



Food and Agriculture  
Organization of the  
United Nations



中国援助  
CHINA AID  
FOR SHARED FUTURE



Rural Development  
Administration



EduSoils

# R Basics

Essentials for Getting Started  
*GSP-Secretariat*  
*Isabel Luotto & Marcos Angelini*





# What is R?

R: Engine



RStudio: Dashboard



FIGURE 1.1: Analogy of difference between R and RStudio.

# What is R and RStudio?

- R is an object-based programming language that runs computations
- RStudio is an integrated development environment (IDE) that provides an interface by adding many convenient features and tools

So just as the way of having access to a speedometer, rearview mirrors, and a navigation system makes driving much easier, using RStudio's interface makes using R much easier as well

## Why R?

- **R** is **free** to install, use, update, clone, modify, redistribute, even sell.
- It works in all operative systems.
- R's strong **package ecosystem** and **charting benefits**
- Clean, analyse, plot, and communicate all from the **same place**
- Keep track of your steps -> **Reproducibility**
- Reduce processing time -> **Automation**
- Everything you **do** in the analysis, from deleting outliers to interpreting results, is contained in your code
- State-of-the-art **graphics**
- Maintained by its large community: great support!
- Most tools for **digital soil mapping** are written in R

# Additional learning material

Using R for Digital Soil Mapping

<https://link.springer.com/book/10.1007/978-3-319-44327-0#:~:text=This%20book%20describes%20and%20provides,and%20spatial%20data%20in%20R.>

Soil Organic Carbon Cookbook

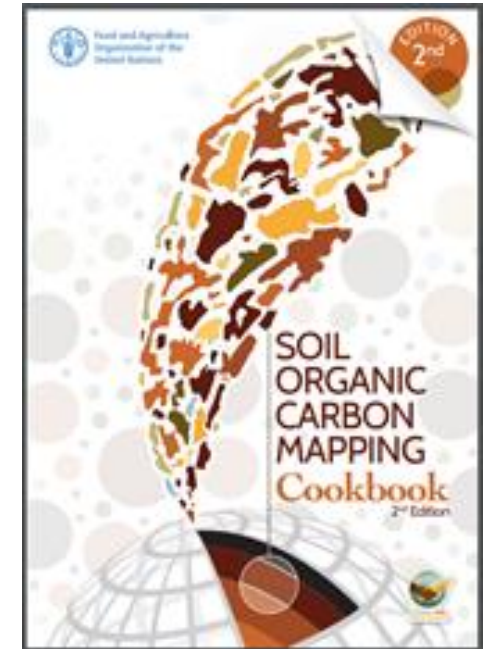
<http://www.fao.org/3/I8895EN/i8895en.pdf>

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>



# Installation

1. Download and install R and Rtools

<https://cloud.r-project.org/> <https://cran.r-project.org/bin/windows/Rtools/rtools42/rtools.html>

1. Download and install RStudio

<https://www.rstudio.com/products/rstudio/download/>

---

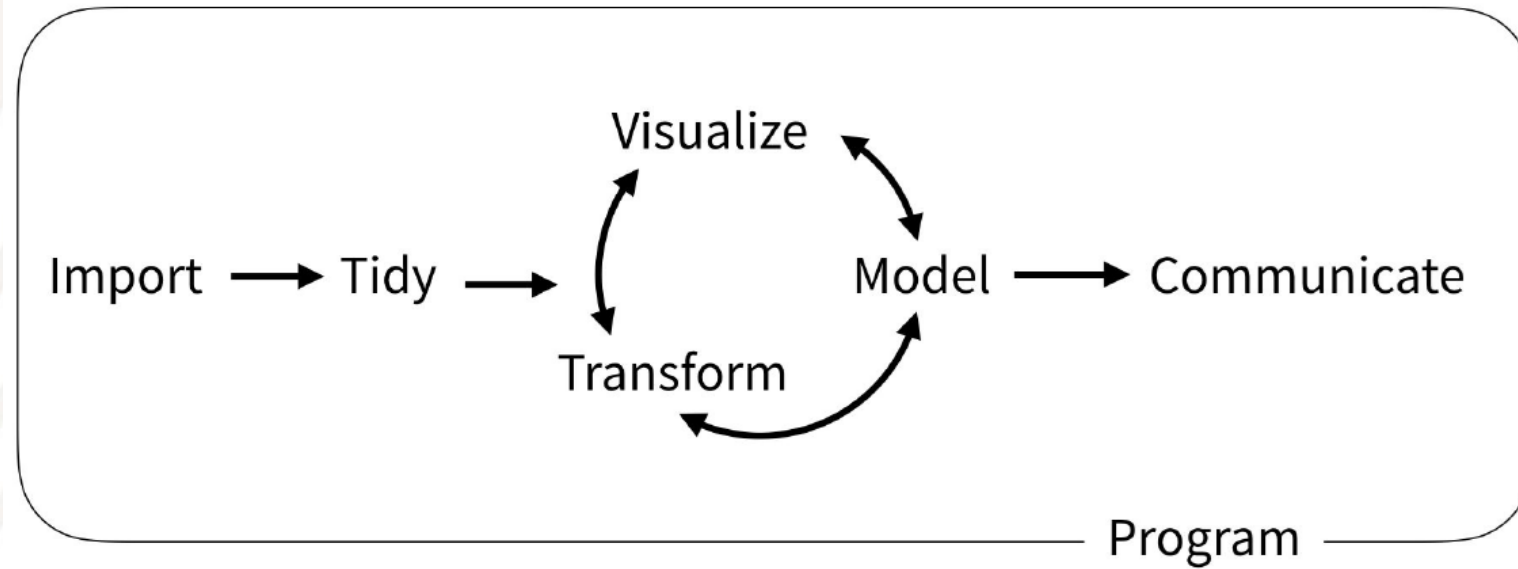
**R: Do not open this**



**RStudio: Open this**



# General workflow in R





# RStudio interface

The image shows the RStudio interface with several panels and annotations. The main editor window on the left contains an R script with the following text:

```
1 # Introduction to R  
2  
3
```

Below the script, there is an annotation: "R script: We write text (scripts/code) here".

The Environment panel on the right shows "Global Environment" and "Environment is empty". Below it, there is an annotation: "Environment: Shows the saved objects".

The Files panel at the bottom right shows a file explorer view of the directory "C:\GIT". The files listed are:

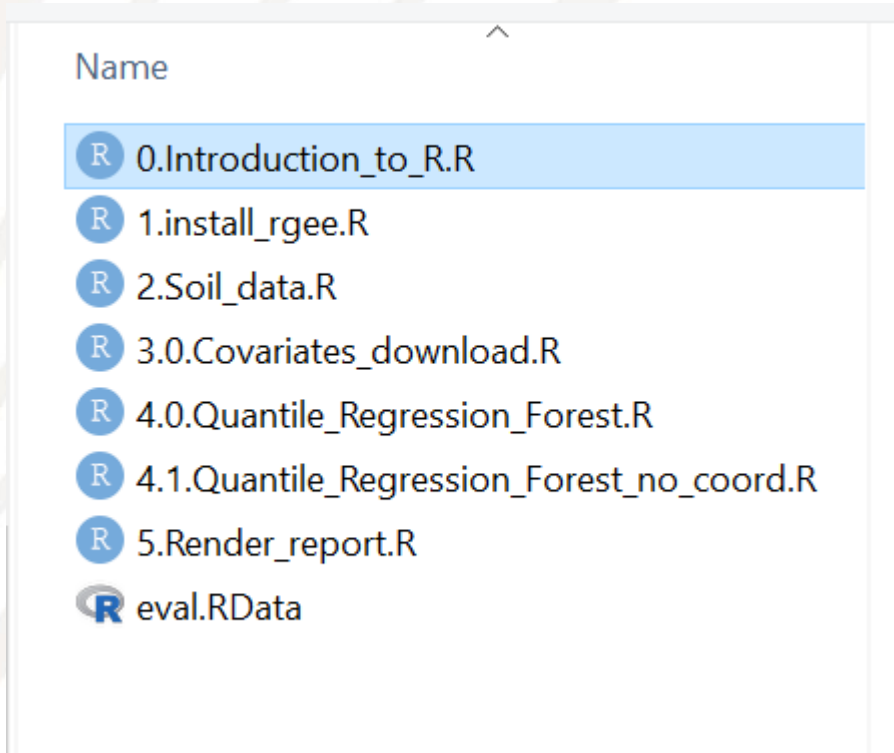
- ..
- ..\_bookdown\_files
- AFACI
- Digital-Soil-Mapping
- Equi7Grid
- SOC-Mapping-Cookbook
- Vietnam-DSM-workshop

Below the file list, there is an annotation: "Files, plot, packages, and help!".

The Console panel at the bottom left shows the R prompt and the current directory: "R 4.1.1 · C:/GIT/Vietnam-DSM-workshop/". Below it, there is an annotation: "Console: The real interface of R".



# Open the script 0.Introduction\_to\_R.R



# Main parts of R syntax

- **Functions:** `mean()`, `read_csv()`, `plot()`
  - Arguments in functions: `mean(x = 1:10)` which is the same than `mean(1:10)`
- **Objects**
  - Vectors: concatenated values
  - Dataframes: spreadsheets
  - Lists: group of any objects, like a folder
  - Others
- **Type of data in objects**
  - Numeric: 1, 2, 3, 4.5, 6.02
  - Character: "a", "b", "1", "this is a character"
  - Factors: characters with levels
- **Packages:**
  - Add-ons that contain functions for an specific use: "base", "tidyverse", "raster", etc.

# Main parts of R syntax

- Operators
  - Assign: `<- (=)`
  - Equal: `a == b`
  - Different: `a != b`
  - Column: `$`, `dataframe$column_1`
  - Subset: `[]`,
    - `vector[1]` is the first element of the vector
    - `Dataframe[row,column]`, e.g. `dataframe[1,]` is the first row of the dataframe, `dataframe[,1:5]` are the first columns of the dataframe
    - `List[[1]]` is the first object of the list
  - Concatenate functions: `%>%` (pipe symbol)
  - Other symbols with expected behavior: `+`, `-`, `*`, `/`, `>`, `<`, `^` (power), `&` (and), `|` (or)
- R is case sensitive
  - “THIS” is not the same than “this”
- Many packages -> many different ways to do the same
  - Some packages require an specific syntax, such as `ggplot2`



# Other logical operators

- `%in%` allows you to verify if an element is part of another object

## Logical and boolean operators to use with `filter()`

<code>==</code>	<code>&lt;</code>	<code>&lt;=</code>	<code>is.na()</code>	<code>%in%</code>	<code> </code>	<code>xor()</code>
<code>!=</code>	<code>&gt;</code>	<code>&gt;=</code>	<code>!is.na()</code>	<code>!</code>	<code>&amp;</code>	

See `?base::Logic` and `?Comparison` for help.



TRUE



FALSE



`%in%`



TRUE

# We will use the following packages

- Tabular-data management: **tidyverse**
- Plotting:
  - Tabular data: **ggplot2**
  - Maps: **mapview** and **tmap**
- For spatial data
  - Rasters: **raster** and **terra**
  - Shape files: **sf** and **terra**
  - Access remote sensing data: **rgee**
  - Other GIS tools: **rgdal**
- Modelling: **caret**

# Tidyverse

<https://www.tidyverse.org/>

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

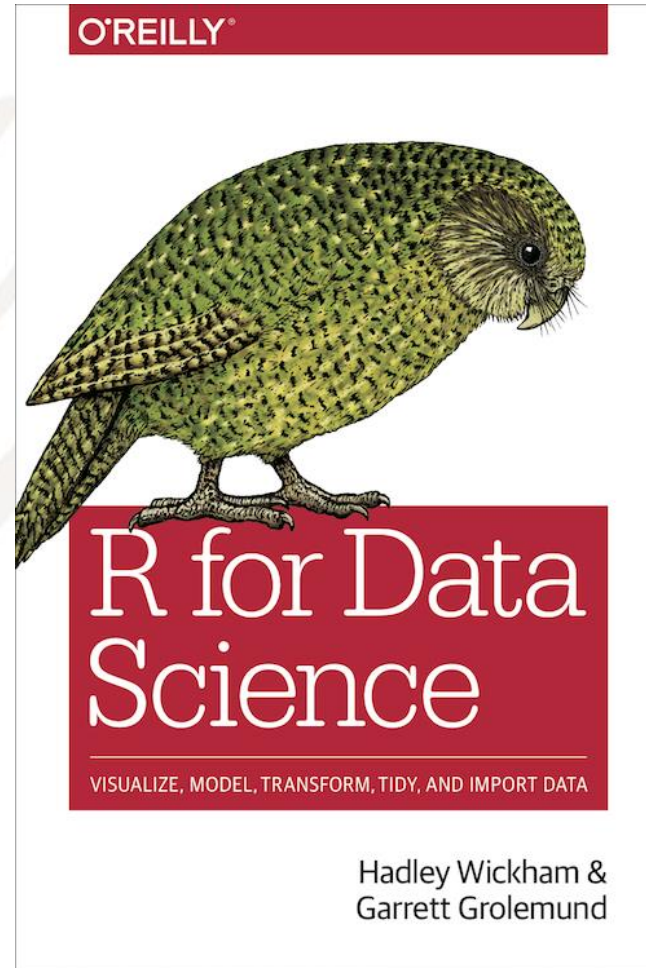




# Tidyverse

<https://r4ds.had.co.nz/>

Chapter 5:  
Data transformation



# Tidyverse

- Concatenate function with (`%>%`).
- Pick variables by their names (`select()`).
- Pick observations by their values (`filter()`).
- Create new variables with functions of existing variables (`mutate()` & `transmute()`).
- Group table by key variable(s) (`group_by()`).
- Collapse many values down to a single summary (`summarise()`).
- Reshape the structure of the table (`pivot_longer()`, `pivot_wider()`).
- Join relational tables (`left_join()`, `right_join()`, `full_join()`).

country	year	cases	population
Afghanistan	1999	7745	19987071
Afghanistan	2000	6666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174004898
China	1999	210258	1272015272
China	2000	210766	1280048583

variables

country	year	cases	population
Afghanistan	1999	7745	19987071
Afghanistan	2000	6666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174004898
China	1999	210258	1272015272
China	2000	210766	1280048583

observations

country	year	cases	population
Afghanistan	1999	7745	19987071
Afghanistan	2000	6666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174004898
China	1999	210258	1272015272
China	2000	210766	1280048583

values

# Tidyverse: Pipe %>%

- Data
  - %>%
    - subproduct 1
      - %>%
        - subproduct 2
          - %>%
            - result



# Tidyverse: select()

- Pick variables by their names (select()).



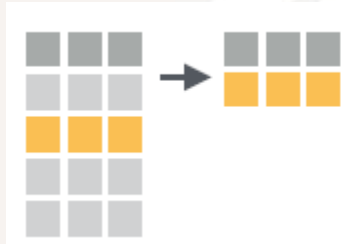
**select(.data, ...)** Extract columns as a table.  
`select(mtcars, mpg, wt)`

## Exercise

- Select variables `id_prof`, `id_hor`, `top`, `bottom`, `ph_h2o`, `cec` and save as a new object called `dat_1`

# Tidyverse: filter()

- Pick observations by their values ([filter\(\)](#)).



**filter(.data, ..., .preserve = FALSE)** Extract rows that meet logical criteria.  
`filter(mtcars, mpg > 20)`

## Exercise

- Filter observations from `dat_1` with more than 50 cmolc/kg cec and save it as `dat_2`

# Tidyverse: mutate()

- Create new variables with functions of existing variables (`mutate()` & `transmute()`).



**mutate**(.data, ..., .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also **add\_column()**, **add\_count()**, and **add\_tally()**.  
`mutate(mtcars, gpm = 1 / mpg)`



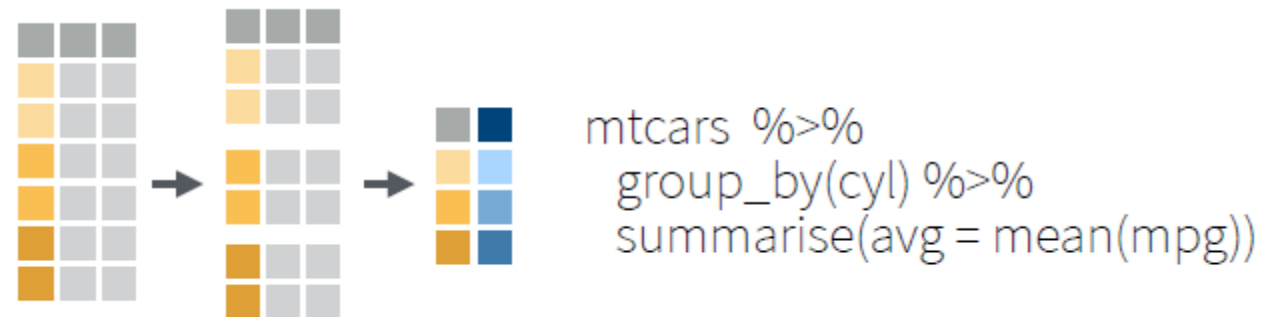
**transmute**(.data, ...) Compute new column(s), drop others.  
`transmute(mtcars, gpm = 1 / mpg)`

## Exercise

- Create a column with the thickness of the horizon. Column name "thickness". Save the new object as `dat_3`. What is the difference with `transmute`?

# Tidyverse: `group_by()` & `summarise()`

- Group table by key variable(s) (`group_by()`).
- Collapse many values down to a single summary (`summarise()`).



## Exercise

- Compute the mean of pH and mean of cec for each soil profile (pid) using `dat_3` as input. Save as `dat_4`



# Tidyverse: pivot

- Reshape the structure of the table (`pivot_longer()`, `pivot_wider()`).

## Exercise

- Using `dat_3`, put the names of the variables `ph_h2o`, `cec` and `thickness` in the column `soil_property`, and the values of the soil property in the value column. Keep the rest of the table. Save in `dat_5`

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

**`pivot_longer`**(data, cols, names\_to = "name", values\_to = "value", values\_drop\_na = FALSE)

"Lengthen" data by collapsing several columns into two. Column names move to a new names\_to column and values to a new values\_to column.

```
pivot_longer(table4a, cols = 2:3, names_to = "year", values_to = "cases")
```

**`pivot_wider`**(data, names\_from = "name", values\_from = "value")

The inverse of `pivot_longer`(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

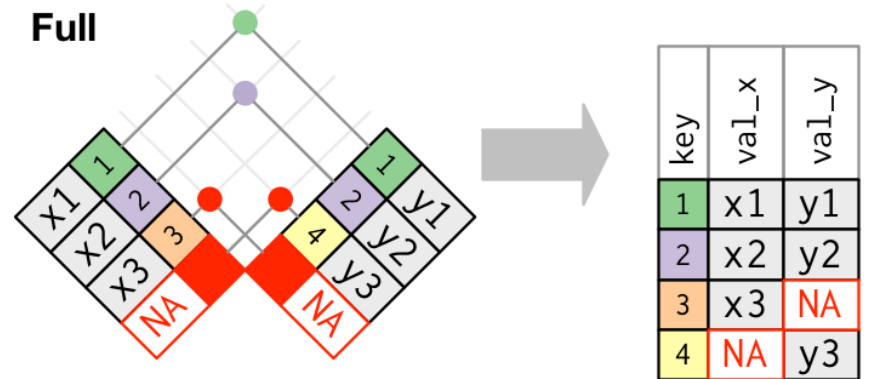
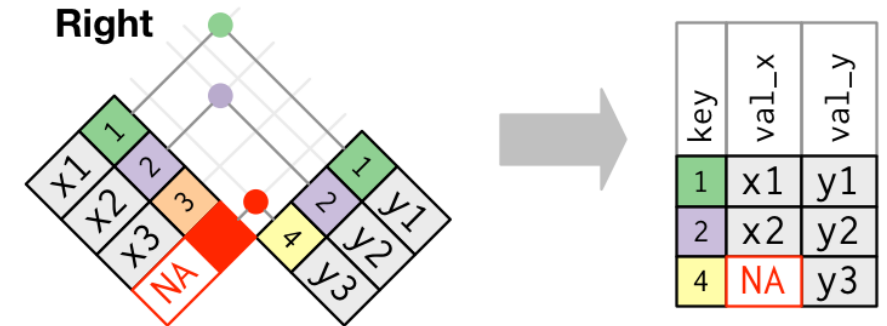
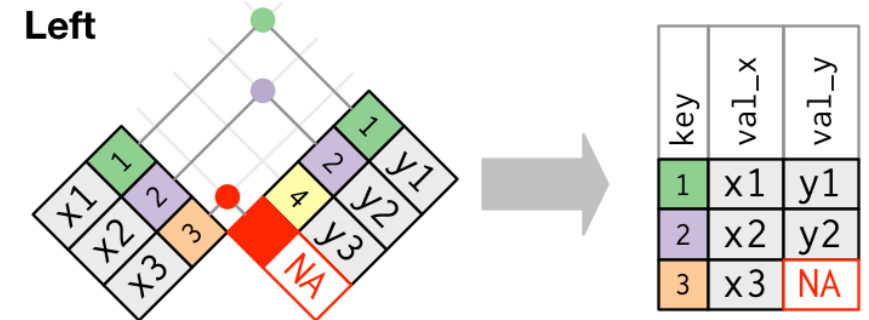
```
pivot_wider(table2, names_from = type, values_from = count)
```

# Tidyverse: join

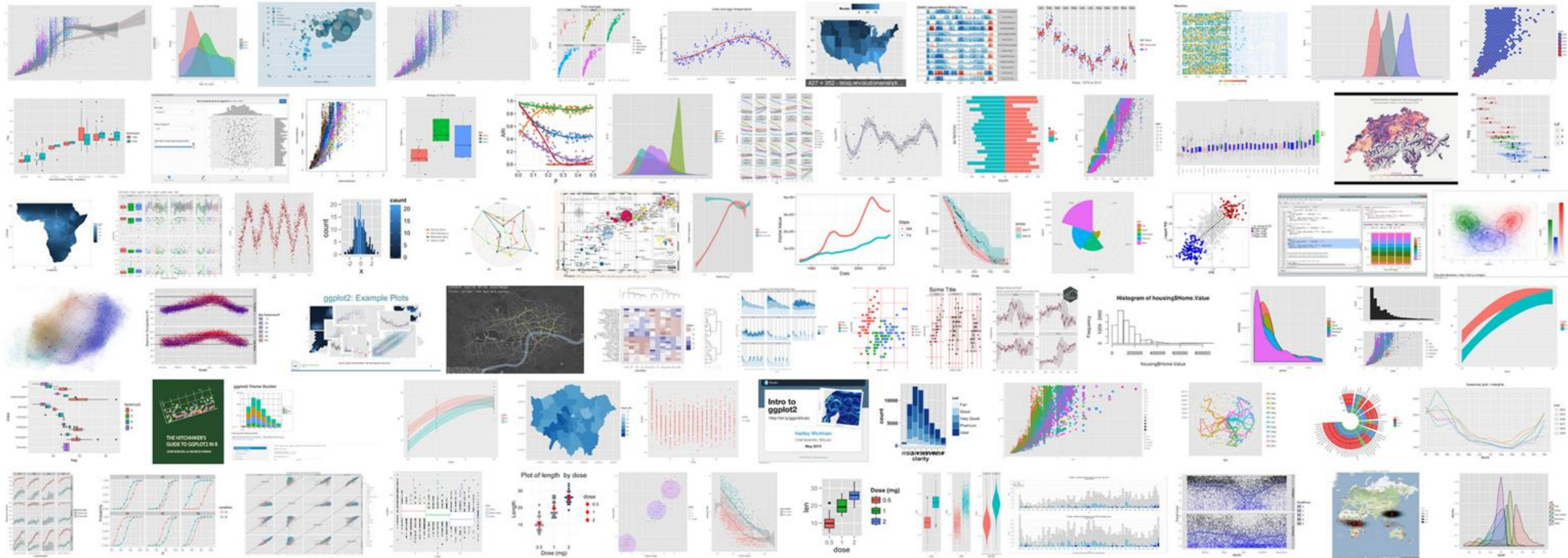
- Join relational tables (`left_join()`, `right_join()`, `full_join()`)

## Exercise

- Load `site.csv` (in 01-Data folder) and join its columns with `dat_3`. Use `pid` as key. Save the result as `dat_6`



# Data Visualization with ggplot2



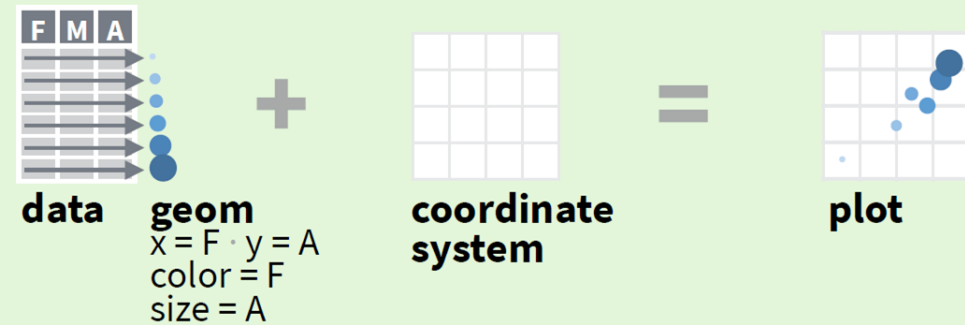
# Data Visualization with ggplot2

## Basics

**ggplot2** is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.





# Data Visualization with ggplot2

data

```
ggplot(mpg, aes(hwy, cty)) +  
  geom_point(aes(color = cyl)) +  
  geom_smooth(method = "lm") +  
  coord_cartesian() +  
  scale_color_gradient() +  
  theme_bw()
```

add layers,  
elements with +

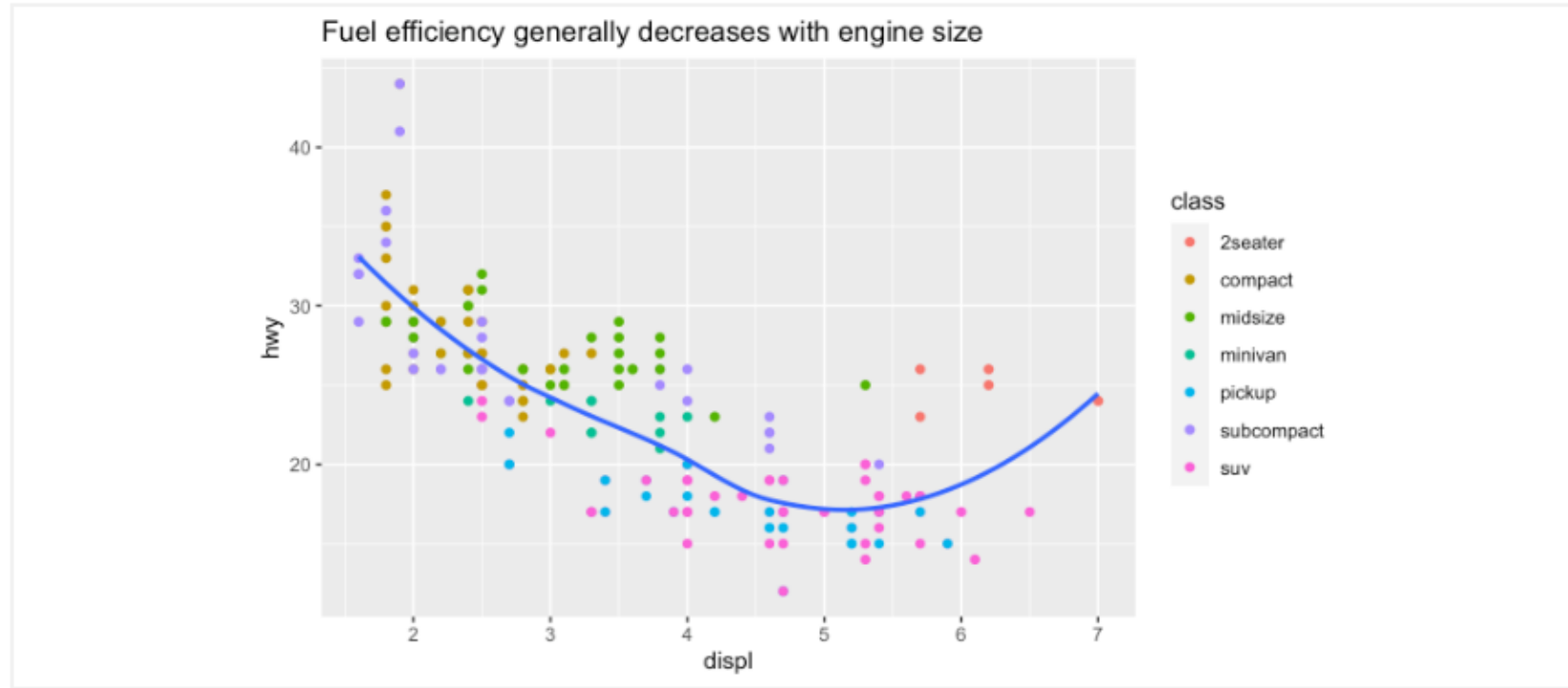
layer = geom +  
default stat +  
layer specific  
mappings

additional  
elements

# Data Visualization with ggplot2

```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point(aes(color = class)) +  
  geom_smooth(se = FALSE) +  
  labs(title = "Fuel efficiency generally decreases with engine size")
```

Copy



## ggplot2: One Dimension

Basic structure

```
ggplot(data=..., aes(x=..., y=...)) +  
  geom_...()
```

Exercise

- Make a histogram (geom\_histogram()) of cec and pH using dat\_3

# ggplot2: Two Dimensions

Basic structure

```
ggplot(data=..., aes(x=..., y=...)) +  
  geom_...()
```

Exercise

- Make a scatterplot (`geom_point()`) of cec in x and pH in y using `dat_3`
- Then, add a fitting line (`geom_smooth()`)



# ggplot2: Three Dimensions

Basic structure

```
ggplot(data=..., aes(x=..., y=...)) +  
  geom_...()
```

Exercise

- Make a scatterplot (`geom_point()`) of cec in x and pH in y using `dat_3`
- Add cec as argument color and size

# ggplot2: Three Dimensions

Basic structure

```
ggplot(data=..., aes(x=..., y=...)) +  
  geom_...()
```

Exercise

- Make a scatterplot (geom\_point()) of cec in x and pH in y using dat\_3
- Add cec as argument color and size

## ggplot2: Facets

Basic structure

```
ggplot(data=..., aes(x=..., y=...)) +  
  geom_...() +  
  facet_wrap(~variable)
```

Exercise

- Make a scatterplot (geom\_point()) of cec in x and pH in y using dat\_3
- Add cec as argument color and size





Food and Agriculture  
Organization of the  
United Nations



中国援助  
CHINA AID  
FOR SHARED FUTURE



Rural Development  
Administration



EduSoils

# R Basics

Spatial data

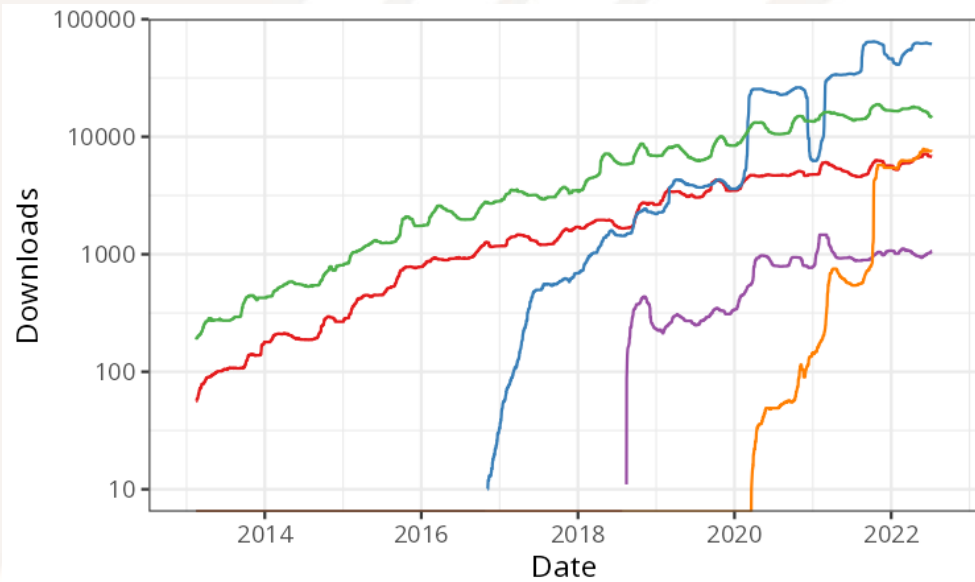


GLOBAL SOIL  
PARTNERSHIP



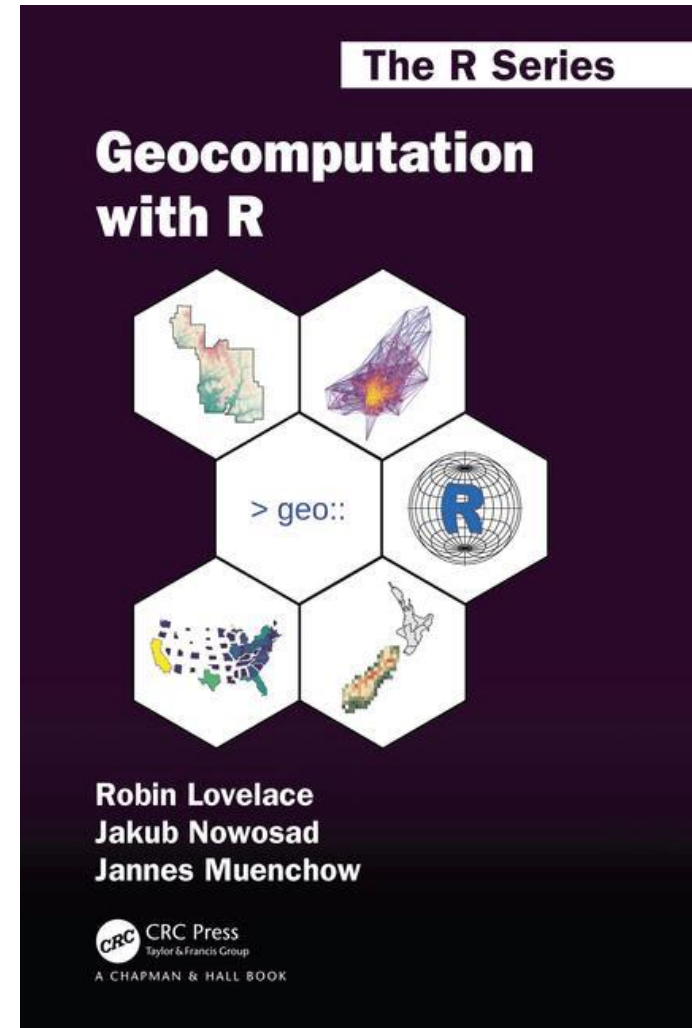
# Geospatial data in R

<https://geocompr.robinlovelace.net/index.html>



Package:

- raster
- sf
- sp
- stars
- terra



EduSoils | e-learning soil educational platform



# Main concepts

Geospatial data is data with spatial coordinates represented in a coordinate reference system (CRS)

Search the CRS of your country at <https://epsg.io/>

**EPSG:32648** Share on:

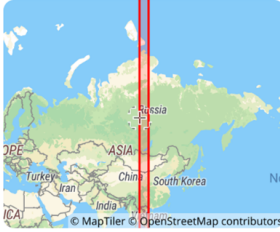
WGS 84 / UTM zone 48N

Attributes

<b>Unit:</b> metre	<b>Scope:</b> Engineering survey, topographic mapping.
<b>Geodetic CRS:</b> WGS 84	<b>Area of use:</b> Between 102°E and 108°E, northern hemisphere between equator and 84°N, onshore and offshore. Cambodia. China. Indonesia. Laos. Malaysia - West Malaysia. Mongolia. Russian Federation. Singapore. Thailand. Vietnam.
<b>Datum:</b> World Geodetic System 1984 ensemble	
<b>Data source:</b> EPSG	
<b>Revision date:</b> 2020-03-14	

**Coordinate system:** Cartesian 2D CS. Axes: easting, northing (E,N). Orientations: east, north. UoM: m.

Covered area powered by MapTiler



**Center coordinates**  
500000.0 4649776.22

**Projected bounds:**  
166021.44 0.0  
534994.66 9329005.18

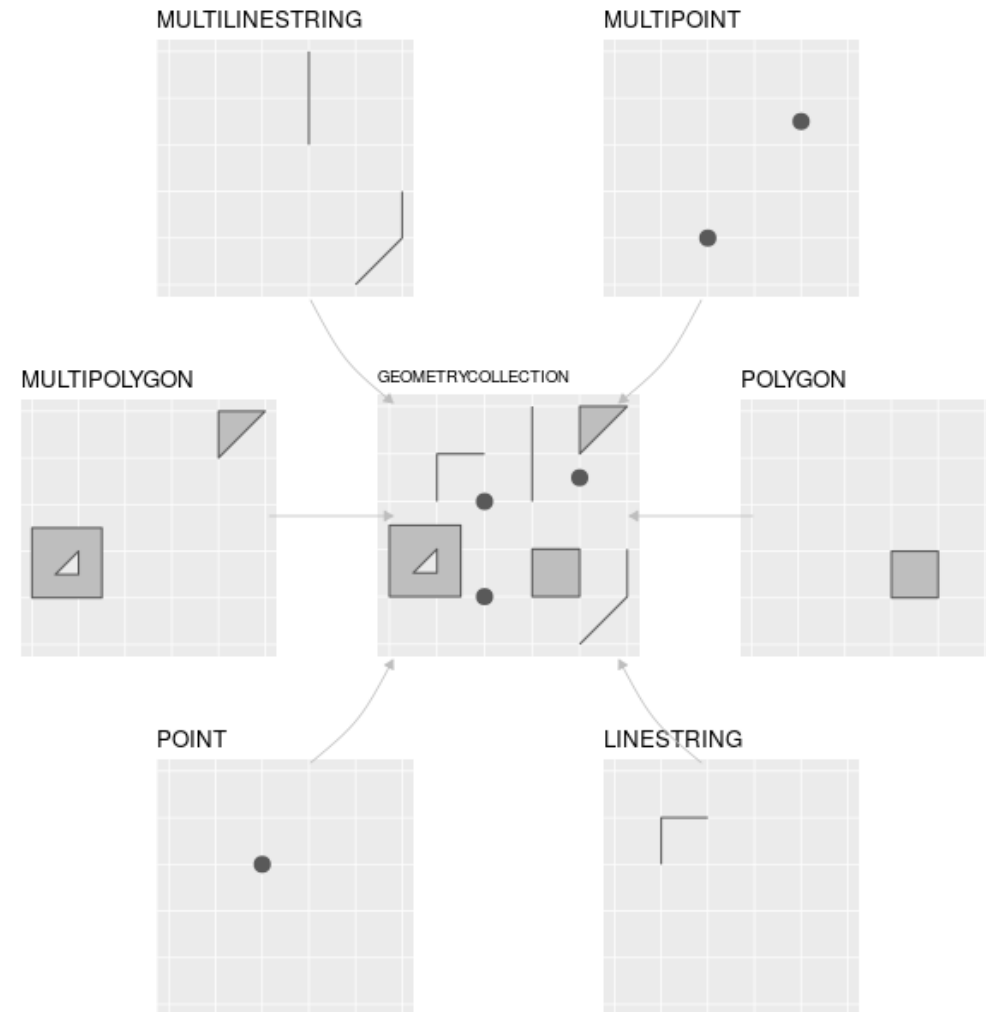
**WGS84 bounds:**  
102.0 0.0  
108.0 84.0



The screenshot shows the epsg.io website. At the top, the logo "epsg.io" is displayed with a QR code icon to its right. Below the logo, it says "From MapTiler Team". The main heading is "Coordinate Systems Worldwide". Below this is a search bar with the placeholder text "Country, code or name of a coordinate system ..." and a "SEARCH" button.

# Vector data

- Vector data represent spatial discrete object, such as regions, roads, rivers, cities, etc.
- Simple features (sf) is an open standard (ISO 19125-1:2004) developed and endorsed by the Open Geospatial Consortium (OGC). The package sf has been developed to manage this type of data
- Attributes are typically stored in data.frame objects. Geometries are also stored in a data.frame column.
- sf objects can be treated as a data.frame

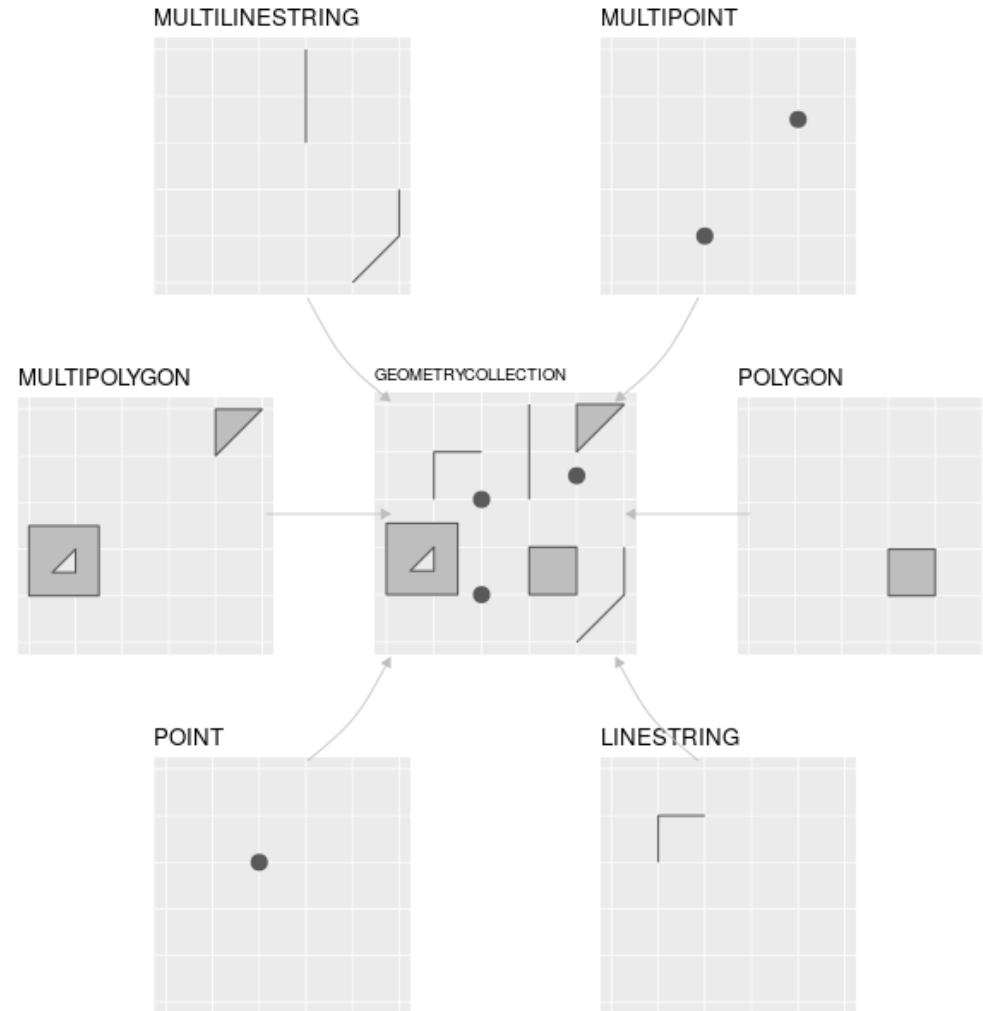


# Vector data

```
## Simple feature collection with 100 features and 6 fields
## geometry type: MULTIPOLYGON
## dimension: XY
## bbox: xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID): 4267
## proj4string: +proj=longlat +datum=NAD27 +no_defs
## precision: double (default; no precision model)
## First 3 features:
```

	BIR74	SID74	NWBIR74	BIR79	SID79	NWBIR79	geom
## 1	1091	1	10	1364	0	19	MULTIPOLYGON((( -81.47275543...
## 2	487	0	10	542	3	12	MULTIPOLYGON((( -81.23989105...
## 3	3188	5	208	3616	6	260	MULTIPOLYGON((( -80.45634460...

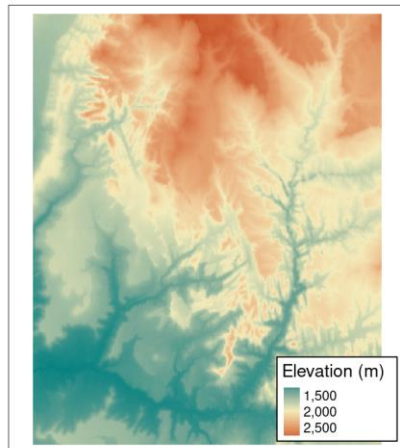
Simple feature (green arrow pointing to row 1)  
 Simple feature geometry list-column (sfc) (red arrow pointing to geom column)  
 Simple feature geometry (sfg) (blue arrow pointing to MULTIPOLYGON text)



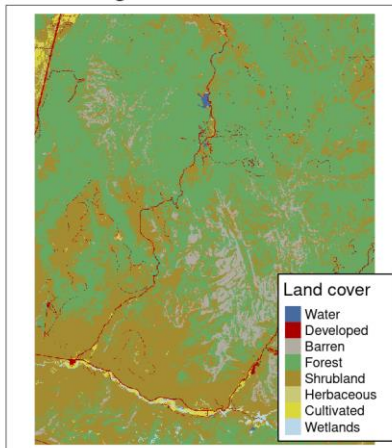
# Raster data

- The spatial raster data model represents the world with the continuous grid of cells
- It is an appropriate format for continuous variables, such as remote sensing data, terrain attributes, etc., although also categorical data can be represented

A. Continuous data



B. Categorical data



A. Cell IDs

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

B. Cell values

92	55	48	21
58	70	NA	37
NA	12	94	11
36	83	4	88

C. Colored values

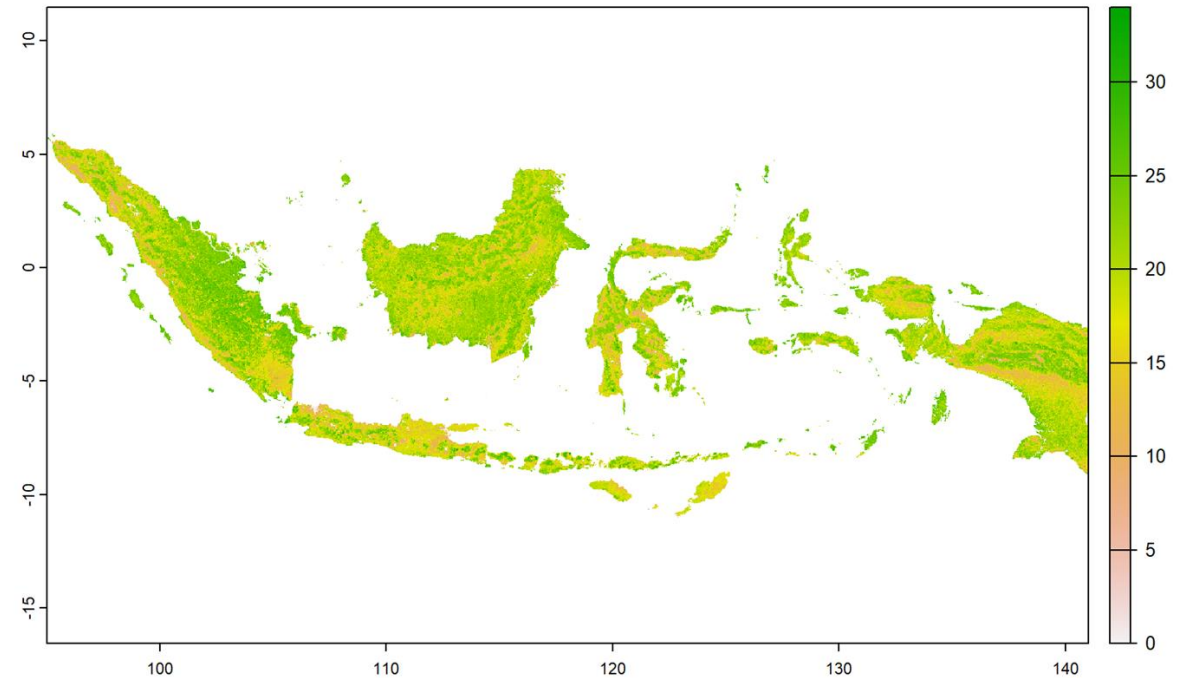




# Raster data

- The main package for handling raster data was called "raster" and has been recently by "terra". However, many packages still use raster as raster data manager
- The main attributes of a raster model are:
  - Number of rows and columns
  - Number of layers
  - Pixel size (or resolution)
  - CRS

```
#> class      : SpatRaster
#> dimensions  : 457, 465, 1 (nrow, ncol, nlyr)
#> resolution  : 0.000833, 0.000833 (x, y)
#> extent     : -113, -113, 37.1, 37.5 (xmin, xmax, ymin, ymax)
#> coord. ref. : lon/lat WGS 84 (EPSG:4326)
#> source     : srtm.tif
#> name       : srtm
#> min value  : 1024
#> max value  : 2892
```



# Raster data

- terra can also handle vector data, which is specially recommended when rasters and vectors need to be combined in any way (convert vector to raster, zonal statistics, etc.)

# Digital soil mapping



Soil sampling



Geo-referencing sample (e.g. with a GPS)

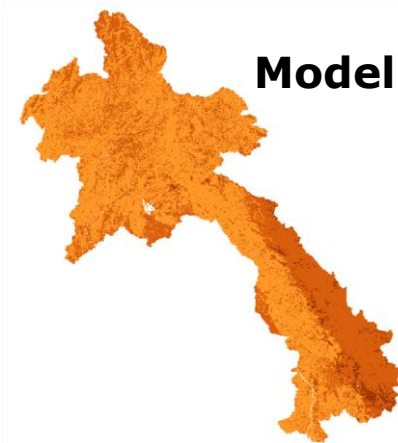
ID	SOC	X	Y
1	20.13	-04.64145	21.95636
2	2.31	-04.62672	21.94856
3	48.46	-04.71992	21.88411
4	38.75	-04.80064	21.86235
5	32.83	-04.82364	21.87175
6	23.43	-04.78833	21.83644
7	25.59	-04.11630	22.20583
8	44.77	-03.98344	22.16227
9	44.19	-03.94178	22.16393
10	28.10	-03.93400	22.16317
11	35.46	-03.92435	22.16515
12	36.61	-03.94355	22.15577

Table with XY coordinates



Point sample data

- 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

# terra package

To familiarize with handling spatial data in R, we will focus on:

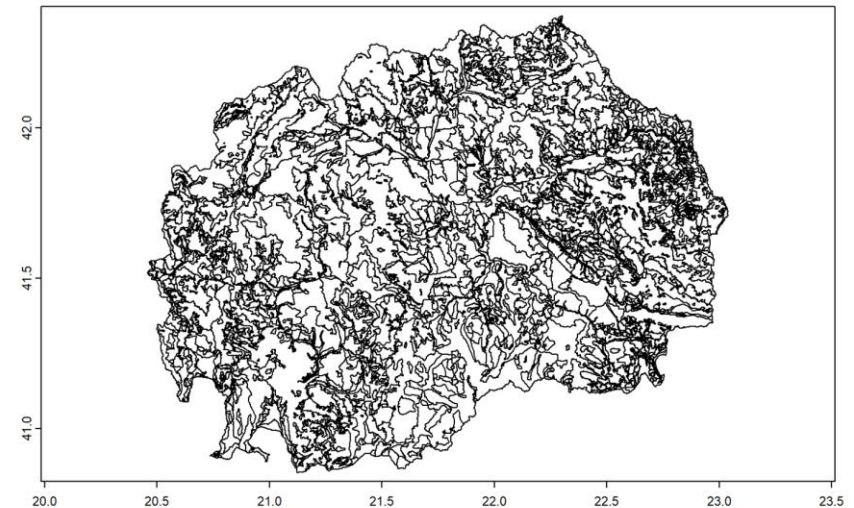
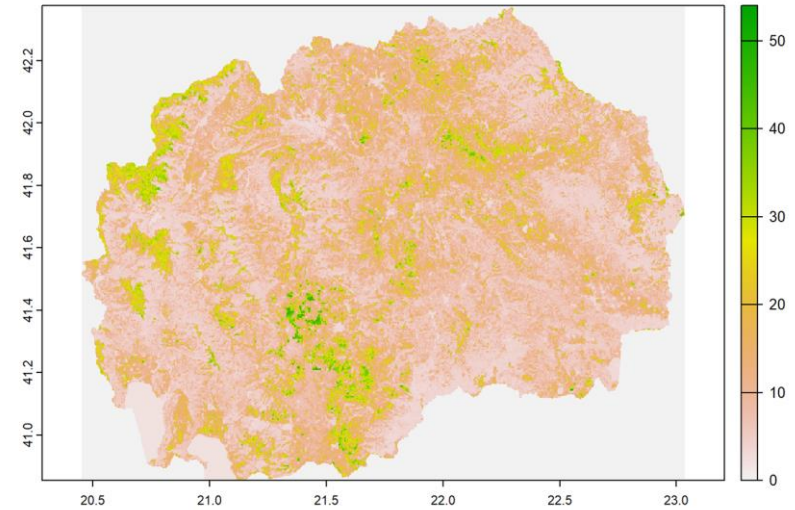
- Load a raster with `rast()` and explore its attributes
- Load a vector with `vect()` and explore its attributes
- Transform their coordinate system (`project()`)
- Cropping (`crop()`) and masking (`mask()`) a raster
- Replace values in a raster by filtering their cells
- Rasterize (`rasterize()`) a vector layer
- Extracting raster values (`extract()`) using points
- Zonal statistics (`zonal()`) using polygons and rasters



# terra: rast() & vect()

## Exercise

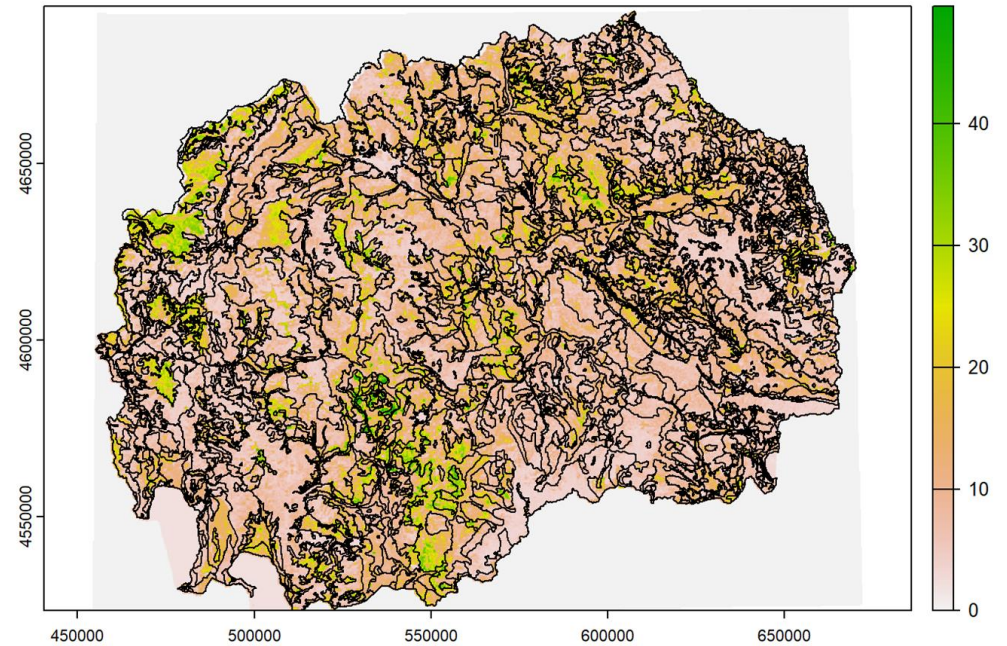
- Load 01-Data/covs/grass.tif using rast() function, then plot it
- Load 01-Data/soil map/SoilTypes.shp using vect() function and plot it
- Explore the attributes of these layers



# terra: project()

## Exercise

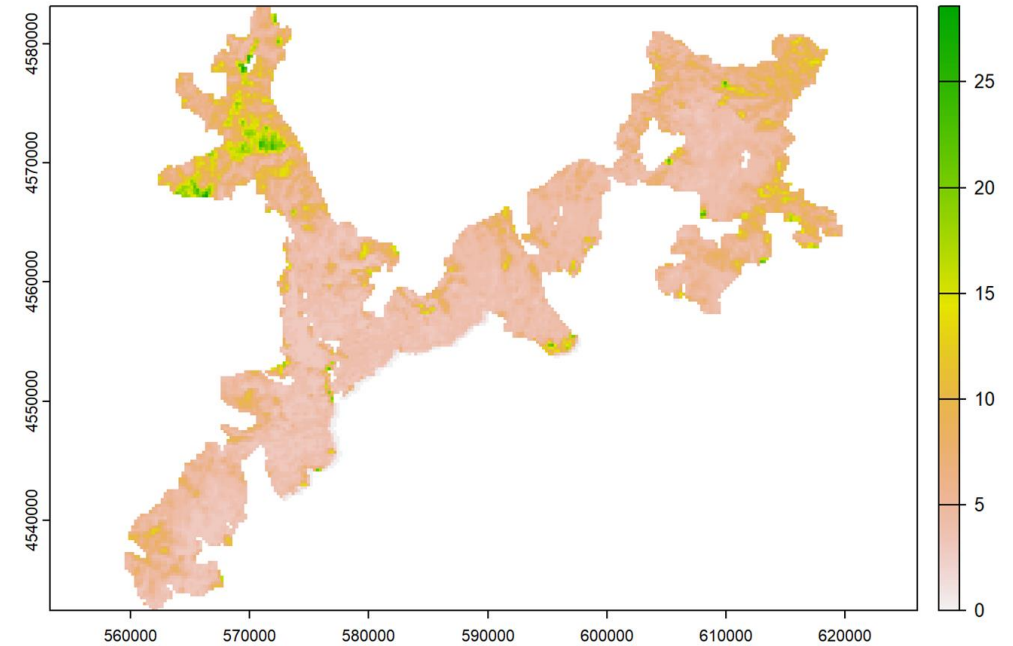
- Check the current CRS (EPSG) of the raster and the vector.
- Find a \*projected\* CRS in <http://epsg.io> for Macedonia and copy the number
- Check the Arguments of function project (?project) that need to be defined
- Save the new object as r\_proj and v\_proj
- plot both objects (used add=TRUE in plot function)



# terra: crop() and mask()

## Exercise

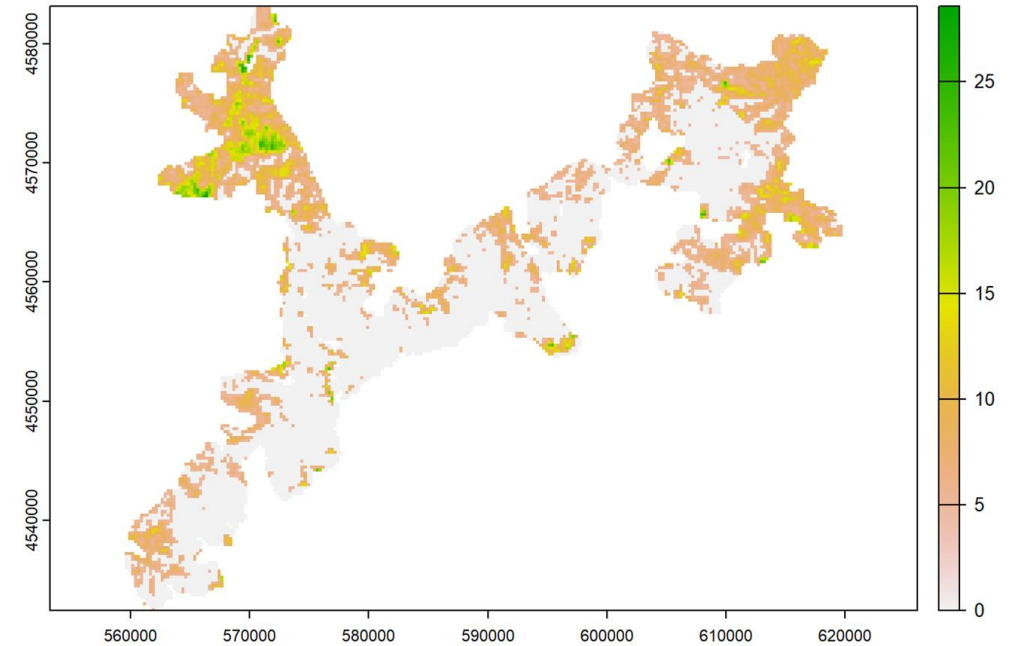
- Compute the area of the polygons in `v_proj` (search for a function) and
- assign the values to a new column named `area`
- select the largest polygon using `[], $, ==` and `max()` func. and save it as `pol`
  - Use it as it were a dataframe, for example `df[df$col == max(df$col)]`
- crop the raster with `pol` using the `crop()` function and save it as `r_pol`
- mask the raster `r_pol` with the polygon `pol` and save it with the same name
- Plot each result



# terra: replace cell values

## Exercise

- Explore the following link to understand how terra manage cell values
- <https://rspatial.org/terra/pkg/4-algebra.html>
- Replace values lower than 5 in r+pol by 0

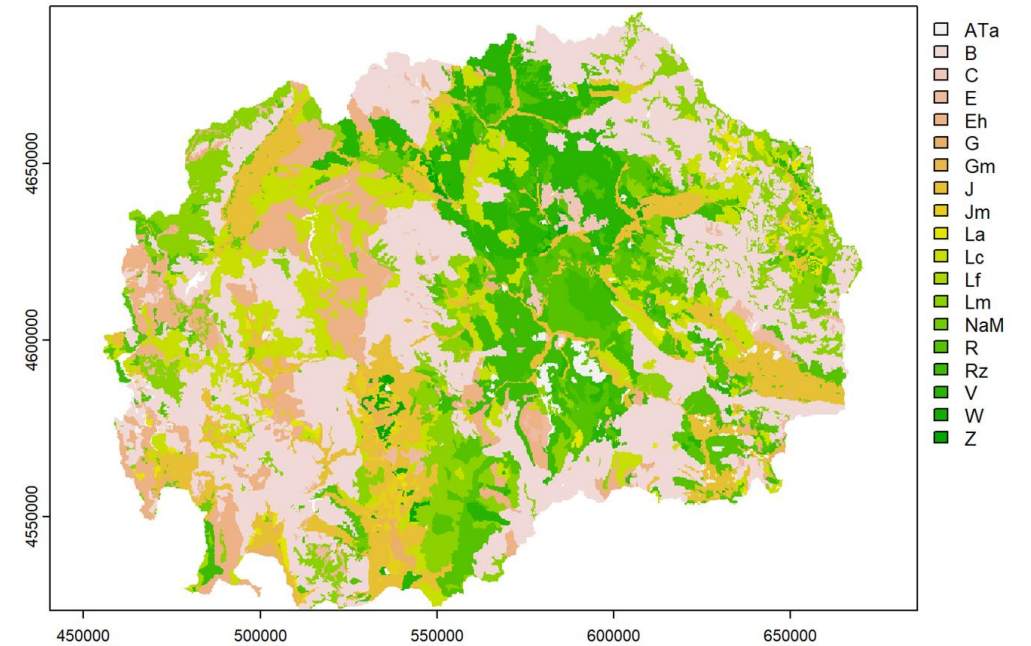




# terra: rasterize()

## Exercise

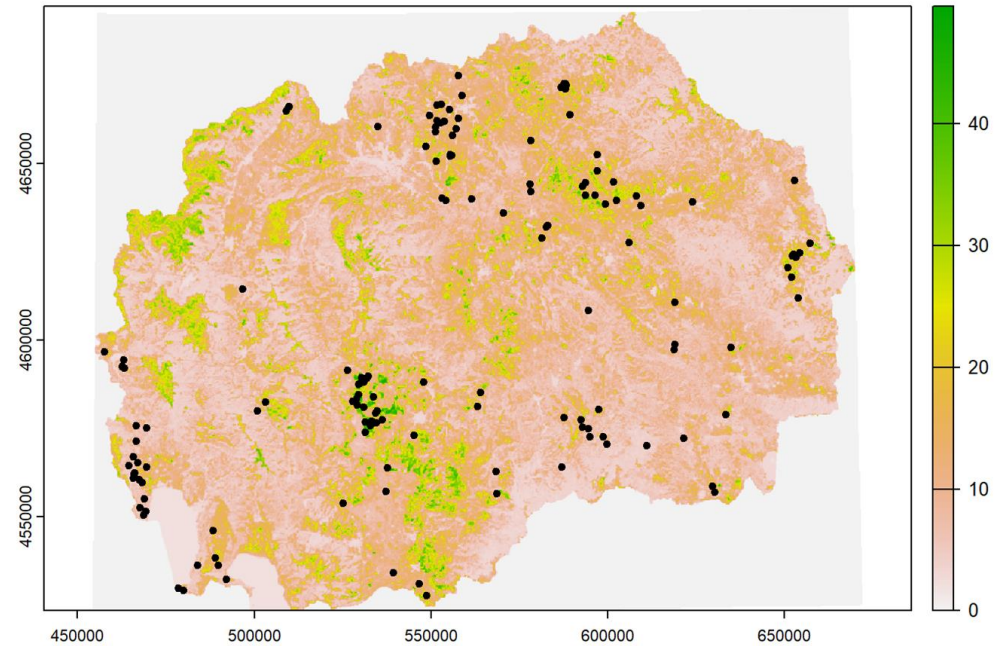
- Use rasterize() function to convert v\_proj to raster
- Use r\_proj as reference raster
- Use field Symbol to assign cell values, and plot the new map



# terra: extract()

## Exercise

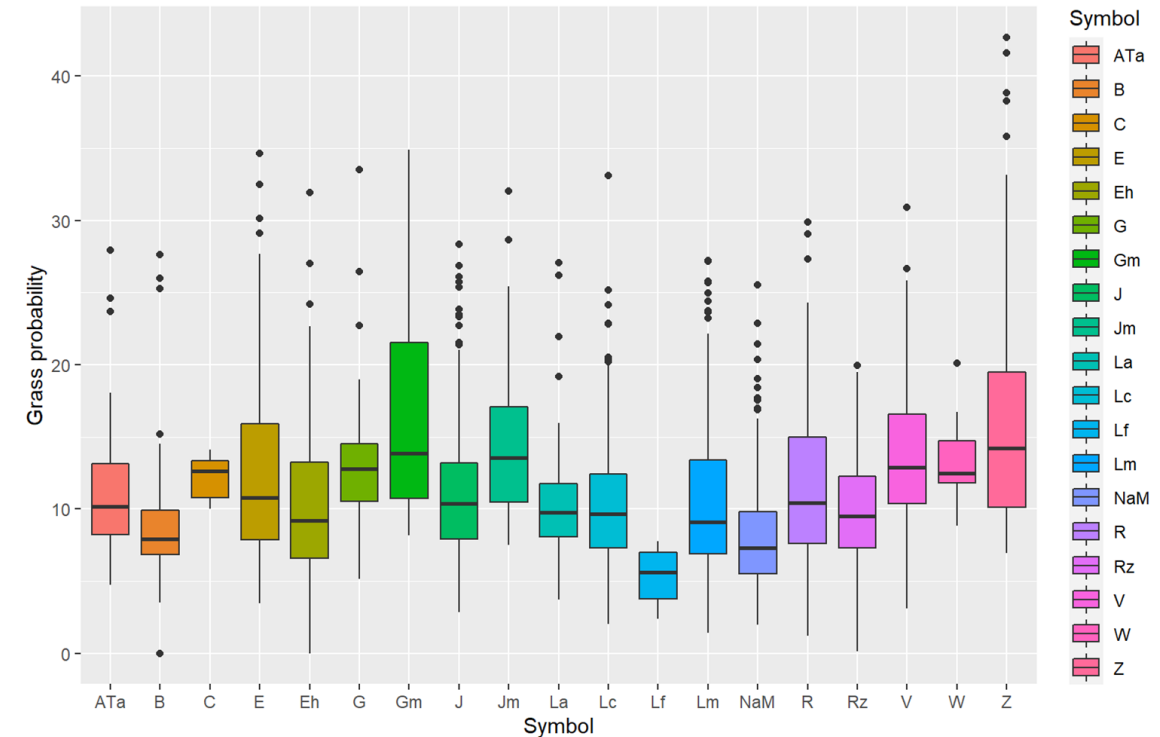
- Convert `dat_6` to spatial points using `vect()` function (check help of `vect()`)
- Note that the EPSG number is 6204
- Save the points as `s`
- Plot `s` and `r_proj` together in the same map (Argument `add=TRUE`)
- Extract the values of the raster using `extract()` function (check the help)
- Remove the ID column of the extracted values
- Merge the extracted data with `s` using `cbind()` function
- Convert `s` as a dataframe



# terra: zonal statistics

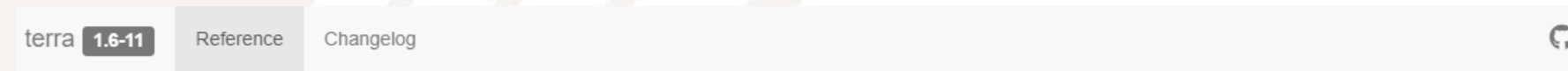
## Exercise

- Use the `extract()` func. to estimate the mean value of `r_proj` at each polygon
- Use the `fun=` argument (check the help)
- Use the `cbind()` func. to merge `v_proj` and the extracted values
- Convert `v_proj` to a dataframe
- Create a ggplot boxplot (`geom_boxplot`) with `x=Symbol` and `y=grass`



# terra package

<https://rspatial.github.io/terra/reference/terra-package.html>



## Description of the methods in the terra package

`terra` provides methods to manipulate geographic (spatial) data in "raster" and "vector" form. Raster data divide space into rectangular cells (pixels) and they are commonly used to represent spatially continuous phenomena, such as elevation or the weather. Satellite images also have this data structure. In contrast, "vector" spatial data (points, lines, polygons) are typically used to represent discrete spatial entities, such as a road, country, or bus stop.

### Contents

SpatRaster

I. Creating, combining and sub-setting

II. Changing the spatial extent or

<https://rspatial.org/terra/index.html>

A screenshot of the 'Spatial Data Science' documentation website. The page has a yellow header with the title 'Spatial Data Science' and a search bar. A dark sidebar on the left lists navigation items: 'Spatial data with terra', 'Spatial data analysis', 'Remote Sensing with terra', and 'Processing MODIS data'. The main content area has a breadcrumb trail 'Docs » Spatial Data Science with R and "terra"', followed by the title 'Spatial Data Science with R and "terra"'. Below the title is a paragraph of introductory text. In the bottom right corner, there is a logo for 'orm' and the 'GLOBAL SOIL PARTNERSHIP' logo.

## Spatial Data Science

Search docs

Spatial data with *terra*

Spatial data analysis

Remote Sensing with *terra*

Processing MODIS data

Docs » Spatial Data Science with R and "terra"

## Spatial Data Science with R and "terra"

These resources teach spatial data analysis and modeling with *R*. *R* is a widely used programming language and software environment for data science. *R* also provides unparalleled opportunities for analyzing spatial data and for spatial modeling.







# EduSoils

