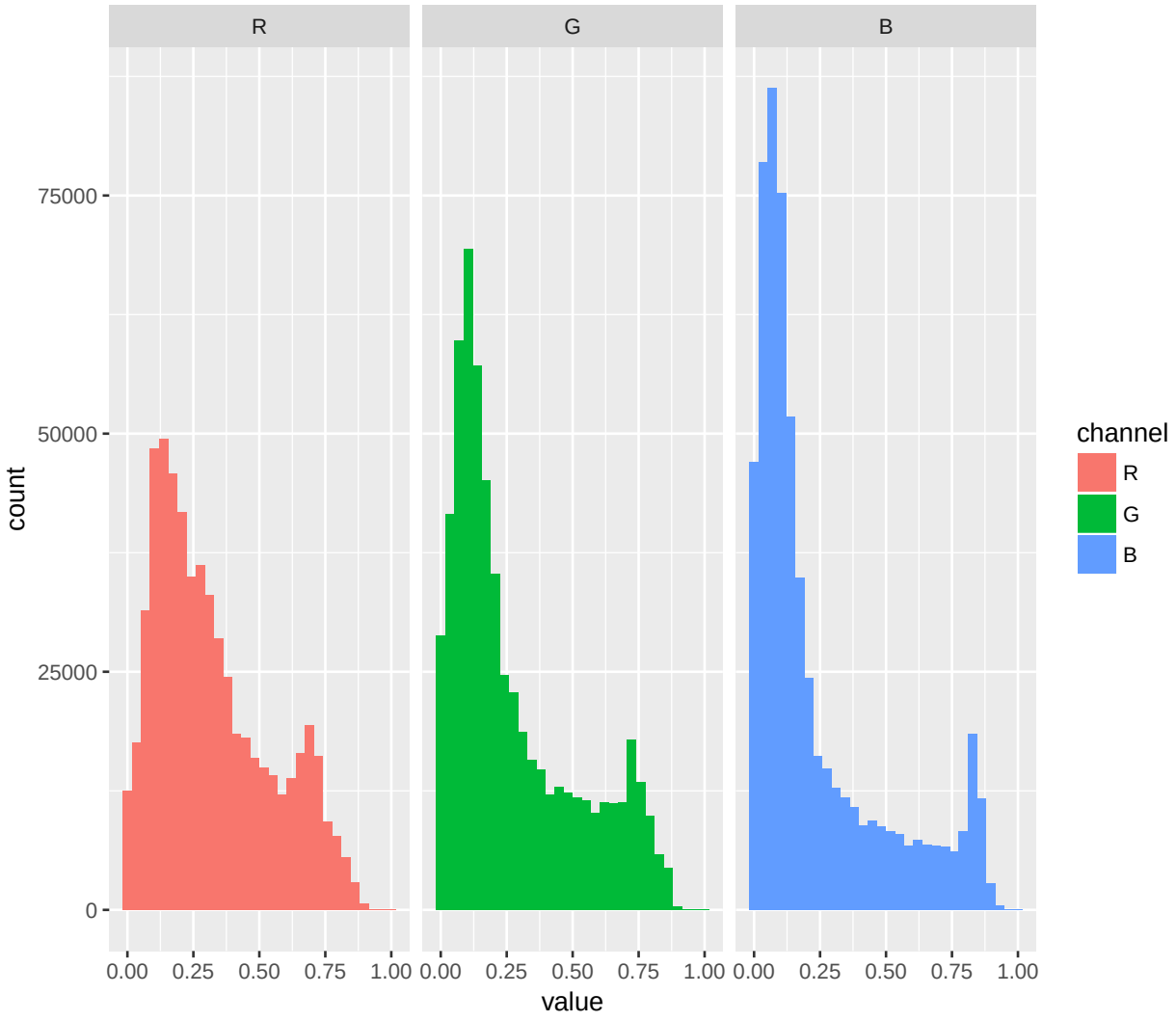



Figure 2: histograms of the different channels

```
## 1 1 1  1 0.6588235
## 2 2 1  1 0.6588235
## 3 3 1  1 0.6549020
```

As we can see, the data consists of the pixel x and y coordinate and a numerical value to indicate the R (red), G (green), and B (blue) channel. The data.frame is *stacked*. We can quickly plot the distributions of the different channels using `ggplot()`, and splitting the plot by channel. The first line of code assigns 'R', 'G' and 'B' to the numerical values for the channels. The next lines of code creates the histograms.

```
image_df <- mutate(image_df,channel=factor(cc,labels=c('R','G','B')))
ggplot(image_df,aes(value,fill=channel))+
  geom_histogram(bins=30)+
  facet_wrap(~ channel)
```

## Creating the average R over R + B ratio

The next step is to calculate the R/(R + B) values, which as outlined represent vegetation growth (Moore et al. 2017). For this we first need to *spread* the data.frame to create separate columns for R, G and B, and in the mean time dropping the column cc using `mutate()`.

```
image_df2 <- image_df %>%
  mutate(cc = NULL) %>%
  spread(key = channel, value = value)
head(image_df2)
```

```
##   x y         R         G         B
## 1 1 1 0.6588235 0.7215686 0.8196078
## 2 1 2 0.6588235 0.7215686 0.8235294
## 3 1 3 0.6627451 0.7215686 0.8352941
## 4 1 4 0.6588235 0.7176471 0.8313725
## 5 1 5 0.6588235 0.7137255 0.8274510
## 6 1 6 0.6549020 0.7098039 0.8235294
```

Now it is simple to calculate the ratio using `mutate()` once again, plot a simple histogram of this data and finally calculate the mean value for the image.

```
image_df3 <- image_df2 %>%
  mutate(ratio = R/(R+B))

pl <- ggplot(image_df3,aes(ratio)) +
  geom_histogram(bins=50) + xlab("R/(R + B) ratio")
print(pl)
```

```
## Warning: Removed 1795 rows containing non-finite values (stat_bin).
```

```
# calculate the mean
avg_ratio <- mean(image_df3$ratio, na.rm=T)
round(avg_ratio,2)
```

```
## [1] 0.65
```

## extracting the latitude and longitude data and plotting

A nice feature of modern digital photography is that it directly comes with a lot of metadata including the latitude and longitude of where the photo was taken, the metadata on the images is called the **exif data**. There is a package in R that does this quite nicely, which is called `exifr`. There is additional package that allows plotting onto google map images. This is called `leaflet`.

There is one small problem with `exifr` on Windows. The package assumes that you have the **perl** computer scripting language (similar to R) installed on your computer. This is the case on all Apple computers, but not on Windows. So on windows you have to first install Perl from:http://strawberryperl.com/. Then you can installl `leaflet` and `exifr`.

```
#install.packages(c("exifr", "leaflet"))
# you have to install Perl on windows, as this is not included, it is included on Apple
# then on windows point exifr to perl dir
#options(exifr.perlpath='c:/strawberry')
# then load the libraries into your workspace
library(exifr)
```

```
## Using ExifTool version 10.61
```

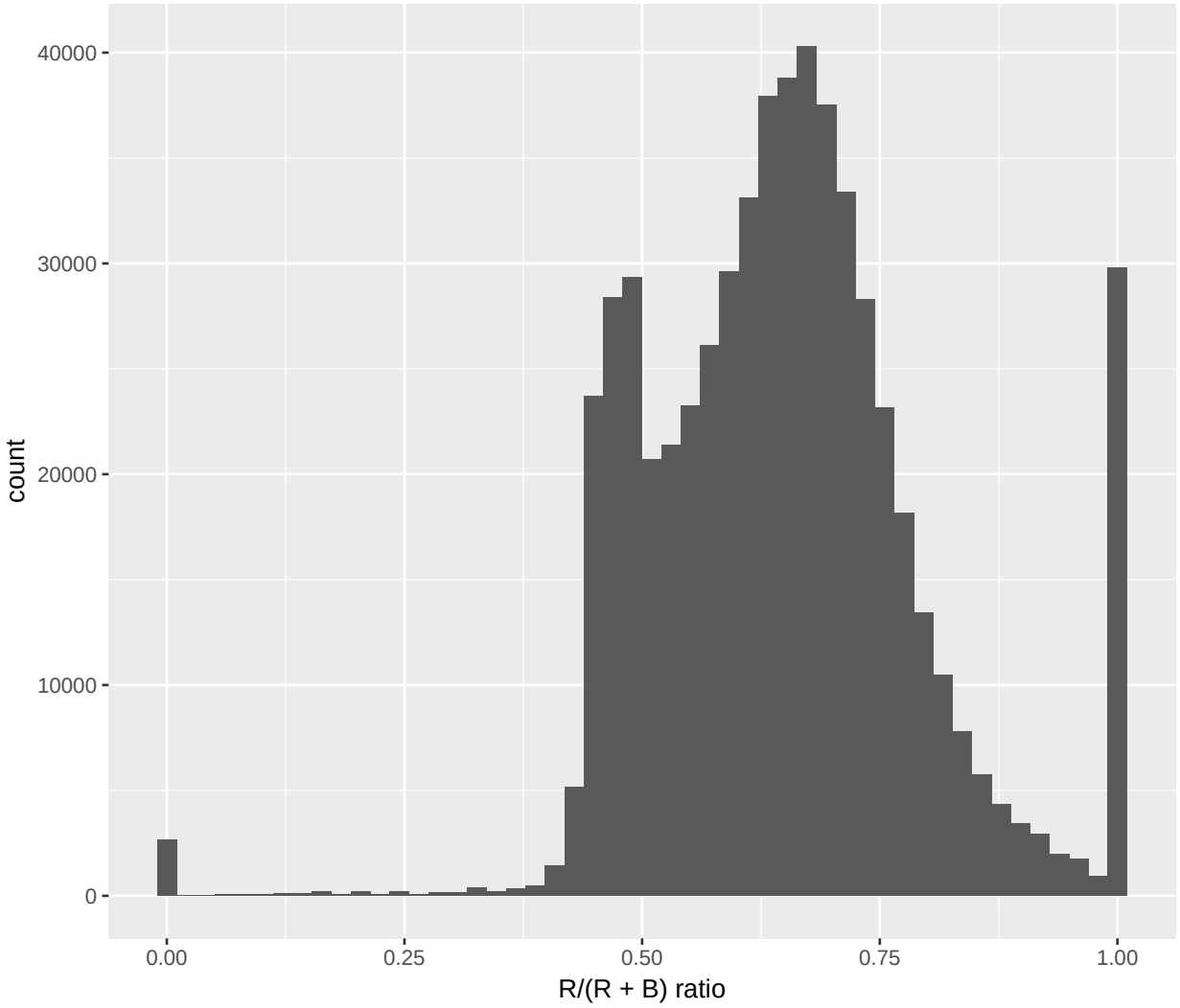



Figure 3: histogram of the R/(R + B) ratio

```
library(leaflet)
```

We can now read all the jpg files from a directory, and extract the information

```
files <- list.files(path = "imageanalysis/vegetation_images",
                    pattern = "*.jpg")
dat <- read_exif(path = paste("imageanalysis/vegetation_images/",
                              files,sep=""))
names(dat)
```

```
##   [1] "SourceFile"               "ExifToolVersion"
##   [3] "FileName"                 "Directory"
##   [5] "FileSize"                 "FileModifyDate"
##   [7] "FileAccessDate"           "FileCreateDate"
##   [9] "FilePermissions"          "FileType"
##  [11] "FileTypeExtension"        "MIMEType"
##  [13] "JFIFVersion"              "ExifByteOrder"
##  [15] "Make"                     "Model"
##  [17] "Orientation"              "XResolution"
##  [19] "YResolution"              "ResolutionUnit"
##  [21] "Software"                 "ModifyDate"
##  [23] "ExposureTime"             "FNumber"
##  [25] "ExposureProgram"          "ISO"
##  [27] "ExifVersion"              "DateTimeOriginal"
##  [29] "CreateDate"               "ComponentsConfiguration"
##  [31] "ShutterSpeedValue"        "ApertureValue"
##  [33] "BrightnessValue"          "ExposureCompensation"
##  [35] "MeteringMode"             "Flash"
##  [37] "FocalLength"              "SubjectArea"
##  [39] "RunTimeFlags"             "RunTimeValue"
##  [41] "RunTimeScale"             "RunTimeEpoch"
##  [43] "AccelerationVector"       "HDRImageType"
##  [45] "ImageUniqueID"            "SubSecTimeOriginal"
##  [47] "SubSecTimeDigitized"      "FlashpixVersion"
##  [49] "ColorSpace"               "ExifImageWidth"
##  [51] "ExifImageHeight"          "SensingMethod"
##  [53] "SceneType"                "CustomRendered"
##  [55] "ExposureMode"             "WhiteBalance"
##  [57] "FocalLengthIn35mmFormat"  "SceneCaptureType"
##  [59] "LensInfo"                 "LensMake"
##  [61] "LensModel"                "GPSLatitudeRef"
##  [63] "GPSLongitudeRef"          "GPSAltitudeRef"
##  [65] "GPSTimeStamp"             "GPSSpeedRef"
##  [67] "GPSSpeed"                 "GPSImgDirectionRef"
##  [69] "GPSImgDirection"          "GPSDestBearingRef"
##  [71] "GPSDestBearing"           "GPSDateStamp"
##  [73] "GPSHPositioningError"     "XMPToolkit"
##  [75] "CreatorTool"              "CurrentIPTCDigest"
##  [77] "CodedCharacterSet"        "ApplicationRecordVersion"
##  [79] "DigitalCreationTime"      "DigitalCreationDate"
##  [81] "DateCreated"              "TimeCreated"
##  [83] "IPTCDigest"               "ImageWidth"
##  [85] "ImageHeight"              "EncodingProcess"
##  [87] "BitsPerSample"            "ColorComponents"
```

```
##  [89] "YCbCrSubSampling"         "Aperture"
##  [91] "DateTimeCreated"          "DigitalCreationDateTime"
##  [93] "GPSAltitude"              "GPSDateTime"
##  [95] "GPSLatitude"              "GPSLongitude"
##  [97] "GPSPosition"              "ImageSize"
##  [99] "Megapixels"               "RunTimeSincePowerUp"
## [101] "ScaleFactor35efl"         "ShutterSpeed"
## [103] "SubSecCreateDate"         "SubSecDateTimeOriginal"
## [105] "CircleOfConfusion"        "FOV"
## [107] "FocalLength35efl"         "HyperfocalDistance"
## [109] "LightValue"               "ImageDescription"
## [111] "YCbCrPositioning"         "LightSource"
## [113] "Warning"                  "SubSecTime"
## [115] "InteropIndex"             "InteropVersion"
## [117] "DigitalZoomRatio"         "Contrast"
## [119] "Saturation"               "Sharpness"
## [121] "GPSVersionID"             "GPSProcessingMethod"
## [123] "Compression"              "ThumbnailOffset"
## [125] "ThumbnailLength"          "SubSecModifyDate"
## [127] "ThumbnailImage"
```

There is a lot of metadata with each photo and we can explore all of this, but we are only interested in the Latitude and Longitude. First we can make a simple plot of Longitude and Latitude.

```
plot(dat$GPSLongitude, dat$GPSLatitude)
```

## Calculate GCC values and plot

The final step is to calculate the GCC values for all the images and plot these on the map. We will do this by running a loop over files, and reusing the code from above inside the loop (but dropping all the plotting and looking at the files)

```
Store <- rep(0,length(files))

for (i in 1:length(files)) {
  our_image <- load.image(paste("imageanalysis/vegetation_images/",
                                files[i],sep=""))

  image_df <- as.data.frame(our_image)
  image_df <- mutate(image_df,channel=factor(cc,labels=c('R','G','B')))
  # spread
  image_df2 <- image_df %>%
      mutate(cc = NULL) %>%
      spread(key = channel, value = value)
  # mutate
  image_df3 <- image_df2 %>%
    mutate(ratio = R/(R+B))
  # calculate the mean
  avg_ratio <- mean(image_df3$ratio, na.rm=T)
  # Store
  Store[i] <-round(avg_ratio,2)
}

plotdat <- data.frame(Latitude = dat$GPSLatitude,
```

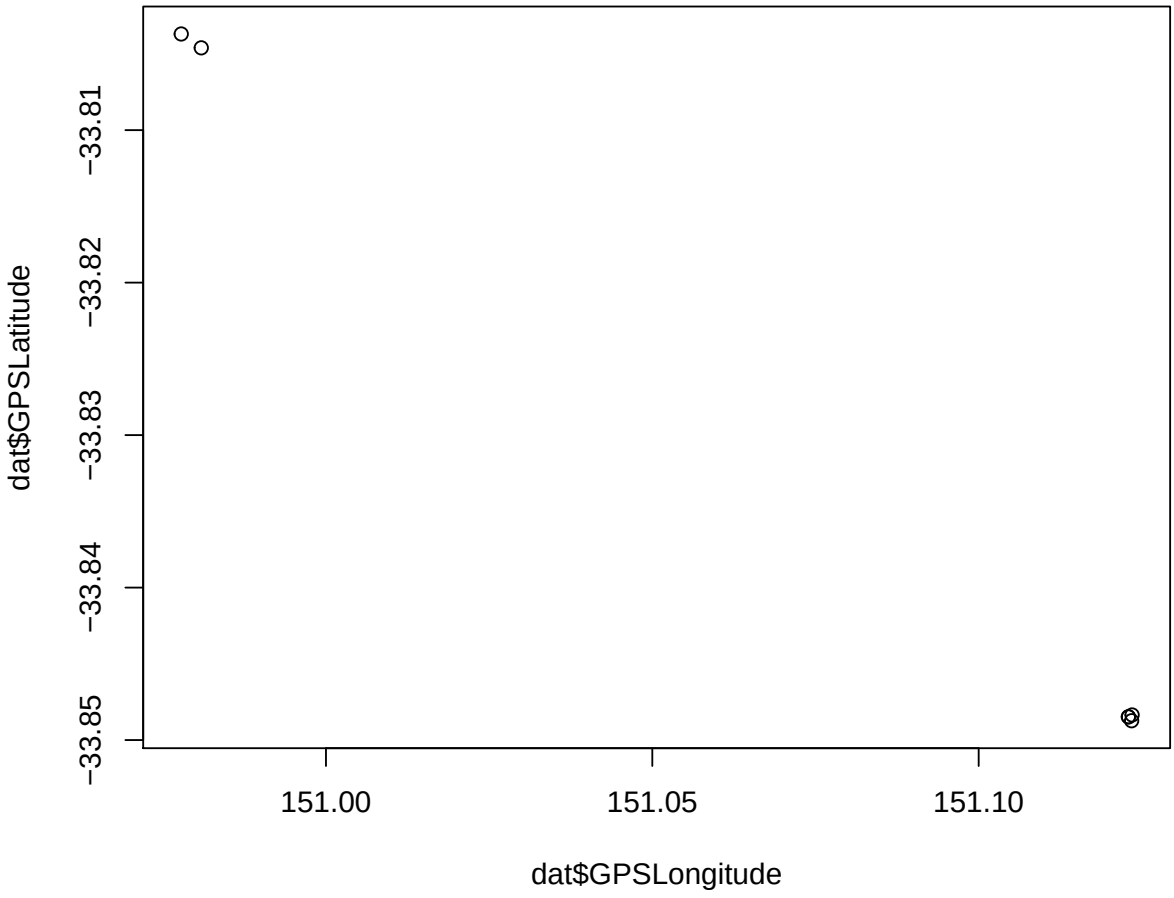



Figure 4: plotting the latitude and longitude of the metadata

```
                    Longitude = dat$GPSLongitude,
                    GCC <- Store)


# now plot values of Store
p <- ggplot(plotdat, aes(x=Longitude, y = Latitude)) +
  geom_point(aes(colour=GCC, size=GCC))
p
```