Technical note
Processing the Harmonized World Soil Database (Version 1.2) in R

D G Rossiter / 罗大伟
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

August 10, 2017

Contents

1 Importing the HWSD into R **2**

2 Selecting a region **3**
2.1 Selecting by a bounding box 3
2.2 Selecting by a bounding polygon 7

3 Attribute database **10**

4 Raster attribute maps **17**

5 Polygon maps **18**
5.1 Raster to polygon 18
5.2 Polygon attribute maps 20

6 Map units with multiple components **20**

7 Cleanup **27**

A Extracting a window **28**

B Extracting a country **30**

References **32**

Version 1.4. Copyright 2012–2014, 2017 © D G Rossiter. All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (d.g.rossiter@cornell.edu).

Index of R concepts 33

This note attempts to explain how to access and query the Harmonized World Soil Database (HWSD) [3] using the open-source R project for statistical computing [7]. This allows integration of the HWSD with any other geographic coverage, as well as statistical summaries.

This note shows how to:

1. Access the HWSD at IIASA and import it to R;
2. Select a geographic window from the HWSD, either by a rectangular bounding box or a boundary polygon(s);
3. Project from the original Plate Carrée (non)projection to the UTM coordinate reference system;
4. Determine the area covered by each soil class;
5. Save the window in the original HWSD format and as a projected raster;
6. Link the attribute database to the raster and save the records for the window as either a CSV or Excel file;
7. Convert from the original raster format to polygons;
8. Create and display attribute raster and polygon maps.

There is certainly more that can be done in R with the HWSD[1], including integration with other freely-available geographic layers such as digital elevation models and satellite imagery. Readers are referred to the excellent textbook of Bivand et al. [1] from the UseR! Springer textbook series.

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. This is possible with the `RODBC` "R interface with Open Database Connectivity" package[2]; however I did not know this at the time I first developed these notes. I chose therefore to use another database format, SQL databases. These are explained in §3. I exported the MS-Access database (44.6 Mb) to SQLite format (19.7 Mb) using the MDB Explorer program[3] on OS X. There are similar programs available for other platforms.

> **Note:** I would welcome an adaptation of these notes to work directly with the Access database using `RODBC`. If you are interested in doing this, please contact me.

The procedures in this note use important R packages, including `sp` for spatial data [1, 6], `rgdal` for spatial data import, export and geometric transformation [4], `raster` for working with large raster (grid) image [2], and `RSQLite` for working with the SQLite format relational databases. These must loaded before their first use, as is shown in the code.

[1] and maybe will be, in later versions

[2] http://cran.r-project.org/web/packages/RODBC/index.html

[3] http://www.mdbexplorer.com/

Note: The code in this document was tested with R version 3.4.0 (2017-04-21) and packages from that version or later running on Mac OS X 10.7.5. 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: 1.16 [11] on R to and automatically generated and incorporated into LaTeX. Then the LaTeX document was compiled into the PDF version you are now reading. The R code (file `R_HWSD.R`, supplied with this document) was also generated by `knitr` from the same source document. 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.

1 Importing the HWSD into R

Task 1 : Download the HWSD database from IIASA. •

The HWSD is found at IIASA[4]. We do not use the HSWD Viewer, instead, we download the data for use in R. Three files are provided (Table 1):

Task 2 : Uncompress the compressed file `HWSD_RASTER.zip`. •

This will create a subdirectory `HWSD_RASTER` with three files: the band-interleaved image (`hwsd.bil`, 1.7 Gb), a small file giving the extent and resolution (`hwsd.blw`), and the header (`hwsd.hdr`). The latter two are automatically consulted on data import.

Task 3 : Import the world raster image to R. •

The `raster` 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 `raster` function associates an R object name with the file on disk. The band-interleaved format is known to this command. The `raster` package depends on the `sp` package, which

[4] `http://webarchive.iiasa.ac.at/Research/LUC/External-World-soil-database/`

file name	contents	format	size
`HWSD_raster.zip`	Raster soil unit map	band-interleaved image (`.bil`, `.blw`, `.hdr`)	19.7 Mb
`HWSD.mdb`	Soil attribute database	MS Access (`.mdb`)	44.6 Mb
`HWSD_META.mdb`	Soil attribute metadata	MS Access (`.mdb`)	0.8 Mb

Table 1: HWSD database files

it automatically loads if needed. We load the `raster` package with the `require` function, which only loads a package if it's not already in the workspace:

```
> require(sp)
> require(raster)
> hwsd <- raster("./HWSD_RASTER/hwsd.bil")
```

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

The `raster` package provides some useful commands for this, which are self-explanatory:

```
> ncol(hwsd); nrow(hwsd); res(hwsd); extent(hwsd); projection(hwsd)

[1] 43200
[1] 21600
[1] 0.008333333 0.008333333
class       : Extent
xmin        : -180
xmax        : 180
ymin        : -90
ymax        : 90
[1] NA
```

This raster is not provided with any projection information (reported as `NA`, "not available"). We know from the documentation [3] that this is a Plate Carrée[5] projection using the WGS84 datum; it just maps latitude and longitude directly to a grid cell, so that the figure is increasingly distorted towards the poles.

Task 5 : Provide the projection information for the raster database. •

This is a very simple "projection"; we use the `proj4string` function, which uses the syntax of the PROJ4 projection system [5]. We provide a "projection" (here, none, i.e., use the geographic coordinates), the datum, elipse, and translation to WGS84 (yes, all three are needed, and the datum name must be in upper case: `WGS84`):

```
> require(rgdal)
> (proj4string(hwsd) <-"+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")

[1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

2 Selecting a region

The entire database is very large; usually we want to work in some region.

2.1 Selecting by a bounding box

The `raster` package can crop an image to an "extent". This can be extracted from the bounding box of any `sp` object, or directly specified using the `extent` function. Here we will select a 2° by 2° tile centred near Nanjing, Jiangsu, China, and covering parts of Jiangsu and Anhui

[5] French: "square plate"

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, extent(c(117.5, 119.5, 31, 33)))
> nrow(hwsd.zhnj); ncol(hwsd.zhnj); bbox(hwsd.zhnj)

[1] 240
[1] 240
     min   max
s1 117.5 119.5
s2  31.0  33.0
```

The `unique` function shows the unique values in a raster:

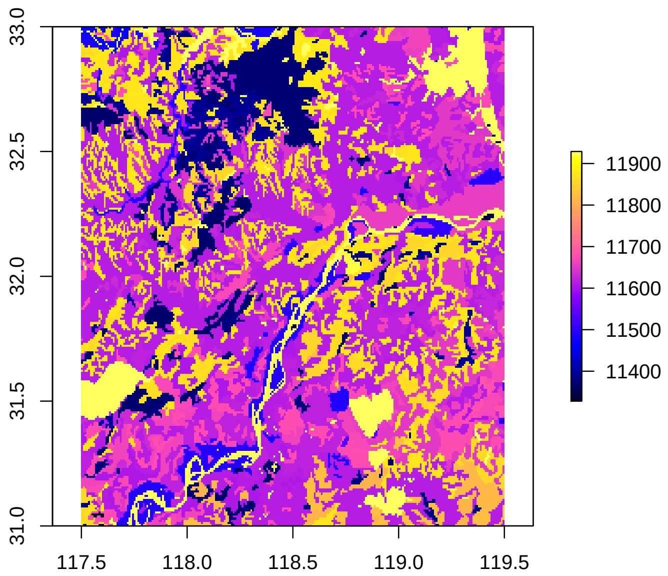
```
> unique(hwsd.zhnj)

 [1] 11328 11331 11341 11365 11367 11368 11372 11373 11375 11376
[11] 11377 11379 11381 11389 11390 11391 11392 11394 11434 11435
[21] 11460 11461 11466 11472 11474 11476 11481 11483 11485 11486
[31] 11488 11489 11490 11491 11492 11493 11495 11499 11501 11513
[41] 11535 11604 11605 11609 11613 11614 11615 11616 11617 11619
[51] 11620 11621 11623 11625 11627 11630 11634 11645 11649 11650
[61] 11651 11652 11655 11656 11657 11661 11663 11665 11667 11668
[71] 11671 11672 11673 11675 11677 11678 11679 11680 11814 11815
[81] 11817 11818 11823 11834 11857 11858 11859 11860 11863 11870
[91] 11875 11876 11877 11878 11925 11927 11928 11929
```

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

Task 6 : Display the tile with a suitable colour scheme. •

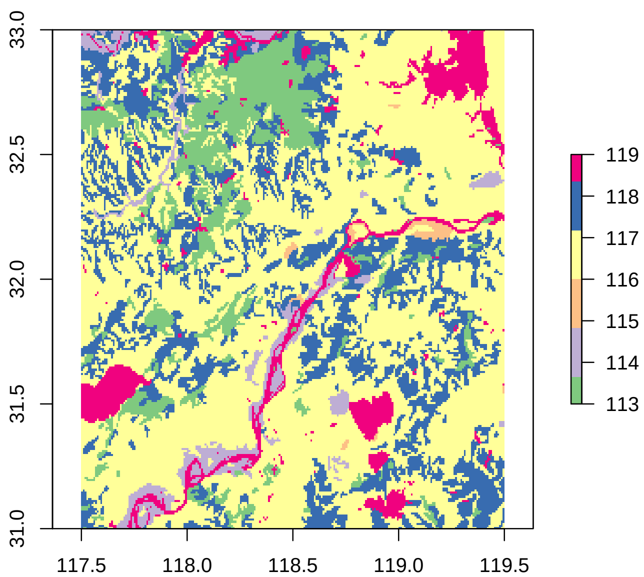
There are too many classes (98) to show with distinct colours. One way is to use a continuous colour ramp:

```
> plot(hwsd.zhnj, col=bpy.colors(length(unique(hwsd.zhnj))))
```



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:

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

     value count
[1,]   113  5919
[2,]   114  2152
[3,]   115   273
[4,]   116 32591
[5,]   118 12536
[6,]   119  4129

> require(RColorBrewer)
> plot(hwsd.zhnj3, col=brewer.pal(length(unique(hwsd.zhnj3)),"Accent"))
```

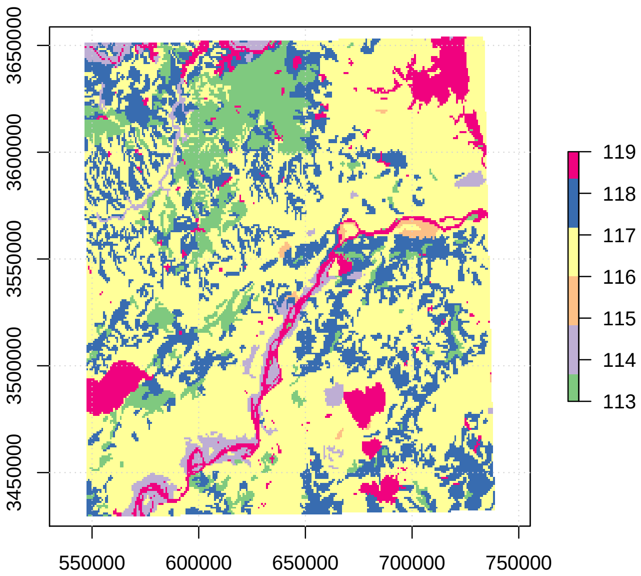


This image is distorted from geographic reality, because it is not projected. We can see the effect of projection, using the projectRaster method and specifying a target coordinate reference system (CRS). Note that we use the nearest-neighbour resampling (method="ngb") since this is a classified map.

We first determine the appropriate UTM zone for the centre of the window, recalling that UTM zone 30 is centred on 3° E.

```
> print(paste("UTM zone:", utm.zone <-
+             floor((sum(bbox(hwsd.zhnj3)[1, ])/2 + 180)/6) + 1))

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

```
> proj4string.utm50 <-
+     paste("+proj=utm +zone=", utm.zone,
+           "+datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0",
+           sep="")
> hwsd.zhnj3.utm <- projectRaster(hwsd.zhnj3, crs=proj4string.utm50,
+                                 method="ngb")
> unique(hwsd.zhnj3.utm)

[1] 113 114 115 116 118 119

> (cell.dim <- res(hwsd.zhnj3.utm))

[1] 787 924

> paste("Cell N dimension is ", round(((cell.dim[2]/cell.dim[1]) - 1)*100,1),
+       "% larger than cell dimension E", sep="")

[1] "Cell N dimension is 17.4% larger than cell dimension E"

> plot(hwsd.zhnj3.utm, col=brewer.pal(6,"Accent"), asp=1)
> grid()
```



Notice how the region is now longer in the N–S direction (as shown by the results of the `res` function, above); at 32° N a degree of latitude is larger than a degree of longitude. 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.

Now that this is geometrically-correct, we can compute the area covered by each code, and the total area of the tile, here in km^2:

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

[1] 72.7188
```

```
> (tmp <- cbind(freq(hwsd.zhnj3.utm)[,1],freq(hwsd.zhnj3.utm)[,2]*cell.area/10^2))

     [,1]       [,2]
[1,]  113  4280.2286
[2,]  114  1570.7261
[3,]  115   198.5223
[4,]  116 23744.1426
[5,]  118  9115.3016
[6,]  119  2999.6505
[7,]   NA  5189.9408

> ix <- which(is.na(tmp[,1]))
> sum(tmp[-ix,2])

[1] 41908.57

> rm(cell.dim, cell.area, tmp, ix)
```

The area of a grid cell is about 72 ha; at the equator this would be about 100 ha (1 km^2). The tile covers almost 42 000 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:

```
> rm(hwsd.zhnj3.utm)
```

Back to the unprojected image, 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 to the east of downtown Nanjing (118° 50' 30" E, 32° 04' 20" N according to Google Earth). The `click` function has optional arguments, which we use, to return the raster attribute value:

```
> plot(hwsd.zhnj, col=bpy.colors(length(unique(hwsd.zhnj))))
> xy <- click(hwsd.zhnj, n=1, id=TRUE, xy=TRUE, type="p")
```

```
> print(xy)

         x          y       hwsd
  118.8292    32.0875 11376.0000

> (zjs.id <- xy["hwsd"])

 hwsd
11376

> rm(xy)
```

The coordinates are in decimal degrees. The result is the soil map unit code of the pixel.

2.2 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.

Task 7 : Make a `SpatialPolygons` object from the boundary of the

Kingdom of Bhutan. •

We obtain the boundaries of Bhutan 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[6]. We extracted the boundaries with the `map` function, and then converted them to a `SpatialPolygons` object with the `map2SpatialPolygons` function of the `maptools` package.

> **Note:** These appear to be the boundaries claimed by the country in question as of that date; in the case of Bhutan it appears to include some small border regions also claimed by the People's Republic of China. We do not resolve border disputes, just use this convenient data source to build a bounding polygon.

> **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('worldHires','Bhutan', fill=TRUE, plot=FALSE))

List of 4
 $ x     : num [1:1666] 91.7 91.7 91.7 91.7 91.7 ...
 $ y     : num [1:1666] 27.8 27.8 27.8 27.8 27.8 ...
 $ range: num [1:4] 88.8 92.1 26.7 28.3
 $ names: chr "Bhutan"
 - attr(*, "class")= chr "map"

> require(maptools)
> bhutan.boundary <-
+     map2SpatialPolygons(tmp, IDs=tmp$names,
+      proj4string=
+      CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
> bbox(bhutan.boundary)

       min      max
x 88.75082 92.11529
y 26.70138 28.32526

> class(bhutan.boundary)

[1] "SpatialPolygons"
attr(,"package")
[1] "sp"

> rm(tmp)
```

Task 8 : Extract the portion of the HWSD database within this polygon as a raster window. •

Working with a `RasterLayer` object we can only extract rectangular areas. So first we use the bounding box of Bhutan to extract Bhutan and some areas of neighbouring countries, using the `crop` function. We then convert the rectangular window to a `SpatialPixelsDataFrame` object as required by the `sp` package. At that point we can use `over` to find the pixels in the country. This command returns `NA` for pixels outside the

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

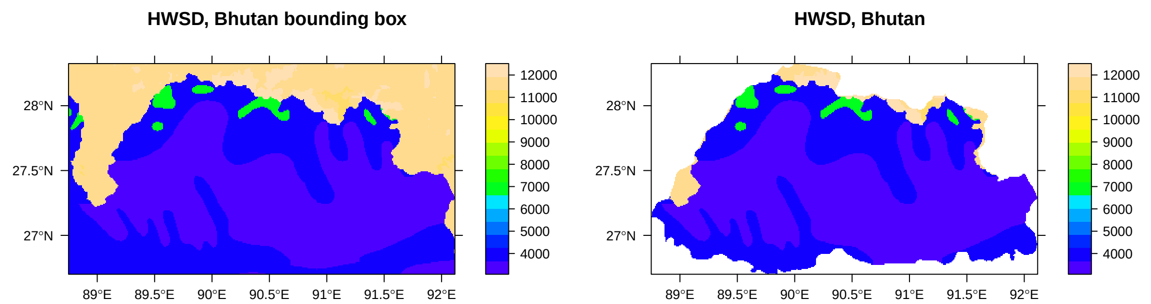
polygon, and the (single) polygon ID for pixels inside. We then select the pixels that are not NA, i.e., have a code.

```
> hwsd.bhutan.box <- crop(hwsd, bbox(bhutan.boundary))
> hwsd.bhutan.box.sp <- as(hwsd.bhutan.box, "SpatialPixelsDataFrame")
> sort(unique(hwsd.bhutan.box.sp$hwsd))

 [1]  3650  3651  3662  3683  3717  3821  3849  3850  6998 11000
[11] 11004 11052 11103 11335 11378 11388 11404 11413 11423 11535
[21] 11540 11705 11710 11711 11718 11719 11721 11724 11727 11730
[31] 11732 11736 11740 11748 11750 11752 11754 11758 11759 11765
[41] 11775 11790 11814 11839 11864 11879 11909 11927 11930 11932

> ix <- over(hwsd.bhutan.box.sp, bhutan.boundary)
> hwsd.bhutan.sp <- hwsd.bhutan.box.sp[!is.na(ix),]
> (bhutan.id <- sort(unique(hwsd.bhutan.sp$hwsd)))

 [1]  3651  3662  3717  3821  3849  6998 11052 11103 11705 11710
[11] 11718 11719 11724 11727 11730 11740 11750 11765 11839 11864
[21] 11879 11909 11930
```

There are only 23 different soil map units in Bhutan.

We display maps with the `spplot` method; the `scales` argument here specifies that we want the axes to be drawn.

```
> spplot(hwsd.bhutan.box.sp, main="HWSD, Bhutan bounding box",
+        col.regions=topo.colors(64), scales=list(draw = TRUE))
> spplot(hwsd.bhutan.sp, main="HWSD, Bhutan",
+        col.regions=topo.colors(64), scales=list(draw = TRUE))
```



This can be converted back to a `RasterLayer` object:

```
> hwsd.bhutan <- as(hwsd.bhutan.sp, "RasterLayer")
> rm(hwsd.bhutan.box, hwsd.bhutan.box.sp, ix)
```

3 Attribute database

There is no R package to read Access databases (file extension `.mdb`). However, R can work with SQL databases[7]; one option is the `RSQLite` package, which provides the interface to an SQL database via the `DBI` package. The author has exported the Access database to SQLite format[8], with file name `HWSD.sqlite`.

> **Note:** As an additional benefit, SQLite databases require much less disk storage than MS-Access databases with the same contents; in this case 12 Mb instead of 44.6 Mb, almost a four-fold reduction!

Task 9 : Connect to the SQLite version of the HWSD attribute database 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.

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

 [1] "D_ADD_PROP"        "D_AWC"             "D_COVERAGE"
 [4] "D_DRAINAGE"        "D_IL"              "D_ISSOIL"
 [7] "D_PHASE"           "D_ROOTS"           "D_SWR"
[10] "D_SYMBOL"          "D_SYMBOL74"        "D_SYMBOL85"
[13] "D_SYMBOL90"        "D_TEXTURE"         "D_USDA_TEX_CLASS"
[16] "HWSD_DATA"         "HWSD_SMU"          "WINDOW_BRETAGNE"
[19] "WINDOW_KH"
```

This database has 17 files, 16 of which are lookup tables of the attribute codes, while the remaining table `HWSD_DATA` is the list of map units.

Task 10 : Display the structure of the main table. •

SQL syntax used in SQLite is explained, with syntax diagrams, at the SQLite web page[9]. This language is not immediately intuitive; the reader who is unfamiliar with it is encouraged to follow a tutorial[10] to understand its principles.

The `dbGetQuery` function requires a database connection and a query string in SQL format. SQL uses the `PRAGMA` command to display database structure; we include it in the query string.

---

[7] see [8, §4] for a discussion of R and relational databases
[8] using the MDB Explorer program on OS X
[9] `http://www.sqlite.org/lang.html`
[10] For example, `http://www.w3schools.com/sql/`

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.

```
> dbGetQuery(con, "pragma table_info(HWSD_DATA)")$name

 [1] "ID"                  "MU_GLOBAL"
 [3] "MU_SOURCE1"          "MU_SOURCE2"
 [5] "ISSOIL"              "SHARE"
 [7] "SEQ"                 "SU_SYM74"
 [9] "SU_CODE74"           "SU_SYM85"
[11] "SU_CODE85"           "SU_SYM90"
[13] "SU_CODE90"           "T_TEXTURE"
[15] "DRAINAGE"            "REF_DEPTH"
[17] "AWC_CLASS"           "PHASE1"
[19] "PHASE2"              "ROOTS"
[21] "IL"                  "SWR"
[23] "ADD_PROP"            "T_GRAVEL"
[25] "T_SAND"              "T_SILT"
[27] "T_CLAY"              "T_USDA_TEX_CLASS"
[29] "T_REF_BULK_DENSITY" "T_BULK_DENSITY"
[31] "T_OC"                "T_PH_H2O"
[33] "T_CEC_CLAY"          "T_CEC_SOIL"
[35] "T_BS"                "T_TEB"
[37] "T_CACO3"             "T_CASO4"
[39] "T_ESP"               "T_ECE"
[41] "S_GRAVEL"            "S_SAND"
[43] "S_SILT"              "S_CLAY"
[45] "S_USDA_TEX_CLASS"    "S_REF_BULK_DENSITY"
[47] "S_BULK_DENSITY"      "S_OC"
[49] "S_PH_H2O"            "S_CEC_CLAY"
[51] "S_CEC_SOIL"          "S_BS"
[53] "S_TEB"               "S_CACO3"
[55] "S_CASO4"             "S_ESP"
[57] "S_ECE"

> dbGetQuery(con, "pragma table_info(HWSD_DATA)")$type

 [1] "INTEGER" "INTEGER" "TEXT"    "INTEGER" "INTEGER" "REAL"
 [7] "INTEGER" "TEXT"    "INTEGER" "TEXT"    "INTEGER" "TEXT"
[13] "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER"
[19] "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "INTEGER"
[25] "INTEGER" "INTEGER" "INTEGER" "INTEGER" "REAL"    "REAL"
[31] "REAL"    "REAL"    "REAL"    "REAL"    "REAL"    "REAL"
[37] "REAL"    "REAL"    "REAL"    "REAL"    "INTEGER" "INTEGER"
[43] "INTEGER" "INTEGER" "INTEGER" "REAL"    "REAL"    "REAL"
[49] "REAL"    "REAL"    "REAL"    "REAL"    "REAL"    "REAL"
[55] "REAL"    "REAL"    "REAL"
```

The field names, data types, and units of measure and lookup tables are explained in detail in [3, §2].

Task 11 : 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 operator. We select all records (by omitting a `where` clause).

```
> dbGetQuery(con, "select count(*) as grid_total from HWSD_DATA")

  grid_total
1      48148
```

Task 12 : Display the ID, map unit code, whether it is a soil unit or not, the percent in map unit, the FAO 1990 class code, 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","MU_GLOBAL","ISSOIL","SHARE","SU_CODE90",
+                      "SU_SYM90","T_USDA_TEX_CLASS"))

[1] "ID"                 "MU_GLOBAL"          "ISSOIL"
[4] "SHARE"              "SU_CODE90"          "SU_SYM90"
[7] "T_USDA_TEX_CLASS"

> tmp <- dbGetQuery(con, paste("select", paste(display.fields, collapse=", "),
+                              "from HWSD_DATA limit 10"))
```

```
> dim(tmp)

[1] 10  7

> print(tmp[,display.fields])

   ID MU_GLOBAL ISSOIL SHARE SU_CODE90 SU_SYM90 T_USDA_TEX_CLASS
1   1      7001      0   100       201       UR               NA
2   2      7002      0   100       202       HD               NA
3   3      7003      0   100       198       WR               NA
4   4      7004      0   100        89      HSf                3
5   5      7005      0   100       199       GG               NA
6   6      7006      1    70        35      ANz               11
7   7      7006      1    20        32      ANh               11
8   8      7006      1    10        37      ANi                9
9   9      7007      1    80        35      ANz               11
10 10      7007      1    20        32      ANh               11

> rm(tmp)
```

We see that some map units (e.g., 7001) are non-soil. Some map units (e.g., 7004) have only one component, others (e.g., 7006) have several, with their proportions.

Task 13 : Display the structure of the lookup table for FAO 1990 soil classes. •

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(dbGetQuery(con, "select * from D_SYMBOL90"))

'data.frame': 193 obs. of  3 variables:
 $ CODE  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ VALUE : chr  "FLUVISOLS" "Eutric Fluvisols" "Calcaric Fluvisols" "Dystric Fluvisols" ...
 $ SYMBOL: chr  "FL" "FLe" "FLc" "FLd" ...
```

Task 14 : 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.

```
> (tuple <- dbGetQuery(con, paste("select * from HWSD_DATA where MU_GLOBAL = ",
+                                 zjs.id)))

    ID MU_GLOBAL MU_SOURCE1 MU_SOURCE2 ISSOIL SHARE SEQ SU_SYM74
1 12184     11376      34200         NA      1   100   1     <NA>
  SU_CODE74 SU_SYM85 SU_CODE85 SU_SYM90 SU_CODE90 T_TEXTURE
1        NA     <NA>        NA      CMd        63         2
  DRAINAGE REF_DEPTH AWC_CLASS PHASE1 PHASE2 ROOTS IL SWR
1        4       100         1     NA     NA    NA NA  NA
  ADD_PROP T_GRAVEL T_SAND T_SILT T_CLAY T_USDA_TEX_CLASS
1        0       10     42     38     20                9
  T_REF_BULK_DENSITY T_BULK_DENSITY T_OC T_PH_H2O T_CEC_CLAY
1               1.41            1.3 1.45      5.1         32
  T_CEC_SOIL T_BS T_TEB T_CACO3 T_CASO4 T_ESP T_ECE S_GRAVEL
1         12   38   4.3       0       0     2   0.1       19
  S_SAND S_SILT S_CLAY S_USDA_TEX_CLASS S_REF_BULK_DENSITY
1     45     35     20                9               1.42
  S_BULK_DENSITY S_OC S_PH_H2O S_CEC_CLAY S_CEC_SOIL S_BS S_TEB
1           1.36  0.5      5.2         35          9   33   2.6
  S_CACO3 S_CASO4 S_ESP S_ECE
1       0       0     2   0.1

> tuple$SU_SYM90

[1] "CMd"
```

This is the code; we can find the corresponding name in the lookup table:

```
> dbGetQuery(con, paste("select * from D_SYMBOL90 where symbol='",
+                       tuple$SU_SYM90,"'",sep=""))

  CODE             VALUE SYMBOL
1   63 Dystric Cambisols    CMd
```

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

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

Task 15 : 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(smu_id=unique(hwsd.zhnj)),
+              overwrite=TRUE)
> dbGetQuery(con, "pragma table_info(WINDOW_ZHNJ)")
```

```
  cid   name    type notnull dflt_value pk
1   0 smu_id INTEGER       0         NA  0
```

Now we join on the common field; the new table does not contribute any new fields. We also show how to sort the results, in this case by the FAO 1990 soil map unit symbol:

> **Note:** The `select T.*` clause selects the fields from the `HWSD_DATA` 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 `HWSD_DATA` table has the same codes in field `MU_GLOBAL`.

```
> records <- dbGetQuery(con,
+                  "select T.* from HWSD_DATA as T
                  join WINDOW_ZHNJ as U on T.MU_GLOBAL=U.SMU_ID
                  order by SU_SYM90")
> dim(records)

[1] 98 57

> head(records)[,display.fields]

     ID MU_GLOBAL ISSOIL SHARE SU_CODE90 SU_SYM90
1 12469     11661      1   100        22     ACp
2 12479     11671      1   100        22     ACp
3 12622     11814      1   100        21     ACu
4 12623     11815      1   100        21     ACu
5 12625     11817      1   100        21     ACu
6 12626     11818      1   100        21     ACu
  T_USDA_TEX_CLASS
1                9
2                3
3               10
4               10
5               11
6                3

> sort(unique(records$SU_SYM90))

 [1] "ACp" "ACu" "ALf" "ALp" "ANh" "AT"  "ATc" "CMc" "CMd" "CMe"
[11] "CMo" "DS"  "FLc" "FLe" "GLe" "GLk" "GLm" "LP"  "LPd" "LPk"
[21] "LVh" "PLd" "PLe" "RGc" "RGd" "RGe" "UR"  "VRe" "WR"
```

In this window all the map units have only one component, as we can see from the SHARE field:

```
> unique(records$SHARE)

[1] 100
```

This was a decision by the compilers of the Chinese portion of the HWSD. See §6, below, for a window where some map units have multiple components.

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.

Task 16 : Convert fields to R factors as appropriate. •

```
> for (i in names(records)[c(2:5,8:15,17:19,28,45)])
+   {
+     eval(parse(text=paste("records$",i," <- as.factor(records$",i,")", sep="")))
+   }
```

> **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`.

We could assign the names for factor levels from the metadata lookup tables (not yet implemented).

Task 17 : Remove fields with no data from the window's attribute table.

•

Some fields are completely undefined in this window. For example, the MU_SOURCE2 field (second source of data) 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)=="MU_SOURCE2")
> all(is.na(records[,ix]))

[1] TRUE
```

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

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

[1] 98 57
[1] 98 48

> records <- df
> 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 for use in spreadsheets or to be imported into a database program.

Task 18 : Export the map unit table as a comma-separated values (CSV) file. •

The write.csv function does just that:

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

We can also write direct to Excel files with the write.xls function of the dataframes2xls package. This has the advantage that it correctly writes R factors as character variables, not as integers.

```
> require(dataframes2xls)
> write.xls(records, file="./HWSD_Nanjing.xls")
```

We can see the names of the map units with another table join. To do this, 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.

> **Note:** Here we use the dbExecute function instead of dbGetQuery, because our aim is not to return a data frame with a query, rather it is to create a temporary table.

```
> dbExecute(con,
+           "create table TMP as select * from HWSD_DATA as T
           join WINDOW_ZHNJ as U on T.MU_GLOBAL=U.SMU_ID
           order by SU_SYM90")

[1] 1

> head(window.fao90 <- dbGetQuery(con,
+           "select CODE, VALUE, SYMBOL from D_SYMBOL90 as U
          join TMP as T  on T.SU_CODE90=U.CODE"))

  CODE             VALUE SYMBOL
1   22 Plinthic Acrisols    ACp
2   22 Plinthic Acrisols    ACp
3   21    Humic Acrisols    ACu
4   21    Humic Acrisols    ACu
5   21    Humic Acrisols    ACu
6   21    Humic Acrisols    ACu

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

4 Raster attribute maps

The `raster` package is not suited to working with attribute databases linked to maps; instead the `sp` package is preferred.

Task 19 : Convert the HWSD window to a `SpatialGridDataFrame`, and add the attributes from the database. •

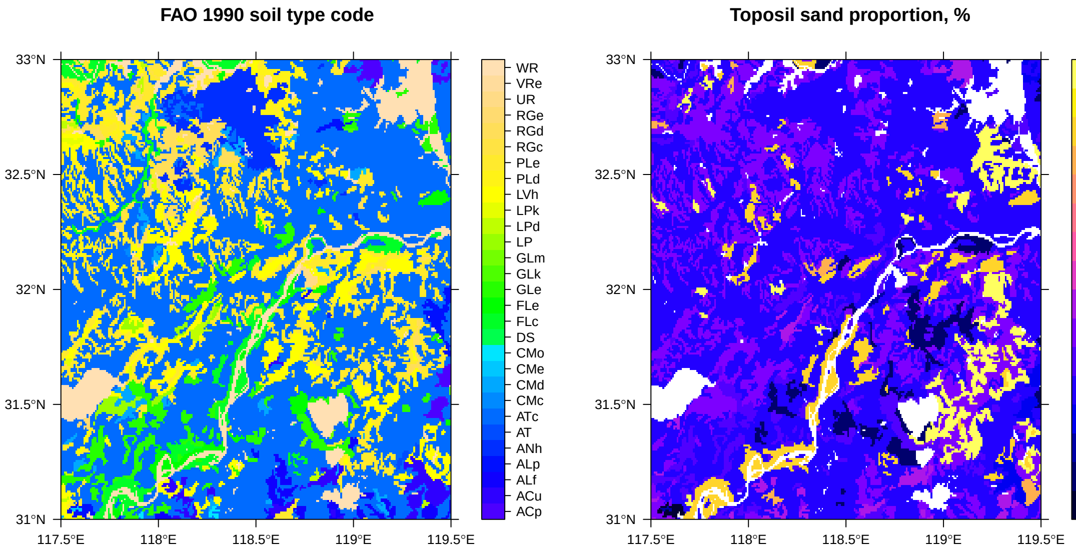
The `match` function finds the position of a given value in a lookup table. Here we match the SMU ID from the converted raster to the record in the attribute data frame. We then use that index to extract the proper record for each pixel, and add it to the dataframe. We then display two attribute maps: one categorical and one continuous.

```
> hwsd.zhnj.sp <- as(hwsd.zhnj, "SpatialGridDataFrame")
> str(hwsd.zhnj.sp@data)

'data.frame':	57600 obs. of  1 variable:
 $ hwsd: int  11466 11466 11466 11466 11466 11466 11466 11466 11875 11875 ...

> m <- match(hwsd.zhnj.sp@data$hwsd, records$MU_GLOBAL)
> str(m)

 int [1:57600] 54 54 54 54 54 54 54 54 87 87 ...

> hwsd.zhnj.sp@data <- records[m,]
> rm(m)
```

Task 20 : Display a map of the FAO 1990 soil types, and a map of the topsoil sand proportion. •

We do this with the `spplot` method, specifying the variable to be displayed with the `zcol` argument:

```
> spplot(hwsd.zhnj.sp, zcol="SU_SYM90",
+        col.regions=topo.colors(length(levels(hwsd.zhnj.sp$SU_SYM90))),
+        main="FAO 1990 soil type code", scales=list(draw = TRUE))
> spplot(hwsd.zhnj.sp, zcol="T_SAND", col.regions=bpy.colors(64),
+        main="Toposil sand proportion, %", scales=list(draw = TRUE))
```



5 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.

5.1 Raster to polygon

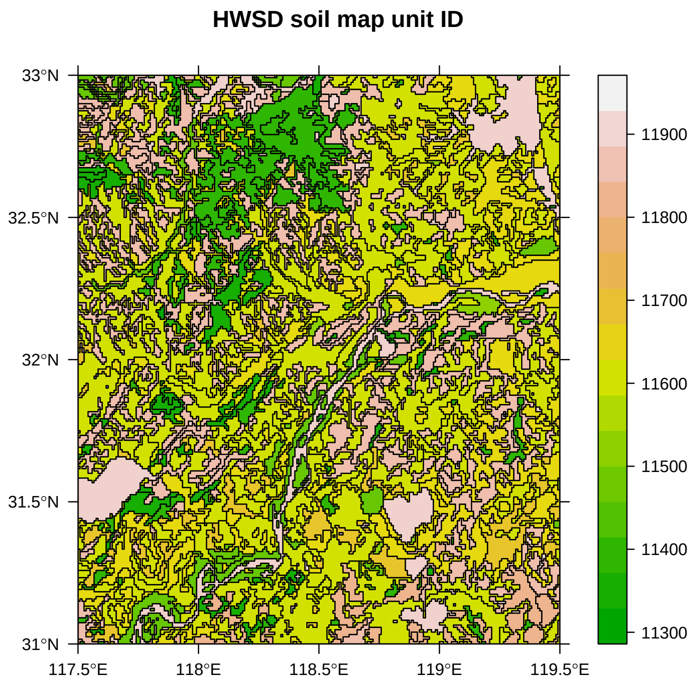
TASK 21 : Convert the raster image to a polygon map; each polygon should be labelled with the code of the contiguous pixels that make up the polygon. •

The `raster` package has a function `rasterToPolygons` for this; it depends on yet another package, `rgeos`, to dissolve boundaries between polygons with the same code.

Note: We time the complicated and slow raster-to-polygon operation with the `system.time` function. The conversion requires somewhat less than one minute on the author's system.

```
> require(rgeos)
> system.time(hwsd.zhnj.poly <-
+             rasterToPolygons(hwsd.zhnj, n=4,
+                              na.rm=TRUE, dissolve=TRUE))

   user  system elapsed
 27.740   0.268  30.992
```

```
> class(hwsd.zhnj.poly)

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

> str(hwsd.zhnj.poly@data)

'data.frame': 98 obs. of  1 variable:
 $ hwsd: num  11328 11331 11341 11365 11367 ...

> spplot(hwsd.zhnj.poly, col.regions=terrain.colors(64),
+        main="HWSD soil map unit ID",
+        scales=list(draw = TRUE))
```



There are only 98 map units (sets of polygons with the same code), as opposed to 57600 raster cells, a very large savings in memory and processing time.

Polygon maps with classes from the `sp` package 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 `spTransform` function of the `rgdal` package. This requires a target Coordinate Reference System (CRS), which is stored in the `proj4string` "PROJ.4 format CRS specification string" in all `sp` objects. We defined the appropriate UTM CRS (including elipsoid, datum and offset from the WGS84 elipsoid) for the UTM version of the raster image in §2.1, so we can extract the required CRS from the reprojected image.

```
> proj4string.utm50

[1] "+proj=utm +zone=50+datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"

> hwsd.zhnj.poly.utm <- spTransform(hwsd.zhnj.poly, CRS(proj4string.utm50))
```

5.2 Polygon attribute maps

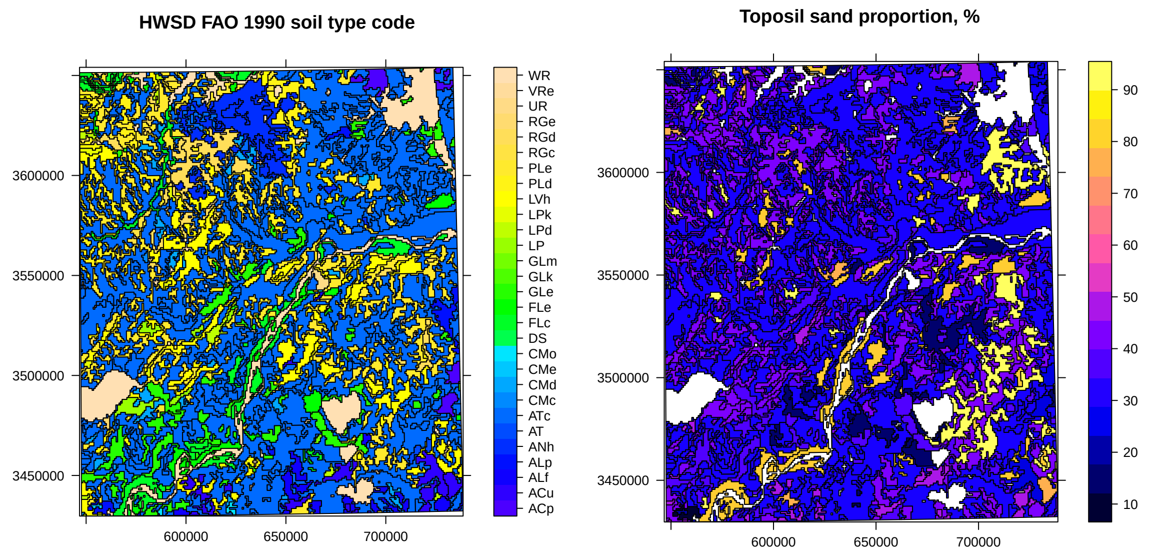
So far the polygons just have the soil map unit code.

Task 22 : Add the attribute database to the data frame, and display soil-type and topsoil sand proportion maps. •

The matching of attributes to codes is the same as in §4. The attribute name in the the polygon map is `hwsd`, this is then compared with attribute `MU_GLOBAL` in the data table:

```
> m <- match(hwsd.zhnj.poly.utm$hwsd, records$MU_GLOBAL)
> str(m)

 int [1:98] 42 46 50 76 79 80 11 12 43 44 ...

> hwsd.zhnj.poly.utm@data <- records[m,]
```

```
> spplot(hwsd.zhnj.poly.utm, zcol="SU_SYM90",
+        col.regions=topo.colors(length(levels(hwsd.zhnj.sp$SU_SYM90))),
+        main="HWSD FAO 1990 soil type code", scales=list(draw = TRUE))
> spplot(hwsd.zhnj.poly.utm, zcol="T_SAND",
+        col.regions=bpy.colors(64),
+        main="Toposil sand proportion, %", scales=list(draw = TRUE))
```



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

6 Map units with multiple components

By contrast to the Nanjing example used in the previous sections, in Bhutan some map units have multiple components.

Task 23 : Create a table with the HWSD records for Bhutan, with attributes, and display the HWSD ID, component ID, its share, and the FAO 1990 and 1974 classifications. •

This is exactly as was done for the Nanjing example:

```
> dbWriteTable(con, name="WINDOW_BHUTAN",
+              value=data.frame(smu_id=unique(bhutan.id)),
+              overwrite=TRUE)
> records.bhutan <-
+  dbGetQuery(con, "select T.* from HWSD_DATA as T
                  join WINDOW_BHUTAN as U on T.MU_GLOBAL=U.SMU_ID
                  order by SU_SYM90")
> dim(records.bhutan)

[1] 35 57

> unique(records.bhutan$MU_GLOBAL)

 [1]  3651  3662  3717  3821  3849  6998 11839 11750 11710 11718
[11] 11719 11740 11930 11103 11727 11705 11730 11765 11724 11864
[21] 11879 11909 11052

> unique(records.bhutan$SHARE)

[1]  40  20  10  60  25  70  30 100

> records.bhutan[,c("ID","MU_GLOBAL","SHARE","SU_SYM90","SU_SYM74","T_SAND")]

      ID MU_GLOBAL SHARE SU_SYM90 SU_SYM74 T_SAND
1  41792      3651    40     <NA>       Ao     49
2  41793      3651    20     <NA>       Ah     46
3  41794      3651    20     <NA>       Pl     49
4  41795      3651    10     <NA>       Bh     41
5  41796      3651    10     <NA>       Dd     31
6  41830      3662    60     <NA>       Bd     41
7  41831      3662    20     <NA>       Nd     44
8  41832      3662    20     <NA>       Rd     42
9  42048      3717    25     <NA>        I     43
10 42049      3717    25     <NA>       Bh     41
11 42050      3717    25     <NA>        U     48
12 42051      3717    25     <NA>       RK     NA
13 42362      3821    60     <NA>       Nd     22
14 42363      3821    20     <NA>       Bd     41
15 42364      3821    20     <NA>       Rd     42
16 42440      3849    70     <NA>       Rd     82
17 42441      3849    30     <NA>       Je     39
18 46656      6998   100     <NA>       GG     NA
19 12647     11839   100      ALh     <NA>     40
20 12558     11750   100      CMc     <NA>     36
21 12518     11710   100      CMi     <NA>     31
22 12526     11718   100      CMi     <NA>     31
23 12527     11719   100      CMi     <NA>     31
24 12548     11740   100      CMi     <NA>     31
25 12738     11930   100       GG     <NA>     NA
26 11911     11103   100      GRh     <NA>     25
27 12535     11727   100      LPe     <NA>     46
28 12513     11705   100      LPi     <NA>     56
29 12538     11730   100      LPi     <NA>     56
30 12573     11765   100      LPi     <NA>     56
31 12532     11724   100      LPm     <NA>     35
32 12672     11864   100      LVh     <NA>     41
33 12687     11879   100      LVh     <NA>     41
34 12717     11909   100      LVh     <NA>     41
35 11860     11052   100      LVk     <NA>     53
```

We can now see map units with multiple components. For example, map unit 3717 (records 42048–42051) has four components, each with 25% share; three have a reported topsoil sand concentration, but one (FAO

1974 symbol RK, "rock outcrop") has none.

> **Note:** This table also reveals different data sources: all the map units with only one component, except 6998, also are named from FAO 1990; these are from the portion of Bhutan claimed by China and so mapped by the Chinese; all the maps units with more than one component are named by FAO 1974 and are presumably from a reconaissance survey within Bhutan.

Task 24 : Clean up the records by converting to factors as appropriate and then removing empty fields; save the cleaned flat file in CSV and Excel formats. •

```
> for (i in names(records.bhutan)[c(2:5,8:15,17:19,28,45)])
+   {
+     eval(parse(text=paste("records.bhutan$",i,
+                          " <- as.factor(records.bhutan$",i,")", sep="")))
+   }
> ix <- which(names(records.bhutan)=="MU_SOURCE2")
> df <- records.bhutan
> for (i in 1:length(names(records.bhutan))) {
+     if (all(is.na(records.bhutan[,i]))) df <- df[-i]
+     }
> dim(records.bhutan); dim(df)

[1] 35 57
[1] 35 50

> records.bhutan <- df
> rm(df, ix, i)
```

```
> write.csv(records.bhutan,  file="./HWSD_Bhutan.csv")
> write.xls(records.bhutan,  file="./HWSD_Bhutan.xls")
```

Map units with more than one component create a problem for making raster attribute maps; the approach of §4 must be modified because more than one record (tuple) will match in the table join. There are several solutions to this problem.

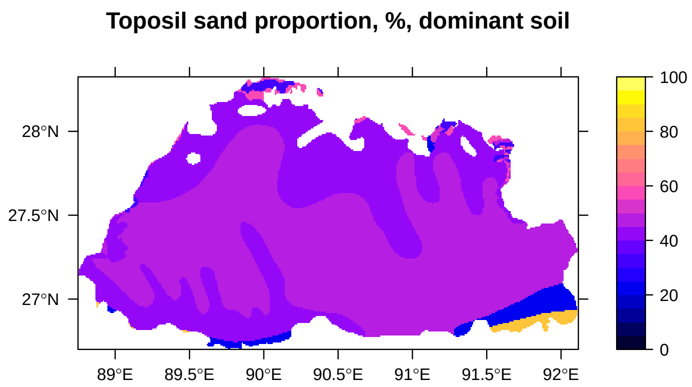
Using the `match` function will find the *first* match, i.e., the first-listed component, so then all attributes from a simple join will be for the first component only. For example, the topsoil sand content:

```
> hwsd.bhutan.sp <- as(hwsd.bhutan, "SpatialGridDataFrame")
> m <- match(hwsd.bhutan.sp@data$hwsd, records.bhutan$MU_GLOBAL)
> hwsd.bhutan.sp@data <- records.bhutan[m, ]
> summary(hwsd.bhutan.sp@data$T_SAND)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   22.0    43.0    49.0    46.1    49.0    82.0   28645
```

Comparing this to the full list of map unit components, we can see that only the first is listed, so for example the sand content is only for that component.

A map shows the attribute of the first-listed component:

```
> spplot(hwsd.bhutan.sp, zcol="T_SAND", col.regions=bpy.colors(64),
+        main="Toposil sand proportion, %, dominant soil",
+        scales=list(draw = TRUE),
+        at=seq(0, 100, 5))
```



There are several choices for computing one value for the pixel:

1. Accept the default from `match`, that is, the value for the first-listed component; since the records should be listed in descending order of `SHARE`, this is the dominant value. Note however that this ordering is not guaranteed.

2. Select a single value, either the highest, lowest, a quantile, or median, according to the application; for example, a map of groundwater pollution risk might want to select the component with the highest sand content;

3. Compute an average value weighted by proportion; this might be selected for a "best" value for land surface modelling.

For some of these choices there is an SQL "aggregate" operator: `MAX`, `MIN`, `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.

For example, to average the sand contents:

```
> dbExecute(con,
+           "create table TMP as select T.* from HWSD_DATA as T
           join WINDOW_BHUTAN as U on T.MU_GLOBAL=U.SMU_ID
           order by SU_SYM90")

[1] 1

> avg.sand <- dbGetQuery(con,
+             "select MU_GLOBAL, SU_SYM90,
             SU_SYM74, AVG(T_SAND) as T_SAND_AVG from TMP
             group by MU_GLOBAL")
> dim(avg.sand)
```

```
[1] 23  4

> print(avg.sand)

   MU_GLOBAL SU_SYM90 SU_SYM74 T_SAND_AVG
1       3651     <NA>       Dd   43.20000
2       3662     <NA>       Rd   42.33333
3       3717     <NA>       RK   44.00000
4       3821     <NA>       Rd   35.00000
5       3849     <NA>       Je   60.50000
6       6998     <NA>       GG         NA
7      11052      LVk     <NA>   53.00000
8      11103      GRh     <NA>   25.00000
9      11705      LPi     <NA>   56.00000
10     11710      CMi     <NA>   31.00000
11     11718      CMi     <NA>   31.00000
12     11719      CMi     <NA>   31.00000
13     11724      LPm     <NA>   35.00000
14     11727      LPe     <NA>   46.00000
15     11730      LPi     <NA>   56.00000
16     11740      CMi     <NA>   31.00000
17     11750      CMc     <NA>   36.00000
18     11765      LPi     <NA>   56.00000
19     11839      ALh     <NA>   40.00000
20     11864      LVh     <NA>   41.00000
21     11879      LVh     <NA>   41.00000
22     11909      LVh     <NA>   41.00000
23     11930       GG     <NA>         NA
```

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

Compare this table with the table of map units; it only has 23 entries, rather than 35, this because the map units with multiple components have been merged. For example, map unit 3717 had four entries, now only one; the topsoil sand contents $(43, 41, 48, \mathrm{NA})$ have been averaged to 44. This is not completely what we want: (1) although in this case the component proportions are equal, that is not in general true; (2) one of the components has no sand, so the average should also include this as an implicit zero. To get the correct weighted average, we would need to also extract the proportions and weight the sand contents.

One solution was provided to me by Ewen Gallic[11]. This is a nice illustration of two packages by Hadley Wickham[12], `dplyr` "dataframe pliers" for manipulating data [9] and `tidyr` for tidying data [10].

The `dplyr` package introduced the pipelining operator `%>%`, which passes the results from one `dplyr` function as the argument to another function in a natural way, similar to the Unix "pipe" operator `|`. You can break this long sequence of pipes down to a step-by-step operation if you are curious how it works.

1. We use the first pipe to pass the records for Bhutan to the `select` function, which extracts just the named columns. A detail here is that we have to explicitly name the package using the `::` package selection operator, because `select` is an overloaded function name, defined in several of our loaded packages.

[11] http://egallic.fr
[12] http://hadley.nz/

2. The column-reduced records are then passed to the `gather` function of the `tidyr` package. This function moves column names into a key column, gathering the column values into a single value column. In this case the `T_SAND` and `T_CLAY` values are put into a single column which we name `value`, and a new column is created to show which record of the new table corresponds to a sand or clay value; we specify `variable` as this column's name.

3. The next step is to use the `mutate` function to add a new variable to this "long format" table; here the variable is named `share_2` and is defined as either a 0 if the value in the new column is missing, otherwise the proportion of the component given by the `SHARE` field.

4. The records are then grouped by the map unit code `MU_GLOBAL`, using the `group_by` function. This does not change the table but does define groups for following operations.

5. We again `mutate` the grouped table to convert the `share_2` field from a percentage to a proportion, using the `sum` operator on the original value of that field. The same could have been done with the command `mutate(share_2) = share_2/100`.

6. The next step is to compute each component's contribution the weighted average, again using `mutate` and a formula multiplying the component's value by its proportion; this is the new value of the `value` field.

7. Then the components' contributions are summed with the `summarise` function; we have to specify how to summarize, and here it is the `sum` function. The `summarise` function works with the groups defined earlier by `group_by`, so the table is now reduced to one entry per map unit and variable combination.

8. The `ungroup` function removes the grouping.

9. Finally, the two variables are separated back into their own value columns with the `spread` function of the `tidyr` package; this is the inverse operation of the `gather` function.

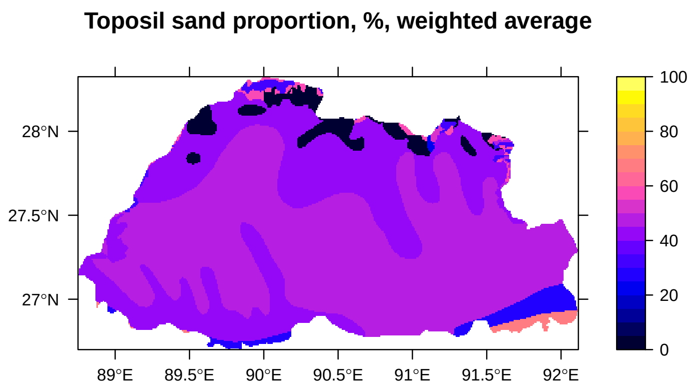
```
> library(dplyr)
> library(tidyr)
> bhutan.avg <- records.bhutan %>%
+   dplyr::select(MU_GLOBAL, SHARE, T_SAND, T_CLAY) %>%
+   gather(variable, value, -MU_GLOBAL, -SHARE) %>%
+   mutate(share_2 = ifelse(is.na(value), yes = 0, no = SHARE)) %>%
+   group_by(MU_GLOBAL, variable) %>%
+   mutate(share_2 = share_2 / sum(share_2)) %>%
+   mutate(value = value * share_2) %>%
+   summarise(value = sum(value, na.rm=TRUE)) %>%
+   ungroup() %>%
+   spread(variable, value)
> print(bhutan.avg)

# A tibble: 23 x 3
   MU_GLOBAL    T_CLAY  T_SAND
 *    <fctr>     <dbl>   <dbl>
 1      3651  20.80000    45.8
```

```
2       3662 20.80000   41.8
3       3717 22.33333   44.0
4       3821 41.20000   29.8
5       3849 11.60000   69.1
6       6998  0.00000    0.0
7      11052 23.00000   53.0
8      11103 21.00000   25.0
9      11705  6.00000   56.0
10     11710 20.00000   31.0
# ... with 13 more rows
```

To map this we again match with the raster grid, but this time there is only one match per map unit, with the average instead of the value from the first-listed unit:

```
> hwsd.bhutan.sp <- as(hwsd.bhutan, "SpatialGridDataFrame")
> m <- match(hwsd.bhutan.sp@data$hwsd, bhutan.avg$MU_GLOBAL)
> # summary(m)
> hwsd.bhutan.sp@data <- bhutan.avg[m, ]
> summary(hwsd.bhutan.sp@data$T_SAND)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00   44.00   45.80   42.72   45.80   69.10   26394

> summary(hwsd.bhutan.sp@data$T_CLAY)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00   20.80   20.80   20.58   22.33   41.20   26394
```

```
> spplot(hwsd.bhutan.sp, zcol="T_SAND", col.regions=bpy.colors(64),
+        main="Toposil sand proportion, %, weighted average",
+        scales=list(draw = TRUE),
+        at=seq(0, 100, 5))
```



This shows clear differences with the previous map based only on the dominant component.

7 Cleanup

Task 25 : Remove temporary tables and disconnect the database. •

```
> dbRemoveTable(con, "WINDOW_ZHNJ")
> dbRemoveTable(con, "WINDOW_BHUTAN")
> dbDisconnect(con)
```

A Extracting a window

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

```
## R script to extract rectangular windows from the Harmonized World Soil Database
## Author: D G Rossiter
## Version: 09-Aug-2017

##### initialize
rm(list=ls())

##### function to find a UTM zone
long2UTM <- function(long) {
    return(floor((long + 180)/6) + 1) %% 60
}

##### function to extract and format one rectangular window

## arguments:

##   bbox: a `raster'-style extent argument, a vector of xmin, xmax, ymin, ymax

##   name: a suffix for the file names
##         (image, UTM image, csv, excel files, PDF of map unit codes)
##         names start with "HWSD_", in subdirectory "window\" and area name

## the image `hwsd' and the SQLite database must
##   be already available in the environment
extract.one <- function(bbox, name="window")
  {
      print(paste("Area name: ", name, "; bounding box:
                  [",paste(bbox,collapse=", "),"]", sep=""))
                                        # extract the window
    dir.create(paste("./window/",name,sep=""), showWarnings = FALSE,
               recursive=TRUE)
    setwd(paste("./window/",name,sep=""))
    hwsd.win <- crop(hwsd, extent(bbox))
                                        # find the zone for the centre of the box
    print(paste("Central meridian:", centre <- (bbox[1]+bbox[2])/2))
    utm.zone <- long2UTM(centre)
    print(paste("UTM zone:", utm.zone))
                                        # make a UTM version of the window
    hwsd.win.utm <- projectRaster(hwsd.win,
                     crs=(paste("+proj=utm +zone=",utm.zone,
                     "+datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0",
                     sep="")), method="ngb")
    print(paste("Cell dimensions:",
                paste((cell.dim <- res(hwsd.win.utm)),
                      collapse=", ")))
                                        # write the raster images to disk
    eval(parse(text=paste("writeRaster(hwsd.win, file='./HWSD_", name,
                   "', format='EHdr', overwrite=TRUE)",sep="")))
    eval(parse(text=paste("writeRaster(hwsd.win.utm, file='./HWSD_", name,
                   "_utm', format='EHdr', overwrite=TRUE)",sep="")))
                                        # extract attributes for just this window
    dbWriteTable(con, name="WINDOW_TMP",
                 value=data.frame(smu_id=unique(hwsd.win)), overwrite=TRUE)
    records <- dbGetQuery(con, "select T.* from HWSD_DATA as T
                          join WINDOW_TMP as U on T.mu_global=u.smu_id
                          order by su_sym90")
    dbRemoveTable(con, "WINDOW_TMP")
                                        # convert to factors as appropriate
    for (i in names(records)[c(2:5,8:15,17:19,28,45)]) {
```

```r
        eval(parse(text=paste("records$",i," <- as.factor(records$",i,")",
                              sep="")))
        }
                                          # 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(paste("Dimensions of attribute table: ",
                paste(dim(records), collapse=", "),
                " (records, fields with data)", sep=""))
                                          # write attribute table in CSV formats
    eval(parse(text=paste("write.csv(records,
                           file='./HWSD_", name, ".csv')",sep="")))
                                          # make a spatial polygons dataframe,
                                          #  add attributes
    print(system.time(hwsd.win.poly <-
                      rasterToPolygons(hwsd.win, n=4, na.rm=TRUE, dissolve=TRUE)))
                                          # transform to UTM for correct geometry
    hwsd.win.poly.utm <- spTransform(hwsd.win.poly,
                                     CRS(proj4string(hwsd.win.utm)))
    m <- match(hwsd.win.poly.utm$value,
               records$MU_GLOBAL); hwsd.win.poly.utm@data <- records[m,]
                                          # plot the map unit ID
    print(paste("Number of legend categories in the map:",
                lvls <- length(levels(hwsd.win.poly.utm$MU_GLOBAL))))
    eval(parse(text=paste("pdf(file='./HWSD_", name, "_SMU_CODE.pdf')",sep="")))
    setwd("../..")
  } # end extract.one


##### main program ##########################################################

## read in HWSD raster database, assign CRS
require(sp)
require(raster)
hwsd <- raster("./HWSD_RASTER/hwsd.bil")
require(rgdal)
proj4string(hwsd) <-"+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

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

## other packages to be used in the function
require(rgeos)

## 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.one(c(-77, -75, 41, 43), "Twin_Tiers")

[1] "Area name: Twin_Tiers; bounding box:\n                  [-77, -75, 41, 43]"
[1] "Central meridian: -76"
[1] "UTM zone: 18"
[1] "Cell dimensions: 690, 925"
[1] "Dimensions of attribute table: 18, 48 (records, fields with data)"
   user  system elapsed
 26.615   0.212  27.056
[1] "Number of legend categories in the map: 9"

## clean up
dbDisconnect(con)
```

B Extracting a country

Here is a script that can be used to extract any rectangular window from the HWSD, using the techniques presented in this note. The R code extracted from this note includes this as a code chunk. The country name is as given in the CIA world database; this was explained in §2.2. Files are written into a subdirectory `./country/<country name>`; this is created if necessary.

```
## R script to extract a county from the Harmonized World Soil Database
## Author: D G Rossiter
## Version: 09-Aug-2017

##### initialize ##########################################################
rm(list=ls())

##### functions ###########################################################

## function to extract and format one country

## 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
##         will also be used a suffix for the file names
##           (image, csv attributes)
##         names start with "HWSD_Country_",
##           in subdirectory "country\" and area name

## the image `hwsd' and the SQLite database
##     must be already available in the environment
extract.one <- function(name="") {
    print(paste("Country:", name))
    dir.create(paste("./country/",name,sep=""), showWarnings = FALSE,
               recursive=TRUE)
    setwd(paste("./country/",name,sep=""))
    tmp <- map('worldHires',name, fill=TRUE, plot=FALSE)
    boundary <- map2SpatialPolygons(tmp, IDs=tmp$names,
                  proj4string=
                    CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"))
    bbox <- bbox(boundary)
    print(paste("Bounding box: [",paste(t(bbox),collapse=", "),"]", sep=""))
                                        # extract the window
    hwsd.win <- crop(hwsd, extent(bbox))
                                        # overlay only works for sp objects
    hwsd.win.sp <- as(hwsd.win, "SpatialGridDataFrame")
    ix <- over(hwsd.win.sp, boundary)
    hwsd.win.sp <- hwsd.win.sp[!is.na(ix),]
    hwsd.win <- as(hwsd.win.sp, "RasterLayer") # convert back to raster
                                        # find the zone for the centre of the box
    print(paste("Central meridian:", centre <- (bbox[1]+bbox[2])/2))
                                        # write unprojected raster window image
    eval(parse(text=paste("writeRaster(hwsd.win, file='./HWSD_raster_", name, "',
                           format='EHdr', overwrite=TRUE)",sep="")))
                                        # extract attributes for this window
    dbWriteTable(con, name="WINDOW_TMP", value=data.frame(smu_id=unique(hwsd.win)),
                  overwrite=TRUE)
    records <- dbGetQuery(con, "select T.* from HWSD_DATA as T
                          join WINDOW_TMP as U on T.mu_global=u.smu_id
                          order by su_sym90")
    dbRemoveTable(con, "WINDOW_TMP")
                                        # convert to factors as appropriate
    for (i in names(records)[c(2:5,8:15,17:19,28,45)]) {
        eval(parse(text=paste("records$",i," <- as.factor(records$",i,")",
                             sep="")))
        }
```

```r
                                        # include all fields
    print(paste("Dimensions of attribute table: ",
                paste(dim(records), collapse=", "), " (records, fields)",
                sep=""))
                                        # write attribute table in CSV format
    eval(parse(text=paste("write.csv(records, file='./HWSD_attributes_",
                   name, ".csv')",sep="")))
    setwd("../..")
  } # end extract.one


##### main program ########################################################

## read in HWSD raster database, assign CRS
require(sp)
require(raster)
hwsd <- raster("./HWSD_RASTER/hwsd.bil")
require(rgdal)
proj4string(hwsd) <-"+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

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

## packages for country boundaries
require(maps)
require(mapdata)
require(maptools)


## call the function for each window 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.one('Sri Lanka')

[1] "Country: Sri Lanka"
[1] "Bounding box: [79.6519622802734, 81.8916549682617, 5.91779470443726, 9.82834625244141]"
[1] "Central meridian: 42.7848784923553"
[1] "Dimensions of attribute table: 107, 57 (records, fields)"

## clean up
dbDisconnect(con)
```

References

[1] Roger S. Bivand, Edzer J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. Springer, 2008. ISBN 978-0-387-78170-9. URL http://www.asdar-book.org/. 1

[2] Robert J. Hijmans and Jacob van Etten. *raster: Geographic analysis and modeling with raster data*, 2012. URL http://CRAN.R-project.org/package=raster. R package version 2.0-12. 1

[3] IIASA; FAO; ISRIC; ISS-CAS; JRC. *Harmonized World Soil Database (version 1.2)*. FAO and IIASA, Feb 2012. URL http://webarchive.iiasa.ac.at/Research/LUC/External-World-soil-database/HWSD_Documentation.pdf. 1, 3, 11

[4] Timothy H. Keitt, Roger Bivand, Edzer Pebesma, and Barry Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2012. URL http://CRAN.R-project.org/package=rgdal. R package version 0.7-20. 1

[5] osgeo.org. PROJ.4. URL https://trac.osgeo.org/proj/. 3

[6] Edzer J. Pebesma and Roger S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, 2005. 1

[7] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0. 1

[8] R Development Core Team. *R Data Import/Export*. The R Foundation for Statistical Computing, version 3.4.1 (2017-07-30) edition, 2017. URL http://cran.r-project.org/doc/manuals/R-data.pdf. 10

[9] Hadley Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 2011. ISSN 1548-7660. doi: 10.18637/jss.v040.i01. 24

[10] Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10): 1–23, 2014. ISSN 1548-7660. doi: 10.18637/jss.v059.i10. 24

[11] Yihui Xie. knitr: Elegant, flexible and fast dynamic report generation with R, 2017. URL http://yihui.name/knitr/. 2

Index of R Concepts

:: operator, 24
%/% operator, 5
%>% operator, 24

all, 16

brewer.pal (RColorBrewer package), 5

click (raster package), 7
click argument (id function), 7
collapse argument (paste function), 12
crop (raster package), 4, 8

dataframes2xls package, 16
dbConnect (SQLite package), 10
dbDriver (SQLite package), 10
dbExecute (RSQLite package), 16
dbGetQuery (RSQLite package), 16
dbGetQuery (SQLite package), 10, 13, 21
DBI package, 10
dbListTables (SQLite package), 10
dbWriteTable (SQLite package), 13, 21
dplyr package, 24

eval, 15
extent (raster package), 3

fill argument (map function), 8

gather (tidyr package), 25
group_by (dplyr package), 25

is.na, 16

knitr package, 2

map (maps package), 8
map2SpatialPolygons (maptools package), 8
mapdata package, 8
maptools package, 8
match, 17, 20, 22, 23
mutate (dplyr package), 25

ncol (raster package), 3
nrow (raster package), 3

over (sp package), 8

parse, 15
paste, 12, 13, 15
paste argument (collapse function), 12
proj4string (rgdal package), 3
projection (raster package), 3
projectRaster (raster package), 5

raster (raster package), 2
raster package, 1–3, 17, 18
RasterLayer class, 8, 9
rasterToPolygons (raster package), 18
RColorBrewer package, 5
require, 3, 10
res (raster package), 3, 6
rgdal package, 1, 19
rgeos package, 18
RODBC package, 1
RSQLite package, 1, 10

scales argument (spplot function), 9
select (dplyr package), 24
sp package, 1–3, 8, 17, 19
SpatialGridDataFrame (sp class), 17
SpatialPixelsDataFrame (sp class), 8
SpatialPolygons (sp class), 7, 8
spplot (sp package), 9, 17
spread (tidyr package), 25
spTransform (rgdal package), 19
sum, 25
summarise (dplyr package), 25
system.time, 18

tidyr package, 24, 25

ungroup (dplyr package), 25
unique (raster package), 4

worldHires dataset, 8
write.csv, 16
write.xls (dataframes2xls package), 16

zcol argument (spplot function), 17