

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

Debora Arlt and Alejandro Ruete for the Swedish Biodiversity Data Infrastructure

2021-10-05

Contents

Introduction	1
R and Mirroreum	2
SBDI4R - an R to search an access data	2
Other packages needed	4
Your collaboration is appreciated	4
1 Example with fish data from SERS	5
1.1 Plotting data on a map	5
1.2 Temporal summary	7
1.3 Species summary	7
1.4 Spatial biodiversity analysis	8
2 Example with opportunistic data on Dragonflies	12
2.1 Name searching	12
2.2 Filter the search to get the observations	13
2.3 Quality and fit-for-use check	14
2.4 Species trends	19

Introduction

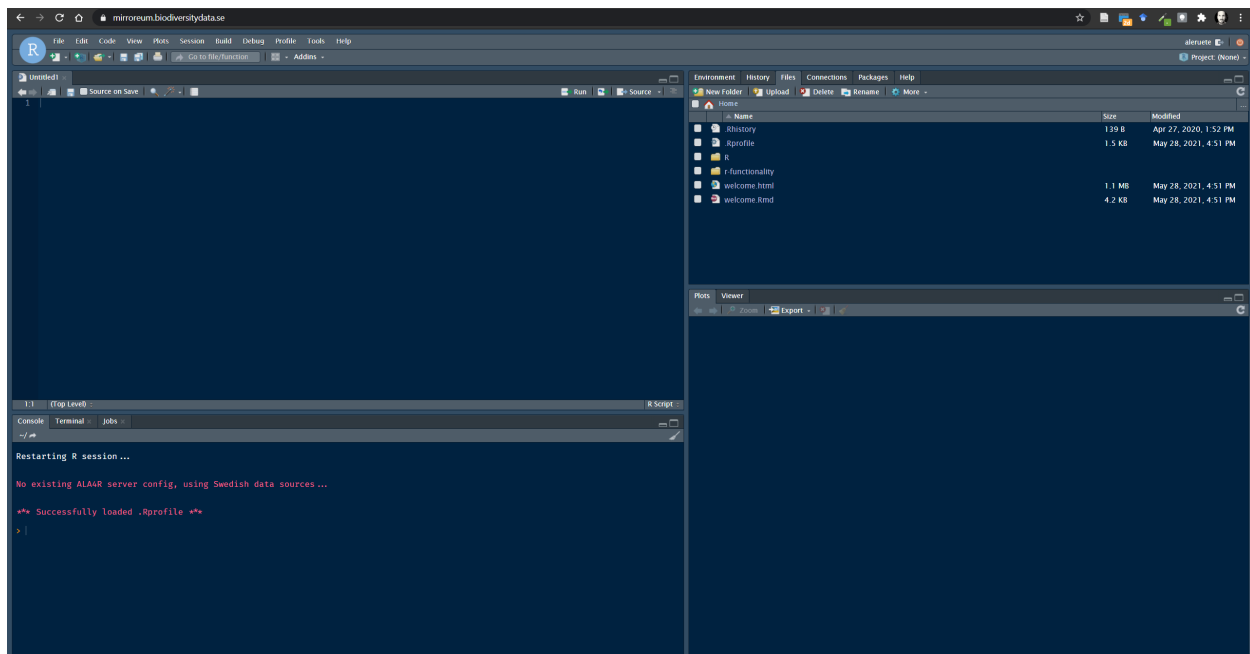
Biodiversity resources are increasingly international. The SBDI has made an effort to canalise biodiversity data and resources to help the research community access and analyse Swedish primary biodiversity data. Each research question draws its own challenges which are unique in themselves. Our aim here 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 analyse primary biodiversity data please refer to our tutorial on [biodiversity analysis tools](#) 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 - an R 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)
```

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 behavior 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”)

Privacy *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("colorRamps", "cowplot", "dplyr", "ggplot2",  
               "leaflet", "maps", "mapdata", "maptools", "sf",  
               "remotes", "rgeos", "tidyr", "xts")  
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")  
  
remotes::install_github("AtlasOfLivingAustralia/ALA4R")  
remotes::install_github("Greensway/BIRDS")
```

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>
- this R-tools 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 – the Swedish Electrofishing Registry - SERS (Department of Aquatic Resources, SLU Aqua). This database has 2.8 M observations starting in the 1950's.

SBDI is a collection of many biodiversity databases. We start by searching for the data resource we are interested in by 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 instructions. Your choices here would have been “in3” :arrow_right: “dr10” (data resource 10 = SERS). Your variable `fq_str` will now contain a string “data_resource_uid:dr10”.

But we are not interested in the complete database, we only want to look at the data from the last 10 years. For this we concatenate (add to a vector) another filter string. Both filter strings (for data resource and for time period) 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 the SBDI APIs [documentation](#).

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

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

# Remove what is not a species
xf$data <- xf$data[xf$data$rank == "species",]

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

```
##
## SLU Aqua  Institute of Freshwater Research Swedish Electrofishing Registry - SERS
##                                                    93200
```

1.1 Plotting data on a map

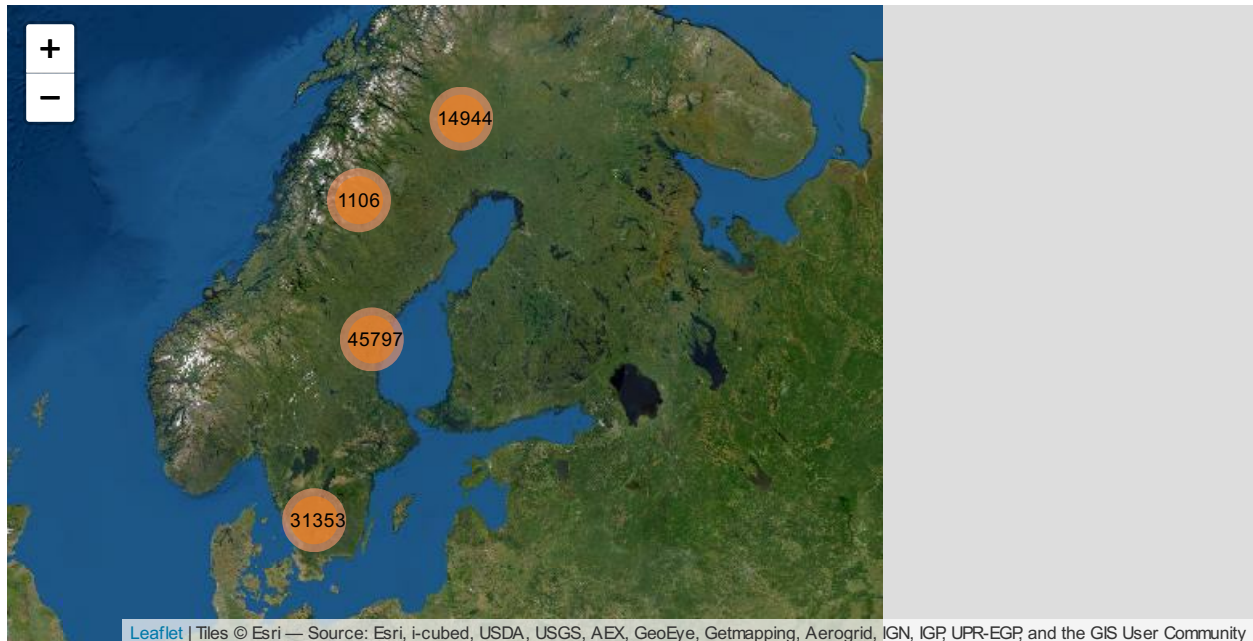
You can quickly plot all the observations as a PDF file with the function `occurrence_plot()`, one page per species:

```
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()`.

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())
```



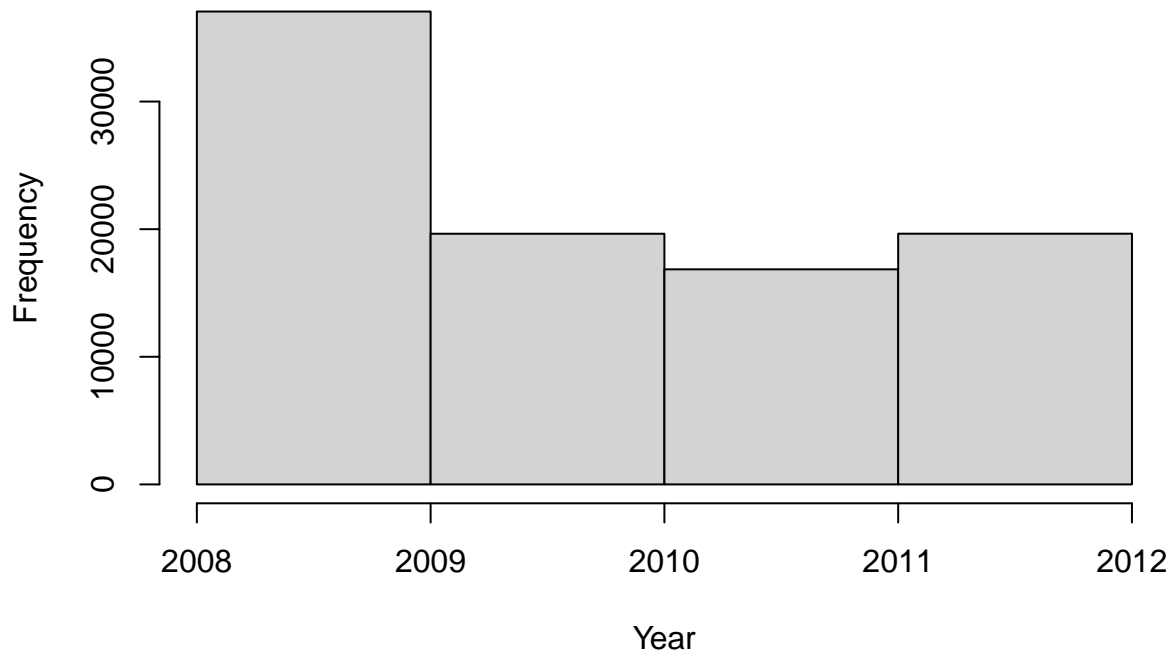
1.2 Temporal summary

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

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

```
##  
## 2008 2009 2010 2011 2012  
## 17757 19300 19643 16853 19647
```

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



1.3 Species summary

In the same way we can summarise 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              61
## 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    Astacus astacus (Linnaeus, 1758)  618
## 5 Barbatula barbatula (Linnaeus, 1758)  620
## 6    Blicca bjoerkna (Linnaeus, 1758)   74
```

Perhaps, you want to send this table as a .CSV file to a colleague. Save the table:

```
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?

For this 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. Here we are using the standard Swedish grids with grid square size of 50, 25, 10 or 5 km provided as data in the SBDI4R package (with Coordinate Reference System = WGS84, EPSG:4326).

```
library(sf) # the function coordinates() and proj4string() are in sp
library(rgeos) # the function over() is in package rgeos
# load some shapes over Sweden's political borders
data("swe_wgs84", package = "SBDI4R", envir = environment())
# a standard 50 km 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 in the columns defining the coordinates
# xf$data[!is.na(xf$data$longitude) | !is.na(xf$data$latitude),]

obs <- st_as_sf(as.data.frame(xf$data),
               coords = c("longitude", "latitude"),
               crs = st_crs(4326))
```



```
# overlay the occurrence data with the grid
ObsInGridListID <- st_intersects(grid, obs)
ObsInGridList <- lapply(ObsInGridListID, function(x) st_drop_geometry(obs[x,]))
wNonEmpty <- unname( which( unlist(lapply(ObsInGridList, nrow)) != 0) )
```

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

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

```
## [1] 93200
```

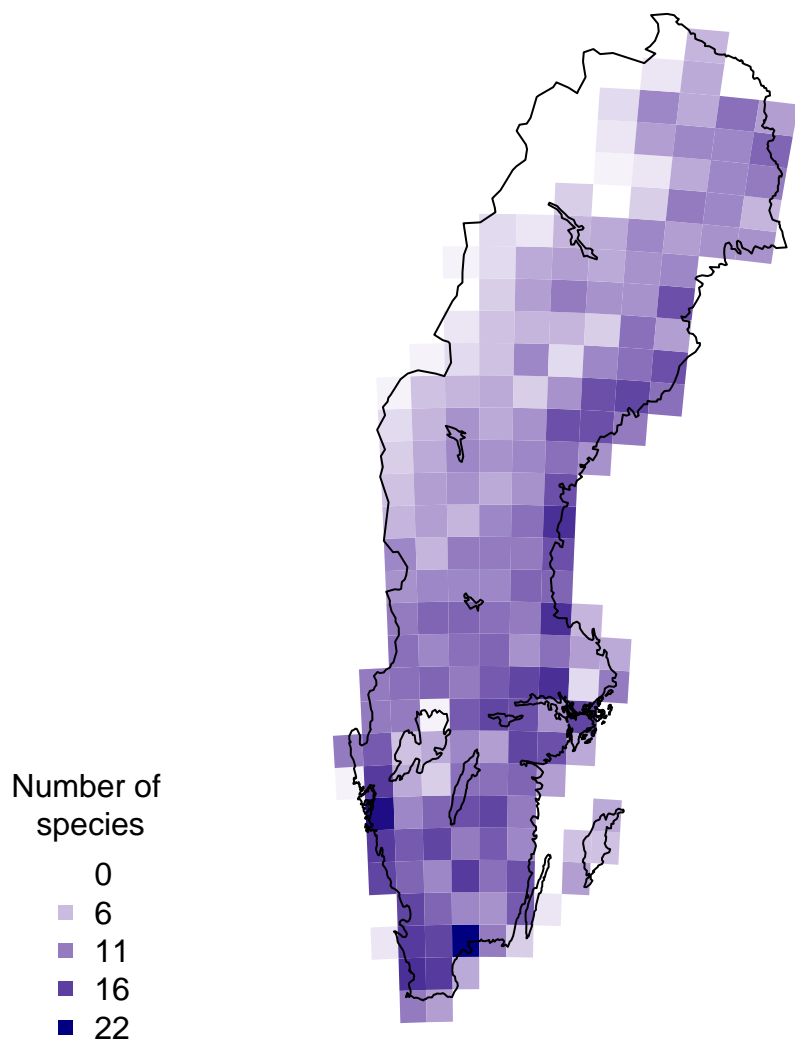
```
# apply a summary over the grid cells
nCells <- length(ObsInGridList)
res <- data.frame("nObs" = as.numeric(rep(NA,nCells)),
                 "nYears" = as.numeric(rep(NA,nCells)),
                 "nSpp" = as.numeric(rep(NA,nCells)),
                 row.names = row.names(grid),
                 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]
resSf <- st_as_sf(data.frame(res, st_geometry(grid)))
```

And finally plot the grid summary as a map:

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

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



We may now ask whether species richness varies across latitude. So we go further by arranging the observations by latitude:

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

```

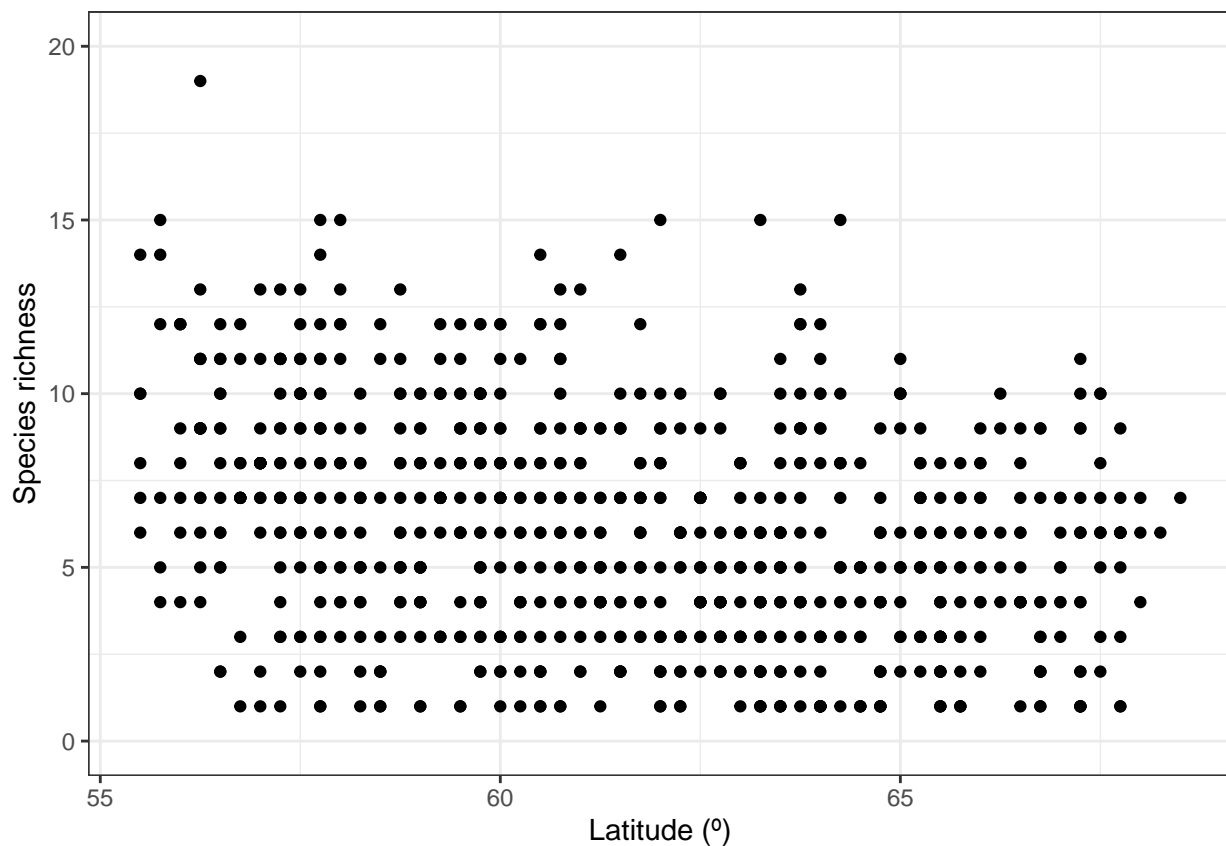
And plot it accordingly:

```

library(ggplot2)

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

```



2 Example with opportunistic data on Dragonflies

In this example we are interested in exploring opportunistically collected data from the Swedish citizen science species observation portal - Artportalen.

2.1 Name searching

To begin with, we want be sure there is an unequivocal way to find the species within the order Odonata (dragonflies) and nothing else, so let's search for "odonata":

```
sx <- search_fulltext("odonata")
sx$data[, c("guid", "scientificName", "rank", "occurrenceCount")]

## [1] "https://species.biodiversitydata.se/ws/search.json?q=odonata&fq=idxtype%3ATAXON"

##      guid      scientificName  rank occurrenceCount
## 1 10072832 Odonata associated gemycircularvirus 2 species          0
## 2  7367071  Ramalina fastigiata var. odonata Hue variety          0
## 3      789          Odonata    order         14121
## 4  8062407 Bdellodes odonata Wallace & Mahon, 1976 species          0
## 5  9829523 Odonata associated gemycircularvirus 1 species          0
```

We quickly see there that other taxonomic levels appear too, and also species that look suspiciously as not belonging to dragonflies. But there is only one order. Let's refine the search. To know which search fields we can use to filter the search we use the function `sbdi_fields(fields_type = "general")`. The search field we are looking for is "order_s".

```
sx <- search_fulltext(fq = "order_s:Odonata", page_size = 10)
sx$data[, c("scientificName", "rank", "occurrenceCount")]

## [1] "https://species.biodiversitydata.se/ws/search.json?fq=order_s%3AOdonata&fq=idxtype%3ATAXON&page=1"

##      guid      scientificName  rank occurrenceCount
## 1 11034676  Notoneura xanthe Lieftinck, 1938 species          0
## 2 11034731  Protoneura bifurcata Sjöstedt, 1918 species          0
## 3 11034937  Oxygomphus chapini Klots, 1944 species          0
## 4 11035128  Xerolestes pallidus (Rambur, 1842) species          0
## 5 11035335  Mesothemis mithroides Brauer, 1900 species          0
## 6 11029091 Paracercion luzonicum (Asahina, 1968) species          0
## 7 11029136  Onychargia stellata Ris, 1915 species          0
## 8 11029184  Nehalennia sophia Selys, 1840 species          0
## 9 11029206  Lestes lundquisti Lieftinck, 1949 species          0
## 10 11029310  Lestes concinnus Selys species          0
```

Now we can download the taxonomic data (note that the search is case-sensitive):

```
tx <- taxinfo_download("order_s:Odonata",
  fields = c("guid", "order_s", "genus_s", "specificEpithet_s",
    "scientificName", "canonicalName_s", "rank"),
  verbose = FALSE)
tx <- tx[tx$rank == "species" & tx$genusS != "",] ## restrict to species and not hybrids
```

You can save the tx object as the complete species list for later use.

2.2 Filter the search to get the observations

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).

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

Follow the instructions. Your choices here would have been “in3” → “dr5”. Your variable `fq_str` will now contain a string “data_resource_uid:dr5”.

We only want to look at data from year 2000 to 2010:

```
y1 <- 2000
y2 <- 2010
fq_str <- c(fq_str, paste0("year:[", y1, " TO ", y2, "]"))
# Note the square brackets are hard limits
```

We also want to filter spatially for Southern Sweden ([Götaland](#)).

Vector spatial layers (eg. polygons) can be imported in a number of different ways. SBDI APIs take as search input polygons in the so-called WKT [Well Known Text](#) format. So the first step is to load a vector layer and transform it into a WKT string. You could instead use the data we provide in the SBDI4R package `data("swe")`.

```
data("swe", package = "SBDI4R")
wGotaland <- swe$Counties$LnNamn %in% c("Blekinge", "Gotlands", "Hallands",
                                       "Jönköpings", "Kalmar", "Kronobergs",
                                       "Östergötlands", "Skåne", "Västra Götalands")
gotaland_c <- swe$Counties[wGotaland,]
```

There are details about this polygon that we need to take care before. The WKT string should not be too long to be accepted by the API service. Also, the polygon we just got is projected in the coordinate system SWEREF99 TM, and the API service only accepts coordinates in a geodesic coordinate system WGS84. Let's construct the WKT string:

```
# transform the CRS
gotaland_c <- st_transform(gotaland_c,
                           crs = st_crs(4326))

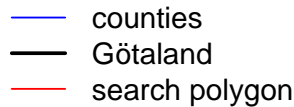
# dissolve the counties into one polygon
gotaland <- st_union(gotaland_c)

# create a convex hull of the polygon to simplify the geometry and
# reduce the length of the WKT string
gotaland_ch <- st_convex_hull(gotaland)

# cast it as MULTIPOLYGON as this is what SBDI's API need
# NOTE: as of today, the SBDI APIs will only work properly if the polygon is
# submitted as a MULTIPOLYGON
gotaland_ch <- st_cast(gotaland_ch, to = "MULTIPOLYGON")

# create WKT string
wkt <- st_as_text(gotaland_ch)
```

```
## [1] "MULTIPOLYGON (((13.33575 55.34003, 12.81633 55.38594, 11.25342 58.35786, 11.13161 58.90942, 11.13161 58.90942, 11.25342 58.35786, 12.81633 55.38594, 13.33575 55.34003)))"
```



```
xf <- occurrences(taxon = "order:Odonata",
  fq = fq_str,
  wkt = wkt,
  extra = "collector",
  email = "sbdi4r-test@biodiversitydata.se",
  download_reason_id = 10)
```

```
save(xf, file = "an_appropriate_name.rdata")
load(file = "an appropriate name.rdata")
```

Before we can use the observation records we need to know if the observation effort (sampling effort) has varied over time and in space. We can approximate observation effort from the data by defining field visits

i.e. occasions at which an observer has sampled observations. We reconstruct field visits (that is, assign each observation a visitUID) using using the package [BIRDS](#). Additionally we want the data to be summarized over a grid of 25 km (provided through the SBDI4R package). The following functions will perform many different summaries at the same time. Please refer to the BIRDS package documentation for more detail.

```
remotes::install_github("Greensway/BIRDS")
library(BIRDS)

OB <- organiseBirds(xf$data, sppCol = "species" ,
  # We only want observations identified at the species level
  taxonRankCol = "rank", taxonRank = "species",
  # the visits are defined by collector and named locality
  idCols = c("locality", "collector"),
  timeCols = c("year", "month", "day"),
  xyCols = c("longitude", "latitude") )

# We don't need the whole grid, just the piece that overlaps our searching polygon
wInt <- unlist(st_intersects(gotlandand, Sweden_Grid_25km_Wgs84))
gotland_grid25 <- Sweden_Grid_25km_Wgs84[wInt,]

SB <- summariseBirds(OB, grid = gotland_grid25, spillOver = "unique")
```

Once summarised, we can see over space and for a few selected years how the number of observations is distributed:

```
maxC <- max(SB$spatial$nObs, na.rm = TRUE)
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
  c(0, maxC),
  na.color = "transparent")

oldpar <- par()
par(mar = c(1,1,1,1), mfrow=c(1,3))
plot(SB$spatial$geometry, col=palBW(SB$spatial$nObs),
  border = "grey", main="All years") ## with palette
legend("bottomleft", inset = c(0,0.05),
  legend = round(seq(0, maxC, length.out = 5)),
  col = palBW(seq(0, maxC, length.out = 5)),
  title = "Number of \nobservations", pch = 15, bty="n")

## or export other combinations, e.g. one map per observed year
yearlySp <- exportBirds(SB,
  dimension = "spatial",
  timeRes = "yearly",
  variable = "nObs",
  method = "sum")

maxC <- max(yearlySp$'2005', na.rm = TRUE)
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
  c(0, maxC),
  na.color = "transparent")

plot(yearlySp$geometry, col=palBW(yearlySp$'2005'),
  border = "grey", main="2005")
legend("bottomleft", inset = c(0,0.05),
```

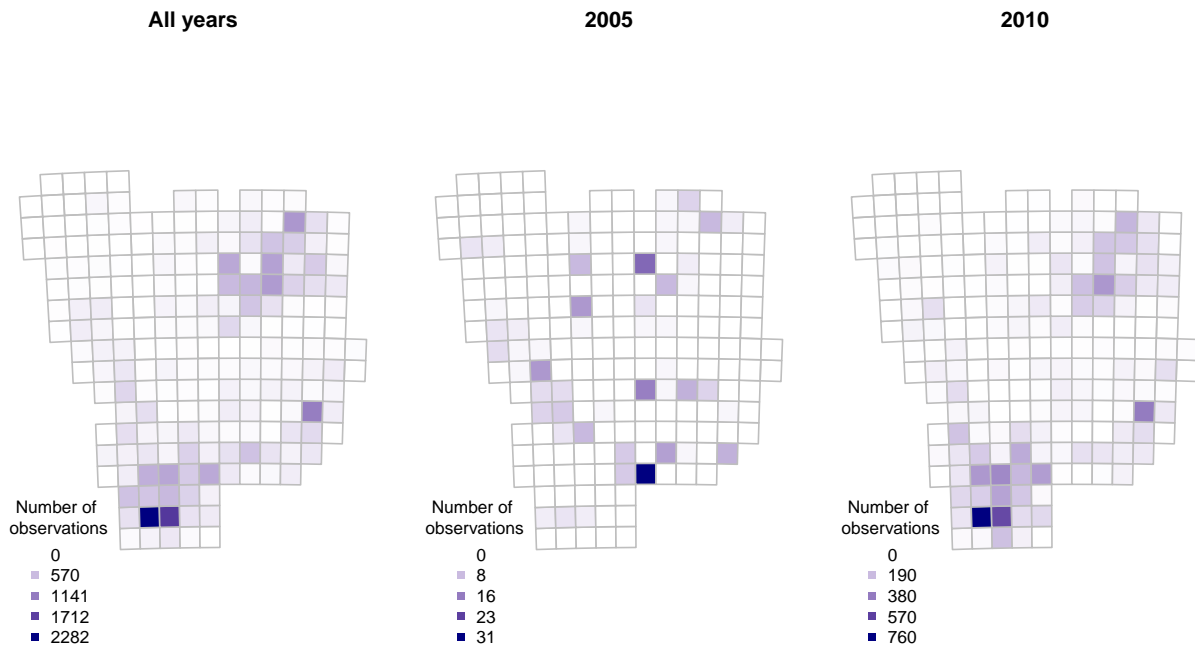
```

legend = round(seq(0, maxC, length.out = 5)),
col = palBW(seq(0, maxC, length.out = 5)),
border = "grey",
title = "Number of \nobservations", pch = 15, bty="n")

maxC <- max(yearlySp'2010', na.rm = TRUE)
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
                             c(0, maxC),
                             na.color = "transparent")

plot(yearlySp$geometry, col=palBW(yearlySp$'2010'),
     border = "grey", main="2010")
legend("bottomleft", inset = c(0,0.05),
     legend = round(seq(0, maxC, length.out = 5)),
     col = palBW(seq(0, maxC, length.out = 5)),
     border = "grey",
     title = "Number of \nobservations", pch = 15, bty="n")
par(oldpar)

```



We now want to use the number of field visits as the measure for sampling effort. :

```

library(cowplot)
library(ggplot2)
library(colorRamps)
library(gridExtra)

vis <- ggplot(data = SB$spatial, aes( fill = nVis)) +
  geom_sf() +
  ggtitle("Visits") +
  scale_fill_gradient(low = "#56B1F7",
                     high = "#132B43",

```



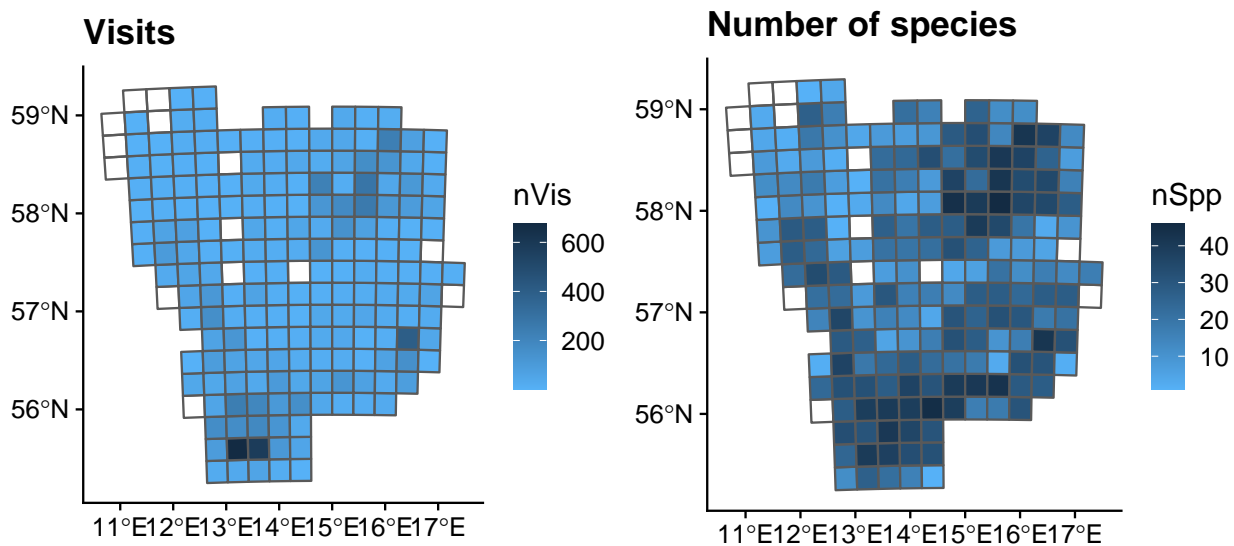
```

      na.value = NA) +
  theme(plot.margin = margin(1, 1, 1, 1, "pt")) +
  theme_cowplot()

spp <- ggplot(data = SB$spatial, aes( fill = nSpp)) +
  geom_sf() +
  ggtitle("Number of species") +
  scale_fill_gradient(low = "#56B1F7",
                     high = "#132B43",
                     na.value = NA) +
  theme(plot.margin = margin(1, 1, 1, 1, "pt")) +
  theme_cowplot()

grid.arrange(vis, spp, ncol = 2)

```



2.3.0.1 Temporal check We see that SB contains an element called `SB$temporal` that contains a daily time series with time-specific rows when there is information. `xts` also supports day time, but dating below day resolution is not yet implemented in the BIRDS package.

```

sb.xts <- SB$temporal
head(sb.xts, 5)

```

```

##           nObs nVis nSpp
## 2000-03-24     1     1     1
## 2000-04-05     4     3     3
## 2000-04-06    11     6     3
## 2000-04-10     1     1     1
## 2000-04-12     3     3     1

```

Sub-setting is convenient in `xts` as you can do it with its dates and with a `/` for a range of dates.

```
sb.xts["2010-09-07"] #a specific day
```

```
##           nObs nVis nSpp
## 2010-09-07   19   10   12
```

```
sb.xts["2010-09-01/2010-09-15"] #for a period
```

```
##           nObs nVis nSpp
## 2010-09-01   46   19   14
## 2010-09-02   28   14   12
## 2010-09-03   23   10   10
## 2010-09-04   64   20   18
## 2010-09-05   74   27   12
## 2010-09-06   18    5   11
## 2010-09-07   19   10   12
## 2010-09-08   13    6    8
## 2010-09-09   32   12   14
## 2010-09-10    1    1    1
## 2010-09-11   16    9    8
## 2010-09-12   20   10    8
## 2010-09-13   14    5    9
## 2010-09-14    1    1    1
## 2010-09-15    3    3    2
```

```
sb.xts["2010-09"] #a specific month
```

```
##           nObs nVis nSpp
## 2010-09-01   46   19   14
## 2010-09-02   28   14   12
## 2010-09-03   23   10   10
## 2010-09-04   64   20   18
## 2010-09-05   74   27   12
## 2010-09-06   18    5   11
## 2010-09-07   19   10   12
## 2010-09-08   13    6    8
## 2010-09-09   32   12   14
## 2010-09-10    1    1    1
## 2010-09-11   16    9    8
## 2010-09-12   20   10    8
## 2010-09-13   14    5    9
## 2010-09-14    1    1    1
## 2010-09-15    3    3    2
## 2010-09-17    3    2    3
## 2010-09-18    9    5    5
## 2010-09-19   12    7    5
## 2010-09-21    3    2    3
## 2010-09-22    4    4    2
## 2010-09-23    3    3    2
## 2010-09-24   10    5    5
## 2010-09-25    7    4    6
```

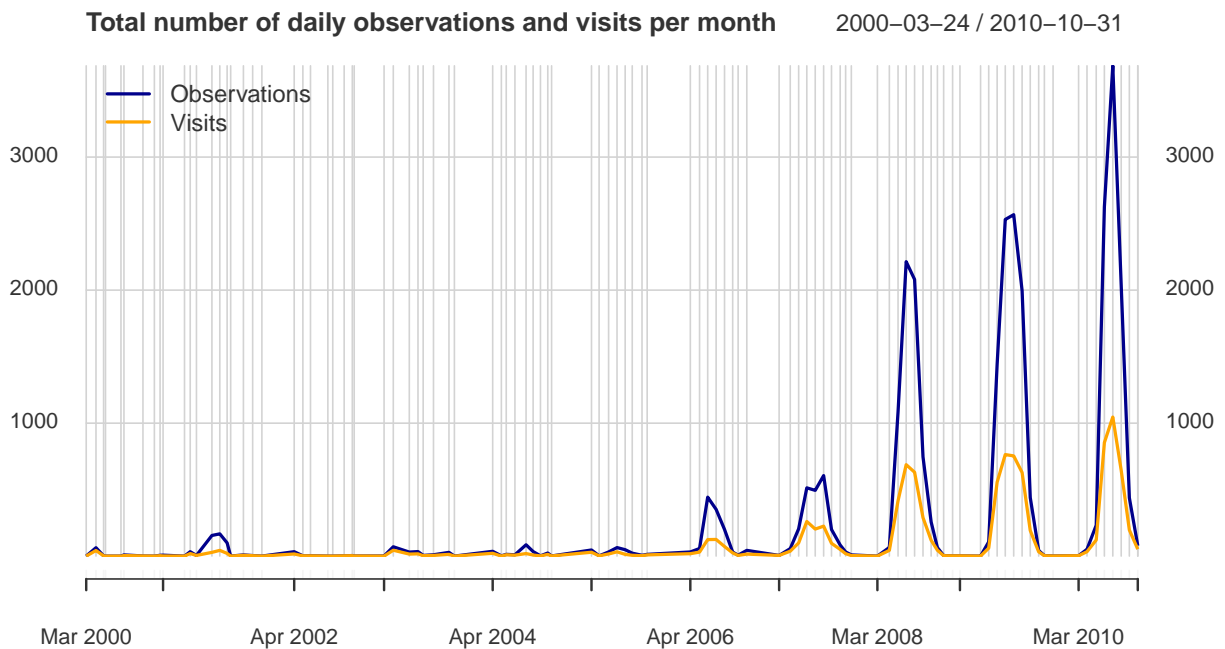
```
## 2010-09-26    7    6    2
## 2010-09-28    2    2    2
## 2010-09-29    5    3    4
## 2010-09-30    2    2    2
```

The package `xts` has several tools for converting to different time periods. Here we use `apply.monthly` to obtain the total number of observations and visits per month. The plot command for an object of class `xts` provides a many features. This makes it fairly easy to customize your plots. Read more in `?plot.xts`.

```
library(xts)
obs.m <- apply.monthly(sb.xts$nObs, "sum", na.rm = TRUE)
vis.m <- apply.monthly(sb.xts$nVis, "sum", na.rm = TRUE)

plot(obs.m,
      col = "darkblue",
      grid.ticks.on = "month",
      major.ticks = "year",
      grid.col = "lightgrey",
      main = "Total number of daily observations and visits per month")

lines(vis.m, col = "orange", lwd = 2, on = 1)
```



2.4 Species trends

We can now look at some particular species and ask whether those have changed in occurrence over time:

```
speciesSummary(SB)[,1:4]
```

```
##           species nCells nObs nVis
```

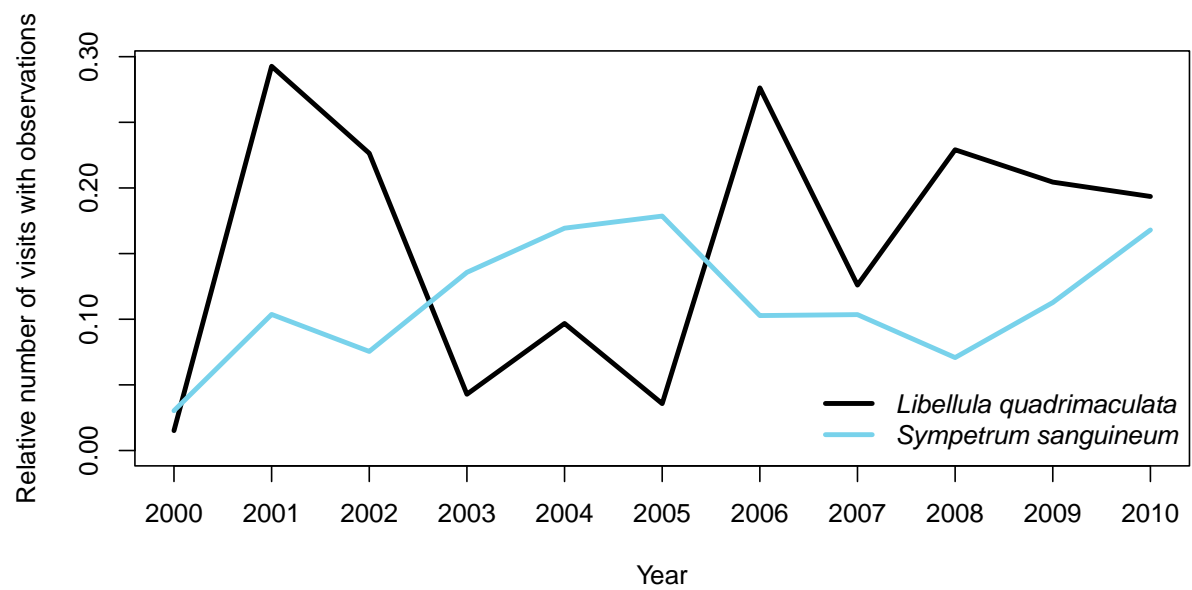
## 1	Aeshna affinis	3	32	27
## 2	Aeshna caerulea	5	12	12
## 3	Aeshna cyanea	118	859	825
## 4	Aeshna grandis	145	1691	1661
## 5	Aeshna isoceles	14	102	99
## 6	Aeshna juncea	93	334	321
## 7	Aeshna mixta	72	645	614
## 8	Aeshna serrata	13	39	38
## 9	Aeshna subarctica	35	108	99
## 10	Aeshna viridis	13	40	35
## 11	Anax imperator	43	514	474
## 12	Anax parthenope	2	5	5
## 13	Brachytron pratense	82	434	421
## 14	Calopteryx splendens	101	681	627
## 15	Calopteryx virgo	122	1058	1004
## 16	Coenagrion armatum	23	74	67
## 17	Coenagrion hastulatum	116	930	897
## 18	Coenagrion johanssoni	15	75	70
## 19	Coenagrion lunulatum	42	111	104
## 20	Coenagrion puella	113	1313	1258
## 21	Coenagrion pulchellum	114	1364	1317
## 22	Cordulegaster boltonii	75	500	493
## 23	Cordulia aenea	116	1036	1017
## 24	Enallagma cyathigerum	138	1547	1461
## 25	Epithea bimaculata	16	36	35
## 26	Erythromma najas	92	760	727
## 27	Erythromma viridulum	13	143	126
## 28	Gomphus vulgatissimus	48	160	155
## 29	Ischnura elegans	112	1210	1153
## 30	Ischnura pumilio	18	99	85
## 31	Lestes dryas	44	176	168
## 32	Lestes sponsa	137	1366	1295
## 33	Lestes virens	44	189	176
## 34	Leucorrhinia albifrons	40	143	138
## 35	Leucorrhinia caudalis	34	143	136
## 36	Leucorrhinia dubia	85	338	320
## 37	Leucorrhinia pectoralis	82	354	337
## 38	Leucorrhinia rubicunda	93	434	421
## 39	Libellula depressa	100	521	499
## 40	Libellula fulva	9	101	90
## 41	Libellula quadrimaculata	150	1971	1923
## 42	Nehalennia speciosa	3	34	33
## 43	Onychogomphus forcipatus	76	447	445
## 44	Orthetrum cancellatum	118	924	872
## 45	Orthetrum coerulescens	70	238	231
## 46	Platycnemis pennipes	91	544	528
## 47	Pyrrhosoma nymphula	124	967	938
## 48	Somatochlora arctica	28	45	42
## 49	Somatochlora flavomaculata	80	365	358
## 50	Somatochlora metallica	100	560	551
## 51	Sympecma fusca	40	179	174
## 52	Sympetrum danae	127	777	742
## 53	Sympetrum flaveolum	73	261	254
## 54	Sympetrum fonscolombii	2	2	2

```
## 55      Sympetrum sanguineum      124 1180 1136
## 56      Sympetrum striolatum       73  270  258
## 57      Sympetrum vulgatum      118 1028  972
```

We pick two species and compare their trends in number of visits where the species were reported, relative to the total number of visits.

```
library(dplyr)
sppCount <- obsData(OB) |>
  group_by(year, visitUID) |>
  summarise("focalCountLq" = sum(scientificName == "Libellula quadrimaculata"),
            "focalCountSd" = sum(scientificName == "Sympetrum sanguineum"),
            "sppLength" = length(unique(scientificName)),
            .groups = "drop") |>
  ungroup() |>
  group_by(year) |>
  summarise("focalCountLq" = sum(focalCountLq),
            "focalCountSd" = sum(focalCountSd),
            "nVis" = length(unique(visitUID)),
            "relCountLq" = focalCountLq / nVis,
            "relCountSd" = focalCountSd / nVis,
            .groups = NULL)

oldpar <- par(no.readonly = TRUE)
plot(sppCount$year, sppCount$relCountLq,
     type = "l", lwd = 3, xlab = "Year",
     ylab = "Relative number of visits with observations",
     ylim = c(0, max(sppCount$relCountLq)),
     xaxp = c(2000, 2010, 10))
lines(sppCount$year, sppCount$relCountSd, lwd = 3, col = "#78D2EB")
legend("bottomright",
      legend = c("Libellula quadrimaculata", "Sympetrum sanguineum"),
      text.font = 3, col = c("black", "#78D2EB"), lwd = 3, bty = "n")
```



```
par(oldpar)
```