Wrangling rasters: from satellite image to dataframe format in R

Jacinta Holloway

12 March 2019

The purpose of this document is to provide an example of how to extract data from a large image file to then fit statistical models and do your own analysis. Back when I first started working on these types of problems, I couldn't find clear answers to what I was asking online. I found my way through trial and plenty of error, and am sharing this so that it might answer the same questions for someone else.

This code imports Landsat imagery of Injune in Queensland, Australia into R in raster format and prepares the data for statistical analyses.

Note: I loaded the satellite image, example_raster.img, which is 493MB to github using Git large file storage (Git-lfs) which I set up on my local machine. For details on the Git-lfs see https://git-lfs.github.com/

Now, to the code. Require packages rgdal, sp, raster, rgeos.

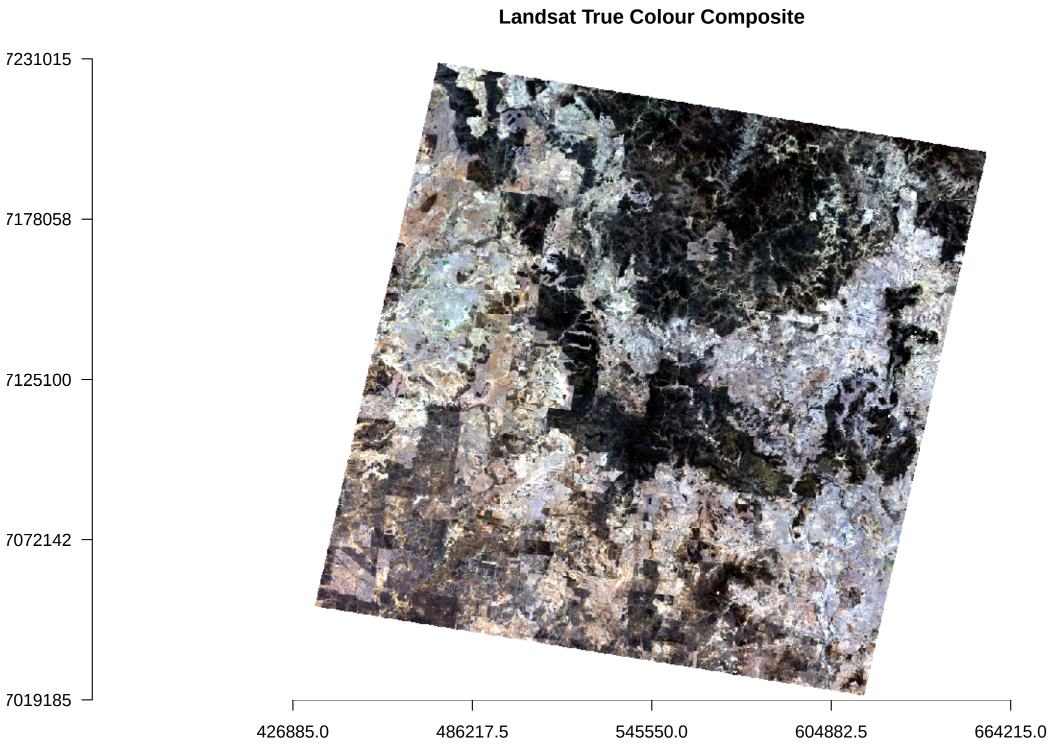
```r
library(raster)
library(sp)
library(rgdal)
library(rgeos)
```

Read the Injune .img file, called example_raster.img, for 16 July 2000 - use the stack function to import multiple layers of the image.

```r
img <-stack('data/example_raster.img')
```

Plot and view structure of Injune data Visualise the landsat scene using plotRGB - this function makes a Red-Green-Blue plot based on three layers in a RasterBrick or RasterStack

```r
plotRGB(img, r = 3, g = 2, b = 1, axes = TRUE, stretch = "lin", main = "Landsat True Colour Composite")
```



Lesson Inspecting the structure of raster files can be messy.

Usually for other data types I would use str to check the structure of the data. The output of the structure function for raster stacks can be overwhelming and difficult to read if you're not used to it (below is only a small section of the total output you will get).

```
str(img)
```

Formal class 'RasterStack' [package "raster"] with 11 slots ..@ filename: chr "" ..@ layers :List of 6 .. ..$:Formal class 'RasterLayer' [package "raster"] with 12 slots .. ..@ file :Formal class '.RasterFile' [package "raster"] with 13 slots .. ..@ name : chr "/Users/hollowj/Documents/Thesis/Thesis/data/Injune Data/l7tmre_p093r078_20000716_dbgm5.img" .. ..@ datanotation: chr "INT2S" .. ..@ byteorder : chr "little" .. ..@ nodatavalue : num -Inf .. ..@ NAchanged : logi FALSE .. ..@ nbands : int 6 .. ..@ bandorder : chr "BIL" .. ..@ offset : int 0 .. ..@ toptobottom : logi TRUE .. ..@ blockrows : int [1:6] 64 64 64 64 64 64 .. ..@ blockcols : int [1:6] 64 64 64 64 64 64 .. ..@ driver : chr "gdal" .. ..@ open : logi FALSE .. ..@ data :Formal class '.SingleLayerData' [package "raster"] with 13 slots .. ..@ values : logi(0) .. ..@ offset : num 0 .. ..@ gain : num 1 .. ..@ inmemory : logi FALSE .. ..@ fromdisk : logi TRUE .. ..@ isfactor : logi FALSE .. ..@ attributes: list() .. ..@ haveminmax: logi TRUE .. ..@ min : num 0 .. ..@ max : num 5174 .. ..@ band : int 1 .. ..@ unit : chr "" .. ..@ names : chr "l7tmre_p093r078_20000716_dbgm5.1" .. ..@ legend :Formal class '.RasterLegend' [package "raster"] with 5 slots .. ..@ type : chr(0) .. ..@ values : logi(0) .. ..@ color : logi(0) .. ..@ names : logi(0) .. ..@ colortable: logi(0) .. ..@ title : chr(0) .. ..@ extent :Formal class 'Extent' [package "raster"] with 4 slots .. ..@ xmin: num 426885 .. ..@ xmax: num 664215 .. ..@ ymin: num 7019185 .. ..@ ymax: num 7231015 .. ..@ rotated : logi FALSE .. ..@ rotation:Formal class '.Rotation' [package "raster"] with 2 slots .. ..@ geotrans: num(0) .. ..@ transfun:function ()
.. ..@ ncols : int 7911 .. ..@ nrows : int 7061 .. ..@ crs :Formal class 'CRS' [package "sp"] with

1 slot .. ..@ projargs: chr "+proj=utm +zone=55 +south +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"

Instead, I will ask for specific information about my data to check it has read into R correctly, and for future steps.

Check the number of layers in the image. There are 6 spectral bands in Landsat images, so there should be 6 layers.

```r
nlayers(img)
```

[1] 6

Get the spatial resolution of the image.

```r
res(img)
```

[1] 30 30

This is correct because Landsat images are at 30m by 30m resolution. Later in my analysis I will need the Coordinate Reference System (CRS) to extract spatial points from my image.

```r
crs(img)
```

Gives the crs arguments: +proj=utm +zone=55 +south +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs.

I want to extract data from this image to run statistical analyses on a dataset in dataframe format. Create a data frame from the Injune raster - it includes the band values from Landsat bands 1-6.

```r
img.df <-as.data.frame(img) #'including the NA values there are the same number of obs for lonlat.df an
```

Check structure of img.df. It is a data frame.

```r
str(img.df)
```

Add column names for the Landsat bands.

```r
colnames(img.df) <-c("Band1", "Band2","Band3", "Band4", "Band5","Band6")
```

Check column names are correctly displayed.

```r
head(img.df)
```

There are a number of NAs but these will be handled later. I want to include spatial information in my analyses, so I need to obtain those from the Landsat image and put them into the data frame.

Get longitude and latitude values for raster, using the CRS values obtained earlier using crs(img)

```r
points <-SpatialPoints(img, proj4string=CRS("+proj=utm +zone=55 +south +ellps=WGS84 +towgs84=0,0,0,0,0,0
lonlat <- spTransform(points, CRS("+proj=longlat +zone=55 +south +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +un
```

Look at the first few values of longitude and latitude and copy them into google maps (negative value first for this example) to check you are in the right region.

It is very important you specify the CRS string exactly as it appears for your raster.

```r
head(lonlat)
```

This will give you the following output: class: SpatialPoints. features: 1. extent: 146.2754, 146.2754, -25.03483, -25.03483 (xmin, xmax, ymin, ymax). coord. ref.: +proj=longlat +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +units=m.

Lesson If you don't specify the CRS string EXACTLY as it appears for your raster, you will end up in the wrong place.

When I was first trying to project the CRS, after a number of attempts that didn't even run, I found variations of the advice and code shown here: https://stackoverflow.com/questions/30018098/how-to-convert-utm-coordinates-to-lat-and-long-in-r

```r
sputm <- SpatialPoints(img, proj4string=CRS("+proj=utm +zone=55 +datum=WGS84"))
spgeo <- spTransform(sputm, CRS("+proj=longlat +datum=WGS84"))
head(spgeo)
```

When I used this generic CRS it worked and produced spatial points and then transformed them, I was so happy. Until I entered the longitude and latitude values (145.4384, 65.19418) and appeared on the map in Russia,which is very, very far away from Queensland, where I needed to be. Although my zone 55 was correct, including 'south' in the CRS string is very important. When I included south, I got longitude and latitude (146.2754, -25.03483) in the right region.

You also need to include the full CRS string in both the SpatialPoints and sp Transform function, or it will not run correctly either.

Using our correct longitude and latitude, add them to the img.df dataframe. To do this, convert longitude and latitude values in lonlat from a spatial points object to a dataframe.

```r
lonlat.df <-as.data.frame(lonlat, na.rm=TRUE)
head(lonlat.df)
```

```
         x         y
1 146.2754 -25.03483
2 146.2757 -25.03483
3 146.2760 -25.03483
4 146.2763 -25.03483
5 146.2766 -25.03484
6 146.2769 -25.03484
```

Add column names to lonlat.df.

```r
colnames(lonlat.df) <-c("Longitude", "Latitude")
head(lonlat.df)
```

Add longitude and latitude values to the img.df dataframe and combine as a new dataframe, dat.

```r
dat <- cbind(img.df, lonlat.df)
```

Create dataframe called data removing any na values.

```
data <-na.exclude(dat)
```

Another thing you could do as an alternative is to create a SpatialPixels Dataframe from the image file, then convert this to a standard dataframe. I sometimes do this, particularly if I am stacking two rasters and want to make sure they are aligned at a pixel scale.

```
injune.spdf <- as(img, "SpatialPixelsDataFrame")

#Create a normal data frame from injune.spdf
injune.df <- as.data.frame(injune.spdf)
```

Calculate new values based on your dataframe. Example: Normalised Differenced Vegetation Index (NDVI)

Create a function to calculate NDVI. NDVI is a measure of 'greenness' which can be used to tell if pixels are vegetation, bare earth, water etc. You will do this for the dataframe you made called 'data'.

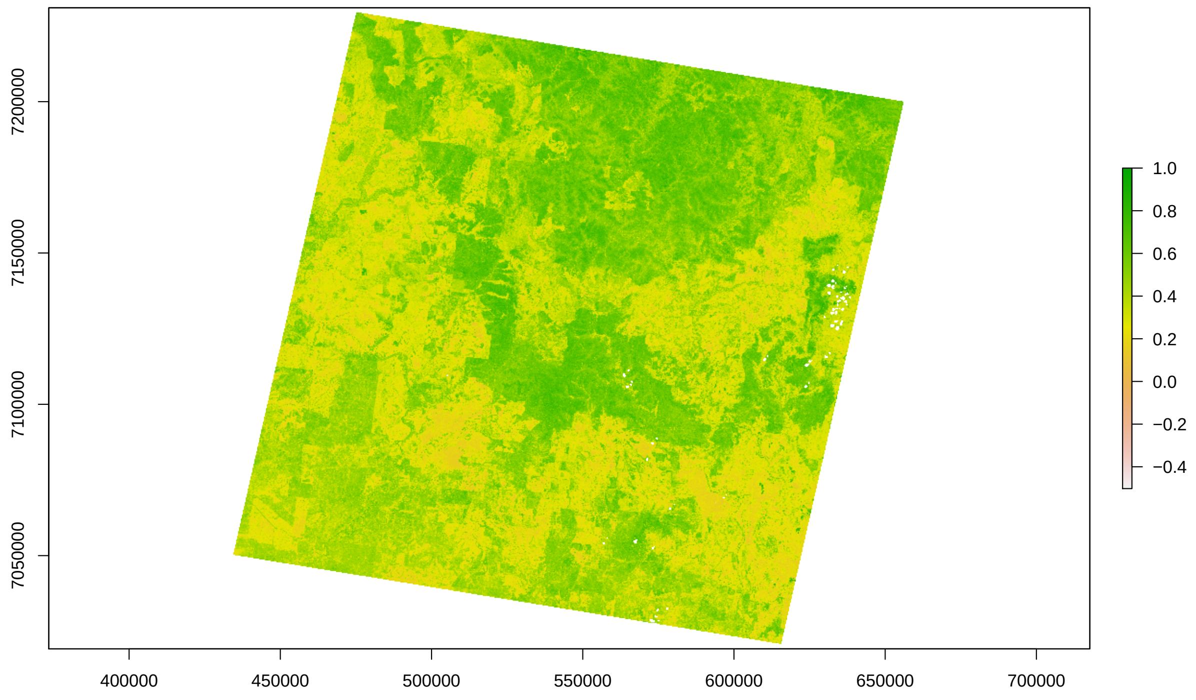
```
ndvCalc <- function(x) {
  ndvi <- (x[[4]] - x[[3]]) / (x[[4]] + x[[3]])
  return(ndvi)
}
```

Calculate NDVI for the entire Injune landsat scene, which is in our Rasterstack img.

```
ndvi <- calc(x=img, fun=ndvCalc)
```

Plot ndvi.

```
plot(ndvi)
```



I want to extract data from the landsat imagery in raster format into data frame format suitable to run analyses on. Store NDVI in a dataframe format.

```
ndvi.df = as.data.frame(ndvi, na.rm = TRUE)

str(ndvi.df) #'ndvi.df created a dataframe
```

```
## 'data.frame':    37405754 obs. of  1 variable:
##  $ layer: num  0.604 0.588 0.598 0.605 0.604 ...
```

```
head(ndvi.df)
```

```
##            layer
## 373427 0.6040269
## 373428 0.5884956
## 373429 0.5984766
## 381337 0.6045752
## 381338 0.6036036
## 381339 0.5886674
```

Label ‘layer’ column as NDVI

```
colnames(ndvi.df) <- "NDVI"
```

Add NDVI to the data dataframe and call it allvars (all variables).

```r
allvars <- cbind(data, ndvi.df[,1, drop=FALSE])
```

I don't want to include any NDVI values less than 0, as these indicate the pixel is water and out of scope of this example.

Store all the observations with NDVI value greater than 0 in a dataframe called 'data.final'.

```r
data.final <-subset(allvars, NDVI > 0)
summary(data.final) #' Confirms minimum value of NDVI is zero
```

```
##      Band1            Band2            Band3            Band4     
##  Min.   :   0.0   Min.   :   0.0   Min.   :   0.0   Min.   : 132  
##  1st Qu.: 210.0   1st Qu.: 351.0   1st Qu.: 450.0   1st Qu.:1491  
##  Median : 318.0   Median : 506.0   Median : 696.0   Median :1638  
##  Mean   : 327.6   Mean   : 517.5   Mean   : 704.3   Mean   :1657  
##  3rd Qu.: 437.0   3rd Qu.: 666.0   3rd Qu.: 933.0   3rd Qu.:1799  
##  Max.   :5174.0   Max.   :5387.0   Max.   :5721.0   Max.   :6541  
##      Band5          Band6        Longitude        Latitude     
##  Min.   :   0   Min.   :   0   Min.   :146.3   Min.   :-26.93  
##  1st Qu.:1623   1st Qu.: 973   1st Qu.:147.0   1st Qu.:-26.40  
##  Median :2219   Median :1483   Median :147.5   Median :-25.99  
##  Mean   :2224   Mean   :1495   Mean   :147.5   Mean   :-25.99  
##  3rd Qu.:2784   3rd Qu.:1971   3rd Qu.:147.9   3rd Qu.:-25.59  
##  Max.   :8233   Max.   :8756   Max.   :148.6   Max.   :-25.05  
##       NDVI        
##  Min.   :0.0006325  
##  1st Qu.:0.2904725  
##  Median :0.4159021  
##  Mean   :0.4247050  
##  3rd Qu.:0.5491667  
##  Max.   :1.0000000
```

Manually calculated NDVI and values are correct. Create a variable called class based on the ndvi.df dataframe, which gives land cover category based on NDVI cut off values. I am using 0.4 as an example, but you could set other thresholds based on remote sensing rules or your own preferences.

```r
class <-c(ifelse(data.final$NDVI >0.4,"Forest","Pasture"))
head(class)
```

```
## [1] "Forest" "Forest" "Forest" "Forest" "Forest" "Forest"
```

Create a dataframe called final.data including data.final and using cbind to add class.

```r
final.data <-cbind(data.final, class)
colnames(final.data[10]) <- "Class"
head(final.data)
```

```
##        Band1 Band2 Band3 Band4 Band5 Band6 Longitude  Latitude      NDVI
## 373427   234   307   354  1434  1031   612  146.7538 -25.04913 0.6040269
## 373428    98   265   372  1436  1145   678  146.7541 -25.04913 0.5884956
## 373429   199   298   369  1469  1220   670  146.7544 -25.04913 0.5984766
## 381337    70   315   363  1473  1309   813  146.7535 -25.04940 0.6045752
## 381338   245   358   374  1513  1257   745  146.7538 -25.04940 0.6036036
## 381339   172   278   392  1514  1343   632  146.7541 -25.04940 0.5886674
##         class
## 373427 Forest
## 373428 Forest
## 373429 Forest
## 381337 Forest
## 381338 Forest
## 381339 Forest
```

```r
summary(final.data)
```

```
##      Band1            Band2            Band3            Band4
##  Min.   :   0.0   Min.   :   0.0   Min.   :   0.0   Min.   : 132
##  1st Qu.: 210.0   1st Qu.: 351.0   1st Qu.: 450.0   1st Qu.:1491
##  Median : 318.0   Median : 506.0   Median : 696.0   Median :1638
##  Mean   : 327.6   Mean   : 517.5   Mean   : 704.3   Mean   :1657
##  3rd Qu.: 437.0   3rd Qu.: 666.0   3rd Qu.: 933.0   3rd Qu.:1799
##  Max.   :5174.0   Max.   :5387.0   Max.   :5721.0   Max.   :6541
##      Band5          Band6        Longitude        Latitude
##  Min.   :   0   Min.   :   0   Min.   :146.3   Min.   :-26.93
##  1st Qu.:1623   1st Qu.: 973   1st Qu.:147.0   1st Qu.:-26.40
##  Median :2219   Median :1483   Median :147.5   Median :-25.99
##  Mean   :2224   Mean   :1495   Mean   :147.5   Mean   :-25.99
##  3rd Qu.:2784   3rd Qu.:1971   3rd Qu.:147.9   3rd Qu.:-25.59
##  Max.   :8233   Max.   :8756   Max.   :148.6   Max.   :-25.05
##       NDVI               class
##  Min.   :0.0006325   Forest :19768928
##  1st Qu.:0.2904725   Pasture:17634154
##  Median :0.4159021
##  Mean   :0.4247050
##  3rd Qu.:0.5491667
##  Max.   :1.0000000
```

Initial data wrangling is complete at this point, so you can move on to analyses using final.data. You can apply these steps to other images and the principles apply to other image files e.g. tifs, not just .img. You can then treat your data as you would any other standard dataframe, e.g. subsample your data, fit models, produce summary statistics, etc.