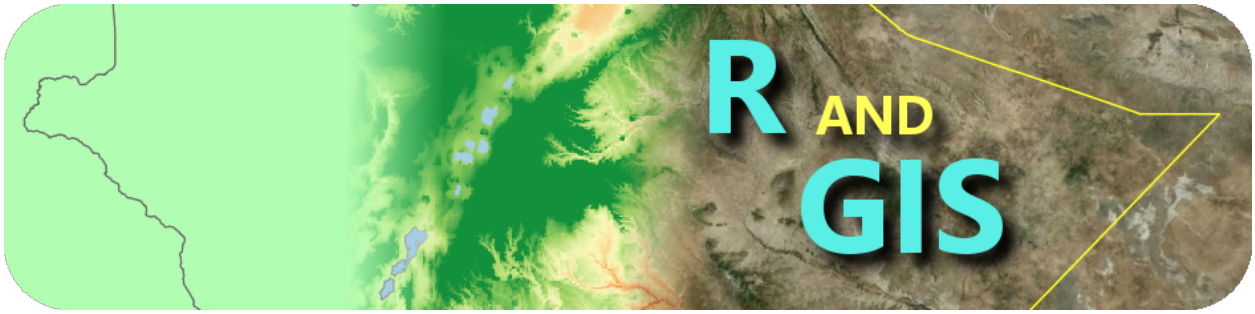


Miguel Alvarez[*1] and **Solomon Estifanos**[†2]

[1]Plant Nutrition, INRES, University of Bonn, Germany
[2]School of Natural Resources Management and Environmental Sciences, Haramaya University, Ethiopia

November 27th–December 1st, 2017

Contents

*malvarez@uni-bonn.de
†solestifa@gmail.com

Haramaya, 2017
doi: 10.13140/RG.2.2.23662.31049
ARBONETH – *Training the trainers of the future!*

1 Introduction to the Workshop

This is a quick introduction to the applications of **R** handling and analysing **GIS** data sets. This workshop is organised by the project **ARBONETH** (The Ethiopian Arboretum Project, http://www.arboneth.com), which is founded by the **German Academic Exchange Service** (DAAD) and the **Federal Ministry for Economic Cooperation and Development** (BMZ).

The content of this booklet represent an imporved version of the previous work prepared by Alvarez & Borchardt (2015).

1.1 Installing Instructions

The main software using in this workshop is RStudio, integrating an installation of R. Both packages will be provided as portable applications in the workshop's pen drive. Both applications are freeware and can be downloaded from their respective sites.

We provide additionally an installation of QGIS, which will be only used to visualise GIS data (shapefiles or rasters).

Further software that should be installed in your computer are Java, Adobe Acrobat Reader and Google Earth.

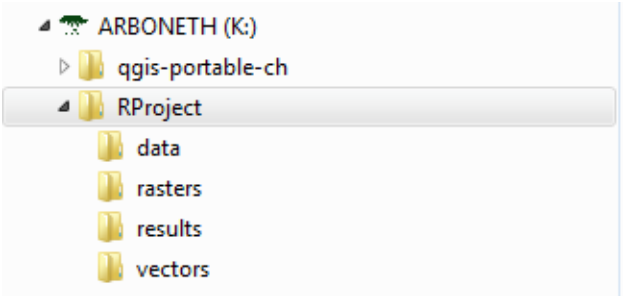
Hint

Extensions of are called packages (Wickham, 2015). They are usually developed to fulfill specific tasks. Many of those extensions are available at the **Comprehensive R Archive Network** (R).
Packages required for this workshop will be distributed during the workshop. Further packages can be installed using following command.

```
install.packages("pck-name", dependencies=TRUE)
```

1.2 Data Sets

In the workshop's pen drive there is a folder called **RProject**. This folder will be used as the `working directory` during the R-sessions. The respective subfolders are **data** containing miscellaneus data sets, **vectors** including spatial vectors (e.g. ESRI shapefiles), **rasters** containing raster data sets, and **results** to store data sets created during the sessions.



Hint

For convenience, the path to the pen drive can be set to **K:**. This can be done in **Microsoft Windows** after plugging the pendrive by right mouse click on `Computer`, then `Manage`, `Storage`, `Disk Management`. On the respective drive, right mouse click, `Change Drive Letter and Paths`, `Change`, and then assign the respective letter. By this way, the command to set the working directory will be in .

```
setwd("K:/RProject")
```

Remember, uses as separators between folders and subfolders either slashes (`/`) or double back slashes (`\\`).

2 Basics on R

The basic components, you may know before starting the workshop are explained as follows. The **console** is the interface where you execute command lines. The **script** is a text file used to write and save command lines. Though commands can be directly writen in the console, it is advisable to use scripts as a way to protocol analysis routines. The **workspace** is the virtual place containing information structured in **objects**. The **working directory** is the folder, where **R** searches for data to load and where data will be written. Finally, the **history** is a record of command lines executed during a **session**.

Objects are structures containing data or functions in **R**. Such objects may belong to a `class`, which determines its `attributes`. Herewith the basic class is the `vector` and almost every object in **R** can be built from vectors.

2.1 Vectors and Matrices

A `vector` is a sequence of values belonging to the same mode or class (Crawley, 2007; Ligges, 2008). A further attribute of the vector is its `length` (its only dimension). Modes for data are `logical`, `numeric`, `factor`, `complex` and `character`. Additional modes used for programming purposes are `function`, `formula` and `expression`. The following are some alternative ways to generate vectors.

```
rep(5, times=10)
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5
```

```
seq(from=10, to=100, by=10)
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

```
sample(letters, size=10, replace=TRUE)
```

```
##  [1] "a" "q" "k" "o" "r" "g" "z" "p" "u" "e"
```

```
c("Peter","Piper","picked","a","peck","of","pickled","peppers")
```

```
## [1] "Peter"   "Piper"   "picked"  "a"       "peck"    "of"      "pickled"
## [8] "peppers"
```

The previously generated vectors were displayed in the `console`, but they are not available for further routines. For it, you have to create new objects in the `workspace`, containing the respective values. That is to say, you may assign the values to new objects. Such operation is carried out by using the arrow symbol (<-).

```
A <- seq(from=1, to=10)
B <- seq(from=10, to=1)
```

The vectors `A` and `B` are numeric ones and this can be confirmed by `is.numeric(A)`, then **R** will answer you `TRUE`. You can also ask using `class(A)`.

While in the previous example vectors are generated by functions, some operations may also result in vectors.

```
# Mathematical operations
A + B
```

```
##  [1] 11 11 11 11 11 11 11 11 11 11
```

```
A*B
```

```
##  [1] 10 18 24 28 30 30 28 24 18 10
```

```
# Logical operations
set.seed(58)
C <- sample(letters, size=10, replace=TRUE)
C == "p" # are values of C equal to "p"?
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
C != "a" # are values of C different from "a"?
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

There are several ways allowing access to elements contained in a vector. The most common is using square brackets as displayed in the console. In such brackets you can indicate the position of elements through `integer` values. Negative integers indicate elements to be excluded. It is also possible to use `logic` values, whereupon `TRUE` indicates elements to be included and `FALSE` indicates elements to be excluded.

```
B[5] # using number
B[1:5] # using numeric vector
B[-5] # using negative number
B[B != 7] # using logical vector
```

Additionally to it, there is also the possibility to name the elements of a vector and access to them using those names as identity.

```
names(B) <- LETTERS[1:length(B)]
B

##  A  B  C  D  E  F  G  H  I  J
## 10  9  8  7  6  5  4  3  2  1

B[c("C","F","H")]

## C F H
## 8 5 3
```

The `matrix` is in **R** a `vector` with 2 dimensions assigned as attribute (Ligges, 2008). Notice that while in mathematics a vector is a special case of matrix, in **R** is the other way round. To produce a `matrix` you can use the functions `matrix`, `rbind` (binding rows), or `cbind` (binding columns).

```
# matrix using function matrix
matrix(1:20, nrow=5)

##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20

matrix(1:20, ncol=5, byrow=TRUE)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20

# matrix using functions cbind and rbind
cbind(A,B)

##    A  B
## A  1 10
## B  2  9
## C  3  8
## D  4  7
## E  5  6
## F  6  5
## G  7  4
## H  8  3
## I  9  2
## J 10  1
```

```
rbind(A,B)
```

```
##    A B C D E F G H I  J
## A  1 2 3 4 5 6 7 8 9 10
## B 10 9 8 7 6 5 4 3 2  1
```

The access to single elements in the matrix is analogous to the access for vectors, but two values (separated by commas) are required, the first value indicates the row and the second, the column.

```
# Creating the matrix in the workspace
M <- cbind(A,B)
M[1,] # access to first row
M[,1] # access to first column
M["A","B"] # access by names of rows and columns
```

2.2 Lists and Data Frames

Lists are more complex but at the same time more flexible. A `list` is an object with elements of different classes (even surrogated lists). The access to elements of a list is similar as for vectors. The use of double square brackets (`[[]]`) and the use of the dollar symbol (`$`) are two ways of access frequently used on lists.

```
# Creating a list
MyList <- list(First="Hello", Second=A, Third=M)
# Access to elements
MyList[1:2] # using a numeric index
MyList[["Third"]] # using element's name
MyList$First # using dollar symbol
MyList$Second[3] # inside of an element
MyList$Third[5,2]
```

One of the most common objects used to handle data in **R** is the `data.frame`. Data frames resemble matrices, but the main difference is that a `matrix` can contain information belonging to only one class (e.g. `numeric`, `factor` or `logical`), while in the data frame every column can be of a different class.

```
## The data.frame
E <- letters[1:10]
MyDataFrame <- data.frame(First=B, Second=C, Third=E, stringsAsFactors=FALSE)
summary(MyDataFrame)
```

```
##      First          Second             Third
##  Min.   : 1.00   Length:10          Length:10
##  1st Qu.: 3.25   Class :character   Class :character
##  Median : 5.50   Mode  :character   Mode  :character
##  Mean   : 5.50
##  3rd Qu.: 7.75
##  Max.   :10.00
```

The access to elements of a `data.frame` can be done in the same way as for a `matrix`. Additionally you can access to the columns by using the symbol `$`, as for lists.

```
MyDataFrame[,"Second"]
```

```
##  [1] "i" "d" "r" "y" "v" "j" "g" "s" "g" "p"
```

```
MyDataFrame$First
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

2.3 Functions and Loops

While operations will be reviewed more into detail during theoretical sessions, here there is a short introduction to functions and loops. A `function` is also an object in `R`. Such functions are usually represented as `foo(argument1=value1, ...)`, where `foo` is the name of the function object and in the brackets and separated by commas you may insert the values for the respective arguments of the function. You will steadily deal with functions during **R** sessions, but in some cases you will need to write your own functions.

```
plus_one <- function(x) x + 1
plus_one(5)

## [1] 6

plus_one(1:10)

##  [1]  2  3  4  5  6  7  8  9 10 11
```

Though loops are not objects, there are some functions executing loops in **R**. The most common one is `for` in combination with `if`.

```
for(i in 1:5) {
        temp <- paste("current value is", i)
        print(temp)
}

## [1] "current value is 1"
## [1] "current value is 2"
## [1] "current value is 3"
## [1] "current value is 4"
## [1] "current value is 5"
```

Here, there curly brackets enclose many command lines that may be evaluated in every loop.

Hint

Be aware of missuse of loops. When working with big objects, the use of loops can cost a lot of time for finish the process in **R**. Some advices about how to avoid loops can be found in Ligges & Fox (2008) and Crawley (2007).

2.4 Data Import and Export

There are many ways to import data to the `workspace` in **R**. One of the most basic ways is using comma separated values (CSV files) as data. It is also possible to import Microsoft Excel files by using the package `xlsx`. Previous to the import and export of data, you may check for the `working directory` used in the current `session`. By typing `getwd()` in the `console`, you will get the respective path. Since this path is the place where **R** will look for files to load, you have to set it according to the location of the files, for instance by `setwd("K:/Rproject")`. Notee that the separation between folders and subfolders are either the common slash symbol (`/`) or twice the backslash (`\\`).

```
Distrib <- read.csv("data/Distrib.csv")
```

Notice that you will always need to assign the loaded data to an `object`, otherwise you just get a print in the `console`, while the content will get lost after the function finish the work.

For the next example we will load the exemplary data `trees` (for details, see `?trees`). This data set will be imported as data frame. There are some functions provided to explore the structure and content of data frames.

```
data(trees)
str(trees)
```

```
## 'data.frame': 31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

```
summary(trees)
```

```
##      Girth           Height       Volume
##  Min.   : 8.30   Min.   :63   Min.   :10.20
##  1st Qu.:11.05   1st Qu.:72   1st Qu.:19.40
##  Median :12.90   Median :76   Median :24.20
##  Mean   :13.25   Mean   :76   Mean   :30.17
##  3rd Qu.:15.25   3rd Qu.:80   3rd Qu.:37.30
##  Max.   :20.60   Max.   :87   Max.   :77.00
```

```
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

```
tail(trees)
```

```
##    Girth Height Volume
## 26  17.3     81   55.4
## 27  17.5     82   55.7
## 28  17.9     80   58.3
## 29  18.0     80   51.5
## 30  18.0     80   51.0
## 31  20.6     87   77.0
```

In this case, `str` display an overview on the structure of the object `trees`, while `summary` shows statistic summaries of each single variable. The functions `head` and `tile` display respectively the first and the last rows. To write this data set into a file we will use the function `write.csv`.

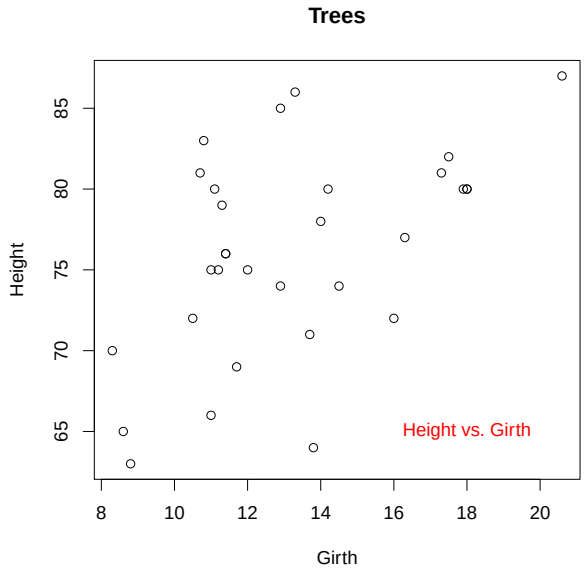
```
write.csv(trees, "results/trees.csv")
```

For general routines to import data in r, look at the help of `read.table` or check the manual `"R Data Import/Export"` in `help.start()`.

2.5 Basics on R Plotting

The very basic function for plotting in R is `plot()` and most of the plotting parameters can be set by using `par()` (take a look in the help file for more details). Plotting functions are classified into two types, on the one side the **high level** functions that produce a whole graphic as in the case of `plot()`, on the other side the **low level** functions are able to introduce single elements in a drawn plot. Some high level functions offer the possibility to use them as low level, usually by setting the argument `add=TRUE`.

```
plot(trees[c("Girth","Height")], main="Trees") # high level function
text(18, 65, labels="Height vs. Girth", col="red") # low level function
```



In this workshop we will not handle further statistical assessments. For analyses of ecological data we recommend a view on Dormann & Kühn (2008), Logan (2010), Hörmann & Unkel (2015), and Zerihun Woldu (2017).

Exercises

- Create a matrix of 10 rows and 10 columns filling it with random digits, then compute the sums of columns and rows.
- Create a data frame including two variables used for a factorial experiment design (e.g. temperature: 30, 50 and 100 degrees, time: 0, 5, 20 min). Use `expand.grid` to get all possible combinations of levels. Randomize sorting of treatments in the data frame.
- Make box plots for the variable `Height` in the data `trees`. Check `data(iris)` and draw box plots for the variable `Sepal.Length` separated by `Species`.
- Prepare a plot for publication.

3 The Spatial Vector Files in R

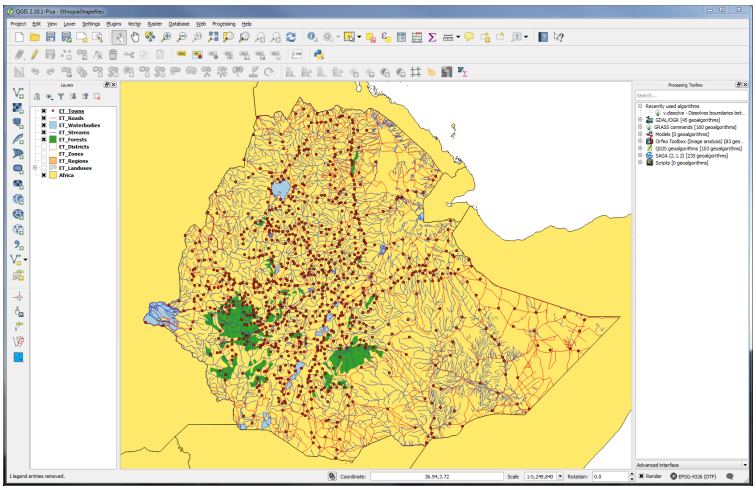
There is a series of formats available for handling spatial data sets (GIS), but we will focus on the classes related to `Spatial*DataFrame`, which can content information usually stored in ESRI Shapefiles (including attribute tables) and can be accessed as data frames. Those object classes are provided by the package `sp`.

3.1 Importing ESRI Shapefiles

To import shapefiles we will use the function `readOGR` from the package `rgdal`.

```
library(rgdal) # load package to the session
Africa <- readOGR(dsn="vectors", layer="Africa")

## OGR data source with driver: ESRI Shapefile
## Source: "vectors", layer: "Africa"
## with 762 features
## It has 3 fields
```

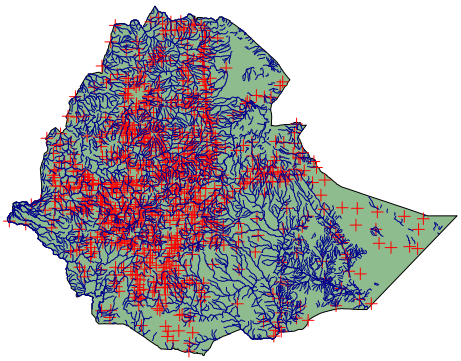
The loaded file contains spatial polygons corresponding to the African countries. In such file we can create a subset, for example considering those countries that are members of Eastern Africa.



```
HornAfrica <- subset(Africa, COUNTRY %in% c("Eritrea","Djibouti","Ethiopia",
                                            "Somalia"))

plot(Africa, col="grey")
plot(HornAfrica, col="orange", add=TRUE)
```

Further type of shapefiles (points and lines) can be also loaded and plotted in a similar way as done in the QGIS project (alternatively in an ArcGIS project).

```
Towns <- readOGR(dsn="vectors", layer="ET_Towns")
Streams <- readOGR(dsn="vectors", layer="ET_Streams")
Ethiopia <- subset(Africa, COUNTRY == "Ethiopia")

plot(Ethiopia, col="darkseagreen")
plot(Towns, col="red", add=TRUE)
plot(Streams, col="darkblue", add=TRUE)
```



Hint

Many of the methods to extract variables from `data.frame` objects are implemented for the `Spatial*DataFrame` classes. These variables are contained in the slot `data`. Therefore the command

```
Africa$COUNTRY
```

is equivalent to

```
Africa@data$COUNTRY
```

For calculating distances between towns, we require the package `rgeos`. Note that such calculation requires a transformation of the coordinates to a projected reference system, for instance the UTM system, which uses meters as coordinate units.

```
proj4string(Towns)

## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

Towns_UTM <- spTransform(Towns, CRS("+proj=utm +zone=37 +north +datum=WGS84"))
Towns_UTM <- subset(Towns_UTM, TOWN_NAME %in% c("Addis Abeba","Jimma",
                                "Dire Dawa"))
rownames(Towns_UTM@data) <- Towns_UTM$TOWN_NAME

library(rgeos)
round(gDistance(Towns_UTM, byid=TRUE)/1000) # Distance in kilometers

##             Dire Dawa Addis Abeba Jimma
## Dire Dawa           0         347   591
## Addis Abeba       347           0   258
## Jimma             591         258     0
```

Looking into a mileage table from Ethiopia, you may realize that the distance values are not that bad, although they are underestimated. The question is, which values are the wrong ones?

	Dire Dawa	Addis Abeba
Addis Abeba	445	
Jimma	791	355

Hint

Spatial Reference Identifiers (**SRIDs**) indicate the coordinate system and the ellipsoid used to georeference spatial objects. The function `CRS` from the package **sp** offers an interface to the PROJ.4 library and set the attribute `proj4string` in spatial objects. A database with numeric identifiers is available from the EPSG (European Petroleum Survey Group Geodesy) and can be inserted in the argument init.

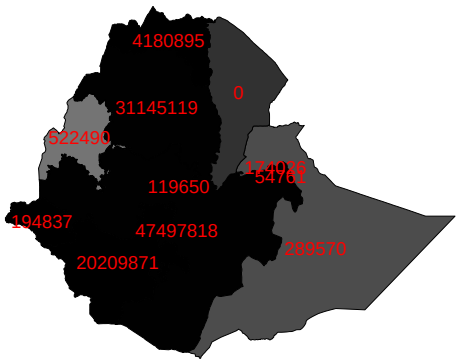
```
CRS("+init=epsg:4326")

## CRS arguments:
##  +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
## +towgs84=0,0,0
```

A **Spatial Reference System** is composed by a **Coordinate Reference System**, an **ellipsoid** and a **datum**. For more details on this topic, see Obe & Hsu (2015).

3.2 Attribute Joins

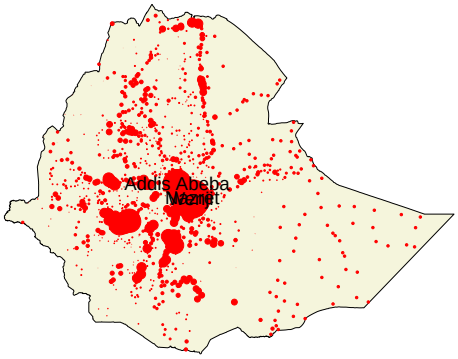
As in ESRI shapefiles, elements included in the spatial objects have attributexs in an attribute table. Such attributes are stored as data frames in the slot `data` (access through `objectName@data`). New attributes can be passed from different sources, for example the population size to the object `Towns`. For it, we have to load the table `"KE_Population.csv"` to our workspace.

```
Regions <- readOGR(dsn="vectors", layer="ET_Regions")
Towns <- readOGR(dsn="vectors", layer="ET_Towns")
# Calculate total population from Towns
Population <- aggregate(POPTOT ~ REGION, Towns@data, sum)
Regions <- merge(Regions, Population, by.x="NAME_1", by.y="REGION")

# Graded grey by population for plot
Palette <- rev(grey(Regions$POPTOT/sum(Regions$POPTOT)))
plot(Regions, col=Palette)
text(coordinates(Regions), labels=Regions$POPTOT, col="red")
```



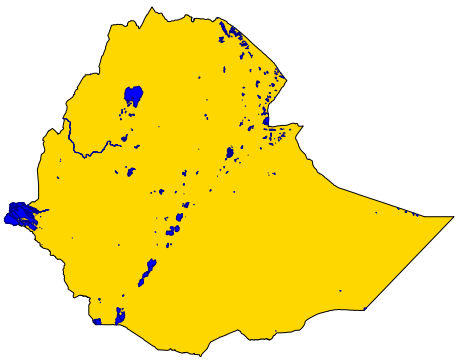
Accordingly, we can select those cities with a population higher than 100,000 inhabitants in order to display them in a map. We will also use the function `symbols` to produce a display equivalent to `bubble` (bubble plots).

```
VeryBigCities <- subset(Towns, POPURB > 100000)
plot(Ethiopia, col="beige")
symbols(coordinates(Towns),
                circles=Towns$POPURB/max(Towns$POPURB, na.rm=TRUE)*0.6,
                fg="red", bg="red", inches=FALSE, add=TRUE)
text(VeryBigCities, labels=VeryBigCities$TOWN_NAME)
```



Calculation of areas by using the function `gArea` from the package `rgeos`. Again as the example of distance calculations, you may transform the coordinate reference system to UTM.

```
Waterbodies <- readOGR(dsn="vectors", layer="ET_Waterbodies")
plot(Ethiopia, col="gold")
plot(Waterbodies, col="blue", border="darkblue", add=TRUE)
```



```
Waterbodies_UTM <- spTransform(Waterbodies,
                CRS("+proj=utm +zone=37 +north +datum=WGS84"))
# Total area covered by lakes (in square kilometers)
gArea(Waterbodies_UTM, byid=FALSE)/1000000
```

```
# Respective area size to attribute table
Waterbodies$Area <- gArea(Waterbodies_UTM, byid=TRUE)/1000000
```

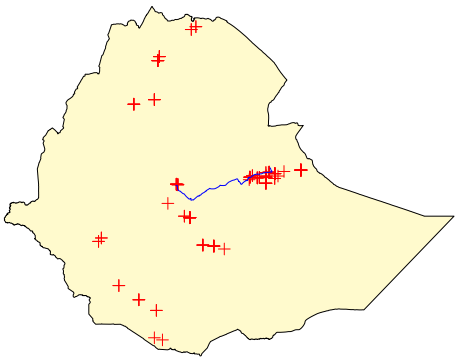
3.3 Creating Spatial Objects

Spatial objects can be created from non-spatial ones, for example using coordinate values from a table. For `SpatialPointsDataFrame` the way to produce it is straight forward. We will take from the data sets a table containing observations of *Juniperus procera* in Ethiopia. Such table content coordinate values in the columns `decimalLongitude` and `decimalLatitude`.

```
Juniperus <- read.csv("data/Distrib.csv")
Juniperus <- subset(Juniperus, paste(species) == "Juniperus")

coordinates(Juniperus) <- ~ Longitude + Latitude # get spatial
proj4string(Juniperus) <- CRS("+proj=longlat +datum=WGS84") # get projection
```

A bit longer is the way to create spatial objects containing lines or polygons. For example, the data set `Route.csv` contains coordinates values along the way from Addis Ababa to Dire Dawa.

```
Route <- read.csv("data/Route.csv")
# Get Line object
Route <- Line(Route)
# Get Lines object (can include many objects of class Line)
Route <- Lines(list(Route), ID="AddisToDire")
# Get SpatialLines (lines with spatial reference)
Route <- SpatialLines(list(Route),
                proj4string=CRS("+proj=longlat +datum=WGS84"))
# Get SpatialLinesDataFrame (SpatialLines object with an attribute table)
Route <- SpatialLinesDataFrame(Route,
                data.frame(Start="Addis Abeba", End="Dire Dawa"),
                match.ID=FALSE)

plot(Ethiopia, col="lemonchiffon")
plot(Juniperus, col="red", add=TRUE) # Juniperus procera points
plot(Route, col="blue", add=TRUE) # Route Addis Abeba to Dire Dawa
```



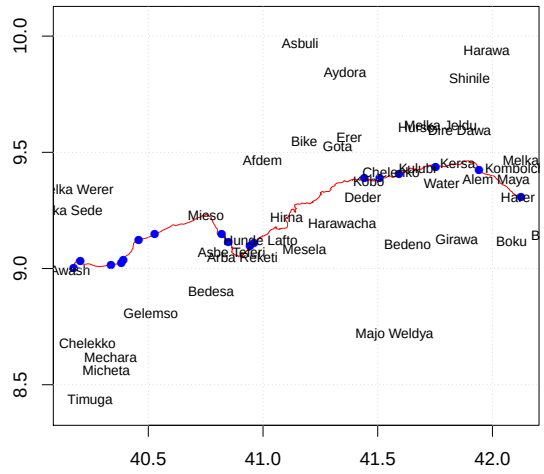
The way to get polygons in spatial objects is analogous but using the sequence of functions `Ploygon`, `Polygons`, `SpatialPolygons` and `SpatialPolygonsDataFrame`. Once done the transformations, we are able to write the output as GIS data sets using the function `writeOGR`.

```
# write ESRI shapefiles
writeOGR(Juniperus, dsn="results", layer="Juniperus", driver="ESRI Shapefile",
                overwrite_layer=TRUE)
writeOGR(Route, dsn="results", layer="Route", driver="ESRI Shapefile",
                overwrite_layer=TRUE)
# write KML files (Google Earth)
writeOGR(Juniperus, dsn="results/Juniperus.kml", layer="points", driver="KML",
                overwrite_layer=TRUE)
writeOGR(Route, dsn="results/Route.kml", layer="lines", driver="KML",
                overwrite_layer=TRUE)
```

3.4 Reading GPX Files

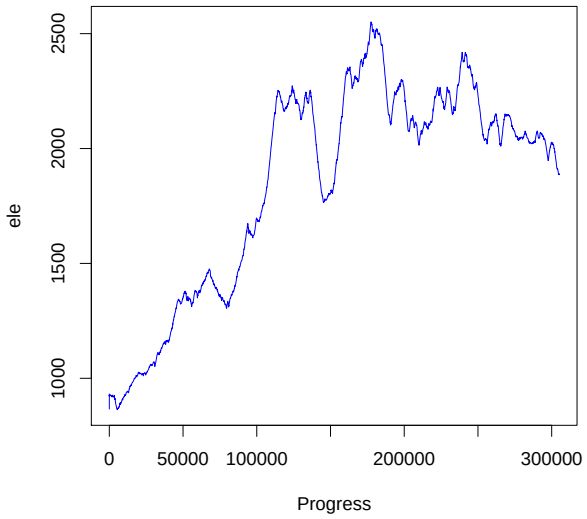
The function `readOGR` have also routines for the import of GPX files, which is an exchange file format used byGarmin GPS devices. Two examples are provided here, one for a tracklog and one for waypoints. The tracklog was collected during a trip from Awash to Harer. The waypoints are sites of interest marked during the trip.

```
GPStrack <- readOGR(dsn="vectors/track_awash_harar.gpx", layer="tracks")

## OGR data source with driver: GPX
## Source: "vectors/track_awash_harar.gpx", layer: "tracks"
## with 1 features
## It has 13 fields

GPSwaypoints <- readOGR(dsn="vectors/waypoints_awash_harar.gpx",
        layer="waypoints")

## OGR data source with driver: GPX
## Source: "vectors/waypoints_awash_harar.gpx", layer: "waypoints"
## with 17 features
## It has 23 fields

plot(GPStrack, col="red")
points(GPSwaypoints, pch=16, col="blue")
text(Towns, labels=Towns$TOWN_NAME, cex=0.7)
# Making map nicier
box()
axis(1)
axis(2)
grid()
```



In the case of track logs, there is also the possibility to load single points of the tracks with the respective records (altitude and time). In the following exercise we attempt to produce a profile of the trip according to the altitude values recorded by the device along the way.

```
GPStrack_points <- readOGR(dsn="vectors/track_awash_harar.gpx",
                layer="track_points")

## OGR data source with driver: GPX
## Source: "vectors/track_awash_harar.gpx", layer: "track_points"
## with 6521 features
## It has 26 fields

# Calculating progress in m
GPStrack_points <- spTransform(GPStrack_points,
                CRS("+proj=utm +zone=37 +north +datum=WGS84"))
Coords <- coordinates(GPStrack_points)
Progress <- sqrt(diff(Coords[,1])^2 + diff(Coords[,2])^2)
GPStrack_points$Progress <- cumsum(c(0, Progress))
# Plot track profile
with(GPStrack_points@data, plot(Progress, ele, type="l", col="blue"))
```



Exercises

- Calculate proportion of surface of the country covered by water bodies.
- Calculate the distance between Addis Abeba and Dire Dawa by using the provided track log. Compare including the altitude.
- Make a profile of the change in velocity along the track log.

4 Raster Data Files in R

Raster Data are collections of pixels containing numeric values. Such data sets are recognised as images (e.g. satellite imagery, aerial photographs) but may also content variables other than values for colors such as altitude, climatic conditions, terrain, etc.

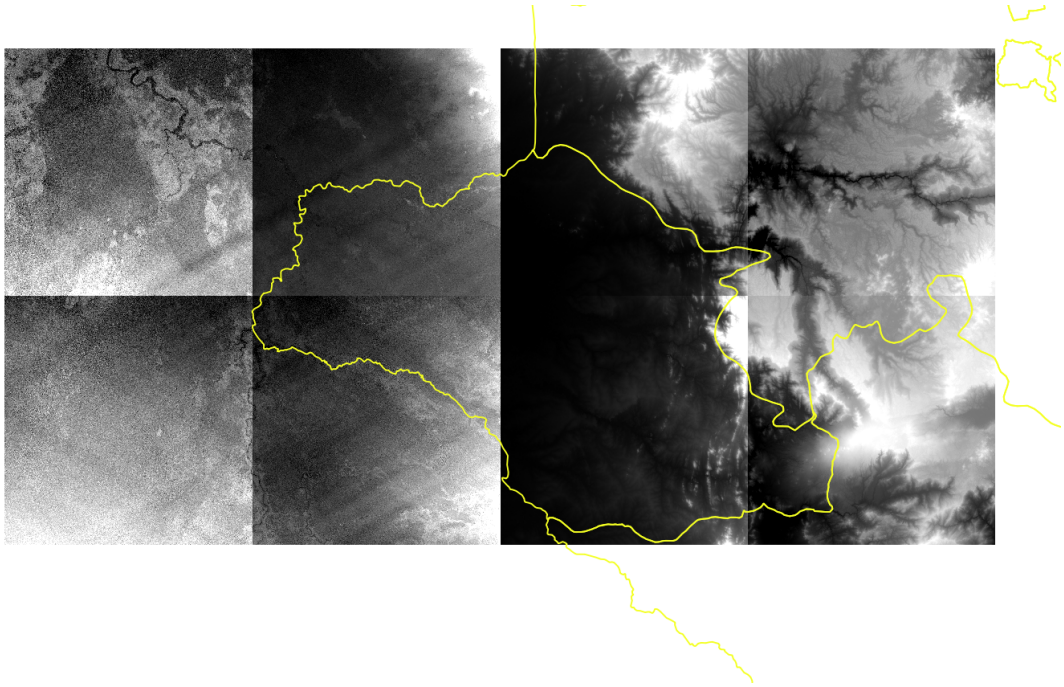
4.1 Import and Basic Process of Rasters

During this workshop we will use the package `raster`, which provides de classes `Raster*` (e.g. `RasterLayer`, `RasterStack`, etc.).

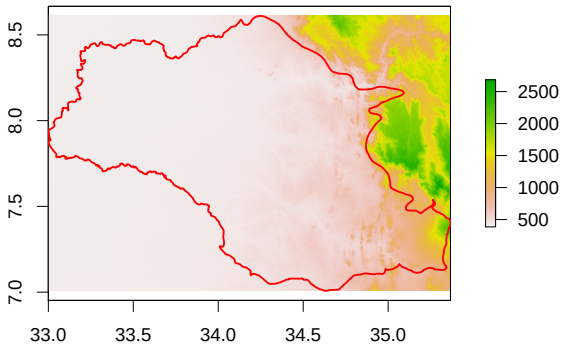
In the first example, we will import elevation models obtained form ASTER GDEM (a product of METI and NASA), which were downloaded from EarthExplorer. Such models have a resolution of 1 arcsec. Since this data set is distributed in 1 by 1 degree tiles, we require for example 8 tiles to display elevation landscape of the Gambela Region (displayed in the next figure).

So, the very first step is to load those tiles in the `workspace` and then to merge them into one data set.

```
library(rgdal)
library(raster)
Gambela <- subset(Regions, NAME_1 == "Gambela") # The region Gambela
Files <- c(
                "N07_E032.tif",
                "N07_E033.tif",
                "N07_E034.tif",
                "N07_E035.tif",
                "N08_E032.tif",
                "N08_E033.tif",
                "N08_E034.tif",
                "N08_E035.tif")
```



```
Files <- file.path("rasters", Files) # relative paths of single tiles
DEM <- lapply(Files, raster) # load using function raster
DEM <- do.call(merge, DEM) # merging tiles in one data set
Ext <- extent(bbox(Gambela))
DEM <- crop(DEM, Ext, snap="out") # cropping by the extension of Gambela
# Display result
plot(DEM)
plot(Gambela, lwd=2, border="red", add=TRUE)
```



Notice that the object `Files` only contain the relative paths of the GeoTiff files (relative to the `working directory`), while `lapply` executes the function `raster` to import those files and store them as elements of a list (object `DEM`). The function `do.call` apply the method `merge` for the tiles in `DEM`, merging them into one `RasterLayer`. Finally `crop` cuts the resulting raster according to the extension of the Gambela Region.

Hint

In many cases a **Raster*** object does not contain any cell (pixel) values in (RAM) memory, but only the parameters that describe the **RasterLayer**. Therefore saving an **R Image** (workspace) will not necessarily include the values of the rasters itself.

4.2 Raster Calculation

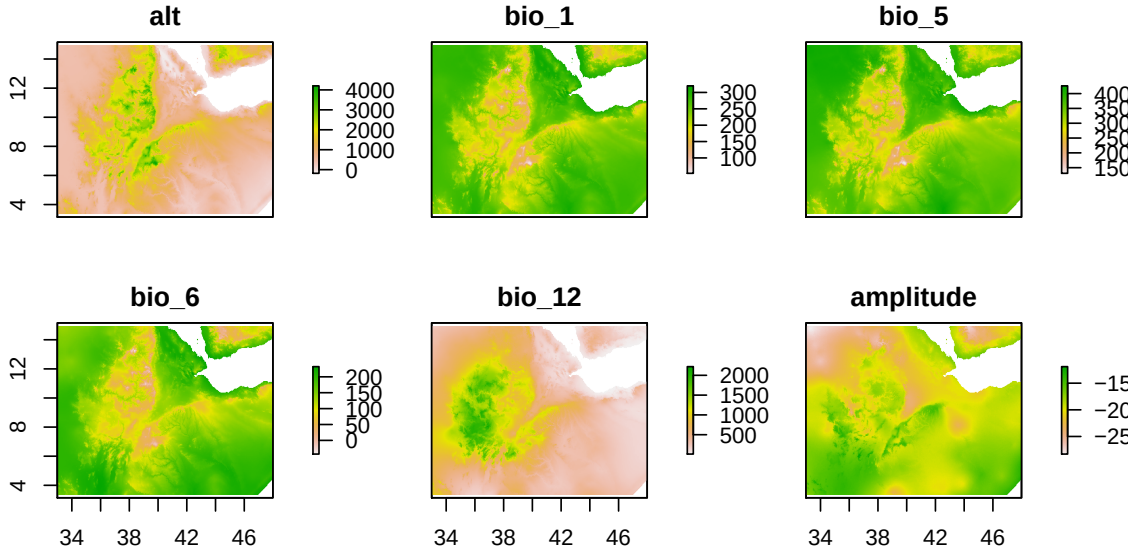
Rasters contain numeric variables representing a continuous distribution in the surface (Hijmans, 2017). For calculations using rasters, they may be considered as matrices, provided that rasters have the very same resolution, the same projection and the same extent. In other words, they may have a perfect spatial matching of pixels. Such kind of files can be contained by objects of the class `RasterBrick` (single multiple-layer file) or `RasterStack` (single files for every layer).

In the following example, we will work with a subset of the WorldClim database. This database provide an elevation model and climatic variables obtained from interpolation of historical records from climatic stations distributed worldwide (for more details, see Hijmans et al. 2005).

```
Files <- c(
                "alt.grd", # Altitude (m asl.)
                "bio_1.grd", # Mean temperature (degrees*10)
                "bio_5.grd", # Max temperature in warmest month (degrees*10)
                "bio_6.grd", # Min temperature in coldest month (degrees*10)
                "bio_12.grd") # Annual precipitation (mm)
Files <- file.path("rasters", Files)
# Creating stack
ClimData <- stack(Files)
ClimData <- crop(ClimData, extent(Ethiopia), snap="out")
```

Following import, we may convert temperature values into Celsius degrees for further display. We will additionally calculate the amplitude as the difference between the minimum temperature during the coldest month (variable `bio_6`) and the maximum temperature during the warmest month (varaible `bio_5`).

```
ClimData[["amplitude"]] <- ClimData[["bio_6"]] - ClimData[["bio_5"]]

plot(ClimData)
```



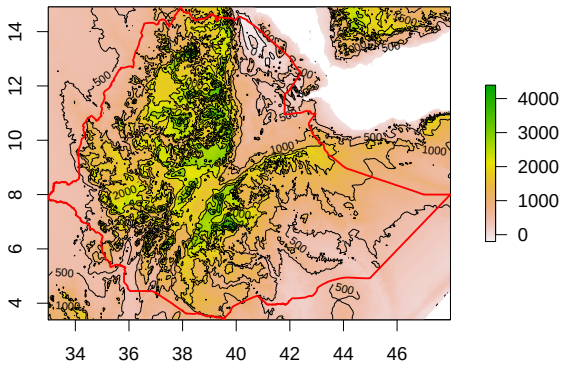
Such operations can be also carried out using the functions `overlay` or the function `calc`. We calculate amplitude again as follows:

```
Vars <- subset(ClimData, c("bio_6","bio_5")) # subset of variables
Amplitude <- overlay(Vars, fun=function(x,y) x-y) # alternative 1
Amplitude <- calc(Vars, fun=function(x) x[1]-x[2]) # alternative 2
```

4.3 Working with Digital Elevation Models

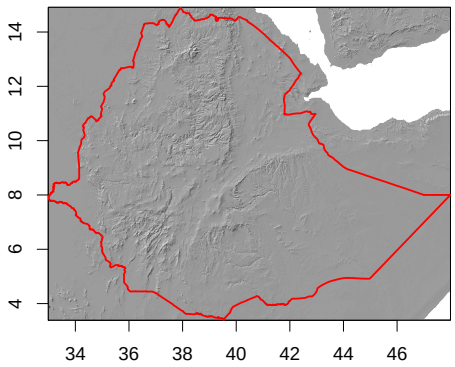
Further exercises use an elevation model downloaded from the WorldClim database (subset for Eastern Africa). We will then display the extention of Ethiopia.

```
Altitude <- raster("rasters/alt.grd")
Ext <- extent(bbox(Ethiopia))
Altitude <- crop(Altitude, Ext, snap="out")
# Display
plot(Altitude)
contour(Altitude, add=TRUE)
plot(Ethiopia, border="red", lwd=2, add=TRUE)
```



The function `terrain` offers options to calculate the slope and exposition after DEMs and work using "moving windows" (if you are interested on those techniques, look at the help for the function `focal`). In addition we will produce a display of hill shades.

```
Slope <- terrain(Altitude, opt="slope")
Aspect <- terrain(Altitude, opt="aspect")
Hill <- hillShade(Slope, Aspect, 40, 315)
# Display
plot(Hill, col=grey(0:100/100), legend=FALSE)
plot(Ethiopia, border="red", lwd=2, add=TRUE)
```



4.4 Moving Windows

Moving windows is a technique to find relations between pixels in rasters and their neighbours (operations on continuous fields according to Burrough & McDonnell 2016). Such kind of analyses may be required for example to calculate contagion in catogorical variables (Li & Reynolds, 1993; Ricotta *et al.*, 2003), but it may be also useful for estimations of landscape parameters, such as slope or the Topographic Position Index. In following

example, we will apply it to the calculation of TPI for Ethiopia at a scale of 20,000 m (Weiss, 2001). The very first step will be to estimate the resolution of the raster in meters by converting it to UTM.

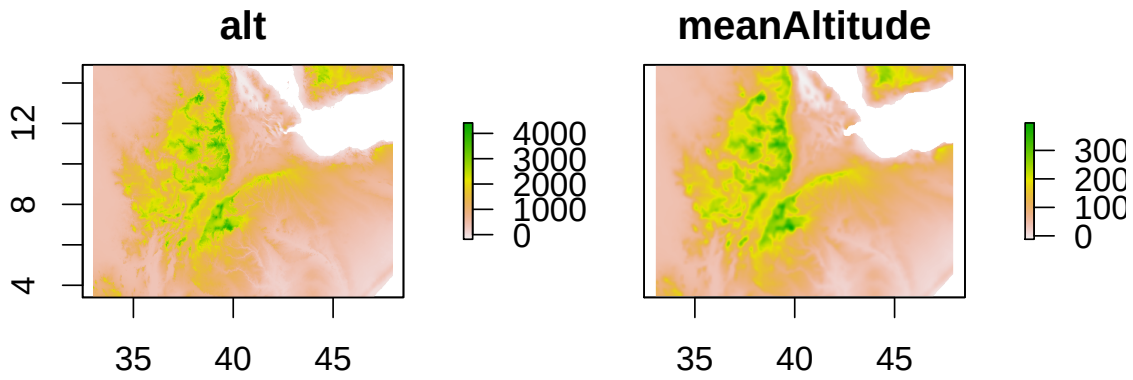
```
(Altitude_utm <- projectRaster(Altitude,
                                        crs="+proj=utm +zone=37 +north +datum=WGS84"))

## class       : RasterLayer
## dimensions  : 1410, 1835, 2587350  (nrow, ncol, ncell)
## resolution  : 916, 922  (x, y)
## extent      : -173213, 1507647, 371557.3, 1671577  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=37 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : alt
## values      : -182.5247, 4433.818  (min, max)
```

Here the resolution is approximatelly 919 m per pixel, thus 20,000 m will be covered by 21.7627856. We will then create a window of 21 by 21 pixels and weight them by their sum.

```
wind <- matrix(rep(1, 21*21), ncol=21)
wind <- wind/sum(wind)
```

Then we use the function `focal` to sweep the window along the terrain.

```
mean_Altitude <- focal(Altitude, wind, filename="results/meanAltitude.tif",
                progress="window", overwrite=TRUE)
plot(stack(Altitude, mean_Altitude))
```

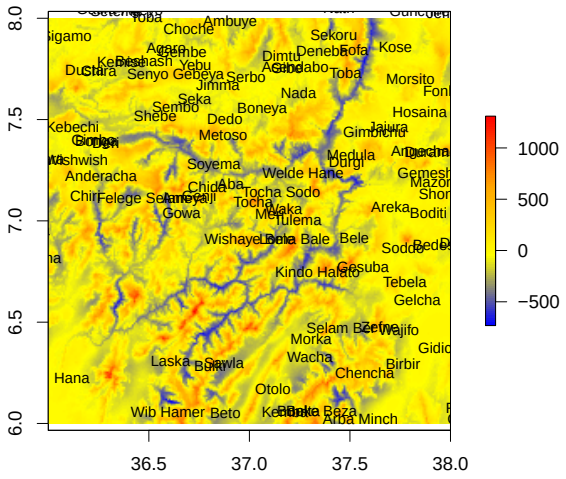


The mean altitude have a lower top compare with the original altitude values and is more difuse in the image, since every pixel contains now the mean value of 21 by 21 pixels from the original raster, with the summarized pixel in the center of the matrix. The TPI is then defined as the altitude of the pixel minus the focal mean Weiss (2001).

```
tpi_20000 <- overlay(Altitude, mean_Altitude,
                fun=function(x,y) as.integer(round(x - y)),
                filename="results/tpi20000.tif", progress="window", overwrite=TRUE)
my.colors <- colorRampPalette(c("blue","yellow","orange","red"))
plot(tpi_20000, col=my.colors(99))
```

Extreme negative values (blues in the map) represent pixels located very deep withing the surrounding ones, situation found in the canyons, for instance. High values of TPI (red colors) indicate peaks of mountains. Values around zero (yellow colors) are pixels located in flatlands. This situation can be more evident making a close up of accidented areas.

```
plot(crop(tpi_20000, extent(c(36,38,6,8))), col=my.colors(99))
text(coordinates(Towns), labels=Towns$TOWN_NAME, cex=0.8)
```



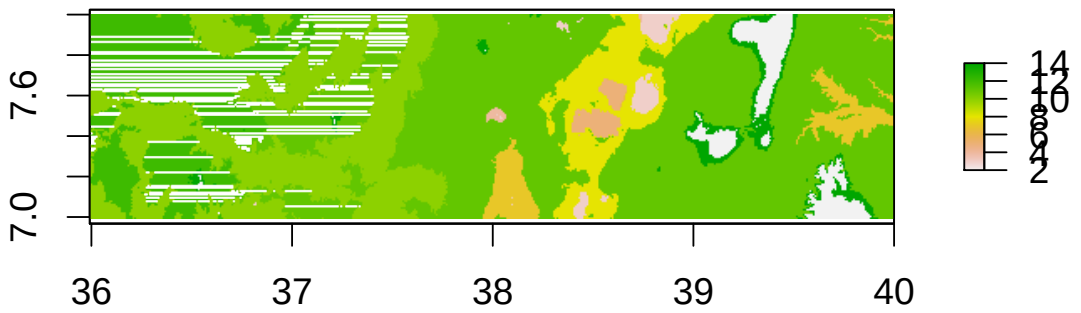
Hint

Rounding and converting numeric into integer values in raster not only save storage size but also speeds calculations. See the case of tempertature in the WorldClim database (Hijmans *et al.*, 2005).

4.5 Rasterizing and Masking

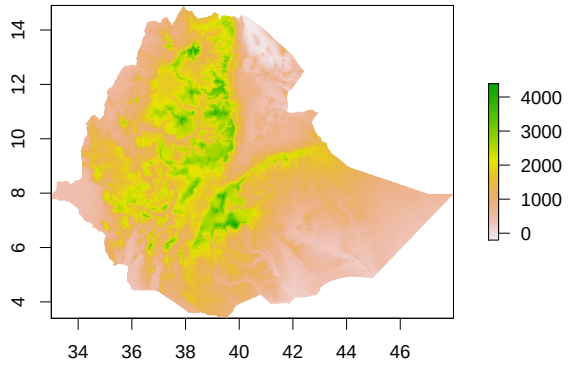
Information contained as attributes of s `SpatialPolygonsDataFrame` (slot `data`) can be also transfered to raster files. In the following example, we will use a shapefile of the potential natural vegetation of Ethiopia (obtained from Friis et al. 2010). In this file ther is a numeric index for vegetation units, included in the attribute table. We will just work with a strip of the country to save some processing time.

```
DEM <- crop(ClimData[["alt"]], extent(c(36,40,7,8)), snap="out")
Vegetation <- readOGR(dsn="vectors", layer="ET_Vegetation")
Vegetation_raster <- rasterize(Vegetation, DEM, "value")
plot(Vegetation_raster)
```



While the function `crop` is cutting rasters according to extensions, therefore in rectangular shapes, you may also desire to cut rasters according to shapes of polygons, for example to get all altitude values of pixels included in Ethiopia.

```
DEM <- mask(ClimData[["alt"]], Ethiopia)
DEM <- crop(DEM, extent(Ethiopia)) # Adjust the extent of raster
plot(DEM)
```

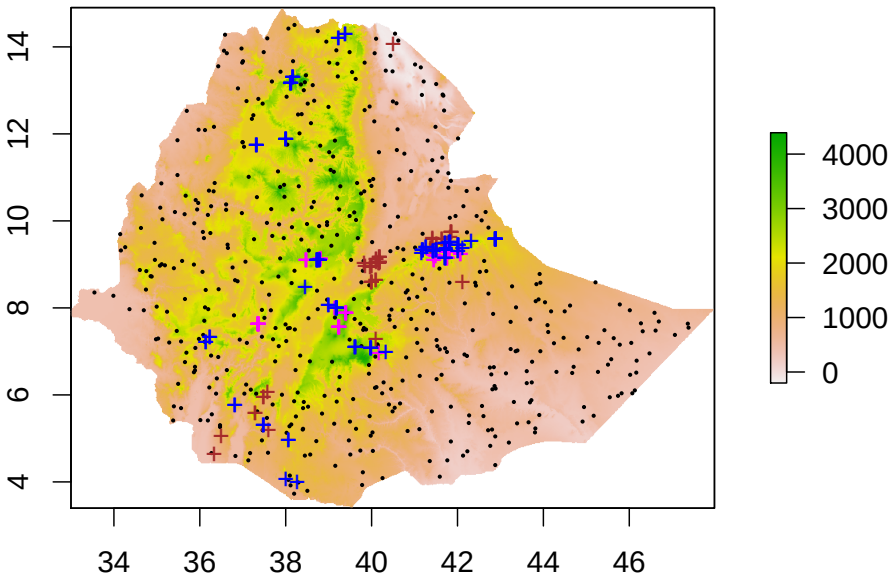


4.6 Extracting Data from Rasters

In the folder `data` you will find three files in csv format (comma separated values). Those are tables containing observations of occurrence of three woody species, namely *Acacia mellifera*, *Hagenia abyssinica* and *Juniperus procera*. Such observations were obtained from herbarium vouchers, which data are downloaded from Global Biodiversity Information Facility by using the package `rgbif`. The mentioned data sets contain those observations that are geo-referenced and made in Ethiopia.

In the following `command lines` we will display the location of observations for each species. Additionally, we will select a set of 500 random points distributed across Ethiopia in order to explore climatic conditions in the whole country (the background). Random points will be selected by using the function `randomPoints` of the package `dismo`.

```
Distrib <- read.csv("data/Distrib.csv", stringsAsFactors=FALSE)

library(dismo)
set.seed(246)
Background <- randomPoints(DEM, 500)

# Colors for plots
COL <- c("brown","magenta","blue")[match(paste(Distrib$species),
                                c("Acacia","Hagenia","Juniperus"))]

plot(DEM)
points(Background, pch=20, cex=0.3)
points(Distrib[,c("Longitude","Latitude")], pch="+", col=COL)
```

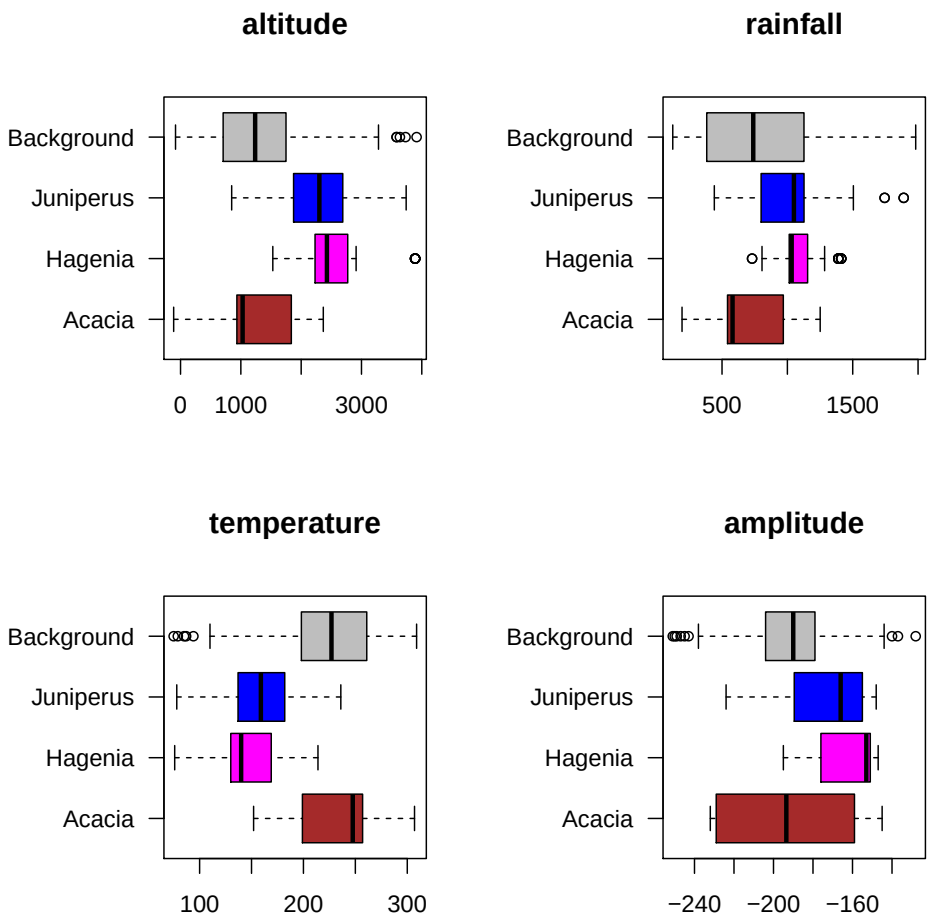


Botanists and plant ecologists will wander, if some climatic conditions may determine the occurrence of those species. In other words, which are the preferences of those species regarding climatic conditions (ecological answer). So, we will create a `SpatialPointsDataFrame` object including all the points shown in the previous map in order to extract the values of climatic variables from the object `ClimData`.

```
# Merge background and species distribution
Background <- data.frame(species="Background", Longitude=Background[,1],
                  Latitude=Background[,2], stringsAsFactors=FALSE)
Distrib <- do.call(rbind, list(Distrib, Background))

# Now convert it to spatial object
coordinates(Distrib) <- ~ Longitude + Latitude
proj4string(Distrib) <- CRS("+proj=longlat +datum=WGS84")
```

With this object we can now extract the values of climatic variables using the function `extract`.

```
Distrib <- extract(ClimData, Distrib, sp=TRUE)
Distrib$species <- factor(Distrib$species, levels=c("Acacia","Hagenia",
                                "Juniperus","Background"))

par(mfrow=c(2,2), las=1, mar=c(3,7,5,1))
boxplot(alt ~ species, data=Distrib, col=c("brown","magenta","blue","grey"),
                  main="altitude", horizontal=TRUE)
boxplot(bio_12 ~ species, data=Distrib, col=c("brown","magenta","blue","grey"),
                  main="rainfall", horizontal=TRUE)
boxplot(bio_1 ~ species, data=Distrib, col=c("brown","magenta","blue","grey"),
                  main="temperature", horizontal=TRUE)
boxplot(amplitude ~ species, data=Distrib, col=c("brown","magenta","blue", "grey"),
                  main="amplitude", horizontal=TRUE)
```

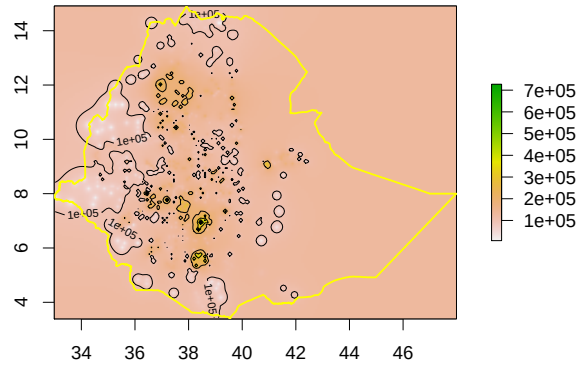


After a quick view on the resulting boxplots, we can stat that *H. abyssinica* and *J. procera* prefer high altitudes and places with relatively high rainfall, low temperatures and low amplitude. On the other hand, *A. mellifera* is a species growing in lowlands, in places with low rainfall and high temperatures, while it seems to be relatively indifferent to amplitude of temperature.

4.7 Interpolation and Regression

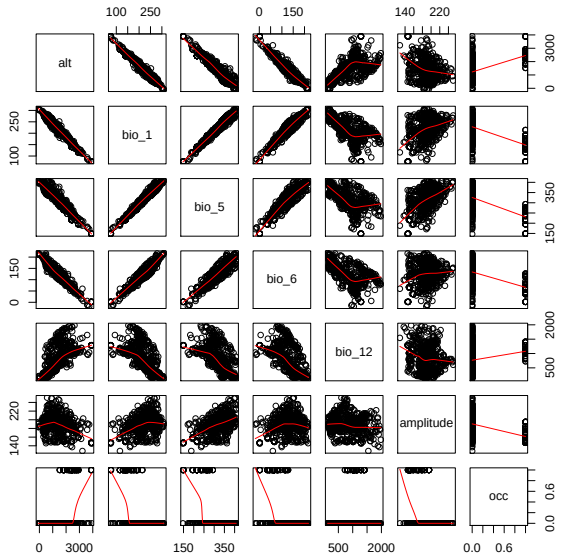
A series of interpolation methods are available in packages. Those techniques are explained into detail by Hengl *et al.* (2004) and Hengl (2009).

```
Towns <- spTransform(Towns, CRS(proj4string(DEM))) # Equalize spatial references
DEM2 <- aggregate(DEM, 10, mean)
POP <- rasterize(Towns, DEM2, "POPTOT", sum)
POP <- as(POP, "SpatialPointsDataFrame")
names(POP) <- "POPTOT"
POP <- subset(POP, POPTOT > 0)

library(gstat)
Population <- gstat(formula=POPTOT ~ 1, locations=POP)

# Reduce resolution of DEM
IDW <- interpolate(DEM2, Population)

plot(IDW)
contour(IDW, add=TRUE)
plot(Ethiopia, border="yellow", lwd=2, add=TRUE)
```



We can also explore the dependence of occurrence of a species (e.g. *Hagenia abyssinica*) on environmental variables such as mean annual temperature (bio_1), annual rainfall (bio_12), and minimum temperature in the coldest month (bio_6).

```
Hagenia <- subset(Distrib, paste(species) %in% c("Background","Hagenia"))
Hagenia$occ <- as.integer(paste(Hagenia$species) == "Hagenia")
pairs(Hagenia@data[,-1], panel=panel.smooth)
```

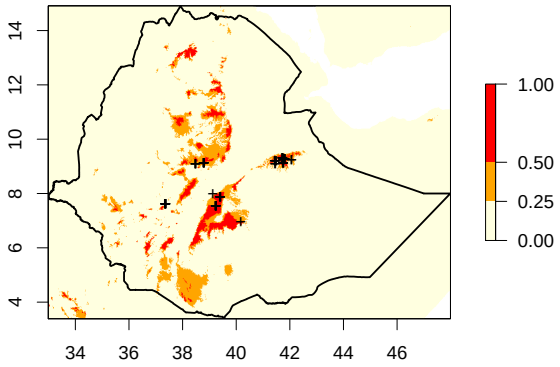



```
Mod_glm <- glm(occ ~ bio_1 + bio_12 + bio_6, binomial, Hagenia@data)
summary(Mod_glm)

##
## Call:
## glm(formula = occ ~ bio_1 + bio_12 + bio_6, family = binomial,
##     data = Hagenia@data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.74847  -0.36647  -0.15072  -0.07974   2.36691
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept) 14.2719752  1.8301443   7.798 6.28e-15 ***
## bio_1       -0.1182564  0.0158755  -7.449 9.41e-14 ***
## bio_12      -0.0013888  0.0005275  -2.633  0.00846 **
## bio_6        0.0741402  0.0142767   5.193 2.07e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 500.20  on 588  degrees of freedom
## Residual deviance: 270.68  on 585  degrees of freedom
## AIC: 278.68
##
## Number of Fisher Scoring iterations: 7
```

This model can than used to estimate probabilities of occurrence of *H. abyssinica* depending on the mentioned variables.

```
Mod_spatial <- predict(ClimData, Mod_glm, type="response")
plot(Mod_spatial, breaks=c(0, 0.25, 0.5, 1),
                col=c("lightyellow","orange","red"))
plot(Hagenia[Hagenia$occ == 1,], pch="+", add=TRUE)
plot(Ethiopia, lwd=2, add=TRUE)
box()
```



Of course, this is just an approach to spatial modelling. A series of steps (e.g. variable selection and spatial auto-correlation) are required to obtain a final model. More details can be found in (Hengl, 2009).

Exercises

- Create a `SpatialPointsDataFrame` for the observations of each woody species as explained in chapter 4.4. Plot them in a map with a personalised layout and add a legend (look the help for `legend`).
- Extract altitude (or other environmental variable) for each type of Potential Natural Vegetation from Ethiopia.
- Check in the `rasters` folder for the SRTM files. Process them to get a digital elevation model with the shape of Ethiopia.
- In the tracklog Marigat to Niahururu, extract altitude values from the provided elevation model (`alt.grd`). Compare values from elevation model with those measured by the GPS device.
- Draw an altitude profile for the trip from Addis Ababa to Dire Dawa extracting altitude values from the provided elevation model. Use for the profile alternativelly distance progress and longitude as x axis.
- Check the climatic distribution of towns in Ethiopia, comparing with the background.

5 Bibliography

Alvarez, M. & Borchardt, P. (2015). *Quick introduction to R and the work with GIS*.

Burrough, P.A. & McDonnell, R.A. (2016). *Principles of geographical information systems*.

Crawley, M.J. (2007). *The R book*. John Wiley, Chichester.

Dormann, C.F. & Kühn, I. (2008). *Angewandte Statistik für die biologischen Wissenschaften*. vol. 2. UFZ, Leipzig-Halle.

Hengl, T. (2009). *A practical guide to geostatistical mapping of environmental variables*. Open Access.

Hengl, T., Heuvelink, G.B.M. & Stein, A. (2004). A generic framework for spatial prediction of soil variables based on regression-kriging. *Geoderma*, 120, 75–93.

Hijmans, R. (2017). *Spatial data analysis with R*. vol. Release 0.1.

Hijmans, R.J., Cameron, S.E., Parra, J.L., Jones, P.G. & Jarvis, A. (2005). Very high resolution interpolated climate surfaces for global land areas. *International Journal of Climatology*, 25, 1965–1978.

Hörmann, G. & Unkel, I. (2015). *Analysis of ecological data with R*.

Li, H. & Reynolds, J.F. (1993). A new contagion index to quantify spatial patterns of landscapes. *Landscape Ecol*, 8, 155–162.

Ligges, U. (2008). *Programmieren mit R*. Springer, Berlin.

Ligges, U. & Fox, J. (2008). How can I avoid this loop or make it faster? *R News*, 8, 46–50.

Logan, M. (2010). *Biostatistical design and analysis using R. A practical guide*. Wiley, Chichester.

Obe, R.O. & Hsu, L. (2015). *PostGIS in Action*. Manning.

Ricotta, C., Corona, P. & Marchetti, M. (2003). Beware of contagion! *Landscape and Urban Planning*, 62, 173–177.

Weiss, A.D. (2001). Topographic position and landforms analysis. http://www.jennessent.com/downloads/tpi-poster-tnc_18x22.pdf.

Wickham, H. (2015). *R Packages*. O'Reilly, Cambridge.

Zerihun Woldu (2017). *Comprehensive analysis of vegetation and ecological data*.