Technical note
Processing the Harmonized World Soil Database (Version 2.0) in R

D G Rossiter
Guest Researcher
ISRIC-World Soil Information, Wageningen (NL)

Visiting Scientist / 客座教授
Institute of Soil Science, Chinese Academy of Sciences
中国科学院南京土壤研究所

June 4, 2023

Contents

1 Introduction

This note explains how to access and query the Harmonized World Soil Database (HWSD) version 2.0 [1] using the open-source R project for statistical computing [2]. This allows integration of the HWSD with any other geographic coverage, as well as statistical summaries.

This note explains the following procedures:

1. Accessing the HWSD v2.0 and importing it to R;
2. Selecting a geographic window from the HWSD v2.0, either by a rectangular bounding box or boundary polygons;
3. Projecting from the original Plate Carrée (non)projection to the UTM coordinate reference system;
4. Determining the area covered by each soil class;
5. Saving the window in the native HWSD v2.0 format and as a projected raster;
6. Linking the attribute database to the raster and saving the records for the window as a CSV file;
7. Converting from the original raster format to polygons;
8. Creating and displaying attribute and class raster and polygon maps.

There is certainly more that can be done in R with the HWSD, including integration with other freely-available geographic layers such as digital elevation models and satellite imagery.

The only operation that is not carried out in R is directly working with MS-Access databases (file extension `.mdb`), which is the format in which the HWSD attributes are supplied. Because MS-Access database access from R is only possible[1] from MS-Windows, in this tutorial we work with another database format, SQL databases. These are explained in §4.

> **Note:** In MS-Windows with Office 365 including MS-Access, it is possible to connect to the HWSD 2 database by using the `RODBC` "R interface with Open Database Connectivity" package[2]. The `RODBC` package has a different set of database functions than the `RSQLite` package used in this tutorial. See [3, §4] for a discussion of R and relational databases.

I have exported the MS-Access database (91.6 Mb) to SQLite format (66.8 Mb) using the MDB ACCDB Viewer program[3] on macOS. This SQLite database is provided as a download with this tutorial.

The procedures in this note use several R packages, including `sf` for spatial data, spatial data import, export and geometric transformation, `terra` for working with large raster (grid) image, `RSQLite` for working with the SQLite

[1] without quite some manipulation
[2] http://cran.r-project.org/web/packages/RODBC/index.html
[3] http://eggerapps.at/mdbviewer/

format relational databases, and DBI for the database interface commands. These must loaded before their first use, as is shown in the code.

> **Note:** The code in this document was tested with R version 4.2.3 (2023-03-15) and packages from that version or later running on Mac OS X 12.6.3. The text and graphical output you see here was written as a NoWeb file, including both R code and regular LaTeX source, and then run through the excellent **knitr** package Version: 1.42 [4] on R to and automatically generated and incorporated into LaTeX. Then the LaTeX document was compiled into the PDF version you are now reading, using the **knit** function. The R code (file R_HWSD2.R, supplied with this document) was also generated by **knitr** from the same source document, using the **purl** function. If you run this R code, or copy code from this document, your output may be slightly different on different versions and on different platforms.

2 Importing the HWSD into R

Task 1 : Download the HWSD v2.0 database. •

The HWSD is found at the FAO's Global Agroecological Zoning (GAEZ) page[4]. We do not use the HSWD Viewer on-line application, instead, we download the database and grid for use in R. Two compressed files are provided (Table 1):

You will not work directly with the MS Access database, rather, the SQLite conversion that is provided with these notes.

Task 2 : Uncompress the raster image HWSD2_raster.zip. •

This will create a subdirectory HWSD2_RASTER with four files: the band-interleaved image (HWSD2.bil), a small file giving the extent and resolution (HWSD2.stx), the projection information (HWSD2.prj), , and the header (HWSD2.hdr). The latter two are automatically consulted on data import. This raster file is quite large, 1.74 Tb.

```
round(file.size("./HWSD2_RASTER/hwsd2.bil")/1024^3,2)

[1] 1.74
```

[4] https://gaez.fao.org/pages/hwsd

file name	contents	format	size
HWSD2_raster.zip	Raster soil unit map	band-interleaved image (.bil, .blw, .hdr)	22.4 Mb
HWSD2_DB.zip	Soil attribute database	MS Access (.mdb)	9.3 Mb

Table 1: HWSD v2 compressed files

```r
file.mtime("./HWSD2_RASTER/hwsd2.bil")

[1] "2022-11-12 01:36:00 CET"
```

Task 3 : Import the HWSD v2 raster image to R. •

The `terra` package can work with very large images, such as this one, because it only reads the image into memory as necessary, otherwise keeping the image on disk. The `rast` function `terra` package associates an R object name with the file on disk. The band-interleaved format is known to this command.

The `terra` package depends on functions in the `sf` package. We load these two packages with the `require` function, and then set up a pointer to the raster, as an R object.

```r
require(sf)
require(terra)
hwsd <- rast("./HWSD2_RASTER/hwsd2.bil")
print(hwsd)

class       : SpatRaster
dimensions  : 21600, 43200, 1  (nrow, ncol, nlyr)
resolution  : 0.008333333, 0.008333333  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84
source      : HWSD2.bil
name        : HWSD2
min value   :     2
max value   : 49830
```

Task 4 : Examine the raster image's properties. •

The `terra` and `sf` packages provide some useful commands for this, which are self-explanatory:

```r
class(hwsd); ncol(hwsd); nrow(hwsd); ncell(hwsd)

[1] "SpatRaster"
attr(,"package")
[1] "terra"
[1] 43200
[1] 21600
[1] 933120000

res(hwsd); ext(hwsd)

[1] 0.008333333 0.008333333
SpatExtent : -180, 180, -90, 90 (xmin, xmax, ymin, ymax)

st_crs(hwsd)$proj4string

[1] "+proj=longlat +datum=WGS84 +no_defs"
```

This raster is in a Plate Carrée[5] projection using the WGS84 datum. This projection maps latitude and longitude directly to a grid cell, so that the figure is increasingly distorted towards the poles.

[5] French: "square plate"

3 Selecting a region

The entire database is very large; usually we want to work in some region. We can subset to a region either by a a bounding box (§3.1) or by a bounding polygon (§3.2).

3.1 Selecting by a bounding box

The `terra` package can crop an image to a rectangular extent. This extent can be extracted from the bounding box of any `sf` object, or directly specified using the `ext` function. Here we will select a 1° by 1° tile centred near Nanjing 南京, Jiangsu 江苏, People's Republic of China 中国人民共和国, and covering parts of Jiangsu 江苏省 and Anhui 安徽省 provinces. This same procedure can be used to select any tile of interest. We then crop to this extent with the `crop` function.

```
hwsd.zhnj <- crop(hwsd, ext(c(118, 119, 31.5, 32.5)))
nrow(hwsd.zhnj); ncol(hwsd.zhnj); st_bbox(hwsd.zhnj)

[1] 120
[1] 120
 xmin  ymin  xmax  ymax
118.0  31.5 119.0  32.5
```

The `unique` function shows the unique values for each band in a raster; here we only have one band.

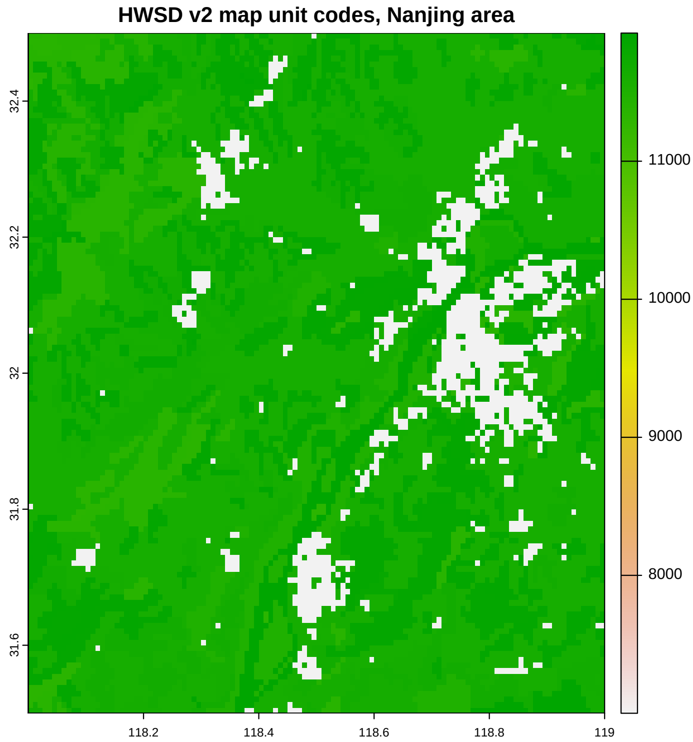
```
dim(unique(hwsd.zhnj))

[1] 58  1

head(unique(hwsd.zhnj)[1])

  HWSD2
1  7001
2 11341
3 11365
4 11367
5 11368
6 11372
```

This is the only content of the raster database: each pixel has a code, which links to the attribute database, see below.

Task 5 : Display this tile, with the map unit code of each pixel. •

There are too many classes to show with distinct colours. One way is to use a continuous colour ramp. However, the first-listed class (7001) is numerically much less than the others (in the 11000's) and will compress the colour ramp for these:

```r
plot(hwsd.zhnj, main = "HWSD v2 map unit codes, Nanjing area")
```

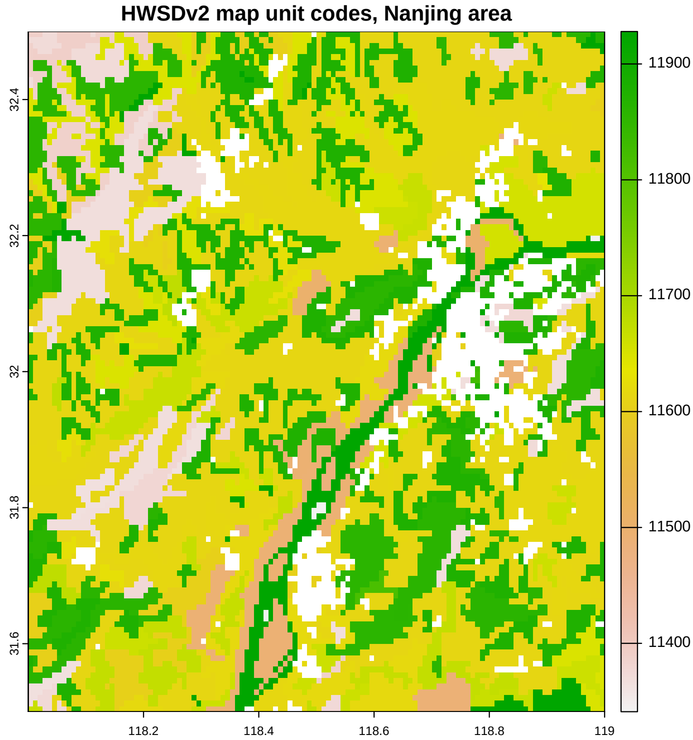


We see that the 7001 class (dark green in the image) must be urban areas, for which we do not have soil information (unfortunately).

Task 6 : Replace code 7001 with `NA` and re-draw the map. •

This is easily done with the `subst` function of the `terra` package:

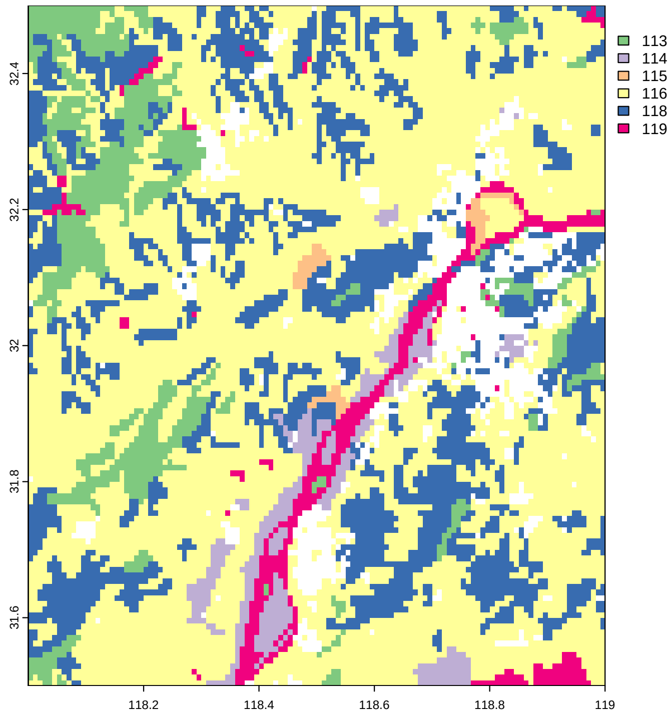
```r
hwsd.zhnj <- subst(hwsd.zhnj, 7001, NA)
head(unique(hwsd.zhnj)[1])

  HWSD2
1 11341
2 11365
3 11367
4 11368
5 11372
6 11373

plot(hwsd.zhnj, main = "HWSDv2 map unit codes, Nanjing area")
```



This looks good, since the codes appear to be ordered by similar soils.

We can also use just the first three digits of the map unit codes, which presumably are also a meaningful grouping; to remove the 'hundreds' places we use the `%/%` "integer divide" operator . The `RColorBrewer` package provides colour palettes; here we select one (named "Accent") that emphasizes differences between classes; we select it with the `brewer.pal` function: The `freq` function of the `terra` package computes a frequency table of the unique values in the raster image.

```
hwsd.zhnj3 <- (hwsd.zhnj%/%100)
freq(hwsd.zhnj3)

  layer value count
1     1   113  1192
2     1   114   516
3     1   115    86
4     1   116  7855
5     1   118  3124
6     1   119   521

require(RColorBrewer)
plot(hwsd.zhnj3,
     col=brewer.pal(n = dim(unique(hwsd.zhnj3))[1],"Accent"))
```



This image is distorted, i.e., distances E-W are not equal to those N-S, because it is in geographic coordinates and not projected to metric coordinates with approximately equal distances on both axes. To correct this, the image can be reprojected to a suitable metric coordinate system. For a small areas such as this, including any N-S oriented area less than 3° in the E-W dimension, a reasonable choice is the Universal Transmercator (UTM) coordinate reference system (CRS). For other areas, consult the EPSG database[6] for an appropriate CRS.

Task 7 : Project this image to the Universal Transmercator (UTM) coordinate reference system (CRS). •

We can see the effect of projection, using the `project` method and specifying a target (CRS, using the EPSG reference codes for the CRS.

We first determine the appropriate UTM zone for the centre of the window, recalling that UTM zone 30 is centred on 3° E, and that the EPSG codes for UTM northern-hemisphere zones on the WGS84 datum begin with 32601[7].

```
print(paste("UTM zone:", utm.zone <-
            floor(((st_bbox(hwsd.zhnj3)$xmin +
                    st_bbox(hwsd.zhnj3)$xmax)/2 + 180)/6)
            + 1))

[1] "UTM zone: 50"
```

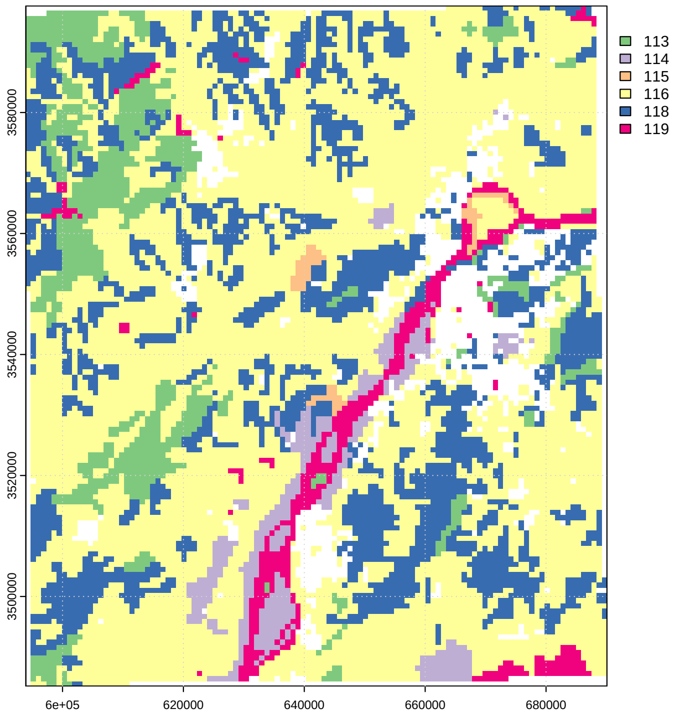
[6] epsg.io

[7] https://epsg.org/; "The IOGP's EPSG Geodetic Parameter Dataset is a collection of definitions of coordinate reference systems and coordinate transformations ... [it] is maintained by the Geodesy Subcommittee of the IOGP Geomatics Committee."

```
(epsg <- 32600 + utm.zone)

 xmin
32650
```

We then project to this CRS. Note that we use the nearest-neighbour re-sampling (`method="near"`) since this is a map of classes, not properties.

```
hwsd.zhnj3.utm <- project(hwsd.zhnj3, paste0("EPSG:", epsg), method = "near")
(cell.dim <- res(hwsd.zhnj3.utm))

[1] 858.3385 858.3385

plot(hwsd.zhnj3.utm,
     col=brewer.pal(n = dim(unique(hwsd.zhnj3))[1],"Accent"),
     type = "classes"); grid()
```



Notice how the region is now shorter in the E–W direction, by the cosine of the N latitude, here about $\cos(32°) = 0.84805$. Also, notice the region is slightly angled with respect to UTM north; this is because the region is not centred on the meridian of zone 50 (117° E = UTM 500 000 E) and the UTM projection is equal-angle but not equal area, becoming most distorted at the edge of the 6° zone.

Task 8 : Compute the area covered by each code, and the total area of the tile, here in km^2. •

This is now possible because the raster had been projected into a metric system.

```
(cell.area <- cell.dim[1]*cell.dim[2]/10^4)
```

```
[1] 73.6745

print(freq(hwsd.zhnj3.utm))

  layer value count
1     1   113  1185
2     1   114   540
3     1   115    77
4     1   116  7778
5     1   118  3050
6     1   119   520

(total.area <- sum(freq(hwsd.zhnj3.utm)[,"count"]*cell.area/10^2))

[1] 9688.197
```

The area of a grid cell is about 73.7 ha; at the equator this would be about 100 ha (1 km^2). The tile covers about 9688.2 km^2. Notice also that there are some `NA` cells; these are the ones at the edges of the projected image, needed to keep the raster square.

We are done with the generalized map, so remove it and some temporary calculations. Keep the EPSG code for this UTM zone, it will be used for polygon maps, below.

```
rm(hwsd.zhnj3.utm)
rm(utm.zone, cell.dim, cell.area, total.area)
```

Task 9 : Query the image at a location. •

Back to the unprojected image with the five-digit class codes, we can query at any location with the `click` function. When this is called, click with the mouse at a cell in the displayed image; this will return the coordinates of the point, and, if the optional argument `click` is set to `TRUE`, the code at the raster cell is returned.

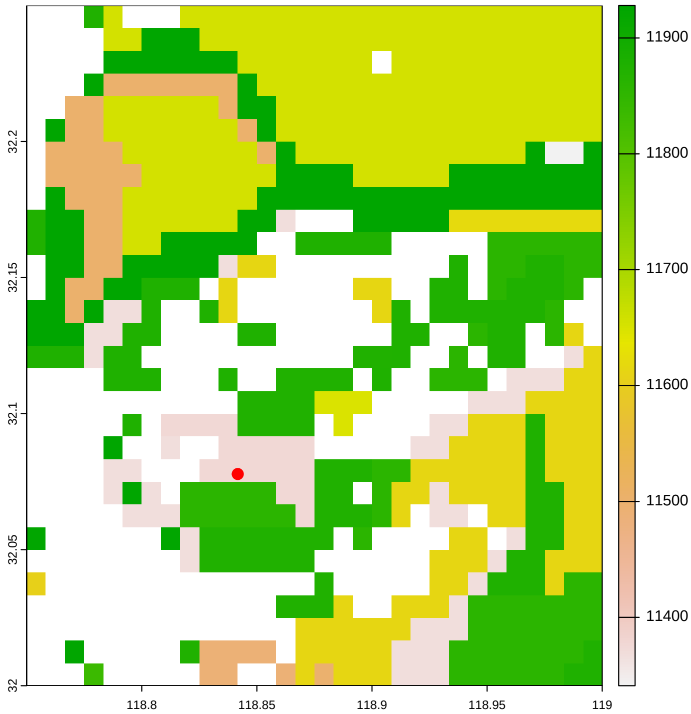
For example, clicking on the approximate peak of the Purple Mountain 紫金山 to the east of downtown Nanjing (118° 50' 30" E, 32° 04' 40" N according to Google Earth). The `click` function has optional arguments, which we use, to return the raster attribute value.

This location in decimal degrees (as shown on the plot):

```
print(paste("longitude:" , x <- round(118 + 50/60 + 30/(60^2), 4)))

[1] "longitude: 118.8417"

print(paste("latitude:" , y <- round(32 + 4/60 + 40/(60^2), 4)))

[1] "latitude: 32.0778"
```

To make the location easier to find on the plot, we temporarily crop the image with an extent, using the `ext` "extent" function of the `terra` package.

```
# zoom in to find the right pixel for sure
tmp <- crop(hwsd.zhnj, ext(c(118.75, 119, 32, 32.25)))
plot(tmp)
terra::points(x, y, pch = 20, cex = 2, col = "red")
(xy <- click(tmp, n=1, id=TRUE, xy=TRUE, type="p"))
rm(tmp)
```




```
print(xy)

         x          y      HWSD2
  118.8417    32.0778 11376.0000

(zjs.id <- xy["HWSD2"])

HWSD2
11376

rm(x, y, xy)
```

The coordinates are in decimal degrees. The result is the soil map unit code of the pixel. Below (§4.1) we will see its WRB class by linking with the attribute database.

3.2 Selecting by a bounding polygon

Another way to select a subset of the database is with the polygon boundary of a region, e.g., a country. In this example, we'll use my home country; you can replace this with any of your choice.

Task 10 : Make a `SpatVector` "spatial vector" object from the boundary of the Kingdom of the Netherlands. •

We obtain the boundaries of Netherlands from the `worldHires` dataset of the `mapdata` package, which was created from what the authors call a "cleaned-up" version of the CIA World Data Bank II data of 2003[8]. We extract the

[8] http://www.evl.uic.edu/pape/data/WDB/

boundary with the `map` function, and then convert it to an `sf` "simple features vector" object with the `st_as_sf` function of the `sf` "Simple Features" package, also specifying the CRS, which is given by the documentation of the data source.

> **Note:** The `fill` argument to the `map` function converts the boundary coordinates into a polygon by joining the last and first points.

```
require(maps)
require(mapdata)
str(tmp <- map('world', 'Netherlands', fill = TRUE, plot = FALSE))

List of 4
 $ x    : num [1:291] 4.23 4.21 4.17 4.04 3.9 ...
 $ y    : num [1:291] 51.4 51.3 51.3 51.2 51.2 ...
 $ range: num [1:4] 3.35 7.2 50.75 53.51
 $ names: chr [1:10] "Netherlands:South" "Netherlands:Schouwen-Duiveland" "Netherlands:Texel" "Netherl
 - attr(*, "class")= chr "map"

nl.bound <- st_as_sf(tmp, ID = "Netherlands",
                     crs = "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")
class(nl.bound)

[1] "sf"         "data.frame"

(ext(nl.bound))

SpatExtent : 3.35009741783142, 7.197265625, 50.7504386901855, 53.5149879455566 (xmin, xmax, ymin, ymax

summary(nl.bound)

      ID                        geom
 Length:1           MULTIPOLYGON :1
 Class :character   epsg:NA      :0
 Mode  :character   +proj=long...:0

rm(tmp)
```

Task 11 : Convert this boundary to a `SpatVector` "spatial vector" polygon object. •

This boundary (currently a `MULTIPOLYGON` geometry in `sf`) can be converted to a `SpatVector` "spatial vector" polygon, using the `vect` function of the `terra` package, followed by the `as.polygons` function of the `terra` package.

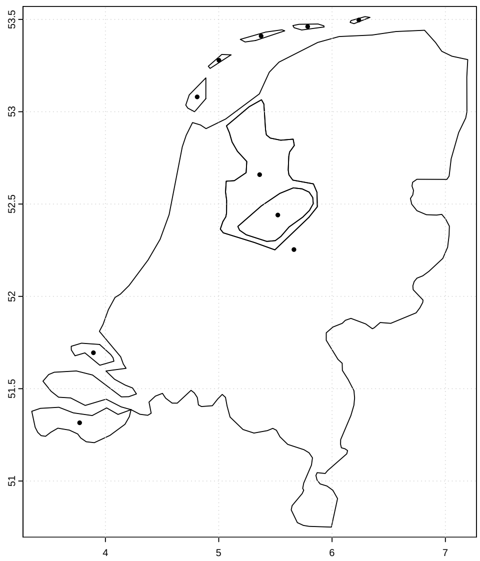
```
nl.poly <- as.polygons(vect(nl.bound))
print(nl.poly)

 class       : SpatVector
 geometry    : polygons
 dimensions  : 10, 1  (geometries, attributes)
 extent      : 3.350097, 7.197266, 50.75044, 53.51499  (xmin, xmax, ymin, ymax)
 coord. ref. : +proj=longlat +ellps=clrk66 +no_defs
 names       :          ID
 type        :       <chr>
 values      : Netherlands


nrow(nl.poly)

[1] 10
```

This is a vector object with 10 polygons. None of these are named.

Task 12 : Plot this polygon with its centroids. •

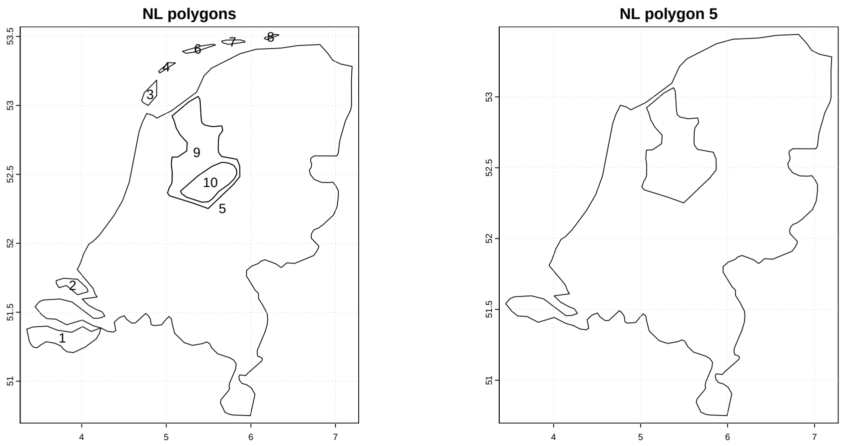
The `centroids` function extracts the centroids of each polygon.

```
plot(nl.poly); grid()
points(centroids(nl.poly))
```



Note: This multipolygon can be disaggregated into its components, using the `disagg` function, in case you want to work with a smaller area:

```
par(mfrow=c(1,2))
nl.polys <- disagg(nl.poly)
plot(nl.polys, main = "NL polygons"); grid(); text(centroids(nl.polys))
plot(nl.polys[5], main = "NL polygon 5"); grid()
par(mfrow=c(1,1))
```



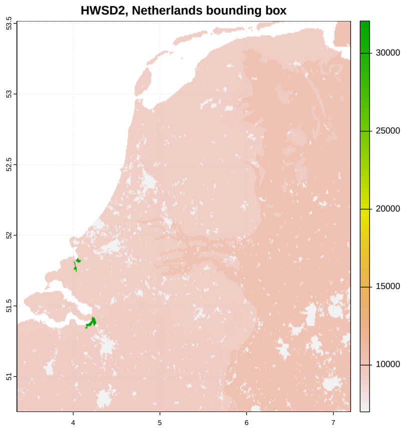
Task 13 : Extract the portion of the HWSD database within this polygon as a raster window. •

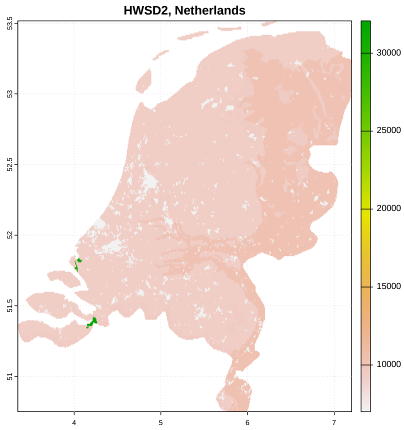
Working with a `SpatRaster` object (i.e., the raster map) we can only extract rectangular areas. So first we use the bounding box of the Netherlands to extract it and some areas of neighbouring countries, using the `crop` function on the bounding box's extent, extracted with the `ext` function. We then remove the areas outside Netherlands with the `mask` function, whereby areas outside the bounding polygon will be assigned a value of `NA`.

```
hwsd.nl.box <- crop(hwsd, ext(nl.poly))
hwsd.nl <- mask(hwsd.nl.box, nl.poly)
class(hwsd.nl)

[1] "SpatRaster"
attr(,"package")
[1] "terra"

(nl.id <- sort(unique(values(hwsd.nl))))

 [1]  7001  7003  9345  9346  9347  9348  9349  9350  9351  9352
[11]  9353  9354  9355  9356  9357  9358  9359  9360  9361  9362
[21]  9363  9364  9365  9367  9373  9375  9382  9386  9387  9388
[31] 10184 10186 10187 10189 10215 10218 10241 10242 10243 10244
[41] 10245 10246 10248 10249 10250 10251 32068 32069
```

```
plot(hwsd.nl.box,
                main="HWSD2, Netherlands bou
                grid()
```

```
plot(hwsd.nl,
        main="HWSD2, Netherlands"); grid()
```





There are 48 soil map units in the Netherlands. Because the codes have such a wide range, and only a small area have the very large codes ($\approx$ 32000), it's difficult to see the map units here. In this map we can only see their codes, which are links into the attribute database, which we now examine.

3.3 Saving raster images

Extracted windows or countries can be written to disk for later use in R, or for export to other programs, with the `writeRaster` function. For example, to write the Nanjing and Netherlands rasters as an "ESRI `.hdr` Labelled" formatted raster:

```
terra::writeRaster(hwsd.zhnj, file='./HWSD_zhnj', filetype = 'EHdr', overwrite = TRUE)
terra::writeRaster(hwsd.nl, file='./HWSD_nl', filetype = 'EHdr', overwrite = TRUE)
```

Raster maps of class `SpatRaster` can be saved for later use in a variety of vector GIS formats; see the GDAL drivers list[9]. To see the drivers available

[9] https://gdal.org/drivers/raster/index.html

on your system, use the `gdal` command with the `drivers` argument set to `TRUE`:

```
g.drivers <- gdal(drivers = TRUE)
head(r.drivers <- g.drivers[g.drivers$type == "raster", ], 12)

      name   type        can  vsi
1  AAIGrid raster read/write TRUE
2     ACE2 raster       read TRUE
3     ADRG raster read/write TRUE
4      AIG raster       read TRUE
5      ARG raster read/write TRUE
6   AirSAR raster       read TRUE
7      BAG raster read/write TRUE
8   BIGGIF raster       read TRUE
9      BLX raster read/write TRUE
10     BMP raster read/write TRUE
11     BSB raster       read TRUE
12      BT raster read/write TRUE
                            long.name
1                 Arc/Info ASCII Grid
2                                ACE2
3        ARC Digitized Raster Graphics
4                Arc/Info Binary Grid
5            Azavea Raster Grid format
6            AirSAR Polarimetric Image
7           Bathymetry Attributed Grid
8    Graphics Interchange Format (.gif)
9                Magellan topo (.blx)
10 MS Windows Device Independent Bitmap
11          Maptech BSB Nautical Charts
12  VTP .bt (Binary Terrain) 1.3 Format
```

You can search for a file format by part of the long name, or by the GDAL code, using the `grep` "General Regular Expression" function:

```
r.drivers[grep("ESRI", r.drivers$long.name, fixed = TRUE),]

   name   type        can  vsi          long.name
32 EHdr raster read/write TRUE ESRI .hdr Labelled

r.drivers[grep("EHdr", r.drivers$name, fixed = TRUE),]

   name   type        can  vsi          long.name
32 EHdr raster read/write TRUE ESRI .hdr Labelled
```

4 Attribute database

As explained in the introduction, in this tutorial we acces the HWSD v2 database as an SQL database, using the `RSQLite` package. This package provides the interface to an SQL database via the `DBI` package. I have exported the Access databaset to SQLite format[10], with file name `HWSD2.sqlite`; this is provided with the tutorial.

SQLite is a dialect of the SQL "Structured Query Language". The complete reference to its syntax is at the SQLite webpage[11]. There are many tutorials on-line, e.g., from W3Schools[12]; however here we will only use a small subset of SQL operations, since we already have all the data in prepared tables.

[10] using the *MDB ASCCDB Viewer* program on macOS
[11] https://www.sqlite.org/lang.html
[12] https://www.w3schools.com/sql/default.asp

Task 14 : Connect to the SQLite version of the HWSD v2 attribute databse and list the tables. •

We first load the RSQLite package with require; this automatically loads the DBI package if necessary. We then use the dbDriver function to specify the database driver to be used by DBI (in this case, SQLite), and then the dbConnect function with this driver and the name of the database on disk to set up a database *connection*; this variable in the R workspace then refers to the database and is used in every command which queries or manipulates it. The dbListTables function lists the relational tables in the database.

```
file.size("./HWSD2.sqlite")/1024^2

[1] 64.03516

file.mtime("./HWSD2.sqlite")

[1] "2023-06-04 12:18:01 CEST"

require(RSQLite)
m <- dbDriver("SQLite")
con <- dbConnect(m, dbname="HWSD2.sqlite")
dbListTables(con)

 [1] "D_ADD_PROP"              "D_AWC"
 [3] "D_COVERAGE"              "D_DRAINAGE"
 [5] "D_FAO90"                 "D_IL"
 [7] "D_KOPPEN"                "D_PHASE"
 [9] "D_ROOTS"                 "D_ROOT_DEPTH"
[11] "D_SWR"                   "D_TEXTURE"
[13] "D_TEXTURE_SOTER"         "D_TEXTURE_USDA"
[15] "D_WRB2"                  "D_WRB2code"
[17] "D_WRB4"                  "D_WRB_PHASES"
[19] "HWSD2_LAYERS"            "HWSD2_LAYERS_METADATA"
[21] "HWSD2_SMU"               "HWSD2_SMU_METADATA"
[23] "WINDOW_NL"               "WINDOW_TMP"
[25] "WRB_Class"               "WRB_Layer"
[27] "WRB_Library"
```

This database size is 64.04 Mb. It has 27 files, most of which are lookup tables of the attribute codes (prefix D_).

There are two main data tables: HWSD2_SMU for map units and HWSD2_LAYERS for the map unit components and their layers. The meaning of each field is conveniently given in the corresponding metadata tables HWSD2_SMU_METADATA and HWSD2_LAYERS_METADATA. The table WRB_CLASS lists the WRB Reference Groups and their preferred colour representation on class maps.

Task 15 : Display the structure of the map unit table. •

SQL syntax used in SQLite is explained, with syntax diagrams, at the SQLite web page[13]. This language is not immediately intuitive; the reader who is unfamiliar with it is encouraged to follow a tutorial[14] to understand its principles.

[13] http://www.sqlite.org/lang.html

[14] For example, http://www.w3schools.com/sql/

The `dbGetQuery` function requires a database connection and a query string in SQL format. It returns the result of a query as an R `data.frame`. SQL uses the `PRAGMA` command to display database structure; we include it in the query string.

> **Note:** Unlike R, SQL is not case-sensitive, so the command strings can be upper, lower, or mixed case. By convention I use upper-case for database names and lower-case for SQL commands.

```
print(data.frame(
    name = dbGetQuery(con, "pragma table_info(HWSD2_SMU)")$name,
    type = dbGetQuery(con, "pragma table_info(HWSD2_SMU)")$type))

                name    type
1                 ID INTEGER
2       HWSD2_SMU_ID INTEGER
3     WISE30s_SMU_ID    TEXT
4       HWSD1_SMU_ID INTEGER
5           COVERAGE INTEGER
6              SHARE INTEGER
7               WRB4    TEXT
8         WRB_PHASES    TEXT
9               WRB2    TEXT
10         WRB2_CODE INTEGER
11             FAO90    TEXT
12            KOPPEN    TEXT
13      TEXTURE_USDA INTEGER
14 REF_BULK_DENSITY  DOUBLE
15      BULK_DENSITY  DOUBLE
16          DRAINAGE    TEXT
17        ROOT_DEPTH INTEGER
18               AWC INTEGER
19            PHASE1 INTEGER
20            PHASE2 INTEGER
21             ROOTS INTEGER
22                IL INTEGER
23          ADD_PROP INTEGER
```

The field names and linked lookup tables are shown in the associated metadata table.

Task 16 : Display the metadata for the map unit table. •

The `select` SQL function selects records from a table. Here we want to see the whole table, so we specify the `*` placeholder.

```r
print(dbGetQuery(con, "select * from HWSD2_SMU_METADATA"), width=100)
```

```
   ID            FIELD  UNIT                          DESCRIPTION DATATYPE         DOMAIN
1   1               ID  <NA>                 Database internal ID   Number           <NA>
2   2      HWSD2_SMU_ID  <NA>              Soil Mapping Unit HWSD-2   Number           <NA>
3   3    WISE30s_SMU_ID  <NA>             Soil Mapping Unit WISE30s   String           <NA>
4   4      HWSD1_SMU_ID  <NA>              Soil Mapping Unit HWSD-1   Number           <NA>
5   5          COVERAGE  <NA>                              Coverage   Number     D_COVERAGE
6   6             SHARE     %              Share in Soil Mapping Unit   Number           <NA>
7   7              WRB4 Class             Soil Unit Symbol (WRB 2022)   String         D_WRB4
8   8        WRB_PHASES Class             Soil Unit Symbol (WRB'2022)   String   D_WRB_PHASES
9   9              WRB2 Class            Dominant Soil Group (WRB3)   String         D_WRB2
10 10        WRB2_CODE Class Dominant Soil Group (WRB  +PHASES)   String     D_WRB2code
11 11            FAO90 Class           Dominant Soil Group (FAO 90)   String        D_FAO90
12 12           KOPPEN Class                    Koppen-Geiger Class   String       D_KOPPEN
13 13      TEXTURE_USDA Class                        Topsoil Texture   Number D_TEXTURE_USDA
14 14 REF_BULK_DENSITY g/cm3                 Reference Bulk Density   Number           <NA>
15 15      BULK_DENSITY g/cm3                           Bulk Density   Number           <NA>
16 16         DRAINAGE Class                Reference Soil Drainage   Number     D_DRAINAGE
17 17        ROOT_DEPTH Class                          Rooting Depth   Number   D_ROOT_DEPTH
18 18              AWC  mm/m               Available Water Capacity   Number           <NA>
19 19           PHASE1 Class                                 PHASE1   Number        D_PHASE
20 20           PHASE2 Class                                 PHASE2   Number        D_PHASE
21 21            ROOTS Class               Obstacles to Roots (ESDB)   Number        D_ROOTS
22 22               IL Class             Impermeable Layer  (ESDB)   Number           D_IL
23 23         ADD_PROP Class                 Additional Properties   Number     D_ADD_PROP
```

The lookup tables are shown for the coded fields. For example, the Koppen-Geiger Class climate codes are linked to their names in table `D_KOPPEN`.

Task 17 : Show the Koppen-Geiger Class climate codes and their definitions. •

```r
dbGetQuery(con, "select * from D_KOPPEN")
```

```
  CODE     VALUE
1    A  Tropical
2    B      Arid
3    C Temperate
4    D      Cold
5    E     Polar
```

Task 18 : Determine the number of records in the main table. •

We use the `count` SQL function to count selected records (in this case, all of them, as symbolized by the `*`), and name the result with the `AS` SQL command. We select all records (by omitting a `where` clause).

```r
dbGetQuery(con, "select count(*) as grid_total from HWSD2_SMU")
```

```
  grid_total
1      29538
```

Task 19 : Display the ID, map unit code, the percent in map unit, the WRB 2022 and FAO 1990 class codes, the Koppen-Geiger climate zone, the rooting depth, and the topsoil texture codes, for the first ten records of the main database. •

An SQLite database is not guaranteed to have any particular ordering, so "the first" may vary by implementation. We use the `limit` SQL operator to limit the number of records returned, and specify the fields to return.

> **Note:** The `paste` function with the `collapse` argument collapses a character vector into a single string, with the elements separated by the argument to `paste`.

```
(display.fields <- c("ID","HWSD2_SMU_ID","SHARE",
                     "WRB4",
                     "KOPPEN",
                     "ROOT_DEPTH",
                     "TEXTURE_USDA"))

[1] "ID"           "HWSD2_SMU_ID" "SHARE"        "WRB4"
[5] "KOPPEN"       "ROOT_DEPTH"   "TEXTURE_USDA"

print(tmp <- dbGetQuery(con, paste("select", paste(display.fields, collapse=", "),
                             "from HWSD2_SMU limit 10")))

    ID HWSD2_SMU_ID SHARE WRB4 KOPPEN ROOT_DEPTH TEXTURE_USDA
1  669        12707    40 ALfr      A          1           11
2  695        11825   100 ALfr      C          1            7
3  696        11823   100 ALfr      C          1            7
4  697        13458   100 ALfr      C          1            7
5  698        11824   100 ALfr      C          1            7
6  713        17690   100 PTha      A          1           10
7  715        12680    80 PTha      A          1           10
8  718        12726    45 PTha      A          1           10
9  719        31602   100 PTha      A          1           10
10 720        12712    50 PTha      A          1           10
```

Task 20 : Display the structure of the lookup table for WRB 2022 soil classes, and convert this to an R `data.frame` for easier use within R. •

From the HWSD documentation we know that the lookup tables have names with pattern `D_*`; the table for FAO 1990 classes is `D_SYMBOL90`. Here we know the table is fairly small, so we read it into memory by selecting all rows; then we examine the structure.

```
str(d.wrb4 <- dbGetQuery(con, "select * from D_WRB4"))

'data.frame': 191 obs. of  3 variables:
 $ ID   : num  1 2 3 4 5 6 7 8 9 10 ...
 $ VALUE: chr  " Acrisols" "Ferric Acrisols" "Gleyic Acrisols" "Haplic Acrisols" ...
 $ CODE : chr  "AC" "ACfr" "ACgl" "ACha" ...
```

4.1 Attributes in the bounding box

Task 21 : Show the map unit record for the pixel identified in the previous section. •

Again we use the `dbGetQuery` function, but now with a query string to find the map unit's record. Note the use of the `paste` function to build a query string with some fixed text (in quotes) and some text taken from a variable, here the soil map unit code saved as variable `zjs.id` during the interactive map query, above.

We then use the lookup table to show the WRB4 name.

```
(tuple <- dbGetQuery(con, paste("select * from HWSD2_SMU where HWSD2_SMU_ID = ",
                                zjs.id)))

     ID HWSD2_SMU_ID WISE30s_SMU_ID HWSD1_SMU_ID COVERAGE SHARE
1 51253        11376     WD30011376        11376        2   100
  WRB4 WRB_PHASES WRB2 WRB2_CODE FAO90 KOPPEN TEXTURE_USDA
1 CMdy       CMdy   CM         8   CMd      C            9
  REF_BULK_DENSITY BULK_DENSITY DRAINAGE ROOT_DEPTH AWC PHASE1
1             1.71         1.25       MW          1 144     NA
  PHASE2 ROOTS IL ADD_PROP
1     NA    NA NA        0

(tuple$WRB4)

[1] "CMdy"

ix <- which(d.wrb4$CODE == tuple$WRB4)
print(paste("WRB 2022 class:", d.wrb4[ix, "VALUE"]))

[1] "WRB 2022 class: Dystric Cambisols"
```

We could also find this via an SQL query on the lookup table:

```
dbGetQuery(con, paste0("select VALUE from D_WRB4 where CODE='", tuple$WRB4,"'"))

              VALUE
1 Dystric Cambisols
```

Indeed, the soils of the Purple Mountain area are in general shallow and with low base saturation, so Dystric Cambisols is a reasonable classification.

Now we make a derived soil properties map in the raster window.

Task 22 : Extract a table of the map units in the raster window. •

One way to extract the appropriate records from the map unit database is to make a database table of the list of map units in the window, and then use this as a selection criterion with a JOIN. The `dbWriteTable` function creates a table; it requires an R data frame as the initial value. From this it infers the table structure.

```
dbWriteTable(con, name="WINDOW_ZHNJ",
             value=data.frame(hwsd2_smu = sort(unique(values(hwsd.zhnj)))),
             overwrite=TRUE)
dbGetQuery(con, "pragma table_info(WINDOW_ZHNJ)")

  cid      name type notnull dflt_value pk
1   0 hwsd2_smu REAL       0         NA  0

head(dbGetQuery(con, "select * from WINDOW_ZHNJ"))

  hwsd2_smu
1     11341
2     11365
3     11367
4     11368
5     11372
6     11373
```

Now we create a new table using the `create table` command, from a join on the common field, using the `join` command. The new table does not contribute any new fields. We also show how to sort the results, in this case by the map unit ID. We save an in-memory version of the table, as an R `data.frame`.

> **Note:** The `select T.*` clause selects the fields from the `HWSD2_SMU` table; this is represented by `T` in the `join` clause. We do not need the fields from the table with the list of map units in the window, since the `HWSD2_SMU` table has the same codes in field `WRB4`.

> **Note:** Here we use the `dbExecute` function, to execute an SQL command, formatted as a string. This command returns the number of rows in an existing table affected by the command, which in this case is zero.

```
dbExecute(con, "drop table if exists ZHNJ_SMU") # to overwrite

[1] 0

dbExecute(con,
         "create TABLE ZHNJ_SMU AS select T.* from HWSD2_SMU as T
          join WINDOW_ZHNJ as U
          on T.HWSD2_SMU_ID=U.HWSD2_SMU
          order by HWSD2_SMU_ID")

[1] 0

records <- dbGetQuery(con, "select * from ZHNJ_SMU")
dim(records)

[1] 57 23

head(records)[,display.fields]

      ID HWSD2_SMU_ID SHARE WRB4 KOPPEN ROOT_DEPTH TEXTURE_USDA
1  82553        11341   100 FLca      D          1            9
2 138155        11365   100   LP      C          3            9
3 151139        11367   100 LPrz      C          3            9
4 190823        11368   100 LVha      C          3            9
5  13161        11372   100   AN      C          1            9
6  13163        11373   100   AN      C          1            9

sort(unique(records$WRB4))

 [1] "ALfr"       "AN"         "ATpa&ATtr" "CMdy"       "CMfl"
 [6] "FLca"       "FLeu"       "GLeu"       "LP"         "LPdy"
[11] "LPrz"       "LVha"       "PLdy"       "PLeu"       "RGdy"
[16] "TC"         "UMac"       "WR"
```

This window has 18 different WRB 2022 classes.

In this window all the map units have only one component, as we can see from the `SHARE` field, which shows 100% for the map unit share.

```
unique(records$SHARE)

[1] 100
```

This was a decision by the compilers of the Chinese portion of the HWSD. See §7, below, for a window where some map units have multiple components. The `HWSD2_SMU` (map unit) table only has the *dominant* soil, i.e., the one with the largest share in the map unit, which for the Chinese example is the unique component.

Many of these fields are R **factors** although they were in the relational database as integers or characters; we have to inform R of this, in order to make categorical raster maps (see below).

Task 23 : Convert fields to R factors as appropriate. •

```
str(records)

'data.frame':	57 obs. of  23 variables:
 $ ID            : int  82553 138155 151139 190823 13161 13163 51231 51253 139174 258349 ...
 $ HWSD2_SMU_ID  : int  11341 11365 11367 11368 11372 11373 11375 11376 11379 11389 ...
 $ WISE30s_SMU_ID : chr  "WD40011341" "WD30011365" "WD30011367" "WD30011368" ...
 $ HWSD1_SMU_ID  : int  11341 11365 11367 11368 11372 11373 11375 11376 11379 11389 ...
 $ COVERAGE      : int  2 2 2 2 2 2 2 2 2 2 ...
```

```
 $ SHARE           : int  100 100 100 100 100 100 100 100 100 100 ...
 $ WRB4            : chr  "FLca" "LP" "LPrz" "LVha" ...
 $ WRB_PHASES      : chr  "FLca" "LP" "LPrz" "LVle" ...
 $ WRB2            : chr  "FL" "LP" "LP" "LV" ...
 $ WRB2_CODE       : int  10 18 18 19 3 3 8 8 18 26 ...
 $ FAO90           : chr  "FLc" "LP" "LPk" "LVh" ...
 $ KOPPEN          : chr  "D" "C" "C" "C" ...
 $ TEXTURE_USDA    : int  9 9 9 9 9 9 9 9 11 11 ...
 $ REF_BULK_DENSITY: num  1.7 1.76 1.81 1.66 1.69 1.69 1.71 1.71 1.65 1.64 ...
 $ BULK_DENSITY    : num  1.38 1.11 1.23 1.5 0.86 0.86 1.25 1.25 1.2 1.58 ...
 $ DRAINAGE        : chr  "MW" "I" "I" "MW" ...
 $ ROOT_DEPTH      : int  1 3 3 3 1 1 1 1 3 1 ...
 $ AWC             : int  161 38 34 41 151 151 144 144 33 133 ...
 $ PHASE1          : int  NA NA NA 2 20 20 NA NA NA 20 ...
 $ PHASE2          : int  NA NA NA NA NA NA NA NA NA NA ...
 $ ROOTS           : int  NA NA NA NA NA NA NA NA NA NA ...
 $ IL              : int  NA NA NA NA NA NA NA NA NA NA ...
 $ ADD_PROP        : int  0 0 0 0 0 0 0 0 0 0 ...

for (i in names(records)[c(2:5,7:13,16:17,19:23)])
  {
    eval(parse(text=paste0("records$",i," <- as.factor(records$",i,")")))
  }
str(records)

'data.frame': 57 obs. of  23 variables:
 $ ID              : int  82553 138155 151139 190823 13161 13163 51231 51253 139174 258349 ...
 $ HWSD2_SMU_ID    : Factor w/ 57 levels "11341","11365",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ WISE30s_SMU_ID  : Factor w/ 57 levels "WD30011365","WD30011367",..: 56 1 2 3 4 5 6 7 8 9 ...
 $ HWSD1_SMU_ID    : Factor w/ 57 levels "11341","11365",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ COVERAGE        : Factor w/ 1 level "2": 1 1 1 1 1 1 1 1 1 1 ...
 $ SHARE           : int  100 100 100 100 100 100 100 100 100 100 ...
 $ WRB4            : Factor w/ 18 levels "ALfr","AN","ATpa&ATtr",..: 6 9 11 12 2 2 4 4 10 15 ...
 $ WRB_PHASES      : Factor w/ 19 levels "ALfr","ANsk",..: 6 9 11 13 2 2 4 4 10 16 ...
 $ WRB2            : Factor w/ 13 levels "AL","AN","AT",..: 5 7 7 8 2 2 4 4 7 10 ...
 $ WRB2_CODE       : Factor w/ 13 levels "2","3","5","8",..: 5 7 7 8 2 2 4 4 7 10 ...
 $ FAO90           : Factor w/ 18 levels "ACu","ALf","ANh",..: 7 10 12 13 3 3 5 5 11 16 ...
 $ KOPPEN          : Factor w/ 2 levels "C","D": 2 1 1 1 1 1 1 1 1 1 ...
 $ TEXTURE_USDA    : Factor w/ 4 levels "5","7","9","11": 3 3 3 3 3 3 3 3 4 4 ...
 $ REF_BULK_DENSITY: num  1.7 1.76 1.81 1.66 1.69 1.69 1.71 1.71 1.65 1.64 ...
 $ BULK_DENSITY    : num  1.38 1.11 1.23 1.5 0.86 0.86 1.25 1.25 1.2 1.58 ...
 $ DRAINAGE        : Factor w/ 4 levels "I","MW","P","VP": 2 1 1 2 2 2 2 2 1 2 ...
 $ ROOT_DEPTH      : Factor w/ 2 levels "1","3": 1 2 2 2 1 1 1 1 2 1 ...
 $ AWC             : int  161 38 34 41 151 151 144 144 33 133 ...
 $ PHASE1          : Factor w/ 4 levels "2","8","13","20": NA NA NA 1 4 4 NA NA NA 4 ...
 $ PHASE2          : Factor w/ 0 levels: NA NA NA NA NA NA NA NA NA NA ...
 $ ROOTS           : Factor w/ 0 levels: NA NA NA NA NA NA NA NA NA NA ...
 $ IL              : Factor w/ 0 levels: NA NA NA NA NA NA NA NA NA NA ...
 $ ADD_PROP        : Factor w/ 1 level "0": 1 1 1 1 1 1 1 1 1 1 ...
```

Note: This is an example of building a valid R command string using `paste` to include both fixed and variable text (which changes each time through the loop), then parsing it with `parse` to build a valid R expression and finally evaluating it with `eval`.

Note: We could assign the names for factor levels from the metadata lookup tables; I have not yet implemented this.

Task 24 : Remove fields with no data from the window's attribute table. •

Some fields are completely undefined in this window. For example, the `PHASE2` field (second phase modifier) is not used in data from China; we check this with the `all` function applied to a logical vector created by the `is.na` function and the `!` ("not") logical operator:

```
ix <- which(names(records)=="PHASE2")
all(is.na(records[,ix]))

[1] TRUE
```

We find all these and remove them from the dataframe, thus simplifying the table:

```
dim(records)

[1] 57 23

df <- records
for (i in 1:length(names(records))) {
    if (all(is.na(records[,i]))) df <- df[-i]
    }
records <- df
dim(records)

[1] 57 21

rm(df, ix, i)
```

Now we have a table of just the units in our window, with just the defined fields. This table is a flat file, and can be exported as a comma-separated values (CSV) file for use in spreadsheets or to be imported into a database program, using the `write.csv` function.

Task 25 : Export the map unit table as a comma-separated values (CSV) file. •

```
write.csv(records, file="./HWSD_Nanjing.csv")
```

Task 26 : Display the names of the soil classes for each map unit in the window. •

We do this with another table join. We repeat the previous query but save the results as a new table, which we name `tmp`. We can then use this for the next `join`, to return the map unit codes, symbols and names. We save the final result of the query.

> **Note:** Here we use the `dbExecute` function instead of `dbGetQuery` to create a temporary table, and then the `dbGetQuery` to get the results of a query on this table into an R `data.frame`.

```
dbExecute(con,
          "drop table if exists TMP") # to overwrite
```

```
[1] 0
```

```
dbExecute(con,
          "create temp table TMP as select HWSD2_SMU, WRB4 from HWSD2_SMU as T
           right join WINDOW_ZHNJ as U on T.HWSD2_SMU_ID=U.HWSD2_SMU
           order by HWSD2_SMU_ID")
```

```
[1] 0
```

```
# each map unit with its map unit ID, WRB4 code and name
print(window.wrb4 <- dbGetQuery(con,
         "select T.HWSD2_SMU, u.CODE, u.VALUE from D_WRB4 as U
          inner join TMP as T  on T.WRB4=U.CODE"))
```

```
   hwsd2_smu CODE              VALUE
1      11814 UMac    Acric Umbrisols
2      11823 ALfr     Ferric Alisols
3      11372   AN           Andosols
4      11373   AN           Andosols
5      11925   TC         Technosols
6      11375 CMdy  Dystric Cambisols
7      11376 CMdy  Dystric Cambisols
8      11870 CMdy  Dystric Cambisols
9      11834 CMfl Ferralic Cambisols
10     11341 FLca Calcaric Fluvisols
11     11492 FLca Calcaric Fluvisols
12     11495 FLca Calcaric Fluvisols
13     11499 FLca Calcaric Fluvisols
14     11501 FLca Calcaric Fluvisols
15     11483 FLeu   Eutric Fluvisols
16     11485 FLeu   Eutric Fluvisols
17     11489 FLeu   Eutric Fluvisols
18     11490 FLeu   Eutric Fluvisols
19     11493 FLeu   Eutric Fluvisols
20     11663 GLeu    Eutric Gleysols
21     11665 GLeu    Eutric Gleysols
22     11667 GLeu    Eutric Gleysols
23     11365   LP          Leptosols
24     11379 LPdy  Dystric Leptosols
25     11367 LPrz  Rendzic Leptosols
26     11368 LVha    Haplic Luvisols
27     11857 LVha    Haplic Luvisols
28     11858 LVha    Haplic Luvisols
29     11859 LVha    Haplic Luvisols
30     11860 LVha    Haplic Luvisols
31     11877 PLdy  Dystric Planosols
32     11875 PLeu   Eutric Planosols
33     11876 PLeu   Eutric Planosols
34     11389 RGdy   Dystric Regosols
35     11390 RGdy   Dystric Regosols
36     11925   TC         Technosols
37     11927   WR         Open Water
38     11928   WR         Open Water
```

```
dbRemoveTable(con, "TMP")
```

5 Raster attribute maps

In this section we show how to display maps of both categorical (classified) and continuous attributes from the HWSD2 database. We have a `SpatRaster` object with the map unit IDs; we now want to re-classify this from linked entries in the database. Examine its internal structure:

```r
names(hwsd.zhnj)
```

```
[1] "HWSD2"
```

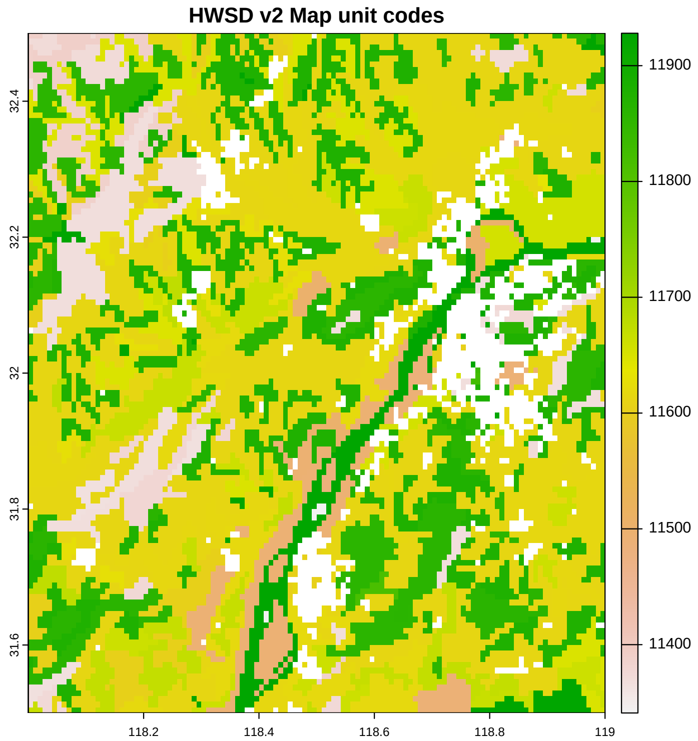
```r
class(values(hwsd.zhnj$HWSD2))
```

```
[1] "matrix" "array"
```

```r
is.numeric(values(hwsd.zhnj$HWSD2))
```

```
[1] TRUE
```

```r
plot(hwsd.zhnj, main="HWSD v2 Map unit codes")
```



The values for each pixel are integers. In fact these are classes, i.e., HWSD2 identifiers.

Task 27 : Display a map of the WRB 2022 soil types •

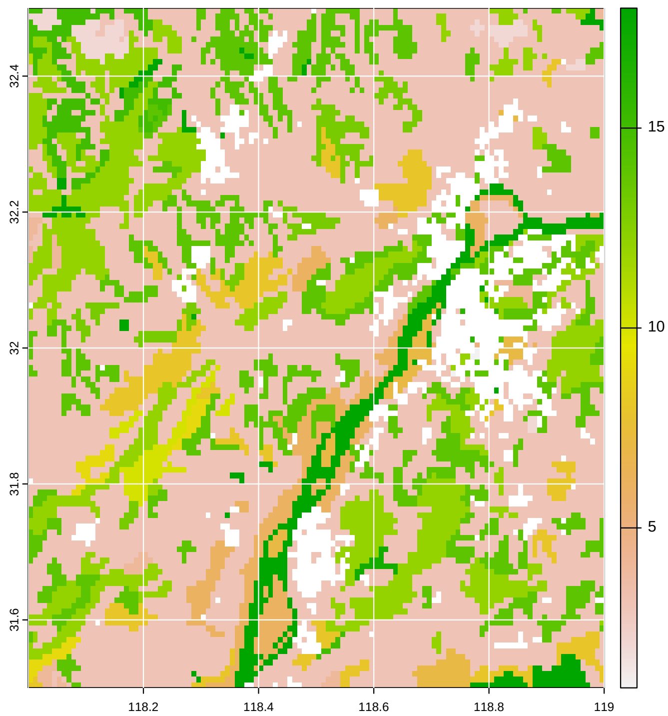
We reclassify this "continuous" raster (as implied by integers) to another integer-values raster, containing the level codes of the WRB4 classes. These are the factors in the `WRB4` field of the `records` object we extracted from the database.

```r
as.numeric(as.character(records$HWSD2_SMU_ID))
```

```
 [1] 11341 11365 11367 11368 11372 11373 11375 11376 11379 11389
[11] 11390 11483 11485 11489 11490 11492 11493 11495 11499 11501
[21] 11604 11605 11613 11614 11615 11616 11617 11619 11620 11627
[31] 11634 11645 11649 11651 11652 11655 11656 11663 11665 11667
[41] 11668 11672 11675 11814 11823 11834 11857 11858 11859 11860
[51] 11870 11875 11876 11877 11925 11927 11928
```

```r
as.numeric(records$WRB4)
```

```
 [1]   6   9 11 12   2   2   4   4 10 15 15   7   7   7   7   6   7   6   6   6
[21]   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   8   8   8
[41]   3   3   3 17   1   5 12 12 12 12   4 14 14 13 16 18 18

# reclassify the numeric class codes with the levels of the WRB4 codes
rcl.matrix <- cbind(id = as.numeric(as.character(records$HWSD2_SMU_ID)),
                    wrb4 = as.numeric(records$WRB4))
dim(rcl.matrix)

[1] 57  2

head(rcl.matrix)

        id wrb4
[1,] 11341    6
[2,] 11365    9
[3,] 11367   11
[4,] 11368   12
[5,] 11372    2
[6,] 11373    2

hwsd.zhnj.wrb4 <- classify(hwsd.zhnj, rcl.matrix)
plot(hwsd.zhnj.wrb4)
grid(col="white", lty=1)
```



Finally, we specify that the raster is categorical, with the `as.factor` method, and assign the labels

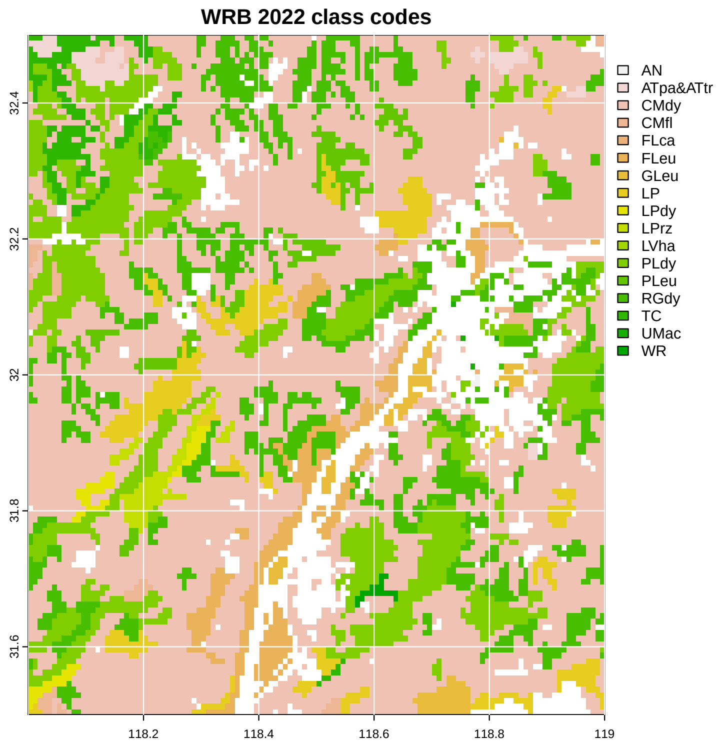
```
hwsd.zhnj.wrb4 <- as.factor(hwsd.zhnj.wrb4)
levels(hwsd.zhnj.wrb4)

[[1]]
  ID HWSD2
1  1     1
2  2     2
3  3     3
4  4     4
```

```
5   5     5
6   6     6
7   7     7
8   8     8
9   9     9
10  10    10
11  11    11
12  12    12
13  13    13
14  14    14
15  15    15
16  16    16
17  17    17
18  18    18
```

```r
levels(records$WRB4)
```

```
 [1]  "ALfr"       "AN"         "ATpa&ATtr"  "CMdy"       "CMfl"
 [6]  "FLca"       "FLeu"       "GLeu"       "LP"         "LPdy"
[11]  "LPrz"       "LVha"       "PLdy"       "PLeu"       "RGdy"
[16]  "TC"         "UMac"       "WR"
```

```r
levels(hwsd.zhnj.wrb4) <- levels(records$WRB4)
plot(hwsd.zhnj.wrb4, main = "WRB 2022 class codes")
grid(col="white", lty=1)
```



Task 28 : Display a map of the topsoil sand proportion. •

For this, we need to refer to the layers table, which contains the by-layer soil properties, such as sand concentration.

Task 29 : Display the metadata for the layers table. •

```r
print(dbGetQuery(con, "select * from HWSD2_LAYERS_METADATA"), width=100)
```

```
   ID           FIELD       UNIT                      DESCRIPTION DATATYPE          DOMAIN
1   1              ID       <NA>             Database Internal ID   Number            <NA>
2   2    HWSD2_SMU_ID       <NA>          Soil Mapping Unit HWSD2   Number            <NA>
3   3 WISE30s_SMU_ID       <NA>        Soil Mapping Unit WISE30s   String            <NA>
4   4    HWSD1_SMU_ID       <NA>          Soil Mapping Unit HWSD1   Number            <NA>
5   5        COVERAGE       <NA>                         Coverage   Number      D_COVERAGE
6   6        SEQUENCE       <NA>     Sequence in Soil Mapping Unit   Number            <NA>
7   7           SHARE          %        Share in Soil Mapping Unit   Number            <NA>
8   8 NSC_MU_SOURCE1       <NA>      National Soil Classification   String            <NA>
9   9 NSC_MU_SOURCE2       <NA>      National Soil Classification   String            <NA>
10 10      WRB_PHASES      Class         Soil Unit Symbol (WRB 2022)   String    D_WRB_PHASES
11 11            WRB4      Class Soil Unit Symbol (WRB 2022 - 4 digit)   String          D_WRB4
12 12            WRB2      Class Soil Unit Symbol (WRB 2022 - 2 digit)   String          D_WRB2
13 13           FAO90      Class         Soil Unit Symbol (FAO 1990)   String         D_FAO90
14 14      ROOT_DEPTH      Class              Rootable Soil Depth   Number    D_ROOT_DEPTH
15 15          PHASE1       <NA>                           PHASE1   Number         D_PHASE
16 16          PHASE2       <NA>                           PHASE2   Number         D_PHASE
17 17           ROOTS         cm           Obstacle to Roots (ESDB)   Number         D_ROOTS
18 18              IL         cm          Impermeable Layer  (ESDB)   Number            D_IL
19 19             SWR      Class          Soil Water Regime (ESDB)   Number           D_SWR
20 20        DRAINAGE      Class           Reference Soil Drainage   Number      D_DRAINAGE
21 21             AWC         mm          AWC for Rootable Soil Depth     <NA>            <NA>
22 22        ADD_PROP      Class              Additional Properties   Number      D_ADD_PROP
23 23           LAYER       <NA>        Depth Layer (from D1 to D7)   String            <NA>
24 24          TOPDEP         cm              Depth of top of layer   Number            <NA>
25 25          BOTDEP         cm           Depth of bottom of layer   Number            <NA>
26 26          COARSE   % volume                  Coarse fragments   Number            <NA>
27 27            SAND   % weight                              Sand   Number            <NA>
28 28            SILT   % weight                              Silt   Number            <NA>
29 29            CLAY   % weight                              Clay   Number            <NA>
30 30    TEXTURE_USDA       <NA>     Texture class (USDA conventions)   Number  D_TEXTURE_USDA
31 31   TEXTURE_SOTER       <NA>    Texture class (SOTER conventions)   String D_TEXTURE_SOTER
32 32            BULK      g/cm3                      Bulk Density   Number            <NA>
33 33        REF_BULK      g/cm3            Reference Bulk Density   Number            <NA>
34 34      ORG_CARBON   % weight            Organic Carbon Content   String            <NA>
35 35        PH_WATER   -log(H+)                        pH in water   Number            <NA>
36 36         TOTAL_N       g/kg            Total nitrogen content   Number            <NA>
37 37        CN_RATIO       <NA>        Carbon/Nitrogen ratio (C/N)   Number            <NA>
38 38        CEC_SOIL   cmolc/kg                          CEC soil   Number            <NA>
39 39        CEC_CLAY   cmolc/kg                          CEC clay   Number            <NA>
40 40         CEC_EFF   cmolc/kg                              ECEC   Number            <NA>
41 41             TEB   cmolc/kg                               TEB   Number            <NA>
42 42            BSAT %  CECsoil                   Base Saturation   Number            <NA>
43 43        ALUM_SAT     %  ECEC               Aluminium saturation   Number            <NA>
44 44             ESP          %     Exchangeable Sodium Percentage   Number            <NA>
45 45      TCARBON_EQ   % weight                  Calcium Carbonate   Number            <NA>
46 46          GYPSUM   % weight                    Gypsum content   Number            <NA>
47 47       ELEC_COND       dS/m              Electric Conductivity   Number            <NA>
```

```r
dbGetQuery(con, "select * from HWSD2_LAYERS_METADATA as T
                where (T.FIELD='SAND'  or T.FIELD='LAYER')")
```

```
  ID FIELD     UNIT                 DESCRIPTION DATATYPE DOMAIN
1 23 LAYER     <NA> Depth Layer (from D1 to D7)   String   <NA>
2 27  SAND % weight                        Sand   Number   <NA>
```

We see that the field `SAND` is a numeric value, without a lookup table, i.e., the value `NA` is in the `DOMAIN` field for this property. The `LAYER` field gives the layer number in the profile, from `"D1"` to `"D7"`. The limits of these layers are given in the technical report [1, §2.3.3] as $D1 = 0 \ldots 20$, $D2 = 20 \ldots 40$, $D3 = 40 \ldots 60$, $D4 = 60 \ldots 80$, $D5 = 80 \ldots 100$, $D6 = 100 \ldots 150$, $D7 = 150 \ldots 200$. Note that these do *not* correspond to the layer definitions of

`GlobalSoilMap.net`; in particular, the top layer is quite thick in comparison.

To extract the value of a soil property for a layer, we must join the layer table with the map unit table and then select the layer and property. We select only the first layer with the `where` operator.

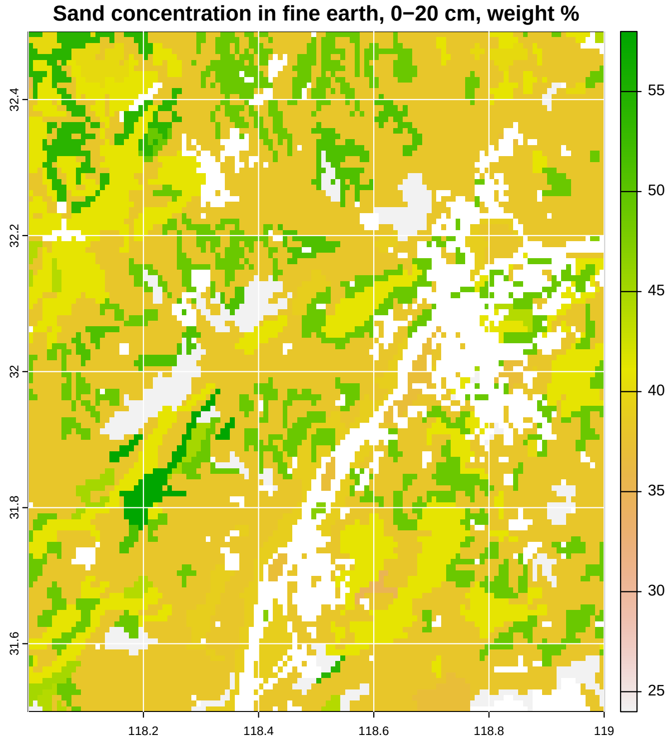
```
sand.d1 <- dbGetQuery(con,
                "select U.HWSD2_SMU_ID, U.SAND from ZHNJ_SMU as T
                 join HWSD2_LAYERS as U on T.HWSD2_SMU_ID=U.HWSD2_SMU_ID
                 where U.LAYER='D1'
                 order by U.HWSD2_SMU_ID")
head(sand.d1)

  HWSD2_SMU_ID SAND
1        11341   47
2        11365   45
3        11367   36
4        11368   41
5        11372   40
6        11373   40

sand.d1[ix <- which(sand.d1$SAND < 0),]

   HWSD2_SMU_ID SAND
55        11925   -9
56        11927   -1
57        11928   -1
```

Negative values must indicate missing values; it is unclear why there are different numbers. Anyway these should be either replaced with 0 (no sand) or `NA` (not applicable), depending on whether we use them to compute or display.

```
sand.d1[ix, "SAND"] <- NA
```

Now this two-column table can be used to reclassify the raster. Note the `NA` areas, mostly water bodies, as well as urban areas and large rivers already marked as `NA`.

```
hwsd.zhnj.sand.d1 <- classify(hwsd.zhnj, sand.d1)
plot(hwsd.zhnj.sand.d1,
     main = "Sand concentration in fine earth, 0-20 cm, weight %")
grid(col="white", lty=1)
```



We can combine layers in various ways, using arithmetic or summary functions in R. Much of this is possible also in SQL, but it may be easier to get the relevant data frame into R and manipulate it there. And some operations are quite difficult in SQL, as in the following example[15].

For example, to find the wieghted average sand concentration, we read all layers into a table and then summarize. We must use the depth slices as given in the HWSD report, see above.

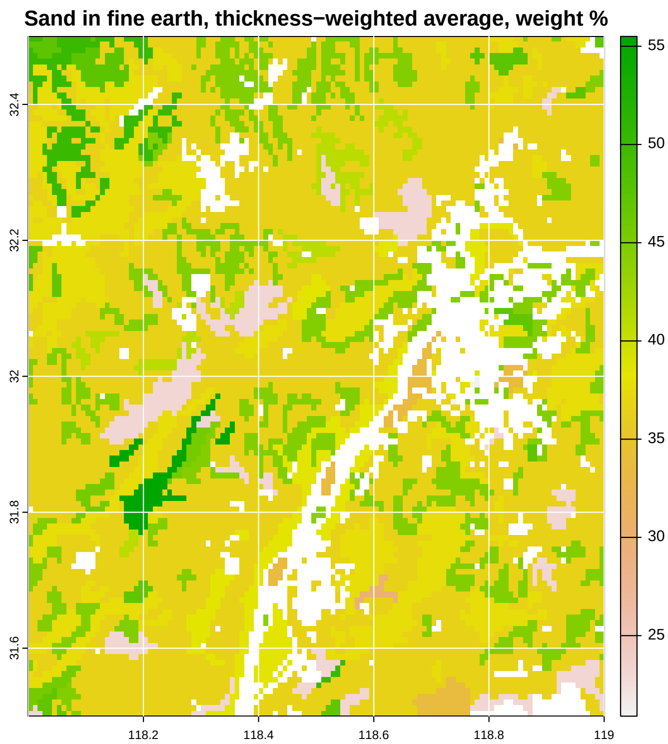
```
sand.all <- dbGetQuery(con,
                "select U.HWSD2_SMU_ID, U.LAYER, U.SAND from ZHNJ_SMU as T
                 join HWSD2_LAYERS as U on T.HWSD2_SMU_ID=U.HWSD2_SMU_ID
                 order by U.HWSD2_SMU_ID")
head(sand.all, 28)

   HWSD2_SMU_ID LAYER SAND
1         11341    D1   47
2         11341    D2   45
3         11341    D3   45
4         11341    D4   45
5         11341    D5   44
6         11341    D6   45
7         11341    D7   41
8         11365    D1   45
9         11365    D2   46
10        11365    D3   -7
11        11365    D4   -7
12        11365    D5   -7
13        11365    D6   -7
14        11365    D7   -7
15        11367    D1   36
16        11367    D2   36
17        11367    D3   -7
18        11367    D4   -7
```

[15] You are welcome to try to do this in SQL.

```
19         11367     D5    -7
20         11367     D6    -7
21         11367     D7    -7
22         11368     D1    41
23         11368     D2    36
24         11368     D3    34
25         11368     D4    34
26         11368     D5    36
27         11368     D6    39
28         11368     D7    39

 # thickness of each layer
thick <- c(20, 20, 20, 20, 20, 50, 50)
```

Note that all layers are present for all profiles, even if the soil is not present to 200 cm. This is indicated by negative numbers for the corresponding value.

```
sand.all$HWSD2_SMU_ID <- as.factor(sand.all$HWSD2_SMU_ID)
sand.all.s <- split(sand.all, sand.all$HWSD2_SMU_ID) # make a list for `lapply`
sand.avg <- unlist(lapply(sand.all.s, function(x)
    {   v <- which(x$SAND >= 0) # identify layers with non-missing values
        s <- x$SAND[v]; t <- thick[v]  # weighted average
        return(sum(s*t)/sum(t))
    }))
hwsd.zhnj.sand.avg <- classify(hwsd.zhnj,
                               data.frame(HWSD2_SMU_ID=sand.d1$HWSD2_SMU_ID, sand.avg))
plot(hwsd.zhnj.sand.avg,
     main = "Sand in fine earth, thickness-weighted average, weight %")
grid(col="white", lty=1)
```



6 Polygon maps

Although the HWSD is a raster dataset, it was created from a polygon map. These use much less storage and are generally more attractive. Modellers will want to use the raster but many others will prefer polygons.

6.1 Raster to polygon

Task 30 : Convert the raster image to a polygon map; each polygon should be labelled with the code of the contiguous pixels that make up the polygon. •

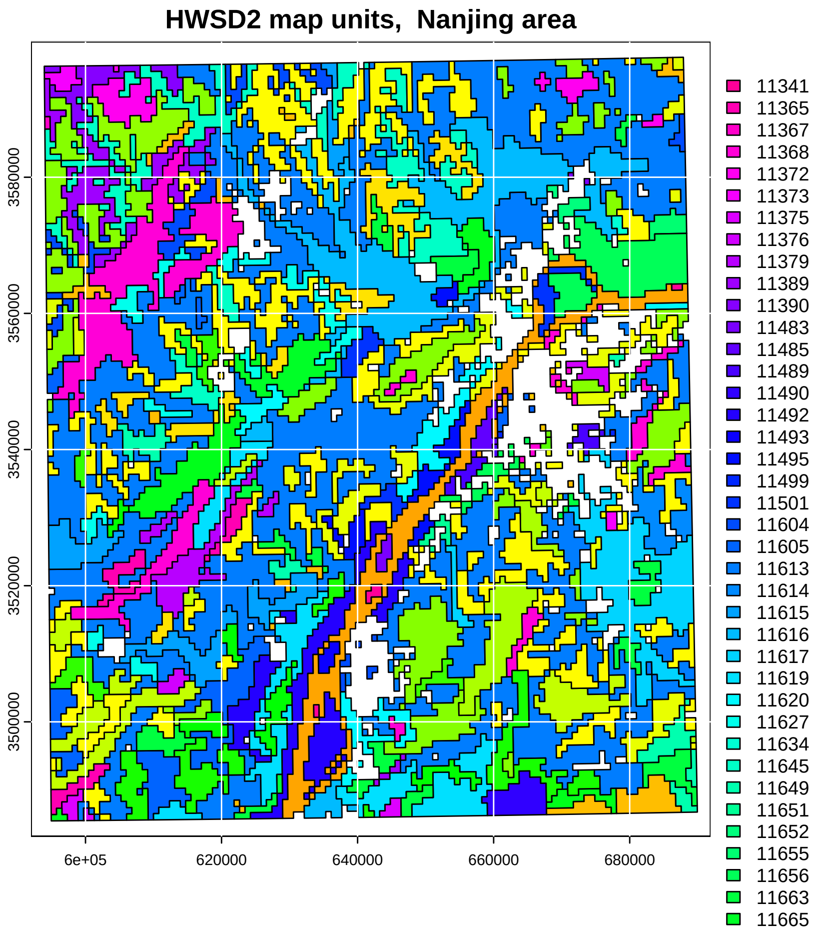
The `terra` package has a function `as.polygons` for this.

```
hwsd.zhnj.poly <- as.polygons(hwsd.zhnj, dissolve = TRUE, values = TRUE)
class(hwsd.zhnj.poly)

[1] "SpatVector"
attr(,"package")
[1] "terra"

summary(hwsd.zhnj.poly)

     HWSD2
 Min.   :11341
 1st Qu.:11490
 Median :11620
 Mean   :11621
 3rd Qu.:11675
 Max.   :11928
```

There are only 57 map units (sets of polygons with the same code), as opposed to 1.44×10^4 raster cells, a very large savings in memory and processing time.

Polygon maps of classes `spatVector` are not projected in the same way as raster maps; there is no re-sampling necessary, just a re-projection of all the boundaries. This is accomplished by using the `project` function of the `terra` package. This requires a target Coordinate Reference System (CRS), which can be specified by EPSG code. We defined the appropriate UTM CRS (including elipsoid, datum and offset from the WGS84 elipsoid) for the UTM version of the raster image in §3.1, and determined its EPSG code.

```
hwsd.zhnj.poly.utm <- project(hwsd.zhnj.poly, paste0("EPSG:", epsg))
plot(hwsd.zhnj.poly.utm, "HWSD2",  type = "classes",
     axes = TRUE, main = "HWSD2 map units,  Nanjing area")
grid(col="white", lty=1)
```



Again notice how the UTM grid is not square nor perfectly N-S and E-W oriented.

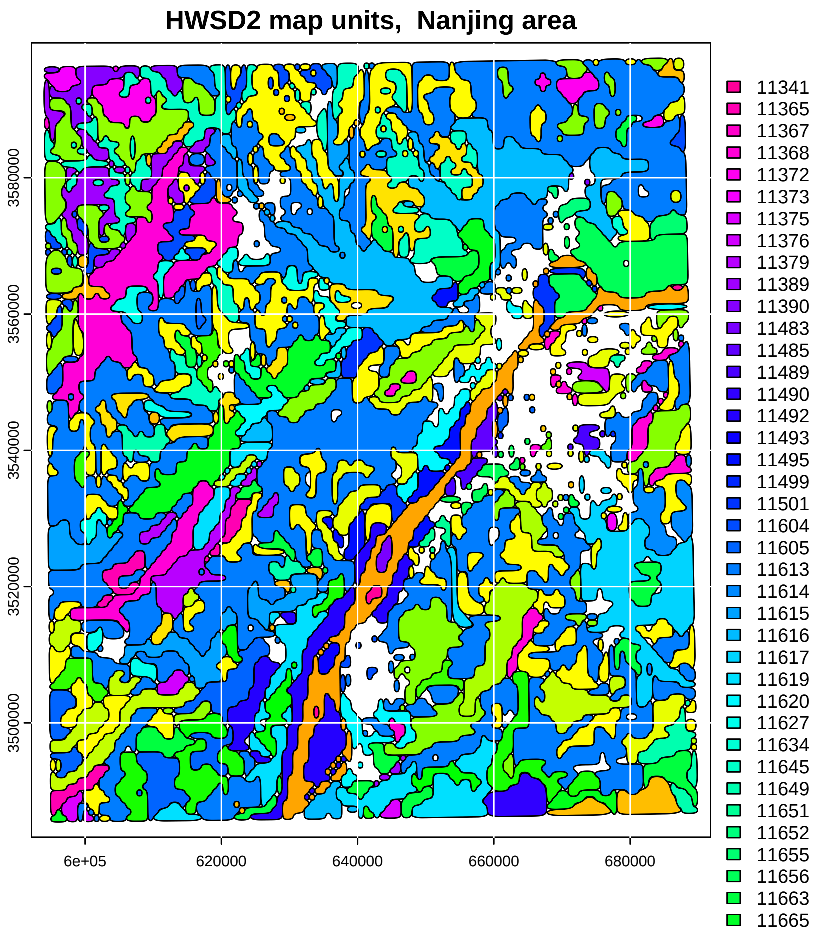
The jagged edges of the polygons obviously do not correspond to reality, and they are ugly. Smooth these out with functions from the `smoothr` package. This works with spatial polygons of classes from the `sf` package, not directly with `terra` package objects. So we first convert to an `sf` object with the `st_as_sf` method, do the smoothing, and convert back to a `SpatVector`.

There are three smoothing algorithms that can be used; we try with a kernel smoother of the boundaries, specified with argument `method` and with a smoothness parameter specified with the `smoothness` argument. For this latter, the default value `1` selects a bandwith of the mean distance between vertices. See `?smooth` for other options.

```
require(smoothr)
hwsd.zhnj.sf.utm <- st_as_sf(hwsd.zhnj.poly.utm)
# default smoothness is mean distance between adjacent points
hwsd.zhnj.sf.utm.smooth <- smooth(hwsd.zhnj.sf.utm,
                                  method = "ksmooth", smoothness = 1)
hwsd.zhnj.poly.utm.smooth <- vect(hwsd.zhnj.sf.utm.smooth)
summary(hwsd.zhnj.poly.utm.smooth)

     HWSD2
 Min.   :11341
 1st Qu.:11490
 Median :11620
```

```
 Mean   :11621
 3rd Qu.:11675
 Max.   :11928
```

```
plot(hwsd.zhnj.poly.utm.smooth, "HWSD2", type = "classes",
   axes = TRUE, main = "HWSD2 map units,  Nanjing area")
grid(col="white", lty=1)
```



Notice the loss of geographic precision compared to the grid. The grid was obviously created from a polygon map, and this reconstruction should be close to the original. However, note also the small "unknown" areas between some of the smoothed polygons, a result of the smooth, rounded polygon corners.

6.2 Polygon attribute maps

So far the polygons just have the soil map unit code. A `SpatVector` object can also have an attached data frame, with many attributes per polygon.

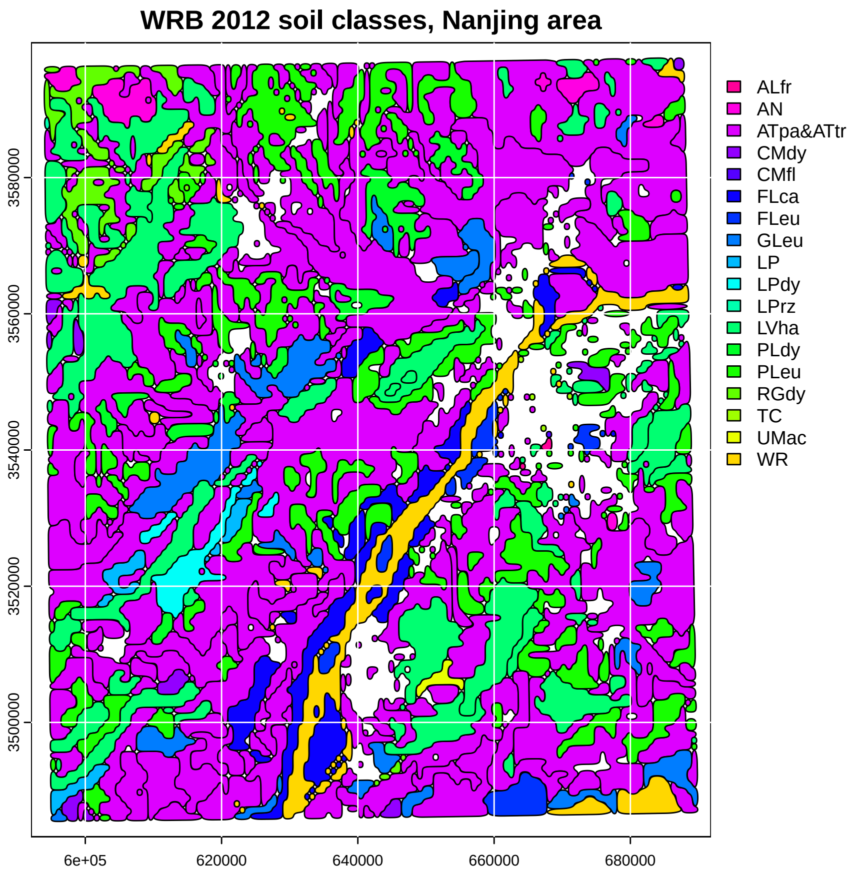
Task 31 : Convert the polygon map to a `SpatVector` object. Then add the attribute database to this polygon map, and display soil class (WRB4) and topsoil sand proportion maps. •

First we add the database. Note it is in the same sequence as the polygon IDs.

```
hwsd.zhnj.utm.smooth <- vect(hwsd.zhnj.sf.utm.smooth)
values(hwsd.zhnj.utm.smooth) <- records
names(hwsd.zhnj.utm.smooth)

 [1] "ID"                "HWSD2_SMU_ID"      "WISE30s_SMU_ID"
 [4] "HWSD1_SMU_ID"      "COVERAGE"          "SHARE"
 [7] "WRB4"              "WRB_PHASES"        "WRB2"
[10] "WRB2_CODE"         "FAO90"             "KOPPEN"
[13] "TEXTURE_USDA"      "REF_BULK_DENSITY"  "BULK_DENSITY"
[16] "DRAINAGE"          "ROOT_DEPTH"        "AWC"
[19] "PHASE1"            "ROOTS"             "ADD_PROP"
```

Now the map of the WRB 2022 classes:

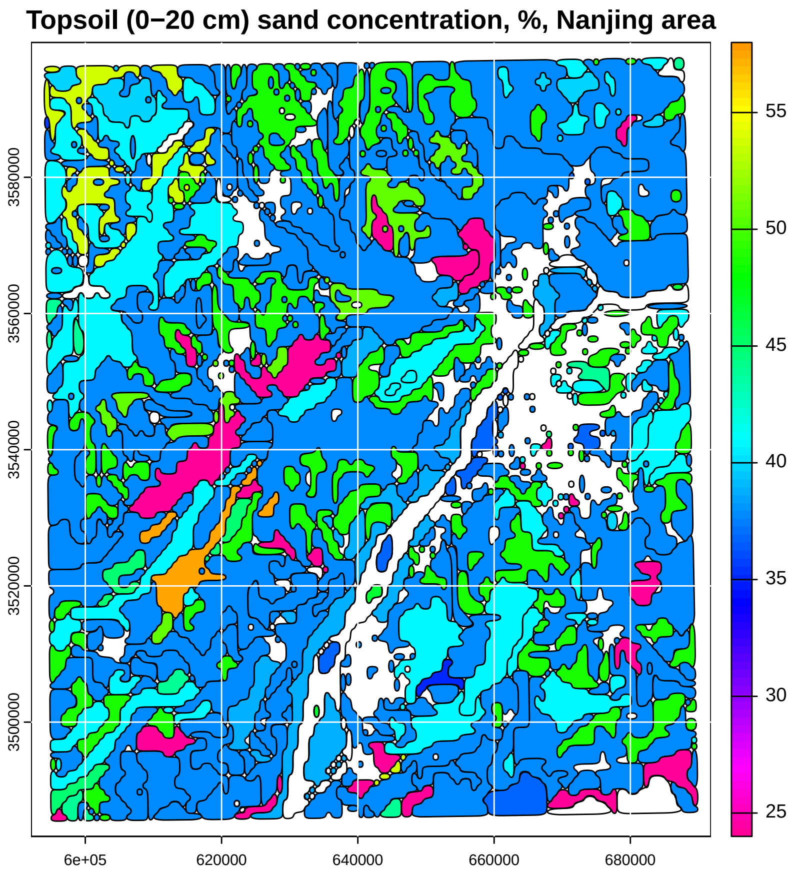
```
plot(hwsd.zhnj.utm.smooth, "WRB4", legend = TRUE,
     main = "WRB 2012 soil classes, Nanjing area",
     axes = TRUE)
grid(col="white", lty=1)
```



For the topsoil sand we need to use the result of the table join (see above) and add this as a new attribute to the map unit table. In other words, flattening the layer table.

```
m <- match(values(hwsd.zhnj.utm.smooth)$HWSD2, sand.d1$HWSD2_SMU_ID)
# add a new field to the data frame
values(hwsd.zhnj.utm.smooth) <-  data.frame(values(hwsd.zhnj.utm.smooth)[m,],
                                            sand.d1 = sand.d1[m, "SAND"])
```

```
names(hwsd.zhnj.utm.smooth)

 [1] "ID"                "HWSD2_SMU_ID"      "WISE30s_SMU_ID"
 [4] "HWSD1_SMU_ID"      "COVERAGE"          "SHARE"
 [7] "WRB4"              "WRB_PHASES"        "WRB2"
[10] "WRB2_CODE"         "FAO90"             "KOPPEN"
[13] "TEXTURE_USDA"      "REF_BULK_DENSITY"  "BULK_DENSITY"
[16] "DRAINAGE"          "ROOT_DEPTH"        "AWC"
[19] "PHASE1"            "ROOTS"             "ADD_PROP"
[22] "sand.d1"

# few values, show them all directly in legend
sort(unique(hwsd.zhnj.utm.smooth$sand.d1))

 [1] 24 35 36 37 38 39 40 41 44 45 47 49 51 54 58

plot(hwsd.zhnj.utm.smooth, "sand.d1", legend = TRUE, type = "continuous",
     main = "Topsoil (0-20 cm) sand concentration, %, Nanjing area")
grid(col="white", lty=1)
```



Some areas have no information for the attribute; these are water bodies, some urban areas, and other unsurveyed areas.

6.3 Saving polygon maps

Polygon maps of class `spatVect` can be saved for later use in a variety of vector GIS formats; see the GDAL drivers list[16]. To see the drivers available on your system, use the `gdal` command with the `drivers` argument set to `TRUE`:

```
g.drivers <- gdal(drivers = TRUE)
head(v.drivers <- g.drivers[g.drivers$type == "vector", ], 12)

              name   type        can   vsi
143         AVCBin vector       read  TRUE
144         AVCE00 vector       read  TRUE
145     AmigoCloud vector read/write FALSE
146            CSV vector read/write  TRUE
147            CSW vector       read FALSE
148          Carto vector read/write FALSE
149            DGN vector read/write  TRUE
150            DXF vector read/write  TRUE
151         EDIGEO vector       read  TRUE
152           EEDA vector       read FALSE
153 ESRI Shapefile vector read/write  TRUE
154       ESRIJSON vector       read  TRUE
                                 long.name
143             Arc/Info Binary Coverage
144          Arc/Info E00 (ASCII) Coverage
145                             AmigoCloud
146           Comma Separated Value (.csv)
147 OGC CSW (Catalog  Service for the Web)
148                                  Carto
149                        Microstation DGN
150                              AutoCAD DXF
151             French EDIGEO exchange format
152                   Earth Engine Data API
153                          ESRI Shapefile
154                                ESRIJSON
```

You can search for a file format by part of the long name, or by the GDAL code, using the `grep` "General Regular Expression" function:

```
v.drivers[grep("Shapefile", v.drivers$long.name, fixed = TRUE),]

              name   type        can  vsi      long.name
153 ESRI Shapefile vector read/write TRUE ESRI Shapefile

v.drivers[grep("ESRI", v.drivers$name, fixed = TRUE),]

              name   type        can  vsi      long.name
153 ESRI Shapefile vector read/write TRUE ESRI Shapefile
154       ESRIJSON vector       read TRUE       ESRIJSON
```

Task 32 : Save the smoothed polygon map as an ESRI shapefile. Note that this map includes all the map unit attributes joined to the polygons. •

This uses the `writeVector` function of the `terra` package.

> **Note:** Each output file format has its own requirements for optional arguments and its idiosyncracies. In the case of the ESRI Shapefile format, the `overwrite` argument by itself does not work, probably because the shapefile is written as separate files in a directory. So any previous copy has to be deleted first with the `unlink` function applied to the directory where the

[16] https://gdal.org/drivers/vector/index.html

shapefile is stored. This is only necessary if the **dir.exists** function returns **TRUE**; we test this with the **if** command.

```
if (dir.exists("./HWSD_Nanjing")) unlink("./HWSD_Nanjing", recursive = TRUE)
writeVector(hwsd.zhnj.utm.smooth, filename = "./HWSD_Nanjing",
            filetype = 'ESRI Shapefile', overwrite = TRUE)
```

To check the export, you can open the shapefile in QGIS or another shapefile reader.

7 Map units with multiple components

By contrast to the Nanjing example used in the previous sections, in the Netherlands some map units have multiple components. This is because at the design resolution of the HWSD v2 (and v1.2) of 30-arcseconds ($\approx$ 1×1 km) there are often several contrasting soil types, and some contributing soil survey organizations have more detailed maps which allow the identification of these components and an estimate of their proportional area in the grid cell.

Task 33 : Create an R **data.frame** with the HWSD records for Netherlands, with attributes, and display the HWSD ID, component ID, its share, and the WRB 2022 classifications. •

This is exactly as was done for the Nanjing example, except this time selecting map units from the Netherlands:

```
dbExecute(con,
          "drop table if exists WINDOW_NL") # to overwrite

[1] 0

dbWriteTable(con, name="WINDOW_NL",
             value=data.frame(smu_id=unique(nl.id)),
             overwrite=TRUE)
# dbGetQuery(con, "select * from WINDOW_NL")
records.nl <-
 dbGetQuery(con, "select T.* from HWSD2_SMU as T
                  join WINDOW_NL as U on T.HWSD2_SMU_ID=U.SMU_ID
                  order by HWSD2_SMU_ID")
```

```
print(dim(records.nl))

[1] 48 23

print(unique(records.nl$HWSD2_SMU_ID))

 [1]  7001  7003  9345  9346  9347  9348  9349  9350  9351  9352
[11]  9353  9354  9355  9356  9357  9358  9359  9360  9361  9362
[21]  9363  9364  9365  9367  9373  9375  9382  9386  9387  9388
[31] 10184 10186 10187 10189 10215 10218 10241 10242 10243 10244
[41] 10245 10246 10248 10249 10250 10251 32068 32069

print(unique(records.nl$SHARE))

 [1] 100  90  75  70  80  35  30  95  40  50  60  45
```

We see that not every map unit has only one component. Here are the map unit IDs with less than 100% share.

```
length(ix <- which(records.nl$SHARE != 100))
```

```
[1] 38
```

```
(records.nl[ix,c("ID","HWSD2_SMU_ID", "SHARE")])
```

```
       ID HWSD2_SMU_ID SHARE
3   89224         9345    90
6   81925         9348    75
7  118857         9349    70
9  255589         9351    90
10 258267         9352    80
11  47631         9353    35
12 190848         9354    80
13 248619         9355    30
14 248625         9356    95
15 248616         9357    95
16 246889         9358    40
18 132827         9360    70
20 246891         9362    50
21  89171         9363    90
22  81904         9364    95
23  81903         9365    75
24 258154         9367    90
25 190956         9373    75
26 190990         9375    60
27 220070         9382    40
28 248553         9386    50
29 248546         9387    95
30 246916         9388    45
31  81891        10184    70
32 133119        10186    60
33 127563        10187    60
34  89135        10189    30
35 194559        10215    40
36 194646        10218    50
38 246901        10242    40
39 248580        10243    95
40 111386        10244    60
41  89057        10245    40
42 248579        10246    30
43 190954        10248    80
44 190947        10249    60
45 248596        10250    95
46  89053        10251    90
```

Now we get into a complication. The map unit table only has the dominant soil (field `HSWS2_SMU_ID`) and its share (field `SHARE`). For the others, we have to use the `HWSD2_LAYERS` table, which has not only the layers in each profile (field `LAYER`), but also the components of each map unit (field `SEQUENCE`).[17]

For example, we see from the previous results that map unit 9355 has multiple components.

Task 34 : Display each component for map unit 9355 and its share, along with its WRB classification code and name. •

For components, these are in a strangely-named field, `WRB_PHASES`, although these are not mapping phases in the usual sense of the word, they are soil

[17] This is quite poor relational database design, but we are stuck with it. There should be two relational tables, one for the components and one for the layers in each component. With the current structure the `SEQUENCE` field is repeated for each depth slice.

types. Since this is repeated for each of the seven layers, just show the records for the first layer. We also use the lookup table to display the WRB 2022 class name.

```
comp.9355 <- dbGetQuery(con,
                        "select T.HWSD2_SMU_ID, U.SEQUENCE, U.SHARE, U.WRB_PHASES
                                from HWSD2_SMU as T
                         join  HWSD2_LAYERS as U on T.HWSD2_SMU_ID = U.HWSD2_SMU_ID
                         where ((T.HWSD2_SMU_ID)=9355 and (U.LAYER='D1'))
                         order by T.HWSD2_SMU_ID, U.SEQUENCE")
print(cbind(comp.9355, WRB_NAME=d.wrb4[match(comp.9355$WRB_PHASES,
                                             d.wrb4$CODE), "VALUE"]))

  HWSD2_SMU_ID SEQUENCE SHARE WRB_PHASES         WRB_NAME
1         9355        1    30       PZal    Albic Podzols
2         9355        2    30       LVha  Haplic Luvisols
3         9355        3    20       LVgl  Gleyic Luvisols
4         9355        4    10       GLeu  Eutric Gleysols
5         9355        5     5       RGdy Dystric Regosols
6         9355        6     5         AR       Arenosols
```

This map unit has six components.

This can be expanded to show the soil properties for all the depth slices, by removed the second `where` clause and selecting the properties fields.

```
layers.comp.9355 <- dbGetQuery(con, "select T.HWSD2_SMU_ID,
                  U.SEQUENCE, U.SHARE, U.TOPDEP, U.BOTDEP,
                  U.COARSE, U.SAND, U.SILT, U.CLAY,
                  U.BULK, U.ORG_CARBON, U.PH_WATER from HWSD2_SMU as T
                  join  HWSD2_LAYERS as U on T.HWSD2_SMU_ID = U.HWSD2_SMU_ID
                  where ((T.HWSD2_SMU_ID)=9355)
                  order by T.HWSD2_SMU_ID, U.SEQUENCE, U.LAYER")
```

```
print(layers.comp.9355, width = 100)

   HWSD2_SMU_ID SEQUENCE SHARE TOPDEP BOTDEP COARSE SAND SILT CLAY BULK ORG_CARBON PH_WATER
1          9355        1    30      0     20     10   67   27    6 1.13      7.326      4.3
2          9355        1    30     20     40     17   69   25    6 1.25      2.694      4.7
3          9355        1    30     40     60     20   72   22    6 1.28      1.790      4.9
4          9355        1    30     60     80     23   73   21    6 1.38      1.731      4.9
5          9355        1    30     80    100     26   77   18    5 1.57      1.376      5.0
6          9355        1    30    100    150     29   78   17    5 1.64      1.008      5.0
7          9355        1    30    150    200     29   80   13    7 1.83      0.391      5.2
8          9355        2    30      0     20     11   41   42   17 1.50      1.510      5.9
9          9355        2    30     20     40     11   36   39   25 1.60      0.631      6.0
10         9355        2    30     40     60     12   34   36   30 1.64      0.400      6.0
11         9355        2    30     60     80     14   34   36   30 1.66      0.298      6.0
12         9355        2    30     80    100     16   36   35   29 1.67      0.223      6.1
13         9355        2    30    100    150     16   39   35   26 1.67      0.151      6.2
14         9355        2    30    150    200     15   39   35   26 1.67      0.111      6.2
15         9355        3    20      0     20      3   31   51   18 1.52      1.384      5.7
16         9355        3    20     20     40      2   27   47   26 1.64      0.578      5.8
17         9355        3    20     40     60      2   25   43   32 1.70      0.379      5.8
18         9355        3    20     60     80      3   24   43   33 1.72      0.279      6.0
19         9355        3    20     80    100      3   27   43   30 1.73      0.204      6.2
20         9355        3    20    100    150      2   29   44   27 1.72      0.132      6.4
21         9355        3    20    150    200      4   31   43   26 1.74      0.090      6.5
22         9355        4    10      0     20      8   24   46   30 1.22      1.889      6.3
23         9355        4    10     20     40      7   22   45   33 1.29      0.909      6.6
24         9355        4    10     40     60     10   23   44   33 1.29      0.620      6.7
25         9355        4    10     60     80     12   24   44   32 1.36      0.416      6.8
26         9355        4    10     80    100      8   23   46   31 1.38      0.382      6.9
27         9355        4    10    100    150     11   22   50   28 1.49      0.342      7.2
28         9355        4    10    150    200      4   24   53   23 1.54      0.300      7.5
29         9355        5     5      0     20      9   54   30   16 1.58      1.159      5.3
30         9355        5     5     20     40     23   53   29   18 1.66      0.558      5.6
31         9355        5     5     40     60     16   60   23   17 1.65      0.415      5.8
32         9355        5     5     60     80     14   59   23   18 1.66      0.170      5.8
33         9355        5     5     80    100     16   53   26   21 1.62      0.187      5.8
34         9355        5     5    100    150     19   50   28   22 1.77      0.389      5.8
35         9355        5     5    150    200     24   39   38   23 1.83      0.801      6.3
36         9355        6     5      0     20      9   87    9    4 1.44      0.628      5.7
37         9355        6     5     20     40     15   88    8    4 1.49      0.319      5.9
38         9355        6     5     40     60     16   88    8    4 1.51      0.237      6.0
39         9355        6     5     60     80     14   87    9    4 1.61      0.181      5.9
40         9355        6     5     80    100     13   87    8    5 1.62      0.118      5.9
41         9355        6     5    100    150     11   87    8    5 1.62      0.112      5.9
42         9355        6     5    150    200     11   85    9    6 1.60      0.104      5.6
```

We see that the particle-size separates are quite diverse in this map unit. Also, that the organic C % has unrealistic precision.

To make raster or polygon maps of properties, one has to decide what is to be shown in the pixel or polygon. There are several choices:

1. Use just the value for the first-listed component, i.e., the one in the `HWSD2_SMU` table, linked to the layer(s) for that component in the `HWSD2_LAYERS` table. This can be a single layer or some summary, such as average, minimum, or maximum. This was the example above.

2. Aggregated the values for the components, by a weighted average or other summary. This can be for a non-layer property, e.g., `DRAINAGE` or `ROOT_DEPTH`, or for layered properties, e.g. particle-size separates, bulk density, CEC. For example, we may want to report the minimum rooting depth or most-restrictive drainage class. For layered proper-

ties, these could be aggregated across all the components, or a single value such as a maximum could be chosen.

For some of these choices there is an SQL aggregation operator, for example `MAX`, `MIN`, and `AVG`. These all require an additional clause in the SQL statement, introduced by the SQL `GROUP BY` statement, to first group the related records and then apply the function. These functions can also take an optional `AS` modifier, to re-name the resulting field.

All of this can be done in SQL, but it may be more convenient to first make R data frames from the tables and work in R.

Here is an example.

Task 35 : Compute the organic carbon stocks in the Netherlands soils. •

This requires the SOC % content (weight basis), the bulk density, and the layer thickness of each component, with the total being a weighted average. In this database:

- SOC content is in units of % weight, i.e., 10 g C $\texttt{kg}^{-1}$; Bulk density is weight per volume, in g cm^{-3}, where water weighs 1 g cm^{-3}, i.e., 1 Mg m^{-3}.

SOC and bulk density are reported for depth slices to $2m$ thickness, or shallower if the soil is thinner to rock.

SOC stock is conventionally reported in Mg $(100\mathrm{m})^2$ over the soil column, i.e., in non-SI notation T ha^{-1}. To compute stocks, first the SOC weight is converted to a volume basis: kg C m^{-3}; this is then multplied by the thickness in m.

We first read the required dataframes in R, for the study area. The map unit ID of these map units was extracted by masking the raster, see above. Notice that both data frames are sorted by the map unit ID.

First, the map unit table:

```
# make a list for the `in` operator
idString <- toString(sprintf("'%s'", nl.id))
# format this into an SQL statement
sql.stmt <- "select * from HWSD2_SMU
            where HWSD2_SMU.HWSD2_SMU_ID in (%s)
            order by HWSD2_SMU_ID"
sql.stmt <- sprintf(sql.stmt, idString)
nl.smu <- dbGetQuery(con, sql.stmt)
nl.smu$HWSD2_SMU_ID <- as.factor(nl.smu$HWSD2_SMU_ID)
print(dim(nl.smu))

[1] 48 23

print(names(nl.smu))

 [1] "ID"                "HWSD2_SMU_ID"      "WISE30s_SMU_ID"
 [4] "HWSD1_SMU_ID"      "COVERAGE"          "SHARE"
 [7] "WRB4"              "WRB_PHASES"        "WRB2"
[10] "WRB2_CODE"         "FAO90"             "KOPPEN"
[13] "TEXTURE_USDA"      "REF_BULK_DENSITY"  "BULK_DENSITY"
[16] "DRAINAGE"          "ROOT_DEPTH"        "AWC"
```

```
[19] "PHASE1"            "PHASE2"            "ROOTS"
[22] "IL"                "ADD_PROP"
```

Second, the layers table:

```
sql.stmt <- "select * from HWSD2_LAYERS
             where HWSD2_LAYERS.HWSD2_SMU_ID in (%s)
             order by HWSD2_SMU_ID"
sql.stmt <- sprintf(sql.stmt, idString)
nl.layers <- dbGetQuery(con, sql.stmt)
nl.layers$HWSD2_SMU_ID <- as.factor(nl.layers$HWSD2_SMU_ID)
print(dim(nl.layers))

[1] 903  48

print(names(nl.layers))

 [1] "ID"               "HWSD2_SMU_ID"   "NSC_MU_SOURCE1"
 [4] "NSC_MU_SOURCE2" "WISE30s_SMU_ID" "HWSD1_SMU_ID"
 [7] "COVERAGE"        "SEQUENCE"       "SHARE"
[10] "NSC"             "WRB_PHASES"     "WRB4"
[13] "WRB2"            "FAO90"          "ROOT_DEPTH"
[16] "PHASE1"          "PHASE2"         "ROOTS"
[19] "IL"              "SWR"            "DRAINAGE"
[22] "AWC"             "ADD_PROP"       "LAYER"
[25] "TOPDEP"          "BOTDEP"         "COARSE"
[28] "SAND"            "SILT"           "CLAY"
[31] "TEXTURE_USDA"    "TEXTURE_SOTER"  "BULK"
[34] "REF_BULK"        "ORG_CARBON"     "PH_WATER"
[37] "TOTAL_N"         "CN_RATIO"       "CEC_SOIL"
[40] "CEC_CLAY"        "CEC_EFF"        "TEB"
[43] "BSAT"            "ALUM_SAT"       "ESP"
[46] "TCARBON_EQ"      "GYPSUM"         "ELEC_COND"
```

Replace negative values (i.e., missing) of SOC, coarse fragments and bulk density by `NA`, so they won't be used in calculations of SOC stock. Find these by the missing texture class.

```
length(v <- which(is.na(nl.layers$TEXTURE_USDA)))

[1] 24

nl.layers[v,c("BULK", "COARSE", "ORG_CARBON")] <- NA
```

There were 24 of 927 modified layers.

Now we have two R `data.frame`s. To begin the SOC stock calculation, do the per-layer computations, and record the results as a new field in the layers table. The units are Note that we know the layer thicknesses from the metadata, but these are also in two fields in the layers table.

```
# check the same map unit as shown with the SQL query above
nl.layers[nl.layers$HWSD2_SMU_ID==9355, c("SEQUENCE","SHARE","LAYER",
                                          "TOPDEP", "BOTDEP",
                                          "COARSE","ORG_CARBON","BULK")]

    SEQUENCE SHARE LAYER TOPDEP BOTDEP COARSE ORG_CARBON BULK
148        6     5    D1      0     20      9      0.628 1.44
149        6     5    D2     20     40     15      0.319 1.49
150        6     5    D3     40     60     16      0.237 1.51
151        6     5    D4     60     80     14      0.181 1.61
152        6     5    D5     80    100     13      0.118 1.62
153        6     5    D6    100    150     11      0.112 1.62
154        6     5    D7    150    200     11      0.104 1.60
155        4    10    D1      0     20      8      1.889 1.22
156        4    10    D2     20     40      7      0.909 1.29
157        4    10    D3     40     60     10      0.620 1.29
```

```
158        4    10    D4     60     80      12       0.416 1.36
159        4    10    D5     80    100       8       0.382 1.38
160        4    10    D6    100    150      11       0.342 1.49
161        4    10    D7    150    200       4       0.300 1.54
162        3    20    D1      0     20       3       1.384 1.52
163        3    20    D2     20     40       2       0.578 1.64
164        3    20    D3     40     60       2       0.379 1.70
165        3    20    D4     60     80       3       0.279 1.72
166        3    20    D5     80    100       3       0.204 1.73
167        3    20    D6    100    150       2       0.132 1.72
168        3    20    D7    150    200       4       0.090 1.74
169        2    30    D1      0     20      11       1.510 1.50
170        2    30    D2     20     40      11       0.631 1.60
171        2    30    D3     40     60      12       0.400 1.64
172        2    30    D4     60     80      14       0.298 1.66
173        2    30    D5     80    100      16       0.223 1.67
174        2    30    D6    100    150      16       0.151 1.67
175        2    30    D7    150    200      15       0.111 1.67
176        1    30    D1      0     20      10       7.326 1.13
177        1    30    D2     20     40      17       2.694 1.25
178        1    30    D3     40     60      20       1.790 1.28
179        1    30    D4     60     80      23       1.731 1.38
180        1    30    D5     80    100      26       1.376 1.57
181        1    30    D6    100    150      29       1.008 1.64
182        1    30    D7    150    200      29       0.391 1.83
183        5     5    D1      0     20       9       1.159 1.58
184        5     5    D2     20     40      23       0.558 1.66
185        5     5    D3     40     60      16       0.415 1.65
186        5     5    D4     60     80      14       0.170 1.66
187        5     5    D5     80    100      16       0.187 1.62
188        5     5    D6    100    150      19       0.389 1.77
189        5     5    D7    150    200      24       0.801 1.83

nl.layers$soc.wt <-
 # correct for coarse fragments proportion
    ((100 - nl.layers$COARSE)/100) *
 # SOC weight %, as a proportion
    (nl.layers$ORG_CARBON/100) *
# g soil cm^3 soil
    nl.layers$BULK *
# cm in depth slice
    (nl.layers$BOTDEP - nl.layers$TOPDEP)
```

Show the results of the per-layer calculation, and the source variable values:

```
tail(nl.layers$soc.wt)

[1] 0.1586880 0.1162800 0.1044810 0.0968513 0.2469240 0.2343600

print(nl.layers[nl.layers$HWSD2_SMU_ID==9355,
          c("SEQUENCE","SHARE","LAYER","COARSE","ORG_CARBON","BULK","soc.wt")])

    SEQUENCE SHARE LAYER COARSE ORG_CARBON BULK     soc.wt
148        6     5    D1      9      0.628 1.44 0.16458624
149        6     5    D2     15      0.319 1.49 0.08080270
150        6     5    D3     16      0.237 1.51 0.06012216
151        6     5    D4     14      0.181 1.61 0.05012252
152        6     5    D5     13      0.118 1.62 0.03326184
153        6     5    D6     11      0.112 1.62 0.08074080
154        6     5    D7     11      0.104 1.60 0.07404800
155        4    10    D1      8      1.889 1.22 0.42404272
156        4    10    D2      7      0.909 1.29 0.21810546
157        4    10    D3     10      0.620 1.29 0.14396400
158        4    10    D4     12      0.416 1.36 0.09957376
159        4    10    D5      8      0.382 1.38 0.09699744
160        4    10    D6     11      0.342 1.49 0.22676310
161        4    10    D7      4      0.300 1.54 0.22176000
162        3    20    D1      3      1.384 1.52 0.40811392
163        3    20    D2      2      0.578 1.64 0.18579232
```

```
164         3    20    D3       2       0.379 1.70 0.12628280
165         3    20    D4       3       0.279 1.72 0.09309672
166         3    20    D5       3       0.204 1.73 0.06846648
167         3    20    D6       2       0.132 1.72 0.11124960
168         3    20    D7       4       0.090 1.74 0.07516800
169         2    30    D1      11       1.510 1.50 0.40317000
170         2    30    D2      11       0.631 1.60 0.17970880
171         2    30    D3      12       0.400 1.64 0.11545600
172         2    30    D4      14       0.298 1.66 0.08508496
173         2    30    D5      16       0.223 1.67 0.06256488
174         2    30    D6      16       0.151 1.67 0.10591140
175         2    30    D7      15       0.111 1.67 0.07878225
176         1    30    D1      10       7.326 1.13 1.49010840
177         1    30    D2      17       2.694 1.25 0.55900500
178         1    30    D3      20       1.790 1.28 0.36659200
179         1    30    D4      23       1.731 1.38 0.36787212
180         1    30    D5      26       1.376 1.57 0.31972736
181         1    30    D6      29       1.008 1.64 0.58685760
182         1    30    D7      29       0.391 1.83 0.25401315
183         5     5    D1       9       1.159 1.58 0.33328204
184         5     5    D2      23       0.558 1.66 0.14264712
185         5     5    D3      16       0.415 1.65 0.11503800
186         5     5    D4      14       0.170 1.66 0.04853840
187         5     5    D5      16       0.187 1.62 0.05089392
188         5     5    D6      19       0.389 1.77 0.27885465
189         5     5    D7      24       0.801 1.83 0.55701540
```

Units are now g SOC cm^{-2} integrated over the depth slice.

Sum these for each map unit to get this integrated over the profile. At the same time, correct for the proportion of the component in the map unit, given by the `SHARE` field.

Split the dataset into separate map units, do the calculation, and then make a single vector of the results.

```r
nl.layers.s <- split(nl.layers, nl.layers$HWSD2_SMU_ID)
soc.stock <- unlist(lapply(nl.layers.s, function(x) {
    return(sum(x$soc.wt*(x$SHARE/100)))
    }
))
print(soc.stock)
```

```
      7001       7003       9345       9346       9347       9348 
        NA         NA  1.4159405  1.4403736  1.2066807  1.2651039 
      9349       9350       9351       9352       9353       9354 
 1.9956910  1.1213798  0.9588286  2.0098508  1.2167164  1.1126174 
      9355       9356       9357       9358       9359       9360 
 1.9527085  3.8232803  3.8232803  1.6476926 30.9367494 22.0878367 
      9361       9362       9363       9364       9365       9367 
18.4220668 10.1803390  1.4856812  2.6931841  1.2651039  1.7680601 
      9373       9375       9382       9386       9387       9388 
 1.0775377         NA  1.1892636  2.3630468  3.8332971  1.5890637 
     10184      10186      10187      10189      10215      10218 
 2.7516074 17.4031772 16.5212743  1.6053251  1.3391740  1.0846769 
     10241      10242      10243      10244      10245      10246 
30.9367494  2.6744411  3.8232803  1.7441507  1.2634550  2.2371359 
     10248      10249      10250      10251      32068      32069 
 1.1126174         NA  3.8232803  1.4159405  1.4877413  1.2066807 
```

These are in units of g cm^{-2} land area, integrated over the soil profile. Convert to the conventional unit of T ha^{-1} land area:

```r
# kg m^{-2} * 10 = T ha^{-1}
# 100 cm m^{-1}; 1000 g kg^{-1}
# 10000 m^2 ha^{-1}; 1000 kg T^{-1}
head(soc.stock.kg.m2 <- soc.stock * 100^2 / 10^3)
```

```
    7001      7003      9345      9346      9347      9348 
      NA        NA  14.15940  14.40374  12.06681  12.65104 

head(soc.stock.t.ha <- soc.stock.kg.m2 * 10^4 / 10^3)

    7001      7003      9345      9346      9347      9348 
      NA        NA  141.5940  144.0374  120.6681  126.5104 
```

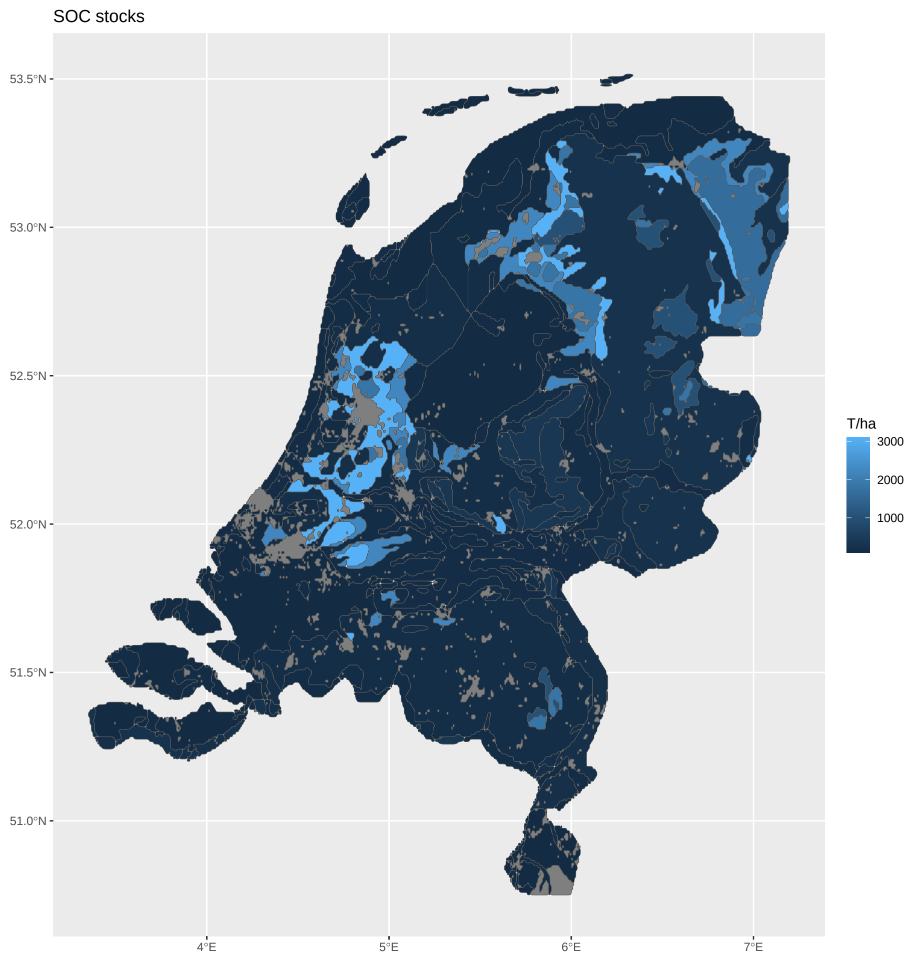
Add this to the map unit table. Even though that table only shows the dominant soil, its geometry represents all the components.

```
nl.smu$soc.stock <- soc.stock.t.ha
summary(nl.smu$soc.stock)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
  95.88  126.47  169.59  491.62  382.33 3093.67       4 
```

The very low values are map units with sandy or gravelly soils, the very high values are organic soils.

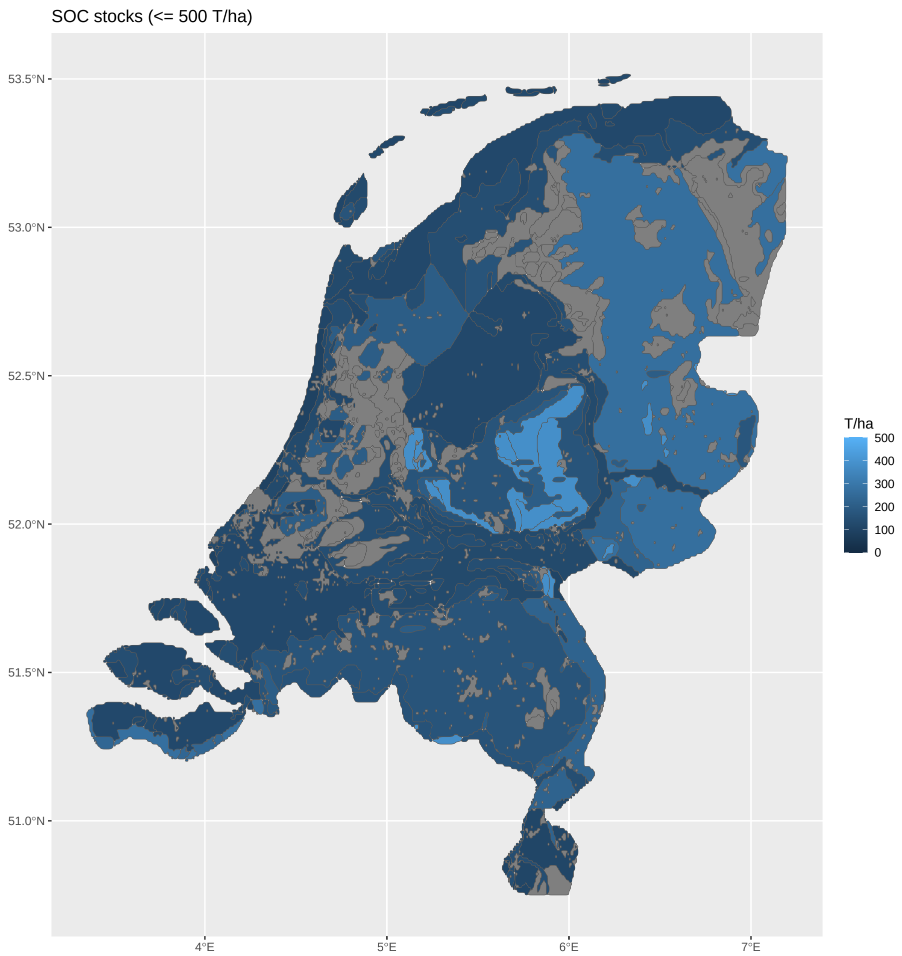
Reclassify the polygon map of the Netherlands to show the SOC stock.

```
hwsd.nl.poly <- as.polygons(hwsd.nl, dissolve = TRUE, values = TRUE)
hwsd.nl.sf <- st_as_sf(hwsd.nl.poly)
hwsd.nl.sf <- smooth(hwsd.nl.sf,  method = "ksmooth", smoothness = 1)
hwsd.nl.sf$soc.stock <- nl.smu$soc.stock
require(ggplot2)
ggplot(data = hwsd.nl.sf)  +
    geom_sf(aes(fill = soc.stock)) +
    labs(title = "SOC stocks") +
    labs(fill = "T/ha")
```



Because of the organic soils, the difference between map units with lower SOC stocks is not evident. So, repeat the figure with a changed scale.

```
ggplot(data = hwsd.nl.sf)  +
    geom_sf(aes(fill = soc.stock)) +
    lims(fill = c(0,500)) +
    labs(title = "SOC stocks (<= 500 T/ha)") +
    labs(fill = "T/ha")
```


SOC stocks (<= 500 T/ha)
53.5°N
53.0°N
52.5°N
52.0°N
51.5°N
51.0°N
4°E
5°E
6°E
7°E
T/ha
500
400
300
200
100
0

8 Cleanup

Task 36 : Remove temporary tables and disconnect the database. •

```
dbRemoveTable(con, "WINDOW_ZHNJ")
dbRemoveTable(con, "ZHNJ_SMU")
dbDisconnect(con)
```

A Extracting a window

Here is a function that can be used to extract any rectangular (longitude and latitude) window from the HWSD v2, using the techniques presented in this note. Files are written into a subdirectory under subdirectory `./window/`; these are created if necessary.

The function is provided as file `R_HWSD2_ExtractWindow.R`.

To use this function the HWSD v2 raster image and SQL database must be first loaded as follows:

```r
## define the functions
source("./R_HWSD2_ExtractWindow.R")

## read in HWSD raster database
require(sf)
require(terra)
hwsd <- rast("./HWSD_RASTER/hwsd.bil")

## establish connection to attribute database
require(RSQLite)
m <- dbDriver("SQLite")
con <- dbConnect(m, dbname="HWSD2.sqlite")

## call the function for each window we want to extract
### **** NOTE *** change the bounding box:
###      c(Long_WestEdge, Long_EastEdge, Lat_SouthEdge, Lat_NorthEdge)
### and also the name of the tile, according to your area
### this example is for the Southern Tier NY/Norther Tier PA (USA) counties
extract.window(c(-77, -75, 41, 43), "Twin_Tiers")

## clean up
dbDisconnect(con)
```

```
### Function to extract and format a rectangular window from the Harmonized World Soil Database V2
### Author: D G Rossiter   Version: 06-Apr-2023
### Arguments:
##   ext: a `terra'-style extent argument, a vector of xmin, xmax, ymin, ymax
##   name: a suffix for the file names; default `window` (image, UTM image, csv)
##        names start with "HWSD_", and area name
### the image `hwsd` and the SQLite database connection `con` must be available in the environment
### Side effects
##   writes raster image (un/projected, ESRI BIL), smoothed polygon (ESRI Shapefile),
##         and attribute data (CSV)
extract.window <- function(ext, name="window")
  {
    long2UTM <- function(long) { return(floor((long + 180)/6) + 1) %% 60 }
    print(paste0("Area name: ", name, "; bounding box: [",paste(ext,collapse=", "),"]"))
                                        # extract the window
    hwsd.win <- crop(hwsd, ext(ext))
                                        # find the zone for the centre of the box
    print(paste("Central meridian:", centre <- (ext[1] + ext[2])/2))
    print(paste("UTM zone:",  utm.zone <- long2UTM(centre)))
                                        # make a UTM version of the window
    hwsd.win.utm <- project(hwsd.win,
                     paste0("EPSG:",(epsg <- 32600 + utm.zone)), method="near")
                                        # write the raster images to disk
    eval(parse(text=paste0("writeRaster(hwsd.win, file='./HWSD_", name,
                   "', filetype='EHdr', overwrite=TRUE)")))
    eval(parse(text=paste0("writeRaster(hwsd.win.utm, file='./HWSD_", name,
                   "_utm', filetype='EHdr', overwrite=TRUE)")))
                                        # extract attributes for just this window

    dbWriteTable(con, name="WINDOW_TMP",
                 value=data.frame(HWSD2=unique(hwsd.win)), overwrite=TRUE)
    records <- dbGetQuery(con, "select T.* from HWSD2_SMU as T
                          join WINDOW_TMP as U on T.HWSD2_SMU_ID=U.HWSD2
                          order by HWSD2_SMU_ID")
    dbRemoveTable(con, "WINDOW_TMP")
                                        # convert to factors as appropriate
    for (i in names(records)[c(2:5,7:13,16:17,19:23)]) {
        eval(parse(text=paste0("records$",i," <- as.factor(records$",i,")"))) }
                                        # remove all-NA fields
    fields.to.delete <- NULL
    for (i in 1:length(names(records))) {
        if (all(is.na(records[,i]))) { fields.to.delete <- c(fields.to.delete, i) }}
    if (length(fields.to.delete > 1)) records <- records[,-fields.to.delete]
    print(paste0("Dimensions of attribute table: ",
                paste(dim(records), collapse=", "),
                " (records, fields with data)"))
                                        # write attribute table in CSV formats
    eval(parse(text=paste0("write.csv(records, file='./HWSD_", name, ".csv')")))
                                       # polygonize the raster
    hwsd.win.poly <- as.polygons(hwsd.win, dissolve = TRUE, values = TRUE)
                                       # transform to UTM for correct geometry
    hwsd.win.poly.utm <- project(hwsd.win.poly, paste0("EPSG:", epsg))
                                       # smooth the polygons
    require(smoothr)
    hwsd.win.sf.utm <- st_as_sf(hwsd.win.poly.utm)

    hwsd.win.sf.utm.smooth <- smooth(hwsd.win.sf.utm,
                                     method = "ksmooth", smoothness = 1)
    hwsd.win.poly.utm.smooth <- vect(hwsd.win.sf.utm.smooth)
    fn <- paste0("./HWSD_", name)
    if (dir.exists(fn)) unlink(fn, recursive = TRUE)
    writeVector(hwsd.win.poly.utm.smooth, file=fn, filetype='ESRI Shapefile',
                overwrite = TRUE)
  } # end extract.window
```

B Extracting a country

Here is a function that can be used to extract any rectangular window from the HWSD v2, using the techniques presented in this note. The country name is as given in the CIA world database; this was explained in §3.2. Files are written into a subdirectory `./country/<country name>`; this is created if necessary.

The function is provided as file `R_HWSD2_ExtractCountry.R`.

To use this function the HWSD v2 raster image and SQL database must be first loaded as follows:

```
## define the functions
source("./R_HWSD2_ExtractCountry.R")

## read in HWSD raster database
require(sf)
require(terra)
hwsd <- rast("./HWSD_RASTER/hwsd.bil")

## establish connection to attribute database
require(RSQLite)
m <- dbDriver("SQLite")
con <- dbConnect(m, dbname="HWSD2.sqlite")

## call the function for each country we want to extract
## *** Note *** replace this with the official name of the country you want
##  this name must match the CIA database, see help(worldHires)
##  in the `mapdata' package
extract.country('Netherlands')

## clean up
dbDisconnect(con)
```

```
### Function to extract and format country (CIA definition) window from the HWSD V2
### Author: D G Rossiter Version: 09-Apr-2023
### Arguments
##   name: a country name, to extract the appropriate bounding polygon(s)
##           this name must match the CIA database, see help(worldHires) in the `mapdata' package
##         this will also be used a suffix for the file names (image, csv attributes)
##         names start with "HWSD_<country>_", and area name
### the image `hwsd` and the SQLite database connection `con` must be available in the environment
### Side effects
##   writes raster image (un/projected, ESRI BIL), smoothed polygon (ESRI Shapefile),
##          and attribute data (CSV)
extract.country <- function(name="unknown") {
    long2UTM <- function(long) { return(floor((long + 180)/6) + 1) %% 60 }
    print(paste("Country:", name))
                                        # packages for country boundaries
    require(maps); require(mapdata)
    tryCatch( country <- map('worldHires',name, fill=TRUE, plot=FALSE),
             silent = TRUE,
             error = function(e) { print("No such country name"); return() } )
    boundary <- st_as_sf(country, IDs = country$names,
                 proj4string=
                   CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
    extent <- ext(boundary); poly <- as.polygons(vect(boundary))
                                        # extract the window
    hwsd.box <- crop(hwsd, ext(poly)); hwsd.country <- mask(hwsd.box, poly)
    print(paste("Central meridian:", centre <- (extent[1] + extent[2])/2))
    print(paste("UTM zone:",  utm.zone <- long2UTM(centre)))
                                        # make a UTM version of the country
    hwsd.country.utm <- project(hwsd.country,
                     paste0("EPSG:",(epsg <- 32600 + utm.zone)), method="near")
                                        # write un/projected raster images
    eval(parse(text=paste0("writeRaster(hwsd.country, file='./HWSD_country_", name, "',
                           filetype='EHdr', overwrite=TRUE)")))
    eval(parse(text=paste0("writeRaster(hwsd.country.utm, file='./HWSD_", name,
                   "_utm', filetype='EHdr', overwrite=TRUE)")))
                                        # extract attributes for this window
    dbWriteTable(con, name="WINDOW_TMP",
                 value=data.frame(HWSD2=unique(hwsd.country)), overwrite=TRUE)
    records <- dbGetQuery(con, "select T.* from HWSD2_SMU as T
                          join WINDOW_TMP as U on T.HWSD2_SMU_ID=U.HWSD2
                          order by HWSD2_SMU_ID")
    dbRemoveTable(con, "WINDOW_TMP")
                                        # convert to factors as appropriate
    for (i in names(records)[c(2:5,7:13,16:17,19:23)]) {
        eval(parse(text=paste0("records$",i," <- as.factor(records$",i,")")))}
                                        # include all fields
    print(paste0("Dimensions of attribute table: ",
                paste(dim(records), collapse=", "), " (records, fields)"))
                                        # write attribute table in CSV format
    eval(parse(text=paste0("write.csv(records, file='./HWSD_attributes_",
                          name, ".csv')")))
                                        # polygonize the raster
    hwsd.country.poly <- as.polygons(hwsd.country, dissolve = TRUE, values = TRUE)
                                        # transform to UTM for correct geometry
    hwsd.country.poly.utm <- project(hwsd.country.poly, paste0("EPSG:", epsg))
                                       # smooth the polygons
    require(smoothr)
    hwsd.country.sf.utm <- st_as_sf(hwsd.country.poly.utm)
    hwsd.country.sf.utm.smooth <- smooth(hwsd.country.sf.utm,
                                     method = "ksmooth", smoothness = 1)
    hwsd.country.poly.utm.smooth <- vect(hwsd.country.sf.utm.smooth)
    fn <- paste0("./HWSD_", name)
    if (dir.exists(fn)) unlink(fn, recursive = TRUE)
    writeVector(hwsd.country.poly.utm.smooth, file=fn, filetype='ESRI Shapefile',
                overwrite = TRUE)
    } # end extract.country
```

References

[1] FAO; IIASA. *Harmonized World Soil Database version 2.0*. FAO; International Institute for Applied Systems Analysis (IIASA);, Rome; Laxenburg, January 2023. ISBN 978-92-5-137499-3. doi: 10.4060/cc3823en. URL `http://www.fao.org/documents/card/en/c/cc3823en`. 1, 28

[2] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2023. URL `http://www.R-project.org/`. 1

[3] R Development Core Team. *R Data Import/Export*. The R Foundation for Statistical Computing, version 4.2.3 (2023-03-15) edition, 2017. URL `http://cran.r-project.org/doc/manuals/R-data.pdf`. 1

[4] Yihui Xie. knitr: Elegant, flexible and fast dynamic report generation with R, 2023. URL `http://yihui.name/knitr/`. 2

Index of Commands