UNIVERSIDAD DE CHILE
Facultad de Ciencias Agronómicas
Depto. de Ciencias Ambientales y Rec. Nat. Ren.
Laboratorio para la Investigación en Ciencias Ambientales

LARES

Módulo 6: Manipulación de información espacial

Guillermo S. Fuentes Jaque

August 20, 2021

El modelo vectorial

Es posible distinguir 3 tipos de geometrías en el modelo vectorial para el análisis espacial, estos corresponden a **puntos**, **líneas** y **polígonos**, los cuales hacen referencia a elementos con características particulares y su respectiva representación en el espacio, los polígonos representan elementos unidimensionales como por ejemplo superficies, las líneas representan elementos unidimensionales como por ejemplo calles o ríos, mientras que los puntos representarán la posición de algún objeto en el espacio, por ejemplo, un individuo, el centro de una ciudad e incluso la presencia de algún evento.

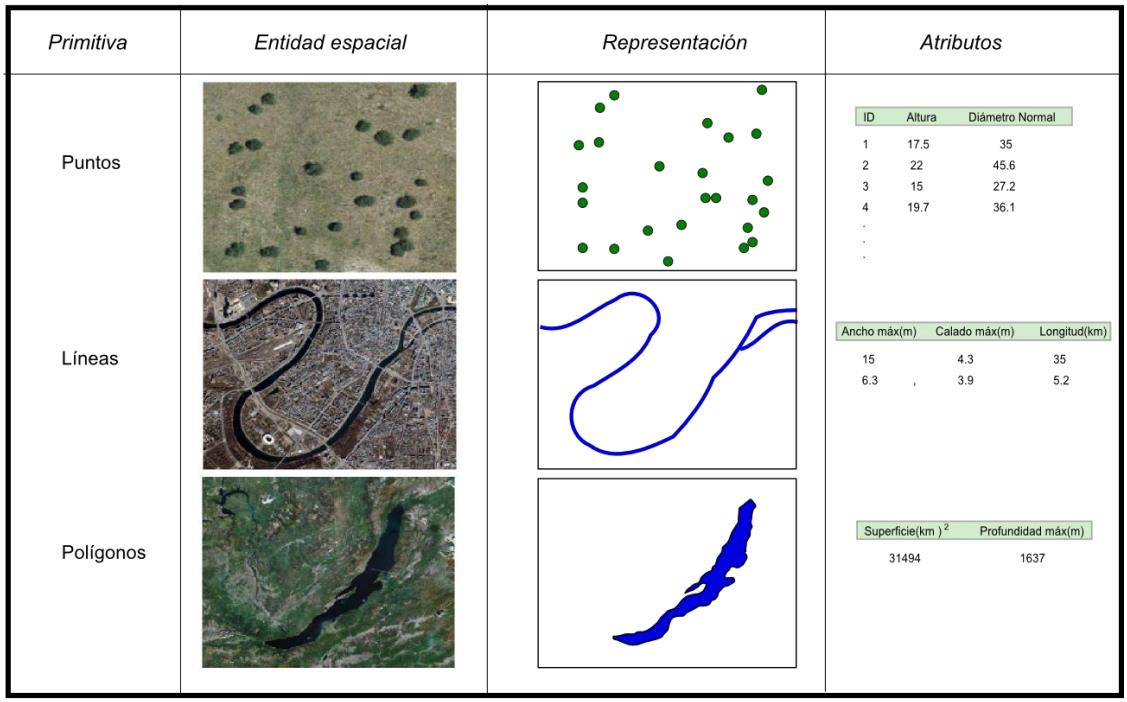


Figura 1: Diferentes entidades y su representación espacial.

Importación de datos puntuales

Existe un gran número de formatos en que se puede importar datos, un ejemplo de estos es el ya conocido **csv** (*Comma Separated Values*), o el versátil **xlsx** de Excel, a continuación se importarán datos desde los formatos csv, xlsx, shp y kml. Para estos ejemplos utilizaremos los materiales encontrados en el repositorio https://github.com/djwillichile/MODULO_06/tree/gh-pages/DATA, también pueden acceder a todo el material utilizado para esta actividad mediante el siguiente **enlace**.

Importar datos desde csv

Este método es uno de los más sencillos de implementar y a la vez, el que permite más formas distintas de realizarlo. utilizaremos el archivo *bradypus.csv*

```
# Asignamos al objeto "path" la ruta del archivo que utilizaremos
path <- "https://djwillichile.github.io/MODULO_06/DATA"

# Asignamos al objeto "file" el nombre del archivo que utilizaremos
file <- "bradypus.csv"

#ruta completa del archivo
fileStrig <- paste0(path,"/",file)
fileStrig
```

```
## [1] "https://djwillichile.github.io/MODULO_06/DATA/bradypus.csv"
```

```
# Leemos el archivo mediante la función "read.csv"
bradypus <- read.csv(file = fileStrig)

# Es posible observar una porcion de los datos cargados
head(bradypus)
```

```
##                species      lon      lat
## 1 Bradypus variegatus -65.4000 -10.3833
## 2 Bradypus variegatus -65.3833 -10.3833
## 3 Bradypus variegatus -65.1333 -16.8000
## 4 Bradypus variegatus -63.6667 -17.4500
## 5 Bradypus variegatus -63.8500 -17.4000
## 6 Bradypus variegatus -64.4167 -16.0000
```

Cabe destacar que es posible leer estos datos utilizando también la función `read.table()` o `read.delim()`

```
# Leemos el archivo mediante la función "read.table"
bradypus <- read.table(file = fileStrig,sep = ",",dec = ".",header = T)
head(bradypus)
```

```
##                species      lon      lat
## 1 Bradypus variegatus -65.4000 -10.3833
## 2 Bradypus variegatus -65.3833 -10.3833
```

```
## 3 Bradypus variegatus -65.1333 -16.8000
## 4 Bradypus variegatus -63.6667 -17.4500
## 5 Bradypus variegatus -63.8500 -17.4000
## 6 Bradypus variegatus -64.4167 -16.0000
```

```
# Leemos el archivo mediante la función "read.delim"
bradypus <- read.delim(file = fileStrig,sep = ",",dec = ".",header = T)
head(bradypus)
```

```
##                species      lon      lat
## 1 Bradypus variegatus -65.4000 -10.3833
## 2 Bradypus variegatus -65.3833 -10.3833
## 3 Bradypus variegatus -65.1333 -16.8000
## 4 Bradypus variegatus -63.6667 -17.4500
## 5 Bradypus variegatus -63.8500 -17.4000
## 6 Bradypus variegatus -64.4167 -16.0000
```

Otros formatos para importar

Así como importamos la base de datos a partir de un *csv* que contenía la información, también es posible utilizar distintas extensiones de archivos, como por ejemplo los archivos de texto delimitado (*txt*), la extensión de Microsoft Excel (*xls* o *xlsx*) o extensiones más conocidas en el mundo del análisis espacial como lo son el Keyhole Markup Language (*kml*) de Google Earth o el Shape File (*shp*) de ESRI, los cuales veremos más a delante.

Para las primeras 3 extensiones, la manera en que se pueden cargar los archivos son relativamente parecidas, la gran diferencia es que el archivo de texto delimitado no necesita de paquetes externos al de *r base* para su importación. A continuación veremos de manera muy resumida como importar la misma base de datos proveniente de distintos ficheros con estas 3 extensiones.

Archivos de texto delimitado (*txt*)

```
# Asignamos al objeto "fileStrig" ruta completa del archivo
fileStrig <- "https://djwillichile.github.io/MODULO_06/DATA/bradypus.txt"

# Leemos el archivo mediante la función "read.table"
bradypus <- read.table(file = fileStrig,sep = "\t",dec = ".",header = T)
head(bradypus)
```

```
##                species      lon      lat
## 1 Bradypus variegatus -65.4000 -10.3833
## 2 Bradypus variegatus -65.3833 -10.3833
## 3 Bradypus variegatus -65.1333 -16.8000
## 4 Bradypus variegatus -63.6667 -17.4500
## 5 Bradypus variegatus -63.8500 -17.4000
## 6 Bradypus variegatus -64.4167 -16.0000
```

```r
# Leemos el archivo mediante la función "read.delim"
bradypus <- read.delim(file = fileStrig,sep = "\t",dec = ".",header = T)
head(bradypus)
```

```
##               species      lon      lat
## 1 Bradypus variegatus -65.4000 -10.3833
## 2 Bradypus variegatus -65.3833 -10.3833
## 3 Bradypus variegatus -65.1333 -16.8000
## 4 Bradypus variegatus -63.6667 -17.4500
## 5 Bradypus variegatus -63.8500 -17.4000
## 6 Bradypus variegatus -64.4167 -16.0000
```

Libro de MS. Excel 1997-2003 (*xls*)

```r
# cargamos paquete para soportar el formato
library(readxl)

# creamos un archivo temporal en el objeto "temp"
temp <- tempfile(fileext = ".xls")

# Asignamos al objeto "fileStrig" ruta completa del archivo y lo
descargamos
fileStrig <- "https://djwillichile.github.io/MODULO_06/DATA/bradypus.xls"
curl::curl_download(fileStrig, temp) # descargar el fichero en el archivo
temporal

# Leemos el archivo temporal mediante la función "read_excel"
bradypus <- read_excel(temp,1)
head(bradypus)
```

```
## # A tibble: 6 x 3
##   species               lon   lat
##   <chr>               <dbl> <dbl>
## 1 Bradypus variegatus -65.4 -10.4
## 2 Bradypus variegatus -65.4 -10.4
## 3 Bradypus variegatus -65.1 -16.8
## 4 Bradypus variegatus -63.7 -17.4
## 5 Bradypus variegatus -63.8 -17.4
## 6 Bradypus variegatus -64.4 -16
```

Libro de MS. Excel actual(*xlsx*)

```r
# cargamos paquete para soportar el formato
library(openxlsx)

# Asignamos al objeto "fileStrig" ruta completa del archivo
fileStrig <-
"https://djwillichile.github.io/MODULO_06/DATA/bradypus.xlsx"

# Leemos el archivo temporal mediante la función "read_excel"
bradypus <- read.xlsx(fileStrig,1)
head(bradypus)
```

```
##                species      lon      lat
## 1 Bradypus variegatus -65.4000 -10.3833
## 2 Bradypus variegatus -65.3833 -10.3833
## 3 Bradypus variegatus -65.1333 -16.8000
## 4 Bradypus variegatus -63.6667 -17.4500
## 5 Bradypus variegatus -63.8500 -17.4000
## 6 Bradypus variegatus -64.4167 -16.0000
```

Es posible apreciar que el objeto `bradypus` es un data.frame común y que no constituye un objeto de tipo espacial

```
class(bradypus)

## [1] "data.frame"
```

Es importante precisar que los datos puntuales requieren contar con al menos 2 variables correspondientes a las coordenadas geográficas verticales (*latitud*) y horizontales (*longitud*), a las que adicionalmente se les puede incluir una tercera variable correspondiente a la coordenada ortogonal de altura (*altitud*). Para este ejemplo es posible apreciar que contamos con la variable horizontal y vertical de coordenadas, *lon* y *lat* respectivamente.

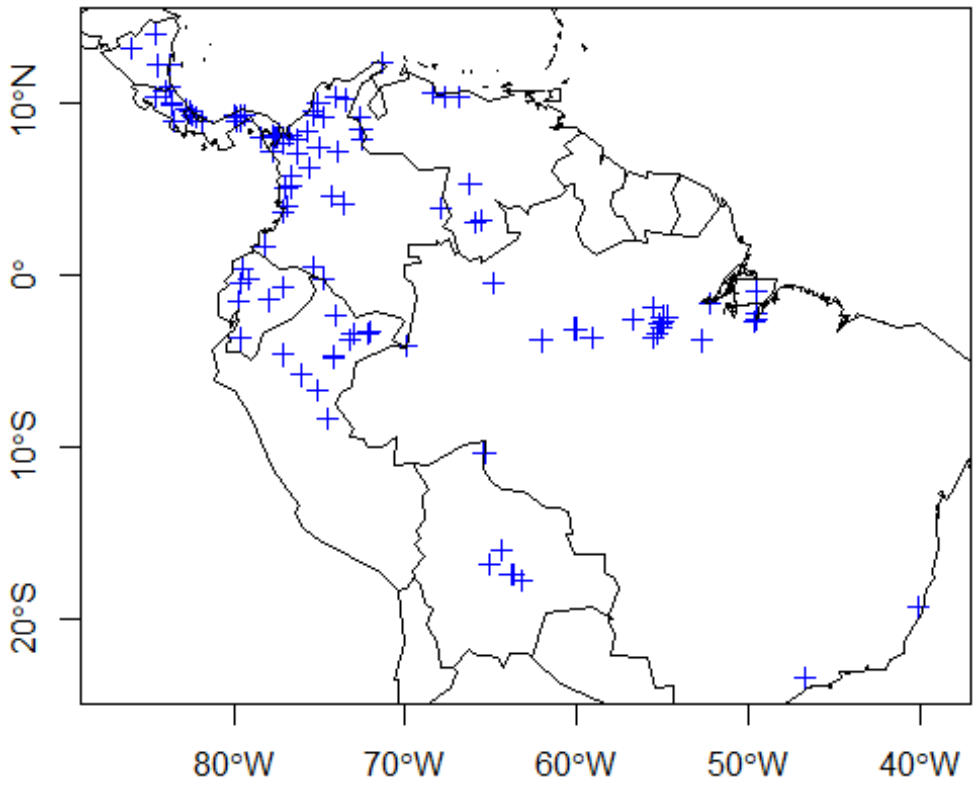
species	lon	lat
Bradypus variegatus	-65.4000	-10.3833
Bradypus variegatus	-65.3833	-10.3833
Bradypus variegatus	-65.1333	-16.8000
Bradypus variegatus	-63.6667	-17.4500
Bradypus variegatus	-63.8500	-17.4000
Bradypus variegatus	-64.4167	-16.0000

Crear capas espaciales de puntos

Para crear objetos espaciales a partir de matrices o data.frames con coordenadas espaciales es posible utilizar las funciones `SpatialPoints()`,`SpatialPointsDataFrame()` o `coordinates()`. Es preciso mencionar que para utilizar dichas funciones es necesario llamar al paquete `sp` el cual se carga automáticamente al llamar a los paquetes `raster` y/o `rgdal`. También es necesario que las coordenadas espaciales se encuentren registradas en un único sistema de referencia con su respectivo DATUM y tener muy claro a cual corresponde para evitar problemas al momento de georreferenciar las coordenadas.

```
# Cargamos paquetes espaciales
library(raster)

## Loading required package: sp

library(rgdal)
```

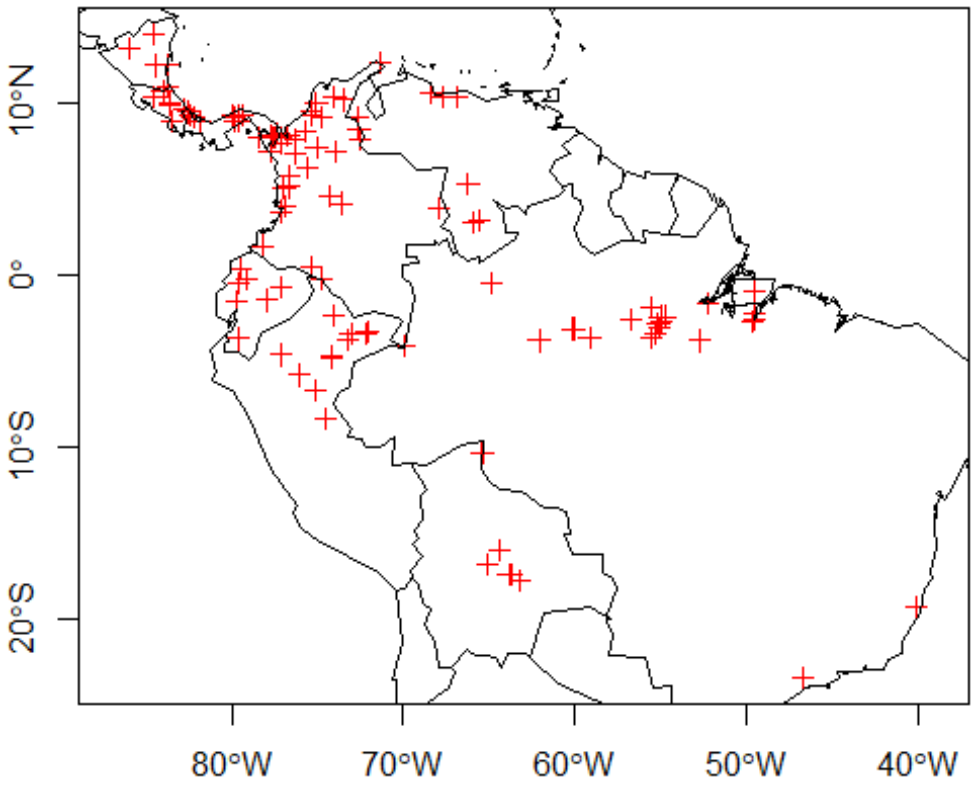
```
## rgdal: version: 1.5-12, (SVN revision 1018)
## Geospatial Data Abstraction Library extensions to R successfully
loaded
## Loaded GDAL runtime: GDAL 3.0.4, released 2020/01/28
## Path to GDAL shared files: C:/Users/Admin/Documents/R/win-
library/3.6/rgdal/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 6.3.1, February 10th, 2020, [PJ_VERSION:
631]
## Path to PROJ shared files: C:/Users/Admin/Documents/R/win-
library/3.6/rgdal/proj
## Linking to sp version:1.4-2
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading
rgdal.

library(maptools)

## Checking rgeos availability: TRUE

# importamos una capa espacial del mundo para contextualizar
data(wrld_simpl)
par(mar = c(2, 2, 0.1, 0.1))

# Creamos el objeto espacial "bradypus.sp" donde proj4string corresponde
al sistema de referencia
bradypus.sp <- SpatialPoints(bradypus[c("lon","lat")],
proj4string=crs("+init=epsg:4326"))
bradypus.sp

## class       : SpatialPoints
## features    : 116
## extent      : -85.9333, -40.0667, -23.45, 13.95  (xmin, xmax, ymin,
ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs

# graficamos el objeto espacial, cosa que no era posible hacer con el
data.frame
plot(bradypus.sp,axes=T,col="blue")

# añadimos a la gráfica la capa espacial del mundo
plot(wrld_simpl,add=T)
```



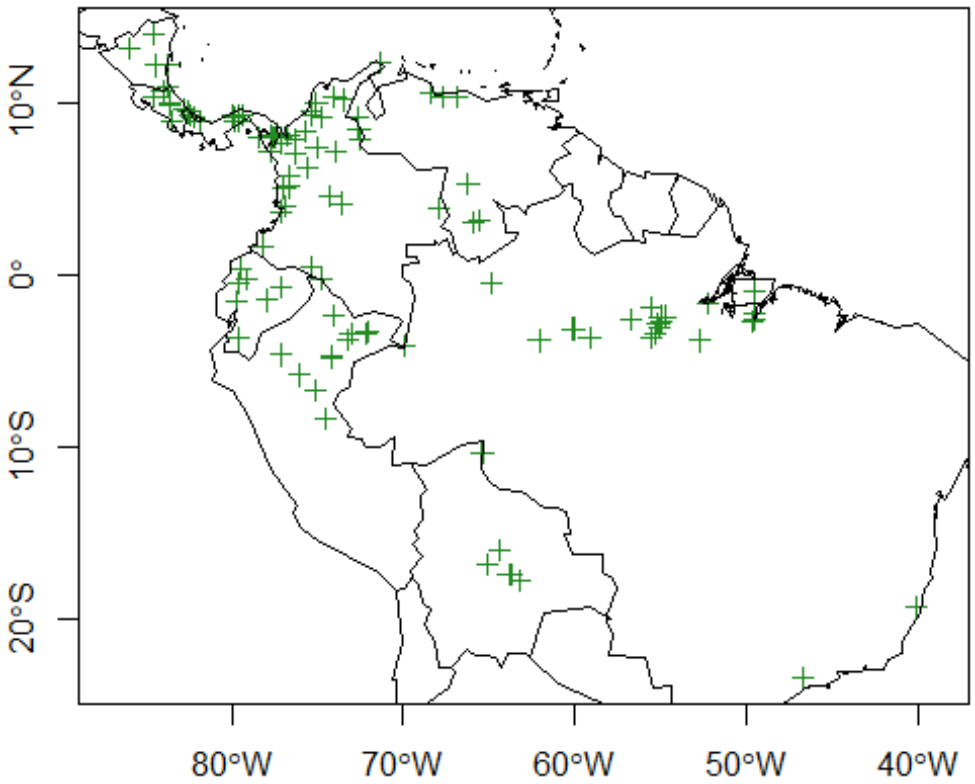

```
# realizamos el mismo procedimiento con la función
"SpatialPointsDataFrame"
bradypus.sp2 <- SpatialPointsDataFrame(bradypus[c("lon","lat")], data =
bradypus["species"], proj4string=crs("+init=epsg:4326"))
bradypus.sp2
```

```
## class       : SpatialPointsDataFrame
## features    : 116
## extent      : -85.9333, -40.0667, -23.45, 13.95  (xmin, xmax, ymin,
ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## variables   : 1
## names       :             species
## min values  : Bradypus variegatus
## max values  : Bradypus variegatus
```

```
# graficamos
plot(bradypus.sp2,axes=T,col="red")
plot(wrld_simpl,add=T)
```




```
# ahora realizamos el mismo procedimiento con la función "coordinates"
bradypus.sp3 <- bradypus
coordinates(bradypus.sp3) <- ~lon+lat
crs(bradypus.sp3)=crs("+init=epsg:4326")
bradypus.sp3

## class       : SpatialPointsDataFrame
## features    : 116
## extent      : -85.9333, -40.0667, -23.45, 13.95  (xmin, xmax, ymin,
ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## variables   : 1
## names       :             species
## min values  : Bradypus variegatus
## max values  : Bradypus variegatus

# graficamos
plot(bradypus.sp3,axes=T,col="forestgreen")
plot(wrld_simpl,add=T)
```



Cómo habrán podido notar, los 3 métodos permiten obtener resultados muy parecidos utilizando a su vez una sintaxis simple que no requiere mayor análisis, es decir, se requiere la base de datos con coordenadas y el sistema de referencia, la principal diferencia radica en que la función `SpatialPoints()` devuelve solo las coordenadas espaciales georreferenciadas mientras que las otras 2 georeferencian todo el contenido de la base de datos.

Importar capas vectoriales

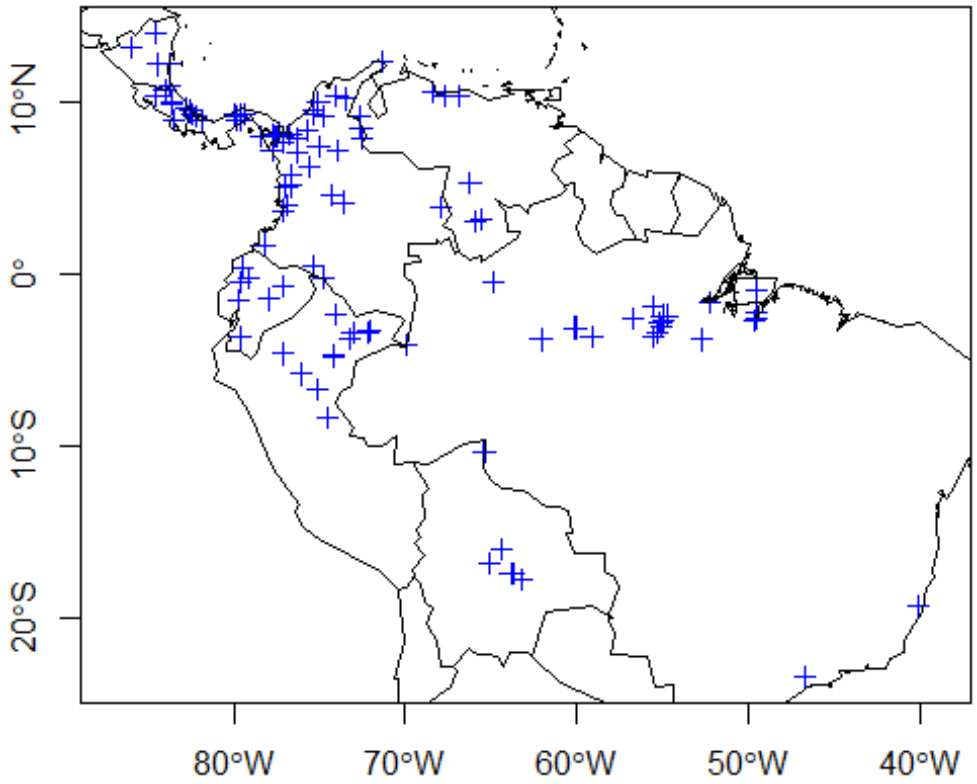
Ya aprendimos como importar información y crear capas vectoriales a partir de bases de datos puntuales, pero en el mundo del análisis espacial se manejan distintos formatos y archivos con los que van a toparse de vez en cuando, algunos de estos formatos con los ya mencionados *shp* y *kml*, por suerte existen paquetes para importar y manipular dichos archivos, una de las funciones más utilizadas para cargar estos archivos `readOGR()` del paquete `rgdal`

```
# Asignamos al objeto "fileStrig" ruta completa del archivo kml
fileStrig <- "https://djwillichile.github.io/MODULO_06/DATA/bradypus.kml"

# importamos la capa mediante la función "readOGR"
bradypus1 <- readOGR(fileStrig,layer = "bradypus")

## OGR data source with driver: KML
## Source: "https://djwillichile.github.io/MODULO_06/DATA/bradypus.kml",
layer: "bradypus"
## with 116 features
## It has 2 fields
```

```
bradypus1
```

```
## class       : SpatialPointsDataFrame
## features    : 116
## extent      : -85.9333, -40.0667, -23.45, 13.95  (xmin, xmax, ymin,
ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## variables   : 2
## names       : Name, Description
## min values  :     ,
## max values  :     ,
```

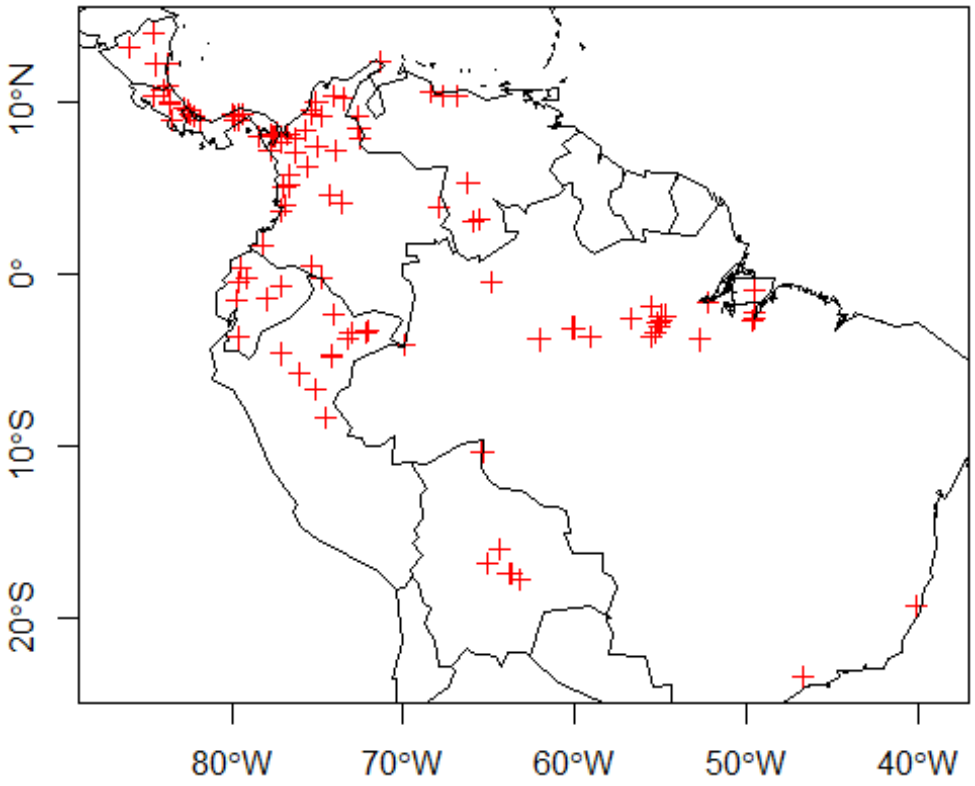
```
# graficamos
par(mar = c(2, 2, 0.1, 0.1))
plot(bradypus1,axes=T,col="blue")
plot(wrld_simpl,add=T)
```




```
# Asignamos al objeto "fileStrig" ruta completa del archivo shp
fileStrig <- "https://djwillichile.github.io/MODULO_06/DATA/bradypus.shp"

# importamos la capa mediante la función "readOGR"
bradypus2 <- readOGR("DATA/bradypus.shp",encoding = "ESRI Shapefile")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "G:\github\MODULO_06\DATA\bradypus.shp", layer: "bradypus"
## with 116 features
## It has 3 fields
```

```
bradypus2

## class       : SpatialPointsDataFrame
## features    : 116
## extent      : -85.9333, -40.0667, -23.45, 13.95  (xmin, xmax, ymin,
ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## variables   : 3
## names       :              species,      lon,    lat
## min values  : Bradypus variegatus, -85.9333, -23.45
## max values  : Bradypus variegatus, -40.0667,  13.95

# graficamos
par(mar = c(2, 2, 0.1, 0.1))
plot(bradypus2,axes=T,col="red")
plot(wrld_simpl,add=T)
```



Como ya habrán visto, la importación de estos datos espaciales sigue el mismo patrón, la función `readOGR()` requiere el nombre del archivo y algunas veces la codificación de este para que no se desconfigure la base de datos no espacial asociada al archivo, de todos modos podrán revisar el detalle de esta función incorporando un sigo de interrogación antes de la función (`?readOGR()`) u ocupando la función help (`help("readOGR")`)

El modelo raster

Como ya fue mencionado en el módulo anterior, el modelo raster corresponde a una representación matricial de elementos o fenómenos que poseen algún tipo de expresión espacial, esto tiene una serie de ventajas, como por ejemplo, que el análisis matemático y lógico de este modelo es mucho más simple que en el modelo vectorial, de hecho, esa ventaja es la que nos impulsa a desarrollar el **análisis multicriterio** utilizando este modelo de representación espacial. es así que procederemos a estudiar la importación de imágenes raster a partir de distintos formatos, además de las distintas operaciones aritméticas y lógicas básicas que podemos aplicar a las imágenes representadas mediante el modelo espacial raster.

Importación de archivos raster

Cabe destacar que existen 2 principales tipos de archivos raster, por un lado están los de tipo monobanda (también conocida como single band), es decir, que cada archivo contienen una única imagen raster, mientras que por otro lado existen los de tipo multibanda (multi band), lo que quiere decir que dentro de ese archivo coexisten varias imágenes agrupadas de distinta maneras, inclusive, con distinta resolución.

a su vez, los archivos raster podrían tener distintas extensiones dependiendo del proveedor de servicios, el proceso realizado, e inclusive el software utilizado para su procesamiento. A continuación se presenta un listado con los principales archivos en que es posible almacenar información raster.

Tipo	Descripción	Extensión	Soporta multibanda
raster	Formato raster nativo	.grd	Si
ascii	ESRI Ascii	.asc	No
SAGA	SAGA GIS	.sdat	No
IDRISI	IDRISI	.rst	No
CDF	netCDF (require `ncdf4`)	.nc	Si
GTiff	GeoTiff (require `rgdal`)	.tif	Si
ENVI	Extensión hdr de ENVI	.envi	Si
EHdr	Extensión hdr de ESRI	.bil	Si
HFA	Imágenes de Erdas	.img	Si

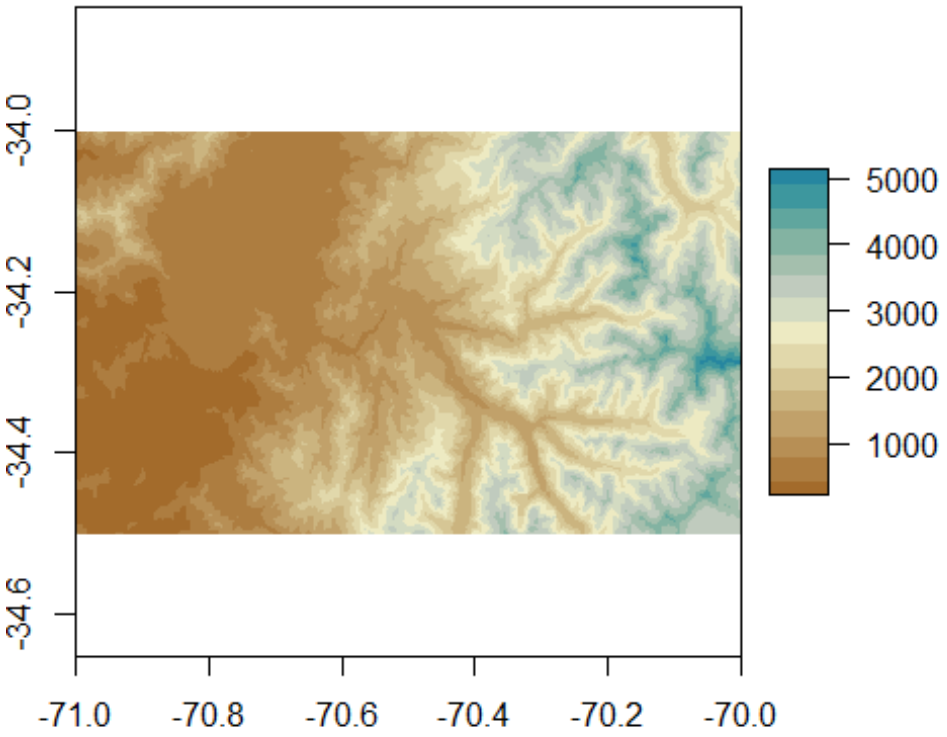
Es preciso declarar que para efectos prácticos no trabajaremos con los formatos *netCDF*, *hdr* ni *Erdas*, ya que estos formatos son poco utilizados al momento de realizar análisis multicriterio y aceptan distintas formas de almacenar las imágenes, las cuales no daría tiempo de estudiar en este módulo.

Importar de archivos monobanda

Los archivos de tipo monobanda, como ya se mencionó anteriormente, corresponden a archivos que solo contienen un raster en su arquitectura, por lo tanto es posible

importarlos mediante la función `raster()` del paquete homónimo, siendo éste, capaz de reconocer qué extensión tienen los archivos y configurar la imagen de acuerdo a ésto. A continuación se observa el procedimiento standar para importar archivos monobanda.

```
# Asignamos al objeto "fileStrig" ruta completa del archivo "ascii"
fileStrig <- "https://djwillichile.github.io/MODULO_06/DATA/alt.asc"

# Importamos el archivo con la función "raster"
ALT <- stack(fileStrig)
ALT

## class      : RasterStack
## dimensions : 200, 400, 80000, 1  (nrow, ncol, ncell, nlayers)
## resolution : 0.0025, 0.0025  (x, y)
## extent     : -71, -70, -34.5, -34  (xmin, xmax, ymin, ymax)
## crs        : NA
## names      : layer

# Graficamos el raster de altura
par(mar = c(3, 1.8, 0.3, 1)) #configuramos los márgenes
plot(ALT,legend.width=2,col=hcl.colors(15,"Earth"))
```

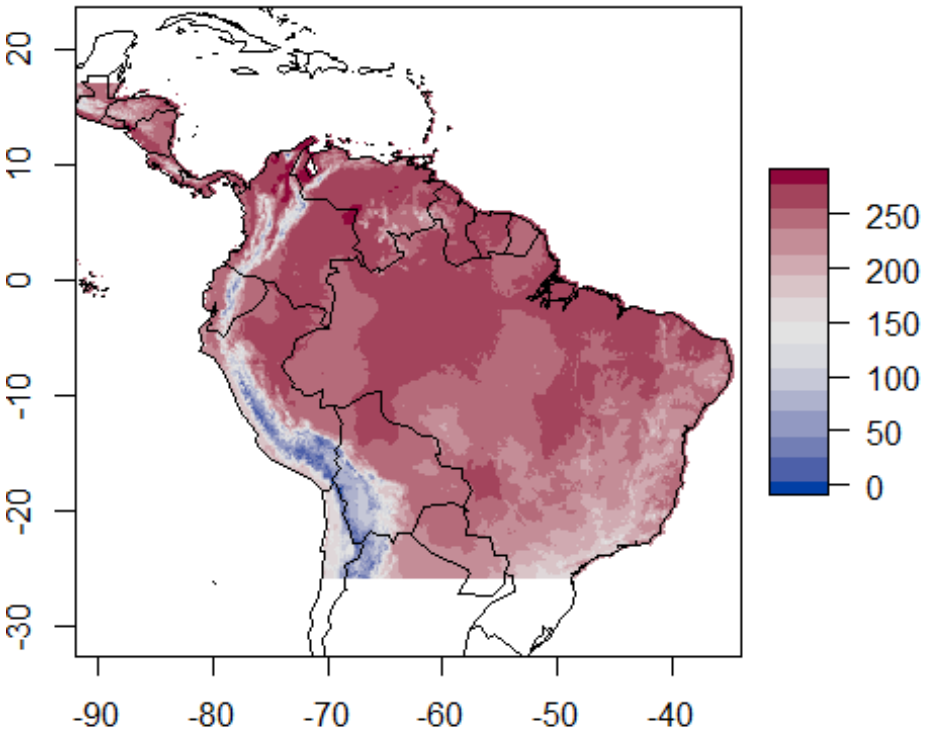



```
# Asignamos al objeto "fileStrig" ruta completa del archivo "grd"
fileStrig <- "DATA/TEMP.rst"

# Importamos el archivo con la función "stack"
```

```r
TEMP <- stack(fileStrig)
TEMP
```

```
## class      : RasterStack
## dimensions : 344, 464, 159616, 1  (nrow, ncol, ncell, nlayers)
## resolution : 0.125, 0.125  (x, y)
## extent     : -92, -34, -26, 17  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## names      :     layer
## min values : -8.425926
## max values :  290.9167
```

```r
# Graficamos el raster de temperatura
par(mar = c(3, 1.8, 0.3, 1)) #configuramos los márgenes
plot(TEMP,legend.width=2,col=hcl.colors(15,"Blue-Red"))
plot(wrld_simpl,add=T)
```



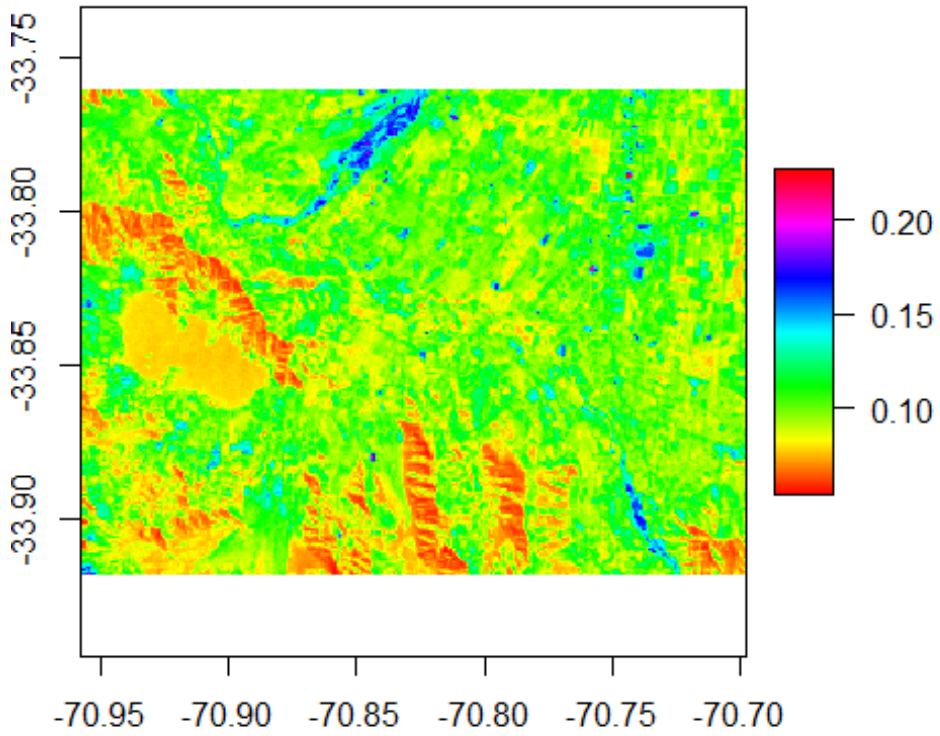
Importar de archivos multibanda

Por otra parte, los archivos de tipo multibanda, corresponden a archivos que, en su arquitectura, pueden contener más de na imagen raster, por lo tanto, poseen un sistema interno algo más complejo generando una arquitectura conocida como pila o stack, es así que, para importar archivos multibanda se utilizan las funciones `brick()` y `stack()` provenientes ambas del paquete `raster`, siendo éste, capaz también de reconocer qué extensión tienen los archivos y configurar las imágenes de acuerdo a ésto. A continuación se observa el procedimiento standar para importar archivos multibanda.

```r
# Asignamos al objeto "fileStrig" ruta completa del archivo tif
fileStrig <-
"https://djwillichile.github.io/MODULO_06/DATA/LANDSAT_5.tif"

# Importamos el archivo con la función "stack"
LANDSAT <- stack(fileStrig)
LANDSAT
```

```
## class      : RasterStack
## dimensions : 195, 268, 52260, 7  (nrow, ncol, ncell, nlayers)
## resolution : 0.000972, 0.00081  (x, y)
## extent     : -70.95806, -70.69756, -33.91797, -33.76002  (xmin, xmax,
ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## names      : LANDSAT_5.1, LANDSAT_5.2, LANDSAT_5.3, LANDSAT_5.4,
LANDSAT_5.5, LANDSAT_5.6, LANDSAT_5.7
## min values :  0.08824112,  0.05395551,  0.03655327,  0.02836457,
0.01677339,  0.01488787,  0.04529983
## max values :   0.2298414,   0.2268108,   0.2492448,   0.4743345,
0.4027909,   0.3041459,   0.2840599
```

```r
# Graficamos la segunda banda
par(mar = c(3, 1.8, 0.3, 1)) #configuramos los márgenes
plot(LANDSAT[[2]],legend.width=2,col=rainbow(255))
plot(wrld_simpl,add=T)
```

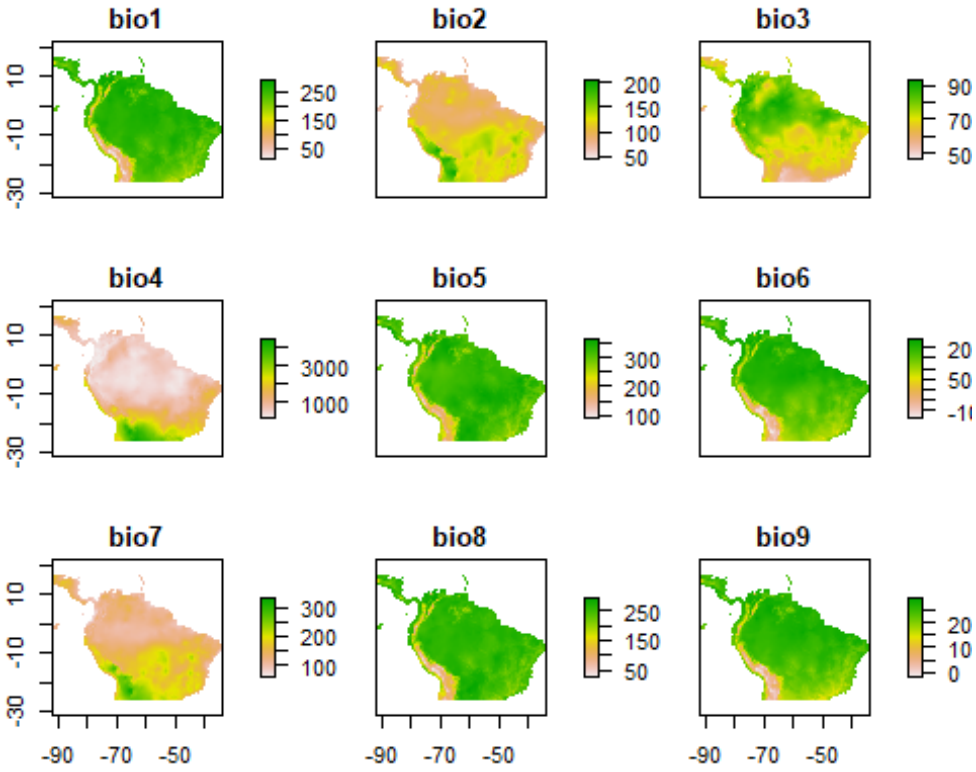


```r
# Graficamos 3 bandas en RGB
plotRGB(LANDSAT, stretch = "lin",
        r = 3, g = 2, b = 1)
box()
```

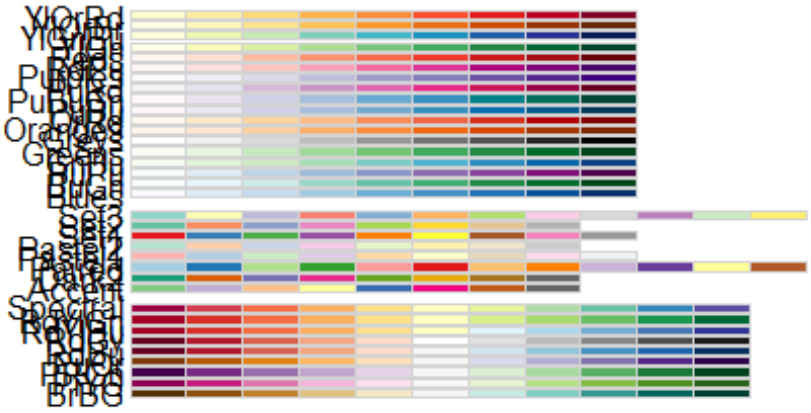
```r
# Asignamos al objeto "fileStrig" ruta completa del archivo grd
fileStrig <- "DATA/bio.grd"

# Importamos el archivo con la función "stack"
bio <- stack(fileStrig)
bio
```

```
## class      : RasterStack
## dimensions : 129, 174, 22446, 19  (nrow, ncol, ncell, nlayers)
## resolution : 0.3333333, 0.3333333  (x, y)
## extent     : -92, -34, -26, 17  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## names      :       bio1,       bio2,       bio3,       bio4,
bio5,       bio6,       bio7,       bio8,       bio9,      bio10,
bio11,      bio12,      bio13,      bio14,      bio15, ...
## min values :   14.29688,   46.00000,   47.00000,  160.09375,
91.26562, -133.34375,   61.33333,   29.26562,  -15.56250,   29.34375,  -
30.25000,    0.00000,    0.00000,    0.00000,    0.00000, ...
## max values :  288.63333,  204.00000,   92.76562, 4458.46875,
363.95312,  238.76190,  330.12500,  290.56250,  295.18750,  299.71429,
279.81250, 7754.70312,  895.75472,  481.29688,  220.75000, ...
```

```
plot(bio[[1:9]],nr=3,legend.width=1.5)
```




```
library(RColorBrewer)
display.brewer.all()
```



```
pTemp=brewer.pal(n = 11, name = "Spectral")
pAlt=brewer.pal(n = 11, name = "BrBG")
```