Essentials of R

Bas Kempen

Content

- R and RStudio
- Setting-up an R session
- Data types
- Data structures
- Selecting subsets
- Functions
- Import/export of data
- Plotting
- Working with spatial data in R (sp and raster packages)

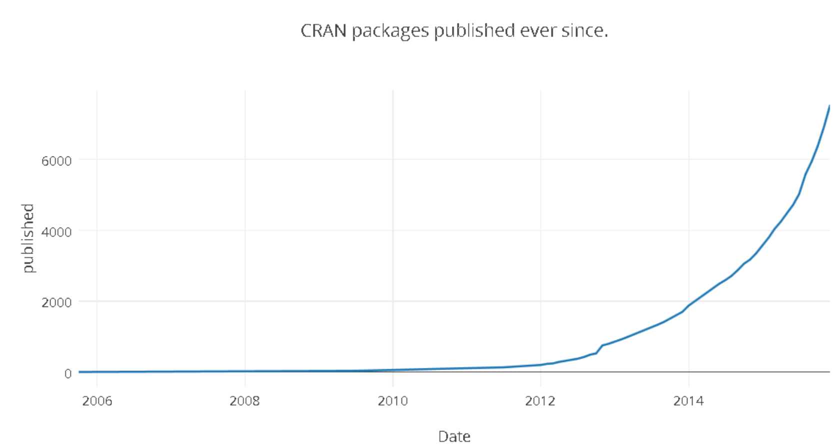
What is R?

- R is a free software environment for statistical computing and graphics

- R provides a wide variety of **data processing and analyses**, **statistical** (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and **graphical** techniques

- R is highly extensible:
  - Base functionality (comes with R)
  - Extension via ‘**packages**’ (>10,000)

- Homepage: www.r-project.org

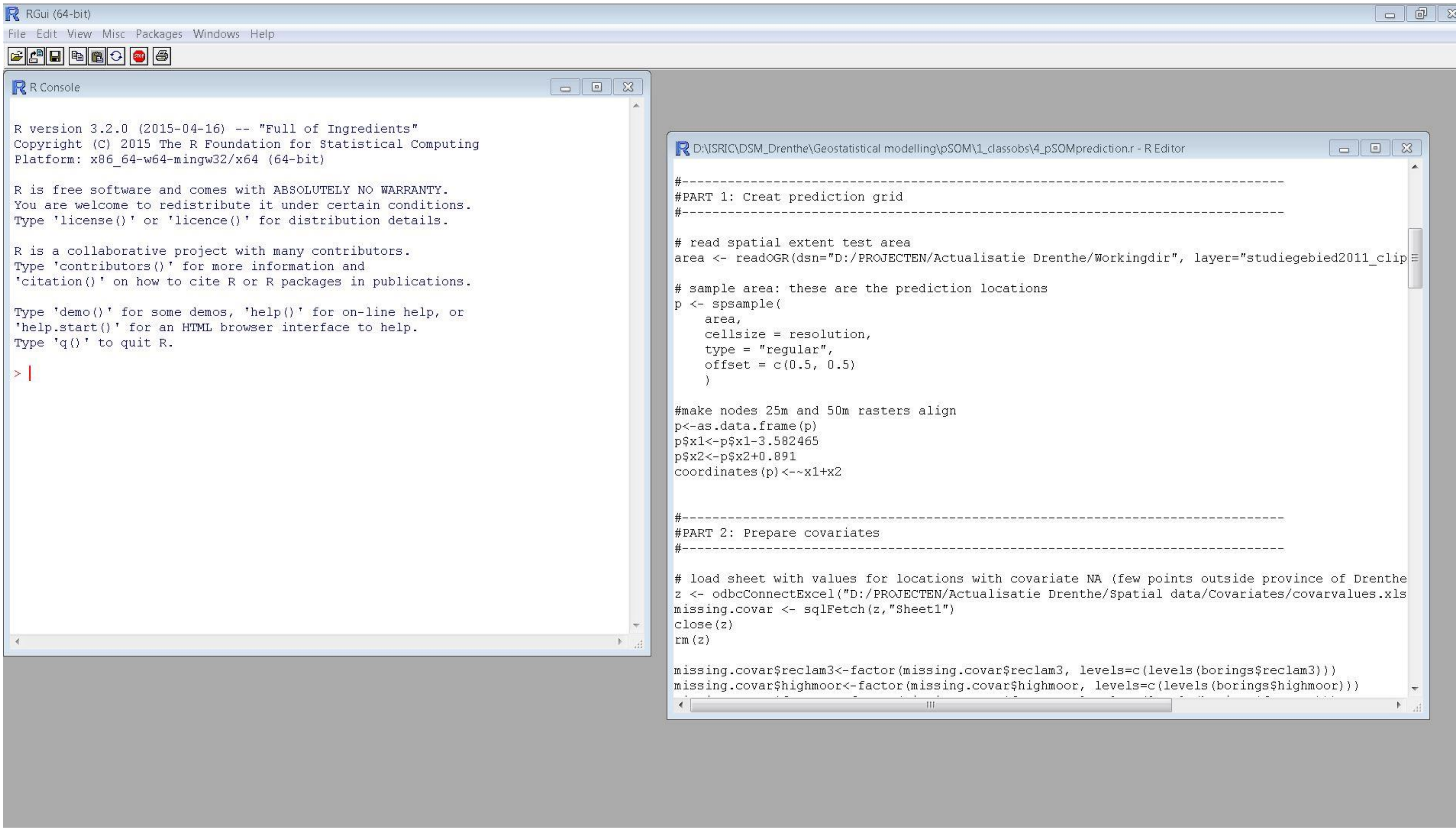


Why R?

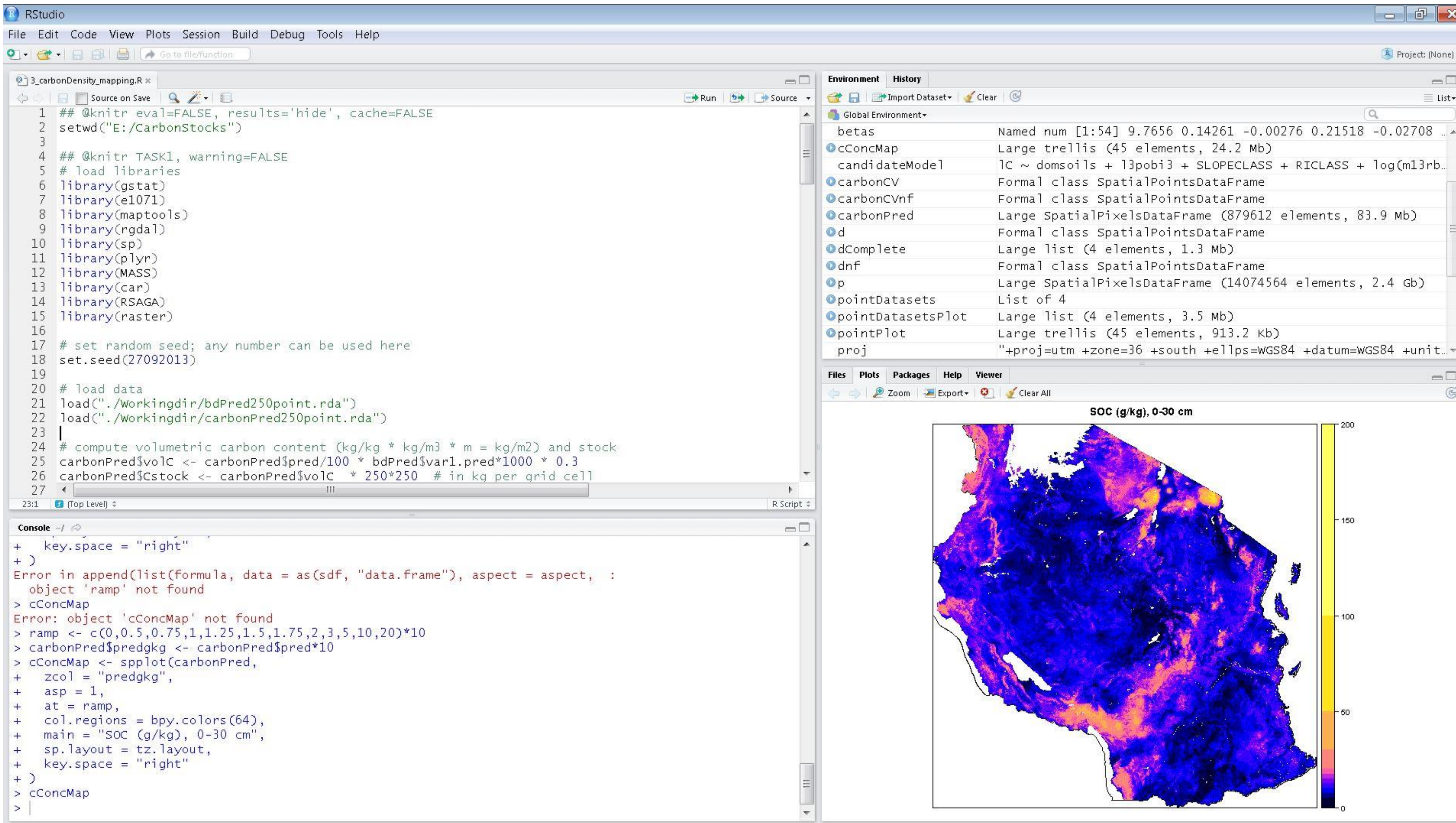
- It is free and it runs on a variety of platforms
- Platform for (advanced) statistical data analyses
- State-of-the-art graphic capabilities
- Very large user community on the web; lots of resources
- Supports **reproducible/transparent** and **collaborative** research (GitHub)
- Connects with other software (SAGA GIS, GE, Python)

- R has a steep learning curve
- Thousands of packages, not always easy to find what you are looking for
- Sometimes cryptic error messages
- Not a GIS

Working with R



RStudio

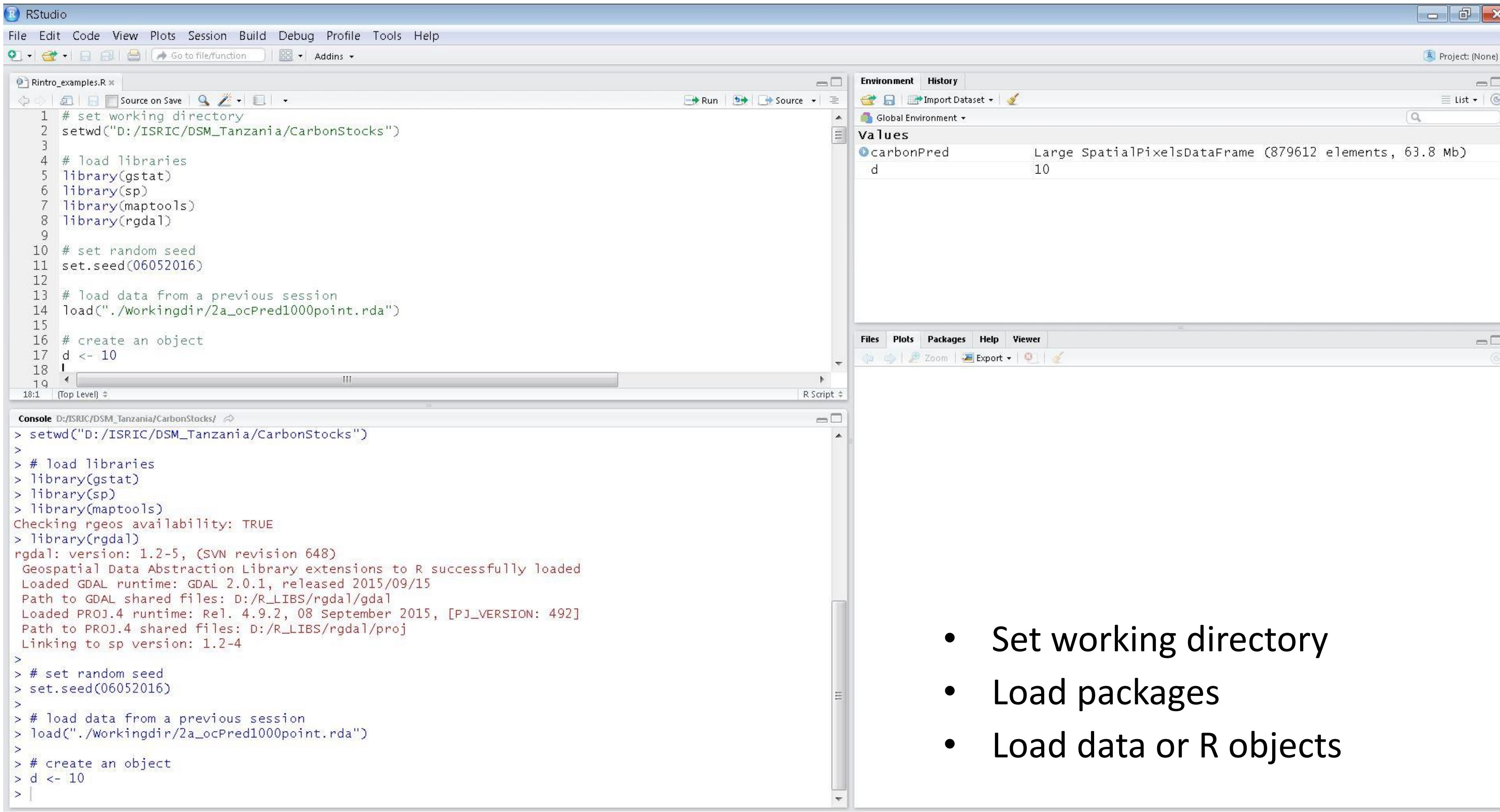

RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Project: (None)
3_carbonDensity_mapping.R
Source on Save
Run
Source
@knitr eval=FALSE, results='hide', cache=FALSE
setwd("E:/CarbonStocks")
@knitr TASK1, warning=FALSE
# load libraries
library(gstat)
library(e1071)
library(maptools)
library(rgdal)
library(sp)
library(plyr)
library(MASS)
library(car)
library(RSAGA)
library(raster)
# set random seed; any number can be used here
set.seed(27092013)
# load data
load("./Workingdir/bdPred250point.rda")
load("./Workingdir/carbonPred250point.rda")
# compute volumetric carbon content (kg/kg * kg/m3 * m = kg/m2) and stock
carbonPred$volC <- carbonPred$pred/100 * bdPred$var1.pred*1000 * 0.3
carbonPred$Cstock <- carbonPred$volC * 250*250 # in kg per grid cell
23:1
(Top Level)
R Script
Console ~/
+ key.space = "right"
+)
Error in append(list(formula, data = as(sdf, "data.frame"), aspect = aspect, :
object 'ramp' not found
> cConcMap
Error: object 'cConcMap' not found
> ramp <- c(0,0.5,0.75,1,1.25,1.5,1.75,2,3,5,10,20)*10
> carbonPred$predgkg <- carbonPred$pred*10
> cConcMap <- spplot(carbonPred,
+ zcol = "predgkg",
+ asp = 1,
+ at = ramp,
+ col.regions = bpy.colors(64),
+ main = "SOC (g/kg), 0-30 cm",
+ sp.layout = tz.layout,
+ key.space = "right"
+)
> cConcMap
>
Environment History
Import Dataset
Clear
List
Global Environment
betas Named num [1:54] 9.7656 0.14261 -0.00276 0.21518 -0.02708 ...
cConcMap Large trellis (45 elements, 24.2 Mb)
candidateModel lC ~ domsoils + l3pobi3 + SLOPECLASS + RICLASS + log(ml3rb...
carbonCV Formal class SpatialPointsDataFrame
carbonCVnf Formal class SpatialPointsDataFrame
carbonPred Large SpatialPixelsDataFrame (879612 elements, 83.9 Mb)
d Formal class SpatialPointsDataFrame
dComplete Large list (4 elements, 1.3 Mb)
dnf Formal class SpatialPointsDataFrame
p Large SpatialPixelsDataFrame (14074564 elements, 2.4 Gb)
pointDatasets List of 4
pointDatasetsPlot Large list (4 elements, 3.5 Mb)
pointPlot Large trellis (45 elements, 913.2 Kb)
proj "+proj=utm +zone=36 +south +ellps=WGS84 +datum=WGS84 +unit...
Files Plots Packages Help Viewer
Zoom
Export
Clear All
SOC (g/kg), 0-30 cm
200
150
100
50
0

First steps.....

- Tell R where to find and save your files: setting the **working directory** using the ‘setwd’ command:
  - setwd(“D:/SpringSchool/Rintro/workingdir”)
  - setwd(“D:\\SpringSchool\\Rintro\\workingdir”)
  - or use the RStudio file browser
- Load packages that are required for your analyses:
  - library(gstat)
  - require(gstat)

  packages should be **installed** on your computer
- Package repository: windows user profile
- Note: R is **case-sensitive**!

Setting up your session



- Set working directory
- Load packages
- Load data or R objects

R scripts and data objects

- R scripts are saved as "<name>.R" files

- R objects in the environment can be saved for future use.

- Saving:
  - entire environment: **save.image** function
  - couple of objects: **save** function

- Output saved as "<name>.rda" or "<name>.RDATA" files

Console D:/ISRIC/DSM_Tanzania/CarbonStocks/

```
> #save data
> save.image("./Workingdir/carbon.rda")
> save(d, file = "./Workingdir/data.rda")
> |
```

Basic data types

- Numeric
- Integer
- Logical
- Character

```
> x <- 10.1
> x
[1] 10.1
> class(x)
[1] "numeric"
```

```
> y <- as.integer(x)
> y
[1] 10
> class(y)
[1] "integer"
```

```
> x <- 1; y<-2
> z <- x > y
> z
[1] FALSE
> class(z)
[1] "logical"
>
```

```
> z <- "Bas"
> z
[1] "Bas"
> class(z)
[1] "character"
```

vector

```
> lc <- c("arable","grassland","forest","arable")
> lc
[1] "arable"    "grassland" "forest"    "arable"
> class(lc)
[1] "character"
```

- Factor: vector that can only contain **pre-defined** values, used to store **categorical** data.

function

```
> lc.f <- as.factor(lc)
> lc.f
[1] arable    grassland forest    arable
Levels: arable forest grassland
> class(lc.f)
[1] "factor"
```

```
> summary(lc)
   Length     Class      Mode
        4 character character
> summary(lc.f)
   arable    forest grassland
        2         1         1
```

Vectors

- Sequence of elements of the **same** basic type (one-dimensional).

```
> c(6, 9, 2016)
[1]    6    9 2016
> c("spring","school")
[1] "spring" "school"
```

- Vectors can be combined.

```
> a <- c(6, 9, 2016)
> b <- c("spring","school")
>
> z <- c(a,b)

> z
[1] "6"      "9"      "2016"   "spring" "school"
> class(z)
[1] "character"
> length(z)
[1] 5
```

Vector arithmetic

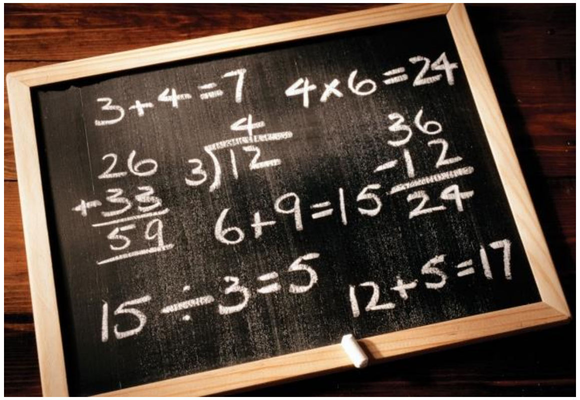
- Vectorized operations: most operations work on vectors with the same syntax as they work on scalars (no need for looping).

- Vector arithmetic:

```
> a <- c(1, 2, 3)
> b <- c(4, 5, 6)
>
> 5 * a
[1]  5 10 15
> a + b
[1] 5 7 9
```

- Recycling of vector elements:

```
> a <- c(1, 2, 3)
> b <- c(4, 5, 6, 7, 8, 9)

> a + b
[1]  5  7  9  8 10 12

> a - mean(a)
[1] -1  0  1
```



Other data structures

- **Matrix**: two-dimensional vector
- **List**: data structure that can hold any number of any types of other data structures
- **Data frame**: table

- Data frame is the **fundamental** data structure for statistical modelling in R.
- Data frame is a **table** with columns and rows (fields and records).

Data frame

- Columns can have different data types (numeric, integer, logical, character, factor)

- All columns must have the same length

```
> 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 ...
```

Selecting subsets

- Selection is done with '**[…]**'
- Vector:

```
> a
[1] 1 2 3
> a[1]
[1] 1
> a[-1]
[1] 2 3
> a[c(1,3)]
[1] 1 3
```

- Data frame:

```
> trees[1,]
  Girth Height Volume
1   8.3     70   10.3
> trees[,3]
 [1] 10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 24.2 21.0 21.4 21.3 19.1 22.2 33.8 27.4 25.7
[20] 24.9 34.5 31.7 36.3 38.3 42.6 55.4 55.7 58.3 51.5 51.0 77.0
> trees[1,3]
[1] 10.3

> trees$Girth
 [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4 11.4 11.7 12.0 12.9 12.9 13.3 13.7 13.8 14.0
[22] 14.2 14.5 16.0 16.3 17.3 17.5 17.9 18.0 18.0 20.6
> trees$Height
 [1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78 80 74 72 77 81 82 80 80 80 87
> trees$Volume
 [1] 10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 24.2 21.0 21.4 21.3 19.1 22.2 33.8 27.4 25.7 24.9 34.5
[22] 31.7 36.3 38.3 42.6 55.4 55.7 58.3 51.5 51.0 77.0
```

Functions

- Data analyses and modelling is done through functions.
- Functions can be very simple:

```
> sqrt(25)
[1] 5
```

- More complex functions have multiple arguments (inputs):

```
# compute sample variogram
v <- variogram(carbon~1, data = d, width = 1)

# define variogram model: initial values for psill, range, nugget based on sample variogram
vm <- vgm(psill = 30, model = "Sph", range = 5000, nugget = 80)

# fit variogram model to the experimental variogram
vmf <- fit.variogram(v, model = vm)
```

- Arguments have specific requirements
- Access help: ?fit.variogram

Importing data

- Importing from tables:
  - **csv**: read.csv()
  - **txt**: read.table()
  - **xlxs**: read.xlsx() [requires package 'xlsx']

- Data is imported as a **data.frame** object

```
d <- read.csv("Soil.csv")
d <- read.table("Soil.txt", sep="\t", header=TRUE)
d <- read.xlsx("Soil.xlsx", sheetName = "All_Sam_Pit")
```

Exporting data

- Variety of exporting formats for tabular data:
  - write.table()
  - write.csv()
  - write.xlsx()

```
write.table(d, file = "soildata.txt", sep = ";")
write.csv(d, file = "soildata.csv")
write.xlsx(d, file="soildata.xlsx",sheetName="Nepal",col.names=TRUE, row.names=FALSE)
```

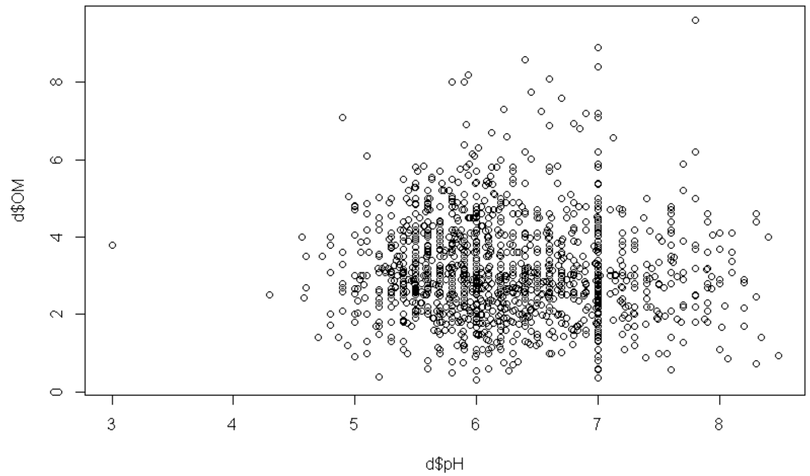
Plotting

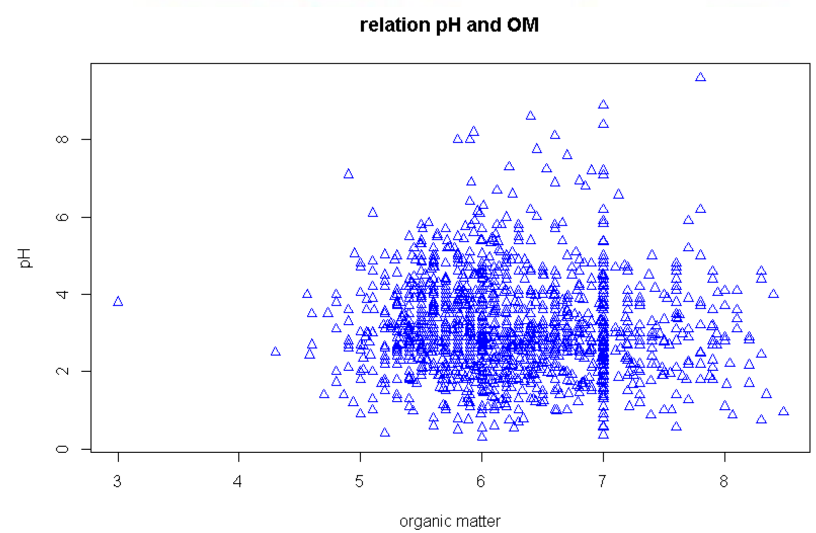
- Large number of packages and functions for generating plots with basic functionality to 'high-level': e.g. lattice and ggplot.

- The basic function for plotting is 'plot'.

```
plot(d$pH,d$OM)
```

```
plot(d$pH,d$OM, xlab="organic matter", ylab = "pH", main="relation pH and OM", pch=2, col="blue")
```



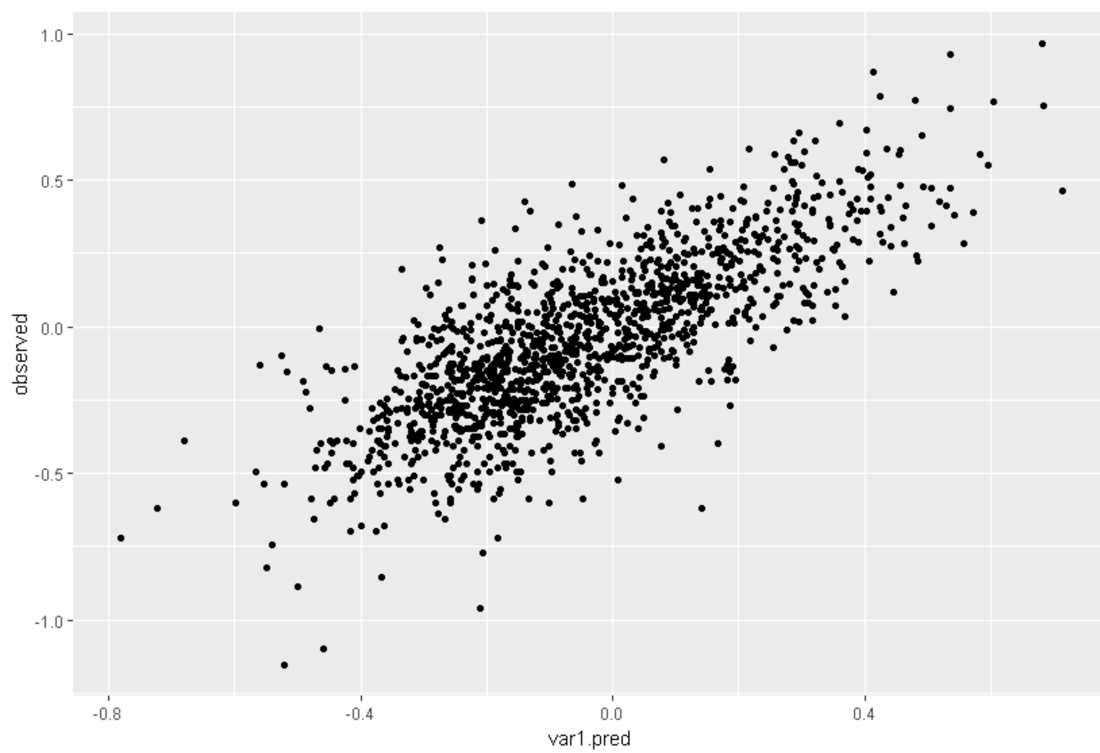


ggplot

- library(ggplot2)
- Build your plot layer by layer
- Building blocks:
  - **geom**: the geometric object that describes the type of plot that is produced.
  - **aes**: ‘aesthetics’, defines how variables in the data are mapped to visual properties.
  - **scales**: control the legend, plot layout.
  - **theme**: controls the appearance of all non-data components.
- http://ggplot2.org/

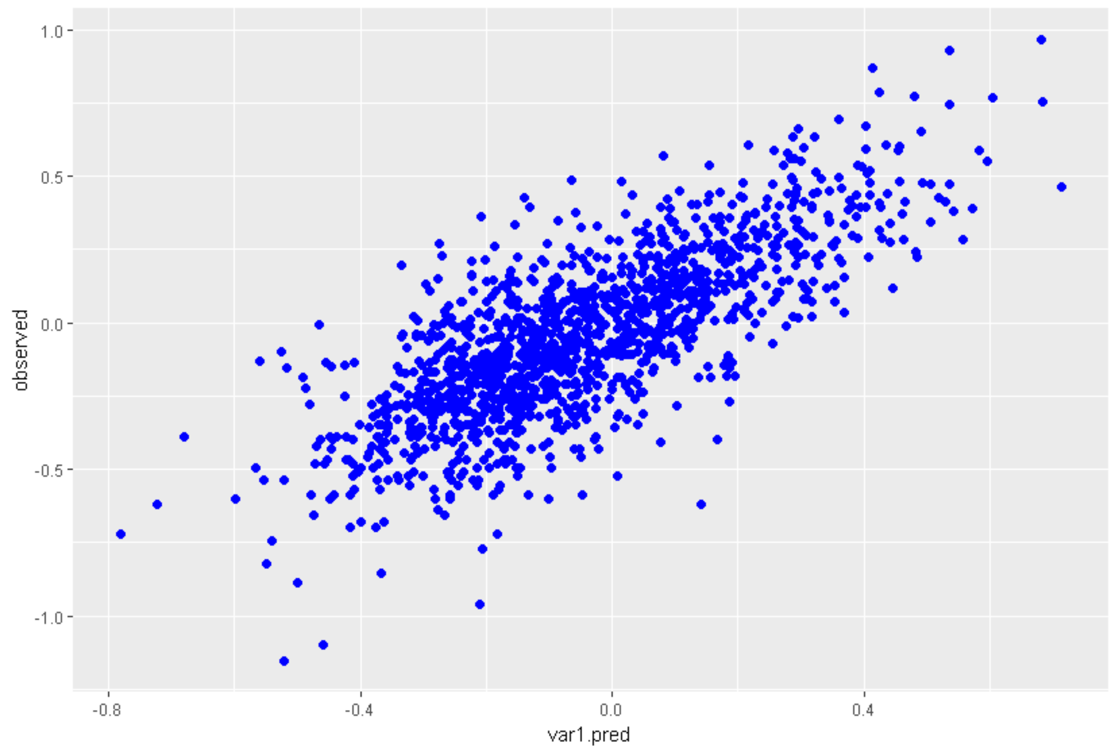
Plotting with ggplot

```
ggplot(data = d) +
  geom_point(mapping = aes(x = var1.pred, y = observed))
```



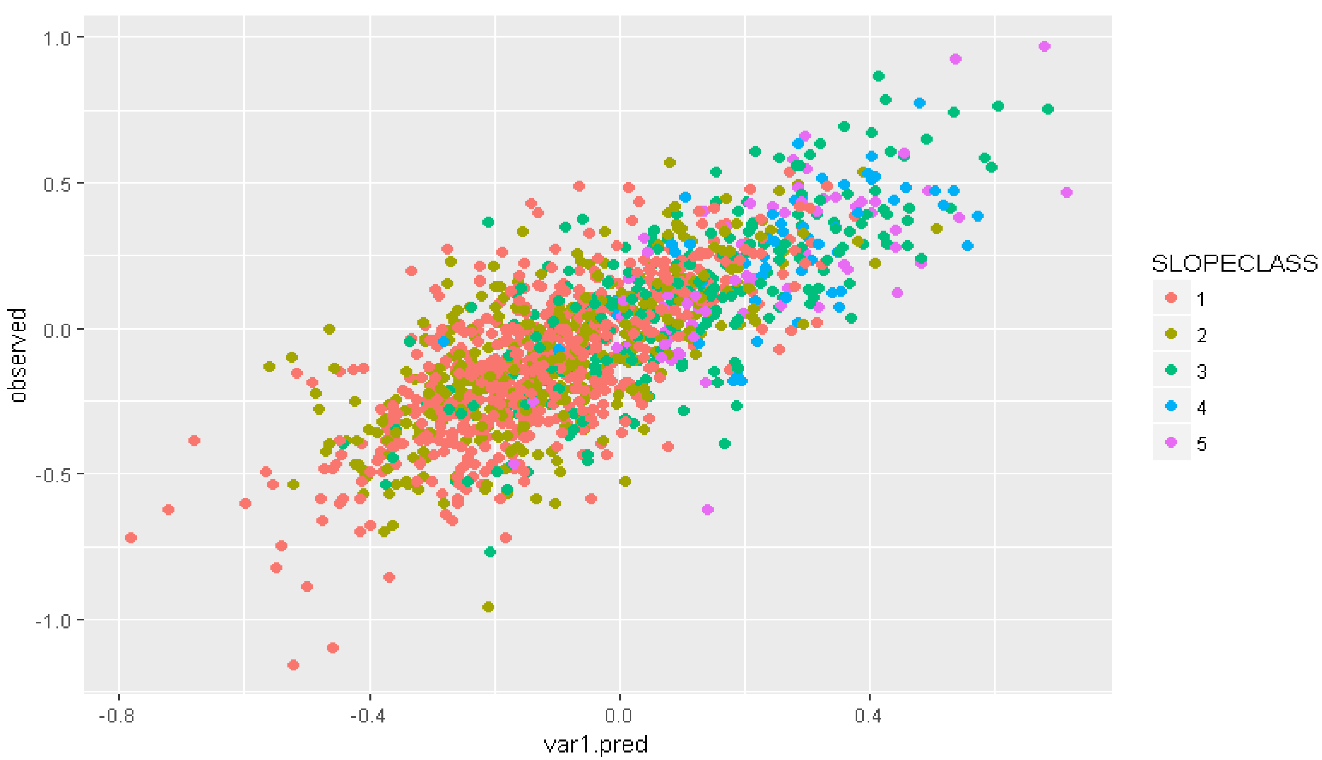
Plotting with ggplot

```
ggplot(data = d) +
  geom_point(mapping = aes(x = var1.pred, y = observed), color="blue", size=2)
```



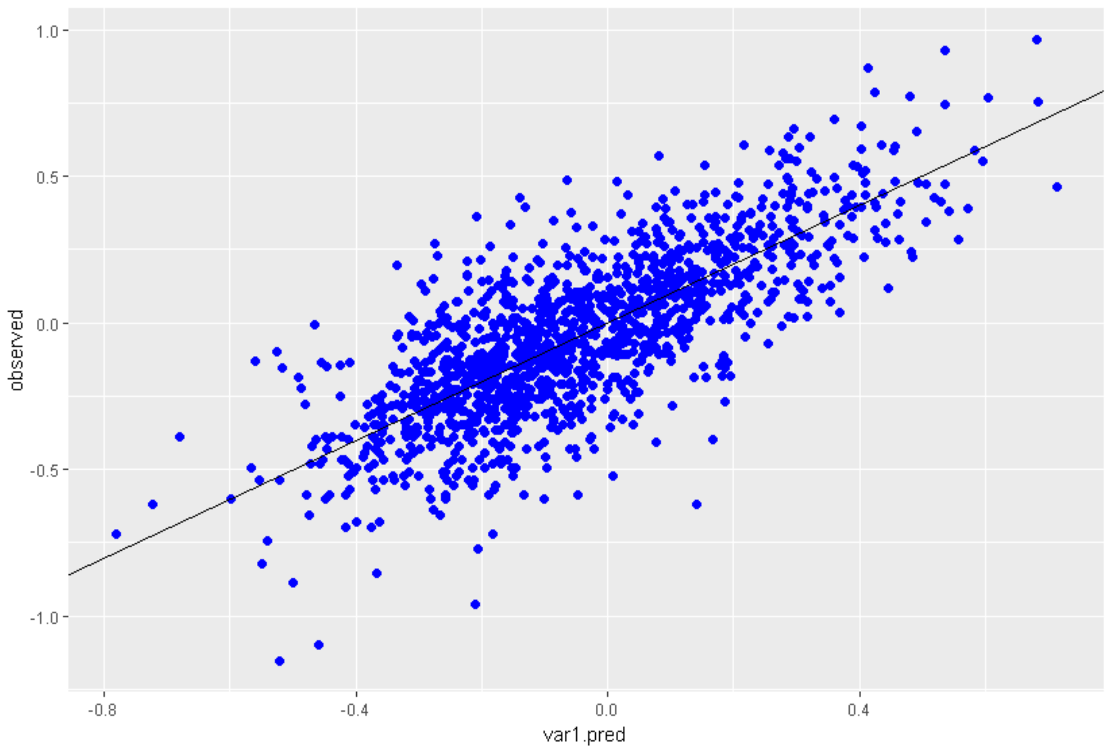
Plotting with ggplot

```
ggplot(data = d) +
  geom_point(mapping = aes(x = var1.pred, y = observed, color=SLOPECLASS), size=2)
```



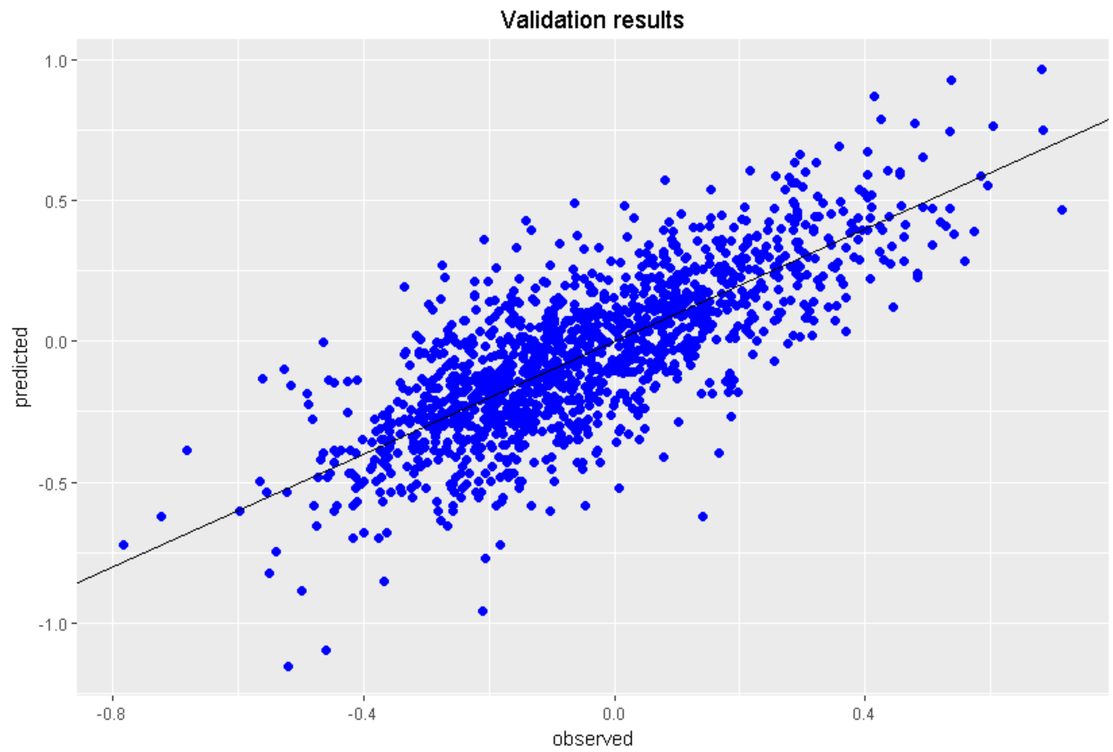
Plotting with ggplot

```
ggplot(data = d) +
  geom_point(mapping = aes(x = var1.pred, y = observed), color="blue", size=2) +
  geom_abline(intercept = 0, slope = 1)
```



Plotting with ggplot

```
ggplot(data = d) +
  geom_point(mapping = aes(x = var1.pred, y = observed), color="blue", size=2) +
  geom_abline(intercept = 0, slope = 1) +
  scale_x_continuous(name = "observed") +
  scale_y_continuous(name = "predicted") +
  ggtitle("Validation results")+
  theme(plot.title = element_text(hjust = 0.5))
```



Exporting plots

- Save plots to png, jpg or pdf

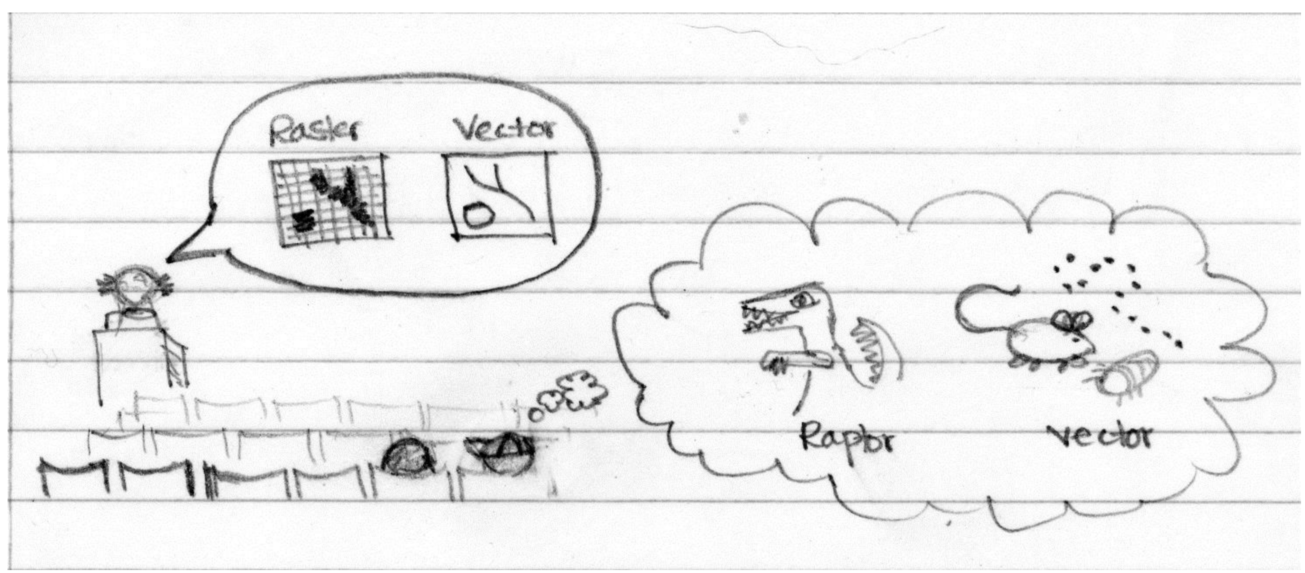
```
png("scatterPlot.png", width=550, height=500)
plot(d$pH,d$OM, xlab="organic matter", ylab = "pH", main="relation pH and OM")
dev.off()
```

```
scatter <- ggplot(data = d) +
            geom_point(mapping = aes(x = var1.pred, y = observed))

png("scatterPlot.png", width=550, height=500)
scatter
dev.off()

jpeg("scatterPlot.jpg", width=550, height=500)
scatter
dev.off()

pdf("scatterPlot.pdf")
scatter
dev.off()
```

Working with spatial data in R

Content

- Spatial data classes (sp, raster)
- Importing/exporting spatial data
- Projections
- Plotting

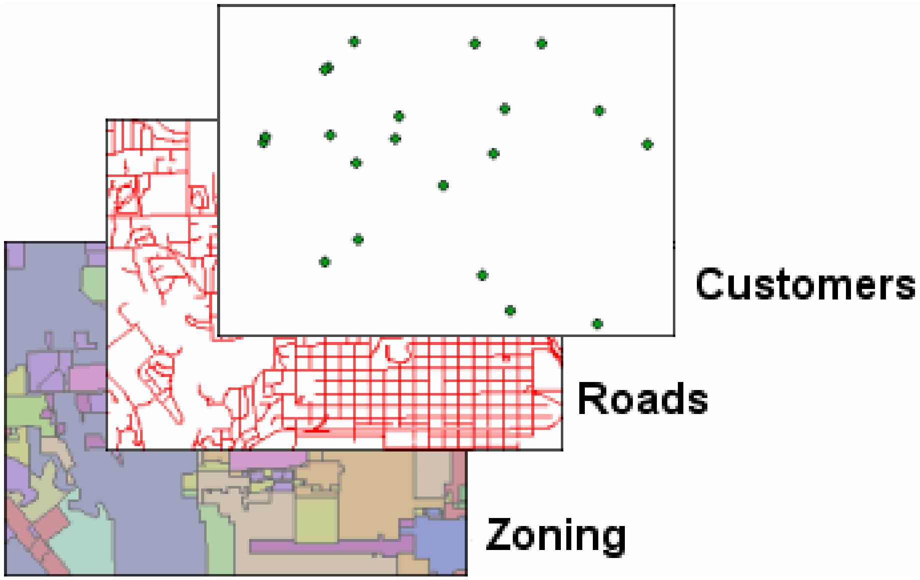


Spatial Data in R

- R offers a wide variety of packages and tools that can handle spatial data.
- Note: R is not a GIS.
- R is not so memory efficient.
- Relevant packages:
  - **sp**: handling spatial data
  - **raster**: reading/manipulating/writing spatial raster data
  - **rgdal**: reading/writing spatial data

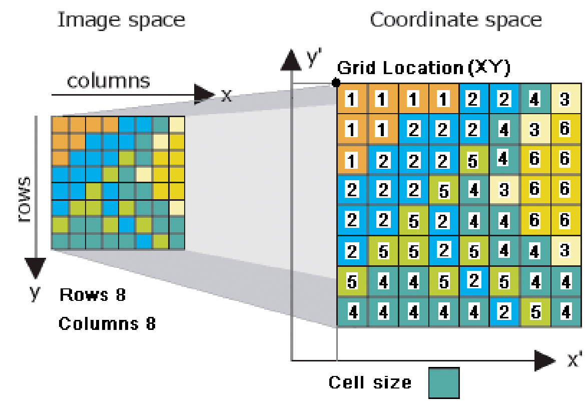
Spatial data classes and formats

- **Vector**: points, lines and polygons (areal).
- For storing data that has **discrete boundaries**, such as country borders, land parcels, and streets.
- Format: **shapefile**



Spatial data classes and formats

- **Raster**: surface divided into a regular grid of cells.
- For storing data that varies **continuously**, as in a satellite image, (surface of chemical concentrations, or an elevation surface).



- Format:
  - **GeoTiff:** allows embedding spatial reference information, metadata and color legends. It also supports internal compression
  - Ascii
  - ESRI Grid

Structures for spatial data

- Spatial data is nothing more than a **data frame** that has columns with **X and Y coordinates** (longitude and latitude).
- Example:

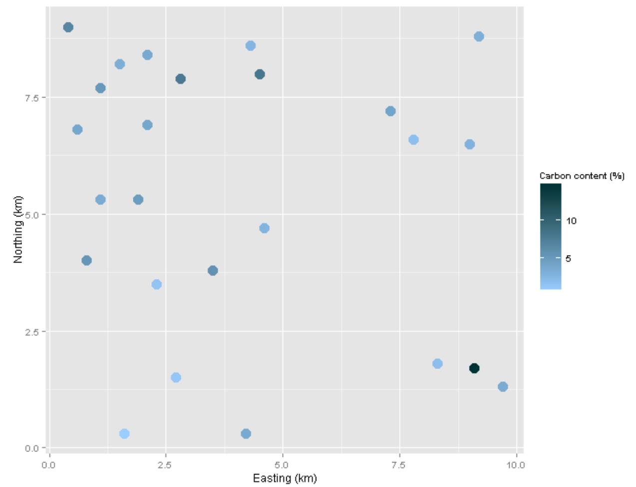
```
> str(d)
'data.frame':	25 obs. of  3 variables:
 $ x     : num  1.9 9.1 1.1 9.7 1.5 0.8 4.6 3.5 9 7.8 ...
 $ y     : num  5.3 1.7 5.3 1.3 8.2 4 4.7 3.8 6.5 6.6 ...
 $ carbon: num  4.8 14.7 3.5 3.5 3.3 5.3 2.8 5.7 2.9 1.8 ...
> head(d)
    x   y carbon
1 1.9 5.3    4.8
2 9.1 1.7   14.7
3 1.1 5.3    3.5
4 9.7 1.3    3.5
5 1.5 8.2    3.3
6 0.8 4.0    5.3
```

- Let’s now take a look at R classes for spatial data: **sp** package

Spatial data classes I

- Convert a data frame to a **SpatialPointsDataFrame** object with the **coordinates** function.

```
> coordinates(d) <- ~x+y
> str(d)
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 25 obs. of  1 variable:
  .. ..$ carbon: num [1:25] 4.8 14.7 3.5 3.5 3.3 5.3 2.8 5.7 2.9 1.8 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:25, 1:2] 1.9 9.1 1.1 9.7 1.5 0.8 4.6 3.5 9 7.8 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:25] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 0.4 0.3 9.7 9
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```

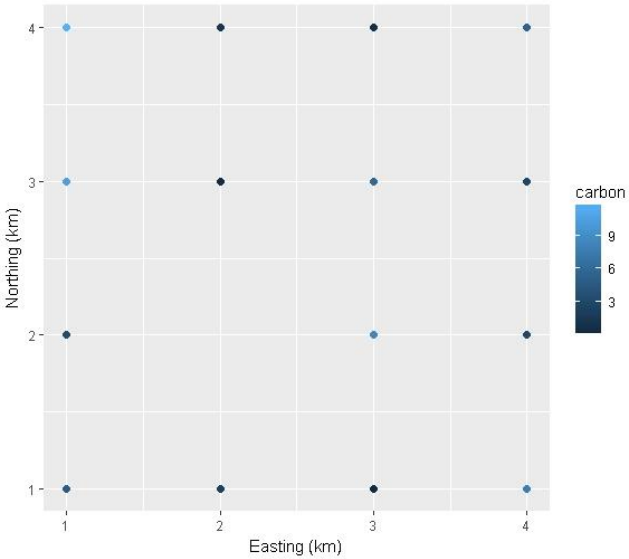


Spatial data classes I

- Data frame with data points at regular intervals.

```
> str(d)
'data.frame':   15 obs. of  3 variables:
 $ x     : num  1 2 3 4 1 3 4 1 2 3 ...
 $ y     : num  1 1 1 1 2 2 2 3 3 3 ...
 $ carbon: num  5 2.4 0.4 7.7 3.4 8.4 3.1 10.4 0.5 5.9 ...
```

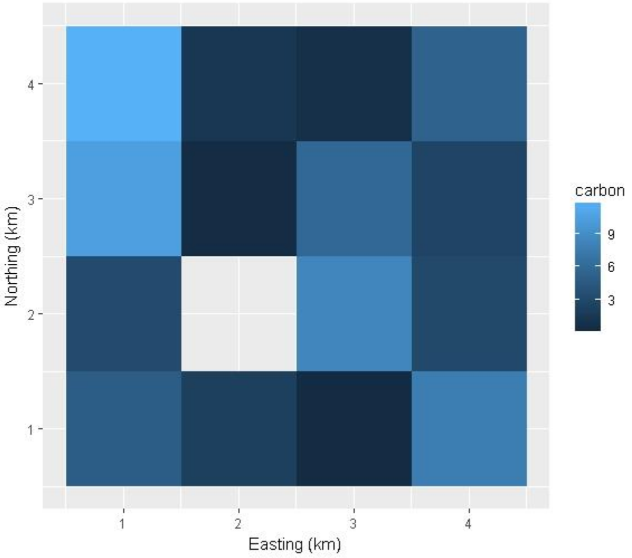
```
> coordinates(d) <- ~x+y
> str(d)
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 15 obs. of  1 variable:
  .. ..$ carbon: num [1:15] 5 2.4 0.4 7.7 3.4 8.4 3.1 10.4 0.5 5.9 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:15, 1:2] 1 2 3 4 1 3 4 1 2 3 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:15] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 1 1 4 4
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```



Spatial data classes II

- Create a grid object with the **gridded** function: **SpatialPixelsDataFrame**

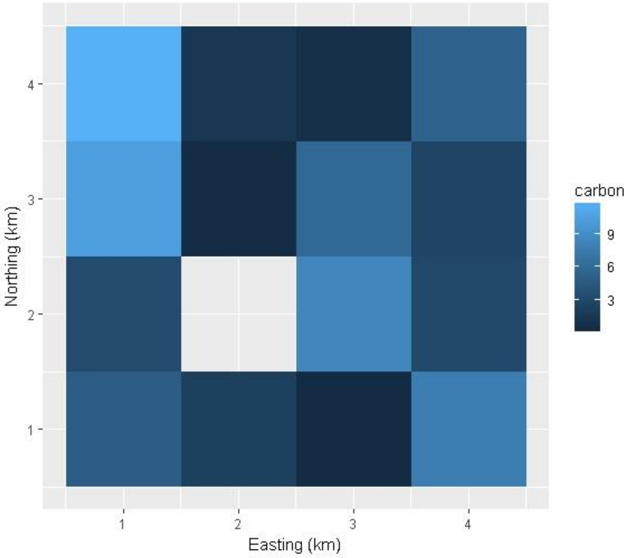
```
> gridded(d) <- TRUE
> str(d)
Formal class 'SpatialPixelsDataFrame' [package "sp"] with 7 slots
  ..@ data       :'data.frame': 15 obs. of  1 variable:
  .. ..$ carbon: num [1:15] 5 2.4 0.4 7.7 3.4 8.4 3.1 10.4 0.5 5.9 ...
  ..@ coords.nrs : num(0)
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] 1 1
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cellsize         : Named num [1:2] 1 1
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cells.dim        : Named int [1:2] 4 4
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  ..@ grid.index : int [1:15] 13 14 15 16 9 11 12 5 6 7 ...
  ..@ coords     : num [1:15, 1:2] 1 2 3 4 1 3 4 1 2 3 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:15] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 0.5 0.5 4.5 4.5
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```



Spatial data classes III

- Create a full grid object with the **fullgrid** function: **SpatialGridDataFrame**

```
> fullgrid(d) <- TRUE
> str(d)
Formal class 'SpatialGridDataFrame' [package "sp"] with 4 slots
  ..@ data       :'data.frame': 16 obs. of  1 variable:
  .. ..$ carbon: num [1:16] 11.8 1.5 0.9 5.5 10.4 0.5 5.9 2.8 3.4 NA ...
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] 1 1
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cellsize         : Named num [1:2] 1 1
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cells.dim        : Named int [1:2] 4 4
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 0.5 0.5 4.5 4.5
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```



Spatial data classes IV

- The **sp** class for polygon data is the **SpatialPolygonsDataFrame**.

```
> str(p,3)
Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 16 obs. of  2 variables:
  .. ..$ Dominant_S: Factor w/ 16 levels "CMe","CMg","CMo",..: 1 2 3 4 5 6 7 8 9 10 ...
  .. ..$ Soil_ag   : Factor w/ 10 levels "CM","FL","GG",..: 1 1 1 1 1 2 3 4 5 6 ...
  ..@ polygons   :List of 16
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  ..@ plotOrder  : int [1:16] 9 1 14 4 5 13 8 11 2 6 ...
  ..@ bbox       : num [1:2, 1:2] -177746 2915230 618131 3379062
  .. ..- attr(*, "dimnames")=List of 2
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

Spatial data classes VI

- The **raster** package comes with its own class for raster data: **RasterLayer**.

```
> str(r,2)
Formal class 'RasterLayer' [package "raster"] with 12 slots
  ..@ file    :Formal class '.RasterFile' [package "raster"] with 13 slots
  ..@ data    :Formal class '.SingleLayerData' [package "raster"] with 13 slots
  ..@ legend  :Formal class '.RasterLegend' [package "raster"] with 5 slots
  ..@ title   : chr(0)
  ..@ extent  :Formal class 'Extent' [package "raster"] with 4 slots
  ..@ rotated : logi FALSE
  ..@ rotation:Formal class '.Rotation' [package "raster"] with 2 slots
  ..@ ncols   : int 2360
  ..@ nrows   : int 1611
  ..@ crs     :Formal class 'CRS' [package "sp"] with 1 slot
  ..@ history : list()
  ..@ z       : list()
```

- Multiple layers: **RasterStack / RasterBrick**

Spatial data class summary

- Spatial data classes for packages:
  - **sp**:
    - SpatialPointsDataFrame
    - SpatialPixelDataFrame
    - SpatialGridDataFrame
    - SpatialPolygonDataFrame
    - SpatialLinesDataFrame
  - **raster**:
    - RasterLayer (single layer)
    - RasterStack / RasterBrick (multiple layers)

Importing spatial data

- rgdal: **readOGR** (vector), **readGDAL** (raster)

- raster: **raster**

```
setwd("D:/ISRIC/DSM_Nepal/Data/Covariates/")

# polgyon (rgdal package)
p <- readOGR(dsn="./shape", layer = "soter_domsoil")

# raster (rgdal package)
r <- readGDAL(fname="./raster/ORCDRC_M_sl3_1km_ll.tif")

# raster (raster package)
r <- raster("./raster/ORCDRC_M_sl3_1km_ll.tif")
```

Exporting spatial data

- rgdal: **writeOGR** (vector), **writeGDAL** (raster)

- raster: **writeRaster**

```
# rgdal
writeOGR(d, dsn="D:/", layer="sampleSites", driver="ESRI Shapefile")
writeGDAL(r["SOC"], "./SOC_Nepal.tif", drivername = "Gtiff", type = "Int16",mvFlag = "-99999")
```

```
# raster
writeRaster(r, "SOC_Nepal2.tif", format="GTiff")
```

Projections

- Once you have loaded your spatial data in R, you might need to tell R its geographic projection.
- Check the current projection: **proj4string** function (sp package).
- Setting a projection: **CRS** function (sp package).
- Reprojecting to another coordinate system: **spTransform** function (sp package) or **projectRaster** (raster package).

Projections

```
> utm.proj <- "+proj=utm +zone=45 +north +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
> wgs84.proj <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
>
> d <- read.csv("./Sample/Processed/Soil.csv")
> coordinates(d) <- ~x+y
> proj4string(d)
[1] NA
> proj4string(d) <- CRS(utm.proj)
> proj4string(d)
[1] "+proj=utm +zone=45 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +towgs84=0,0,0"
> head(d@coords)
        x       y
1 80393.8 3138118
2 49817.4 3140610
3 82986.8 3127692
4 67575.2 3146391
5 63857.7 3127477
6 90005.3 3149250
> d <- spTransform(d, CRS(wgs84.proj))
> proj4string(d)
[1] "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0"
> head(d@coords)
         x        y
1 82.72255 28.30223
2 82.41068 28.31454
3 82.75265 28.20920
4 82.58911 28.37253
5 82.55837 28.20109
6 82.81638 28.40548
```

Projections

```
> utm.proj <- "+proj=utm +zone=45 +north +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
> wgs84.proj <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
>
> wgs84.epsg <- "+init=epsg:4362"
> utm.epsg <- "+init=epsg:32645"
>
> d <- read.csv("./Sample/Processed/Soil.csv")
> coordinates(d) <- ~x+y
> proj4string(d) <- CRS(utm.epsg)
> proj4string(d)
[1] "+init=epsg:32645 +proj=utm +zone=45 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
>
> d <- read.csv("./Sample/Processed/Soil.csv")
> coordinates(d) <- ~x+y
> proj4string(d) <- CRS(utm.proj)
> proj4string(d)
[1] "+proj=utm +zone=45 +ellps=WGS84 +datum=WGS84 +units=m +no_defs +towgs84=0,0,0"
>
```

Find projection definitions at:

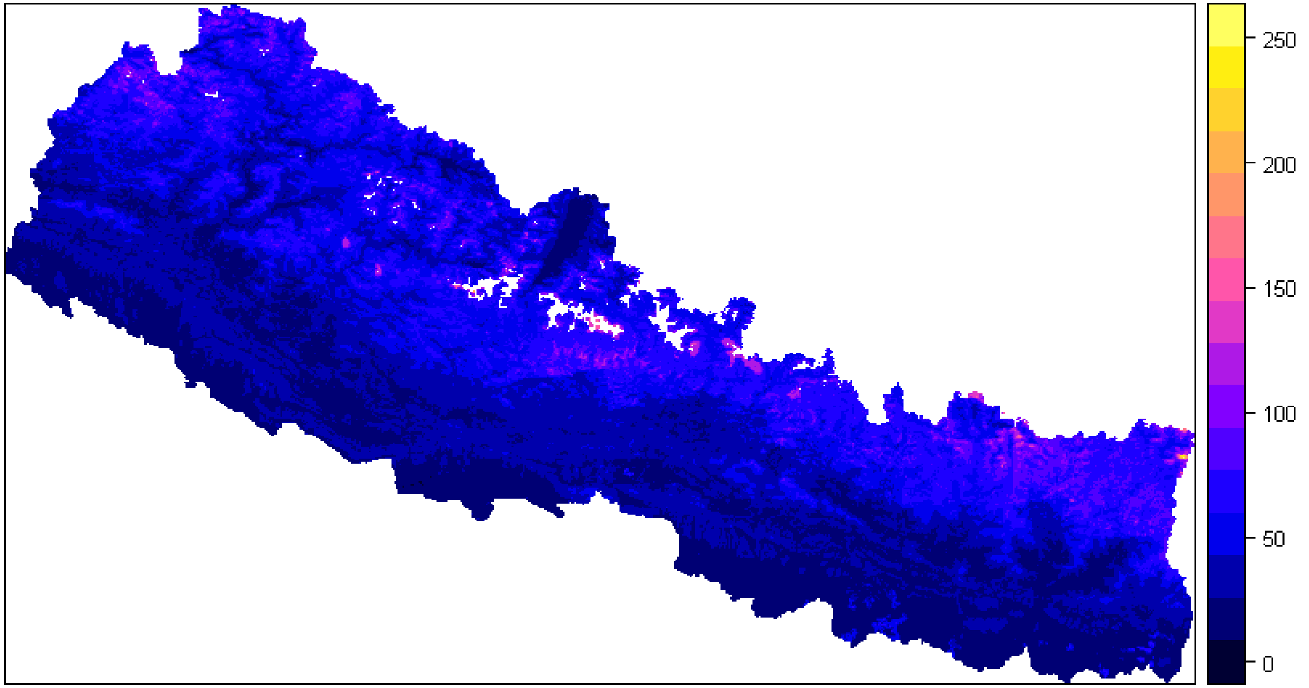
- http://spatialreference.org/
- https://epsg.io/

e.g. UTM Zone 45 N

Plotting

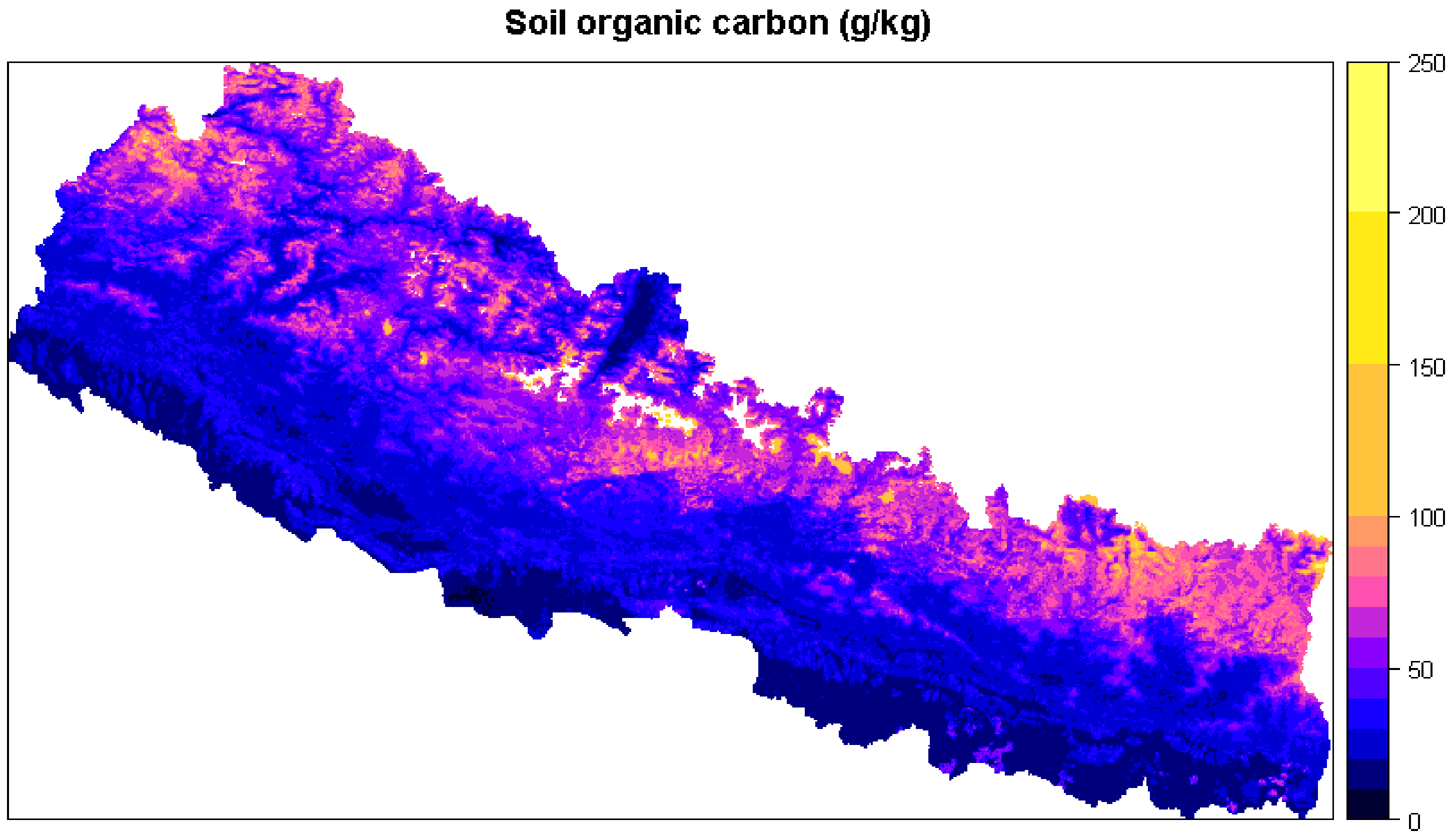
- sp package: **spplot**

```
spplot(r, zcol="SOC")
```



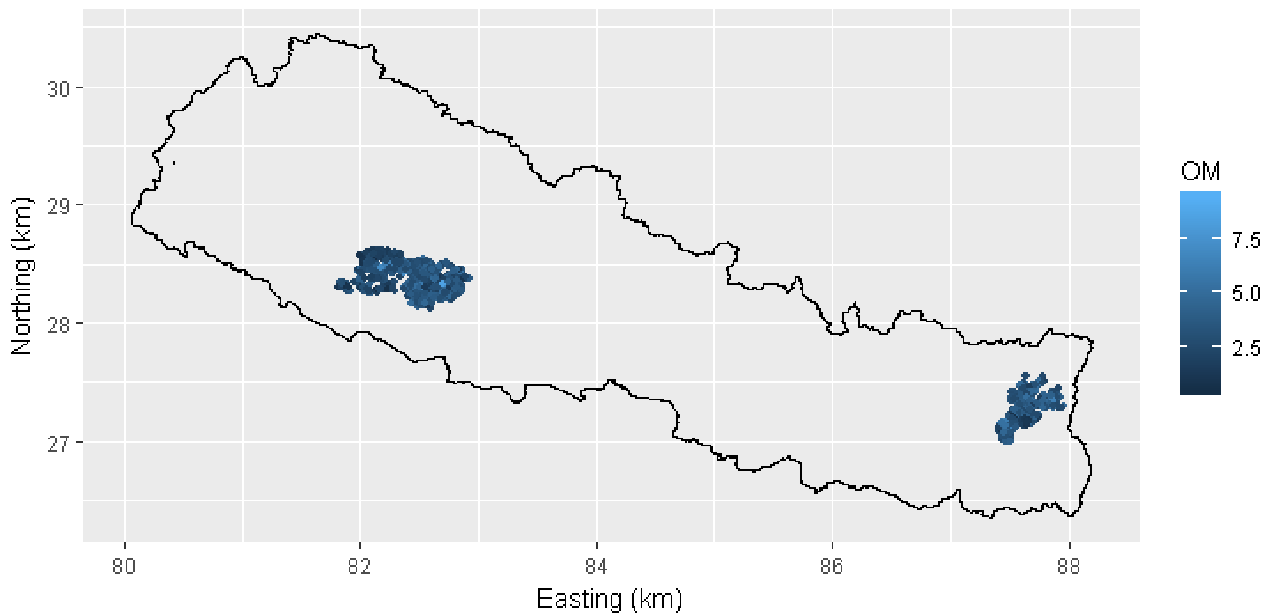
Plotting

```
ramp<-c(0,10,20,30,40,50,60,70,80,90,100,150,200,250)
spplot(r, zcol = "SOC", at = ramp, main = "Soil organic carbon (g/kg)")
```



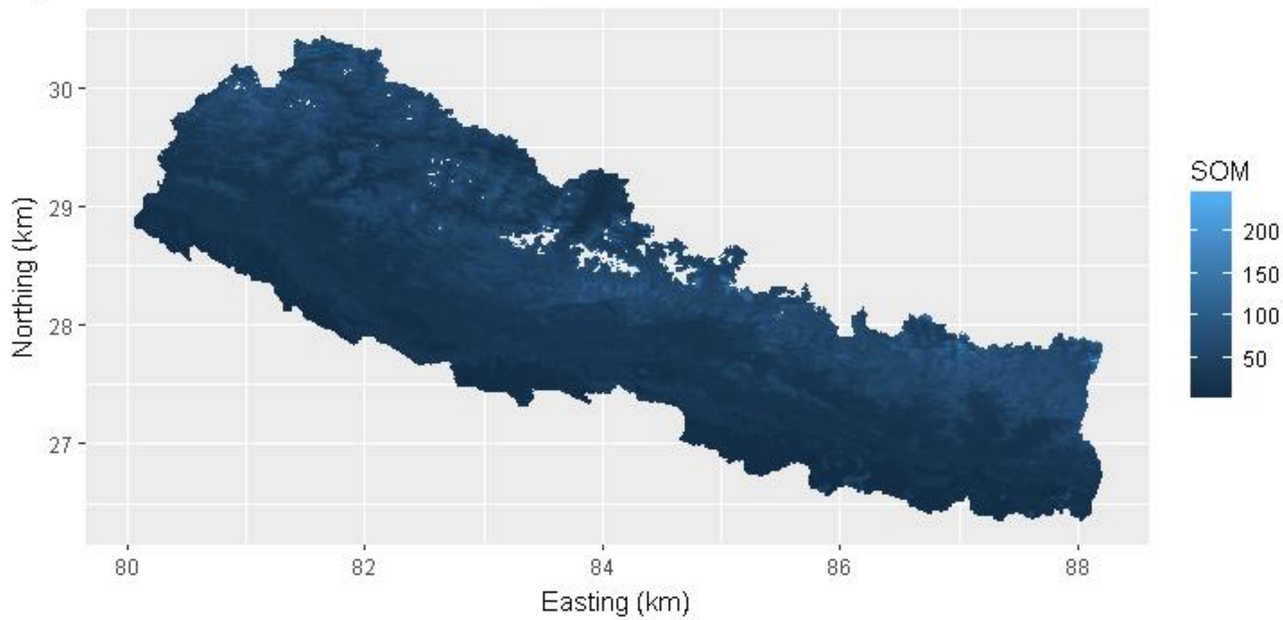
ggplot I

```
ggplot()+
  geom_point(
    data = d,
    mapping = aes(x = x, y = y, colour = OM), size = 1
  )+
  geom_path(
    data = boundary,
    mapping = aes(x = long, y = lat, group = group)
  )+
  scale_x_continuous(name = "Easting (km)") +
  scale_y_continuous(name = "Northing (km)") +
  coord_equal(ratio = 1)
```



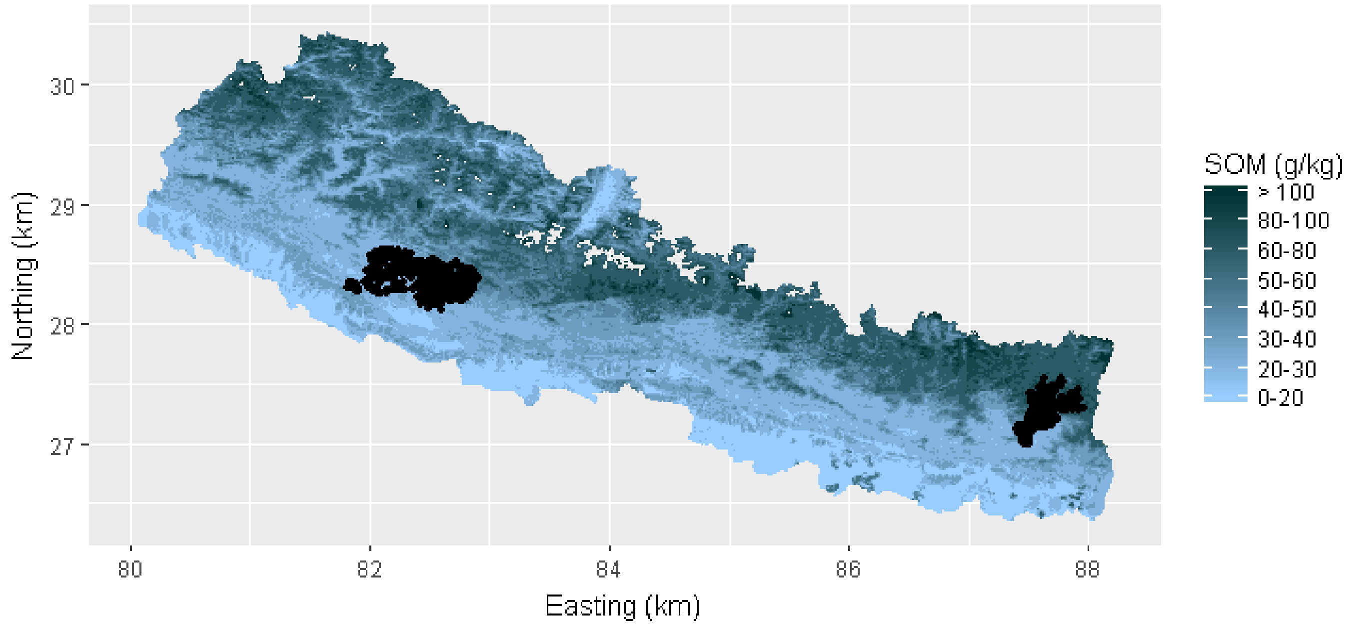
ggplot II

```
ggplot()+
  geom_point(
    data = d,
    mapping = aes(x = x, y = y), size = 1
  )+
  geom_raster(
    data = r,
    mapping = aes(x = x, y = y, fill = SOM)
  )+
  scale_x_continuous(name = "Easting (km)") +
  scale_y_continuous(name = "Northing (km)") +
  coord_equal(ratio = 1)
```



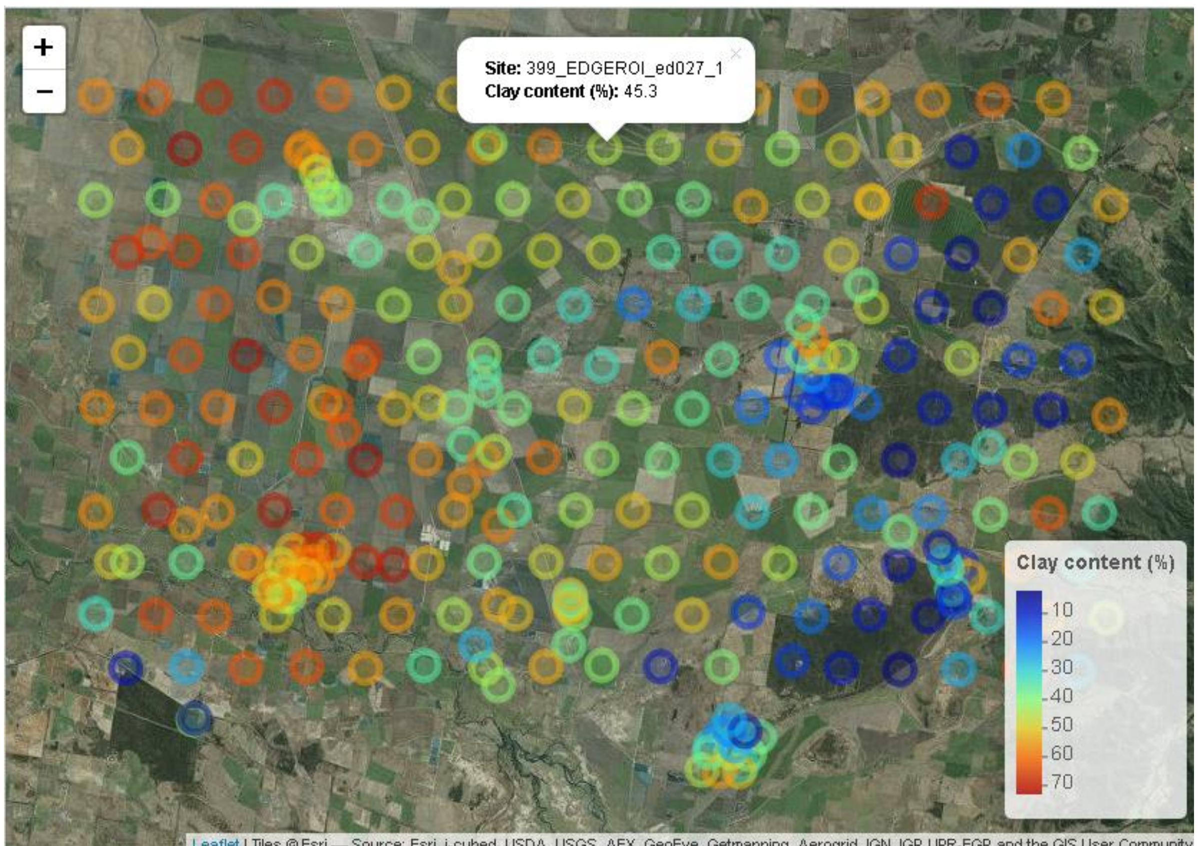
ggplot III

```
lab <- c("0-20","20-30","30-40","40-50","50-60","60-80","80-100","> 100")

ggplot()+
  geom_raster(
    data = r,
    mapping = aes(x = x, y = y, fill = klasse)
  )+
  geom_point(
    data = d,
    mapping = aes(x = x, y = y), size = 1
  )+
  scale_fill_continuous(name = "SOM (g/kg)",low = "#99CCFF",high = "#003333",breaks = c(1:8),labels = lab)+
  scale_x_continuous(name = "Easting (km)")+
  scale_y_continuous(name = "Northing (km)")+
  coord_equal(ratio = 1)
```



Interactive maps

- Interactive maps can be generated with the **leaflet** package.



Resources

- R tutor
- Rwiki
- Advanced R
- R for Data Science
- Making maps in R
- ggplot cheat sheet

Problem solving:

- StackOverflow
- R-FAQ

Now lets practice. Have fun!!