
Food and Agriculture Organization of the United Nations
AFACI
Asian Food & Agriculture Cooperation Initiative
Rural Development Administration
EduSoils
R Basics
Essentials for Getting Started
GLOBAL SOIL PARTNERSHIP

Table of contents:

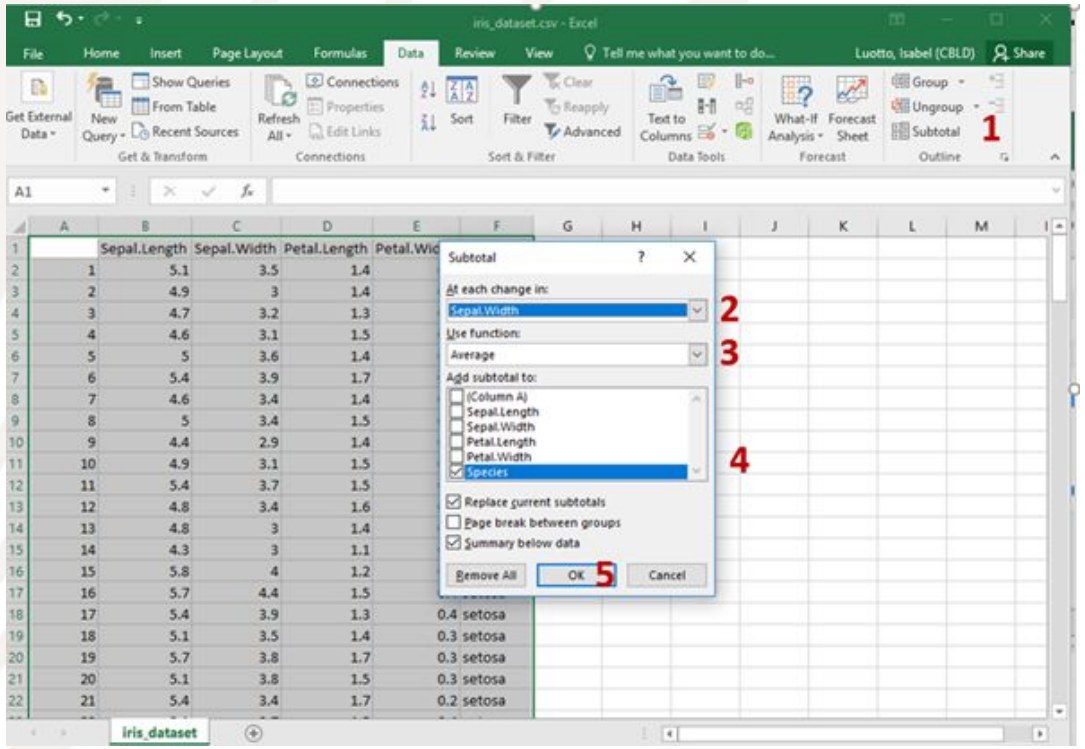
What is R?

- R is a high level programming language and environment for statistical computing
- It provides a wide variety of statistical (e.g. linear modeling, statistical tests, time-series, classification, clustering, etc.) and graphical methods, and is highly extensible
- It's Open Source, meaning that it's not only free but also based on the contribution of Users → modular package structure
- Add-ons (in R lingo "Packages") extend the applicability of R to many fields, like in our case for geostatistics
- e.g. `install.packages('raster')`

Why R?

- Transparency and reproducibility everything you do in the analysis, from deleting outliers to interpreting results, is contained in your code
- It is really hard to properly document the thought process behind a spreadsheet, and values (not calculations) can be changed with no record of their change
- Powerful data manipulation capabilities. It can handle large datasets
- State-of-the-art graphics
- Click-oriented programs require more steps and can be very time consuming when trying to complete slightly demanding tasks

Why R? E.g.Taking the average of something per group



In excel it would take minimum 5 steps

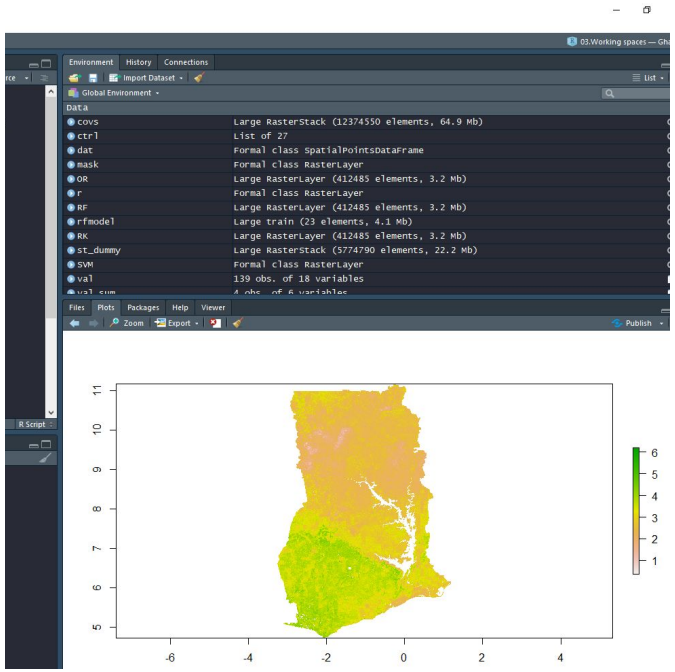
While in R only one line of code is required

VS.

```
swm <- aggregate(Sepal.Width ~ Species, iris, mean)
```

Why R?

- Packages are basically a collection of functions for a given topic, which allow you to perform specific tasks
- CRAN, the global repository of open-source packages that extend the capabilities of R have more than 10,000 R packages available for download
- This has made R an essential tool for geostatistics and many other fields
- Lot's of free learning material!



EduSoils | e-learning soil educational platform

GLOBAL SOIL PARTNERSHIP

R vs SAS

SAS	Features	R
Expensive	$	Open Source
Only with each new version	Extendability	>10000 Packages (Add-ons)
Fast	Learning curve	Slow
Limited	Graphics	Advanced

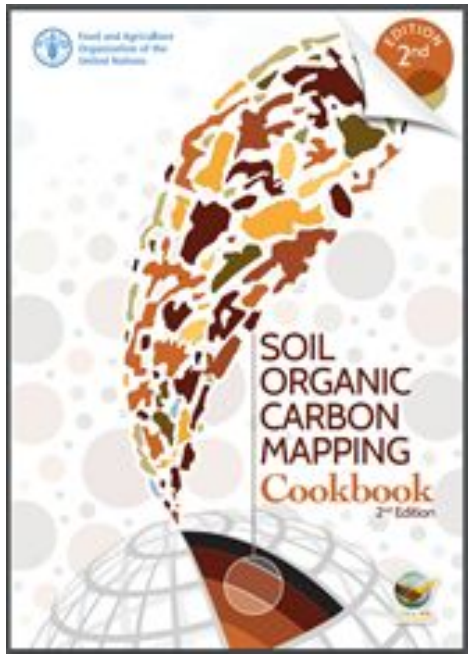
Additional learning material

Soil Organic Carbon Cookbook
http://www.fao.org/3/I8895EN/i8895en.pdf

Introduction to the R Project for Statistical Computing for use at ITC by D G Rossiter
https://cran.r-project.org/doc/contrib/Rossiter-RIntro-ITC.pdf

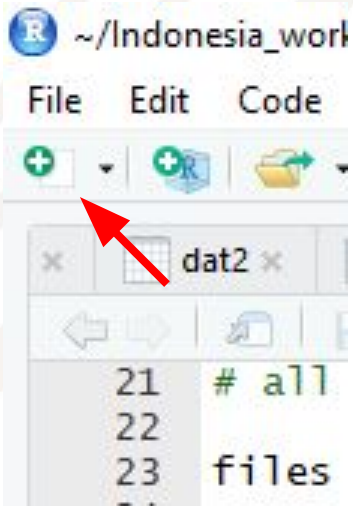
Youtube channel: MarinStatsLectures- R Programming & Statistics
https://www.youtube.com/channel/UCaNIxVagLhqupvUiDK01Mgg

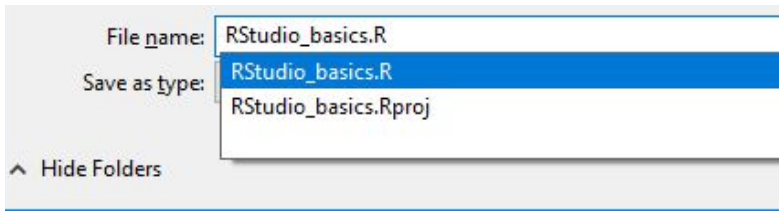
Paid course on Udemy by R-Tutorials Training
https://www.udemy.com/r-level1/



Let’s get our hands dirty!

- Open a new script page and save it
- Scripts are essentially a collection of code
- It is good practice to name and organize your scripts based on what you’re trying to do
- This will be useful for your future R sessions

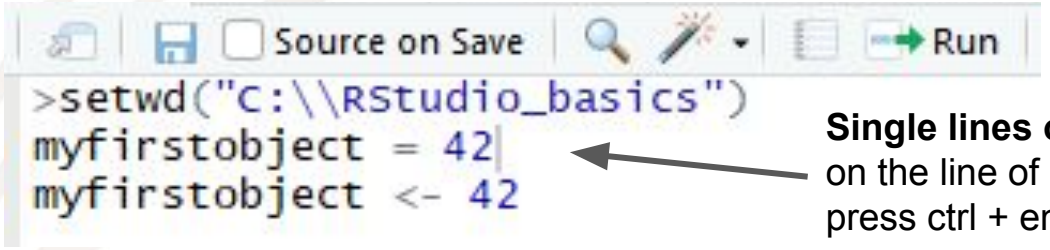




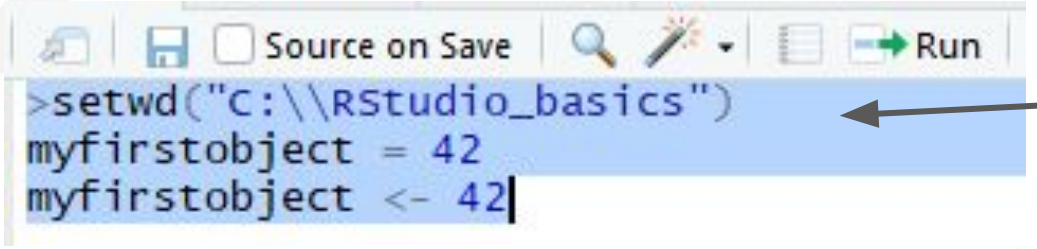
Let’s get our hands dirty!

```
#My first object
myfirstobject = 42
myfirstobject <- 42
```

Throughout the course you will be asked to copy lines of code.
The lines of code to be copied in your script will appear with the following font and color coding.
Look out for this font and try to not copy text that does not correspond to this.



Single lines of code: Keep the cursor on the line of code you want to run and press ctrl + enter, or the button Run



Multiple lines of code: Select the line of codes you want to run with your curso and press ctrl + enter, or the button Run

EduSoils | e-learning soil educational platform

Let’s get our hands dirty!

- First we set the working directory: copy and paste the location of the folder RStudio_basics into the code
- Change the single \ to a double one \\ or to /

```
setwd("C:/Users/hp/Documents/FAO/EduSoils/AFACI_training/Training_material")
```

- Next we're going to create our first object!
- You can also use "<-" to assign an object

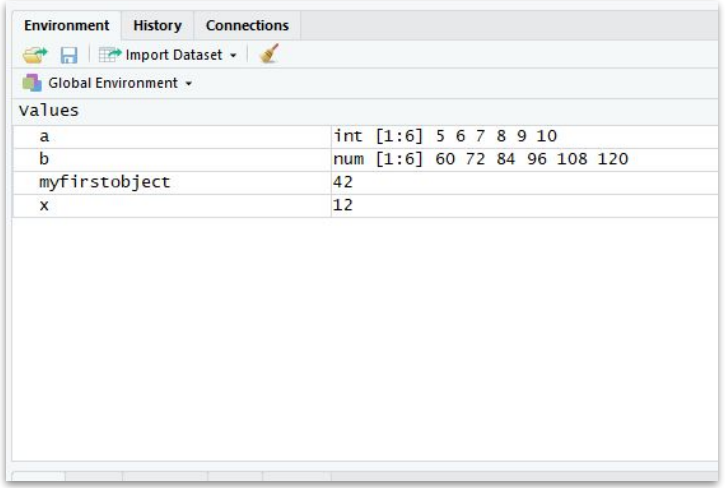
```
myfirstobject = 42

myfirstobject <- 42
```

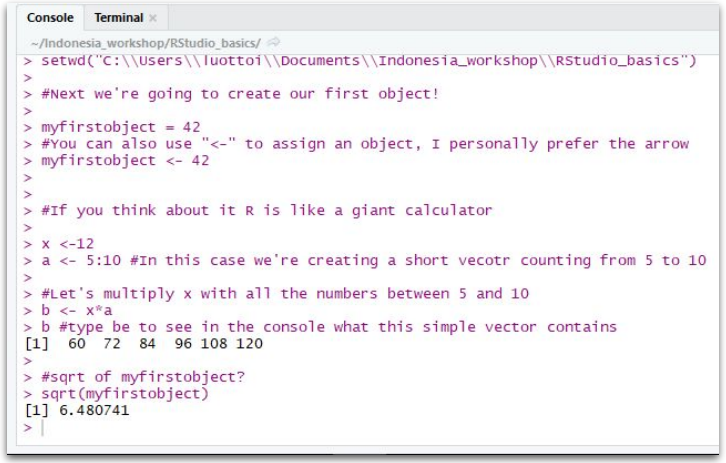
Copy and paste Code

R as a giant calculator

- # are for #comments in your script
- Operators: + - / *

```
a <- 5:10 #In this case we're creating a short #vector
counting from 5 to 10

#Let's multiply x with all the numbers between 5 #and 10

b <- x*a

b #type b to see in the console what this #simple vector
contains

#sqrt of myfirstobject?

sqrt(myfirstobject)
```



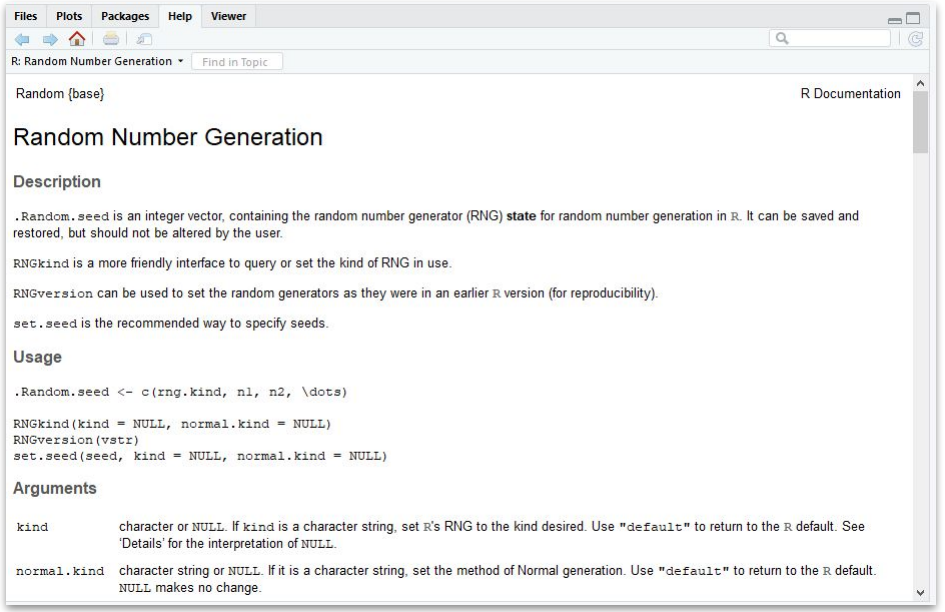


Question 1:
Square root of 142?

R Functions

- R is a higher programming language meaning that the functions (e.q. sqrt(), hist() and maaaany more) were already written for you
- Let's take a look at some functions and at how they can be customized

```
#Set seed is a function to make a

#random sample reproducible

?set.seed #we can get some info in

#the help pane by using "?"

set.seed(42)
```



R Functions

- To see how the function is structured and how you can customize the tool "?" is essential
- In this example we can see what each number within the brackets and commas of the function rnorm does

```
?rnorm() #if you leave something between the commas blank it goes #to default
```

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

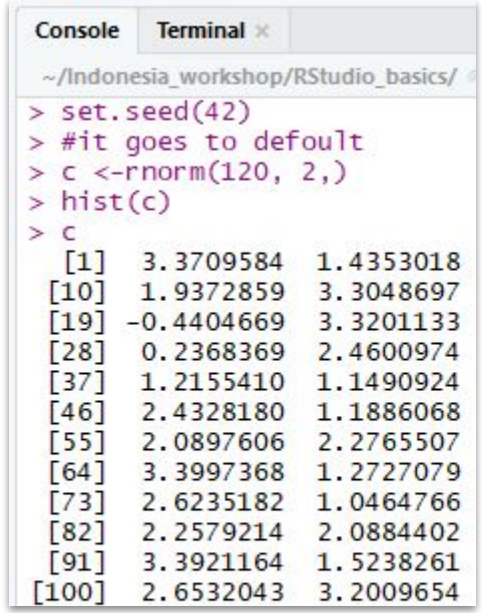
Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

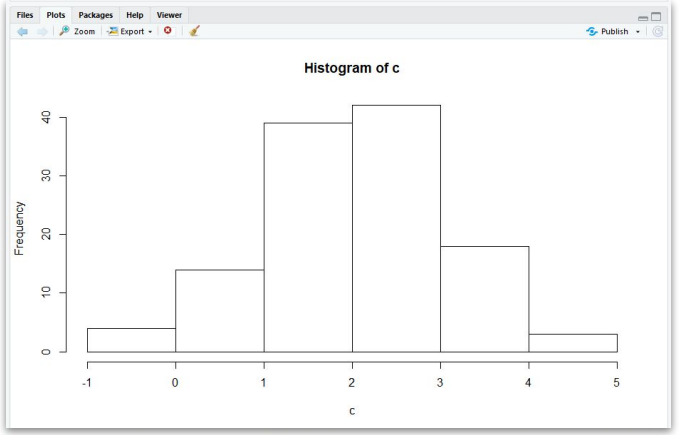
R Functions

- To see how the function is structured and how you can customize the tool "?" is essential
- In this example we can see what each number within the brackets and commas of the function rnorm does

```
?rnorm #if you leave something between the commas blank
#it goes to default
b<-rnorm(120, 2,)
hist(b)
b
```



Question 2: What is the first number appearing when running b?



Three common `Error` sources in R

1. Installing and loading packages
2. Case sensitivity
3. Concatenation

Three common `Error` sources in R

1. Installing and loading packages
    - R base relies on user contributions (packages) for many of its functions
    - When starting a new session of R the packages need to be loaded each time
    - Add-on packages may be required when running a script make sure to have them installed

```
#If you see this in your script
library("raster") #or
require("raster")
#And you don't have the installed than you should run this line
install.packages('raster')
```

Three common Error sources in R

1. Installing and loading packages
    - During the workshop we will need several packages please install them now:

```
# Install all required R packages used in the training
install.packages(c("aqp", "automap", "car", "caret", "e1071",
"htmlwidgets", "leaflet", "mapview",
"Metrics", "openair", "plotKML", "psych",
"quantregForest", "randomForest", "raster",
"rasterVis", "reshape", "rgdal", "RSQLite",
"snow", "soiltexture", "sf", "sp"))
# Alternative spline function using the ihir package if GSIF doesn't install
install.packages("devtools")
library(devtools)
install_bitbucket("brendo1001/ithir/pkg")
```

Three common `Error` sources in R

1. Installing and loading packages
2. Case sensitivity
   - R is case sensitive not only for objects but also when using functions
   - Most things in R are lower-case, to make your life easier you should consider it when naming columns
   - Look out for exceptions like the function View()!

```
data <- c(2,3)

Data #For R data and Data are two different things

#Error: object 'Data' not found

Library(raster)#It's library not Library

#Error in Library(raster) : could not find function "Library"
```

Three common Error sources in R

3. Concatenation

- The c() function allows you to put an entire vector into a function instead of a single value

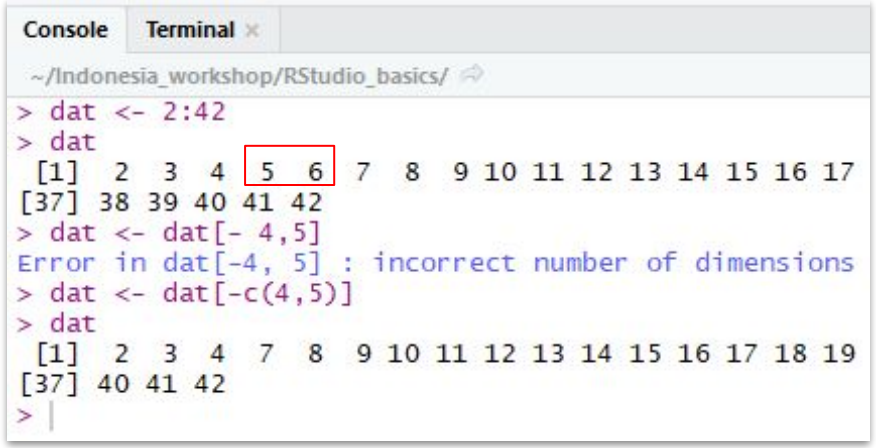
```
numbers <- 1:42

numbers <- numbers[-5,6]

#Error in numbers[-5, 6] : incorrect number

#of dimensions

numbers <- numbers[-c(5,6)]

numbers
```

```
> numbers <- 1:42
> numbers <- numbers[-c(5,6)]
> numbers
 [1]  1  2  3  4  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[31] 33 34 35 36 37 38 39 40 41 42
> numbers <- numbers[-5,6]
Error in numbers[-5, 6] : incorrect number of dimensions
>
```

Indices

- Numbers within the [ ] indicate the position of an object, e.g. in vectors the position of a number

```
dat <- 1:42

dat

dat <- dat[- 4,5]#[]are indices

dat <- dat[-c(4,5)];dat

dat <- 2:42

dat
```



Question 3: What numbers do you have to insert to get rid of the numbers 9 and 42? `dat <- dat[-c(?,?)];dat`

Types of Objects in R

- A *data object* can be a dataframe, a result of a calculation, a function etc., that you assign a name to
- There are different *object classes* (factor, integer, numeric, etc.) that determine what you can do with an object
- *Lists* are objects that can contain different data types

Data frame:

ID	Name	Value
1	Person1	67.5
2	Person2	33.75
3	Person3	16.875
4	Person4	8.4375
5	Person5	4.21875
6	Person6	2.109375

Numeric Vector:

Value
67.5
33.75
16.875
8.4375
4.21875
2.109375

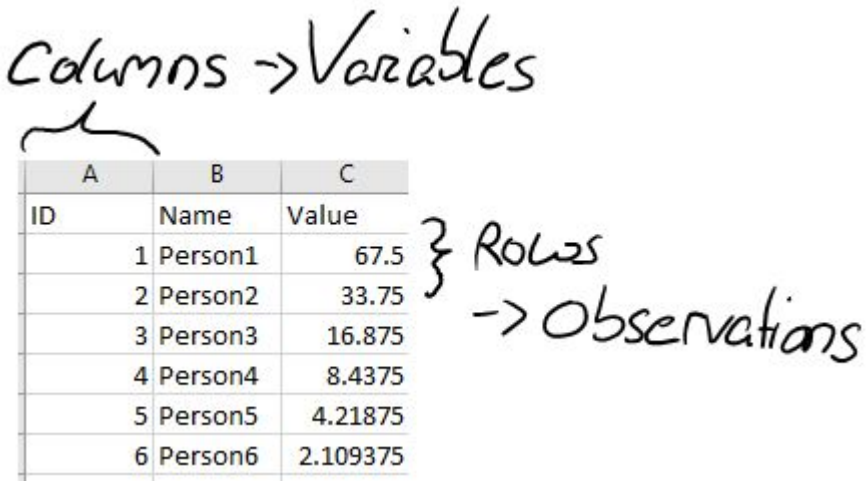
Time series:

Date	Value
2/3/2019	67.5
2/4/2019	33.75
2/5/2019	16.875
2/6/2019	8.4375
2/7/2019	4.21875
2/8/2019	2.109375
2/9/2019	1.054688

Datasets and dataframes

- Columns can have different data types (numeric, integer, logical, character, factor)
- All columns must have the same length

Data frame:



A	B	C
ID	Name	Value
1	Person1	67.5
2	Person2	33.75
3	Person3	16.875
4	Person4	8.4375
5	Person5	4.21875
6	Person6	2.109375

Datasets and dataframes

- R has many example datasets to practice on (e.g. iris and mtcars)
- In this example we're going to use iris and assign it to the object data.frame **dat.**

```
dat <- data.frame(iris)
```

- Here are some ways to explore your dataframe.

Question 4: What's the mean overall petal length?

```
View(dat)#Opens up a window showing you the whole dataset.

#You can also open it from the global environment by clicking on it

head(dat) #first 6 observations. tail(dat) gives you the last 6

names(dat) #Tells you the names and position of each variable

str(dat)#how many variables, observations, the class/type of data

summary(dat)#Tells you about the distribution of the data
#Minimum, Mean,Median,Maximum of each variable
```

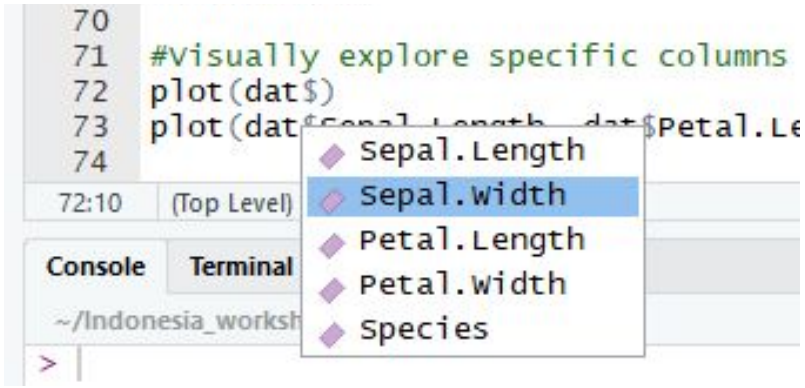
Datasets and dataframes

- R has many example datasets to practice on (e.g. iris and mtcars)
- In this example we're going to use iris and assign it to the object data.frame **dat**

```
dat <- data.frame(iris)
```

- Here are some ways to explore your dataframe

```
#Visually explore specific columns

hist(dat$Sepal.Width)#The $ sign allows you

#to select specific columns

plot(dat$Sepal.Length, dat$Petal.Length)
boxplot(dat$Petal.Width)
```



Datasets and dataframes

- Let's add another column. The new column will contain the species names with the number 2 attached to them and will be called "Species2"

```
dat$Species2 <- paste(dat$Species, 2)#with the $ sign you can #create a new column. The
paste function allows you to add #sequences to characters. This is useful when creating
an ID

View(dat)
```

- Let's check out the class type of the new column and change it to factor

```
str(dat)
dat$Species2 <- as.factor(dat$Species2); str(dat)
```

Datasets and dataframes

- To change the name of the columns we can use setnames()

```
names(dat)#check the name and position of each variable in the #console
```

- [1]"Sepal.Length""Sepal.Width""Petal.Length""Petal.Width""Species"

```
dat <- setNames(dat, c("SL","SW","PL","PW","S")) ;head(dat)
#the ; allows you to run two commands in the same line
```

- Calculate the mean Petal.Length

```
mean(dat$PL)
```

Question 5: What's the standard deviation of the petal length?

Exporting a dataset

- Let’s try exporting the train dataset as a csv

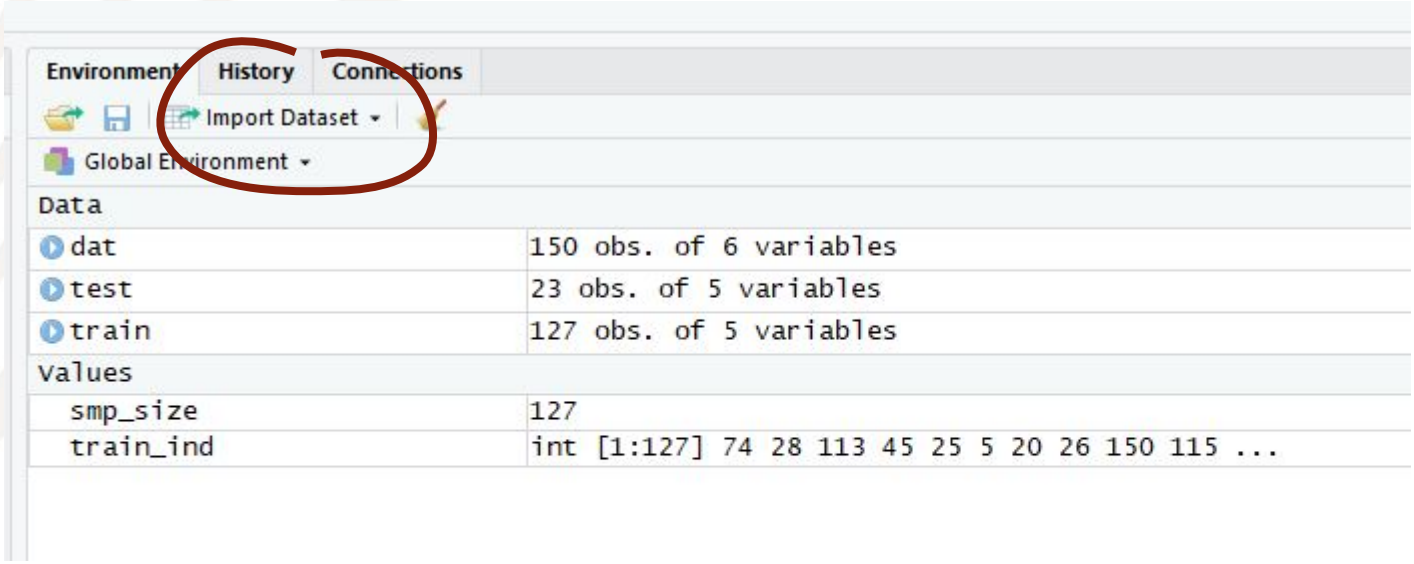
```
write.csv(dat, file= "iris2.csv")#Put the
#name of the file in ""
#If you want to save the file somewhere else other than the wd you #just have to specify
where
write.csv(dat, file= "folder_path/iris2.csv")
```

- You can export it into other formats as well with the functions

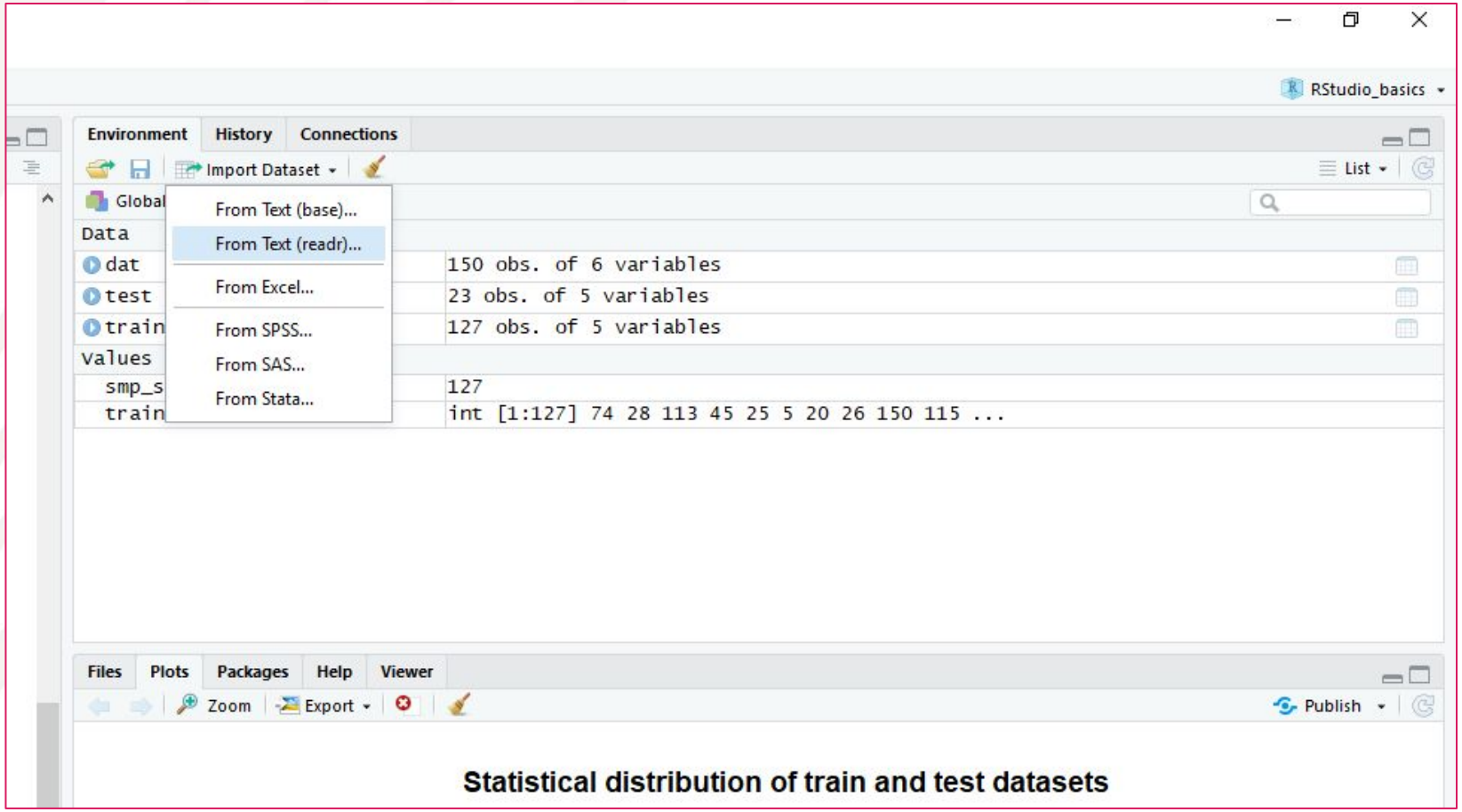
```
write.table()#Saves it as a .txt file
write.xlsx()#it requires the package xlsx
```

How to import datasets

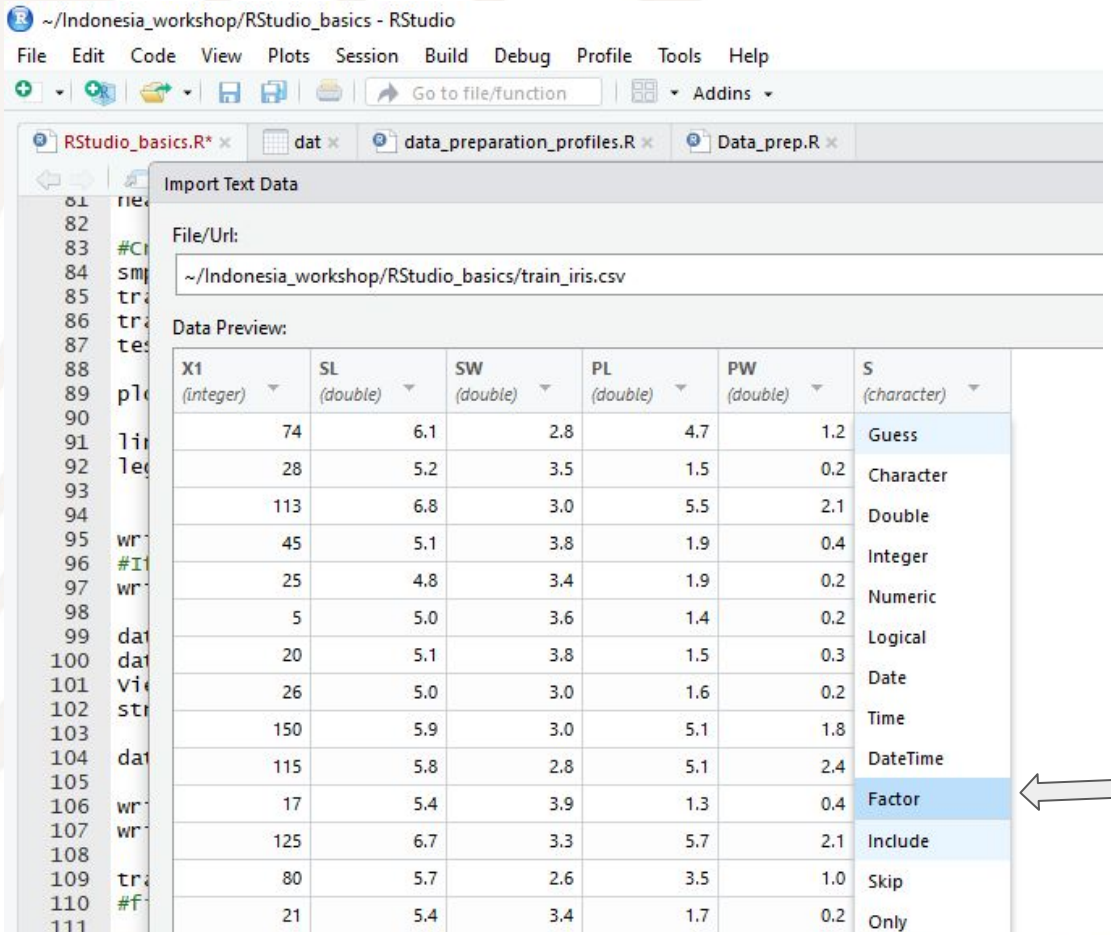
```
iris2 <- read.csv(file.choose()) #with file.choose() you don't #have to write file
location BUT it can freeze your R session

read.table()# for .txt files

read.xlsx()# requires the package xlsx
```



Importing data with readr



Importing data with readr



- You can define the data type of a specific variable directly when importing
- In this example we're defining that the species is a factor

File/Url:

~/Indonesia_workshop/RStudio_basics/train_iris.csv

Data Preview:

X1 (integer)	SL (double)	SW (double)	PL (double)	PW (double)	S (factor)
74	6.1	2.8	4.7	1.2	versicolor
28	5.2	3.5	1.5	0.2	setosa
113	6.8	3.0	5.5	2.1	virginica
45	5.1	3.8	1.9	0.4	setosa
25	4.8	3.4	1.9	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
20	5.1	3.8	1.5	0.3	setosa
26	5.0	3.0	1.6	0.2	setosa
150	5.9	3.0	5.1	1.8	virginica
115	5.8	2.8	5.1	2.4	virginica
17	5.4	3.9	1.3	0.4	setosa
125	6.7	3.3	5.7	2.1	virginica
80	5.7	2.6	3.5	1.0	versicolor
21	5.4	3.4	1.7	0.2	setosa
123	7.7	2.8	6.7	2.0	virginica
35	4.9	3.1	1.5	0.2	setosa
137	6.3	3.4	5.6	2.4	virginica
103	7.1	3.0	5.9	2.1	virginica
31	4.8	3.1	1.6	0.2	setosa
41	5.0	3.5	1.3	0.3	setosa
68	5.8	2.7	4.1	1.0	versicolor
44	5.0	3.5	1.6	0.6	setosa

Previewing first 50 entries.

Import Options:

Name: dat
Skip: 0

First Row as Names
Trim Spaces
Open Data Viewer

Delimiter: Comma
Quotes: Default
Locale: Configure...

Escape: None
Comment: Default
NA: Default

Code Preview:

```
library(readr)
dat <- read_csv("train_iris.csv", col_types = cols(S = col_factor(levels = c("versicolor",
    "setosa", "virginica"))))
View(dat)
```

Reading rectangular data using readr

Types of Objects in R

- A *data object* can be a dataframe, a result of a calculation, a function etc., that you assign a name to
- There are different *object classes* (factor, integer, numeric, etc.) that determine what you can do with an object
- *Lists* are objects that can contain different data types

Data frame:

ID	Name	Value
1	Person1	67.5
2	Person2	33.75
3	Person3	16.875
4	Person4	8.4375
5	Person5	4.21875
6	Person6	2.109375

Numeric Vector:

Value
67.5
33.75
16.875
8.4375
4.21875
2.109375

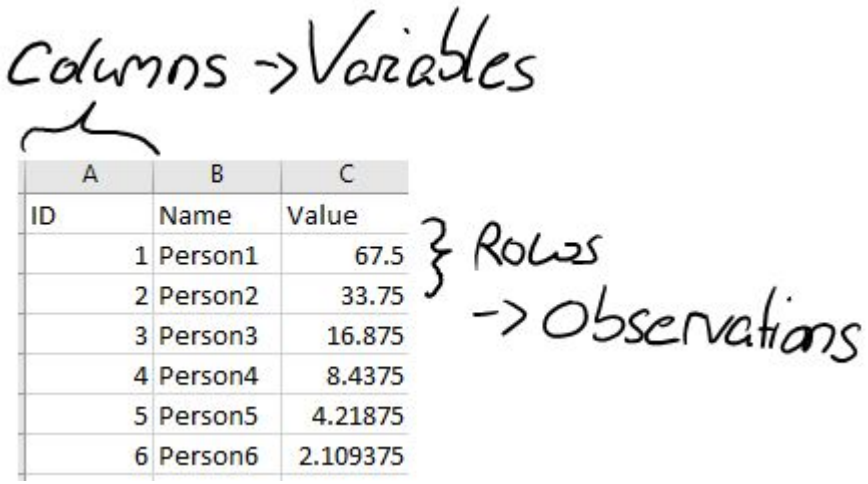
Time series:

Date	Value
2/3/2019	67.5
2/4/2019	33.75
2/5/2019	16.875
2/6/2019	8.4375
2/7/2019	4.21875
2/8/2019	2.109375
2/9/2019	1.054688

Datasets and dataframes

- Columns can have different data types (numeric, integer, logical, character, factor)
- All columns must have the same length

Data frame:



A	B	C
ID	Name	Value
1	Person1	67.5
2	Person2	33.75
3	Person3	16.875
4	Person4	8.4375
5	Person5	4.21875
6	Person6	2.109375

Conditional Selection []

- Show only certain rows or columns [row index , column index] a way to remember this is by saying: rows comma column

```
dat[c(1,2), ]#by leaving the section after the comma blank we're selecting all the
columns
 dat <- dat[ , -6];head(dat)#let's get rid of the last column
```

- This is one way to create a random subsample of the data

```
 #Create a random subset with 85% of the data

smp_size <- floor(0.85 * nrow(dat))#define the subsample size
train_ind <- sample(seq_len(nrow(dat)), size = smp_size)
train <- dat[train_ind, ]
test <- dat[-train_ind, ]

wrist.csv(train, file = "train_iris.csv")
wrist.csv(test, file = "test_iris.csv")
```

Question 6: What number is in column 3 row 42?

Other logical operators

- ==, >, >=, <, <=, !=, &, |
- Let's subset the data for petals with a length greater than 5

```
Subset <- dat[dat$PL> 5,]
```

- Let's subset the data for petals with a length greater than 5 AND that are of the Setosa species

```
dat[dat$PL >5 & dat$S == "virginica", ]
```

- Let's subset the data for petals with a length greater than 5 OR that have sepals longer than 3

```
dat[dat$PL >5 | dat$SL > 3, ]
```

Question 7: How many observations are in the object Subset?

Other logical operators

- %in% allows you to verify if an element is part of another object
- It can be used for instance to get rid of outliers
- Let's add an outlier in our iris dataset

```
dat[c(9,3,4,6),4]=c(20,24,15,42)

View(dat)
```

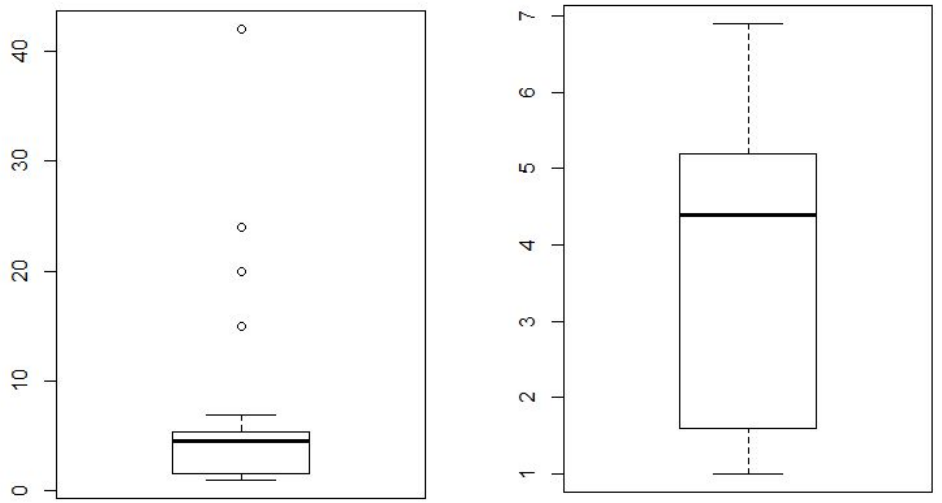
TRUE

FALSE

TRUE

How to remove outliers

```
out <-boxplot(dat$PW, range = 1.5, plot= FALSE)$out

out

dat <- dat[-which(dat$PW %in% out),] ;boxplot(dat$PW)
View(dat)

boxplot(dat$PW)
```



How to remove NAs

- Let’s remove NAs from our dataset

```
dat <- data.frame(iris)

dat <- setNames(dat, c("SL","SW","PL","PW","S"))
dat$PL2 <- dat$PL#first let’s create a column with NAs

dat[c(9,3,12,6),6]=NA


summary(dat)#the summary function is good
#to check if there are any NAs


dat <- dat[complete.cases(dat),]#the complete
#cases function selects all the rows without
#NAs
```

PL	PW	S	PL2
4.7	1.2	versicolor	4.7
1.5	0.2	setosa	1.5
5.5	2.1	virginica	NA
1.9	0.4	setosa	1.9
1.9	0.2	setosa	1.9
1.4	0.2	setosa	NA
1.5	0.3	setosa	1.5
1.6	0.2	setosa	1.6
5.1	1.8	virginica	NA

Basic graphs in R

- To check which graphical parameters do what the command ?par is very useful
- To tweak your plot type ?plot
- Let's make a boxplot with the Iris dataset

```
dat <- data.frame(iris); ?par

dat$Species <- as.factor(dat$Species)

par(bty="l", family = "mono")#we want an L shaped graph with #font "mono"

boxplot(dat$Petal.Length ~ dat$Species,

        main = "Petal length by Species",

        col.main = "#009999",

        ylab = "Petal length [cm]",xlab= '')
```

Basic graphs in R

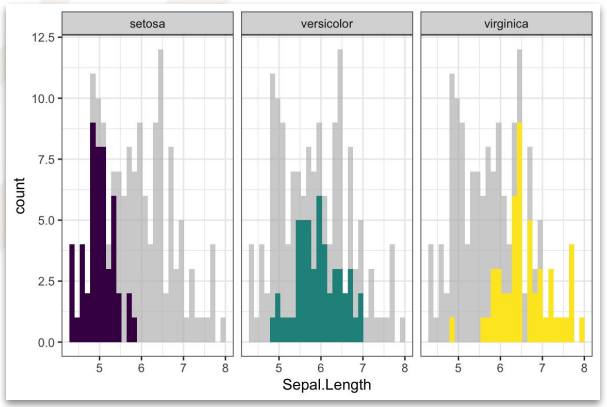
- To check which graphical parameters do what the command ?par is very useful
- To tweak your plot type ?plot
- Let's add points showing the mean petal length per species

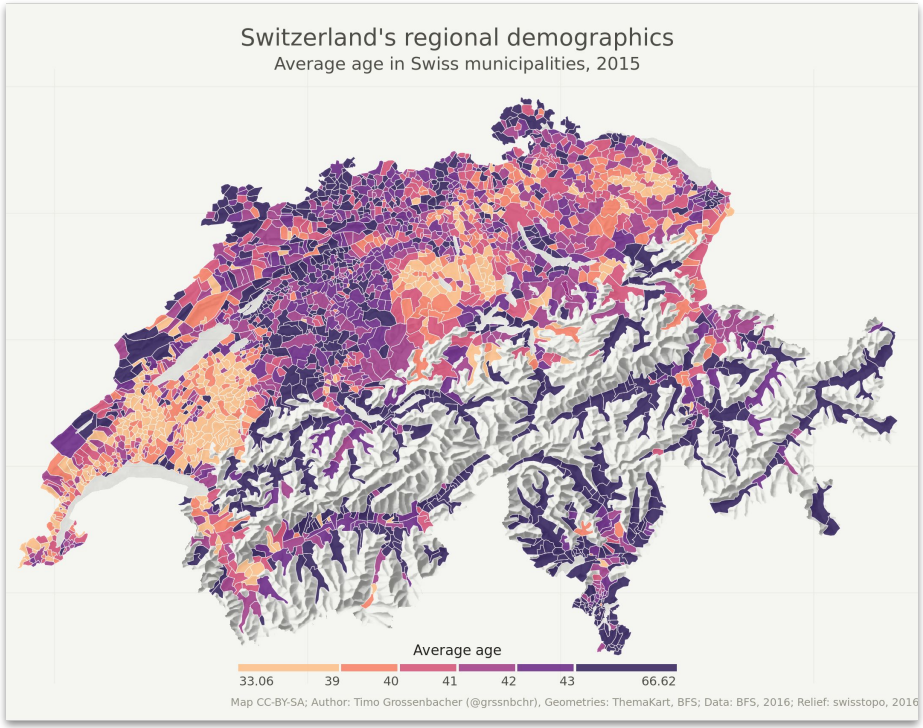
```
attach(dat)#makes it so that you don't have to use $

means <- by(Petal.Length, Species, mean)#The by function is used for data frames

points(means, col ="#48D1CC", pch=19, cex=1.2)
```

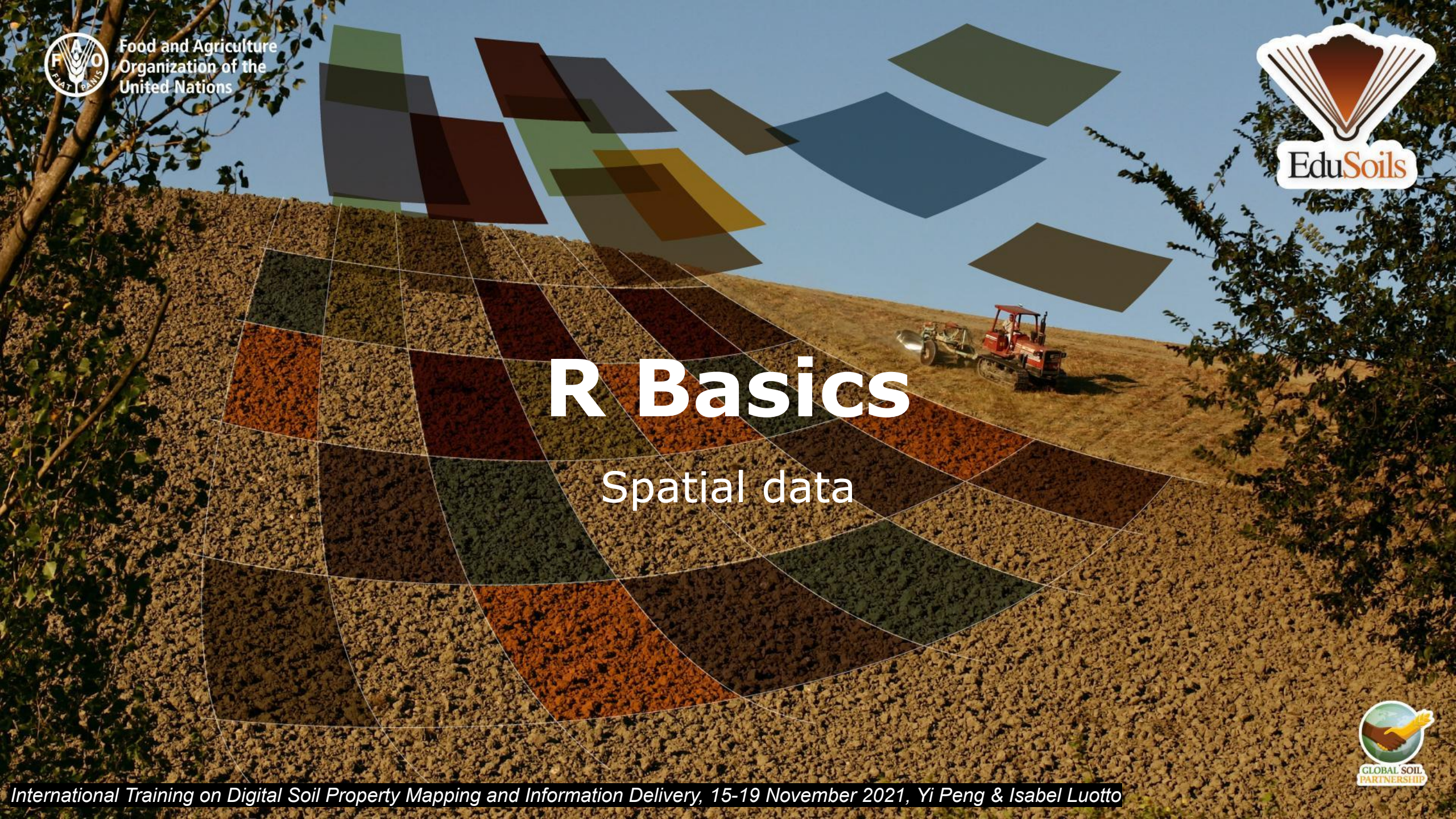
- Different HEX color codes can be found here:
  https://www.rapidtables.com/web/color/html-color-codes.html

ggplot2

- A famous and widely used package for making graphs and maps is ggplot2
- ggplot2 has its own language and it takes time to master it, but it’s definitely worth it

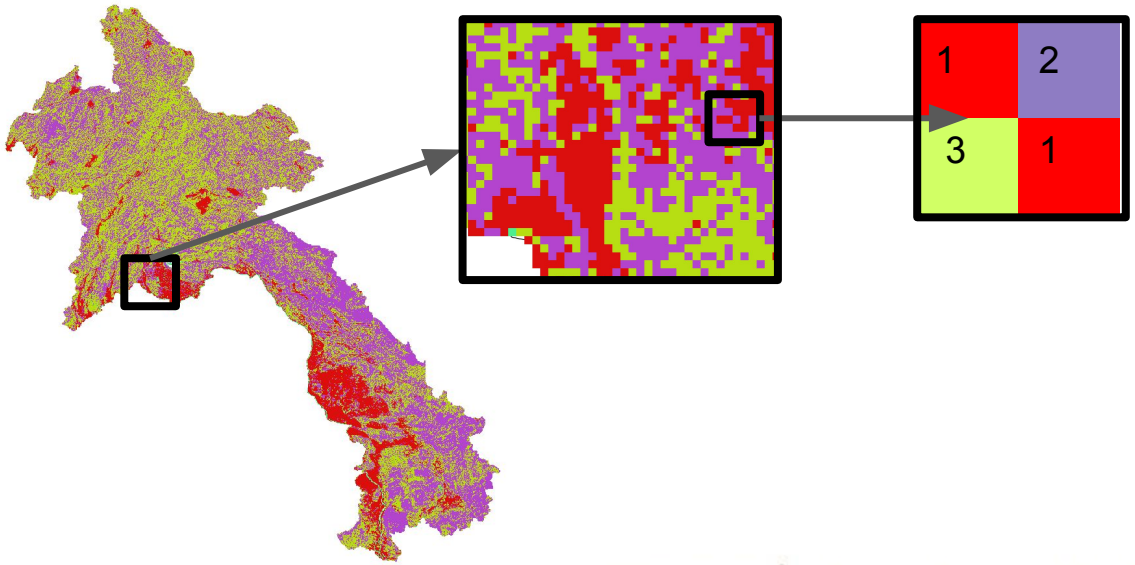





Food and Agriculture
Organization of the
United Nations
EduSoils
R Basics
Spatial data
GLOBAL SOIL
PARTNERSHIP

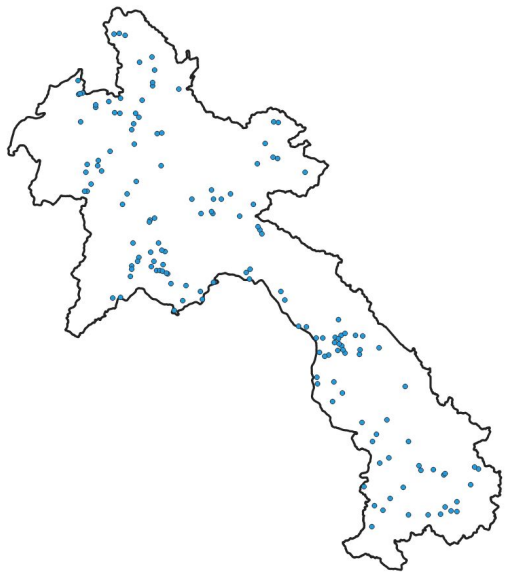
Rasters vs. Vectors

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

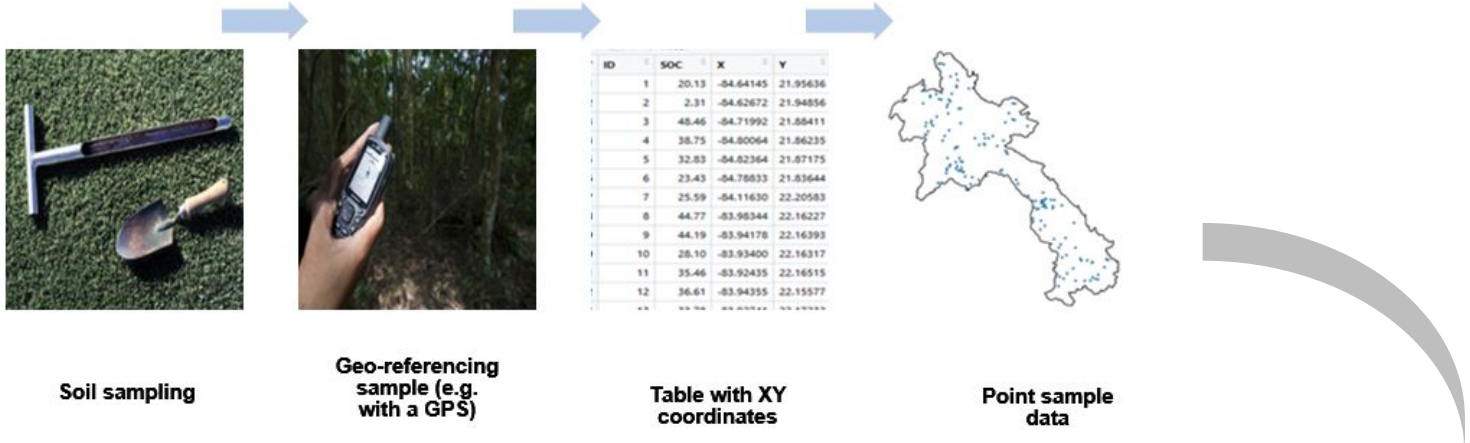


Rasters vs Vectors

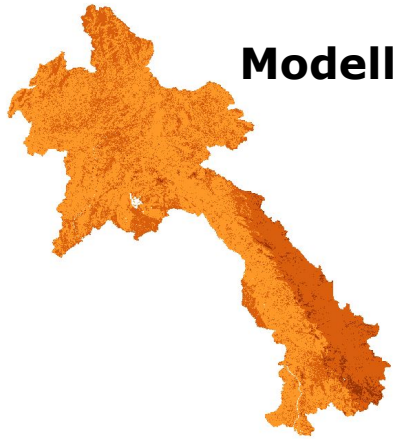
- Vector: points, lines and polygons
- For storing data that has discrete boundaries, such as country borders, land parcels, and streets.
- Format: shapefile



Digital soil mapping



- In digital soil mapping we mostly work with data in table format and then rasterize this data so that we can make a continuous map



Modelling/Mapping

EduSoils | e-learning soil educational platform

R packages for digital soil mapping

`aqp`	• Algorithms for quantitative pedology • We will use it to restructure our soil profile dataset into a database form that is easier to work with
`raster`	• Reading, writing, manipulating, analyzing and modeling of gridded spatial data
`rgdal`	• Provides access to the 'Geospatial' Data Abstraction Library ('GDAL') to projection/transformation operations from the 'PROJ.4' • We will use it to the define the Coordinate Reference System (CRS)
`sp`	• It is used to turn data frames into spatial objects (e.g. SpatialPointdataframe)

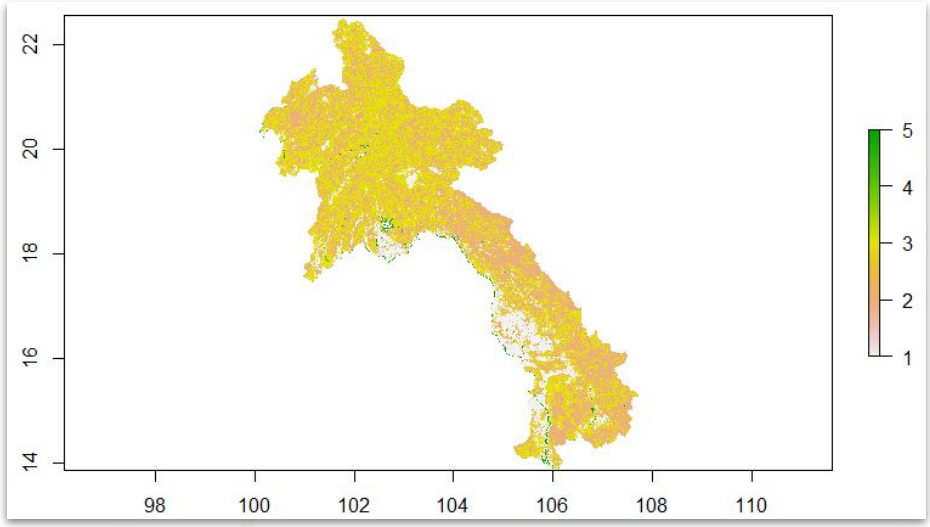
Working with spatial data in R

- To familiarize with handling spatial data in R today we will
  1. Load a raster and explore it
  2. Explore some of the plotting functionalities
  3. Match the extent of one raster to another
  4. Make a rasterStack
  5. Change the CRS of a raster
  6. Save a raster
  7. Load a data frame with XY coordinates
  8. Plot points and overlay them over a raster

Load a raster file into R

- Most of the functions for handling raster data are available in the **raster** package

```
library(raster)

landcover <- raster("01-Data\\land cover\\LandCover.tif")

#Let's check the CRS and extent of

#our landcover map

landcover

plot(landcover)#Let's plot our lc

summary(landcover)
```

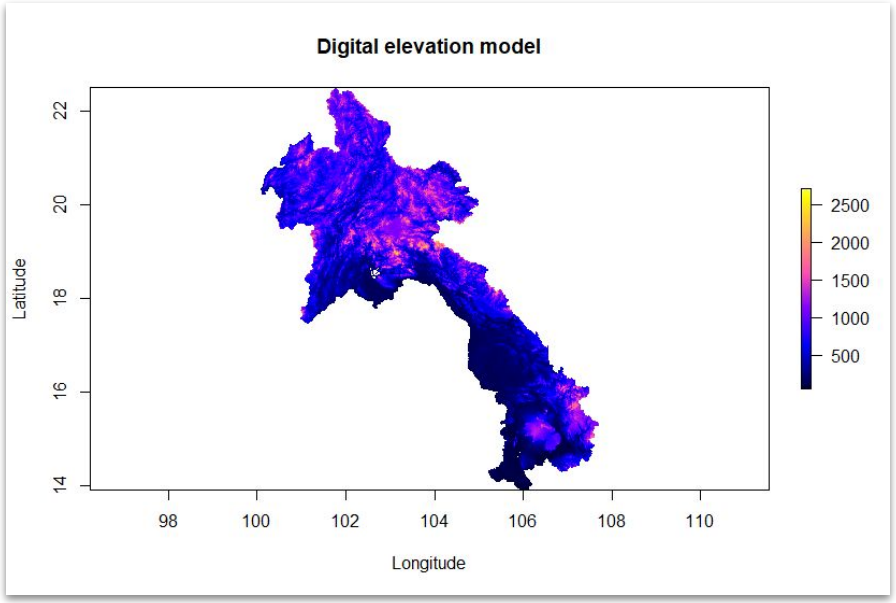


Load a raster file into R

- Most of the functions for handling raster data are available in the **raster** package
- Let's load a digital elevation model

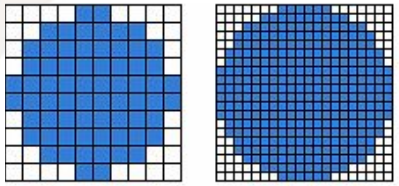
```
DEM <- raster("01-Data\\covs\\DEMENV5.tif")
#You can customize your map similarly
#like we did for the graphs
plot(DEM, col = bpy.colors(255),
main= "Digital elevation model",
xlab = "Longitude", ylab = "Latitude")
```

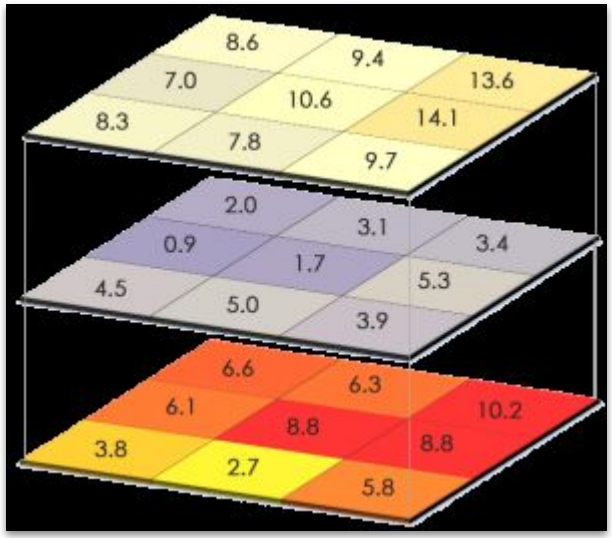
Question 8: What's the exact highest point in our DEM?



The RasterStack

- Most of the functions for handling raster data are available in the **raster** package
- Multiple layers can be combined into the object class RasterStack, which allows you to perform things on multiple rasters at once
- Rasters can be *stacked* only if they have the same:
  - Extent (or in other words they cover the same area),
  - The same file extension
  - Same projection
  - Pixel resolution (cell size)





The RasterStack

```
#let's try stacking our landcover and DEM map
lc_and_dem <-stack(landcover, DEM)
```

- Error in compareRaster(x) : different extent

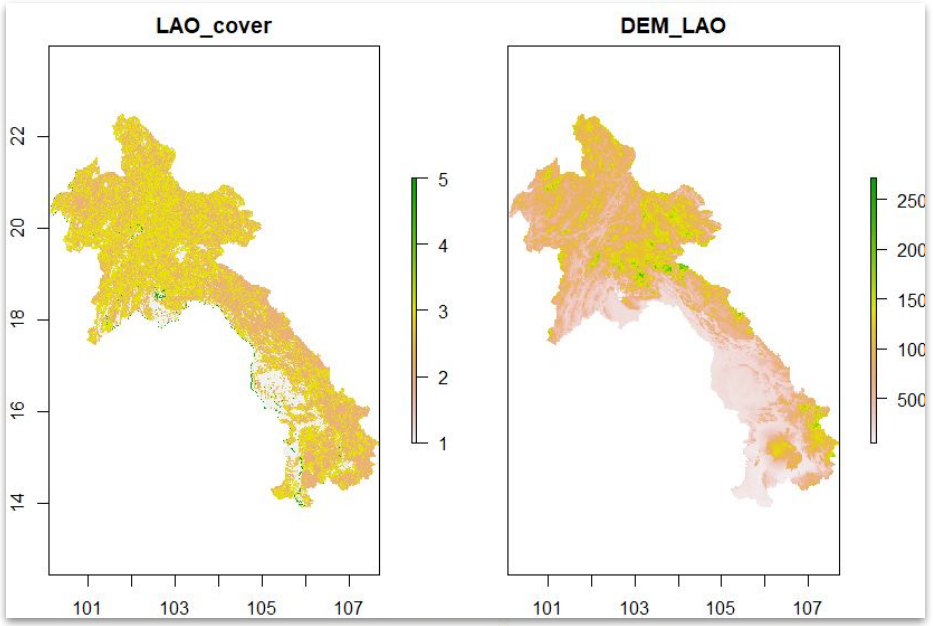
Console | Terminal

~/Indonesia_workshop/RStudio_basics/

```
> plot(landcover)
> #Let's check the CRS and extent of our landcover map
> landcover
class       : RasterLayer
dimensions  : 1041, 925, 962925  (nrow, ncol, ncell)
resolution  : 0.00833, 0.00833  (x, y)
extent      : 100.0434, 107.7487, 13.87458, 22.54611  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=krass +towgs84=44.585,-131.212,-39.544,0,0,0,0 +no_defs
data source : C:/Users/luottoi/Documents/Indonesia_workshop/RStudio_basics/LAO_cover.tif
names       : LAO_cover
values      : 1, 5  (min, max)

> DEM
class       : RasterLayer
dimensions  : 1031, 915, 943365  (nrow, ncol, ncell)
resolution  : 0.008329293, 0.008332328  (x, y)
extent      : 100.0849, 107.7062, 13.91399, 22.50462  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : C:/Users/luottoi/Documents/Indonesia_workshop/RStudio_basics/DEM_LAO.tif
names       : DEM_LAO
values      : -32768, 32767  (min, max)
```

How to reproject a raster

```
#Match the extent and resolution to DEM

landcover <- projectRaster(landcover, DEM, method = 'ngb')


lc_and_dem <-stack(landcover, DEM)
plot(lc_and_dem)
#Now you can plot two

#rasters at the same time
```



How to change the CRS of a raster

- With the function reproject you can directly define the CRS of your raster

```
landcover <- raster("01-Data\\land cover\\LandCover.tif")#Reload #the original raster

#Let's check the CRS and extent of our landcover map

landcover
```

```
class       : RasterLayer
dimensions  : 564, 957, 539748  (
resolution  : 231, 308  (x, y)
extent      : 452142.3, 673209.3,
coord. ref. : +proj=utm +zone=34
data source : C:/Users/Isabe/Desk
ndCover.tif
names       : LandCover
values      : 11, 210  (min, max)
```

How to change the CRS of a raster

- With the function reproject you can directly define the CRS of your raster

```
landcoverWGS84 <- projectRaster(landcover, crs = CRS("+init=epsg:4326"),method='ngb')
```

EPSG:4326

WGS 84 -- WGS84 - World Geodetic System 1984, used in GPS

Center coordinates

0.00000000 0.00000000

WGS84 bounds:

-180.0 -90.0

180.0 90.0

World.

How to save a raster

- The function writeRaster() allows you to save rasters in the .tif format
- This function can also be used to change convert other formats to .tif

```
writeRaster(landcoverWGS84, file= "LandcoverWGS84.tif", overwrite=TRUE)
```

Let’s create a SpatialPointDataframe

- SpatialPointsDataFrame structure is essentially like a data frame, except that additional “spatial” elements have been added
- To define the **CRS**, we must know where our data is from, and what was the corresponding **CRS** used when recording the spatial information **in the field**
- First we’re going to import a dataframe

```
#Import a csv file

points <- read.csv("01-Data\\site-level.csv")

head(points)
```

Let’s create a SpatialPointDataframe

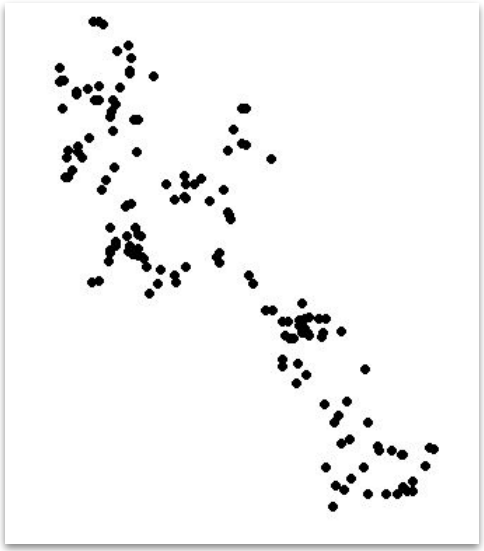
- Let’s turn our data frame into a SpatialPointDataframe
- We can use the coordinates() function to tell R which columns contain a spatial reference

```
library(sp)
# Promote to SpatialPointsDataFrame
coordinates(points) <- ~ X + Y
#Check the the class
class(points)
```

Plotting points

- For now we only told R that our data frame contains XY coordinates, we still have to define the projection

```
#Let's define the projection to EPSG 4326
points@proj4string <- CRS(projargs = "+init=epsg:4326")
points@proj4string


#Let's check the physical distribution of our data points
plot(points, pch = 16)
```



Overlay points on a raster

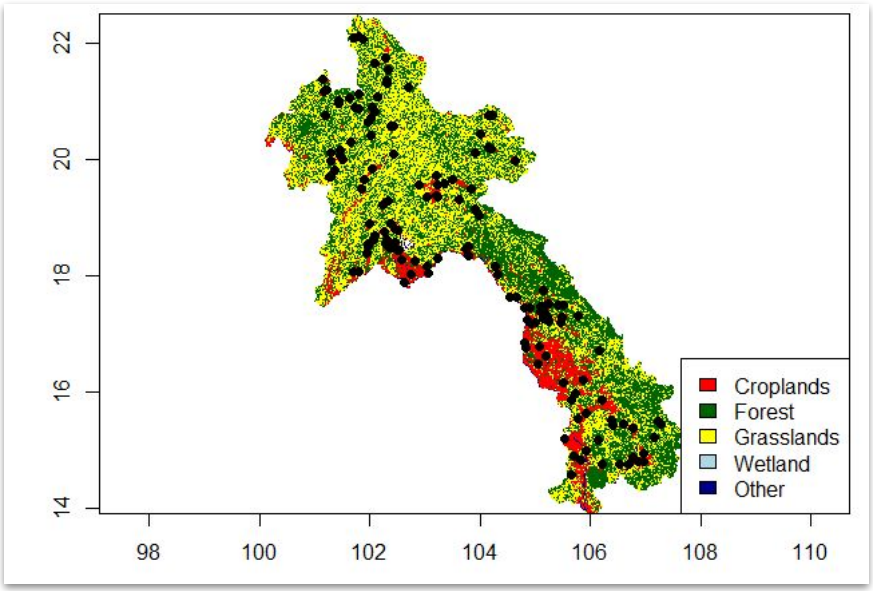
- To overlay the points over a raster we use the points() function
- First we need to make sure that that the raster and map are in the same projection
- Plotting your data points is useful to explore the spatial distribution of your points (e.g. detect cluster, areas with very few representative points

```
landcover <- projectRaster(landcover, crs = CRS("+init=epsg:4326"),method='ngb')
```

Overlay points on a raster

- To overlay the points over a raster we use the points() function

```
#Let's plot our land cover map and add a legend
plot(landcover, col= c("red", "darkgreen", "yellow", "lightblue", "darkblue"), legend=
FALSE)

legend("bottomright",  legend =
c("Croplands", "Forest", "Grasslands",
"Wetland", "Other"),
fill = c("red", "darkgreen","yellow",
"Lightblue", "darkblue" ))

#Check the distribution of the points
points(points, pch=16)
```



Rural Development Administration

AFACI
Asian Food & Agriculture Cooperation Initiative

Food and Agriculture Organization of the United Nations

EduSoils

GLOBAL SOIL PARTNERSHIP