INDICE CALCULATION WITH R LANGUAGE

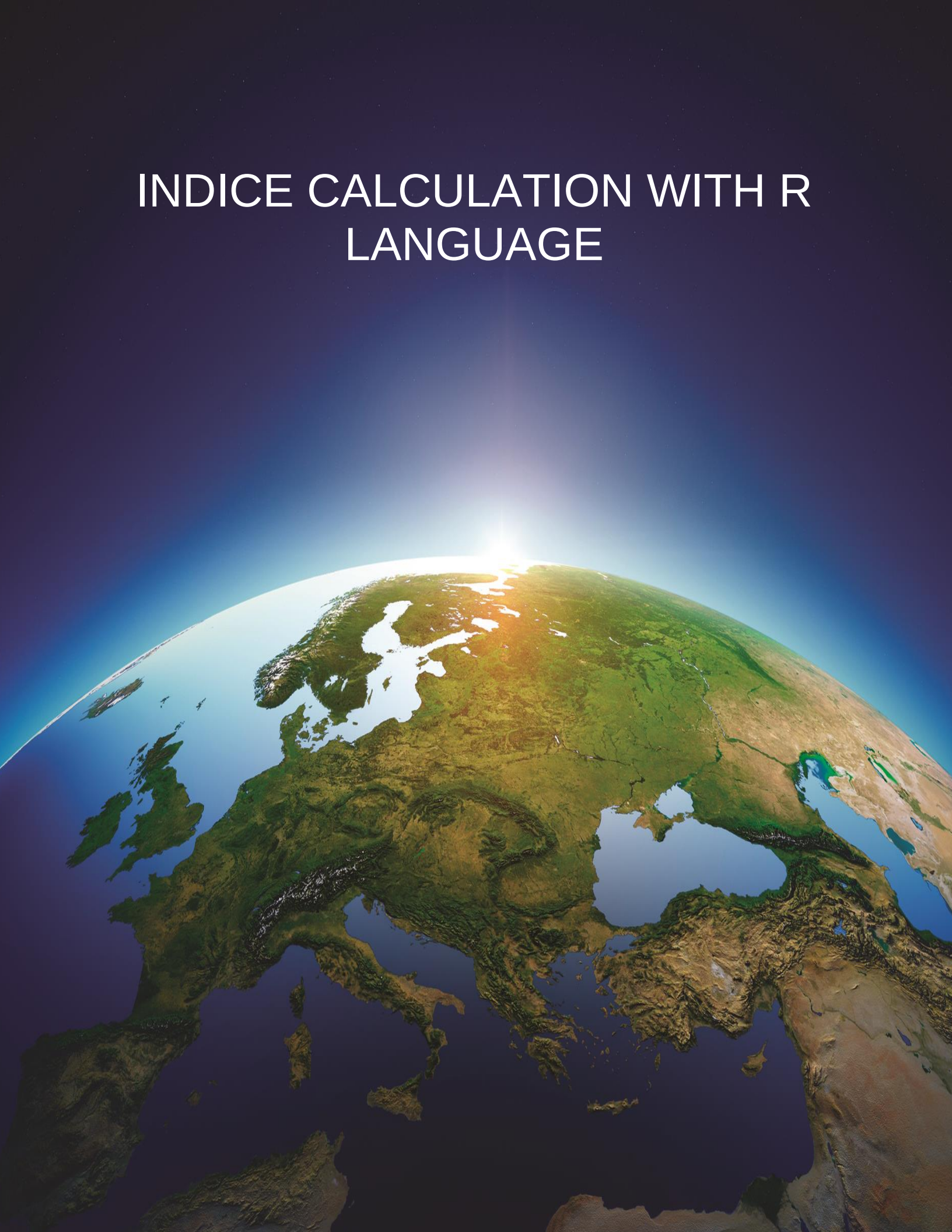
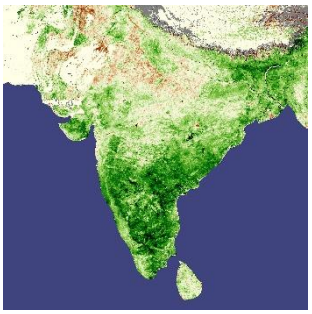


TABLE OF CONTENTS

1. INDICES INTRODUCTION

1.1.Introduction to GIS

A Geographic Information System is a multi-component environment used to



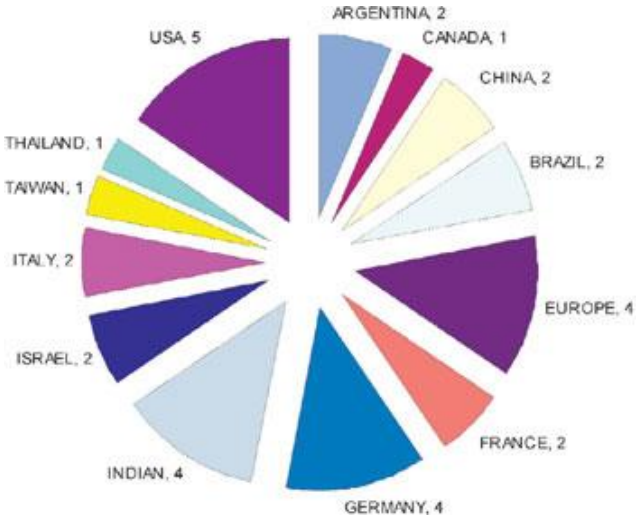
create, manage, visualize and analyze data and its spatial counterpart. It's important to note that most datasets you will encounter in your lifetime can all be assigned a spatial location whether on the earth's surface or within some arbitrary coordinate system (such as a soccer field or a gridded petri dish). So in essence, any dataset can be represented in a GIS: the question then becomes "does it need to be analyzed in a GIS environment?" The answer to this question depends on the purpose of the analysis. What if we are interested in knowing whether countries with a high conflict index score are geographically clustered, does the above table provide us with enough information to help answer this question? The answer, of course, is no. We need additional data pertaining to the geographic location and shape of each country. A map of the countries would be helpful.

1.2. What is Remote Sensing?

Remote sensing is the process of detecting and monitoring the physical characteristics of an area by measuring its reflected and emitted radiation at a distance (typically from satellite or aircraft). Special cameras collect remotely sensed images, which help researchers "sense" things about the Earth. Some examples are: Cameras on satellites and airplanes take images of large areas on the Earth's surface, allowing us to see much more than we can see when standing on the ground. Sonar systems on ships can be used to create images of the ocean floor without needing to travel to the bottom of the ocean. Cameras on satellites can be used to make images of temperature changes in the oceans.

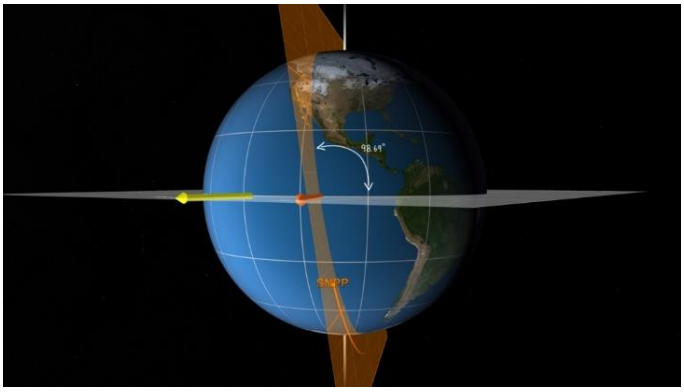
1.3. Remote Sensing Satellites

There are more than 1000 remote sensing satellites available in space, and among these, approximately 593 are from the USA, over 135 are from Russia, and approximately 192 are from China, some widely know satellites that provide data for users are Landsat 5 TM, Landsat 7, Landsat 8, AVHRR/3, MODIS, QuickBird, GOES, Ikonos, SPOT, RADARSAT, ASTER and Sentinel-2 MSI. Satellite remote sensing measuring entering and leaving flux of radiation from top of the atmosphere is one



of the most powerful tools in climate change ecology. NASA at the moment uses 14

satellites of different types of orbits (sun-synchronous and geo-synchronous) with different type of sensors (passive sensors, which record naturally occurring electromagnetic radiation at top of the atmosphere and active sensors which emit electromagnetic radiation toward the Earth and measure its scattering/reflection (http:/earthobservatory.nasa.gov). Satellite remote sensing allowed to study large scale areas and make findings which cannot be done by modeling or field observations. An example is consequence of global sea-rise, which cannot be detected by the models at the moment. We will be using sentinel-2 MSI satellite for our project.



1.4. Sentinel-2 and Its Bands

Sentinel-2 is an Earth observation mission that systematically acquires optical imagery. Sentinel-2 have two twin satellites, Sentinel-2A and Sentinel-2B. The list of band are shown below:

Sentinel-2 Bands	Central Wavelength (μm)	Resolution (m)
Band 1 - Coastal aerosol	0.443	60
Band 2 - Blue	0.49	10
Band 3 – Green	0.56	10
Band 4 – Red	0.665	10
Band 5 - Vegetation Red Edge	0.705	20
Band 6 - Vegetation Red Edge	0.74	20
Band 7 - Vegetation Red Edge	0.783	20
Band 8 - NIR	0.842	10
Band 8A – Narrow NIR	0.865	20
Band 9 - Water vapor	0.945	60
Band 10 - SWIR - Cirrus	1.375	60
Band 11 - SWIR	1.61	20
Band 12 - SWIR	2.19	20

1.5. Indices Definition

A spectral index is a mathematical equation that is applied on the various spectral bands of an image per pixel. So image a Sentinel-2 image. It covers an area of 100Km by 100Km. This translates to approximately 10.000 by 10.000 pixels for the bands that have 10 meters spatial resolution. Half by half for the bands that have 20 meters spatial resolution, etc.

1.6. Types of Indices and Their Formulas

These are the indices that have been used in this project with their corresponding formulas:

Indices Used for This Project	Formula
NDVI	(B8 - B4) / (B8 + B4)
NDWI	(B8 - B12) / (B8+ B12)
SAVI	((1 + L) (B8 - B4)) / (B8 + B4 + L)
EVI2	(2.5 * (B8 – B4)) / ((B8 + B4 + 1.0))
NDRE1	(B8 – B5) / (B8 + B5)
NDRE2	(B8 – B6) / (B8 + B6)
CIRE	(B8 / B5) – 1
NPCRI	(B4 – B2) / (B4 + B2)
LSWI	(B8A – B11) / (B8A + B11)

L is a correction factor which ranges from 0 for very high vegetation cover to 1 for very low vegetation cover. The most typically used value is 0.5 which is for intermediate vegetation cover.

In EVI2 formula 2.5 is the gain factor, Gain is the slope of the sensor response line. All the indices mentioned here are useful for vegetation index.

A Vegetation Index (VI) is a spectral transformation of two or more bands designed to enhance the contribution of vegetation properties and allow reliable spatial and temporal inter-comparisons of terrestrial photosynthetic activity and canopy structural variations.

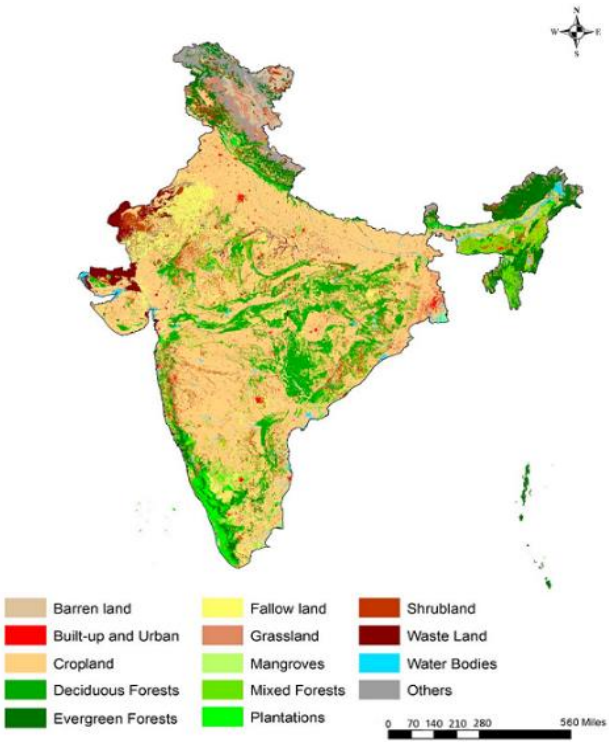
There are many Vegetation Indices (VIs), with many being functionally equivalent. Many of the indices make use of the inverse relationship between red and near-infrared reflectance associated with healthy green vegetation. Since the 1960s scientists have used satellite remote sensing to monitor fluctuation in vegetation at the Earth's surface. Measurements of vegetation attributes include leaf area index (LAI), percent green cover, chlorophyll content, green biomass and absorbed photosynthetically active radiation (APAR).

2. GIS WITH R LANGUAGE

2.1. GIS in R Language

R has a full library of tools for working with spatial data. This includes tools for both vector and raster data, as well as interfacing with data from other sources (like ArcGIS) and making maps.

My own interest in coding and R began with my desire to dip my toes into geographic information systems (GIS) and create maps of an early modern correspondence network. The goal of this post is to introduce the basic landscape of working with spatial data in R from the perspective of a non-specialist. Since the early 2000s, an active community of R developers has built a wide variety of packages to enable R to interface with geographic data. The extent of the geographic capabilities of R is readily apparent from the many packages listed in the CRAN task view for spatial data.



2.2. Understanding ‘raster’ Concept

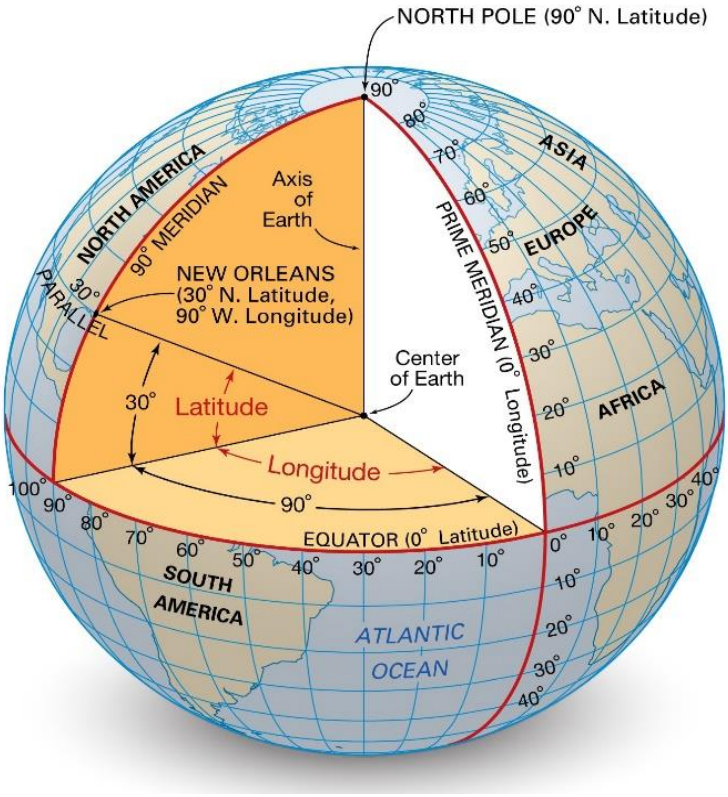
Raster or "gridded" data are data that are saved in pixels. In the spatial world, each pixel represents an area on the Earth's surface. For example in the raster below, each pixel represents a particular land cover class that would be found in that location in the real world.

Raster data is like any image. Although it may portray various properties of objects in the real world, these objects don’t exist as separate objects; rather, they are represented using pixels of various values which are assigned a color.

2.3. Introduction to ‘raster resampling’ with R

A typical workflow for earth analysts always will have to do with several spatial data sources (e.g. Sentinel, MODIS and Landsat) each one with a particular extent and pixel resolution. Resampling is the technique used to homogenize these spatial attributes.

What resampling does is to take randomly drawn (sub) samples of the sample and calculate the statistic from that (sub) sample. Do this enough times and you can get a distribution of statistic values that can provide an empirical measure of the accuracy/precision of the test statistic, with less rigid assumptions



2.4. What is CRS?

A coordinate reference system (CRS) refers to the way in which spatial data that represent the earth's surface (which is round / 3 dimensional) are flattened so that you can "Draw" them on a 2-dimensional surface. However each using a different (sometimes) mathematical approach to performing the flattening resulting in different coordinate system grids (discussed below). These approaches to flattening the data are specifically designed to optimize the accuracy of the data in terms of length and area (more on that later too). To define the location of something you often use a coordinate system. This system consists of an X and a Y value located within a 2 (or more) -dimensional space.

2.5. Downscaling of Sentinel-2 Bands.

Downscaling has an important role to play in remote sensing. It allows prediction at a finer spatial resolution than that of the input imagery, based on either (i) assumptions or prior knowledge about the character of the target spatial variation coupled with spatial optimization, (ii) spatial prediction through interpolation or (iii) direct information on the relation between spatial resolutions in the form of a regression model.

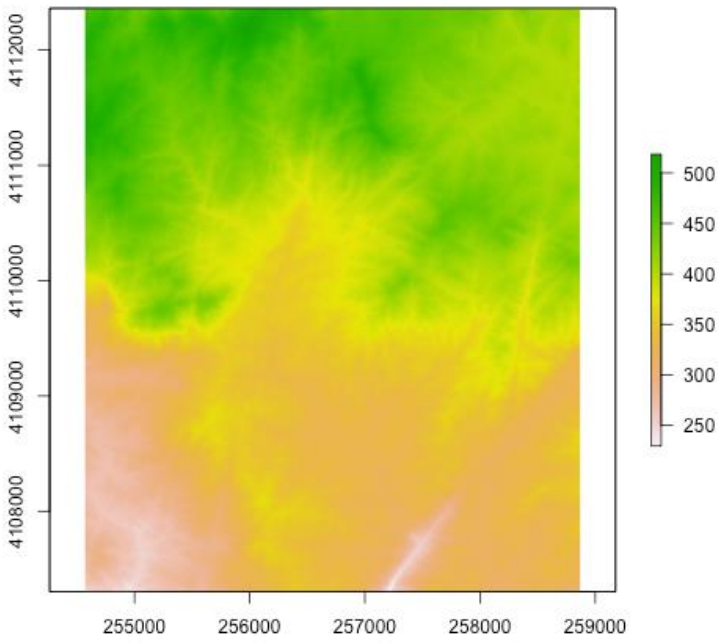
2.6. Upscaling of Sentinel-2 Bands.

Over the last decade, remote-sensing products found their use in numerous public applications and scientific research. Frequent validation of remote-sensing products is necessary to ensure the quality and accuracy. An important step in this process is the upscaling from field measurements of leaf area index (LAI) or other biogeophysical variables to the scale of a high-resolution satellite image. Unlike other vegetation types, semi-arid woodlands are often characterized by a distinct vegetation pattern, a low vegetation cover and site variations in bare soil reflectance which influence the upscaling process.

3. INDICES CALCULATION USING R

3.1. About Raster and RSToolbox Libraries

Raster and RSToolbox libraries are used in the project for getting data from the sentinel-2 band files and calculating indices using some of the inbuilt functions, and writing data to the new GeoTiff files. In Raster library the function used in the program are raster() and wirteraster() on the other hand RSToolbox is used for mathematical calculation that raster library do.



3.2. File Handling in R

In this calculation there are some function which are used for getting data from files and writing data to the files, the function are:

list.files()
unlink()

list.files() : This file function gets all the files in the specified directory, it takes directory, pattern as parameters, by default the pattern will be NULL.

Unlink() : This function deletes the specified file, this function takes full file directory are a parameter and deletes the specified file. This is useful for deleting temporary files that are created while using rasters in the program.

3.3. Indice Calculation Code Explanation

- The summary of the program is, using the code head over to the bands folder of the downloaded ".SAFE" files first, then looping the indices defined in the given list, so based on the number of indices present in the list the code will loop that many times. In each and every iteration it checks which indice is on the loop through `if else()` function then it will calculate based on that indice formula provided in the `if else()`, after one indice calculated then it will store the output generated into a GeoTiff file in the output directory defined. Then it jumps to another indice, after all the indices are done for one folder, the inner loop will break and the outter loop will open another ".SAFE" folder and perform this again.

```r
library(raster)
library(RStoolbox)

main_dir <- "D:/TEST"   #edit this directory to satellite downloaded tile files
resulting_dir <- paste(main_dir,"INDICES/",sep="/")
setwd(main_dir)
dir.create(resulting_dir,recursive = FALSE,showWarnings = TRUE,mode = 0777)
folders <- list.dirs(main_dir, recursive=FALSE)
indices <- c("evi2","lswi","ndvi","ndwi","npcri","savi")
for(indice in indices){
    dir.create(paste(resulting_dir,indice,sep="/"),showWarnings = TRUE,recursive = FALSE,mode = 0777)
}
```

- As I have already explained about the libraries “raster” and “RSToolbox” earlier, they are used for the data calculation
- main_dir is the directory that contains all the downloaded “.SAFE” folders.resulting_dir contains the directory where the calculated indices should be saved. setwd(main_dir) function sets the given directory as current working directory. dir_create() creates a folder with the given directory path from resulting_dir
- list.dirs() gets list of directories under the given path and saved to folder variable which will be used for future code
- c() is used to define vector, here indices names will be stored this vector.
- Then using for() loop and dir.create() functions we will create the folders in the given output location with names taken from the predefined indices name vector.

```r
for(i in 1:length(folders)){
    if(substr(folders[i],nchar(folders[i])-3,nchar(folders[i])) == "SAFE"){
        temp_dir <- paste(folders[i],"GRANULE",sep="/")
        temp_list <- list.dirs(temp_dir,recursive="FALSE")
        temp_dir <- paste(temp_list[1],"IMG_DATA/",sep="/")
        temp_files <- list.files(temp_dir, pattern = "\\.jp2$")
        temp_bands_list <- vector()
        for(j in 1:length(temp_files)){
            if(substr(temp_files[j],nchar(temp_files)-6,nchar(temp_files[j])-4) != "TCI"){
                temp_bands_list <- append(temp_bands_list,temp_files[j])
            }
        }
```

- Now we discussed about folders variable earlier, This variable contains all the folders in the given location, those folders contains the band file of sentinel-2 which we need so using for() loop we will select 1 folder for each iteration.
- The band files will be in a folder called “IMG_DATA” we will find that folder using function list.dirs(), list.files() and paste().

- At line 17 you can see we got into the sub-folders where band files are there after that in line 18 we will be getting all the band file into a list using the function list.files() which takes path and pattern as variable path means the directory to be scanned, pattern means which types of files you want like .jp2, .py, .r so in our case the band files are in jp2 format so we use .jp2 to get all the files.
- At the end of the band files there will be a waste jp2 file which we don't need so in order to exclude it we will use for() loop and if else(), we loop through each file we scanned and using if else() we will check if it is the file we want or not, if it is the file we want it will be added into a new list variable called temp_band_list, if it is the files we don't want then it won't be appended to the new list variable.

```
25      print(paste("----Selected Folder : ",paste(folders[i],"----",sep=""),sep=""))
26      for(j in 1:length(indices)){
27          indice <- indices[j]
28          print(paste("calculating ",paste(indices[j],paste(" for",substr(temp_bands_list[1],1,nchar(temp_bands_list[1])-8),sep=""),sep=""),sep="")
29          if(indices[j] == "ndvi" || indices[j] == "ndwi" || indices[j] == "savi" || indices[j] == "evi2"){
30              nir <- raster(paste(temp_dir,temp_bands_list[8],sep="",collapse = NULL))
31              red <- raster(paste(temp_dir,temp_bands_list[4],sep="",collapse = NULL))
32              if(indices[j] == "ndvi"){
33                  formula <- (nir-red)/(nir+red)
34              }else if(indices[j] == "ndwi"){
35                  formula <- (red-nir)/(red+nir)
36              }else if(indices[j] == "savi"){
37                  L <- 1.5
38                  formula <- (nir-red)/(nir+red+L)*(1.0+L) # (nir-red)*(1.0+L)/(nir+red+L) ?
39              }else if(indices[j] == "evi2"){
40                  formula <- 2.4*(nir-red)/(nir+red+1.0)
41              }else{
42                  print("Cannot define Indice Name!")
43                  break
44              }
45          }else if(indices[j] == "ndre1" || indices[j] == "ndre2"){
46              nir <- raster(paste(temp_dir,temp_bands_list[8],sep="",collapse = NULL))
47              if(indices[j] == "ndre1"){
48                  re <- raster(paste(temp_dir,temp_bands_list[5],sep="",collapse = NULL))
49                  formula <- (nir-re)/(nir+re)
50              }else if(indices[j] == "ndre2"){
51                  re <- raster(paste(temp_dir,temp_bands_list[6],sep="",collapse = NULL))
52                  formula <- (nir-re)/(nir+re)
53              }else{
54                  break
55              }
56          }else if(indices[j] == "cire"){
57              nir <- raster(paste(temp_dir,temp_bands_list[8],sep="",collapse = NULL))
58              re <- raster(paste(temp_dir,temp_bands_list[5],sep="",collapse = NULL))
59              formula <- (nir/re)-1
60          }else if(indices[j] == "npcri"){
61              red <- raster(paste(temp_dir,temp_bands_list[4],sep="",collapse = NULL))
62              blue <- raster(paste(temp_dir,temp_bands_list[2],sep="",collapse = NULL))
63              formula <- (red-blue)/(red+blue)
64          }else if(indices[j] == "lswi"){
65              nnir <- raster(paste(temp_dir,temp_bands_list[13],sep="",collapse = NULL))
66              swir <- raster(paste(temp_dir,temp_bands_list[11],sep="",collapse = NULL))
67              formula <- (nnir-swir)/(nnir+swir)
68          }else{
69              break
70          }
```

- After checking the band files we will be looping indices from the predefined variable indices as we talked in summary. For user understanding we will be displaying a message saying which folder has been selected from all the ".SAFE" folders using print() function.

- After printing the message the indices loop will begin and takes one variable from the start of the list from the indices variable, then again prints message saying which indice is being calculated using print() function.
- Now we will check the indice that was selected in the loop, for each and every indice I wrote a formula in if else() so that the selected indice calculation will be done and then it breaks the program and repeats the process by selecting another indice from the indices variable, this process continues until it reaches the end of the indices list variable.

```
71              file_write_path <- paste(paste(resulting_dir,indice,sep=""),paste(substr(temp_bands_list[1],1,nchar(temp_bands_list[1])-8),".tif",sep=""),sep="/",collapse = NULL)
72              print(file_write_path)
73              writeRaster(x = formula,
74                          filename= file_write_path,
75                          format = "GTiff",
76                          datatype='FLT4S'
77              )
78              r_temp_dir <- paste(tempdir(),"raster",sep="/")
79              if(dir.exists(r_temp_dir)){
80                  r_temp_file <- list.files(r_temp_dir,pattern=NULL,full.names=TRUE)
81                  for(rtmpfile in r_temp_file){
82                      unlink(rtmpfile,recursive=FALSE,force=TRUE)
83                  }
84              }
85          }
86      }
87  }
```

- While calculating the certain indice in done then the calculated data will be stored in the respective folder in the indices folders that have been created on the start of this program under GeoTiff file extension.
- Using writeRaster() function from the library(raster) which will take x = formula, filename, format and datatype as input, the formula variable contains the calculated data of the indice file selected in the indice for loop. The filename will be the name of the file which we want to create. Format will be defined as “Gtiff” which stands for GeoTiff file. Datatype will be FLT4S because the data we will be dealing with are mostly decimal type data.
- When file creation is over we will be clearing the temporary files that are used for creating the Gtiff file because we are dealing with 500mb data for each file, if we won’t clear the temp file after their expiration data user storage will be increased which will slow the program.
- The tempdir() will return the path where R stores the temporary files, we will be checking if any files exist in the given tempdir() using dir.exist() which will take path as input and outputs FLASE if the directory doesn’t exist, and outputs TRUE if given directory exist, after the checking will be deleting all the files in that directory using the unlink() function which takes path as variable and deletes any file under that path so by using this in loop all the expired files will be deleted from the R temporary folder this will be done each and every time an indice is calculated

4. MODIFYING CRS

4.1. Different CRS

As we already talked about what is CRS is, now we will be understanding differences between some common CRS's. Some of the common CRS used in remote sensing are as follows:

- WGS84 (EPSG: 4326)
- NAD83 (EPSG:4269)
- NAD27 (EPSG: 4267)
- UTM

WGS84 (EPSG: 4326):

- The IOGP's EPSG Geodetic Parameter Dataset is a collection of definitions of coordinate reference systems and coordinate transformations which may be global, regional, national or local in application. The EPSG Geodetic Parameter Dataset is maintained by the Geodesy Subcommittee of the IOGP Geomatics Committee.
- WGS 84 -- WGS84 - World Geodetic System 1984, used in GPS
- Coordinate system: Ellipsoidal 2D CS. Axes: latitude, longitude. Orientations: north.
- Horizontal component of 3D system. Used by the GPS satellite navigation system and for NATO military geodetic surveying.
- This is used for worldwide mapping, so this will be out desired CRS

NAD83 (EPSG: 4269):

- Longitude is POSITIVE EAST. The adjustment included connections to Greenland and Mexico but the system was not adopted there. For applications with an accuracy of better than 1m replaced by NAD83 (HARN) in the US and PRVI and by NAD83 (CSRS) in Canada.
- Method: Geocentric translations (geog2D domain)
- This CRS is only limited to some states and countries only, so this won't be the best CRS for out project.

NAD27 (EPSG: 4267):

- Note: this CRS includes longitudes which are POSITIVE EAST. Replaced by NAD27 (76) (code 4608) in Ontario, CGQ77 (code 4609) in Quebec, Mexican Datum of 1993 (code 4483) in Mexico, NAD83 (code 4269) in Canada (excl. Ontario & Quebec) & USA.
- Coordinate system: Ellipsoidal 2D CS. Axes: latitude, longitude. Orientations: north, east. UoM: degree
- Unit: degree (supplier to define representation)
- This CRS is also not usable for out project, this lacks the worldwide usage for other systems like GPS.

UTM:

- The Universal Transverse Mercator (UTM) is a map projection system for assigning coordinates to locations on the surface of the Earth.
- Like the traditional method of latitude and longitude, it is a horizontal position representation, which means it ignores altitude and treats the earth as a perfect ellipsoid.
- However, it differs from global latitude/longitude in that it divides earth into 60 zones and projects each to the plane as a basis for its coordinates.
- Specifying a location means specifying the zone and the x, y coordinate in that plane. The projection from spheroid to a UTM zone is some parameterization of the transverse Mercator projection.
- The parameters vary by nation or region or mapping system.

. So the main CRS's have been explained one by one, you certainly understand why we will be using EPSG: 4326, in case you didn't it's because this is the worldwide usable CRS, this CRS is being used in GPS on all the devices which support GPS system. That makes this CRS the worldwide, and also easily understandable.

Our jp2 files from the ".SAFE" folders will be in this CRS format so after calculating and storing the data into GeoTiff life this CRS will be not changed.

4.2. Why Modifying CRS

We know what is a CRS is and we discussed about 4 most used CRS among world and we already came to conclusion that EPSG: 4326 is the most used CRS worldwide. We will be modifying the CRS for the output GeoTill files because we cannot retrieve the longitude and latitude for the default CRS which is in UTM

4.3. Modifying CRS Code Explanation

- Summary of the code will be, taking all the indices calculated Gtiff files from the directory then looping through each indice for an iteration, checking whether the scanned indice directory is really the indice or not, if TRUE then changing CRS for the Gtiff and storing in the directory created on the program start.

```
1  library(raster)
2  library(sf)
3
4  dir <- "D:/JOB/GIS"
5  indice_list <- list.dirs(dir,recursive=FALSE,full.names=FALSE)
6  copy_dir <- paste(dir,"CRS_CHANGED/",sep="/")
7  dir.create(copy_dir,showWarnings=FALSE,recursive=FALSE,mode=0777)
8  main_indices <- c("ndvi","ndwi","savi")
```

- Initializing library(raster) and library(sf), library(sf) is included when we are performing or modifying anything which is special data set, now that these Gtiff files are special data we included this library
- dir variable holds the path to the directory where the calculated indice files are.
- indice_list variable holds the list of folders in the directory specified path in dir variable using the function list.dirs(). copy_dir variable will have the path to the directory where the CRS modified files will be saved, and using dir.create() we will create a directory for for the CRS modified files.
- main_indices variable will now contain the indices vector list which are going to be modified

```
for(indice in indice_list){
  if(indice %in% main_indices ){
    indice_dir <- paste(dir,indice,sep="/")
    dir.create(paste(copy_dir,indice,sep="/"),recursive=FALSE,mode=0777,showWarnings=FALSE)
    indice_dir_list <- list.files(indice_dir,pattern="\\.tif$",full.names=TRUE)
```

- now we will loop through the folder names that are defined in the variable indice_list, then check if the folder name is equal to any indices specified in the variable main_indices
- if False we will leave that iteration, if it True then we will scan all the jp2 files in that folder into variable indice_dir_list using list.files() which takes path and pattern as input and outputs the desired files we need from that path with desired file patterns.

```
  for(a_file in indice_dir_list){
    indice_one_file_name <- substr(a_file,nchar(indice_dir)+2,nchar(a_file))
    r <- raster(a_file)
    r2 <- projectRaster(r,crs="+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
    writeRaster(r2,filename=paste(copy_dir,paste(indice,indice_one_file_name,sep="/"),sep=""),overwrite=TRUE)
    rtempdir <- paste(tempdir(),"raster",sep="\\")
```

- When the files are scanned into variable indice_dir_list, we will retrieve one file for one iteration using for() loop, then using substr() we will get the name of the file not the full path into a variable called indice_one_file_name
- Using raster() we will be converting the file into raster object and store the data into r variable, after that we will be create a raster r2 with same raster data as the raster object r but with a different CRS string +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
- Now we will be writing the newly created raster object r2 with different CRS into a file with the help of writeRaster() function which will take raster object and filename as the input and output the file with the raster object data.

```
    if(dir.exists(rtempdir)){
      rtempdir_files <- list.files(rtempdir,pattern=NULL,full.names=TRUE)
      for(temp_file in rtempdir_files){
        unlink(temp_file,force=FALSE)
      }
    }else{
      print("Raster Directory Doesn't exist!")
    }
    cat(paste("Projections for ",indice_one_file_name,": \n"))
    print(raster(paste(paste(copy_dir,indice,sep=""),indice_one_file_name,sep="/")))
    cat("\n ---------------------------------- \n")
  }
 }
 cat("\n")
}
```

- In the before picture rtempdir variable stores the path to the temporary directory of R, using that variable we will loop through each and every folder created by R using for() loop, and check if that folder contains inner file, if True we will be deleting those files using unlink() function, if False it will print “Raster Directory Doesn’t exist!”.
- After this whole thing is completed we will be writing a cat() function to output which file CRS is converted and its location for user understanding.
- This whole process repeats until the loop reaches end of the list of indices variable main_indices

5. MOSAICING

5.1 Explanation about Mosaicing

Mosaicing is one of the techniques of image processing which is useful for tiling digital images. Mosaicing is blending together of several arbitrarily shaped images to form one large radio metrically balanced image so that the boundaries between the original images are not seen

Any number of geocoded images can be blended together along user-specified cut lines (polygons). Mosaicing is a special case of geometric correction where registration takes place in the existing image. If ground control points (GCP) are collected the input image is transformed according to the derived polynomial into the output image.

According to our project we are using mosaicing because of only reason that it will take less time if all the images are attached in one image so that using that one image we can extract pixel data, instead of having multiple images in which it will take forever to scan each and every file for extracting the pixel data.

5.2 Mosaicing with R

In R language mosaicing uses gdalUtilis library for performing mosaicing operations using the library built in functions.

This function mosaics a set of input rasters (gdalfile) using parameters found in gdalbuildvrt and subsequently exports the mosaic to an output file (dst_dataset) using parameters found in gdal_translate. The user can choose to preserve the intermediate output.vrt file, but in general this is not needed.

Using gdalUtilis library we can easily convert all the CRS changed tiff files into one single tiff files in which we can use it for extraction of desired pixel point.

5.3 Mosaicing Code Explanation

```
library(gdalUtils)
library(raster)

indice_dir <- "D:/JOB/GIS/TILES_DATA/TS/INDICES/CRS_CHANGED/"
indice_list <- c("ndvi")
for(indice in indice_list){
  setwd(paste(indice_dir,paste(indice,"",sep="/"),sep=""))
  indice_file_vrt <- paste(indice,"vrt",sep=".")
  indice_file_tif <- paste(indice,"tif",sep=".")
  gdalbuildvrt(gdalfile = "*.tif",
               output.vrt = indice_file_vrt)
  gdal_translate(src_dataset = indice_file_vrt,
                 dst_dataset = indice_file_tif,
                 output_Raster = TRUE,
                 options = c("BIGTIFF=YES", "COMPRESSION=LZW"))
  unlink(indice_file_vrt,force=FALSE,recursive=FALSE)
  plot(raster(paste(indice_dir,indice_file_tif,sep="/")))
}
```

- On the start we will be including the required libraries such as gdalUtils and raster for performing mosaicing, then the directory where the CRS changed files will be there will be assigned to a variable called indice_dir, after that a list of vector with indices names in it will be created so that only those of them will get mosaic.

- Now looping each indice folder name using for() loop, then we will be pasting the indice name and the indice_dir variable so using paste() function, then the output will be given into setwd() function which will take path and sets the current working directory to the given path.

- Two variables indice_file_vrt and indice_file_tif will be created by pasting indice name with ".vrt" and ".tif" respectively, these will be used for future purpose, by using gdalbuilvrt() function from gdalUtils library we will create a vrt file which contains the data of all the tif files from the given directory with the name we defined in earlier to variable indice_file_vrt, after that using gdal_translate() function from library gdalUtils we will import that create vrt file and then using the data in it we will be creating a tiff file with the name we define earlier for the variable indice_file_tif.

- After the tif file creation the vrt file is of waste so using unlink() function we will delete the vrt file and we will show a diagram for the mosaic of the specified indice using the plot() function which takes raster object as input and plots the diagram of that raster.

- This process continues until it reaches the end of indice names in the defined indice name vector on line 5.

6. PIXEL DATA EXTRACTION

6.1 What is Pixel Data?

The tiff file is nothing but an image file with extra data stored in it, so after mosaicing every indice we will be having one tiff file for each indice, these tiff image files will contain pixels in it, and each pixel will have data in it, for example: ndvi_mosaic.tiff file is the output file for ndvi indice, each and every pixel in that file contains the value of ndvi only so that data is called a pixel data

We will be extracting that pixel data using the code we created in our project.

6.2 Pixel Data Extraction Code Explanation

- Summary of the code will be, required libraries will be included and a separate directory will be created on the start of the program then we will loop each indice in the given directory on the start, and we will check if the select directory in the loop is the original indice directory or not, it true will scan every tiff file in that specified indice directory and get the pixel data for each file in that directory and write the pixel data into csv file and save it new directory we created for the CSV files.

```
1  library(raster)
2  library(rgdal)
3
4  dir <- "E:/GIS/DOWNLOADED_43PHR/INDICES/"
5  csv_dir <- paste(dir,"CSV",sep="")
6  dir.create(csv_dir,showWarnings=FALSE,recursive=FALSE,mode=0777)
7  indice_from_dir <- list.dirs(dir,recursive=FALSE,full.names=FALSE)
8  original_indice_list <- c("ndvi")
```

- For this program we will be needing raster and rgdal libraries for extraction of raster object pixels. In main program the directory where the indices which pixel data are need to be extracted will be given into the variable dir. After that a new directory with CSV name will be created using the function dir.create()
- After that the list of directories under the given directory will be scanned and stores into a variable called indice_from_dir using list.dirs() function

```
for(dir_indice in indice_from_dir){
    if(dir_indice %in% original_indice_list){
        set_dir <- paste(dir,dir_indice,sep="")
        out_dir <- paste(csv_dir,dir_indice,sep="/")
        dir.create(out_dir,showWarnings=FALSE,recursive=FALSE,mode=0777)
        indice_tiff_files <- list.files(set_dir,full.names=TRUE)
```

- After scanning every directory in the given directory using for() loop we will be looping one directory for one iteration, and will check if the selected directory name is equal to any of the indice names specified in the vector list
- We will be creating set_dir and out_dir variable with current indice directory path and csv directory with indice folder in it respectively.
- Using dir.create() function we will create directory with path defined in the variable out_dir for storing the output csv file after the pixels have been stored in it. And using set_dir variable which consist of current path we will scan the files in the folder with function list.files() and store them into a variable indice_tiff_files.

```
    for(tiff in indice_tiff_files){
        r <- raster(tiff)
        file_name <- names(r)
        pts <- rasterToPoints(r,spatial=TRUE)
        data <- data.frame(pts)
        tile_name <- substr(file_name,1,6)
        unique_id <- sprintf(paste(tile_name,"%d",sep="_"),seq(nrow(data[1])))
        unique_id <- data.frame(Unique_ID=unique_id,stringsAsFactors=FALSE)
        data <- data.frame(unique_id,data[2],data[3],data[1])
        names(data)[names(data) == file_name] <- dir_indice
        csv_write_dir <- paste(csv_dir,paste(names(r),"csv",sep="."),sep="/")
        csv_file_final <- write.csv(data,csv_write_dir,row.names=FALSE)
        if(file.exists(csv_write_dir)){
            cat("Successfully stored ",file_name," values in ",paste(file_name,"csv",sep="."),"File \n")
        }else{
            cat("Failed To Store Values for tiff file ",paste(file_name,"tiff",sep="."),"\n")
        }
    }
}
}
```

- In the for() loop each file path will be selected for each iteration, using that file path we will convert the file into raster object using the function raster(), then we will save the current file name not full path into a variable called file_name using function names() which takes raster object as input and gives the name of that object, which is in raster library.

- Using rasterToPoints() we will get each and every pixel longitude and latitude into a variable pts, this function takes input as raster object and output its pixel points.

- The pixel data we get will be in a matrix form, so for better understanding and better use we will be converting the pts variable into a dataframe using data.fram() function which takes input a matrix object and converts into understandable storage.

- The file name we stored will contain the tile name also so using that file name we will be retrieving the tile name using substr() function into a variable tile_name,

- We will be generating a unique id for each pixel so for that we use seq() function which will generate a sequence of number with 1 as interval upto the given limit, in our case nrows(data[1]) which means in dataframe data we will be selecting data[1] which is a longitude column and nrow() function calculates the number of rows that colum have, and produces that many number as sequence,

- Using that unique id list we will be adding it into the dataframe data along with longitude and latitudes, after the unique id adding we will be changing dataframe column names which have the file name into the indice that was selected in the first for loop.

- For csv directory path we will be naming the csv file with the filename we stored earlier, using paste() function. After that using write.csv() function we will be creating a csv with the dataframe we created and store in the location by giving the path we created earlier.

- Now to check if the file has been created of not we will using file.exists() function which takes input as file directory and outputs False if file doesn't exist, and True if file exist, if True we will be printing a successful file created message, if False we will be displaying failed message using cat() function.

- This loop ends when it meets the end of the vector list of indices we defined, until then it will loop.