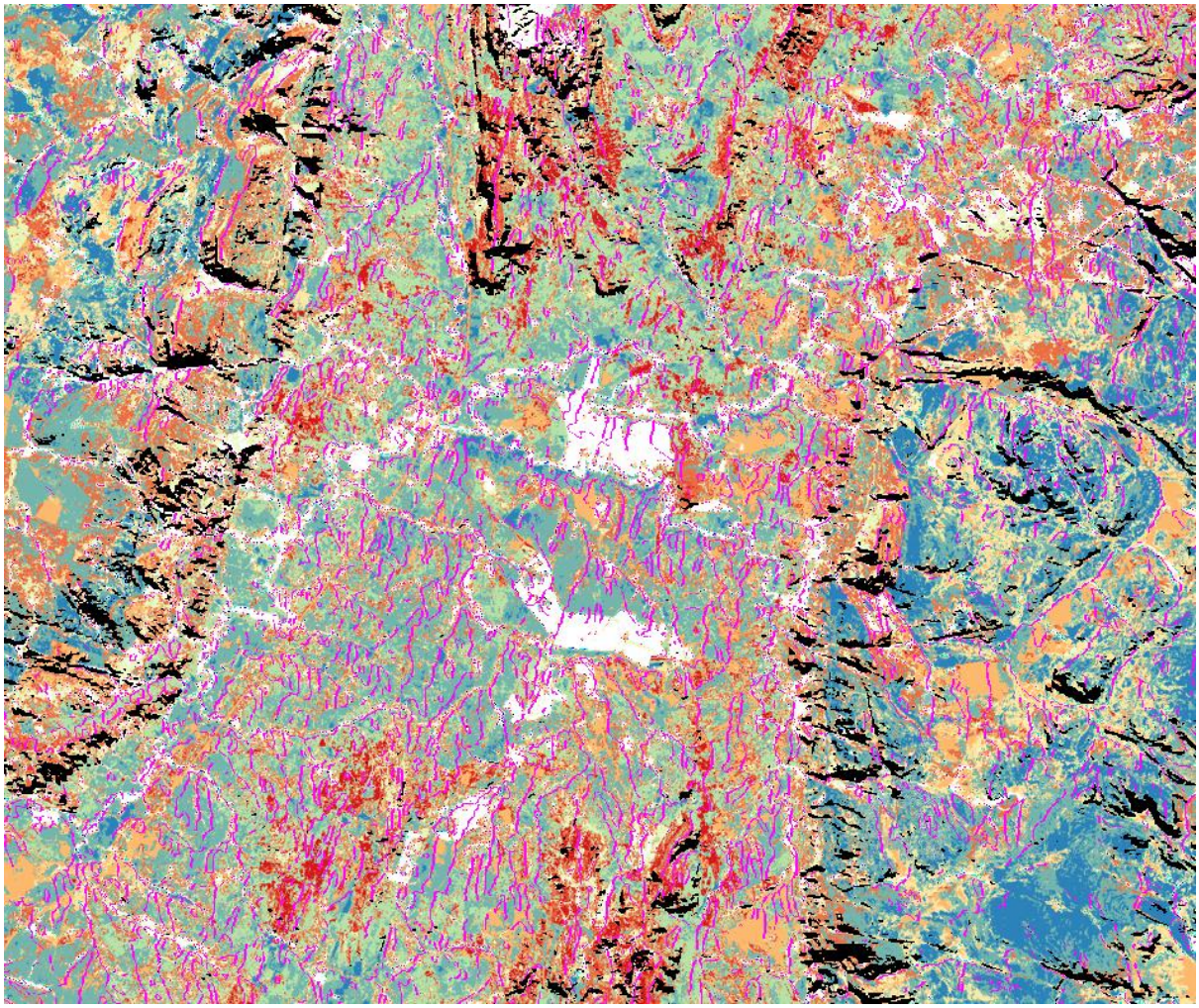


Creating a preliminary “geological-structural” exploration basemap

Visit https://gdatasystems.com

1- Geological Single Band Classification from Multispectral Images

Geological maps can be generated automatically using a set of multispectral satellite images such as Landsat, aster and Sentinel2 products. In here we are going to cover on how to create a map using two distinct approaches, one Unsupervised using K-Means clustering and one Supervised using Spectral Angle Mapper (SAM). We will cover a 1 by 1 degree area, but the same process can be applied over smaller areas.

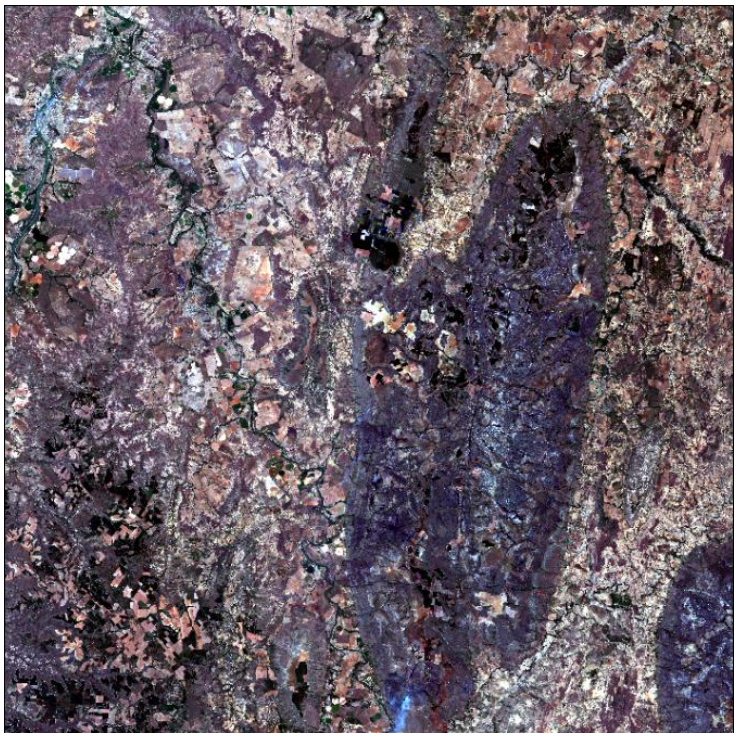
These technics have as objective to distinguish different rock spectral signatures consequently rock types and generate an image as close as possible to a geological map.

The tools used on these processes are **R language** with some geospatial libraries (**terra**, **cluster** and **RStoolbox**) and Sentinel2 satellite image.

STEP 1 – Checking the Image on the visible bandwidth

The Sentinel2 bands 2, 3 and 4 are combined on a true color composition (TCC) or visible bandwidth. Here we will be using the bands with 20m resolution having all working bands (2, 3, 4, 5, 6, 7, 8A, 11 and 12) using the same spatial resolution. The full scene is used here and, due to its size, some processes may take a few minutes to be completed depending on the power of your system.

```
library(terra)
#Working directory where the images and data are located
wd<-'C:/Users/User/Desktop/R algo'
setwd(wd)
# Plotting visible scene (bands 4, 3 and 2)
filenames <- paste0('B0', 4:2, "_20m.jp2")
vis<-rast(filenames)
plotRGB(vis,stretch='lin',axes=TRUE)#see map below
```



In this image we can distinguish most of the different spectral responses that are in most cases associated with different lithologies.

STEP 2 – Masking areas with no desired spectral response

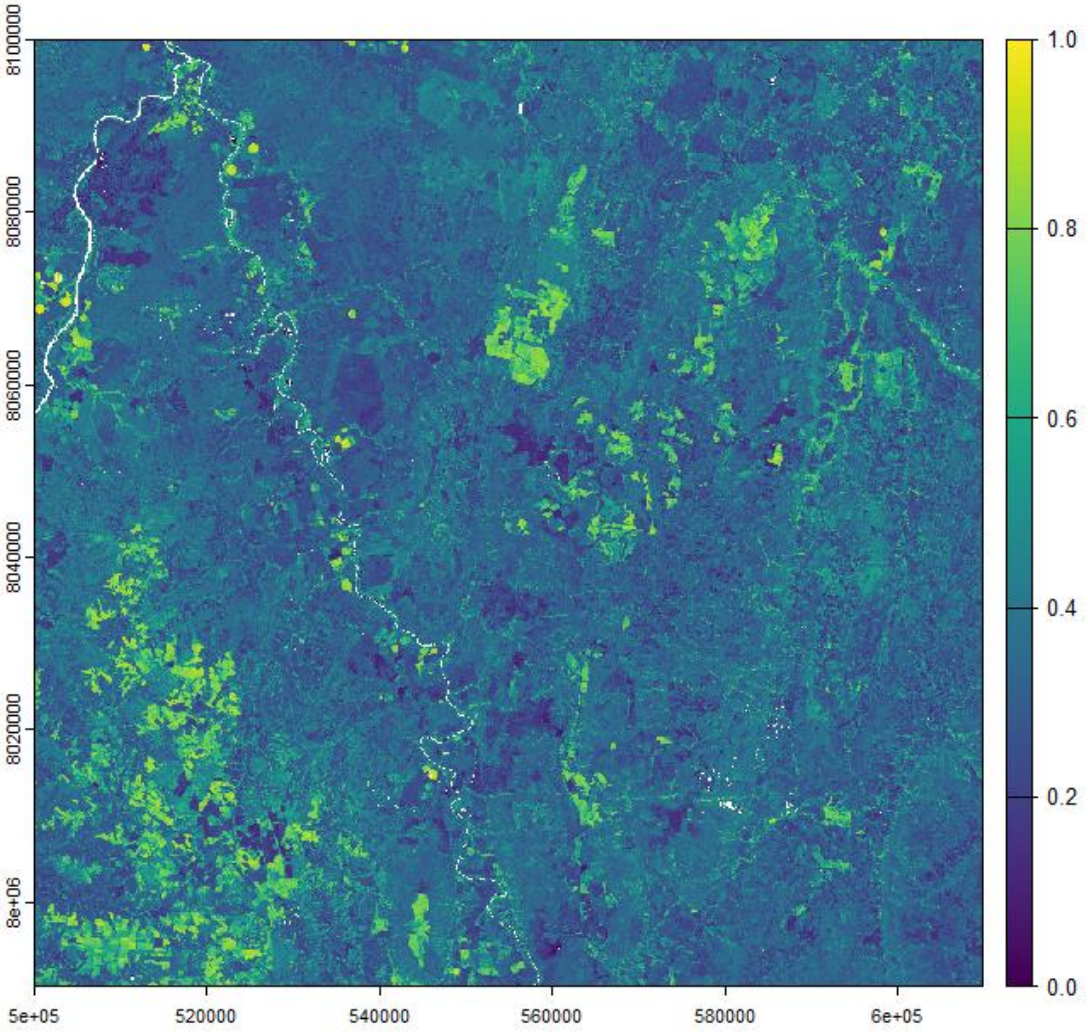
Healthy and dense vegetation areas and water bodies will obliterate the ground spectral response and will generate a not desired signature. These areas can be removed (masked) from our scene using the following indexes:

SAVI

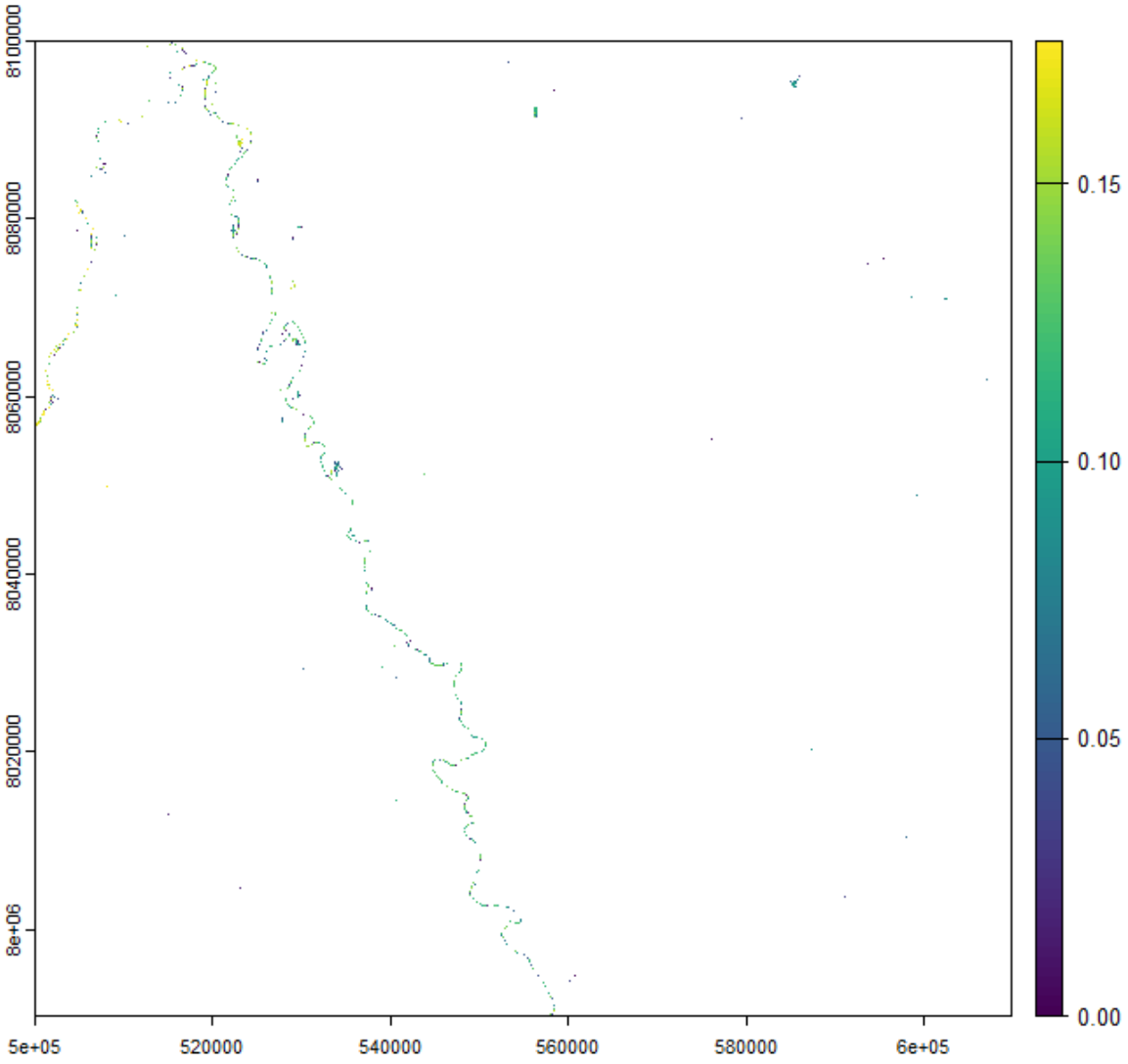
MNDWI

NDTI

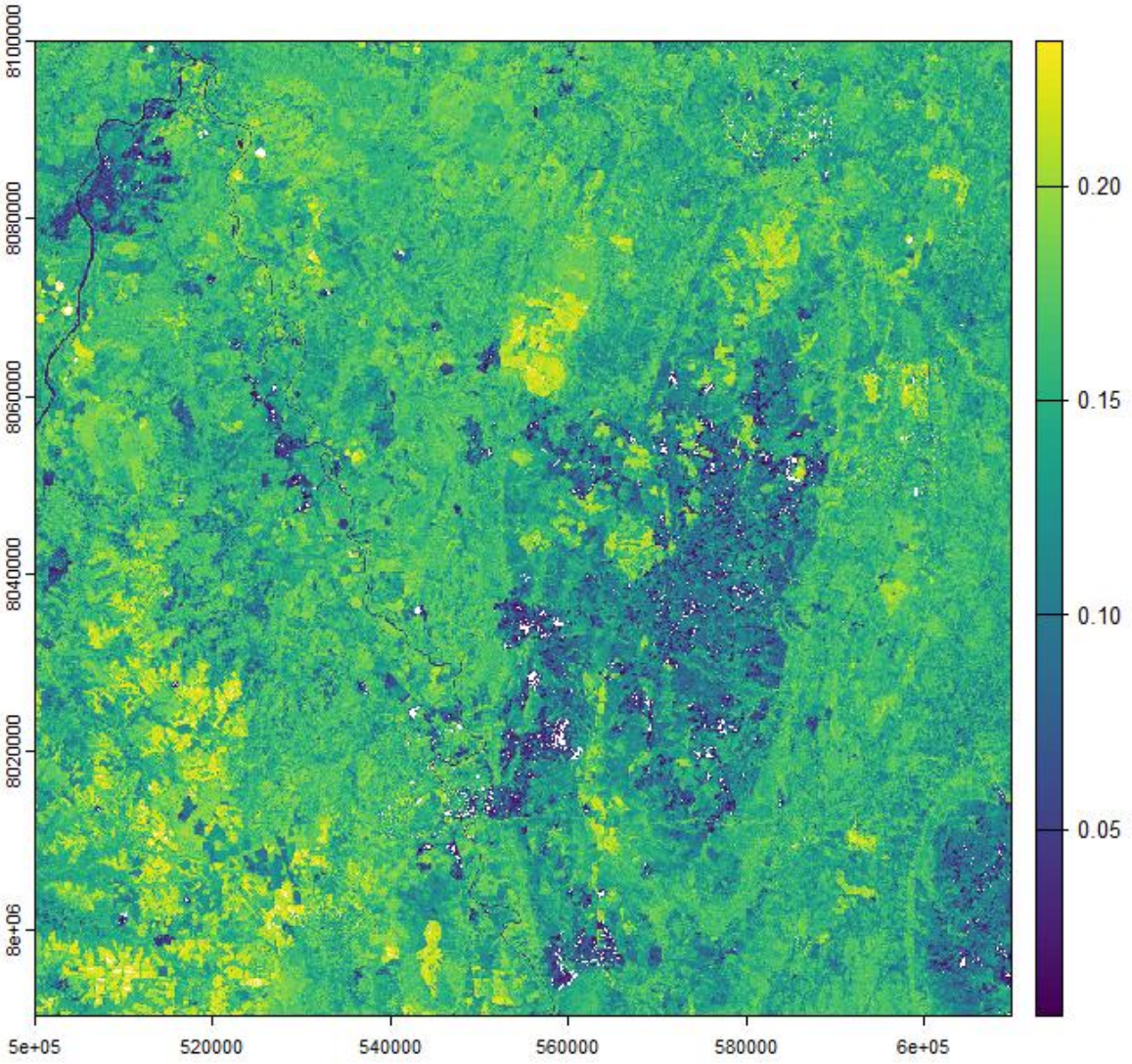
```
#index to separate water, vegetation and exposed soil/rock
nir<-rast('B8A_20m.jp2')
red<-rast('B04_20m.jp2')
savi<-(nir-red)/(nir+red+0.5)*1.5
plot(savi,range=c(0,1))
```



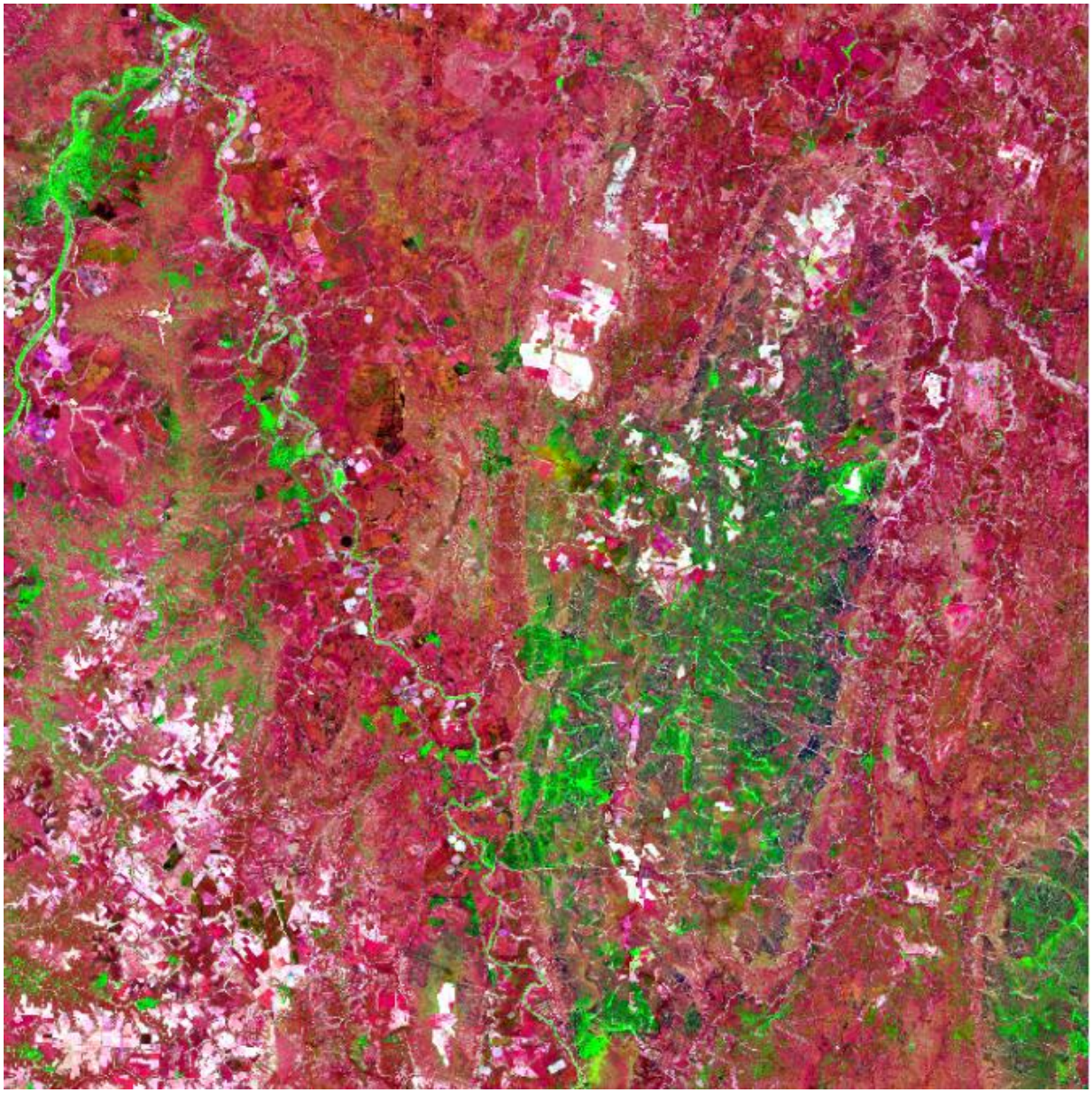
```
green<-rast('B03_20m.jp2')
swir<-rast('B11_20m.jp2')
mndwi<-(green-swir)/(green+swir)
                          plot(mndwi,range=c(0,max(mndwi[,1])))
```



```
swir1<-rast('B11_20m.jp2')
swir2<-rast('B12_20m.jp2')
ndti<-(swir1-swir2)/(swir1+swir2)
plot(ndti,range=c(min(ndti[,1]),max(ndti[,1])))
```

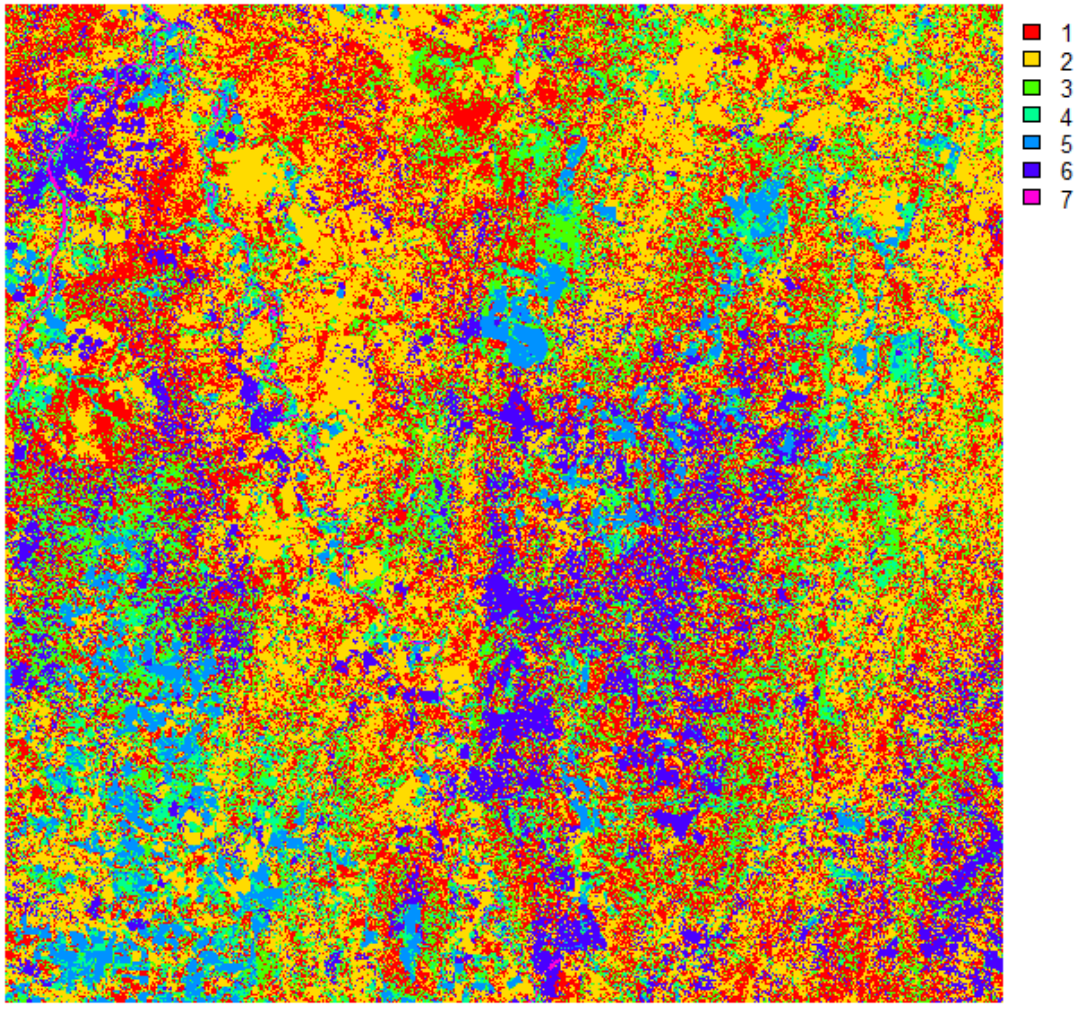


```
rgbi<-rast(list(ndti, mndwi ,savi))
plotRGB(rgbi,stretch='lin')
```

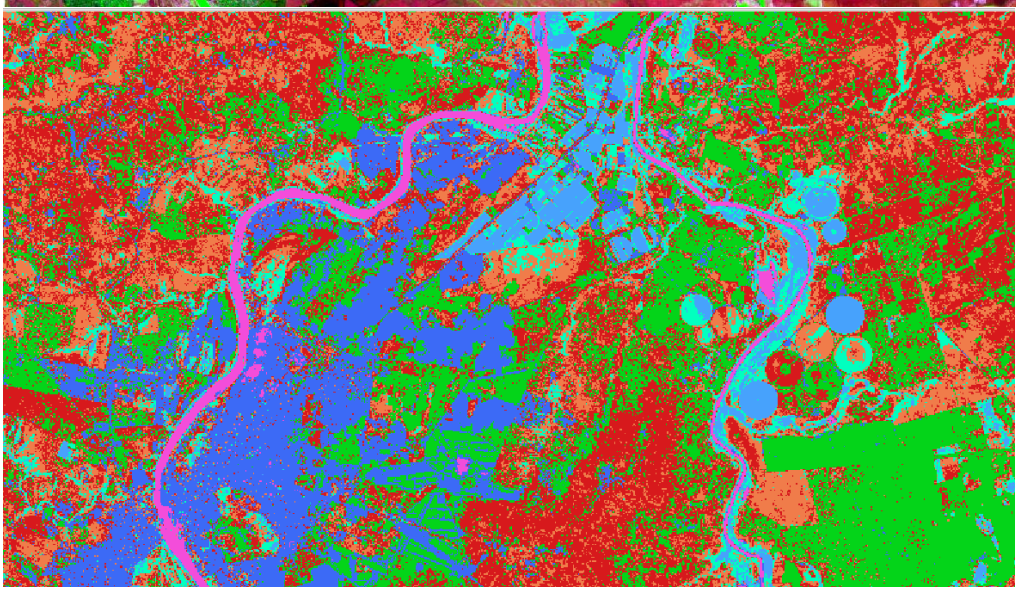


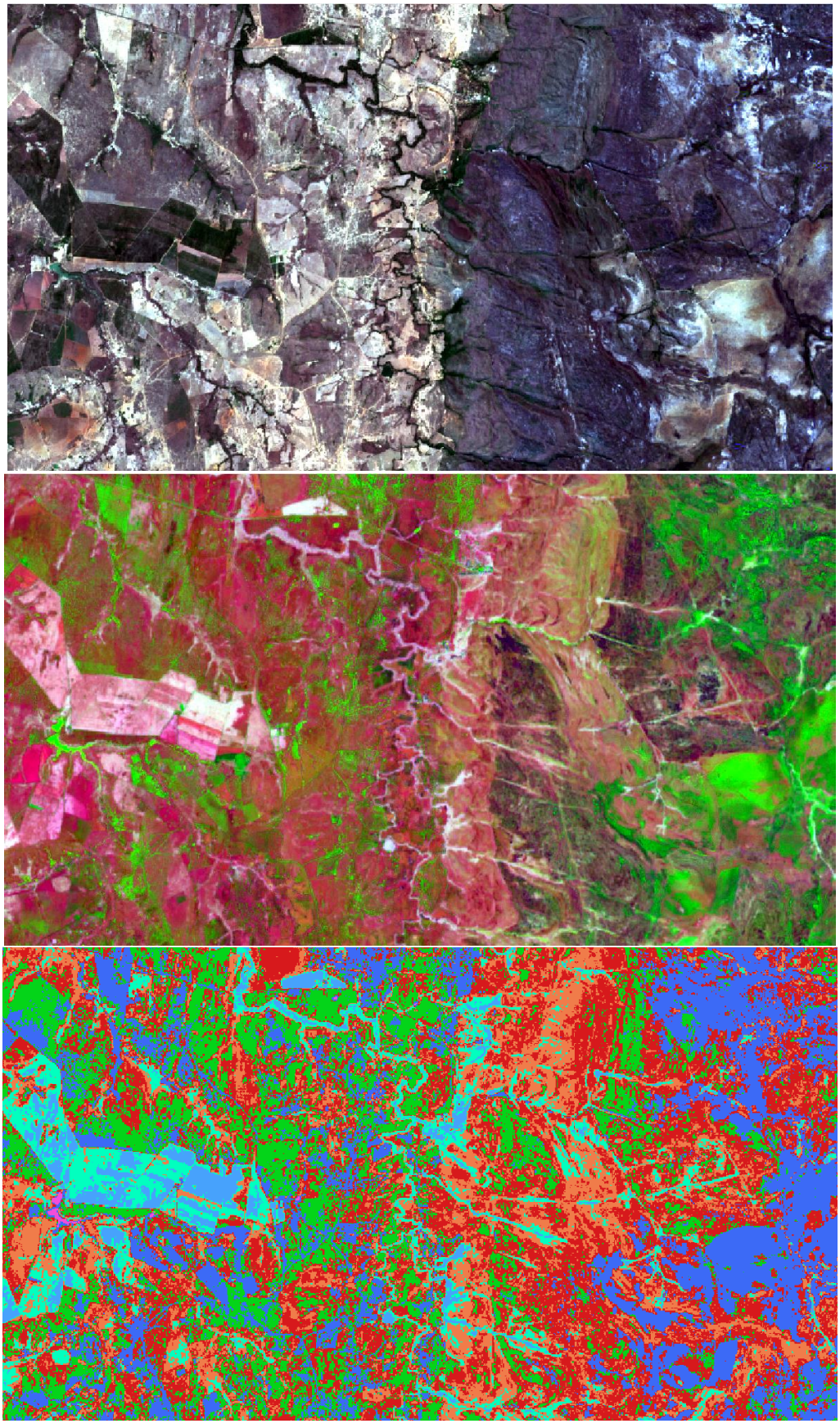
Executing a K-means classification on the FCC above to select the classes to be excluded.

```
library(cluster)
#creating dataframe from raster
img<-c(rgbi[[1]],rgbi[[2]],rgbi[[3]])
#creating vector from dataframe above
ar <- values(img)
#index of valid values which are not NA
values <- which(!is.na(ar))
#removing NA from vector
ar <- na.omit(ar)
# Performing the classificatation (it takes a while depending on the scene size)
Result <- clara(ar,7,samples=500,metric="manhattan",pamLike=T)
# Creating an empty resulting raster
kmraster <- rast (rgbi[[1]])
#loading the resulting class values back creating the resulting raster
values(kmraster) <- Result$clustering
plot(kmraster, legend=T,axes=F, col = rainbow(7))
```



It is important now to identify which class above we are going to be excluded in the process. Here we will be focused on excluding partially healthy vegetation, healthy vegetation and water bodies. Below we show that they are associated with class 4 (cyan), 5 (light blue) and class 7 (magenta) respectively.

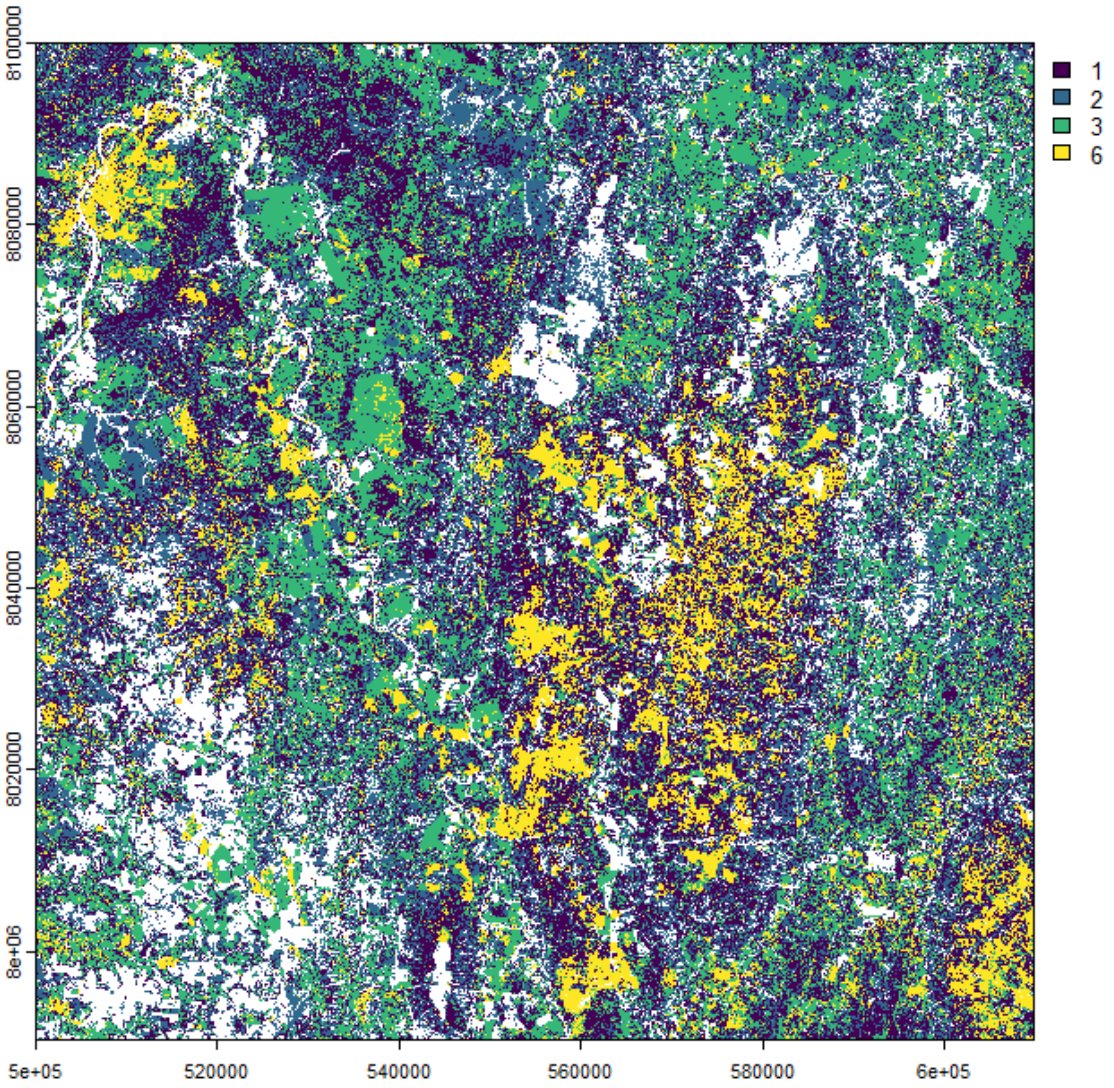




Note: Be aware that class 4 (blue) represents distinct features as exposed soil/rock, urban areas and zones affected by wildfires and will be kept in our analysis.

Below we have our final mask that will be applied in the composite scenes we are going to generate.

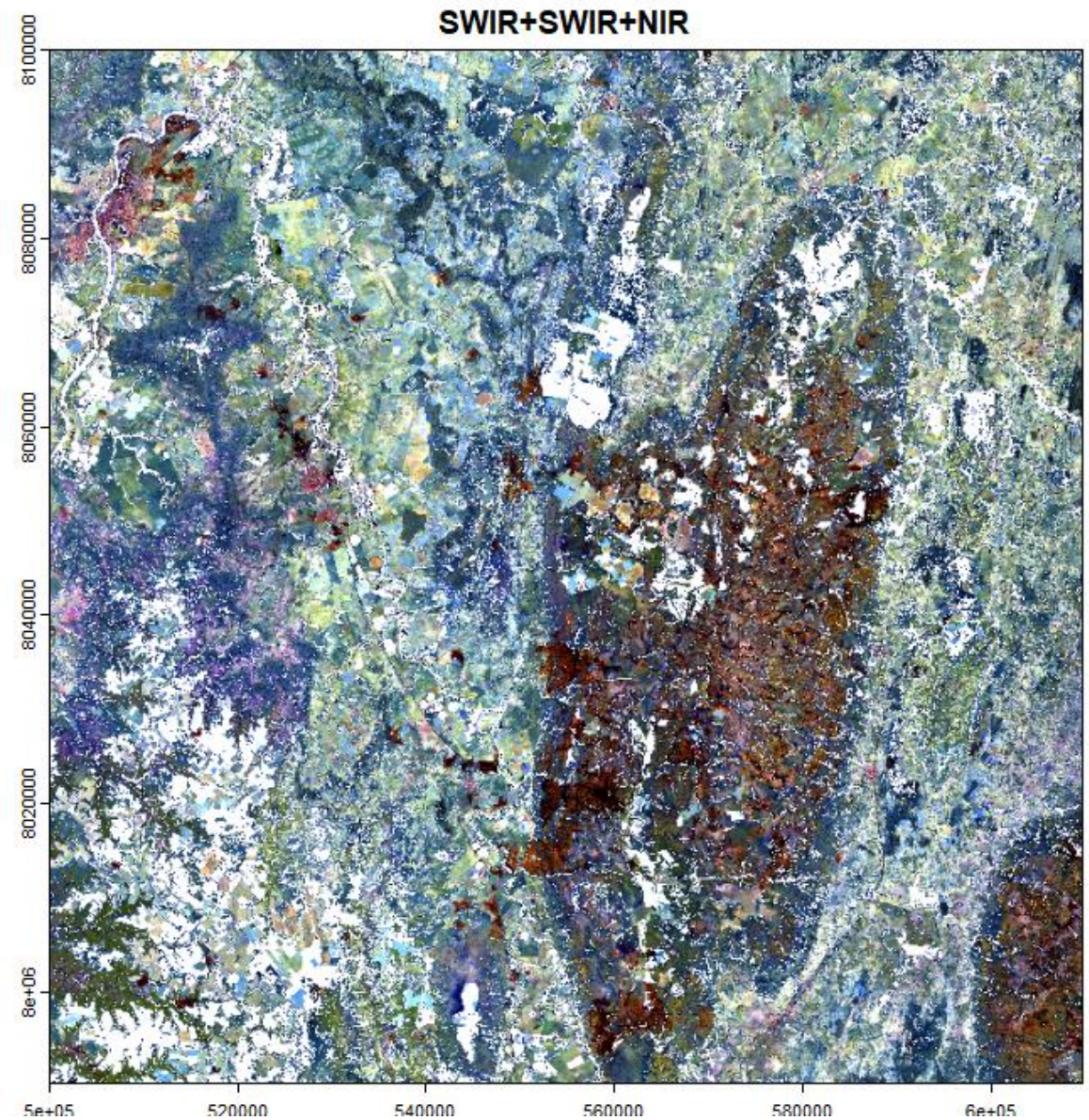
```
mask1<-kmraster
mask1[mask1 == 4 | mask1 == 5 | mask1 == 7] <- NA
plot(mask1)
```



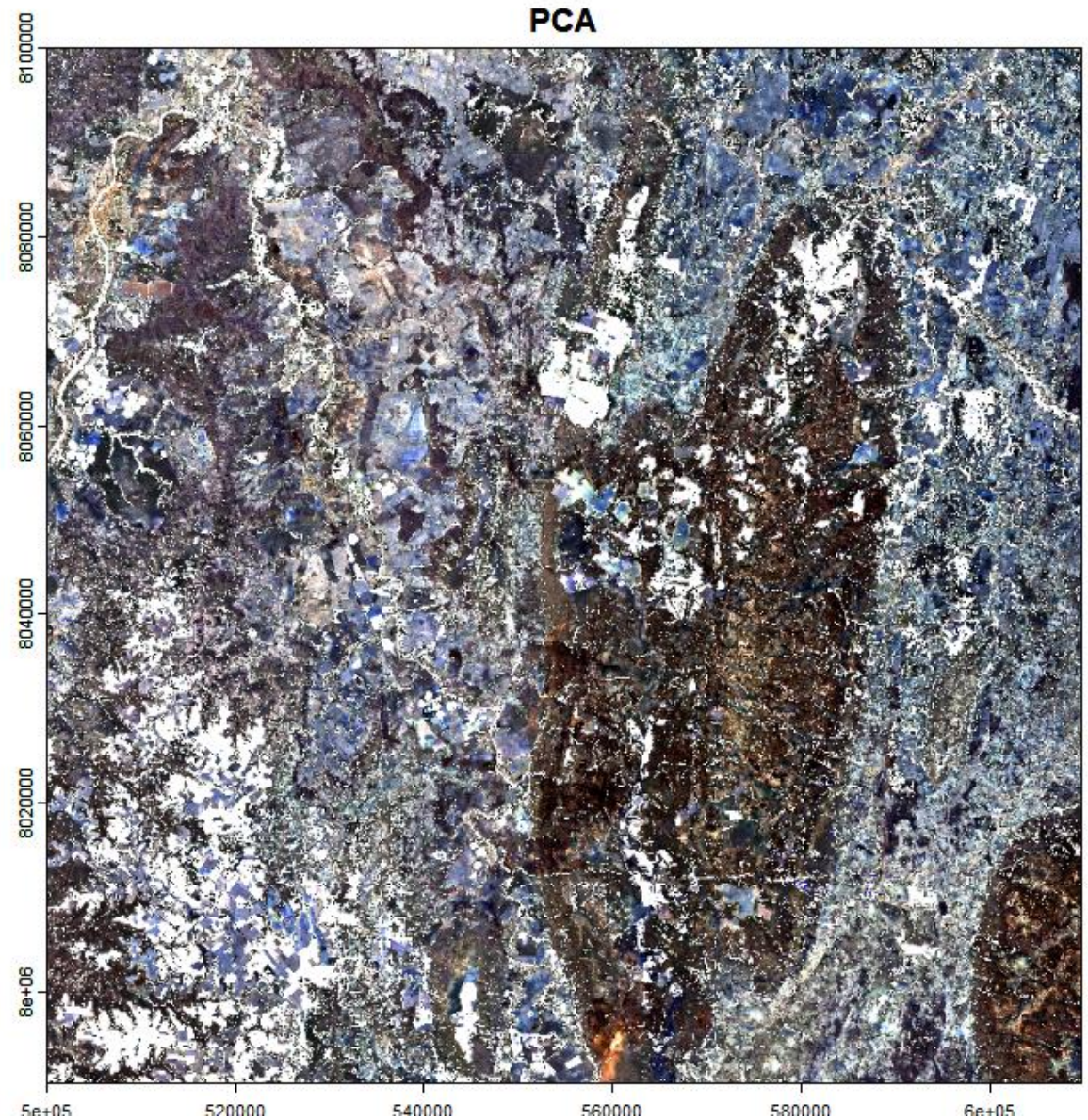
STEP 3 – Creating Two Composite Scenes Using Distinct Methodologies

Two distinct False Color Composite (FCC) scenes will be created, one using bands 12, 11 and 8A called SWIR-SWIR-NIR that enhances exposed soil and outcropping rocks, and another one using the first three Principal Component Analysis results (PCA 1, PCA 2 and PCA 3).

```
filenames <- paste0('B',c('12','11','8A'), "_20m.jp2")
swirnir<- rast(filenames)
masked_swirnir <- mask(swirnir, mask1)
plotRGB(masked_swirnir,stretch='lin',axes=TRUE,main='SWIR+SWIR+NIR',mar=c(1,1,2,1))
```

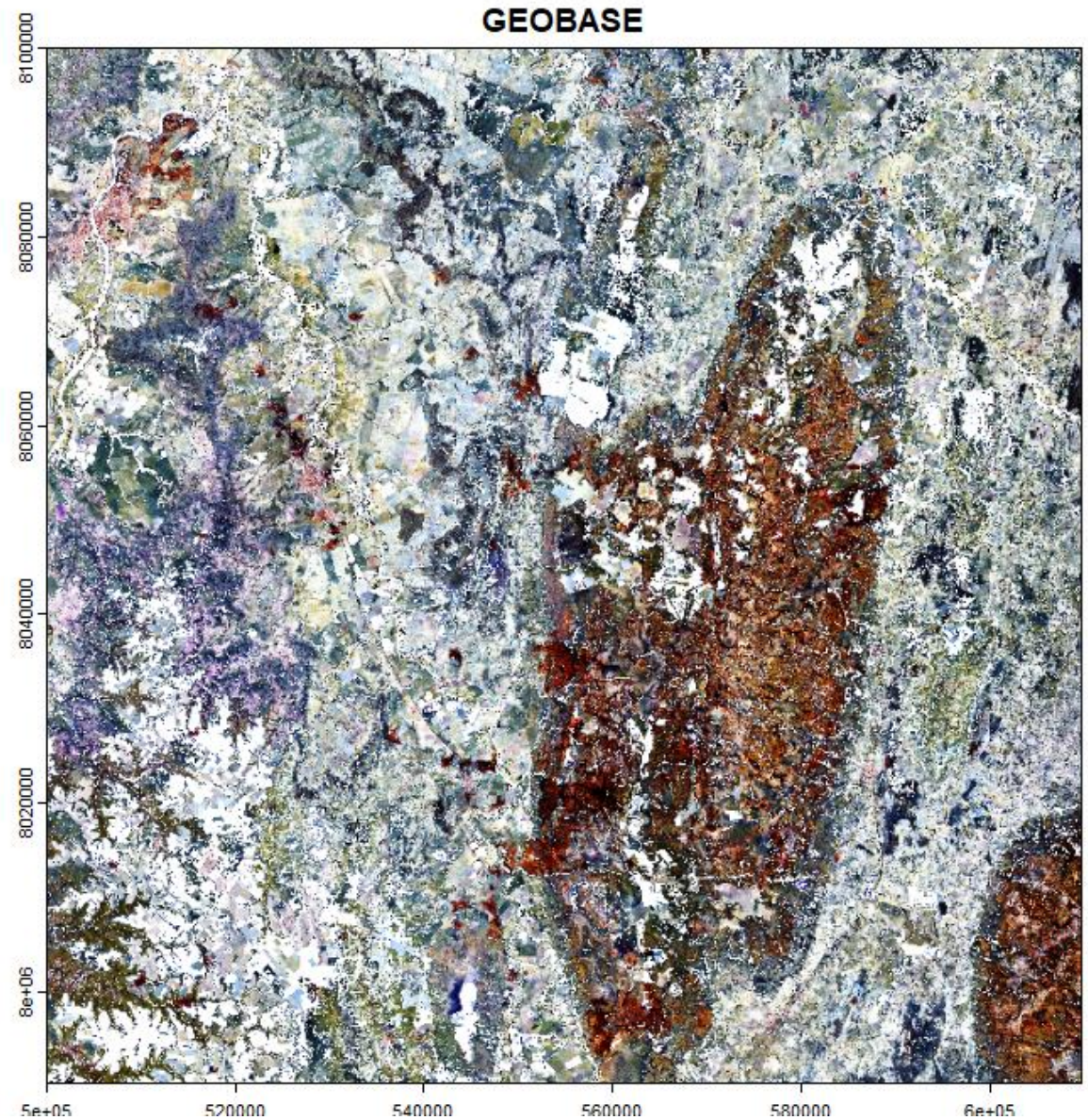


```
rlist<-c("B02_20m.jp2","B03_20m.jp2","B04_20m.jp2","B05_20m.jp2","B06_20m.jp2",
         "B07_20m.jp2","B8A_20m.jp2","B11_20m.jp2","B12_20m.jp2")
rlist2<-c("b2","b3","b4","b5","b6","b7", "b8a","b11","b12")
sentinel<-rast(rlist)
names(sentinel)<-rlist2
set.seed(1)
samples_ <- spatSample(sentinel, 3000000,method="random")# ~10%
pca <- prcomp(samples_, scale = TRUE)
rm(samples_)
pca
pcis<-predict(sentinel,pca,filename='pcis.tif',overwrite=TRUE)
pcaALL<-c(pcis[[1]],pcis[[2]],pcis[[3]],pcis[[4]],pcis[[5]],pcis[[6]],pcis[[7]],
        pcis[[8]] ,pcis[[9]])
pcaALL <- mask(sentinel, mask1)
plotRGB(pcaALL,stretch='lin',r=1,g=2,b=3,axes=TRUE,main='PCA',mar=c(1,1,2,1))
```



Integrating the two scenes into one FCC by multiplication.

```
geobase<-(pcaALL*masked_swirnir)/17089 #nomalizing using range maximum value 17089
plotRGB(geobase,stretch='hist',axes=TRUE,main='GEOBASE',mar=c(1,1,2,1))
writeRaster(geobase,'geobase.tif', overwrite=TRUE)
```



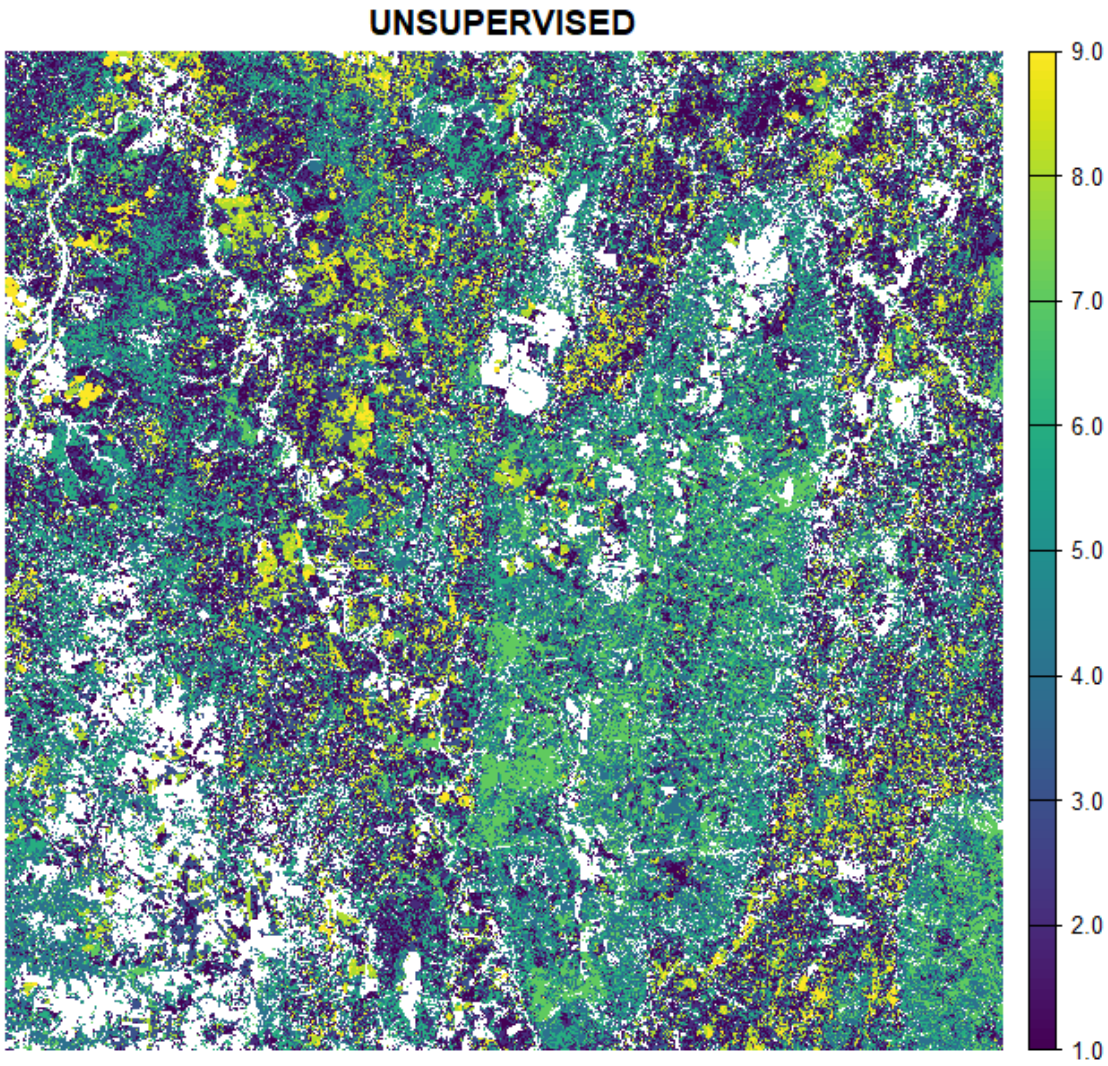
The resulting FCC already looks like a geological map, but it is not a single band classification with each class associated with a spectral response group (or geological unit). Next, we are going to generate this single band classification using two different processes. One unsupervised using K-means and one supervised using Spectral angle mapper (SAM).

STEP 4 – Unsupervised process

Using K-means to generate the spectral map (8 classes) using the GEOBASE scene resulting in the first “Geological Map”.

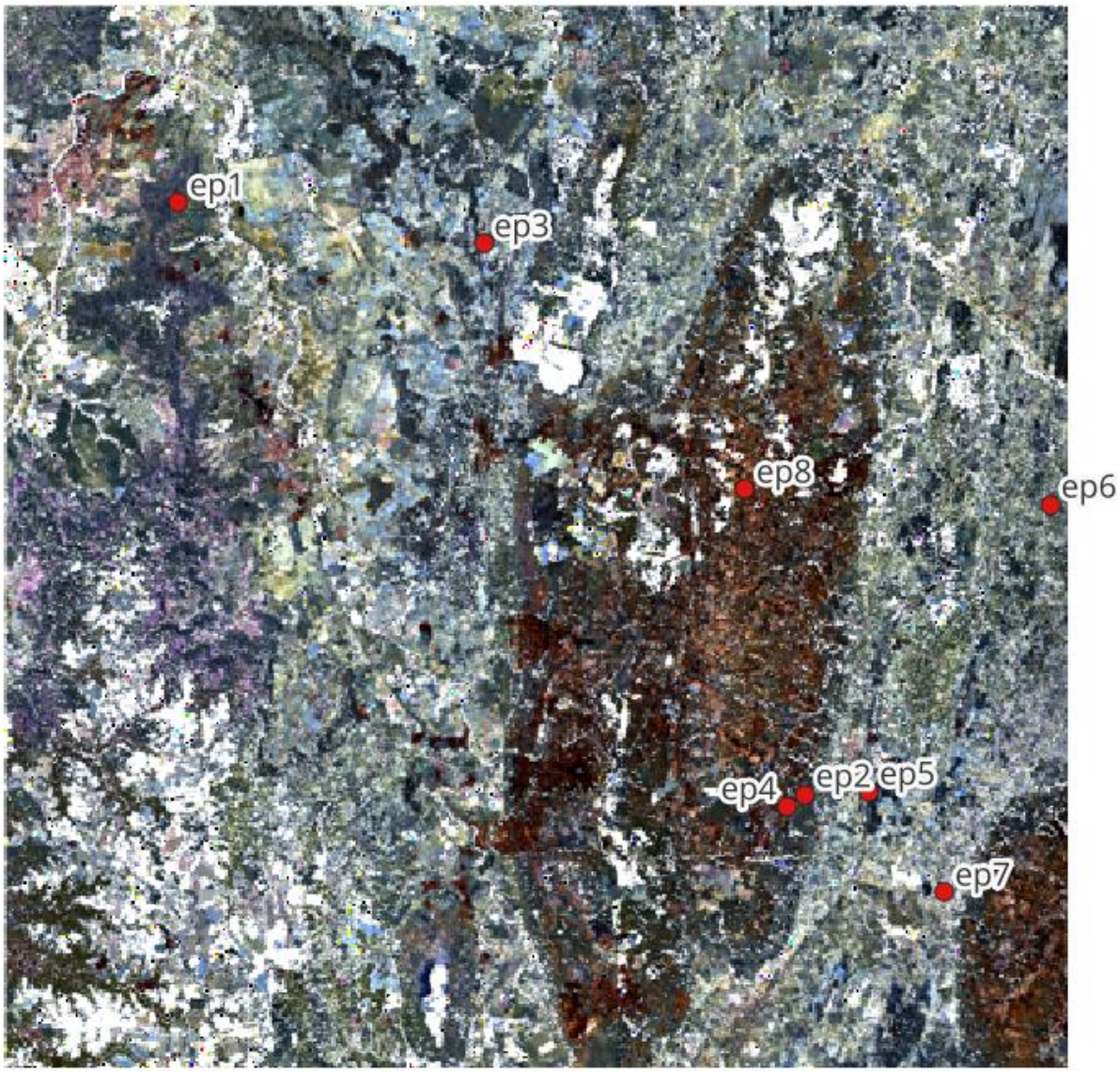
Unsupervised

```
img<-c(geobase[[1]],geobase[[2]],geobase[[3]],geobase[[4]],geobase[[5]],
       geobase[[6]],geobase[[7]],geobase[[8]],geobase[[9]])
ar <- values(img)
values <- ar
#replacing  NA from vector by 0
ar[is.na(ar)] = 0
Result <- clara(ar,9,samples=500,metric="manhattan",pamLike=T)# 12 minutes
Kmraster9 <- rast (geobase[[1]])
values(kmraster9) <- Result$clustering
#Here we mask the result and extract one of class resulting in 8 classes
kmraster8 <- mask(kmraster9, mask1)
plot(kmraster8, legend=T,axes=F,main='UNSUPERVISED')
writeRaster(kmraster8,'UNSUPERVISED.tif', overwrite=TRUE)
```



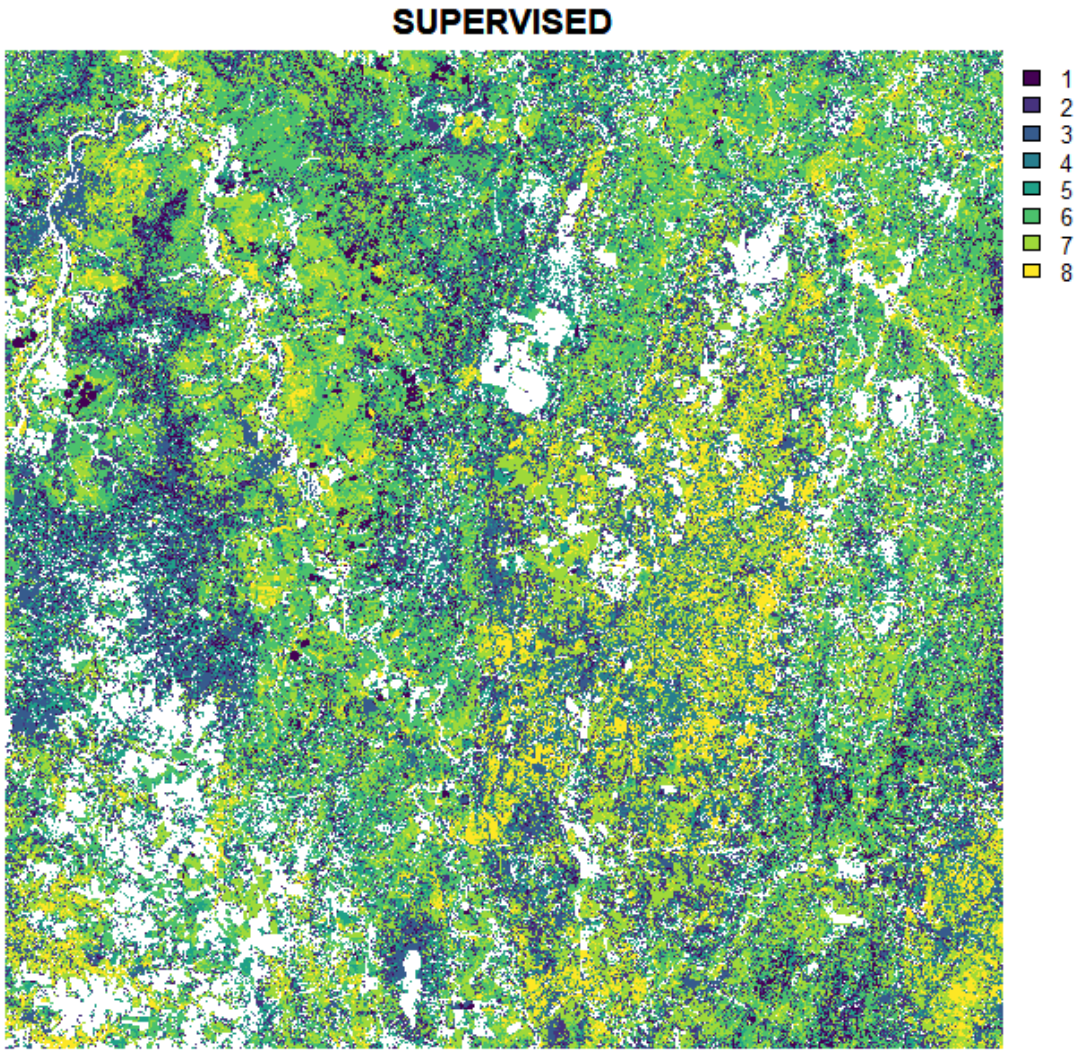
STEP 5 – Supervised process

Using Spectral Angle Mapper (SAM) to generate the spectral map using the GEOBASE scene to create the final "Geological Map".
In here we must select the coordinates of points that will better represent the different spectral response. Open the **geobase.tif** using the QGIS program and extracting one point for each spectral response (lithology) you can distinguish on the scene. Eight points were selected, and the final classification will have eight distinct classes as result too.



Supervised

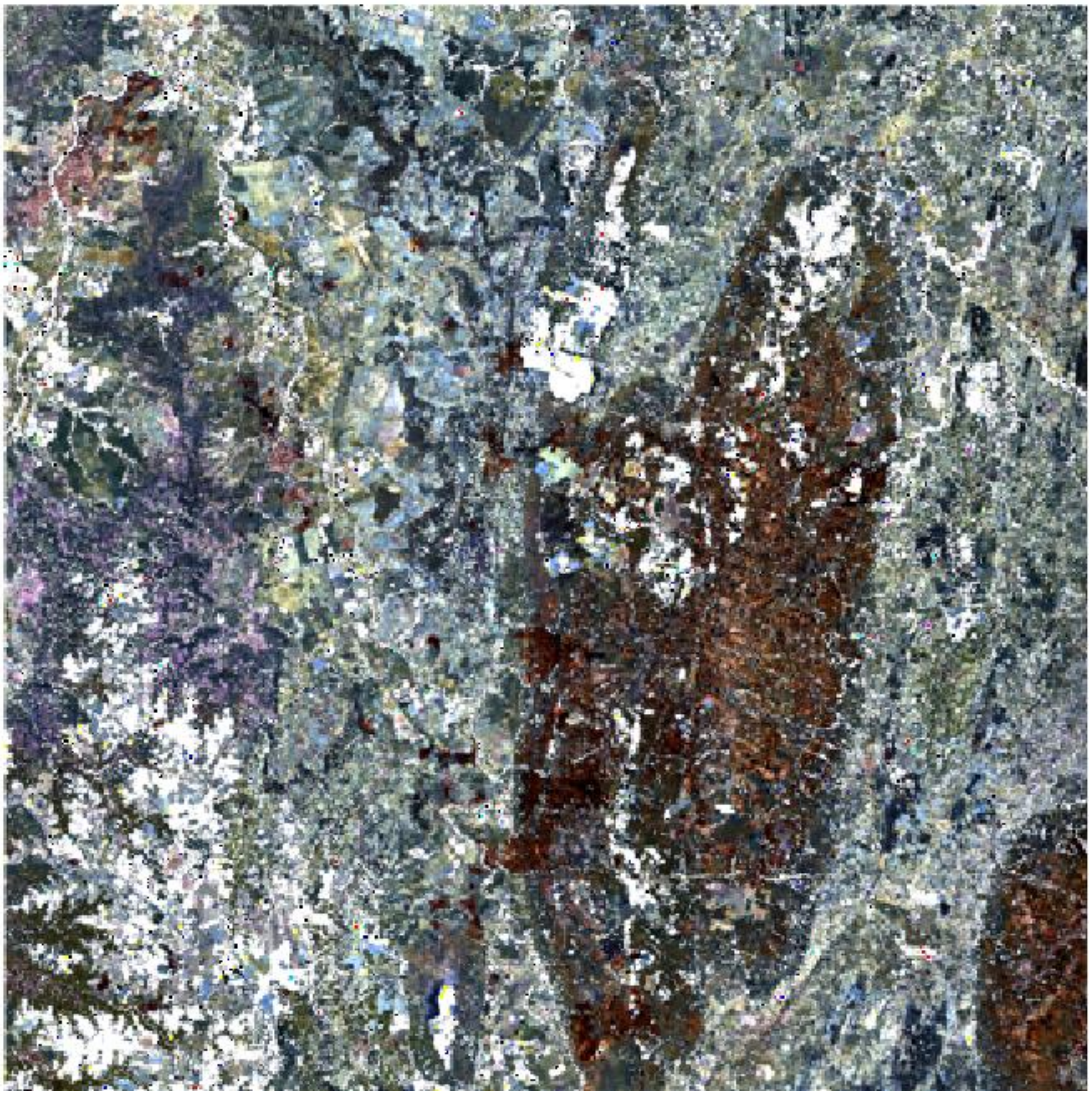
```
library(RStoolbox)
pontos <- data.frame(x = c(517831,582845,549504, 580941,589457,608259,597139,
576555),
                     y = c(8079732,8018364,8075490, 8017184,8018611,8048353
,8008282,8050021))
endmembers <- extract(geobase, pontos)
rownames(endmembers) <- c("litho1","litho2","litho3","litho4","litho5",
"litho6","litho7","litho8")
## Classification based on minimum angles
sam <- sam(geobase, endmembers, angles = FALSE)
plot(sam, legend=T,axes=F,main='SUPERVISED')
writeRaster(sam,'SUPERVISED.tif', overwrite=TRUE)
```



STEP 6 – Comparing the Results

Here we are going to Compare the K-means unsupervised result with the Spectral Angle Mapper (SAM) result. **It is important to note that the resulting classification raster is not intended to be a final geological map but a base for further spatial analysis or final vectorization into a geological map**.
We are going to load the 3 rasters (geobase.tif, SUPERVISED.tif and UNSUPERVISED.tif) into QGIS and visualize the results.

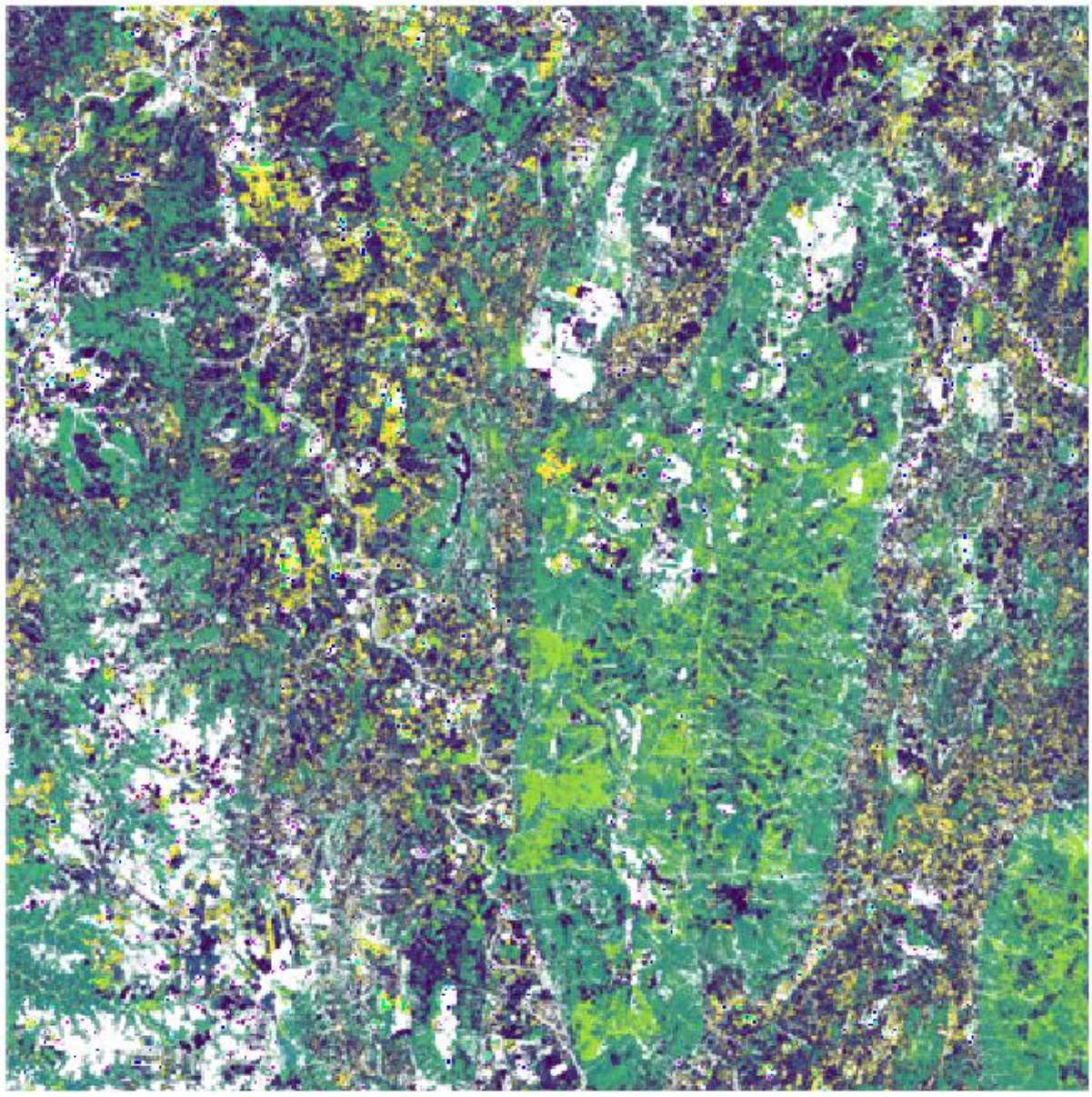
Geobase has 9 bands integrating the PCA and SWIR SWIR NIR FCC. Here you see a FCC with geobase bands 1, 2 and 3. Note the good spectral contrast achieved.



UNSUPERVISED

K-means is a centroid-based clustering algorithm, where the distance between each data point and a centroid is calculated and assigned to a cluster. The main objective is the identification of a determined number of groups (K) in our dataset. We chose to use 9 groups where one is the masked excluding areas. After processing we removed this group the resulting clustering wiil have eight groups. The data point is assigned to each group based on distance and similar features minimizing the sum of distances between the data points and the cluster centroid identifying the group in which each data point should be included.

The resulting K-means single band is presented below.



SUPERVISED

The spectral angle mapper (SAM) calculates the angle between two spectra: a known spectrum represented by the visual selection of our endpoints against unknown pixel spectra (our geobase dataset) by treating these as vectors in a n-dimensional feature space where n is the number of bands used (9 bands in our case).

The resulting SAM single band is presented below:

Geological Classification full code

```
################################################################################
#       Sentinel 2 automated rocktype classification (Geological mapping)
#       GDataSystems 2024
#       Script Created by Andre Costa
################################################################################

#       Scene Visualization
library(terra)
#Working directory where the images and data are located
wd<-'C:/Users/User/Desktop/R algo'
setwd(wd)
# Plotting visible scene (bands 4, 3 and 2)
filenames <- paste0('B0', 4:2, "_20m.jp2")
vis<-rast(filenames)
plotRGB(vis,stretch='lin',axes=TRUE)
#_______________________________________________________________________________
################################################################################
#       Creating the indexes composite Image
#Calculating index to separate water, urban zones, vegetation and exposed soil/rock
nir<-rast('B8A_20m.jp2')
red<-rast('B04_20m.jp2')
savi<-(nir-red)/(nir+red+0.5)*1.5
green<-rast('B03_20m.jp2')
swir<-rast('B11_20m.jp2')
mndwi<-(green-swir)/(green+swir)
swir1<-rast('B11_20m.jp2')
swir2<-rast('B12_20m.jp2')
ndti<-(swir1-swir2)/(swir1+swir2)
nir<-rast('B8A_20m.jp2')
swir<-rast('B11_20m.jp2')
ndbi<-(swir-nir)/(swir+nir)
rgbi<-rast(list(ndti, mndwi ,savi))
plotRGB(rgbi,stretch='lin')
#_______________________________________________________________________________
################################################################################
#       Kmeans Classification based on indexes Composite scene
library(cluster)
#creating dataframe from raster
img<-c(rgbi[[1]],rgbi[[2]],rgbi[[3]])
#creating vector from dataframe above
ar <- values(img)
#index of valid values which are not NA
values <- which(!is.na(ar))
#removing NA from vector
ar <- na.omit(ar)
# Performing the classificatation (it takes a while depending on the scene size)
Result <- clara(ar,7,samples=500,metric="manhattan",pamLike=T)
# Creating an empty resulting raster
kmraster <- rast (rgbi[[1]])
#loading the resulting class values back creating the resulting raster
values(kmraster) <- Result$clustering
plot(kmraster, legend=T,axes=F, col = rainbow(7))
mask1<-kmraster
mask1[mask1 == 4 | mask1 == 5 | mask1 == 7] <- NA
plot(mask1)
#_______________________________________________________________________________
################################################################################
#       Geo compositing
#       Geo compositing SWIR SWIR NIR composite
filenames <- paste0('B',c('12','11','8A'), "_20m.jp2")
swirnir<- rast(filenames)
masked_swirnir <- mask(swirnir, mask1)
plotRGB(masked_swirnir,stretch='lin',axes=TRUE,main='SWIR+SWIR+NIR',mar=c(1,1,2,1))
#       Geo compositing PCA composite
rlist<-c("B02_20m.jp2","B03_20m.jp2","B04_20m.jp2","B05_20m.jp2","B06_20m.jp2",
         "B07_20m.jp2","B8A_20m.jp2","B11_20m.jp2","B12_20m.jp2")
rlist2<-c("b2","b3","b4","b5","b6","b7", "b8a","b11","b12")
sentinel<-rast(rlist)
names(masked_pca)<-rlist2
```

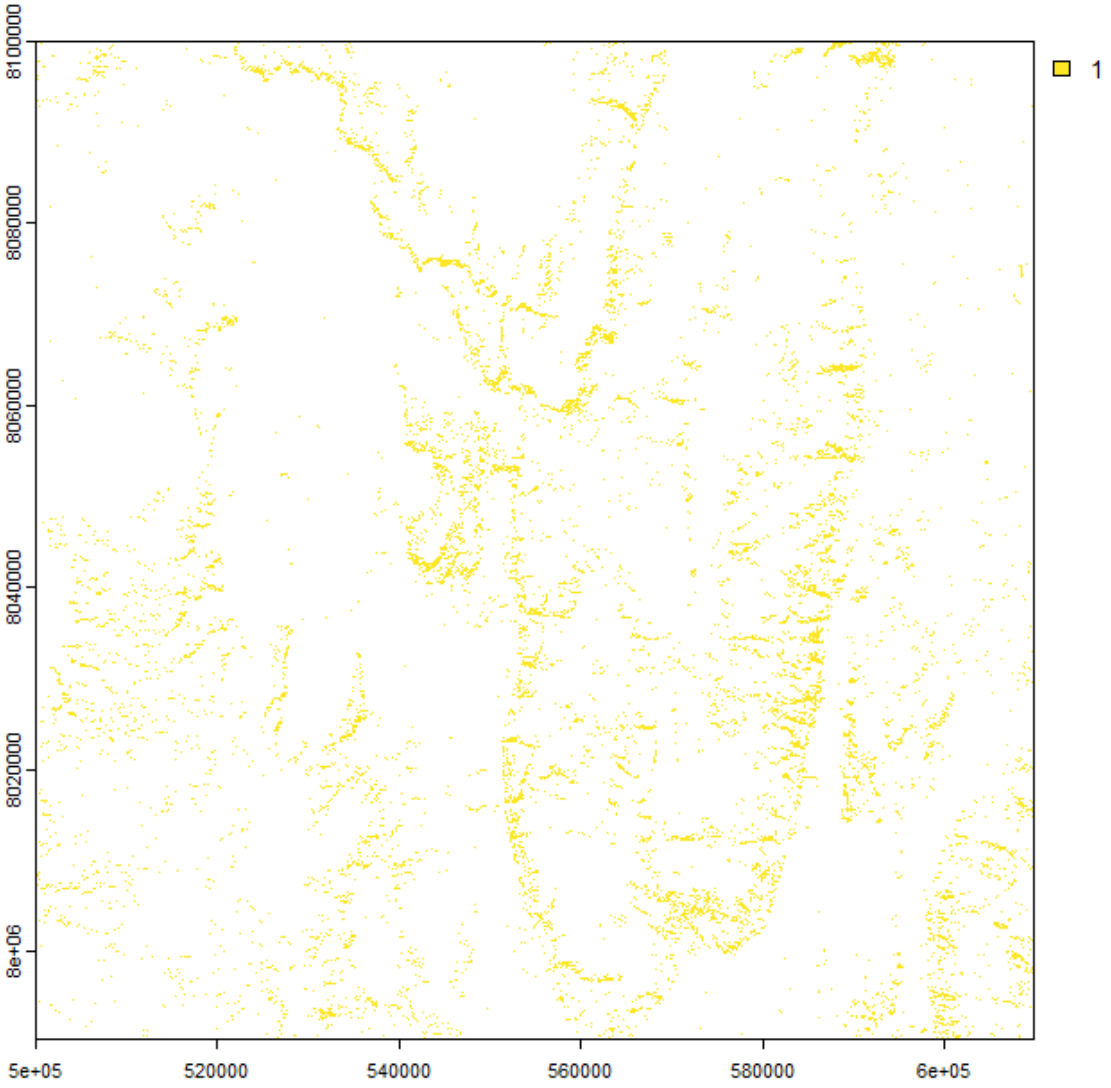
```
set.seed(1)
samples_ <- spatSample(sentinel, 3000000,method="random")# ~10%
pca <- prcomp(samples_, scale = TRUE)
rm(samples_)
pca
pcis<-predict(sentinel,pca,filename='pcis.tif',overwrite=TRUE)
pcaALL<-c(pcis[[1]],pcis[[2]],pcis[[3]],pcis[[4]],pcis[[5]],pcis[[6]],pcis[[7]],
        pcis[[8]] ,pcis[[9]])
pcaALL <- mask(sentinel, mask1)
plotRGB(pcaALL,stretch='lin',r=1,g=2,b=3,axes=TRUE,main='PCA',mar=c(1,1,2,1))
#nomalizing using range maximum value 17089
geobase<-(pcaAll*masked_swirnir)/17089
plotRGB(geobase,stretch='hist',axes=TRUE,main='GEOBASE',mar=c(1,1,2,1))
writeRaster(geobase,'geobase.tif', overwrite=TRUE)

#_____________________________________________________________________________
###############################################################################
#      Unsupervised K-means Classification
#
img<-c(geobase[[1]],geobase[[2]],geobase[[3]],geobase[[4]],geobase[[5]],
       geobase[[6]],geobase[[7]],geobase[[8]],geobase[[9]])
ar <- values(img)
values <- ar
#replacing  NA from vector by 0
ar[is.na(ar)] = 0
Result <- clara(ar,9,samples=500,metric="manhattan",pamLike=T)# 16 minutes
Kmraster9 <- rast (geobase[[1]])
values(kmraster9) <- Result$clustering
#Here we mask the result and extract one of class resulting in 8 classes
kmraster8 <- mask(kmraster9, mask1)
plot(kmraster8, legend=T,axes=F,main='UNSUPERVISED')
writeRaster(kmraster8,'UNSUPERVISED.tif', overwrite=TRUE)
#_____________________________________________________________________________
###############################################################################
#      Supervised SAM Classification
#
library(RStoolbox)
pontos <- data.frame(x = c(517831,582845,549504, 580941,589457,608259,597139, 576555),
             y = c(8079732,8018364,8075490, 8017184,8018611,8048353 ,8008282,8050021))
endmembers <- extract(geobase, pontos)
rownames(endmembers) <- c("litho1","litho2","litho3","litho4","litho5", "litho6",
                          "litho7","litho8")
## Classification based on minimum angles
sam <- sam(geobase, endmembers, angles = FALSE)
plot(sam, legend=T,axes=F,main='SUPERVISED')
writeRaster(sam,'SUPERVISED.tif', overwrite=TRUE)
#_____________________________________________________________________________
```

2- Linear Structures and Inclined Faces from DEM

First, we will create a raster the represents the slopes and crests and after that we will extract the main lineaments using a edge-detection algorithm borrowed from **wmtool**, a library that is not available anymore. I include here the function that we are going to use, they were created by Junji Sugiama and Kayoko Kobayashi.

The following will create the crests and slopes based on the Digital Elevation Model (DEM) of the area.

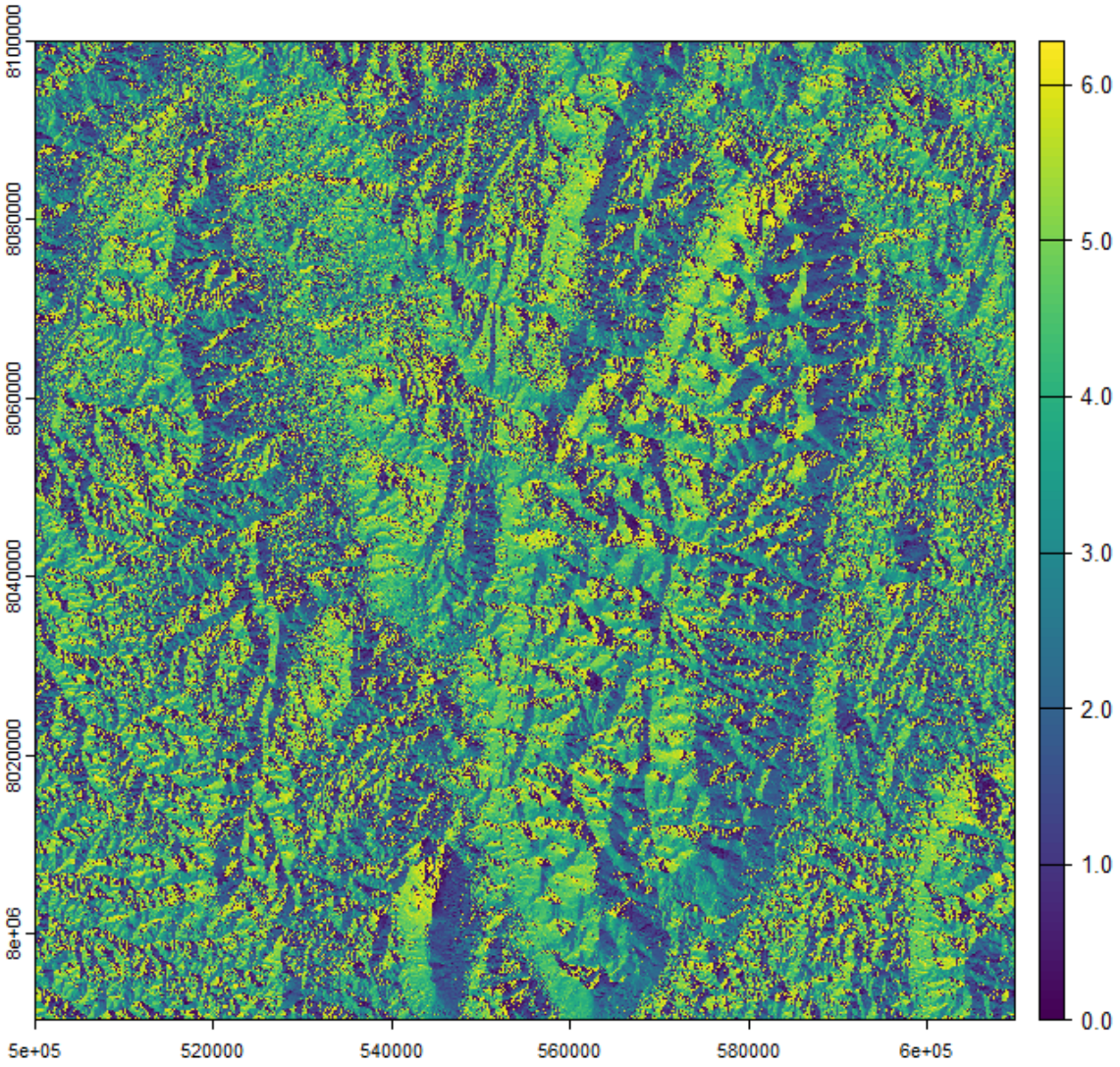
```
library(terra)
#Defining working directory
wd<-'C:/Users/User/Desktop/R algo'
setwd(wd)
dem<-rast('dem.tif')
gradiente<-terrain(dem,v='slope',unit='radians')
aspecto<-terrain(dem,v='aspect',unit='radians')
hshade<-shade(gradiente,aspecto,angle=45,direction=270,normalize= TRUE)
hshade2<-shade(gradiente,aspecto,angle=45,direction=90,normalize= TRUE)
hshade3<-shade(gradiente,aspecto,angle=45,direction=0,normalize= TRUE)
hshade4<-shade(gradiente,aspecto,angle=45,direction=1800,normalize= TRUE)
crests<- hshade + hshade2+ hshade3+ hshade4
crests[crests > 650] <- NA
crests[crests >0] <- 1
plot(crests)
```



Save the raster using:

```
writeRaster(crests,'crests.tif', overwrite=TRUE)
```

Next we will extract the lineaments from the aspect extracted from the dem.



To achieve this, we will need to load the edge-detection related functions

```
################################
# Auxiliary functions
rgb2gray <- function(x, coefs=c(0.30, 0.59, 0.11)) {
 if (is.null(dim(x))) stop("image must have rgb type")
 if (length(dim(x))<3) stop("image must have rgb type")
 if (max(x) <= 1) {
       x <- x*(2^(attr(x,"bits.per.sample"))-1)
 }
 imgdata <- round((coefs[1] * x[,,1] + coefs[2] * x[,,2] + coefs[3] * x[,,3]))
 attr(imgdata, "bits.per.sample")<-attr(x,"bits.per.sample")
 attr(imgdata, "samples.per.pixel") <- 1
 return(imgdata)
}
##############################
noise.filter <- function(x, n=3, method="median") {
if (length(dim(x))>2){warning("data must be grayscale image")
```

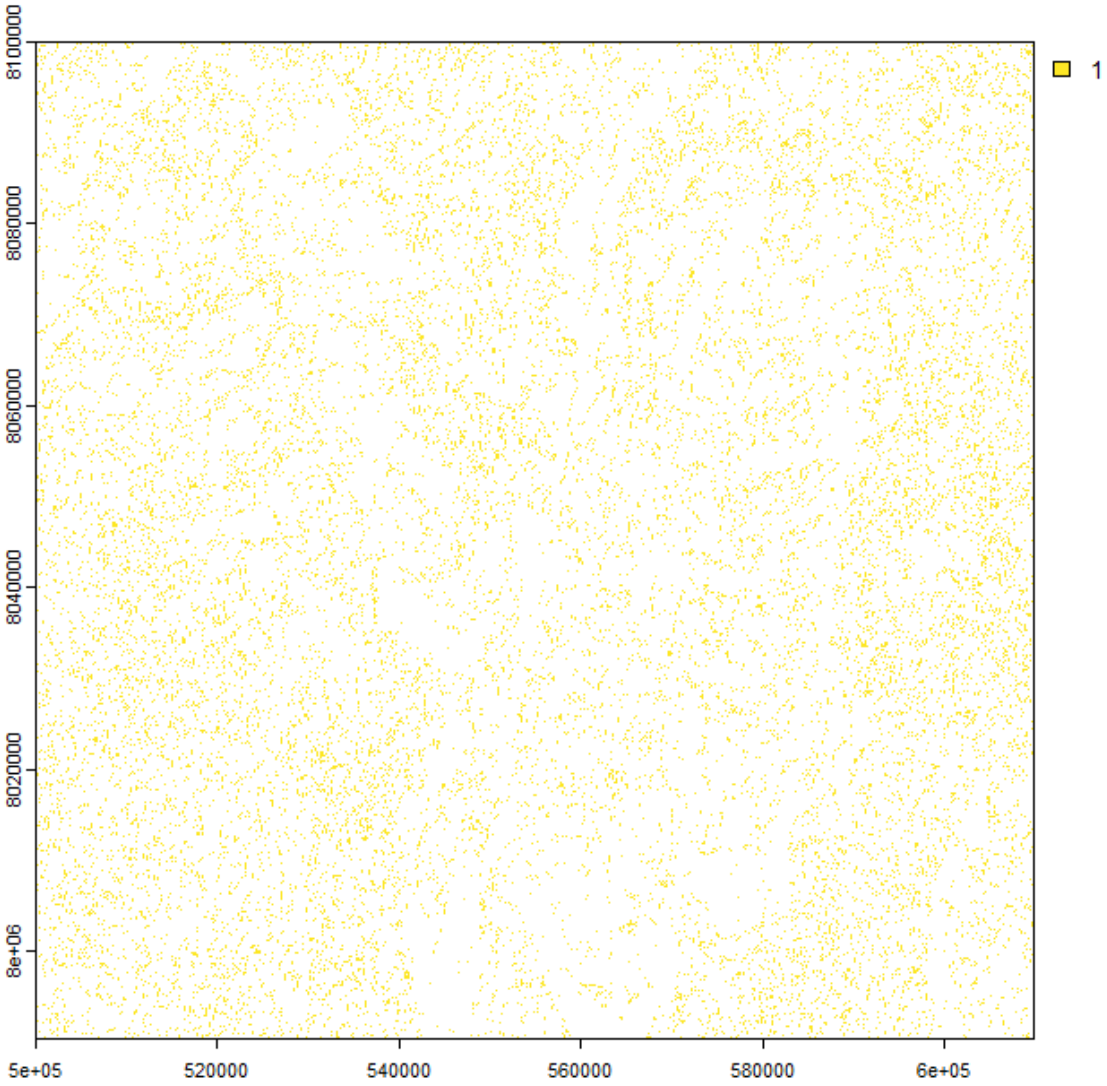
```
} else{
 if(max(x)<=1){
  x <- x*(2^attr(x,"bits.per.sample")-1)
 }
 px <- nrow(x)
 py <- ncol(x)
 img <- x
 f <- n %% 2
 x_edge_plus <- matrix(NA, px+n-1, py+n-1)
 x_edge_plus[(f+1):(f+px),(f+1):(f+py)] <- x
 rasX <- matrix(NA,length(img),(n*n))
 if (method == "median" | method == "mean") {
  for(i in 1:n){
    for(j in 1:n){
     rasX[,(n*(j-1)+i)] <- array(x_edge_plus[(i:(px+i-1)),(j:(py+j-1))])
    }
  }
  if(method == "median"){
        o.img <- apply(rasX,1,median,na.rm=T)
        dim(o.img)<- c(nrow(img),ncol(img))
  } else if(method == "mean"){
        o.img <- apply(rasX,1,mean,na.rm=T)
        dim(o.img)<- c(nrow(img),ncol(img))
  }
 } else if (method == "gaussian"){
        if(n==3){
               gcf<- c(1,2,1,2,4,2,1,2,1)/16
        } else if(n==5){
         gcf<- c(1,4,6,4,1,4,16,24,16,4,6,24,36,24,6,4,16,24,16,4,1,4,6,4,1)/256
        } else {warning("only 3 or 5 for n")}
        rasX <- matrix(NA,length(img),(n*n))
        for(i in 1:n){
         for(j in 1:n){
        rasX[,(n*(j-1)+i)] <- gcf[n*(j-1)+i]*array(x_edge_plus[(i:(px+i-1)),(j:(py+j-1))])
         }
        }
        o.img <- apply(rasX,1,sum,na.rm=T)
        dim(o.img)<- c(nrow(img),ncol(img))
        if(n==3){
         o.img[c(1,nrow(o.img)),c(1,ncol(o.img))] <-
o.img[c(1,nrow(o.img)),c(1,ncol(o.img))]*16/9
        o.img[c(1,nrow(o.img)),2:(ncol(o.img)-1)] <- o.img[c(1,nrow(o.img)),2:(ncol(o.img)-
1)]*16/12
        o.img[2:(nrow(o.img)-1),c(1,ncol(o.img))] <- o.img[2:(nrow(o.img)-
1),c(1,ncol(o.img))]*16/12
 }else if(n==5){
         o.img[c(1,nrow(o.img)),c(1,ncol(o.img))] <-
o.img[c(1,nrow(o.img)),c(1,ncol(o.img))]*256/121
         o.img[c(1,nrow(o.img)),c(2,ncol(o.img)-1)] <- o.img[c(1,nrow(o.img)),c(2,ncol(o.img)-
1)]*256/165
        o.img[c(2,nrow(o.img)-1),c(1,ncol(o.img))] <- o.img[c(2,nrow(o.img)-
1),c(1,ncol(o.img))]*256/165
        o.img[c(1,nrow(o.img)),3:(ncol(o.img)-2)] <- o.img[c(1,nrow(o.img)),3:(ncol(o.img)-
2)]*256/176
        o.img[3:(nrow(o.img)-2),c(1,col(o.img))] <- o.img[3:(nrow(o.img)-
2),c(1,col(o.img))]*256/176
        o.img[c(2,(nrow(o.img)-1)),c(2,(ncol(o.img)-1))] <- o.img[c(2,(nrow(o.img)-
1)),c(2,(ncol(o.img)-1))]*256/189
        o.img[c(2,nrow(o.img)-1),3:(ncol(o.img)-2)] <- o.img[c(2,nrow(o.img)-
1),3:(ncol(o.img)-2)]*256/240
        o.img[3:(nrow(o.img)-2),c(2,ncol(o.img)-1)] <- o.img[3:(nrow(o.img)-
2),c(2,ncol(o.img)-1)]*256/240
 }
 } else {
        print(NULL)
 }
 attr(o.img, "bits.per.sample")<-attr(x,"bits.per.sample")
 attr(o.img, "samples.per.pixel")<-attr(x,"samples.per.pixel")
 out <- round(o.img)
}}
#######################################
```

```
edge.detect <- function(x, thresh1=1, thresh2=15, noise="gaussian", noise.s=3,
method="Canny") {
 if (length(dim(x))>2){warning("data must be grayscale image")
 } else {
  img <- x
  img.n <- noise.filter(img,noise.s, method= noise)
  img.ed <- img[2:(nrow(img)-1),2:(ncol(img)-1)]
  rasSX <- matrix(NA,length(img.ed),9)
  rasSY <- matrix(NA,length(img.ed),9)
  sx <- c(-1,-2,-1,0,0,0,1,2,1)/8
  sy <- c(-1,0,1,-2,0,2,-1,0,1)/8
  for(i in 1:3){
   for(j in 1:3){
    rasSX[,(3*(j-1)+i)] <- sx[3*(j-1)+i]*array(img.n[(i:(nrow(img.n)+i-3)),(j:(ncol(img.n)+j-
3))])
    rasSY[,(3*(j-1)+i)] <- sy[3*(j-1)+i]*array(img.n[(i:(nrow(img.n)+i-3)),(j:(ncol(img.n)+j-
3))])
   }
  }
  img.sx <- apply(rasSX,1,sum)
  img.sy <- apply(rasSY,1,sum)
  dim(img.sx) <- dim(img.sy) <- dim(img.ed)
  img.sxsy <- (img.sx^2+img.sy^2)^0.5          #magnitude of gradient
  if(method=="Sobel"){
   out <- img.sxsy
  } else if(method=="Canny"){
   img.th <- atan(img.sy/img.sx)
   img.th0 <- img.th                          #angles of gradient (0,45,90,135)
   img.th0[abs(img.th0) >= pi*3/8] <- 90
   img.th0[abs(img.th0) <= pi/8] <- 0
   img.th0[img.th0 > pi/8 & img.th0 < pi*3/8] <- 45
   img.th0[img.th0 > -pi*3/8 & img.th0 < -pi/8] <- 135
   #non-maximum suppression
   img.sxsy.bl <- matrix(0,nrow(img),ncol(img))
   img.sxsy.bl[2:(nrow(img)-1),2:(ncol(img)-1)] <- img.sxsy
   rasL <- matrix(NA,length(img.sxsy),9)
   for(i in 1:3){
    for(j in 1:3){
     rasL[,(3*(j-1)+i)] <- array(img.sxsy.bl[(i:(nrow(img.n)+i-3)),(j:(ncol(img.n)+j-3))])
    }
   }
   rasLj <- matrix(0,length(img.sxsy),4)
   rasLj[which(rasL[,5] > rasL[,4] & rasL[,5] > rasL[,6]),3] <- 1
   rasLj[which(rasL[,5] > rasL[,7] & rasL[,5] > rasL[,3]),4] <- 1
   rasLj[which(rasL[,5] > rasL[,2] & rasL[,5] > rasL[,8]),1] <- 1
   rasLj[which(rasL[,5] > rasL[,1] & rasL[,5] > rasL[,9]),2] <- 1
   img.tl <- array(img.sxsy)
   img.tl[which(img.th0==0)] <- img.sxsy[which(img.th0==0)]*rasLj[which(img.th0==0),1]
   img.tl[which(img.th0==45)] <- img.sxsy[which(img.th0==45)]*rasLj[which(img.th0==45),2]
   img.tl[which(img.th0==90)] <-   img.sxsy[which(img.th0==90)]*rasLj[which(img.th0==90),3]
   img.tl[which(img.th0==135)] <- img.sxsy[which(img.th0==135)]*rasLj[which(img.th0==135),4]
   dim(img.tl) <- dim(img.sxsy)
   #hysteresis threshold
   mxth <- sort(img.tl[which(img.tl!=0)])[round(length(img.tl[which(img.tl!=0)])*thresh2/100)]
   mnth <- sort(img.tl[which(img.tl!=0)])[round(length(img.tl[which(img.tl!=0)])*thresh1/100)]
   img.bn0 <- img.bn1 <- matrix(0,nrow(img.tl)+2,ncol(img.tl)+2)
   img.bn0[2:(nrow(img.bn0)-1),2:(ncol(img.bn0)-1)] <- img.bn1[2:(nrow(img.bn0)-
1),2:(ncol(img.bn0)-1)] <- img.tl
   img.bn0[which(img.bn0 <= mnth)] <- 0
   img.bn0[which(img.bn0 > mnth)] <- 1
   img.lb <- matrix(0,nrow(img.tl)+2,ncol(img.tl)+2)
   ren8 <- rbind(c(-1,-1),c(-1,0),c(-1,1),c(0,-1)) #8-connected area
   k <- 0
   for(i in 2:(nrow(img.bn0)-1)){
    for(j in 2:(ncol(img.bn0)-1)){
       z8 <- t(c(i,j)+t(ren8))
       if(img.bn0[i,j]==0){
       }else if(img.bn0[i,j]==1 && sum(img.bn0[z8]==0)==4){
        k <- k+1
        img.lb[i,j] <- k
       }else {
```

```
          zl <- z8[which (img.lb[z8] != 0),,drop=F]
          z.lb <- img.lb[zl]
          z.lb <- sort(z.lb[!duplicated(z.lb)])
          img.lb[i,j] <- z.lb[1]
          if(length(z.lb)>1){
           for(l in 2:length(z.lb)){
            img.lb[which(img.lb==z.lb[l])] <- z.lb[1]
           }
     }
    }
}
}
img.bn <- img.lb
lb <- sort(img.lb[!duplicated(array(img.lb))])
lb <- lb[-1]
for(i in lb){
if(length(which(img.bn1[which(img.lb==i)]>mxth))==0){
 img.bn[which(img.lb==i)] <- 0
}else {
 img.bn[which(img.lb==i)] <- 1
}
}
img.bn <- img.bn[2:(nrow(img.bn0)-1),2:(ncol(img.bn0)-1)]
out <- img.bn
}else {
 print(NULL)
}
}}
```

Execute the code below to generate the raster with the linear structures from the DEM aspect. This execution takes a long time to conclude when the full Sentinel2 scene is used (about 2 hours).

```
resul<-focal(aspecto,w=5, fun=median, na.rm=TRUE)
h<-as.integer(values(resul))
hh<-array(h,c(dim(resul)[1],dim(resul)[2],3))
hh[is.na(hh)]<-0
hi<-rgb2gray(hh)
edg<-edge.detect(hi,thresh1=40, thresh2=60, noise="median", noise.s=5,method='Canny')
r<-rast(nrows=dim(edg)[1],ncols=dim(edg)[2],crs=crs(resul),ext=ext(resul))
a<-as.vector(edg)
final<-setValues(r,a)
final[final==0]<-NA
plot(final)
```



Save the resulting raster using:

```
writeRaster(final,'lines.tif', overwrite=TRUE)
```

In the next section we will integrate the raster created until now into a geological base.

Linear Structures and Inclined Faces full code

```
#############################################################################
#       Sentinel 2 automated structural lineaments extraction (Geological mapping)
#       GDataSystems 2024
#       Script Created by Andre Costa
#       Functions rgb2gray ,noise.filter and edge.detect Authors: Junji Sugiyama,
#                                                                 Kayoko Kobayashi


#############################################################################
#       Loading DEM and extracting gradient aspect hillshade
library(terra)
#Defining working directory
wd<-'C:/Users/User/Desktop/R algo'
```

```
setwd(wd)
dem<-rast('dem.tif')
gradiente<-terrain(dem,v='slope',unit='radians')
aspecto<-terrain(dem,v='aspect',unit='radians')
hshade<-shade(gradiente,aspecto,angle=45,direction=270,normalize= TRUE)
hshade2<-shade(gradiente,aspecto,angle=45,direction=90,normalize= TRUE)
hshade3<-shade(gradiente,aspecto,angle=45,direction=0,normalize= TRUE)
hshade4<-shade(gradiente,aspecto,angle=45,direction=1800,normalize= TRUE)
crests<- hshade + hshade2+ hshade3+ hshade4
crests[crests > 650] <- NA
crests[crests >0] <- 1
writeRaster(crests,'crests.tif', overwrite=TRUE)

################################
# Auxiliary functions
rgb2gray <- function(x, coefs=c(0.30, 0.59, 0.11)) {
 if (is.null(dim(x))) stop("image must have rgb type")
 if (length(dim(x))<3) stop("image must have rgb type")
 if (max(x) <= 1) {
       x <- x*(2^(attr(x,"bits.per.sample"))-1)
 }
 imgdata <- round((coefs[1] * x[,,1] + coefs[2] * x[,,2] + coefs[3] * x[,,3]))
 attr(imgdata, "bits.per.sample")<-attr(x,"bits.per.sample")
 attr(imgdata, "samples.per.pixel") <- 1
 return(imgdata)
}
################################
noise.filter <- function(x, n=3, method="median") {
if (length(dim(x))>2){warning("data must be grayscale image")
} else{
 if(max(x)<=1){
  x <- x*(2^attr(x,"bits.per.sample")-1)
 }
 px <- nrow(x)
 py <- ncol(x)
 img <- x
 f <- n %/% 2
 x_edge_plus <- matrix(NA, px+n-1, py+n-1)
 x_edge_plus[(f+1):(f+px),(f+1):(f+py)] <- x
 rasX <- matrix(NA,length(img),(n*n))
 if (method == "median" | method == "mean") {
  for(i in 1:n){
    for(j in 1:n){
     rasX[,(n*(j-1)+i)] <- array(x_edge_plus[(i:(px+i-1)),(j:(py+j-1))])
    }
  }
  if(method == "median"){
       o.img <- apply(rasX,1,median,na.rm=T)
       dim(o.img)<- c(nrow(img),ncol(img))
  } else if(method == "mean"){
       o.img <- apply(rasX,1,mean,na.rm=T)
       dim(o.img)<- c(nrow(img),ncol(img))
  }
 } else if (method == "gaussian"){
       if(n==3){
              gcf<- c(1,2,1,2,4,2,1,2,1)/16
       } else if(n==5){
        gcf<- c(1,4,6,4,1,4,16,24,16,4,6,24,36,24,6,4,16,24,16,4,1,4,6,4,1)/256
       } else {warning("only 3 or 5 for n")}
       rasX <- matrix(NA,length(img),(n*n))
       for(i in 1:n){
        for(j in 1:n){
       rasX[,(n*(j-1)+i)] <- gcf[n*(j-1)+i]*array(x_edge_plus[(i:(px+i-1)),(j:(py+j-1))])
        }
       }
       o.img <- apply(rasX,1,sum,na.rm=T)
       dim(o.img)<- c(nrow(img),ncol(img))
       if(n==3){
        o.img[c(1,nrow(o.img)),c(1,ncol(o.img))] <-
o.img[c(1,nrow(o.img)),c(1,ncol(o.img))]*16/9
```

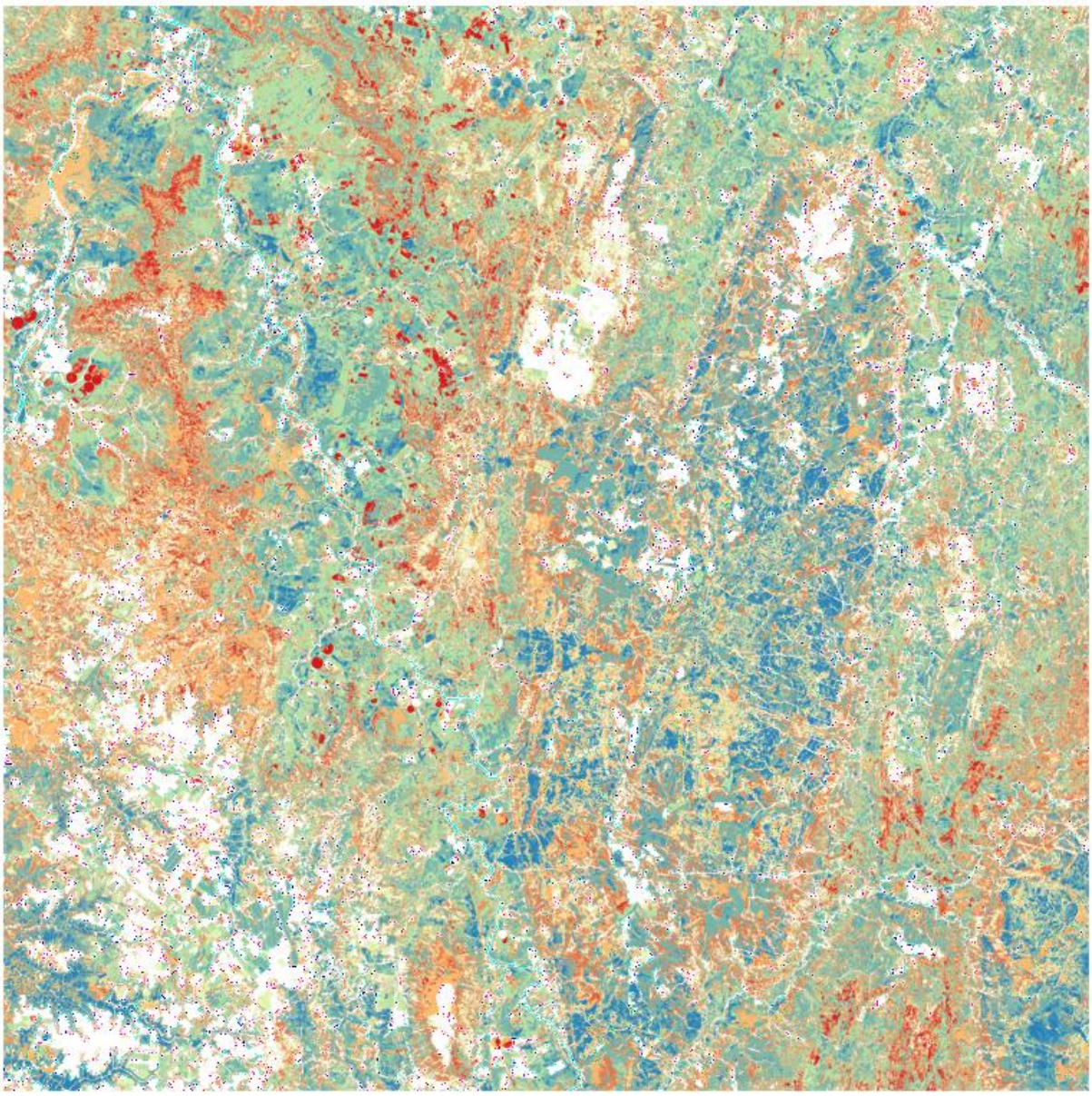
```
        o.img[c(1,nrow(o.img)),2:(ncol(o.img)-1)] <- o.img[c(1,nrow(o.img)),2:(ncol(o.img)-
1)]*16/12
        o.img[2:(nrow(o.img)-1),c(1,ncol(o.img))] <- o.img[2:(nrow(o.img)-
1),c(1,ncol(o.img))]*16/12
 }else if(n==5){
         o.img[c(1,nrow(o.img)),c(1,ncol(o.img))] <-
o.img[c(1,nrow(o.img)),c(1,ncol(o.img))]*256/121
         o.img[c(1,nrow(o.img)),c(2,ncol(o.img)-1)] <- o.img[c(1,nrow(o.img)),c(2,ncol(o.img)-
1)]*256/165
        o.img[c(2,nrow(o.img)-1),c(1,ncol(o.img))] <- o.img[c(2,nrow(o.img)-
1),c(1,ncol(o.img))]*256/165
        o.img[c(1,nrow(o.img)),3:(ncol(o.img)-2)] <- o.img[c(1,nrow(o.img)),3:(ncol(o.img)-
2)]*256/176
        o.img[3:(nrow(o.img)-2),c(1,col(o.img))] <- o.img[3:(nrow(o.img)-
2),c(1,col(o.img))]*256/176
        o.img[c(2,(nrow(o.img)-1)),c(2,(ncol(o.img)-1))] <- o.img[c(2,(nrow(o.img)-
1)),c(2,(ncol(o.img)-1))]*256/189
        o.img[c(2,nrow(o.img)-1),3:(ncol(o.img)-2)] <- o.img[c(2,nrow(o.img)-
1),3:(ncol(o.img)-2)]*256/240
        o.img[3:(nrow(o.img)-2),c(2,ncol(o.img)-1)] <- o.img[3:(nrow(o.img)-
2),c(2,ncol(o.img)-1)]*256/240
 }
 } else {
        print(NULL)
 }
 attr(o.img, "bits.per.sample")<-attr(x,"bits.per.sample")
 attr(o.img, "samples.per.pixel")<-attr(x,"samples.per.pixel")
 out <- round(o.img)
}}
#####################################
edge.detect <- function(x, thresh1=1, thresh2=15, noise="gaussian", noise.s=3,
method="Canny") {
 if (length(dim(x))>2){warning("data must be grayscale image")
 } else {
  img <- x
  img.n <- noise.filter(img,noise.s, method= noise)
  img.ed <- img[2:(nrow(img)-1),2:(ncol(img)-1)]
  rasSX <- matrix(NA,length(img.ed),9)
  rasSY <- matrix(NA,length(img.ed),9)
  sx <- c(-1,-2,-1,0,0,0,1,2,1)/8
  sy <- c(-1,0,1,-2,0,2,-1,0,1)/8
  for(i in 1:3){
   for(j in 1:3){
    rasSX[,(3*(j-1)+i)] <- sx[3*(j-1)+i]*array(img.n[(i:(nrow(img.n)+i-3)),(j:(ncol(img.n)+j-
3))])
    rasSY[,(3*(j-1)+i)] <- sy[3*(j-1)+i]*array(img.n[(i:(nrow(img.n)+i-3)),(j:(ncol(img.n)+j-
3))])
   }
  }
  img.sx <- apply(rasSX,1,sum)
  img.sy <- apply(rasSY,1,sum)
  dim(img.sx) <- dim(img.sy) <- dim(img.ed)
  img.sxsy <- (img.sx^2+img.sy^2)^0.5          #magnitude of gradient
  if(method=="Sobel"){
   out <- img.sxsy
  } else if(method=="Canny"){
   img.th <- atan(img.sy/img.sx)
   img.th0 <- img.th                          #angles of gradient (0,45,90,135)
   img.th0[abs(img.th0) >= pi*3/8] <- 90
   img.th0[abs(img.th0) <= pi/8] <- 0
   img.th0[img.th0 > pi/8 & img.th0 < pi*3/8] <- 45
   img.th0[img.th0 > -pi*3/8 & img.th0 < -pi/8] <- 135
   #non-maximum suppression
   img.sxsy.bl <- matrix(0,nrow(img),ncol(img))
   img.sxsy.bl[2:(nrow(img)-1),2:(ncol(img)-1)] <- img.sxsy
   rasL <- matrix(NA,length(img.sxsy),9)
   for(i in 1:3){
    for(j in 1:3){
     rasL[,(3*(j-1)+i)] <- array(img.sxsy.bl[(i:(nrow(img.n)+i-3)),(j:(ncol(img.n)+j-3))])
    }
   }
```

```
   rasLj <- matrix(0,length(img.sxsy),4)
   rasLj[which(rasL[,5] > rasL[,4] & rasL[,5] > rasL[,6]),3] <- 1
   rasLj[which(rasL[,5] > rasL[,7] & rasL[,5] > rasL[,3]),4] <- 1
   rasLj[which(rasL[,5] > rasL[,2] & rasL[,5] > rasL[,8]),1] <- 1
   rasLj[which(rasL[,5] > rasL[,1] & rasL[,5] > rasL[,9]),2] <- 1
   img.tl <- array(img.sxsy)
   img.tl[which(img.th0==0)] <- img.sxsy[which(img.th0==0)]*rasLj[which(img.th0==0),1]
   img.tl[which(img.th0==45)] <- img.sxsy[which(img.th0==45)]*rasLj[which(img.th0==45),2]
   img.tl[which(img.th0==90)] <-   img.sxsy[which(img.th0==90)]*rasLj[which(img.th0==90),3]
   img.tl[which(img.th0==135)] <- img.sxsy[which(img.th0==135)]*rasLj[which(img.th0==135),4]
   dim(img.tl) <- dim(img.sxsy)
   #hysteresis threshold
   mxth <- sort(img.tl[which(img.tl!=0)])[round(length(img.tl[which(img.tl!=0)])*thresh2/100)]
   mnth <- sort(img.tl[which(img.tl!=0)])[round(length(img.tl[which(img.tl!=0)])*thresh1/100)]
   img.bn0 <- img.bn1 <- matrix(0,nrow(img.tl)+2,ncol(img.tl)+2)
   img.bn0[2:(nrow(img.bn0)-1),2:(ncol(img.bn0)-1)] <- img.bn1[2:(nrow(img.bn0)-
1),2:(ncol(img.bn0)-1)] <- img.tl
   img.bn0[which(img.bn0 <= mnth)] <- 0
   img.bn0[which(img.bn0 > mnth)] <- 1
   img.lb <- matrix(0,nrow(img.tl)+2,ncol(img.tl)+2)
   ren8 <- rbind(c(-1,-1),c(-1,0),c(-1,1),c(0,-1)) #8-connected area
   k <- 0
   for(i in 2:(nrow(img.bn0)-1)){
    for(j in 2:(ncol(img.bn0)-1)){
        z8 <- t(c(i,j)+t(ren8))
        if(img.bn0[i,j]==0){
        }else if(img.bn0[i,j]==1 && sum(img.bn0[z8]==0)==4){
         k <- k+1
         img.lb[i,j] <- k
        }else {
         zl <- z8[which (img.lb[z8] != 0),,drop=F]
         z.lb <- img.lb[zl]
         z.lb <- sort(z.lb[!duplicated(z.lb)])
         img.lb[i,j] <- z.lb[1]
         if(length(z.lb)>1){
          for(l in 2:length(z.lb)){
           img.lb[which(img.lb==z.lb[l])] <- z.lb[1]
          }
    }
   }
}
}
img.bn <- img.lb
lb <- sort(img.lb[!duplicated(array(img.lb))])
lb <- lb[-1]
for(i in lb){
if(length(which(img.bn1[which(img.lb==i)]>mxth))==0){
 img.bn[which(img.lb==i)] <- 0
}else {
 img.bn[which(img.lb==i)] <- 1
}
}
img.bn <- img.bn[2:(nrow(img.bn0)-1),2:(ncol(img.bn0)-1)]
out <- img.bn
}else {
 print(NULL)
}
}}

resul<-focal(aspecto,w=5, fun=median, na.rm=TRUE)
h<-as.integer(values(resul))
hh<-array(h,c(dim(resul)[1],dim(resul)[2],3))
hh[is.na(hh)]<-0
hi<-rgb2gray(hh)
edg<-edge.detect(hi,thresh1=40, thresh2=60, noise="median", noise.s=5,method='Canny')
r<-rast(nrows=dim(edg)[1],ncols=dim(edg)[2],crs=crs(resul),ext=ext(resul))
a<-as.vector(edg)
final<-setValues(r,a)
plot(final)
writeRaster(final,'lines.tif', overwrite=TRUE)
```

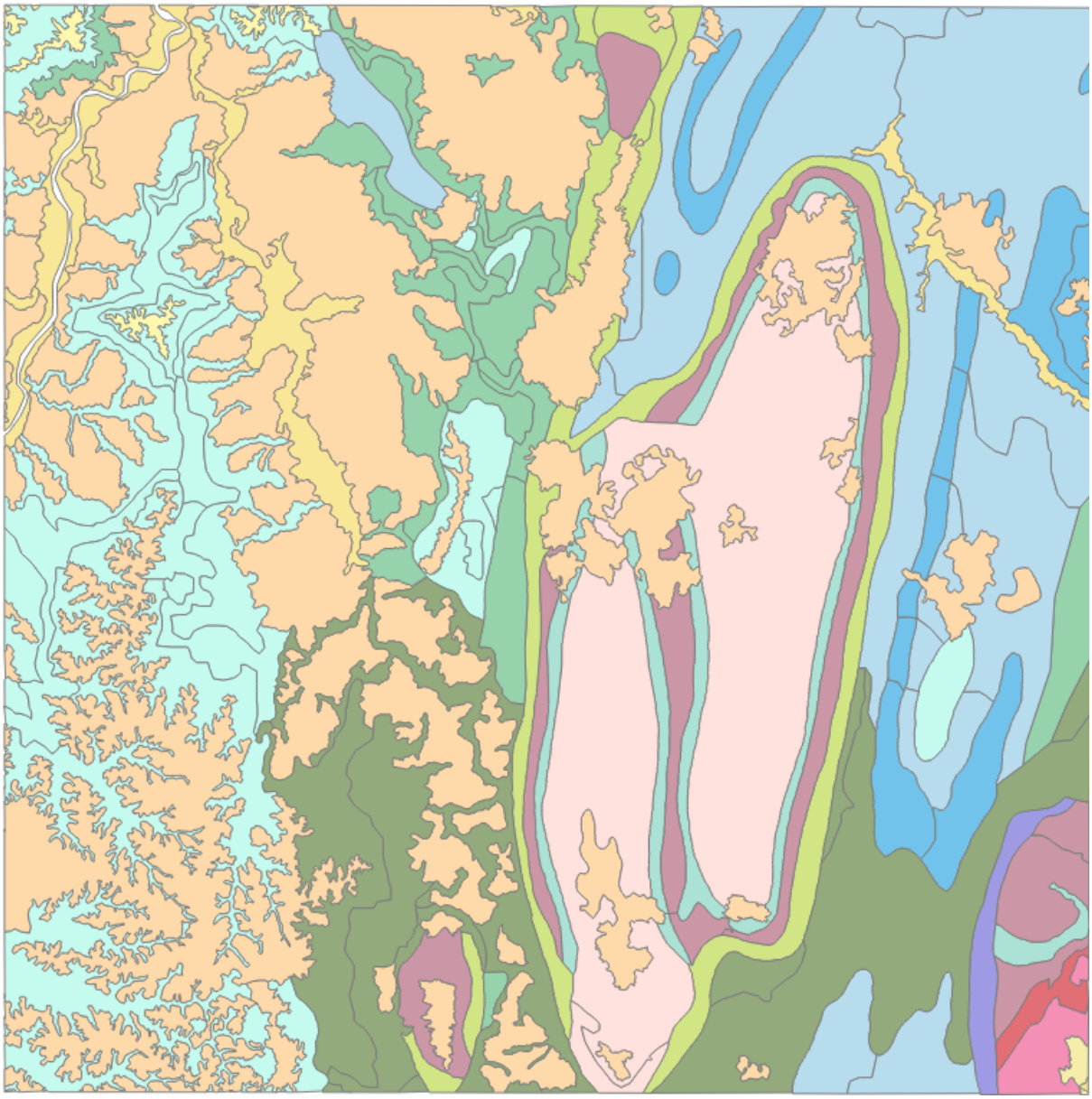
3- Integrated *Spectral Geology Class-Structural Lineaments* Map

Our final product will integrate the one band integrated spectral response (that we can call preliminary geological interpretation) and the structures and lineaments extracted from the DEM.

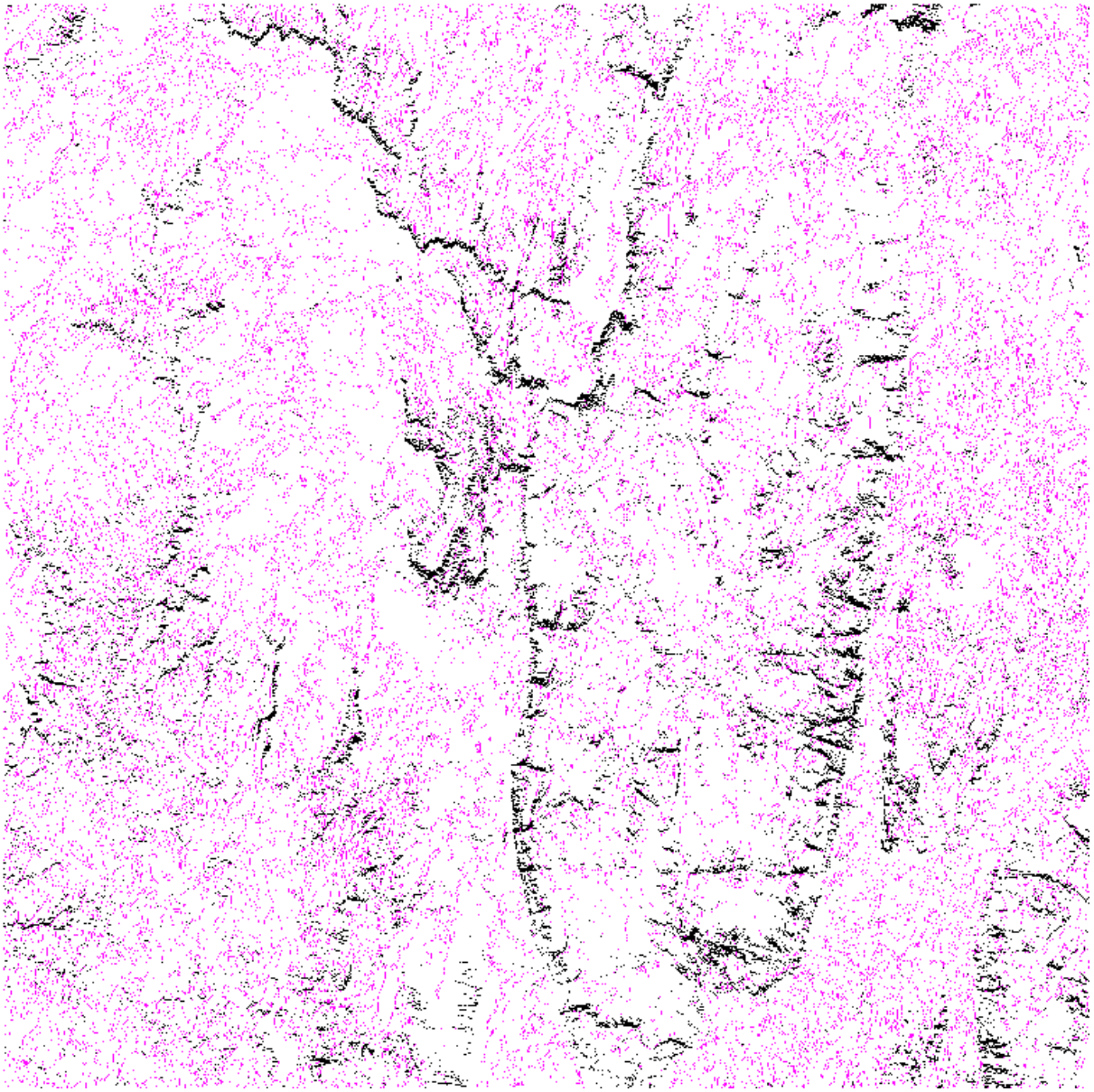
On QGIS the file SUPERVISED.tif will look like:



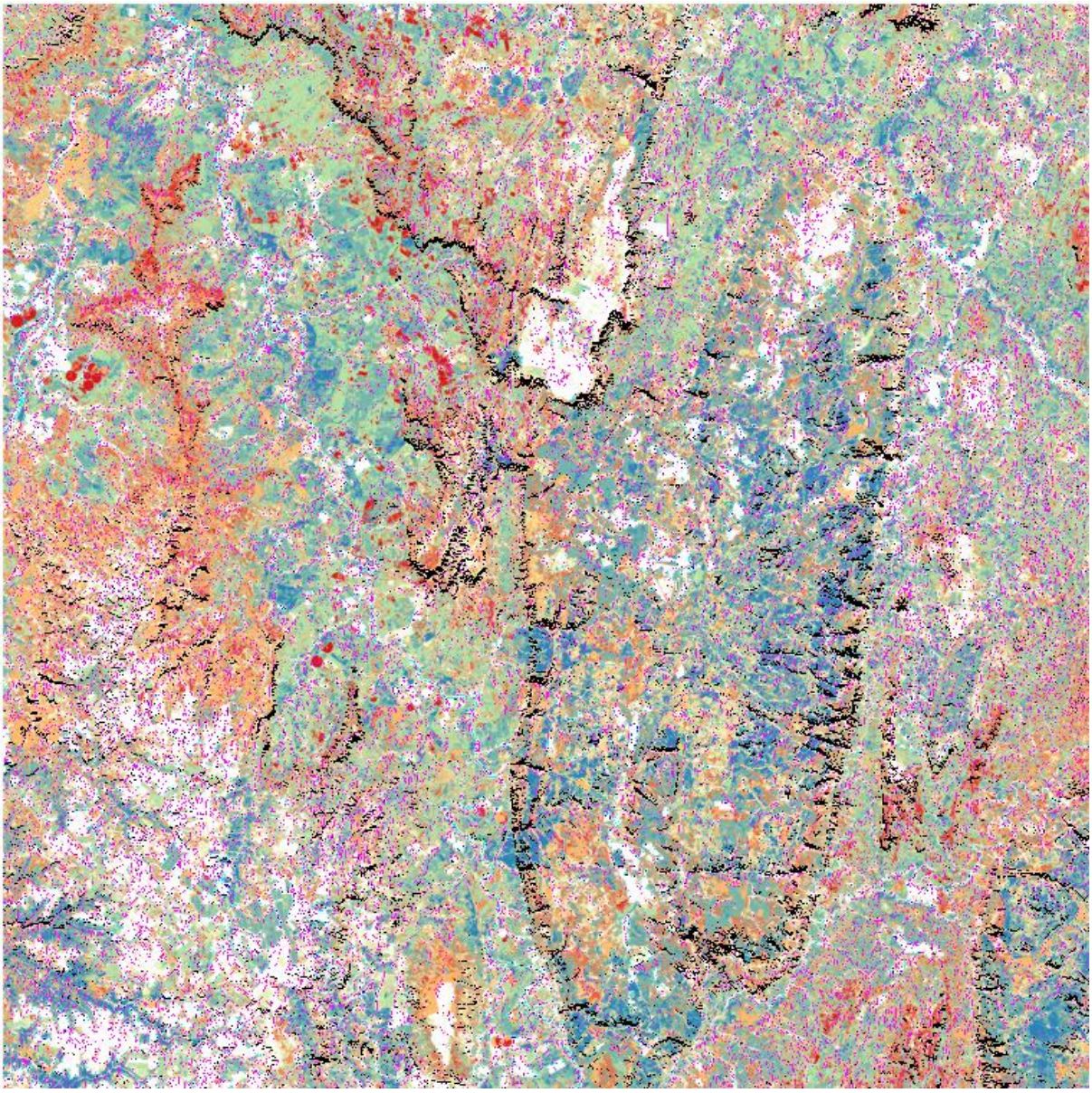
The same area as mapped by the Geological Survey as reference.



The files lines.tif and crests.tif will look like:



The integrated final map will look like:



We demonstrated here that a large area preliminary “geological” and structural map can be generated in a couple of hours using only a set of satellite image and a DEM. This procedure works fine over areas with satisfactory rock exposure and baren soil exposure that still reflect the spectral signature of the original rock.

Areas heavily forested will not work with this procedure only and further processes using geophysical and geochemical data will be necessary. We will cover new processes in the future and also how to integrate these maps in order to create a prospectivity mineral map. Stay tuned!

André Luiz Costa, M.Sc, P.Geo., FAIG