Using R tools for analysis of primary biodiversity data provided by SBDI

Debora Arlt and Alejandro Ruete for the Swedish Biodiversity Data Infrastructure

2021-05-27

Contents

Introduction

Biodiversity resources are increasingly international. The SBDI has made an effort to canalize biodiversity data and resources to help the research community access and analyze Swedish primary biodiversity data. Each research question draws its own challenges which are unique in themselves. Our aim here is to provide a few examples that prompt questions that may be asked at different stages of the process. The validity and appropriateness of a particular method depends on the individual researcher(s). For a comprehensive workflow on how to treat and analyze PBD please refer to our tutorial on biodiversity analysis tool where we go through the complete workflow Data –> Cleaning –> Fitness evaluation –> Analysis

R and Mirroreum

The present tutorial is focused on the statistical programming language R. R is a free software environment for statistical computing and graphics that is widely used within the scientific community and where the complete analysis workflow can be documented in a fully reproducible way.

At SBDI we provide access for researchers and students to Mirroreum – an online web-based environment for Reproducible Open Research in the area of biodiversity analysis. Mirroreum is based on a Free and Open Source stack of software. Logging in, you immediately get access to a web-based version of R Studio with a large number of pre-installed packages such as all the packages offered from ROpenSci and more.

Compared to running R Studio on your own machine, Mirroreum offers more computational resources and a standardized environment where you can rely on all the relevant packages being installed and the configuration parameters being set appropriately. To know more about Mirroreum or to request an account please visit the SBDI documentation site

SBDI4R - R package to search an access data

The SBDI4R package enables the R community to directly access data and resources hosted by SBDI. The goal is to enable observations of species to be queried and output in a range of standard formats. It includes some filter functions that allow you to filter prior to download. It also includes some simple summary functions, and some function for some simple data exploration. The examples included in this tutorial also show you how you can continue exploring and analyzing using other R package.

Please refer to the package documentation for details on how to install it. Once installed the SBDI4R package must be loaded for each new R session:

```
library(SBDI4R)
```

Customizing SBDI4R

Various aspects of the SBDI4R package can be customized.

Caching

SBDI4R can cache most results to local files. This means that if the same code is run multiple times, the second and subsequent iterations will be faster. This will also reduce load on the web servers. By default, this caching is session-based, meaning that the local files are stored in a temporary directory that is automatically deleted when the R session is ended. This behaviour can be altered so that caching is permanent, by setting the caching directory to a non-temporary location. For example, under Windows, use something like:

```
sbdi_config(cache_directory = file.path("c:","mydata","sbdi_cache")) ## Windows
```

or for Linux:

```
sbdi_config(cache_directory = "~/mydata/sbdi_cache") ## Linux
```

Note that this directory must exist (you need to create it yourself).

All results will be stored in that cache directory and will be used from one session to the next. They won't be re-downloaded from the server unless the user specifically deletes those files or changes the caching setting to "refresh".

If you change the cache_directory to a permanent location, you may wish to add something like this to your .Rprofile file, so that it happens automatically each time the SBDI4R package is loaded:

```
setHook(packageEvent("SBDI4R", "onLoad"),
        function(...) sbdi_config(cache_directory=file.path("~","mydata","sbdi_cache")))
```

Caching can also be turned off entirely by:

```
sbdi_config(caching="off")
```

or set to "refresh", meaning that the cached results will re-downloaded from the SBDI servers and the cache updated. (This will happen for as long as caching is set to "refresh" — so you may wish to switch back to normal "on" caching behavior once you have updated your cache with the data you are working on).

E-mail address

Each download request to SBDI servers is also accompanied by an "e-mail address" string that identifies the user making the request. You will need to provide an email address registered with the SBDI. You can create an account here. Once an email is registered with the SBDI, it should be stored in the config:

```
sbdi_config(email="your.valid@emailaddress.com")
```

Else you can provide this e-mail address as a parameter directly to each call of the function occurrences().

Setting the download reason

SBDI requires that you provide a reason when downloading occurrence data (via the SBDI4R `occurrences()` function). You can provide this as a parameter directly to each call of `occurrences()`, or you can set it once per session using:

```
sbdi_config(download_reason_id = "your_reason_id")
```

(See `sbdi_reasons()` for valid download reasons, e.g. * 3 for "education", * 7 for "ecological research", * 8 for "systematic research/taxonomy", * 10 for "testing")

NO other personal identification information is sent. You can see all configuration settings, including the the user-agent string that is being used, with the command:

```
sbdi_config()
```

Other options

If you make a request that returns an empty result set (e.g. an un-matched name), by default you will simply get an empty data structure returned to you without any special notification. If you would like to be warned about empty result sets, you can use:

```
sbdi_config(warn_on_empty=TRUE)
```

Other packages needed

Some additional packages are needed for these examples. Install them if necessary with the following script.

```
to_install <- c("dplyr", "BIRDS","ggplot2", "jpeg", "leaflet","maps", "mapdata",
                "maptools", "sp", "rgeos", "tidyr", "vegan")
to_install <- to_install[!sapply(to_install, requireNamespace, quietly=TRUE)]
if(length(to_install)>0)
    install.packages(to_install, repos="http://cran.us.r-project.org")
```

Your collaboration is appreciated

Open Source also means that you can contribute. You don't need to know how to program but every input is appreciated. Did you find something that is not working? Have suggestions for examples or text? you can always 1. Reach to us via the support center 2. Submit and issue to the GitHub code repository see how 3. Or contribute with your code or documents modifications by "forking" the code and submitting a "pull request"

The repositories you can contribute to are: * Mirroreum https://github.com/mskyttner/mirroreum * SBDI4R https://github.com/biodiversitydata-se/SBDI4R (NOTE: we may not develop this package but instead move to a new one) * the general analysis workflows [https://github.com/biodiversitydata-se/biodiversity-analysis-tools] https://github.com/biodiversitydata-se/biodiversity-analysis-tools * these tutorial https://github.com/biodiversitydata-se/r-tools-tutorial

1 Example with fish data from SERS

In this example we are interested in exploring data from a specific data resource – Swedish Electrofishing Registry - SERS (Institutionen för akvatiska resurser, SLU). This data base has 2.8 M observations starting in the 1950's.

As you may already know, SBDI is a collection of many biodiversity databases. We start by searching for the data resource we are interested in using the function `pick_filter()`. This is an interactive query guiding you through the many resources available to filtering your query (data resources, spatial layers, and curated species lists).

```
library(SBDI4R)
fq_str <- pick_filter("resource")
# follow the instructions
```

Follow the instruction. Your choices here would have been "in3" –> "dr10". Your variable `fq_str` will now contain a string "data_resource_uid:dr10".

But we are not interested in the complete database, but on the last 10 years of data. for this we concatenate (add to a vector) another filter string. These will be treated as AND factors.

```
y1 <- 2008
y2 <- 2012
fq_str <- c(fq_str, paste0("year:[", y1, " TO ", y2,"]"))
# Note the square brackets are hard limits
```

For references on how to use the filters see SBDI APIS documentation.

Using the function `occurrences()` we can the query for the observations fulfilling our filter. If you haven't specified that in the `sbdi_config()` before, you need to pass your email and the download reason.

```
library(SBDI4R)
xf <- occurrences(fq = fq_str,
                  email = "sbdi4r-test@biodiversitydata.se",
                  download_reason_id = 10)
```

```
## Registered S3 methods overwritten by 'ALA4R':
##   method              from
##   subset.occurrences  SBDI4R
##   summary.occurrences SBDI4R
##   unique.occurrences  SBDI4R
```

```
# Simply summarise all records by data source
table(xf$data$dataResourceName)
```

```
##
## SLU Aqua  Institute of Freshwater Research Swedish Electrofishing Registry - SERS
##                                                                             95082
```

```
table(xf$data$dataResourceID)
```

```
##
##  dr10
## 95082
```

1.1 Plotting data on a map

You can quickly plot all the observations as a PDF file with the function `ocurrence_plot()`, one page per species:

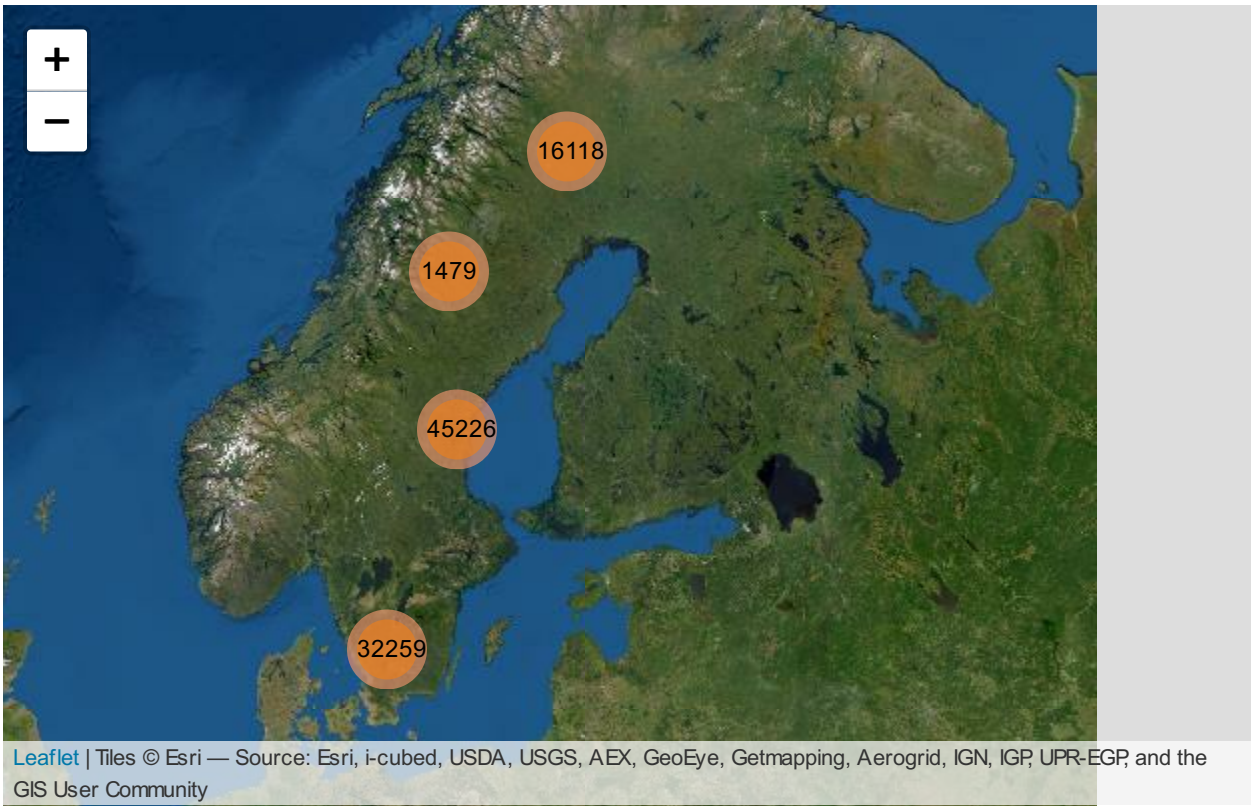
```
occurrences_plot(xf, "obsPlot.pdf",
                  grouped=FALSE,
                  taxon_level="species",
                  pch='+')
```

Note that the plot is saved to a pdf file in the current working directory. You can find that with `getwd()`.

1.1.0.1 Leaflet There are many other ways of producing spatial plots in R. The leaflet package provides a simple method of producing browser-based maps with panning, zooming, and background layers:

```
library(leaflet)
# drop any records with missing lat/lon values
xfl <- xf$data[!is.na(xf$data$longitude) | !is.na(xf$data$latitude),]
marker_colour <- rep("#d95f02", nrow(xfl))
# blank map, with imagery background
leaflet(width = "100%") %>%
  addProviderTiles("Esri.WorldImagery") %>%
  # add markers
  addCircleMarkers(xfl$longitude, xfl$latitude,
                   radius = 1,
```

```
                    fillOpacity =.5,
                    opacity = 1,
                    col=marker_colour,
                   clusterOptions = markerClusterOptions())
```



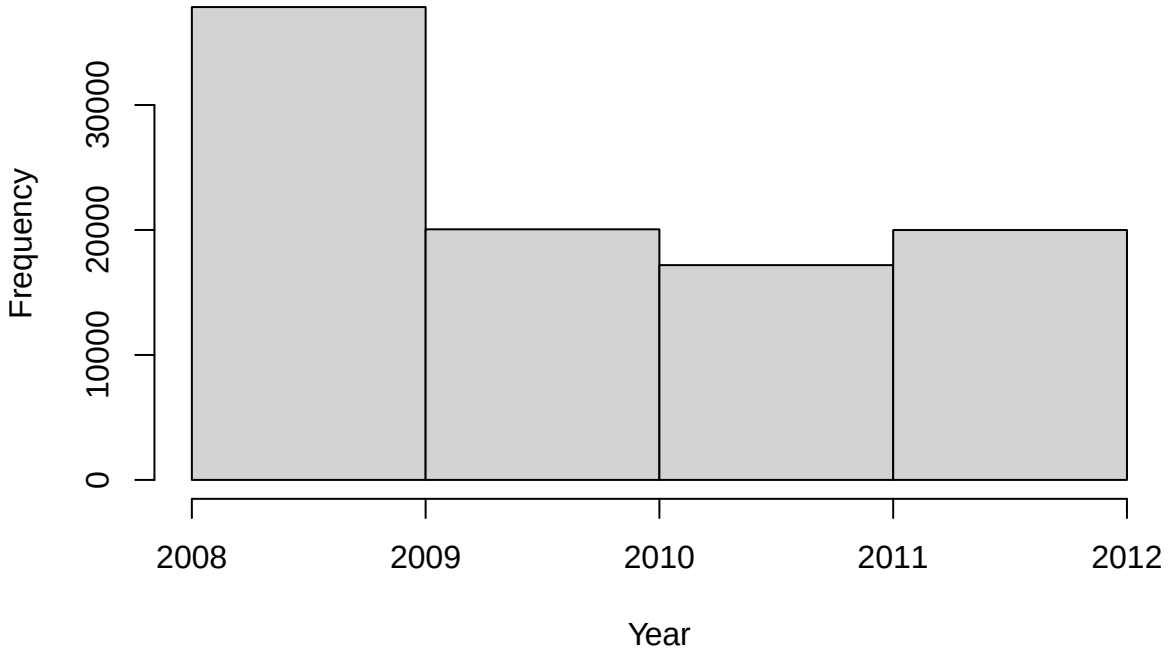
1.2 Temporal summary

A quick summary over the years reveal a drop in number of records over time.

```
table(xf$data$year)
```

```
##
##  2008  2009  2010  2011  2012
## 18168 19674 20055 17188 19997
```

```
hist(xf$data$year,
     breaks = seq(y1, y2),
     xlab = "Year",
     main = "")
```



1.3 Species summary

In the same way we can summaries the number of observations for each species, by common or scientific name.

```
sppTab <- table(xf$data$commonName)
sppDF <- as.data.frame(sppTab)
colnames(sppDF)[1] <- "species"
head(sppDF)
```

```
##           species Freq
## 1                   66
## 2 Alpine bullhead 4615
## 3 American burbot 7081
## 4        Aral asp    6
## 5     Arctic char   46
## 6    aurora trout  856
```

```
sppTab <- table(xf$data$scientificName)
sppDF <- as.data.frame(sppTab)
colnames(sppDF)[1] <- "species"
head(sppDF)
```

```
##                                species Freq
## 1       Abramis brama (Linnaeus, 1758)   61
## 2   Alburnus alburnus (Linnaeus, 1758)  660
## 3   Anguilla anguilla (Linnaeus, 1758) 2140
## 4                            Astacidae  100
## 5     Astacus astacus (Linnaeus, 1758)  618
## 6 Barbatula barbatula (Linnaeus, 1758)  620
```

Perhaps, you need to send this table as a .CSV file to a colleague.

```
write.csv(sppDF, "SERS_species_summary.csv")
# NOTE: again this will be saved on your working directory
```

1.4 Spatial biodiversity analysis

Let's now ask: how does the species richness vary across Sweden? In this case we want to summarise occurrences species-wise over a defined grid instead of plotting every observation point. First we need to overlay the observations with a grid. In this case, the standard Swedish grids at 50, 25, 10 and 5 km are provided as data in the SBDI4R package (with Coordinate Reference System = WGS84, EPSG:4326).

```
library(sp) # the function coordinates() and proj4string() are in sp
```

```
## Warning: package 'sp' was built under R version 4.0.3
```

```
library(rgeos) # the function over() is in package rgeos
```

```
## rgeos version: 0.5-5, (SVN revision 640)
##  GEOS runtime version: 3.8.0-CAPI-1.13.1
##  Linking to sp version: 1.4-2
##  Polygon checking: TRUE
```

```
# load some shapes over Sweden's political borders
data("swe_wgs84", package="SBDI4R", envir=environment())
# A standard 50km grid
data("Sweden_Grid_50km_Wgs84", package="SBDI4R", envir=environment())

grid <- Sweden_Grid_50km_Wgs84

# make the observations spatial
# NOTE: make sure there are no NAs on either column defining the coordinates
# xf$data[!is.na(xf$data$longitude) | !is.na(xf$data$latitude),]

obs <- as.data.frame(xf$data)
coordinates(obs) <- obs[,c("longitude","latitude")]
wkt <- sf::st_crs(4326)[[2]]
```

```
proj4string(obs) <- sp::CRS(wkt) #CRS("+init=epsg:4326")

nObs <- nrow(obs)

# overlay the data with the grid
ObsInGridList <- over(grid, obs, returnList=TRUE)
wNonEmpty <- unname( which( unlist(lapply(ObsInGridList, nrow)) != 0) )
if(length(wNonEmpty)==0) message("Observations don't overlap any grid cell.")
```

The result `ObsInGridList` is a `list` object with a subset of the data on each grid. Now summarise occurrences within grid cells:

```
# check n the total number of observations
sum(unlist(lapply(ObsInGridList, nrow)))
```

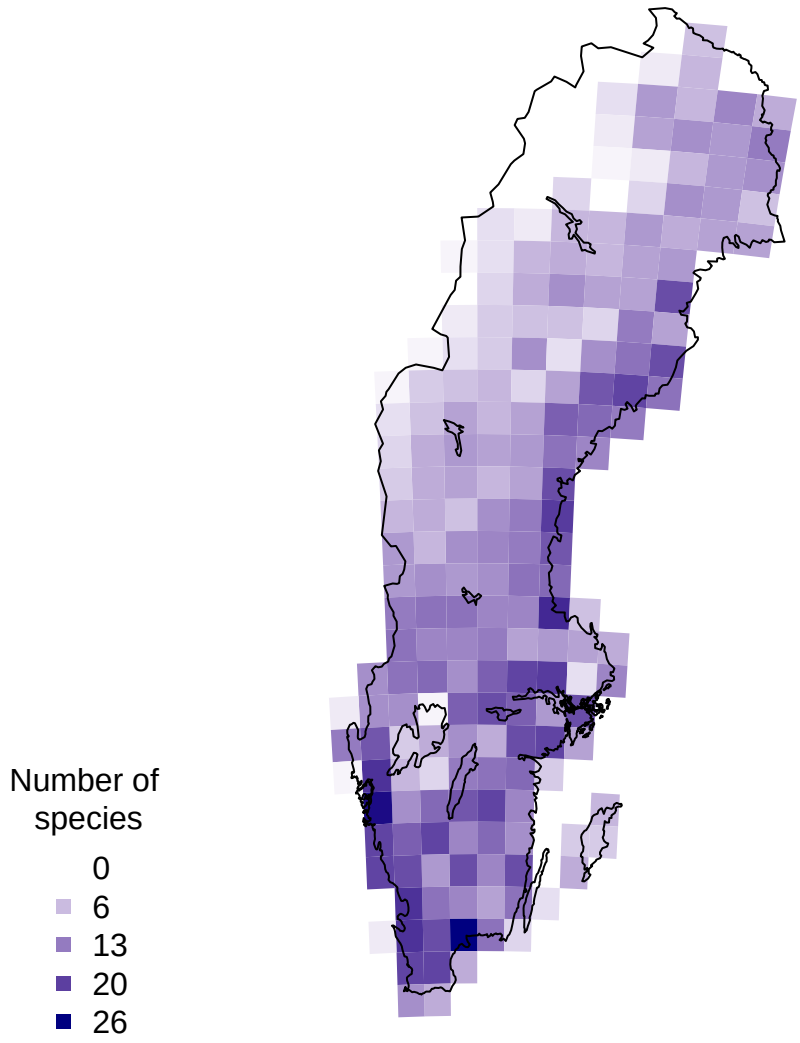
```
## [1] 95082
```

```
# apply a summary over the grid cells
nCells <- length(ObsInGridList)

res <- data.frame("nObs"=as.numeric(rep(NA,nCells)),
                  "nYears"=as.numeric(rep(NA,nCells)),
                  "nSpp"=as.numeric(rep(NA,nCells)),
                  row.names = row.names(grid),
                  stringsAsFactors = FALSE)

cols2use <- c("scientificName", "year")

dataRes <- lapply(ObsInGridList[wNonEmpty],
                  function(x){
                    x <- x[,cols2use]
                    colnames(x) <- c("scientificName", "year")
                    return(c("nObs" = length(x[,"scientificName"]),
                             "nYears" = length(unique(x[,"year"])),
                             "nSpp" = length(unique(x[,"scientificName"]))
                             )
                           )
                    }
                  )

dataRes <- as.data.frame(dplyr::bind_rows(dataRes, .id = "gridID"))

res[wNonEmpty,] <- dataRes[,-1]

resSp <- sp::SpatialPolygonsDataFrame(grid, res)
```

And finally plot the grid summary as a map:

```
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
                               c(0, max(resSp@data$nSpp, na.rm = TRUE)),
                               na.color = "transparent")
oldpar <- par()
```

```
par(mar = c(1,1,0,0))
plot(resSp, col=palBW(resSp@data$nSpp), border = NA)
plot(swe_wgs84$Border, border=1, lwd=1, add=T)
legend("bottomleft",
       legend = round(seq(0, max(resSp@data$nSpp, na.rm = TRUE), length.out = 5)),
       col = palBW(seq(0, max(resSp@data$nSpp, na.rm = TRUE), length.out = 5)),
       title = "Number of \nspecies", pch = 15, bty="n")
```




```
suppressWarnings(par(oldpar))
```

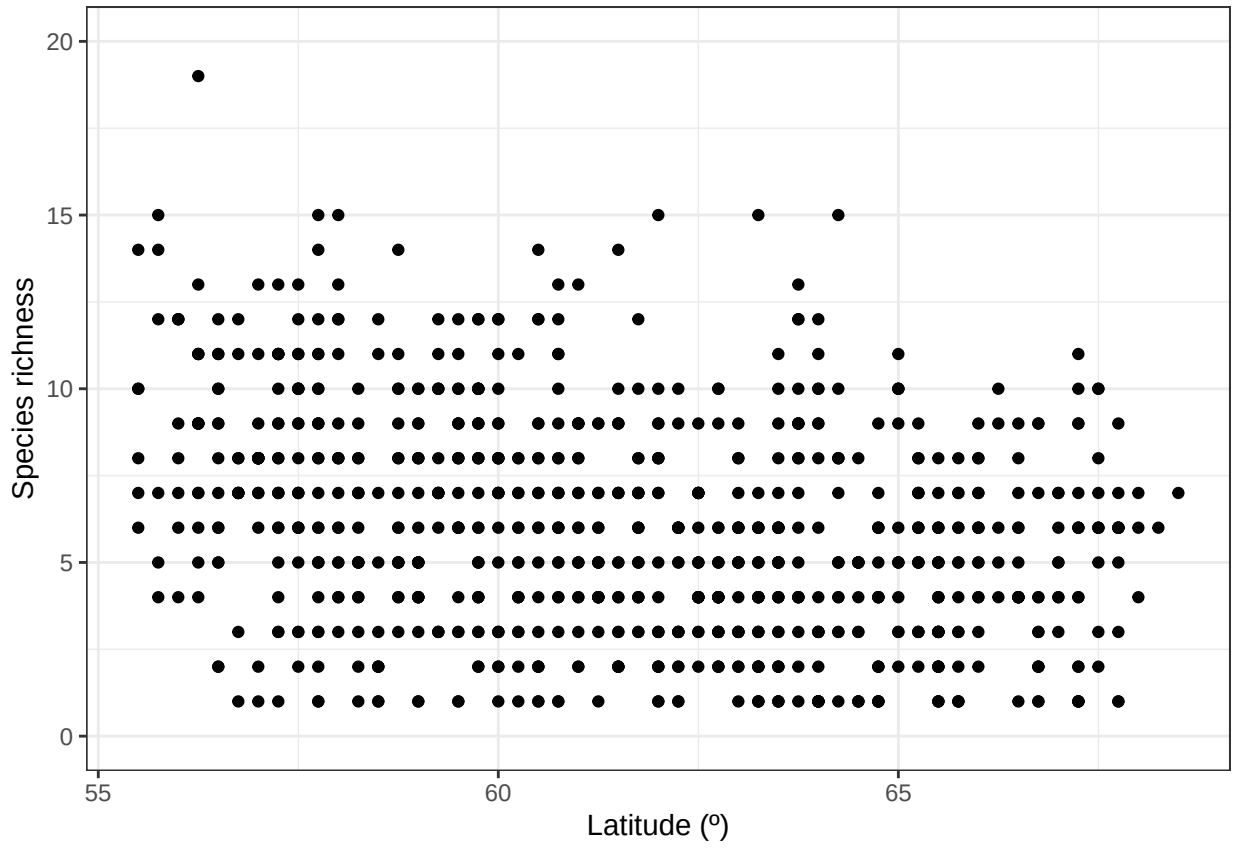
We can go further by gathering the observations by latitude.

```
library(dplyr)
library(tidyr)
xgridded <- xf$data %>%
```

```
    ## discard genus- and higher-level records
    filter(rank %in% c("species", "subspecies", "variety", "form", "cultivar")) %>%
    mutate(longitude = round(longitude * 4)/4,
           latitude = round(latitude * 4)/4) %>%
    group_by(longitude,latitude) %>%
    ## subset to vars of interest
    select(longitude, latitude, species) %>%
    ## take one row per cell per species (presence)
    distinct() %>%
    ## calculate species richness
    mutate(richness=n()) %>%
    ## convert to wide format (sites by species)
    mutate(present=1) %>%
    do(tidyr::pivot_wider(data=.,  names_from=species, values_from=present, values_fill=0)) %>%
    ungroup()
## where a species was not present, it will have NA: convert these to 0
sppcols <- setdiff(names(xgridded),
                   c("longitude", "latitude", "richness"))
xgridded <- xgridded %>%
  mutate_at(sppcols, function(z) ifelse(is.na(z), 0, z))
```

And plot it accordingly

```
library(ggplot2)

ggplot(xgridded, aes(latitude, richness)) +
  labs(x = "Latitude (º)",
       y = "Species richness") +
  lims(y = c(0,20)) +
  geom_point() +
  theme_bw()
```



1.5 Temporal biodiversity analysis

Despite the drop seen in sampling effort, we can still analyse weather there is any trends for species over the sampled years. To do this, we first need to check whether the spatial sampling effort was comparable across those years

```
xf$data %>%
    ## discard genus- and higher-level records
    filter(rank %in% c("species", "subspecies", "variety", "form", "cultivar")) %>%
    group_by(year) %>%
    ## subset to vars of interest
    select(year, locality) %>%
    ## calculate species richness
    mutate(nObs = n())
```

```
## # A tibble: 93,205 x 3
## # Groups:   year [5]
##     year locality                                nObs
##    <int> <chr>                                <int>
##  1  2011 "6846840-1568800 Sjötrösk. Lillfj.utl" 16853
##  2  2009 "6340000-1286200 Göingegården"         19300
##  3  2009 ""                                     19300
##  4  2011 "6846840-1568800 Sjötrösk. Lillfj.utl" 16853
##  5  2009 "6339030-1288080 Holmagärde B st.v. 2" 19300
##  6  2009 "6339030-1288080 Holmagärde B stv. 2"  19300
```

```
##  7  2009 ""                                       19300
##  8  2010 "6321540-1301320 Ned övre kulv. Krono" 19648
##  9  2010 ""                                       19648
## 10  2009 "6466090-1409270 Nedstr Herrekvarn"    19300
## # ... with 93,195 more rows
```

And we ask: can we see a trend (change = frequency of occurrence = number caught) for species X, or species Y?

1.6 Links to further tutorials that could be of interest

Link to analyses of interest – e.g. species distribution models, or trend analyses, or species divesity – link to biodiversity workflow

2 Case 2:

Look at opportunistically collected citizen science data – using the big Swedish cit science species observation data portal Artportalen.

We want to look at dragonflies – but before we can use the observation records we need to know how the observation effort has varied over time and in space. For this we define field visits i.e. occasions at which an observer has sampled observations – if we have information on observer id, location id and date we can aggregate data into "field visits". We do this using BIRDS, and 25km grid: Select data – get records for Southern Sweden (Götaland) and years 2000-2010. How has observation effort (frequency of visits) varied over time and space? – 1) show maps as in Example 7 (all years, year 2000, 2002, 2004, 2006, 2008, 2010), 2 make also a time line plot with no. visits against years, no. of gridcells with visits against years.

We can now look at some particular species and ask whether this has changed in occurrence over time: Plot no. records of species x and no. visits all species over years (we simply explore by comparing records for a species with no visits, can assume that species has increased of stronger positive trend than for no. visits) Plot no. gridcells with visits for species x and no. gridcells with visits for all species over years (we simply explore by comparing records for a species with no visits, can assume that species has increased of stronger positive trend than for no. visits) (species x: Tvåfläckad trollslända Epitheca bimaculata)