Lesson 2

Jan Verbesselt, Loïc Dutrieux, Ben De Vries, Sytze de Bruyn

October 24, 2013

Abstract

This is an introduction to the course Applied Geo-Scripting where we will explore the potential of R and libraries which enable reading, writing, analysis, and visualisation of spatial data.

1 Today's learning objectives

- Read, write, and visualize spatial data (vector/raster) using a script
- Find libraries which offer spatial data handling functions
- Learn to include functions from a library in your script

2 Set Your Working Directory and Load Your Libraries

2.1 Set the Working Directory

Let's do some basic set up first.

- Create a folder which will be your working directory e.g. *Lesson2*
- Create an R script within that folder
- Set your working directory to the *Lesson2* folder
- Create a *data* folder within your working directory

In the code block below type in the file path to where your data is being held and then (if you want) use the setwd() (set working directory) command to give R a default location to look for data files.

```
R> setwd("yourworkingdirectory")
R> #This sets the working directory (where R looks for files)
R> #getwd()
R> # Double check your working directory

R> datdir <- file.path("data") ## path
```

2.2 Load Libraries

Next we will load a series of R packages that will give the functions we need to complete all the exercises in lesson 1 and 2. For this exercise all of the packages should (hopefully) be already installed on your machine (?). We will load them below using the library() command. I also included some comments describing how we use each of the packages in the exercises.

```
R> #----Packages for Reading/Writing/Manipulating Spatial Data---
R> library(rgdal) # reading shapefiles and raster data
R> library(rgeos)
R> library(maptools)
R> library(spdep)   # useful spatial stat functions
R> library(spatstat) # functions for generating random points
R> library(raster)
R> #---Packages for Data Visualization and Manipulation---
R> library(ggplot2)
R> library(reshape2)
R> library(scales)
```

3 Read, plot, and explore spatial data

3.1 Read in a Shapefile

The most flexible way to read in a shapefile is by using the `readOGR` command. This is the only option that will also read in the .prj file associated with the shapefile. NCEAS has a useful summary of the various ways to read in a shapefile: http://www.nceas.ucsb.edu/scicomp/usecases/ReadWriteESRIShapeFiles I recommend always using `readOGR()`.

Read OGR can be used for almost any vector data format. To read in a shapefile, you enter two arguments:

- dsn: the directory containing the shapefile (even if this is already your working directory)
- layer: the name of the shapefile, without the file extension

```
R> download.file('http://rasta.r-forge.r-project.org/kenyashape.zip',
+        file.path(datdir, 'kenyashape.zip'))
R> unzip(file.path(datdir, 'kenyashape.zip'), exdir = datdir)
R> kenya <- readOGR(dsn = datdir, layer = 'kenya')
```

3.2 Plotting the Data

Plotting is easy, use the `plot()` command:

```
R> plot(kenya)
```

Obviously there are more options to dress up your plot and make a proper map/graphic. A common method is to use `spplot()` from the sp package. However I prefer to use the functions available in the ggplot2 package as I think they are more flexible and intuitive. We will address maps and graphics later in the in the class. For now, let us move onto reading in some tabular data and merging that data to our shapefile (similar to the join operation in ArcGIS).

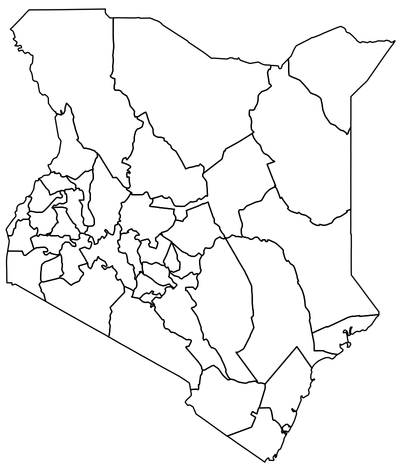

Figure 1: Adminstrative boundaries of Kenya

Here is an example for downloading of administrative boundaries for any country. This will be useful for the exercise.

3.3 Exploring the Data within the vector file

We can explore some basic aspects of the data using `summary()` and `str()`. Summary works on almost all R objects but returns different results depending on the type of object. For example if the object is the result of a linear regression then summary will give you the coefficient estimates, standard errors, t-stats, R^2, et cetera.

```
R> summary(kenya)
```

```
Object of class SpatialPolygonsDataFrame
Coordinates:
        min       max
x 33.908859 41.899078
y -4.678047  4.629333
Is projected: FALSE
proj4string : [+proj=longlat +ellps=clrk80 +no_defs]
Data attributes:
    ip89DId               ip89DName
 Min.   :1010   Baringo        : 1
 1st Qu.:3050   Bugoma         : 1
 Median :5030   Busia          : 1
 Mean   :5090   Elgeyo-Marakwet: 1
 3rd Qu.:7060   Embu           : 1
 Max.   :8030   Garissa        : 1
                (Other)        :35
```

```
R> str(kenya,2)
```

```
Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        41 obs. of  2 variables:
  ..@ polygons   :List of 41
  ..@ plotOrder  : int [1:41] 17 36 21 19 12 15 20 14 26 34 ...
  ..@ bbox       : num [1:2, 1:2] 33.91 -4.68 41.9 4.63
  .. ..- attr(*, "dimnames")=List of 2
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

As mentioned above, the `summary()` command works on virtually all R objects. In this case it gives some basic information about the projection, coordinates, and data contained in our shapefile

The `str()` or structure command tells us how R is actually storing and organizing our shapefile. This is a useful way to explore complex objects in R. When we use `str()` on a spatial polygon object, it tells us the object has five 'slots':

1. *data*: This holds the data.frame
2. *polygons*: This holds the coordinates of the polygons
3. *plotOrder*: The order that the coordinates should be drawn
4. *bbox*: The coordinates of the bounding box (edges of the shape file)

5. *proj4string*: A character string describing the projection system

The only one we want to worry about is data, because this is where the data.frame() associated with our spatial object is stored. We access slots using the @ sign.

```
R> #--------------------------ACCESS THE SHAPEFILE DATA-------------------------------
R> dsdat <- as(kenya, "data.frame")  # extract the data into a regular data.frame
R> head(dsdat)

  ip89DId ip89DName
0    1010   Nairobi
1    2010    Kiambu
2    2020 Kirinyaga
3    2030   Muranga
4    2040 Nyandaura
5    2050     Nyeri

R> kenya$new <- 1:nrow(dsdat) # add a new colunm, just like adding data to a data.frame
R> head(kenya@data)

  ip89DId ip89DName new
0    1010   Nairobi   1
1    2010    Kiambu   2
2    2020 Kirinyaga   3
3    2030   Muranga   4
4    2040 Nyandaura   5
5    2050     Nyeri   6
```

4 Create Random Points and Extract as a Text File

We are going to do a point in polygon spatial join. However before we do that we are going to generate some random points. We will use the function `runifpoint()` from the spatstat package. This function creates N points drawn from a spatial uniform distribution (complete spatial randomness) within a given bounding box. The bounding box can be in a variety of forms but the most straightforward is simply a four element vector with *xmin* (the minimum x coordinate), *xmax*, *ymin*, and *ymax*. In the code below we will extract this box from our Kenya data set, convert it to a vector, generate the points, and then plot the points on top of the Kenya map.

```
R> #--------------------------GENERATE RANDOM POINTS-------------------------------
R> win <- bbox(kenya) #the bounding box around the Kenya dataset
R> win

        min       max
x 33.908859 41.899078
y -4.678047  4.629333

R> win <- t(win) #transpose the bounding box matrix
R> win
```

```
           x         y
min 33.90886 -4.678047
max 41.89908  4.629333

R> win <- as.vector(win) #convert to a vector for input into runifpoint()
R> win

[1] 33.908859 41.899078 -4.678047  4.629333

R> dran1 <- runifpoint(100, win = as.vector(t(bbox(kenya)))) #create 100 random points

R> win <- extent(kenya)
R> dran2 <- runifpoint(n = 100, win = as.vector(win))

R> plot(kenya)
R> plot(dran1, add = TRUE, col = "red")
R> plot(dran2, add = TRUE, col = "blue", pch = 19, cex = 0.5)
```

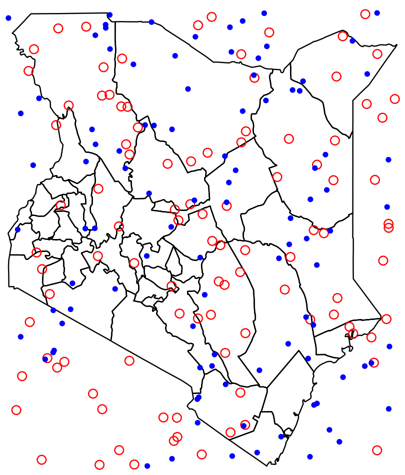

Figure 2: Random points within the Kenya shape file

Now that we have created some random points, we will extract the x coordinates (longitude), y coordinates (latitude), and then simulate some values to go with them.

```
R> #--------------------------CONVERT RANDOM POINTS TO DATA.FRAME------------------
R> dp <- as.data.frame(dran1) #This creates a simple data frame with 2 colunms, x and y
R> head(dp)
```

```
         x          y
1 34.43988  3.8844933
2 38.79437  2.1980005
3 39.59339 -1.0526827
4 38.23731 -0.8857542
5 35.75243 -0.3644901
6 38.77941 -3.8329354

R> #Now we will add some values that will be aggregated in the next exercise
R> dp$values<-rnorm(100,5,10)
R> #generates 100 values from a Normal distribution with mean 5, and sd-10
R> head(dp)

         x          y     values
1 34.43988  3.8844933   1.902255
2 38.79437  2.1980005   4.148144
3 39.59339 -1.0526827 -12.823097
4 38.23731 -0.8857542  -2.254233
5 35.75243 -0.3644901  10.936621
6 38.77941 -3.8329354 -10.917073
```

5 Do a Point in Polygon Spatial Join

In the last exercise we generated some random points along with some random values. Now we will read that data in, convert it to a shapefile (or a SpatialPointsDataFrame object) and then do a point in polygon spatial join. The command for converting coordinates to spatial points is *SpatialPointsDataFrame()*

```
R> #--------------------------CONVERT RANDOM POINTS TO SPATIAL POINTS DATAFRAME----
R> dsp <- SpatialPointsDataFrame(coords = dp[,c('x','y')], data = data.frame('values' = dp
R> summary(dsp)

Object of class SpatialPointsDataFrame
Coordinates:
       min       max
x 34.07705 41.853881
y -4.65340  4.604257
Is projected: NA
proj4string : [NA]
Number of points: 100
Data attributes:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-19.410  -2.511   4.201   4.681  12.960  29.190

R> #---Since the Data was Generated from a source with same projection as our Kenya data,
R> dsp@proj4string <- kenya@proj4string
```

Now that we have created some points and defined their projection, we are ready to do a point in polygon spatial join. We will use the `over()` command (short for overlay()).

In the over() command we feed it a spatial polygon object (ds), a spatial points object (dsp), and tell it what function we want to use to aggregate the spatial point up. In this case we will use the mean (but we could use any function or write our own). The result will give us a data.frame, and we will then put the resulting aggregated values back into the data.frame() associated with ds (ds@data).

See ?over() for more information.

```
R> #--------------------------POINT IN POLY JOIN------------------------------
R>
R> #--The data frame tells us for each point the index of the polygon it falls into
R> dsdat <- over(kenya, dsp, fn = mean) #do the join
R> head(dsdat) #look at the data

     values
0        NA
1        NA
2        NA
3 -4.940730
4  3.011896
5        NA

R> inds <- row.names(dsdat) #get the row names of dsdat so that we can put the data back i
R> head(inds)

[1] "0" "1" "2" "3" "4" "5"

R> str(kenya@data)

'data.frame':        41 obs. of  3 variables:
 $ ip89DId  : int  1010 2010 2020 2030 2040 2050 3010 3020 3030 3040 ...
 $ ip89DName: Factor w/ 41 levels "Baringo","Bugoma",..: 26 11 13 25 30 31 12 17 19 24 ...
 $ new      : int  1 2 3 4 5 6 7 8 9 10 ...

R> kenya@data[inds, 'pntvals'] <- dsdat #use the row names from dsdata to add the aggregat
R> head(kenya@data)

  ip89DId ip89DName new   pntvals
0    1010   Nairobi   1        NA
1    2010    Kiambu   2        NA
2    2020 Kirinyaga   3        NA
3    2030   Muranga   4 -4.940730
4    2040 Nyandaura   5  3.011896
5    2050     Nyeri   6        NA
```

6 Do a Pixel in Polygon Spatial Join

In this section we will explore another common spatial join operation. In this case you you have raster data that you want to aggregate up to the level of the polygons. A common example is that you have a surface of observed or interpolated temperature measurements and you want to find out what the average (or sum, max, min, et cetera) temperature is for each polygon (which could represent states, counties, et cetera).

```
R> #--------------------------READ AND CROP A RASTER--
R> library(rasta)
R> filepath <- system.file("extdata", "anom.2000.03.tiff", package ="rasta")
R> g <- raster(filepath)
R> # plot
R> plot(g)
R> plot(kenya, add = TRUE) #plot kenay on top to get some sense of the extent

R> #------Crop the Raster Dataset to the Extent of the Kenya Shapefile
R> gc <- crop(g, kenya) #clip the raster to the extent of the shapefile
R> #Then test again to make sure they line up
R> plot(gc)
R> plot(kenya, add = TRUE)
```

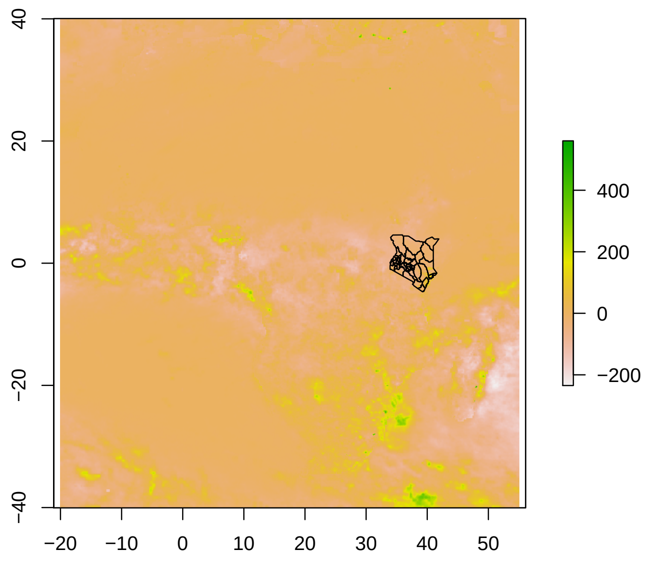

Figure 3: Temperature anomaly for Africa (for March 2003)

In the last step we read in a raster file, cropped it to the extent of the Kenya data (just to cut down on the file size and demonstrate that function). Now we will aggregate the pixel values up the polygon values using the `extract()` function.

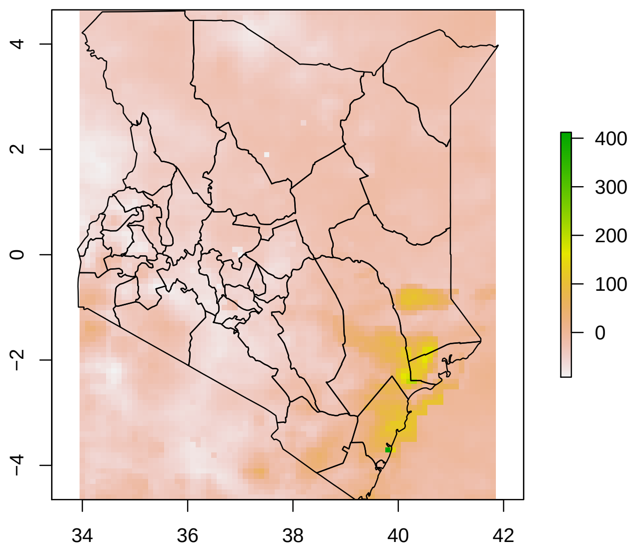

Figure 4: Kenian temperature anomaly for March 2003

```
R> #--------------------------PIXEL IN POLY SPATIAL JOIN------------------------------
R> #Unweighted- only assigns grid to district if centroid is in that district
R> kenya@data$precip <- extract(gc, kenya, fun = mean, weights=FALSE)
```

Weighted (more accurate, but slower) weights aggregation by the amount of the grid cell that falls within the district boundary:

```
R> kenya@data$precip_wght <- extract(gc, kenya, fun = mean, weights = TRUE)
R> #If you want to see the actual values and the weights associated with them do this:
R> rastweight <- extract(gc, kenya, weights = TRUE)
```

Now that we've added all this data to our shapefile, we'll write it out as a new shapefile and then load it in to make some maps in the next exercise.

7 Special thanks and more info

Special acknowledgments go to Frank Davenport (Spatial R class) for excellent R spatial introduction on which this lesson is based.