Image Exercise

Dasapta Erwin Irawan & Willem Vervoort

Today's date is 02-01-2018

Introduction

To demonstrate the overall process of collecting data, creating reproducible research, and publishing data we have designed an exercise using simple 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.

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.

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.

```r
# 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

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

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

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

```r
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)
```

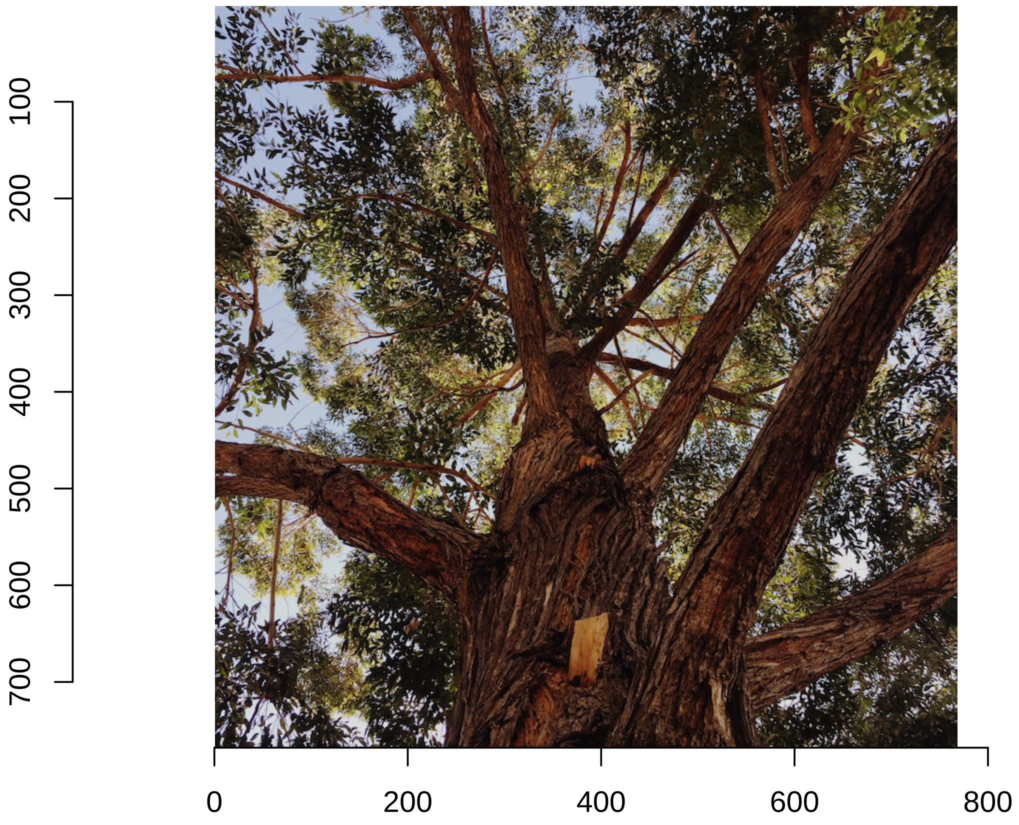

Figure 1: A demonstration picture

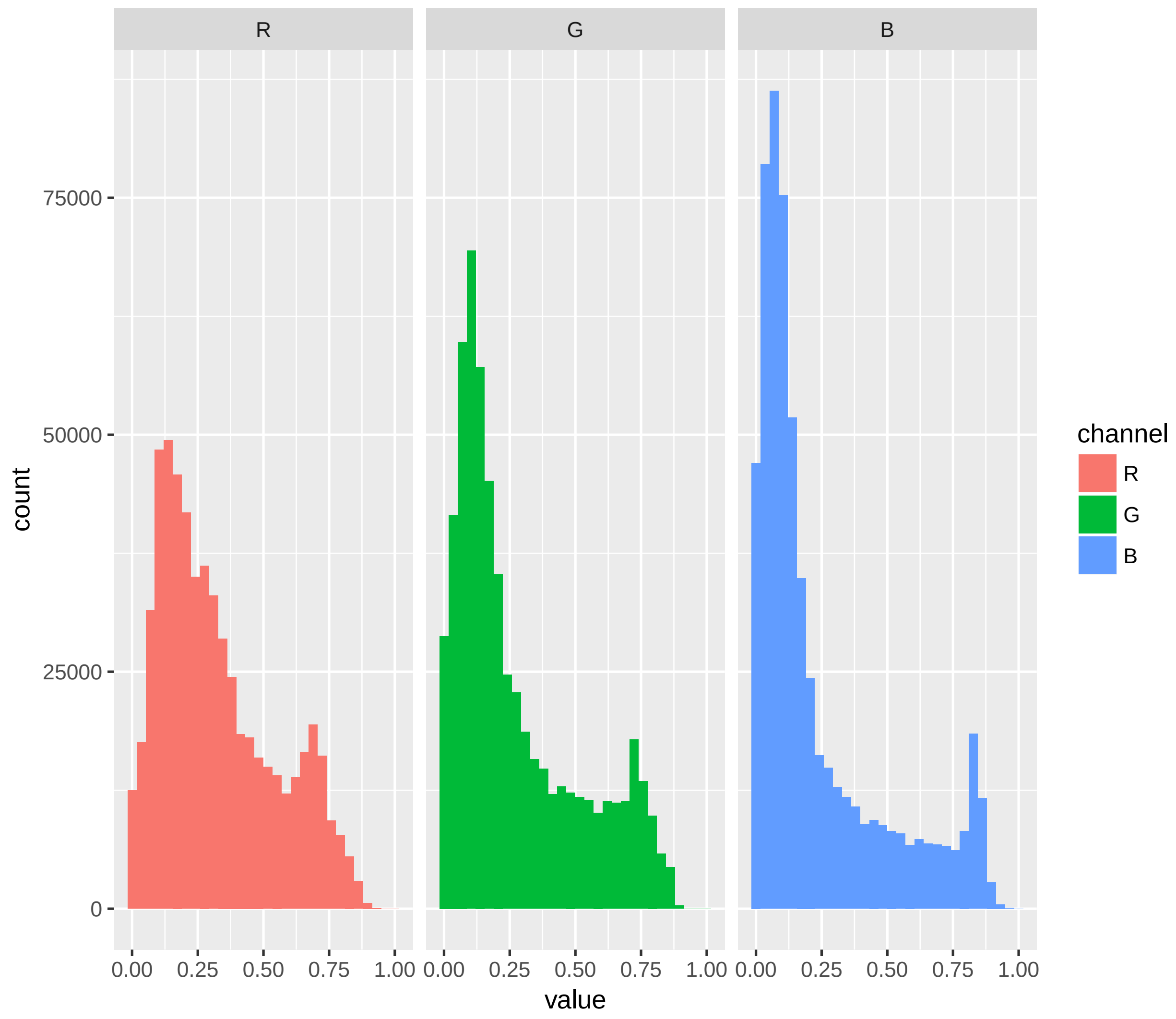

Figure 2: histograms of the different channels

Creating the average R over R + B ratio

The next step is to calculate the R/(R + B) values. 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
```

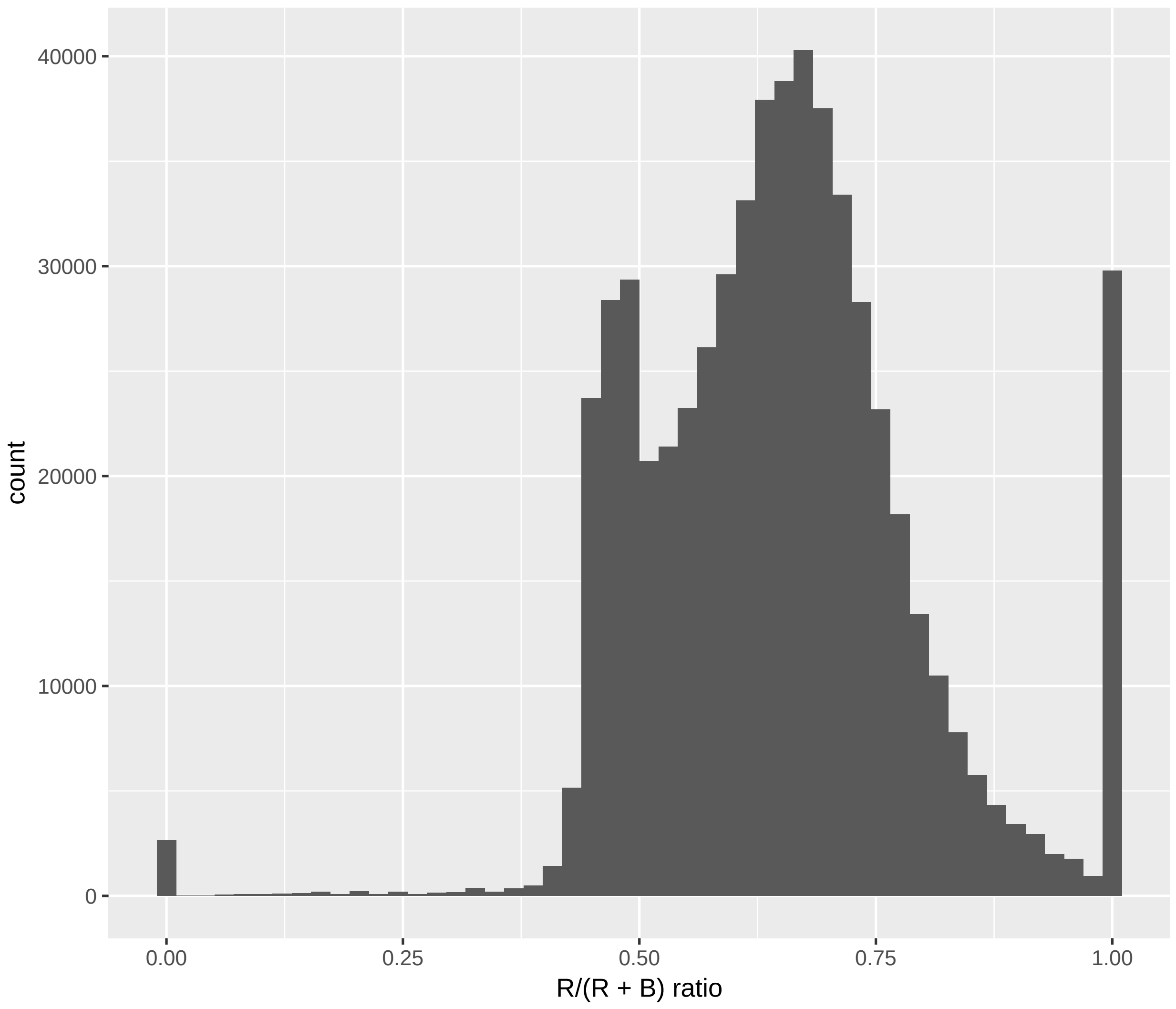

Figure 3: histogram of the R/(R + B) ratio

```r
library(leaflet)
```

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

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

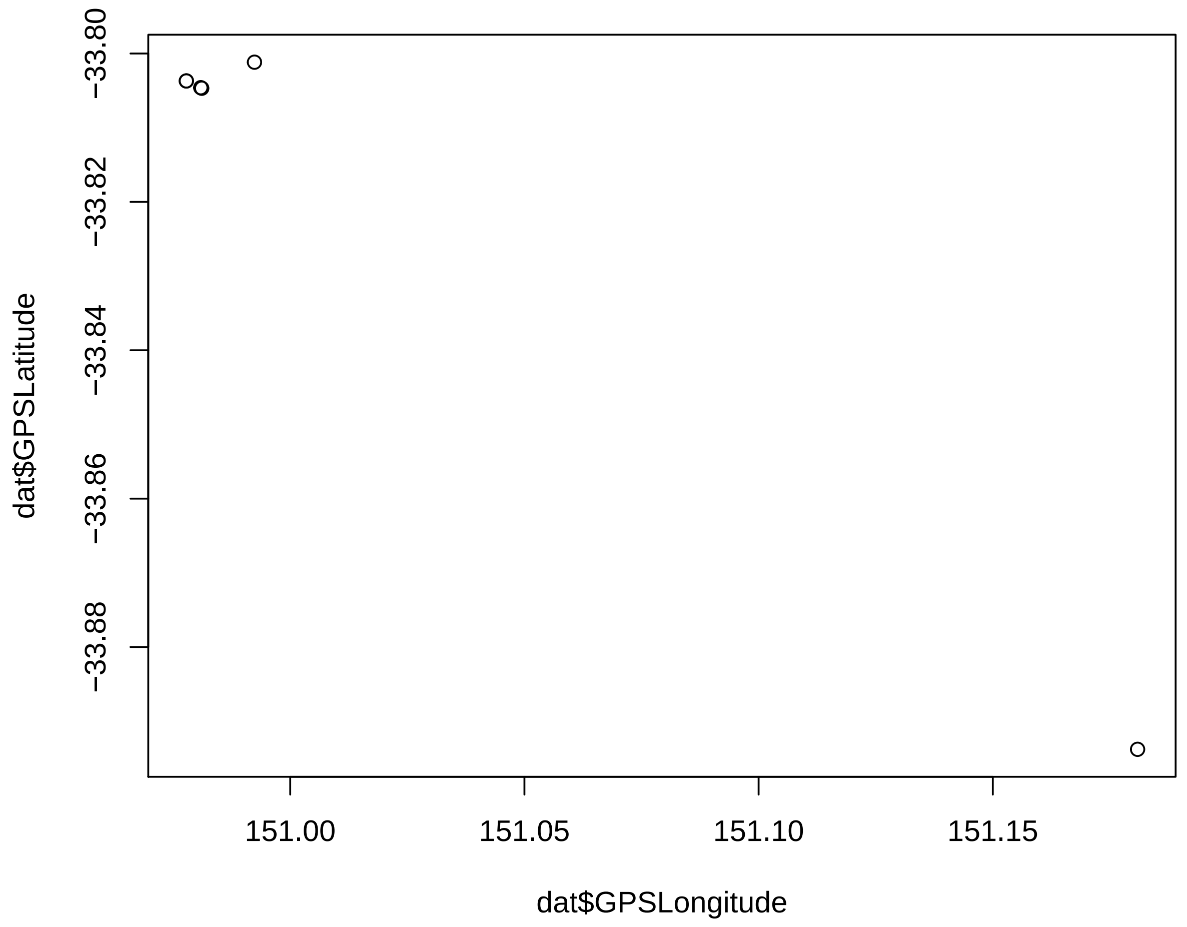

Figure 4: plotting the latitude and longitude of the metadata

```
##  [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"
```

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)
```

Or, nicer plot them onto some google imagery.

```r
# not run
leaflet(dat) %>%
  addProviderTiles("Esri.WorldImagery") %>%
  addMarkers(~ GPSLongitude, ~ GPSLatitude)
```